AUTOMATED STRUCTURE DISCOVERY AND PARAMETER TUNING OF NEURAL NETWORK LANGUAGE MODEL BASED ON EVOLUTION STRATEGY

Tomohiro Tanaka[1], *Takafumi Moriya*[2], *Takahiro Shinozaki*[1],
Shinji Watanabe[3], *Takaaki Hori*[3], *Kevin Duh*[4]

[1]Tokyo Institute of Technology, Japan
[2]NTT Media Intelligence Laboratories, NTT Corporation, Japan
[3]Mitsubishi Electric Research Laboratories, USA
[4]Johns Hopkins University, USA

ABSTRACT

Long short-term memory (LSTM) recurrent neural network based language models are known to improve speech recognition performance. However, significant effort is required to optimize network structures and training configurations. In this study, we automate the development process using evolutionary algorithms. In particular, we apply the covariance matrix adaptation-evolution strategy (CMA-ES), which has demonstrated robustness in other black box hyper-parameter optimization problems. By flexibly allowing optimization of various meta-parameters including layer wise unit types, our method automatically finds a configuration that gives improved recognition performance. Further, by using a Pareto based multi-objective CMA-ES, both WER and computational time were reduced jointly: after 10 generations, relative WER and computational time reductions for decoding were 4.1% and 22.7% respectively, compared to an initial baseline system whose WER was 8.7%.

Index Terms— large vocabulary speech recognition, evolution strategy, long short-term memory, language model, multi-objective optimization

1. INTRODUCTION

Neural network based language models (NNLMs) are very effective for improving speech recognition performance [1, 2, 3, 4, 5, 6], and increasingly many system developers are interested in adopting it. When developing a speech recognition system using a NNLM, the major design question is what kind of network structure and training configuration leads to the best model in terms of recognition performance and computational time. For network structure, important design questions include the number of layers, the number of units per layer, and the unit type (e.g. feedforward, recurrent). If a deep network is desired, there is even more flexibility: for example one can first use a feedforward layer on the input layer and then concatenate a long short-term memory (LSTM) layer on top of it, or vice versa. For training configurations, important design questions include the learning algorithm, learning rate, mini-batch size, dropout ratio [7], and so on. All these meta-parameters interact with each other in subtle ways, so need to be jointly optimized.

Usually, this meta-parameter optimization is manually performed by humans experts and significant effort is required. It is a bottle-neck that prohibits wider adoption of NNLMs. On the other hands, expenses for computer resources are keep decreasing, and access to large computers is becoming easy by using cloud computing services etc. Our goal is to replace the laborious manual effort with automatic computation by computers. Since neural network training is often performed off-line, and the trained model is used repeatedly, it is justifiable to allocate relatively large computational budget for meta-parameter optimization if it alleviates manual effort.

A simple approach for meta-parameter optimization would be a grid search. However, it soon becomes intractable as the number of meta-parameters increase, since the number of the lattice points of grid search is exponential to the number of meta-parameters. For example, when there are 20 types of meta-parameter variables, just trying three values for each variable requires more than 3 billion (3^{20}) evaluations. Each evaluation is extremely expensive in our case, as it involves training and scoring a neural network. So grid search is not tractable even for the fastest computer in the world. We need a black box meta-parameter optimization framework that intelligently searches the meta-parameter space under a realistic budget of computational resources.

There is a large literature on meta-heuristic optimization such as genetic algorithm (GA) [8] that have demonstrated success in many practical engineering problems. These include the design of bullet trains [9], airplanes [10], and antennas [11]. In speech recognition, several researchers have applied GA to discrete- and GMM-HMM acoustic models [12, 13, 14]. Previously, we have investigated several black box optimization frameworks to optimize meta-parameters of feedforward deep neural network (DNN) based acoustic models [15, 16, 17]. In the experiments, we found that covariance matrix adaptation-evolution strategy (CMA-ES) [18, 19, 20] generally works better than GA and Bayesian optimization (BO) [21, 22], in that it finds better models with smaller or similar number of system evaluations. We also found that CMA-ES is convenient to use due to its simple initialization.

In this study, we apply CMA-ES to optimize meta-parameters of NNLMs. Compared to our previous work [15, 16, 17], which focused on acoustic modeling and meta-parameter optimization of feedforward deep neural networks, this work focuses on language modeling and considers a network that can have a heterogeneous structure consisting of feedforward, recurrent, and LSTM layers, as well as many more varieties of learning algorithms. By applying CMA-ES and analyzing its results, we also reveal best practices and insights about the factors that affect word error rate (WER) and computational time for NNLMs, which have not been thoroughly investigated before.

Experiments are performed by applying the NNLMs on top of high performance recognition systems that are publicly available as the Kaldi [23] recipes. Both single-objective optimization based on WER, and multi-objective optimization based on WER and computational time are investigated.

In the following, we review CMA-ES and its multi-objective extension in Section 2, then describe the NNLMs used in this work in Section 3. Section 4 describes meta-parameters subject to the optimizations. Experimental setup and results are in Sections 5 and 6. Section 7 concludes with future work.

