

# Chapter 4

## Automated Development of DNN Based Spoken Language Systems Using Evolutionary Algorithms



Takahiro Shinozaki, Shinji Watanabe, and Kevin Duh

**Abstract** Spoken language processing is one of the research areas that has contributed significantly to the recent revival in neural network research. For example, speech recognition has been at the forefront of deep learning research, inventing various novel models. Their dramatic performance improvements compared to previous state-of-the-art implementations have resulted in spoken language systems being deployed in a wide range of applications today. However, these systems require intensive tuning of their network designs and the training setups in order to achieve maximal performance. The laborious effort by human experts is becoming a prominent obstacle in system development. In this chapter, we first explain the basic concepts and the neural network-based implementations of spoken language processing systems. Several types of neural network models will be described. We then introduce our effort to automate the tuning of the system meta-parameters using evolutionary algorithms.

### 4.1 Spoken Language Processing Systems

An automatic speech recognition system takes a waveform signal of an utterance and outputs the corresponding text as the recognition result. It functionally corresponds to the human ear. Contrary, a speech synthesis system takes text as input and outputs a waveform signal of the synthesized voice as the output, which corresponds to the human mouth. Depending on the applications, they are used as a stand-alone

---

T. Shinozaki (✉)  
Tokyo Institute of Technology, Yokohama, Kanagawa, Japan  
e-mail: [shinot@ict.e.titech.ac.jp](mailto:shinot@ict.e.titech.ac.jp)

S. Watanabe  
Johns Hopkins University, Baltimore, MD, USA  
e-mail: [shinjiw@ieee.org](mailto:shinjiw@ieee.org)

K. Duh  
Johns Hopkins University, Baltimore, MD, USA  
e-mail: [kevinduh@cs.jhu.edu](mailto:kevinduh@cs.jhu.edu)

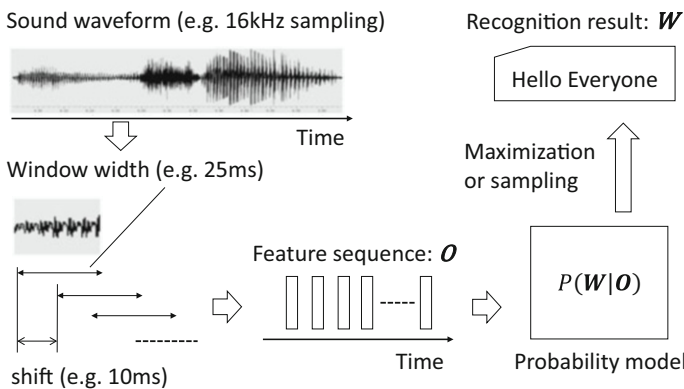
application or as a sub-component of other systems such as spoken dialogue systems and spoken translation systems.

### 4.1.1 Principle of Speech Recognition

In speech recognition systems, the input waveform is typically analyzed by short-time Fourier transform by segmenting the waveform with overlapping short windows as shown in Fig. 4.1. The window width and the shift are typically 25 and 10 ms to balance the frequency and time resolutions to capture temporal changes of frequency patterns of sub-phone units. As a result of the analysis, the waveform is converted as a time sequence of fixed-dimensional vectors, where the rate corresponds to the reciprocal of the window shift. The obtained frequency pattern vector may be used as it is, or further analyzed to obtain Mel-frequency cepstral coefficients (MFCCs) [1] or perceptual linear predictive (PLP) [2]. In either the case, the result is a sequence of vectors that contain useful information for speech recognition, where a time position of a vector is referred to as a frame. The process is called *feature extraction*.

Let  $O = \langle o_1, o_2, \dots, o_T \rangle$  be a sequence of acoustic feature vectors of length  $T$  extracted from an utterance, and  $W = \langle w_1, w_2, \dots, w_N \rangle$  be a word sequence or a text of length  $N$ . Speech recognition is formulated as a problem of finding  $\hat{W}$  that maximizes the conditional probability  $P(W|O)$  as shown in Eq. (4.1), or drawing a sample  $\tilde{W}$  from  $P(W|O)$  as shown in Eq. (4.2).

$$\hat{W} = \underset{W}{\operatorname{argmax}} P(W|O), \tag{4.1}$$



**Fig. 4.1** General framework of speech recognition. A sound signal is first converted to a sequence of feature vectors  $O$  by applying a sliding window and frequency analysis, etc. The recognition result is obtained from the posterior distribution  $P(W|O)$  of a word sequence  $W$  given the feature sequence  $O$  by maximization or probability sampling

$$\tilde{W} \sim P(W|O). \quad (4.2)$$

Finding  $\hat{W}$  means outputting the most likely recognition hypothesis, whereas drawing a sample  $\tilde{W}$  corresponds to output a hypothesis according to the probability. While the former is more direct to the goal of speech recognition, the latter sometimes have an advantage in the model training.

The conditional probability  $P(W|O)$  may be directly or indirectly modeled. In the latter case, the Bayes' rule is applied as shown in Eq. (4.3).

$$P(W|O) = \frac{P(O|W)P(W)}{P(O)} \propto P(O|W)P(W). \quad (4.3)$$

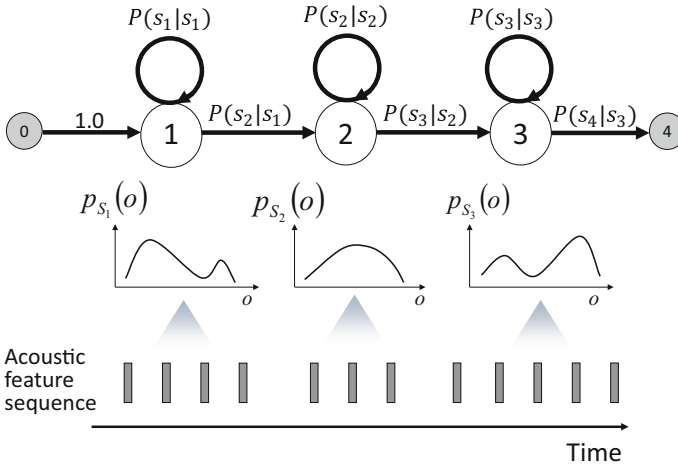
The models of  $P(O|W)$  and  $P(W)$  are referred to as an *acoustic model* and a *language model*, respectively. The acoustic model describes the generative distribution of the acoustic feature sequence  $O$  given the text  $W$ , whereas the language model describes the distribution of the text  $W$ . The denominator  $P(O)$  in Eq. (4.3) may be ignored for the maximization or the sampling purposes, since it is a constant in the processes. Hidden Markov model (HMM) has long been used for acoustic modeling in traditional speech recognition systems. The direct modeling of  $P(W|O)$  had been intricate until recently for large vocabulary speech recognition. However, the approach is rapidly developing as end-to-end speech recognition with the progress of deep learning.

### 4.1.2 Hidden Markov Model Based Acoustic Modeling

HMM consists of a finite set of internal states  $\{0, 1, \dots, F\}$ , a set of emission distributions  $\{P(o|s)\}$  each of which is associated to a state  $s$ , and a set of state transition probabilities  $\{P(s'|s)\}$  from a state  $s$  to a state  $s'$  as shown in Fig. 4.2. The initial state  $s = 0$  and the final state  $s = F$  represent the beginning and end of the state transitions, and they do not have the emission distribution. HMM gives a model of joint probability of  $P(O, S)$  as shown in Eq. (4.4), where  $S = \langle s_0 = 0, s_1, s_2, \dots, s_T, s_{T+1} = F \rangle$  is a state sequence that starts with the initial state and ends in the final state. By marginalizing over all possible state sequences  $S$ , the probability of observing the feature sequence is obtained as shown in Eq. (4.5).

$$P_\theta(O, S) = P_\theta(s_{T+1}|s_T) \prod_{t=1}^T P_\theta(s_t|s_{t-1}) P_\theta(o_t|s_t), \quad (4.4)$$

$$P_\theta(O) = \sum_S P_\theta(O, S), \quad (4.5)$$



**Fig. 4.2** Hidden Markov model. A circle represents a state, and a directed arc represents a transition. This example HMM has a left-to-right structure with three emission states and an initial and a final states

where  $\theta$  indicates a set of all parameters of the emission distributions and the transition probabilities. An acoustic model  $P_\theta(O|W)$  is obtained by preparing an HMM for each word sequence  $W$  as shown in Eq. (4.6).

$$P_\Theta(O|W) = P_{\theta_W}(O) = \sum_S P_{\theta_W}(O, S), \quad (4.6)$$

where  $\theta_W$  indicates  $W$  dependent parameter set, and  $\Theta$  is a union of  $\theta_W$  for all  $W$ . Since the number of possible word sequences is exponential to the length of the sequence, separately preparing an HMM for each sequence is intractable both in terms of required memory and parameter estimation from finite training data. Instead, a set of HMMs is prepared to model each phoneme  $p$ , and an utterance HMM is composed by concatenating the phoneme HMMs according to the pronunciation of the word sequence as shown in Fig. 4.3. The phoneme HMM set is referred to as a mono-phoneme model.

A limitation of the mono-phoneme approach is that the same phoneme HMM is used regardless of the surrounding phoneme context in the utterance. Since human voice is generated by modulating the shape of the vocal tract by moving mouth, the change is not instant. Therefore, the spectral pattern of a phoneme is affected by surrounding phonemes. For example, spectral pattern of the same phoneme /ih/ is notably different when it appears in pronunciations of “big” and “bit.” Context-dependent phoneme model is used to improve the modeling accuracy, where separate HMMs are prepared for the same phoneme for different preceding and succeeding phoneme contexts. The most popular context-dependent phoneme modeling is tri-phoneme, where a set of HMMs for a phoneme is prepared for one

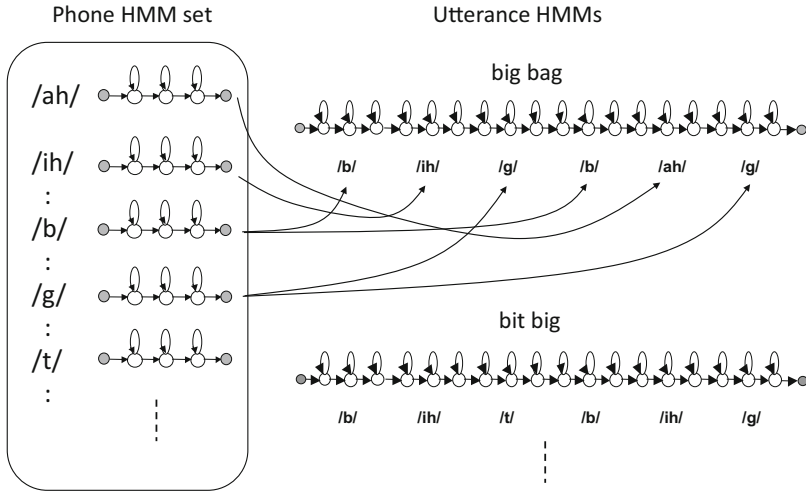


Fig. 4.3 Phoneme HMM based utterance modeling

preceding and one succeeding phonemes. When the number of phonemes is  $N$ , the number of tri-phoneme HMM is  $N^3$ , which is much larger than  $N$  of the mono-phoneme model. It causes a problem in the model parameter estimation especially for rare context and phoneme pairs since few or even no samples are available in the training set. To address the problem, clustering is performed for the context-dependent HMM states to control the model complexity by merging the HMM states [3].

The state emission distribution  $P(o|s)$  has traditionally been modeled by a mixture of Gaussian distributions (GMM) as shown in Eq. 4.7, where  $w_i$  is a mixture weight ( $0 < w_i$  and  $\sum_i w_i = 1$ ) and  $N(o|\mu_i, \Sigma_i)$  is a Gaussian distribution with mean  $\mu_i$  and variance  $\Sigma_i$ .

$$P(o|s) = \sum_i w_i N(o|\mu_i, \Sigma_i). \tag{4.7}$$

Later, it has been replaced by deep neural networks (DNNs) as shown in Eq. 4.8, where  $P(s|o)$  is obtained by the neural network.

$$P(o|s) = \frac{P(s|o) p(o)}{P(s)} \propto \frac{P(s|o)}{P(s)}. \tag{4.8}$$

Figure 4.4 shows the whole structure of DNN-HMM. The DNN-HMM often outperforms GMM-HMM with the recognition performance, especially when a larger amount of training data is available. The number of clustered HMM states, neural network structure, and their learning conditions are meta-parameters to be tuned during the system development.

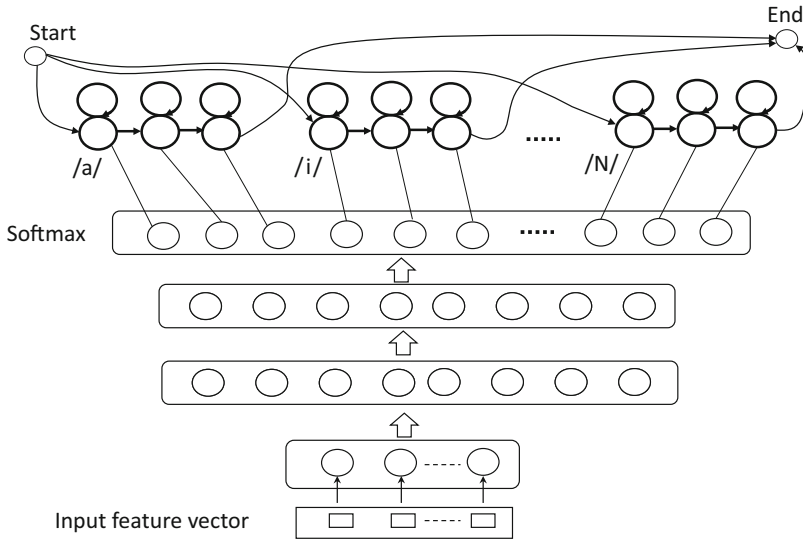
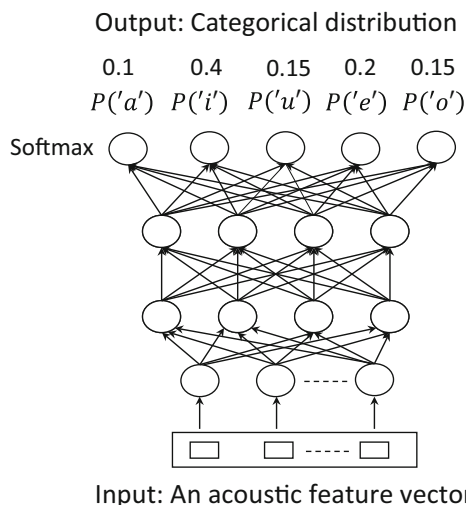


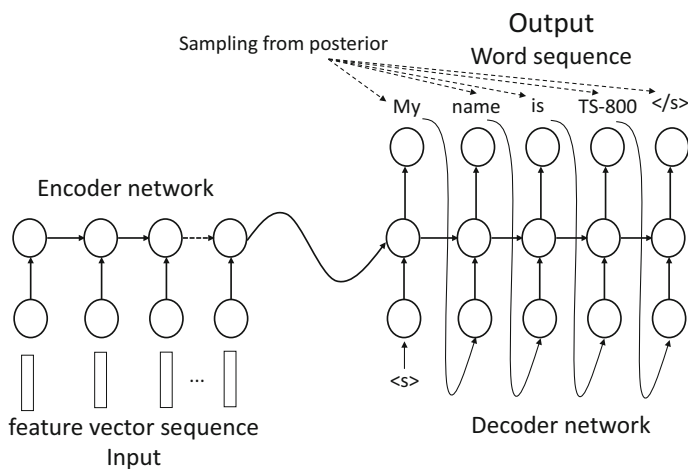
Fig. 4.4 Example of a DNN-HMM mono-phone model