2. CMA-ES META-PARAMETER OPTIMIZATION

2.1. CMA-ES

CMA-ES is a population based meta-heuristics optimization approach. Similar to GA, it encodes possible solutions as genes. The differences of CMA-ES from GA are that it uses fixed length real valued vector $\boldsymbol{x}$ as a gene and a full covariance Gaussian distribution as the gene distribution instead of directly using a set of genes to represent their distribution. In CMA-ES, we assume that we can evaluate the value of the objective function $f(\boldsymbol{x})$, but do not assume the availability of an analytic functional form of the objective function $f(\boldsymbol{x})$ and its differentiability.

In our case, $f(\boldsymbol{x})$ represents the performance of the recognition system using an NNLM trained with meta-parameters $\boldsymbol{x}$, which encodes network structure (e.g. # of layers) and training configurations (e.g. learning rate).

More specifically, CMA-ES estimates parameters θ of a Gaussian distribution for $\boldsymbol{x}$ such that the distribution is concentrated in a region with high values of $f(\boldsymbol{x})$ as shown in Eq. (1).

$$\hat{\boldsymbol{x}} \sim \mathcal{N}(\boldsymbol{x}|\hat{\boldsymbol{\theta}}) \text{ s.t. } \hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \underbrace{\int f(\boldsymbol{x})\mathcal{N}(\boldsymbol{x}|\boldsymbol{\theta})d\boldsymbol{x}}_{\triangleq \mathbb{E}[f(\boldsymbol{x})|\boldsymbol{\theta}]}. \tag{1}$$

To efficiently solve the problem, natural gradient based gradient ascent [24] is used by taking a natural gradient of $\mathbb{E}[f(\boldsymbol{x})|\boldsymbol{\theta}]$ with respect to $\boldsymbol{\theta}$. The expectation in the natural gradient can be approximately computed by using Monte Carlo sampling with the function evaluation $y_k = f(\boldsymbol{x}_k)$ as shown in Eq. (2).

$$\tilde{\nabla}_{\boldsymbol{\theta}}\mathbb{E}[f(\boldsymbol{x})|\boldsymbol{\theta}] \approx \frac{1}{K}\sum_{k=1}^{K} y_k \boldsymbol{F}_{\boldsymbol{\theta}}^{-1}\nabla_{\boldsymbol{\theta}} \log \mathcal{N}(\boldsymbol{x}_k|\boldsymbol{\theta}), \tag{2}$$

where $\boldsymbol{x}_k$ is a sample drawn from the previously estimated distribution $\mathcal{N}(\boldsymbol{x}|\hat{\boldsymbol{\theta}}_{n-1})$, and $\boldsymbol{F}$ is the Fisher information matrix. The set of sampled genes $\{\boldsymbol{x}_1, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_K\}$ corresponds to a population of a generation, and the iteration of the gradient ascent steps corresponds to the iteration of the generations of GA. It is expected superior individuals increase in the population as the iteration proceeds.

The parameters θ of the Gaussian distribution consists of a mean vector $\boldsymbol{\mu}$ and a covariance matrix $\boldsymbol{\Sigma}$. We can obtain analytical forms of their updates $\hat{\boldsymbol{\mu}}_n$ and $\hat{\boldsymbol{\Sigma}}_n$ by substituting the concrete Gaussian form into Eq. (2), leading to:

$$\begin{cases} \hat{\boldsymbol{\mu}}_{n-1} + \epsilon_{\mu} \sum_{k=1}^{K} w(y_k)(\boldsymbol{x}_k - \hat{\boldsymbol{\mu}}_{n-1}) \\ \hat{\boldsymbol{\Sigma}}_{n-1} + \epsilon_{\boldsymbol{\Sigma}} \sum_{k=1}^{K} w(y_k) \\ \qquad \cdot\left((\boldsymbol{x}_k - \hat{\boldsymbol{\mu}}_{n-1})(\boldsymbol{x}_k - \hat{\boldsymbol{\mu}}_{n-1})^{\intercal} - \hat{\boldsymbol{\Sigma}}_{n-1}\right) \end{cases} \tag{3}$$

where $^{\intercal}$ is the matrix transpose. Note that, as in [18], y_k in Eq. (2) is approximated in Eq. (3) as a weight function $w(y_k)$, defined as:

$$w(y_k) = \frac{\max\{0, \log(K/2+1) - \log(\mathrm{R}(y_k))\}}{\sum_{k'=1}^{K} \max\{0, \log(K/2+1) - \log(\mathrm{R}(y_{k'}))\}} - \frac{1}{K}, \tag{4}$$

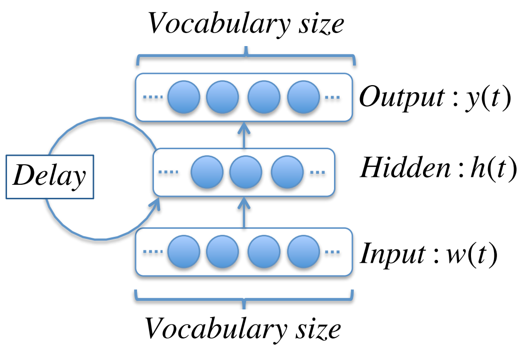

Fig. 1. *Neural network based language model.*

where $\mathrm{R}(y_k)$ is a ranking function that returns the descending order of y_k among $y_{1:K}$ (i.e., $\mathrm{R}(y_k) = 1$ for the highest y_k, $\mathrm{R}(y_k) = K$ for the smallest y_k, etc.). This equation only considers the order of y, which makes the updates less sensitive to evaluation measurements (e.g., to prevent from the different results using word accuracies and the negative sign of error counts). Intuitively, CMA-ES works by iteratively sampling $\boldsymbol{x}$ from a Gaussian distribution, whose mean and covariance is determined by a weighting of previous generations of $\boldsymbol{x}$.