### 4.1.3 End-to-End Speech Recognition System

For a simple speech recognition task such as vowel recognition from a single feature vector, where  $W$  is a set of vowels instead of a variable-length sequence of words and  $O$  is a single fixed-dimensional vector rather than the sequence of the vectors, the probability of  $P(W|O)$  can be directly modeled by a simple feed-forward neural network with a soft-max output layer as shown in Fig. 4.5. For general cases where  $O$  is a feature vector sequence, and  $W$  is a word sequence, variable-length input and output need to be handled. Neural networks realize it with some unique architectures such as encoder-decoder network with an attention mechanism [4] and Connectionist Temporal Classification (CTC) [5]. Figure 4.6 shows the architecture of a simple encoder-decoder network without the attention mechanism. It consists of an encoder network and a decoder network. The encoder network accepts a variable-length input and embeds it to a fixed-dimensional vector. The decoder network works by estimating a probability distribution of the next word given the current word  $w_t$ , from which an output word  $w_{t+1}$  is obtained by random sampling following the distribution. Initially, a special word  $\langle S \rangle$  that represents the beginning of an utterance is input as  $w_0$ , and a word  $w_1$  is sampled. Then,  $w_2$  is obtained using  $w_1$  as the input. The process is repeated until a special word  $\langle /S \rangle$  is sampled that indicates the end of an utterance. The architecture has generality to handle sequential input and output, and can be used for translation [6] and dialogue systems [7], etc., by simply changing the training data and the input/output representations. In addition, the extended architecture with the attention mechanism can explicitly handle the alignment problem between input and output [8].



**Fig. 4.5** Frame-wise vowel recognition using a feed-forward neural network. The network directly models  $P(W|O)$



**Fig. 4.6** End-to-end speech recognition system based on a simple encoder-decoder network without an attention mechanism

These systems are referred to as end-to-end systems since they directly model the input/output relationship from  $O$  to  $W$  by a monolithic neural network in contradistinction to the approaches that construct a system from separately optimized sub-models such as the acoustic and the language models, as discussed in Sect. 4.1.1. The number of hidden layers in the encoder and the decoder networks, the number of neuron units per a hidden layer, the learning conditions, etc., are meta-parameters to be tuned.

### 4.1.4 Evaluation Measures

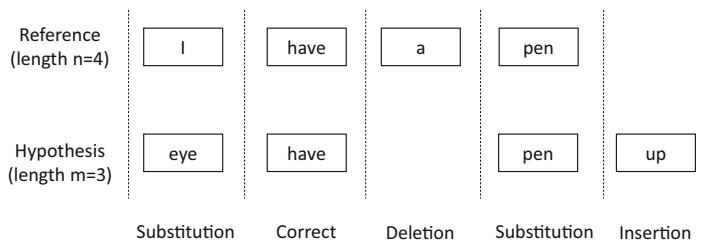
The results of speech recognition are evaluated by comparing the recognition hypothesis  $R = \langle h_1, h_2, \dots, h_m \rangle$  with a reference word sequence  $R = \langle r_1, r_2, \dots, r_n \rangle$ , where  $m$  and  $n$  are their lengths. Let  $h_j$  corresponds to  $r_i$  when we make a word by word alignment of the hypothesis and the reference. Figure 4.7 shows an example of the alignment. The word  $h_i$  is counted as correctly recognized if it is the same as  $r_i$ , and mistakenly substituted to another word if it is not. If there is no  $h_j$  for  $r_i$ , it is counted as a deletion error, and if there is no  $r_i$  for  $h_j$ , it is counted as an insertion error. Based on the alignment, word error rate (WER) is defined by Eq. (4.9).

$$\text{WER} = \frac{N_s + N_i + N_d}{n} = \frac{N_s + N_i + N_d}{N_c + N_s + N_d}, \quad (4.9)$$

where  $N_c$  is the number of correctly recognized words, and  $N_s$ ,  $N_i$ ,  $N_d$  are the numbers of substitution, insertion, and deletion errors. The WER score depends on the alignment, and the lowest score is used as the evaluation score of the recognition hypothesis. The search of the best alignment is efficiently performed by using the dynamic programming [9] algorithm. Smaller WER indicates better recognition performance, and the minimum WER score is 0.0. The WER can take larger values than 1.0 because of the existence of the insertion error. Another measure is word accuracy (WACC), which is obtained by negating WER and adding 1.0 as shown in Eq. (4.10). Larger WACC indicates better performance.

$$\text{WACC} = 1.0 - \text{WER}. \quad (4.10)$$

The WER (or WACC) is evaluated for a development set and an evaluation set. The former score is used during the training of the system for the meta-parameter tuning, and the latter is used as the final performance measure. For dialogue and translation systems where the correct answer is not unique, other measures such as BLEU [10] are used which compare the system output and the reference in a somewhat more relaxed manner in the alignment.



**Fig. 4.7** An example of a word alignment for scoring speech recognition results



**Algorithm 1** Genetic algorithm (GA)

---

```

1: for  $k = 1$  to  $K_0$  do
2:   Initialize  $x_k$ 
3: end for
4: while not convergence do
5:   for  $k = 1$  to  $K$  ( $K_0$  for the first iteration) do
6:     Decode gene  $x_k$  to configuration  $C_k$ 
7:     Evaluate configuration  $C_k$  to obtain score  $y_k = f(C_k)$ 
8:   end for
9:   Generate child genes  $\{x_k\}_{k=1}^K$  from current (parent) genes  $\{x_k\}_{k=1}^K$  and their scores  $\{y_k\}_{k=1}^K$ 
   by selection, mating, and mutation
10: end while
11: return Extract the best gene  $x^*$ 

```

---

## 4.2 Evolutionary Algorithms

Let  $y = f(\mathbf{x})$  be an evaluation function that represents the accuracy of a speech recognition system (or some performance measure of a spoken language processing system) built from tuning meta-parameters represented by  $D$ -dimensional vector  $\mathbf{x}$ . The process of finding the optimal tuning parameter  $\mathbf{x}^*$  to maximize the accuracy can be formulated as the following optimization problem:

$$\mathbf{x}^* = \operatorname{argmax}_{\mathbf{x} \in \hat{\mathcal{X}}} f(\mathbf{x}), \quad (4.11)$$

where  $\hat{\mathcal{X}}$  is a set of candidates for  $\mathbf{x}$ . Because speech recognition systems are extremely complex, there is no analytical form for the solution. We must address this optimization problem without assuming specific knowledge for  $f$ , i.e., by considering  $f$  as a black box. Another important aspect of this problem is that evaluating the function value  $f(\mathbf{x})$  is expensive because training a large vocabulary model and computing its development set accuracy can take considerable time. Thus, the key point here is for the black-box optimization to generate an appropriate set of hypotheses  $\hat{\mathcal{X}}$  to find the best  $\mathbf{x}^*$  in the smallest number of the training and evaluation steps ( $f(\mathbf{x})$ ) as possible.

### 4.2.1 Genetic Algorithm

Genetic algorithm (GA) is a search heuristic motivated by the biological evolution process. This algorithm is based on (1) the selection of genes (also called “chromosome representations”) according to their scores, pruning inferior genes for the next iteration (generation); (2) mating pairs of genes to form child genes that mix the properties of the parents, and (3) mutation of a part of a gene to produce new gene.

A popular selection method, which we will use in the later experiment, is the *tournament method*. This method first extracts a subset of  $M (< K)$  hypotheses ( $\hat{\mathcal{X}}_k = \{\mathbf{x}_{k'}\}_{k'=1}^M$ ) generated from a total of  $K$  genes randomly, and then it selects the best gene  $\mathbf{x}_{k^*}$  in the subset by their scores, i.e.,

$$\mathbf{x}_{k^*} = \operatorname{argmax}_{\mathbf{x}_{k'} \in \hat{\mathcal{X}}_k} f(\mathbf{x}_{k'}). \quad (4.12)$$

The random subset extraction step can provide variations of genes giving a chance of survival not only to the best gene but also to superior genes, and the best selection step in a subset guarantees the exclusion of inferior genes. This process is repeated  $K$  times to obtain a set of survived genes.