2.2. Multi-objective CMA-ES using the Pareto frontier

In addition to high accuracy, objectives such as fast training time are also important in practice. Assume that we wish to maximize J objectives $F(\boldsymbol{x}) \triangleq [f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_J(\boldsymbol{x})]$ with respect to $\boldsymbol{x}$ jointly. As objectives may conflict, we adopt a notion of optimality known as Pareto optimality [25]: First, if $f_j(\boldsymbol{x}_k) \geq f_j(\boldsymbol{x}_{k'})\ \forall\ j = 1, .., J$ and $f_j(\boldsymbol{x}_k) > f_j(\boldsymbol{x}_{k'})$ for at least one objective j, then we say that $\boldsymbol{x}_k$ *dominates* $\boldsymbol{x}_{k'}$ and write $F(\boldsymbol{x}_k) \triangleright F(\boldsymbol{x}_{k'})$. Given a set of candidate solutions, $\boldsymbol{x}_k$ is *Pareto-optimal* iff no other $\boldsymbol{x}_{k'}$ exists such that $F(\boldsymbol{x}_{k'}) \triangleright F(\boldsymbol{x}_k)$.

Given a set of candidates, there are generally multiple Pareto-optimal solutions; this is known as the Pareto frontier. Note that an alternative approach is to combine multiple objectives into a single objective via an weighted linear combination: $\sum_j \beta_j f_j(\boldsymbol{x})$, where $\sum_j \beta_j = 1$ and $\beta_j > 0$. The advantage of the Pareto definition is that weights β_j need not be specified and it is more general, i.e., the optimal solution obtained by any setting of β_j is guaranteed to be included in the Pareto frontier.

CMA-ES can be extended to optimize multiple objectives by modifying the rank function $\mathrm{R}(y_k)$ used in Eq. (4). Given a set of solutions $\{\boldsymbol{x}_k\}$, we first assign rank = 1 to those on the Pareto frontier. Then, we exclude these rank 1 solutions and compute the Pareto frontier again for the remaining solutions, assigning them rank 2. This process is iterated until no $\{\boldsymbol{x}_k\}$ remain, and we obtain a ranking of all solutions according to multiple objectives in the end.

3. NEURAL NETWORK LANGUAGE MODELS

Figure 1 shows an example of a recurrent neural network (RNN) language model having 1 hidden layer that models word probability as shown in Eq. (5).

$$\begin{aligned} \mathrm{P}(\boldsymbol{w}_{t+1}|\boldsymbol{w}_1, \ldots, \boldsymbol{w}_t) &\approx \mathrm{P}(\boldsymbol{w}_{t+1}|\boldsymbol{w}_t, \boldsymbol{h}(t)) \\ &= \mathrm{softmax}(\boldsymbol{W}\boldsymbol{h}(t) + \boldsymbol{b}) \end{aligned} \tag{5}$$

The input is a vector of 1-of-K representation of a current word $\boldsymbol{w}_t$ where K is the vocabulary size of NNLM. The word vector is mapped to a real valued vector by a projection layer placed at the input of the network. The hidden state $\boldsymbol{h}(t)$ is calculated from the current input $\boldsymbol{w}(t)$ and the previous state $\boldsymbol{h}(t-1)$, and the contextual information of the past inputs is stored in the layer. At the final output layer $\boldsymbol{y}(t)$, word occurrence probability $\mathrm{P}(\boldsymbol{w}_{t+1}|\boldsymbol{w}_1, \ldots, \boldsymbol{w}_t)$ is calculated by a softmax function having parameters $\boldsymbol{W}$ and $\boldsymbol{b}$.

The parameters of a RNN are trained using back propagation-through-time (BPTT) so that the context dependency is modeled. However, RNNs cannot effectively use long context information due to the vanishing and exploding gradient problem [26]. To address the problem, LSTM RNN that consists of LSTM blocks has been proposed. A LSTM block has a memory cell and three gates (input, forget and output) to control the value stored in the memory cell [27]. By replacing the unit in recurrent hidden layer of an RNN language model with the LSTM block, a LSTM based language model is obtained. Since all the activation functions in the LSTM block are differentiable, LSTM neural network can be trained by BPTT in the same way as simple RNN.

4. META-PARAMETERS SUBJECT TO THE OPTIMIZATION

To build NNLM, several types of meta-parameters require optimization. In this study, the following meta-parameters are optimized. The complete list of these meta-parameters is given in Table 2.

4.1. Network structure related meta-parameters

To define a network structure, the unit type of network blocks (e.g., RNN, LSTM, Feedforward (FF)), the number of layers, and the number of units per layer need to be specified as meta-parameters. The layer size of the input and output layers corresponds to the vocabulary size. Since the softmax function is expensive to evaluate when the layer size is large, the vocabulary size has a large influence on the computational time. To improve the flexibility of the network design, we allow different layers in a single network to have different block types.

4.2. Training configuration related meta-parameters

There are several variations in the back propagation algorithm regarding how to update network weights and bias. The choice of the update methods affects the training time and recognition performance, and it is a meta-parameter. In this study, we consider four update methods; SGD with momentum, RMSprop with momentum[28], ADADELTA[29], and ADAM[30].

Depending on the choice of the algorithm, some configurations are needed such as the learning rate η and the momentum γ. Equations 6, 7, 8, and 9 summarizes the update formulas of SGD with momentum, RMSprop with momentum, ADADELTA, and ADAM, respectively. In the equations, θ_t is the parameter in the neural network at current time t, and $G_t = \partial L / \partial \theta_t$ is the gradient of the objective function L. The variables η, γ, ϕ_1, ϕ_2, ϕ_3, ρ, ϵ σ_1 σ_2 σ_3 σ_4 are their meta-parameters.

In this study, the learning rate for the SGD with momentum method is initially set to "initial learn rate". After several iterations of training epochs specified by "learn decay epoch", the learning rate is decayed for every epoch with a factor "learn decay". The learning rates of the other methods are automatically determined as are shown in the equations. The update of NN parameters is based on a mini-batch whose size is specified by "minibatch size". The dropout ratio is also tuned.

Table 1. *CSJ and AMI data used for the experiments.*

		CSJ	AMI
Training set	AM	512 hours	78 hours
	LM	7.5 M words	1.4 M words
Development set		7 hours	9 hours
Evaluation set		5 hours	9 hours

$$\begin{aligned} \theta_{t+1} &= \theta_t - v_t, \\ v_{t+1} &= \gamma v_t + \eta G_t. \end{aligned} \tag{6}$$

$$\begin{aligned} \theta_{t+1} &= \theta_t + v_t, \\ v_{t+1} &= \gamma v_t - \phi_2 \frac{G_t}{\sqrt{n_t - {g_t}^2 + \phi_3}}, \\ n_{t+1} &= \phi_1 n_t + (1-\phi_1){G_t}^2, \\ g_{t+1} &= \phi_1 g_t + (1-\phi_1) G_t. \end{aligned} \tag{7}$$

$$\begin{aligned} \theta_{t+1} &= \theta_t - v_t, \\ v_{t+1} & \frac{\sqrt{s_t}+\epsilon}{\sqrt{r_t}+\epsilon} G_t, \\ s_{t+1} &= \rho s_t + (1-\rho){v_t}^2, \\ r_{t+1} &= \rho r_t + (1-\rho){G_t}^2. \end{aligned} \tag{8}$$

$$\begin{aligned} \theta_{t+1} &= \theta_t - \sigma_4 \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}, \\ \hat{v}_{t+1} &= \frac{v_t}{1-{\sigma_1}^t}, \\ \hat{m}_{t+1} &= \frac{m_t}{1-{\sigma_2}^t {\sigma_3}^t}, \\ v_{t+1} &= \sigma_1 v_t + (1-\sigma_1) G_t, \\ m_{t+1} &= \sigma_2 m_t + (1-\sigma_2){G_t}^2. \end{aligned} \tag{9}$$

4.3. Other parameters

In practice, a limited-vocabulary NNLM is interpolated with a full-vocabulary N-gram language model for improved performance. The language model interpolation weight "NNLM weight" is also a meta-parameter as well as acoustic weight "acwt". For LSTM, properly initializing the forget gate bias is important [31], and we treat it as a meta-parameter.

5. EXPERIMENTAL SETUP

We performed two sets of experiments using the Corpus of Spontaneous Japanese (CSJ) [32] and the individual headset microphone (IHM) subset of the AMI meeting speech transcription corpus [33]. Both of them are large vocabulary continuous speech recognition tasks. Table 1 summarizes the amount of data used for the experiments. The development set is used to evaluate WERs used as the objective function of the evolutions, and the evaluation set is used for

Table 2. *Meta-parameters investigated in this work.*

Description	Domain	Initial value	CSJ optimized	AMI optimized
vocabulary size	$0 < x$	$10k$	25743	5587
# of hidden layers	$0 < x$	2 (3)	1	1
# of layer units	$0 < x$	300	407	476
NNLM weight	$0 < x < 1$	0.50	0.55	0.55
acoustic weight	$0 < x$	14.00	14.52	11.72
minibatch size	$0 < x$	32	213	52
dropout ratio	$0 < x < 1$	0.50	0.30	0.50
initial learn rate	$0 < x < 1$	1	0.99	0.98
learn decay	$0 < x < 1$	0.5	0.40	0.53
learn decay epochs	$1 < x$	6	14	13
momentum	$0 < x < 1$	10^{-10}	7.3×10^{-9}	1.3×10^{-9}
gradient clipping	$0 < x$	5	6.0	11.4
unit type	{LSTM, RNN, FF}	LSTM	LSTM	LSTM
# of proj. layer units	$0 < x$	300	563	399
initial forgetgate bias	$0 < x$	1	1.23	0.45
optimizer type	{SGD, ADAM, ADADELTA, RMSprop}	SGD	SGD	SGD
ADAM ϵ	$0 < x$	10^{-8}	1.07×10^{-8}	1.25×10^{-8}
ADAM σ_1	$0 < x < 1$	0.9	0.92	0.92
ADAM σ_2	$0 < x < 1$	0.999	0.999	0.999
ADAM σ_3	$x < 1$	$1 - 10^{-8}$	$1 - 6.0 \times 10^{-9}$	$1 - 1.4 \times 10^{-8}$
ADAM σ_4	$0 < x$	10^{-3}	5.0×10^{-3}	1.3×10^{-3}
RMSprop ϕ_1	$x < 1$	0.95	0.94	0.95
RMSprop ϕ_2	$0 < x < 1$	10^{-4}	7.94×10^{-5}	9.14×10^{-5}
RMSprop ϕ_3	$0 < x$	10^{-4}	1.40×10^{-4}	1.76×10^{-4}
RMSprop γ	$0 < x < 1$	0.90	0.88	0.91
ADADELTA ϵ	$0 < x$	10^{-3}	7.50×10^{-4}	9.73×10^{-4}
ADADELTA ρ	$x < 1$	0.95	0.92	0.96