For the mating process, a typical method is the *one-point crossover*, which first finds a pair of (parent) genes ( $\mathbf{x}_{k_1}^p$  and  $\mathbf{x}_{k_2}^p$ ) from the selected genes and then swaps the  $\{1, \dots, d\}$  elements to  $\{d+1, \dots, D\}$  elements of these two vectors to obtain the following new (child) gene pair ( $\mathbf{x}_{k_1}^c$  and  $\mathbf{x}_{k_2}^c$ ):

$$\mathbf{x}_{k_1}^c = \begin{bmatrix} x_{k_1,1}^p \\ \vdots \\ x_{k_1,d}^p \\ x_{k_2,d+1}^p \\ \vdots \\ x_{k_2,D}^p \end{bmatrix}, \quad \mathbf{x}_{k_2}^c = \begin{bmatrix} x_{k_2,1}^p \\ \vdots \\ x_{k_2,d}^p \\ x_{k_1,d+1}^p \\ \vdots \\ x_{k_1,D}^p \end{bmatrix}. \quad (4.13)$$

The position  $d$  is randomly sampled. As the iteration increases, these processes provide appropriate genes that encode optimal DNN configurations.

Algorithm 1 summarizes the GA procedure. The process is repeated until the evaluation score is converged, and the best gene  $\mathbf{x}^*$  is extracted.

## 4.2.2 Evolution Strategy

Evolution strategy (ES) is a population-based meta-heuristic optimization algorithm that is similar to GA. A difference from GA is that ES represents a gene  $\mathbf{x}$  by a real-valued vector. Covariance matrix adaptation ES (CMA-ES) [11] is an ES, which is closely related to natural ES [12]. Although both CMA-ES and natural ES have several variations, it has been shown that their core parts are mathematically equivalent [13]. CMA-ES was proposed earlier than natural ES, but the mathematical motivation of natural ES is more concise. Here, we follow the derivation of natural ES as the explanation of CMA-ES.

CMA-ES uses a multivariate Gaussian distribution  $\mathcal{N}(\mathbf{x}|\boldsymbol{\theta})$  having a parameter set  $\boldsymbol{\theta} = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$  to represent a gene distribution, where  $\boldsymbol{\mu}$  is a  $D$ -dimensional mean

vector,  $\Sigma$  is a  $D \times D$ -dimensional covariance matrix, and  $D$  is the gene size. It seeks a distribution that is concentrated in a region with high values of  $f(\mathbf{x})$  such that sampling from the distribution provides superior genes. The search of the distribution is formulated as a maximization problem of the expected value  $\mathbb{E}[f(\mathbf{x})|\boldsymbol{\theta}]$  of  $f(\mathbf{x})$  under a Gaussian distribution  $\mathcal{N}(\mathbf{x}|\boldsymbol{\theta})$  as shown in Eqs. (4.14) and (4.15).

$$\mathbb{E}[f(\mathbf{x})|\boldsymbol{\theta}] = \int f(\mathbf{x}) \mathcal{N}(\mathbf{x}|\boldsymbol{\theta}) d\mathbf{x}, \quad (4.14)$$

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \mathbb{E}[f(\mathbf{x})|\boldsymbol{\theta}]. \quad (4.15)$$

To maximize the expectation, the gradient ascent method can be used to iteratively update the current parameter set  $\boldsymbol{\theta}_n$  starting from an initial parameter set  $\boldsymbol{\theta}_0$ , as shown in Eq. (4.16).

$$\hat{\boldsymbol{\theta}}_n = \hat{\boldsymbol{\theta}}_{n-1} + \epsilon \nabla_{\boldsymbol{\theta}} \mathbb{E}[f(\mathbf{x})|\boldsymbol{\theta}] |_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_{n-1}}, \quad (4.16)$$

where  $n$  is an iteration index and  $\epsilon (> 0)$  is a step size. To evaluate the gradient, CMA-ES uses the relation of  $\nabla_{\boldsymbol{\theta}} \log \mathcal{N}(\mathbf{x}|\boldsymbol{\theta}) = \frac{1}{\mathcal{N}(\mathbf{x}|\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} \mathcal{N}(\mathbf{x}|\boldsymbol{\theta})$ , which is called a “log-trick.” By approximating the integration by sampling after applying the log-trick, the gradient is expressed by Eq. (4.19).

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}[f(\mathbf{x})|\boldsymbol{\theta}] |_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_{n-1}} \quad (4.17)$$

$$= \int (f(\mathbf{x}) \nabla_{\boldsymbol{\theta}} \log \mathcal{N}(\mathbf{x}|\boldsymbol{\theta}_{n-1})) \mathcal{N}(\mathbf{x}|\boldsymbol{\theta}_{n-1}) d\mathbf{x} \quad (4.18)$$

$$\approx \frac{1}{K} \sum_k^K y_k \nabla_{\boldsymbol{\theta}} \log \mathcal{N}(\mathbf{x}_k|\boldsymbol{\theta}_{n-1}), \quad (4.19)$$

$$\mathbf{x}_k \sim \mathcal{N}(\mathbf{x}|\boldsymbol{\theta}_{n-1}),$$

where  $\mathbf{x}_k$  is a gene sampled from the previously estimated distribution  $\mathcal{N}(\mathbf{x}|\hat{\boldsymbol{\theta}}_{n-1})$ , and  $y_k$  is the evaluated value of the function  $y_k = f(\mathbf{x}_k)$ . The set of  $K$  samples at an iteration step corresponds to a set of individuals at a generation in an evolution. By repeating the generations, it is expected that superior individuals are obtained. Note the formulation is closely related to the reinforcement learning. If we interpret the Gaussian distribution as a policy function taking no input assuming the world is a constant, and regard the gene as an action, it is a special case of the policy gradient based reinforcement learning [14].

Although simple gradient ascent may be directly performed using the obtained gradient, CMA-ES uses the natural gradient  $\tilde{\nabla}_{\boldsymbol{\theta}} \mathbb{E}[f(\mathbf{x})|\boldsymbol{\theta}] = \mathbf{F}^{-1} \nabla_{\boldsymbol{\theta}} \mathbb{E}[f(\mathbf{x})|\boldsymbol{\theta}]$  rather than the original gradient  $\nabla_{\boldsymbol{\theta}} \mathbb{E}[f(\mathbf{x})|\boldsymbol{\theta}]$  to improve the convergence speed,

where  $F$  is a Fisher information matrix defined by Eq. (4.20).

$$F(\theta) = \int \mathcal{N}(x|\theta) \nabla_{\theta} \log \mathcal{N}(x|\theta) \nabla_{\theta} \log \mathcal{N}(x|\theta)^T dx. \quad (4.20)$$

By substituting the concrete Gaussian form for  $\mathcal{N}(x|\theta)$ , the update formulae for  $\hat{\mu}_n$  and  $\hat{\Sigma}_n$  are obtained as shown in Eq. (4.21).

$$\begin{cases} \hat{\mu}_n = \hat{\mu}_{n-1} + \epsilon_{\mu} \sum_{k=1}^K w(y_k)(x_k - \hat{\mu}_{n-1}), \\ \hat{\Sigma}_n = \hat{\Sigma}_{n-1} + \epsilon_{\Sigma} \sum_{k=1}^K w(y_k) \\ \quad \cdot (x_k - \hat{\mu}_{n-1})(x_k - \hat{\mu}_{n-1})^T - \hat{\Sigma}_{n-1}, \end{cases} \quad (4.21)$$

where  $\top$  is the matrix transpose. Note that, as in [11],  $y_k$  in Eq. (4.19) is approximated in Eq. (4.21) as a weight function  $w(y_k)$ , which is defined as:

$$w(y_k) = \frac{\max\{0, \log(K/2 + 1) - \log(R(y_k))\}}{\sum_{k'=1}^K \max\{0, \log(K/2 + 1) - \log(R(y_{k'}))\}} - \frac{1}{K}, \quad (4.22)$$

where  $R(y_k)$  is a ranking function that returns the descending order of  $y_k$  among  $y_{1:K}$  (i.e.,  $R(y_k) = 1$  for the highest  $y_k$ ,  $R(y_k) = K$  for the smallest  $y_k$ , and so forth). This equation only considers the order of  $y$ , which makes the updates less sensitive to the evaluation measurements (e.g., to prevent different results using word accuracies and the negative sign of error counts).