final evaluation. To evaluate the NNLM, we first generated a N-best list with 100 hypotheses for each task using the publicly available high performance speech recognition recipe included in the Kaldi toolkit. The acoustic models used to generate the N-best lists were DNNs estimated by the sequential training. The NNLMs were then used to rescore the N-best lists, where they were linearly interpolated with 3-gram language models. While the vocabulary size for NNLM was subject to the optimization, it was fixed for the 3-gram model, and it was 72k for the CSJ task and 50k for the AMI task.

Table 2 lists meta-parameters subject to CMA-ES based optimization. Before the experiments, we roughly tuned the meta-parameters by hand using the CSJ data, and used the configuration as an initial value for CMA-ES. The same initial values shown in the third column in the table were used for the two tasks. Therefore, the initial gene is already somewhat optimal for the CSJ task, but is not tuned at all for the AMI task. Both of the settings are likely situations in real-world system development.

While the number of meta-parameters changes according to the number of layers, we need a fixed-dimensional gene vector for CMA-ES. For this, we specified the maximum number of possible layers and included all the meta-parameters in the gene. When an actual layer size specified by the gene is smaller than the maximum, the remaining meta-parameters were simply ignored. Depending on the meta-parameters, a mapping from a real number to a desired domain is needed to translate the gene values to the actual configurations. For meta-parameters such as learning decay need a value ranging $(0, 1)$, and we used the sigmoid function $1/(1 + e(-x))$. Similarly, we used $ceil\left(10^x\right)$ for positives integers (e.g. the number of layers), 10^x for positive real values (e.g. acoustic weight), and $\bmod\left(ceil\left(abs\left(x\right) \times n\right), n\right)$ for multiple choices (layer type) where n is the number of choices. The maximum number of hidden layers were set to six and 10 for the single and multi-objective optimizations. The dimension of the gene vector was 21 for the single-objective optimization, and 37 for the multi-objective optimization.

Experiments were performed using the TSUBAME 2.5 supercomputer that equips with NVIDIA K20X GPGPUs [1]. The population sizes were 20 and 30 for the single and multi-objective optimizations, respectively, and they were run in parallel. NNLMs were implemented using the Chainer toolkit [2] [34]. The optimized configurations are available at our web site.[3]

6. RESULTS

We run three experiments for the CSJ task: **Single-obj** is single-objective optimization on WER with meta-parameters in the first 12 rows of Table 2. Other meta-parameters are held constant at their initial values, e.g. LSTM was used for all the layers. **Single-obj (Fixed vocab)** is single-objective optimization on WER, but with a

[1]`http://www.gsic.titech.ac.jp/en`

[2]`http://chainer.org/`

[3]`http://www.ts.ip.titech.ac.jp`

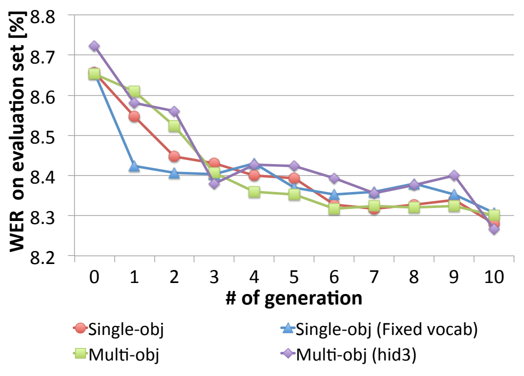

Fig. 2. *Changes in evaluation set WER for the CSJ task across CMA-ES generations*

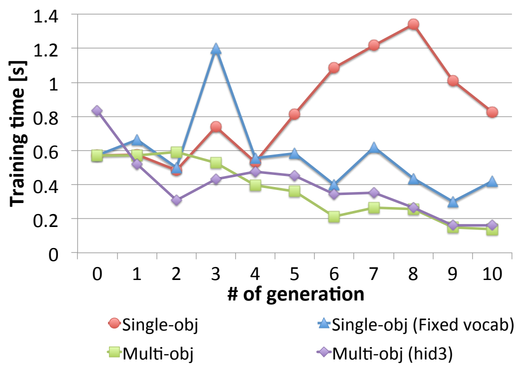

Fig. 3. *Changes in the training time for the CSJ task across CMA-ES generations*

fixed neural network vocabulary size of 10k. **Multi-obj** is multi-objective optimization on WER and training time [4], with all meta-parameters in Table 2. This means it allows layer-wise unit type optimization. Additionally, a supplementary experiment **Multi-obj (hid3)** is performed to see the influence of the initialization, where the configuration is the same as **Multi-obj** excepting that the initial number of hidden layers is set to three instead of two.

Figure 2 shows how evaluation set WER varies with the number of generations. The results are based on the best individuals that were chosen at each generation based on the development set WER. Without rescoring, the original WER was 9.35%. Rescoring with the NNLM of the initial generation reduces the baseline WER to 8.66%. In all the three conditions, WER reduced further as the generation proceeded. The differences from the baseline at 10th generation were all statistically significant with significance level $p = 0.1\%$ under the MAPSSWE significance test [35]. After 10 iterations, the WER of the three conditions were 8.28%, 8.31%, and 8.30%, respectively, and the relative reductions from the baseline were 4.4%, 4.0%, and 4.1%. **Multi-obj (hid3)** gave similar WER of 8.27% as **Multi-obj**. While the unit type was adjustable in the multi-objective optimization condition, the networks that gave the lowest level WER actually consisted of purely LSTM layers as shown in Figure 4, confirming the effectiveness of LSTMs.

Figure 3 shows training time of the networks. While the achieved WERs were similar for the three conditions, there were

[4]Here we used training time as an objective. It is correlated with decoding time, which can easily be added as a third objective if needed.

Table 3. *Computational time for decoding for the CSJ task using NNLM obtained at 10th generation.*

Baseline	Single-obj	Single-obj(Fixed voc)	Multi-obj
2.2E3 (sec)	2.5E3 (sec)	1.9E3 (sec)	1.7E3 (sec)

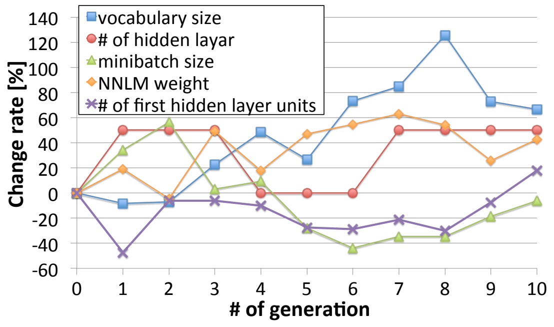

Fig. 5. *Relative changes of some of the meta-parameters from the initial generation observed for the single-objective optimization.*

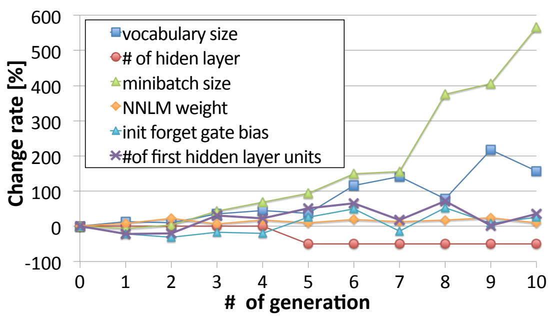

Fig. 6. *Relative changes of some of the meta-parameters from the initial generation observed for the multi-objective optimization.*

large differences in computational time. When single-objective optimization was performed with the adjustable vocabulary size condition (**Single-obj**), CMA-ES drastically increased vocabulary size to optimize WER because training time was not considered. The vocabulary size at 10th generation was 17k, which was 1.7 times larger than the initial size of 10k. Interestingly, with a fixed vocabulary size condition (**Single-obj (Fixed vocab)**), there was no major increase in computation time. When multi-objective optimization was performed, a neural network with the lowest computational time was obtained while keeping the WER comparable to the others. Part of the source of the reduction was the reduced number of hidden layers. It was decreased to 1 while it was 2 in the other two conditions. At the 10th generation, the relative reduction of the computational time was 76.1% compared to the baseline. The computational time for rescoring the evaluation set was 2.2k seconds for the baseline, and 2.5k, 1.9k, and 1.7k seconds for the three conditions at 10th generation, respectively, as shown in Table 3. The relative reduction when the multi-objective optimization was used was 22.7%.

Figures 5 and 6 show how we can analyze the results of CMA-ES to derive insights about the meta-parameters. They show relative percent change of several meta-parameters from their initial values for **Single-obj** and **Multi-obj** conditions, respectively, with positive indicating an increase in value and negative indicating a decrease. In the single-obj experiment, the number of hidden layers was always higher than the initial value (2 layers), while it decreased in