Algorithm 2 summarizes the CMA-ES optimization procedure, which gradually samples neighboring tuning parameters from the initial values. Because CMA-ES uses a real-valued vector as a gene, it is naturally suited for tuning continuous-valued meta-parameters. To tune discrete-valued meta-parameters, it needs a discretization by some means. The evaluation of  $f(x_k)$  can be performed independently for each  $k$ . Therefore, it is easily adapted to parallel computing environments such as cloud computing services for shorter turnaround times. The number of samples,  $K$ , is automatically determined from the number of dimensions of  $\mathbf{x}$  [11], or we can set it manually by considering computer resources.

### 4.2.3 Bayesian Optimization

Even though Bayesian optimization (BO) is motivated differently from ES and GA, in practice, there are several similarities. Especially when it is palatalized, a set of individuals are evaluated at each update stage where a fixed-dimensional vector specifies the configuration of an individual.

While CMA-ES involves a distribution of the tuning parameter  $\mathbf{x}$  taking the expectation over  $\mathbf{x}$ , BO uses a probabilistic model of the output  $y$  to evaluate an acquisition function that evaluates the goodness of  $\mathbf{x}$ . Several acquisition functions

**Algorithm 2** CMA-ES

---

```

1: Initialization of  $\hat{\boldsymbol{\mu}}_0$  and  $\hat{\boldsymbol{\Sigma}}_0$ , and  $y_0^* = \emptyset$ 
2: for  $n = 1$  to  $N$  do
3:   for  $k = 1$  to  $K$  do
4:     Sample  $\mathbf{x}_k$  from  $\mathcal{N}(\mathbf{x} | \hat{\boldsymbol{\mu}}_{n-1}, \hat{\boldsymbol{\Sigma}}_{n-1})$ 
5:     Evaluate  $y_k = f(\mathbf{x}_k)$ 
6:   end for
7:   Rank  $\{y_k\}_{k=1}^K$ 
8:   Update  $\hat{\boldsymbol{\mu}}_n$  and  $\hat{\boldsymbol{\Sigma}}_n$ 
9:   Store  $y_n^* = \max\{y_{1:K}, y_{n-1}^*\}$  corresponding  $\mathbf{x}_n^*$ 
10: end for
11: return  $\{\mathbf{x}_N^*, y_N^*\}$ 

```

---

have been proposed [15]. Here, we use expected improvement, which is suggested as a practical choice [16]. The expected improvement is defined as:

$$a^{EI}(\mathbf{x}_k) = \int \max\{0, y - y_{k-1}^*\} p(y | D_{1:k-1}, \mathbf{x}_k) dy, \quad (4.23)$$

where  $\max\{0, y - y_{k-1}^*\}$  is an improvement measure based on the best score  $y_{k-1}^* = \max_{1 \leq k' \leq k-1} y_{k'}$  among  $k-1$  previous scores, and  $p(y | D_{1:k-1}, \mathbf{x}_k)$  is the predictive distribution of  $y$  given  $\mathbf{x}_k$  and the already observed data set  $D_{1:k-1} = \{\mathbf{x}_{1:k-1}, y_{1:k-1}\}$  modeled by a Gaussian process [17].

BO then performs a deterministic search for the next candidate  $\hat{\mathbf{x}}_k$  by maximizing the expected improvement over  $y$ :

$$\hat{\mathbf{x}}_k = \underset{\mathbf{x}_k}{\operatorname{argmax}} a^{EI}(\mathbf{x}_k). \quad (4.24)$$

Equation (4.24) selects the  $\mathbf{x}_k$  that is likely to lead to a high score of  $y_k$ .

The Gaussian process models the joint probability of the  $k$  scores  $[y_{1:k-1}^\top, y]^\top$  as a  $k$ -dimensional multivariate Gaussian with a zero mean vector and a Gram matrix  $\mathbf{K}$  as covariance matrix:

$$p(\mathbf{y}_{1:k-1}, y | \mathbf{x}_{1:k}) = \mathcal{N}\left(\begin{bmatrix} \mathbf{y}_{1:k-1} \\ y \end{bmatrix} \middle| \mathbf{0}, \mathbf{K}\right), \quad (4.25)$$

$$\mathbf{K} = \begin{bmatrix} \mathbf{G} & \mathbf{g}(\mathbf{x}_k) \\ \mathbf{g}(\mathbf{x}_k)^\top & g(\mathbf{x}_k, \mathbf{x}_k) \end{bmatrix}, \quad (4.26)$$

where  $g(\mathbf{x}, \mathbf{x}')$  is a kernel function,  $\mathbf{G}$  is a Gram matrix with elements  $G_{i,j} = g(\mathbf{x}_i, \mathbf{x}_j)$  for  $1 \leq i, j \leq k-1$ , and  $\mathbf{g}(\mathbf{x}_k) = [g(\mathbf{x}_1, \mathbf{x}_k), \dots, g(\mathbf{x}_{k-1}, \mathbf{x}_k)]^\top$ . The predictive distribution of  $y$  given  $\mathbf{y}_{1:k-1}$  is obtained as a univariate Gaussian distribution by using Bayes' theorem:

$$\begin{aligned} p(y | D_{1:k-1}, \mathbf{x}_k) &= p(y | \mathbf{y}_{1:k-1}, \mathbf{x}_{1:k}) \\ &= \mathcal{N}(y | \mu(\mathbf{x}_k), \sigma^2(\mathbf{x}_k)), \end{aligned} \quad (4.27)$$

**Algorithm 3** Bayesian optimization (BO)

---

```

1: Set the domain  $\mathcal{X}$  of  $\hat{\mathbf{x}}_0$ , and  $\hat{\mathbf{y}}_0 = \emptyset$ 
2: for  $k = 1$  to  $K$  do
3:   Compute  $\hat{\mathbf{x}}_k = \operatorname{argmax}_{\mathbf{x}_k} a^{EI}(\mathbf{x}_k)$ 
4:   Evaluate  $y_k = f(\hat{\mathbf{x}}_k)$ 
5:   Store  $y_k^* = \max\{y_k, y_{k-1}^*\}$  corresponding  $\mathbf{x}_k^*$ 
6: end for
7: return  $\{\mathbf{x}_K^*, y_K^*\}$ 

```

---

where the mean  $\mu(\mathbf{x}_k)$  and variance  $\sigma^2(\mathbf{x}_k)$  are given as:

$$\begin{cases} \mu(\mathbf{x}_k) = \mathbf{g}(\mathbf{x}_k)^\top \mathbf{G}^{-1} \mathbf{y}_{1:k-1}, \\ \sigma^2(\mathbf{x}_k) = g(\mathbf{x}_k, \mathbf{x}_k) - \mathbf{g}(\mathbf{x}_k)^\top \mathbf{G}^{-1} \mathbf{g}(\mathbf{x}_k). \end{cases} \quad (4.28)$$

Based on this predictive distribution, we can analytically evaluate the expected improvement  $a^{EI}(\mathbf{x}_k)$  by substituting Eq. (4.27) into (4.23), and numerically obtain  $\hat{\mathbf{x}}_k$  by Eq. (4.24).

The basic algorithm of BO is shown in Algorithm 3. While one needs to set initial values for  $\mathbf{x}$  for CMA-ES, one needs to set the domain of  $\mathbf{x}$  for BO. Parallelization can be performed when computing the expected improvement function  $a^{EI}(\mathbf{x}_k)$  with Monte Carlo sampling. However, the greedy search resulting from BO often selects tuning parameters on the edges of the parameter domains, which leads to extremely long function evaluations when the dimension of  $\mathbf{x}$  is large. We have observed that these actually make the evaluation difficult in our experiments.

### 4.3 Multi-Objective Optimization with Pareto Optimality

In Sect. 4.2, we explained meta-parameter optimization methods for single objectives, such as the recognition accuracy. Sometimes, other objectives are also important in real applications. For example, smaller DNN size is preferable because it affects the computational costs for both training and decoding. In this section, we explain multi-objective CMA-ES with Pareto optimality.

#### 4.3.1 Pareto Optimality

Without loss of generality, assume that we wish to maximize  $J$  objectives with respect to  $\mathbf{x}$  jointly, which are defined as:

$$F(\mathbf{x}) \triangleq [f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_J(\mathbf{x})]. \quad (4.29)$$

Because objectives may conflict, we adopt a concept of optimality known as Pareto optimality [18]. For jointly optimizing multiple objectives, it needs to satisfy the following terms:

$$\begin{cases} f_j(\mathbf{x}_k) \geq f_j(\mathbf{x}_{k'}) \quad \forall j = 1, \dots, J \\ f_j(\mathbf{x}_k) > f_j(\mathbf{x}_{k'}) \quad \exists j = 1, \dots, J. \end{cases} \quad (4.30)$$

Then, we say that  $\mathbf{x}_k$  *dominates*  $\mathbf{x}_{k'}$  and write  $F(\mathbf{x}_k) \triangleright F(\mathbf{x}_{k'})$ . Given a set of candidate solutions,  $\mathbf{x}_k$  is *Pareto optimal* iff no other  $\mathbf{x}_{k'}$  exists such that  $F(\mathbf{x}_{k'}) \triangleright F(\mathbf{x}_k)$ .

Pareto optimality formalizes the intuition that a solution is good if no other solution outperforms (dominates) it in all objectives. 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 a weighted linear combination:

$$\sum_j \beta_j f_j(\mathbf{x}), \quad (4.31)$$

where  $\sum_j \beta_j = 1$  and  $\beta_j > 0$ . The advantage of the Pareto definition is that weights  $\beta_j$  need not be specified and it is more general, i.e., the optimal solution obtained by any setting of  $\beta_j$  is guaranteed to be included in the Pareto frontier. Every  $\{\mathbf{x}_{1:K}\}$  can be ranked by using the Pareto frontier, which can adapt to meta-heuristics.

### 4.3.2 CMA-ES with Pareto Optimality

We realize multi-objective CMA-ES for a low WER and small model size by modifying the rank function  $R(y_k)$  used in Eq.(4.22). Given a set of solutions  $\{\mathbf{x}_k\}$ , 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  $\{\mathbf{x}_k\}$  remain, and we ultimately obtain a ranking of all solutions according to multiple objectives. The remainder of CMA-ES remains unchanged; by this modification, future generations are drawn to optimize multiple objectives rather than a single objective. With some bookkeeping, this ranking can be computed efficiently in  $O(J \cdot K^2)$  [19].

Algorithm 4 summarizes the CMA-ES optimization procedure with Pareto optimality, which is used to rank the multiple objectives  $F(\mathbf{x}_k)$ . The obtained rank is used to update the mean vector and covariance matrix of CMA-ES. CMA-ES gradually samples neighboring tuning parameters from the initial values and finally provides a subset of solutions,  $\{\mathbf{x}, F(\mathbf{x})\}$ , that lie on the Pareto frontier (rank 1) of all stored  $N \times K$  samples.

---

**Algorithm 4** Multi-objective CMA-ES
 

---

```

1: Initialization of  $\hat{\mu}_0$  and  $\hat{\Sigma}_0$ 
2: for  $n = 1$  to  $N$  do
3:   for  $k = 1$  to  $K$  do
4:     Sample  $\mathbf{x}_k$  from  $\mathcal{N}(\mathbf{x}|\hat{\mu}_{n-1}, \hat{\Sigma}_{n-1})$ 
5:     Evaluate  $J$  objectives  $F(\mathbf{x}_k) \triangleq [f_1(\mathbf{x}_k), f_2(\mathbf{x}_k), \dots, f_J(\mathbf{x}_k)]$ 
6:   end for
7:   Rank  $\{F(\mathbf{x}_k)\}_{k=1}^K$  according to Pareto optimality
8:   Update  $\hat{\mu}_n$  and  $\hat{\Sigma}_n$ 
9: end for
10: return subset of solutions  $\{\mathbf{x}, F(\mathbf{x})\}$  that lie on the Pareto frontier (rank 1) of all stored  $N \times K$ 
    samples
  
```

---

### 4.3.3 Alternative Multi-Objective Methods

There is a rich literature of multi-objective methods for genetic algorithms. Refer to [20, 21] for a survey of techniques. One class of methods utilizes Pareto optimality in estimating the fitness  $F(\mathbf{x})$  of each solution. Examples include the widely used NSGA-II [19], and the Pareto CMA-ES method we described in Sect. 4.3.2 adopts a very similar approach.

There are also multi-objective genetic algorithms that do not utilize the concept of Pareto fitness. For example, VEGA [22] divides the selection of offspring population into separate groups based on different objectives, then allow crossover operations across groups. HGLA [23] runs a genetic algorithm on a linear combination of objectives; the combination weights are not fixed but evolved simultaneously with the solutions. All these methods should be applicable to the problem of automatic optimization of the DNN meta-parameters, but we are not aware of any large-scale empirical evaluation.

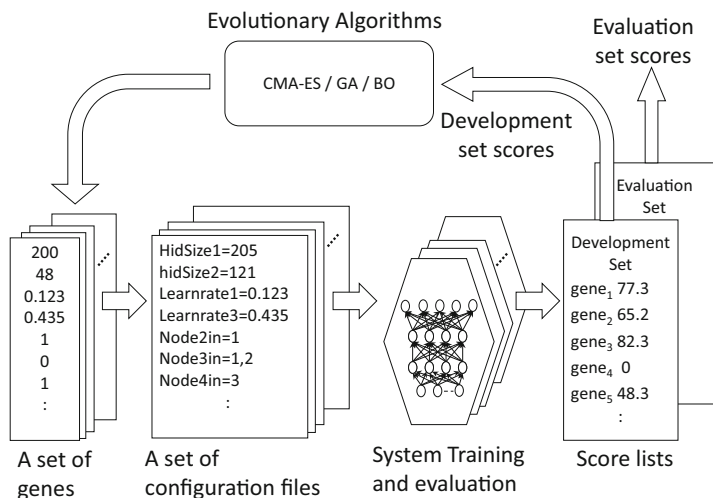
For Bayesian optimization, [24] proposed an acquisition function which chooses the  $\mathbf{x}$  to maximally reduce the entropy of the posterior distribution over the Pareto set. This has been evaluated for automatic optimization of speed and accuracy of DNNs on the MNIST image classification, with promising results. There are also methods based on using a combination of multiple objectives to a single objective, e.g. [25].

## 4.4 Experimental Setups

### 4.4.1 General Setups

We applied the evolutionary algorithms to tune large vocabulary speech recognition systems [26]. Figure 4.8 shows the overall tuning process. The experiments were performed using the Kaldi speech recognition toolkit with speech data from the





**Fig. 4.8** Evolutionary tuning process of ASR systems

corpus of spontaneous Japanese (CSJ) [27], which is a popular Japanese speech dataset. We performed two separate experiments with training sets having different amounts of data: one consists of 240h of academic presentations, whereas the other is a 100-h subset. A common development set consisting of 10 academic presentations was used in GA, CMA-ES, and BO to evaluate the individuals for the black-box optimization. The official evaluation set defined in CSJ consisting of 10 academic presentations totalling 110 min was used as the evaluation set.