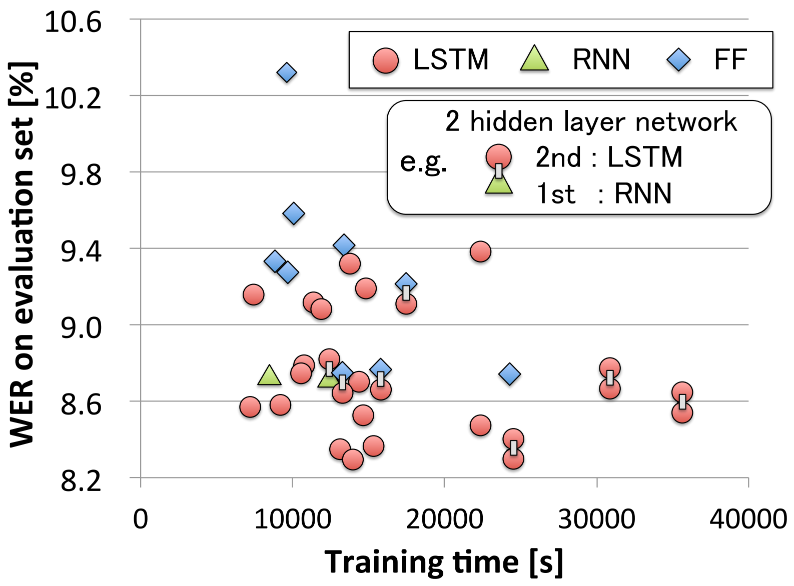

Fig. 4. *Distribution of neural networks with varying layer unit types at 10th generation of multi-objective optimization. Concatenation of multiple layers is represented by concatenation of circles (LSTM), triangles (RNN), and diamonds (FF) that represent the layer types. Note CMA-ES suppressed the number of hidden layers to be at most two.*

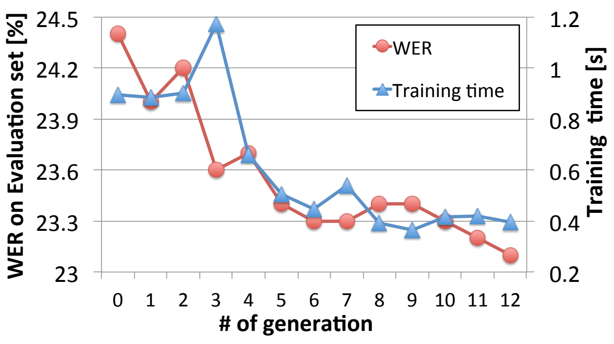

Fig. 7. *Evaluation set WER (left axis) and training time (right) for the AMI task.*

the multi-obj experiment. In the multi-obj experiment, CMA-ES decided to increase the "minibatch size", which likely led to the efficient use of GPUs and faster training times. As for the update methods, SGD with momentum was selected in both of the experiments at the 10th generation. The optimized results of all the meta-parameters for the CSJ task based on the multi-obj experiment are summarized in the fourth column of Table 2.

Based on the result of the CSJ task, we applied the multi-objective optimization to the AMI task where the WER before the rescoring was 25.4% [5]. The result is shown in Table 7. The initial generation WER was 24.4%. After 12 generations, WER reduced to 23.1%. At the same time, the computational time was also reduced to 57.3% of the initial generation. These results indicate that CMA-ES based meta-parameter optimization works well for this task as well, even when using initial values tuned for other tasks.

The last column in Table 2 shows the optimized meta-parameters for this task. Compared to the CSJ results, some meta-parameters have similar optimized values across the task, but some are not. Especially, the vocabulary size and mini-batch size are largely different. This is maybe partly because of the different coverages of the topics, and partly because of the amount of training data.

[5]This is the result without using the Fisher corpus.

7. CONCLUSION

To automate the development of NNLMs for large vocabulary speech recognition, we have applied CMA-ES based meta-parameter optimization. We show that CMA-ES consistently improves WER on two very different datasets (CSJ and AMI), despite using the same initial values for meta-parameters. Further, we analyze how NNLM meta-parameters evolved by comparing single-objective and multi-objective optimization. We demonstrate that by using the multi-objective optimization, not only the WER but also the computational time was reduced from the initial generation at the same time.

Future work includes scaling up the evolution using a huge corpus containing billions of words. One strategy is to parallelize the BPTT to reduce the turn-around time on parallel computers. The other is to introduce a multi-stage strategy to the evolution process to reduce the computation itself, where first a subset of the training data is used for intensive evolution and then full data is utilized for the finishing.

8. ACKNOWLEDGEMENTS

The work of T. Tanaka, T. Moriya, T. Shinozaki and K. Duh was supported by JSPS KAKENHI Grant Number 26280055. The work of S. Watanabe and T. Hori was supported by MERL.

9. REFERENCES

[1] Yoshua Bengio, Rejan Ducharme, Pascal Vincent, and Christian Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Reseach*, vol. 3, pp. 1137–1155, 2003.

[2] Holger Schwenk, “Continuous space language models,” *Computer Speech & Language*, vol. 21, pp. 492–518, 2007.

[3] Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur, “Recurrent neural network based language model,” in *Proc. INTERSPEECH*, 2010, pp. 1045–1048.

[4] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney, “LSTM neural networks for language modeling,” in *INTERSPEECH*, 2012, pp. 194–197.

[5] Ebru Arisoy, Tara N. Sainath, Brian Kingsbury, and Bhuvana Ramabhadran, “Deep neural network language models,” in *Proceedings of the NAACL-HLT*, 2012, pp. 20–28.

[6] Takaaki Hori, Chiori Hori, Shinji Watanabe, and John R. Hershey, “Minimum word error training of long short-term memory recurrent neural network language models for speech recognition,” in *Proc. ICASSP*, 2016, pp. 5990–5994.

[7] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[8] L. Davis, Ed., *Handbook of genetic algorithms*, vol. 115, Van Nostrand Reinhold New York, 1991.

[9] Jorge Muñoz Paniagua, Javier García García, and Antonio Crespo Martínez, “Aerodynamic optimization of high-speed trains nose using a genetic algorithm and artificial neural network,” *Industriales*, pp. 1–19, 2011.

[10] K. Chiba, S. Obayashi, K. Nakahashi, and H. Morino, “High-fidelity multidisciplinary design optimization of aerostructural wing shape for regional jet,” *American Institute of Aeronautics and Astronautics*, pp. 621–635, 2005.

[11] Timothy L. Pitzer, James A. Fellows, Gary B. Lamont, and Andrew J. Terzuoli, “Linear ensemble antennas resulting from the optimization of log periodic dipole arrays using genetic algorithms,” in *IEEE International Conference on Evolutionary Computation(CEC)*, 2006, pp. 3189–3196.

[12] C. W. Chau, S. Kwong, C. K. Diu, and W. R. Fahrner, “Optimization of HMM by a genetic algorithm,” in *Proc. ICASSP*. IEEE, 1997, vol. 3, pp. 1727–1730.

[13] S. Kwong, C. W. Chau, K. F. Man, and K. S. Tang, “Optimisation of HMM topology and its model parameters by genetic algorithms,” *Pattern Recognition*, vol. 34, no. 2, pp. 509–522, 2001.

[14] F. Yang, C. Zhang, and T. Sun, “Comparison of particle swarm optimization and genetic algorithm for HMM training,” in *Proc. ICPR*, 2008, pp. 1–4.

[15] S. Watanabe and J. Le Roux, “Black box optimization for automatic speech recognition,” in *Proc. ICASSP*. IEEE, 2014, pp. 3256–3260.

[16] T. Shinozaki and S. Watanabe, “Structure discovery of deep neural network based on evolutionary algorithms,” in *Proc. ICASSP*, 2015, pp. 4979–4983.

[17] T. Moriya, T. Tanaka, T. Shinozaki, S. Watanabe, and K. Duh, “Automation of system building for state-of-the-art large vocabulary speech recognition using evolution strategy,” in *Proc. ASRU*, 2015, pp. 610–616.

[18] N. Hansen, S. D. Müller, and P. Koumoutsakos, “Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES),” *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003.

[19] Y. Akimoto, Y. Nagata, I. Ono, and S. Kobayashi, “Bidirectional relation between CMA evolution strategies and natural evolution strategies,” in *Proc. Parallel Problem Solving from Nature (PPSN)*, 2010, pp. 154–163.

[20] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, “Natural evolution strategies,” *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 949–980, 2014.

[21] J. Mockus, “On Bayesian methods for seeking the extremum,” in *Proceedings of the IFIP Technical Conference*, London, UK, UK, 1974, pp. 400–404, Springer-Verlag.

[22] J. Snoek, H. Larochelle, and R. P. Adams, “Practical Bayesian optimization of machine learning algorithms,” in *Advances in Neural Information Processing Systems 25*, 2012.

[23] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlıcek, Y. Qian, P. Schwarz, J. Silovskı, G. Stemmer, and K. Veselı, “The Kaldi speech recognition toolkit,” in *Proc. ASRU*, 2011.

[24] S. Amari, “Natural gradient works efficiently in learning,” *Neural computation*, vol. 10, no. 2, pp. 251–276, 1998.

[25] K. Miettinen, *Nonlinear Multiobjective Optimization*, Springer, 1998.

[26] Yoshua Bengio, Patrice Simard, and Paolo Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[27] Sepp Hochreiter and Jürgen Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[28] Alex Graves, “Generating sequences with recurrent neural networks.,” *CoRR*, vol. abs/1308.0850, 2013.

[29] Matthew D. Zeiler, “ADADELTA: an adaptive learning rate method,” *CoRR*, vol. abs/1212.5701, 2012.

[30] Diederik P. Kingma and Jimmy Ba, “Adam: A method for stochastic optimization,” in *The International Conference on Learning Representations (ICLR)*, 2015.

[31] Felix A. Gers, Jürgen Schmidhuber, and Fred A. Cummins, “Learning to forget: Continual prediction with lstm.,” *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[32] S. Furui, K. Maekawa, and H. Isahara, “A Japanese national project on spontaneous speech corpus and processing technology,” in *Proc. ASR’00*, 2000, pp. 244–248.

[33] Jean Carletta, Simone Ashby, Sebastien Bourban, Mike Flynn, Mael Guillemot, Thomas Hain, Jaroslav Kadlec, Vasilis Karaiskos, Wessel Kraaij, Melissa Kronenthal, Guillaume Lathoud, Mike Lincoln, Agnes Lisowska, Iain McCowan, Wilfried Post, Dennis Reidsma, and Pierre Wellner, “The ami meeting corpus: A pre-announcement,” in *Proceedings of the Second International Conference on Machine Learning for Multimodal Interaction*, 2006, pp. 28–39.

[34] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton, “Chainer: a next-generation open source framework for deep learning,” in *Neural Information Processing Systems (NIPS)*, 2015.

[35] L. Gillick and S. Cox, “Some statistical issues in the comparison of speech recognition algorithms,” in *Proc. ICASSP*, 1989, pp. 532–535.