Acoustic models were trained by first creating a GMM-HMM by maximum likelihood estimation and then building a DNN-HMM by pre-training and fine-tuning using alignments generated by the GMM-HMM. For the performance evaluation of the system, the DNN-HMM was used as the final model. The language model was a 3-gram model trained on CSJ with academic and other types of presentations, which amounted to 7.5 million words in total. The vocabulary size was 72 k. Speech recognition was performed using the OpenFST WFST decoder [28]. As an initial configuration, we borrowed the settings from the Kaldi recipe for the Switchboard corpus (i.e., `egs/swbd/s5b`). We chose the recipe because this task was similar, while the language was different and because it was manually well tuned and publicly available.

For the experiments, TSUBAME 2.5 supercomputer<sup>1</sup> was used. A maximum of 44 NVIDIA K20X GPGPUs was used in parallel through the message-passing

<sup>1</sup><https://www.gsic.titech.ac.jp/en/tsubame>.

interface (MPI). We used the Spearmint package<sup>2</sup> for BO and the Python version of Hansen’s implementation<sup>3</sup> for CMA-ES.

Further, we ran two additional experiments utilizing a newer version of the Kaldi toolkit and the CSJ recipe to confirm the effect of the evolution.<sup>4</sup> One is based on the `nnet1` script and the other is based on the `chain` script. While `nnet1` adopts basic neural network structure, `chain` adopts TDNN. The definitions of the training and the evaluation sets are the same as before, but the development is different. The reason is that the recipe scripts internally make the development set by holding-out a subset of the training set, and the new recipe script has a different implementation from the old one. The new development set amounted to 6.5 h having 4000 utterances from 39 academic presentations. The experiments were performed using TSUBAME 3.0 using 30 P100 GPGPUs in parallel.

#### 4.4.2 Automatic Optimizations

In the evolution experiments, feature types, DNN structures, and learning parameters were optimized. The first and second columns of Table 4.1 describe these variables. We specify three base feature types (`feat_type`) for the GMM-HMM and DNN-HMM models: MFCC, PLP, and filter bank (FBANK). The dimensions of these features were 13, 13, and 36, respectively. The GMM-HMMs were first trained directly using the specified base features and their delta [29] and delta-delta. Then, they were re-trained using 40-dimensional LDA [30]-compressed and MLLT [31]-transformed features that were obtained from composite features made by concatenating 9 frames of the base features, and fMLLR [31]-based speaker adaptive training was performed. The DNN-HMMs were trained using features that were expanded again from the fMLLR features, splicing 5 pre- and post-context frames. The other settings were the same as those used in the Kaldi recipe.

CMA-ES uses genes represented as real-valued vectors, mappings from a real scalar value to a required type are necessary, depending on the parameters. For the mapping, we used  $\text{ceil}(10^x)$  for converting positive continuous values to integers (e.g., splice). Similarly, we used  $10^x$  for positive real values (e.g., learning rates), and  $\text{mod}(\text{ceil}(\text{abs}(x) * 3), 3)$  for a multiple choice (feature type). For example, if a value of feature type (`feat_value`) in a gene is  $-1.7$ , it is mapped to 0, and indicates MFCC. If it is 1.4, it is mapped to 2, which corresponds to PLP in our implementation. The third column of the tables presents the baseline settings, which was also used as an initial meta-parameter configuration. The MFCC-based baseline system with the 240-h training set and K20X GPGPU took 12 h for the RBM pre-training and 70 h for fine-tuning.

<sup>2</sup><https://github.com/JasperSnoek/spearmint>.

<sup>3</sup>[https://www.lri.fr/~hansen/cmaes\\_inmatlab.html](https://www.lri.fr/~hansen/cmaes_inmatlab.html).

<sup>4</sup>We ran main experiments in 2015, and the additional experiments in 2018.

**Table 4.1** Meta-parameters subject to optimization and their automatically tuned results for the system using 240-h training data

Name of meta-parameters	Description	Values obtained by evolution using 240 h training data						
		Baseline	gen1	gen2	gen3	gen4	gen5	gen6
feat_type	MFCC, FBANK, or PLP	MFCC	MFCC	MFCC	MFCC	MFCC	MFCC	MFCC
splice	Segment length for DNN	5	6	9	10	17	21	18
nn_depth	Number of hidden layers	6	7	6	6	6	5	7
hid_dim	Units per layer	2048	1755	1907	2575	1905	2904	3304
param_stddev_first	Init parameters in 1st RBM	1.0E-1	1.1E-1	1.3E-1	1.1E-1	1.2E-1	0.7E-1	0.6E-1
param_stddev	Init parameters in other RBMs	1.0E-1	1.0E-1	1.3E-1	1.0E-1	2.3E-1	1.9E-1	1.6E-1
rbm_lr_rate	RBM learning rate	4.0E-1	5.2E-1	5.7E-1	4.1E-1	4.7E-1	3.6E-1	3.6E-1
rbm_lr_rate_low	Lower RBM learning rate	1.0E-2	1.3E-2	1.1E-2	0.8E-2	0.7E-2	0.8E-2	1.1E-2
rbm_l2penalty	RBM Lasso regularization	2.0E-4	2.1E-4	2.2E-4	1.2E-4	1.6E-4	1.9E-4	1.5E-4
learn_rate	Learning rate for fine tuning	8.0E-3	7.3E-3	6.5E-3	7.8E-3	4.4E-3	5.3E-3	3.7E-3
momentum	Momentum for fine tuning	1.0E-5	0.9E-5	0.9E-5	0.4E-5	0.9E-5	0.4E-5	0.7E-5

CMA-ES with Pareto (CMA-ES+P) was used for the automatic tuning

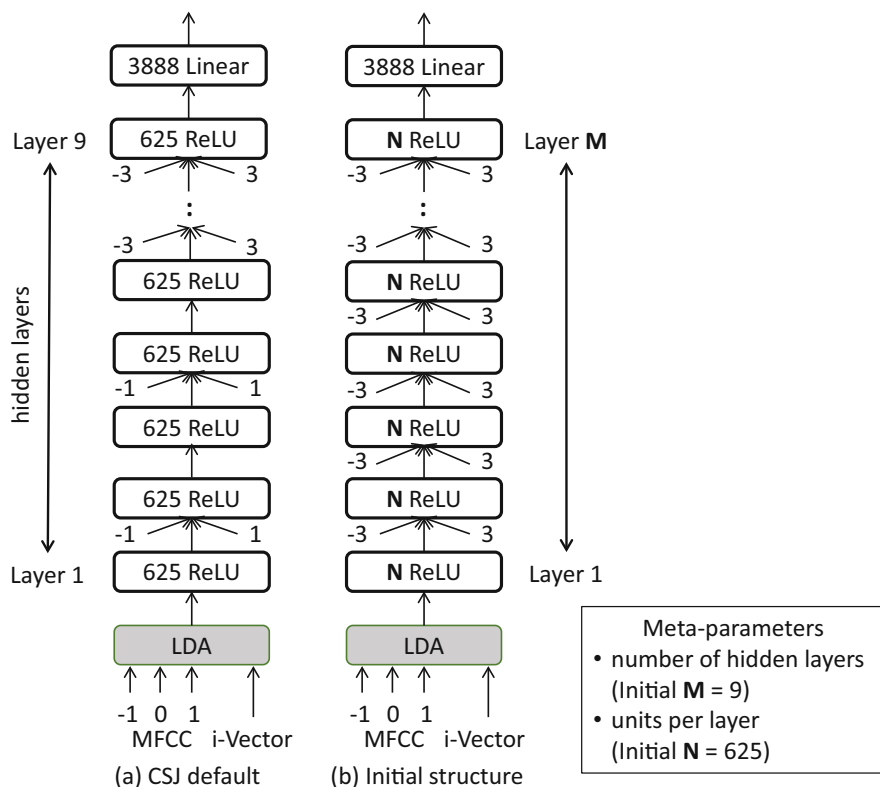
The population sizes of the black-box optimizations were 20 for the 100-h training set and 44 for the 240-h training set. The WERs used for the optimizations were evaluated using the development set. For the evaluations of each individual, a limit was introduced for the training time at each generation. If a system did not finish the training within 2.5 and 4 days for the 100-h training set and the 240-h training set, respectively, the training was interrupted and the last model in the iterative back-propagation training at that time was used as the final model. The GA-based optimization was performed based on WER and DNN size, and it is referred to as GA(WER, Size) in the following experiments. Ten initial genes ( $= N_0$ ) were manually prepared. Basically, gene A wins over gene B if its WER is lower than that of B. However, gene A having a higher WER wins over gene B if the difference of the WER is less than 0.2% and the DNN size of gene A is less than 90% of that of gene B. The tournament size  $M$  was three. For the mutation process, Gaussian noise with zero mean and 0.05 standard deviation was uniformly added to the gene. For CMA-ES, two types of experiments were performed. One was the single-objective experiment based on WER, and the other was the multi-objective experiment based on WER and DNN size using the Pareto optimality. In the following, the former is referred to as CMA-ES, and the latter is referred to as CMA-ES+P. In both cases, the initial mean vector of the multivariate Gaussian was set equal to the baseline settings. For CMA-ES+P, the maximum WER thresholds were set so that they included the top 1/2 and 1/3 of the populations at each generation for the trainings using the 100- and 240-h data sets, respectively. The BO-based tuning was performed using WER as the objective. The search range of the meta-parameters was set from 20 to 600% of the baseline configuration.

For the additional experiments using the newer version of Kaldi, we reduced the number of meta-parameters; our motivation is to evaluate the evolution in more detail under a variety of conditions. For the experiment using `nnet1`, the optimized meta-parameters were `splice`, `nn_depth`, `hid_dim`, `learn_rate` and `momentum`. These were a subset of meta-parameters, deemed to be most important in modern architectures, in Table 4.1. As an initial configuration of the evolution, we borrowed values again from the SWB recipe. For the additional experiment using `chain`, we used the initial value used in the CSJ recipe.

In these evolution experiments, TDNNs were trained using lattice-free MMI [32] without the weight averaging based parallel processing.<sup>5</sup> The initial TDNN structure was slightly modified from its original version to make the meta-parameter setting a little simpler for a variable number of layers as shown in Fig. 4.9. While in the original structure, layers 2 to 4 had different sub-sampling structures than other layers, all the layers had the same sub-sampling structure in our experiment. Note if necessary, it is possible to allow different structures for each layer by preparing separate meta-parameters for them. In total, 7 meta-parameters shown in Table 4.5 were optimized. Unlike the currently released `nnet1` script in the CSJ recipe where

---

<sup>5</sup>We disabled the default option of the parallel training to make the experiments tractable in our environment as it requires a large number of GPUs.



**Fig. 4.9** TDNN model structures for chain based systems. (a) is the original structure used in CSJ recipe and (b) is the one used as an initial configuration in our evolution experiments. The arrows with numbers at the hidden layers indicate the time splicing index

our evolution results had been integrated, the tuning of chain so far is based on the human effort by the Kaldi community, and this is the first evolution based optimization. The training set was the 240-h data set. The initial nnet1 and chain systems spent 14 and 18 h, respectively, using a P100 GPGPU. If a system did not finish the training within 24 h in the evolution processes, the training was interrupted and the last model at that time was used as the final model. The population size was 30.

## 4.5 Results

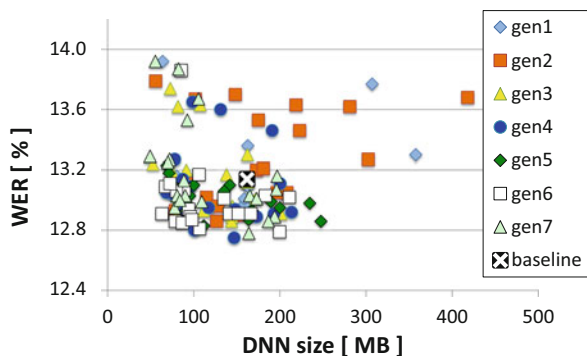
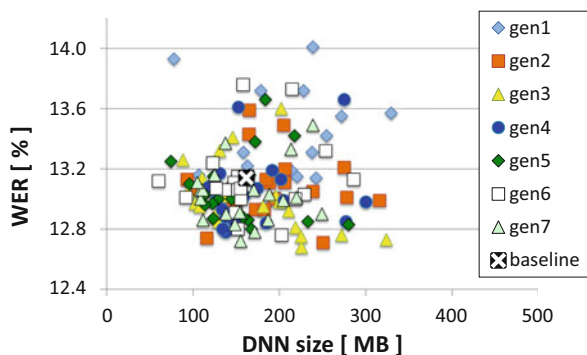
Table 4.2 shows the WERs and DNN sizes for systems with the default configuration using the 100- and 240-h training sets with one of the three types of features. Among the features, MFCC was the default in the Switchboard recipe, and it

**Table 4.2** WER of base systems

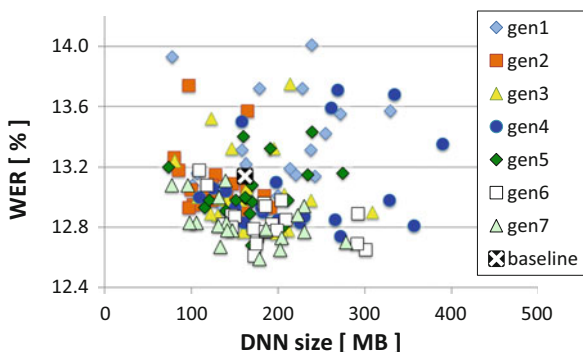
Training data	Dev set	Eval set
MFCC 100 h	14.4	13.1
PLP 100 h	14.5	13.1
FBANK 100 h	15.1	13.8
MFCC 240 h	13.5	12.5
PLP 240 h	13.6	12.5
FBANK 240 h	14.1	13.0

yielded the lowest WERs for the development set for both of the training sets. The corresponding WERs for the evaluation set were 13.1 and 12.5% for the 100- and 240-h training sets, respectively.

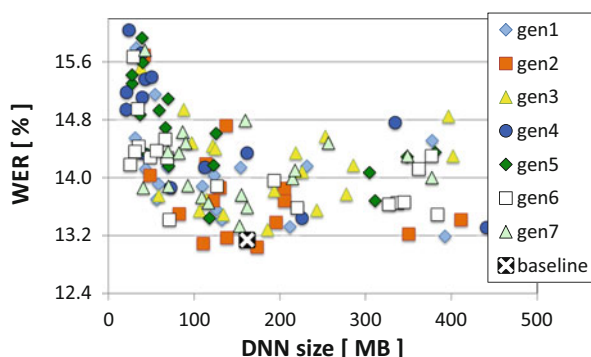
Figures 4.10, 4.11, 4.12, and 4.13 show the results when each optimization method was used with the 100-h training data. The horizontal axis is the DNN size, and the vertical axis is the WER of the evaluation set. The baseline marked on the figure is the MFCC-based system. Ideally, we want systems on the lower side of the plot when WER based single-objective optimizations (CMA-ES, BO) were performed, and on lower-left side of the plot when WER and model size based multi-objective optimizations (GA, CMA-ES+P) were performed. Figure 4.10 is a

**Fig. 4.10** Results of GA(WER, Size) when the 100-h training data were used**Fig. 4.11** Results of CMA-ES when the 100-h training data were used

**Fig. 4.12** Results of CMA-ES with Pareto optimality(CMA-ES+P) when the 100-h training data were used

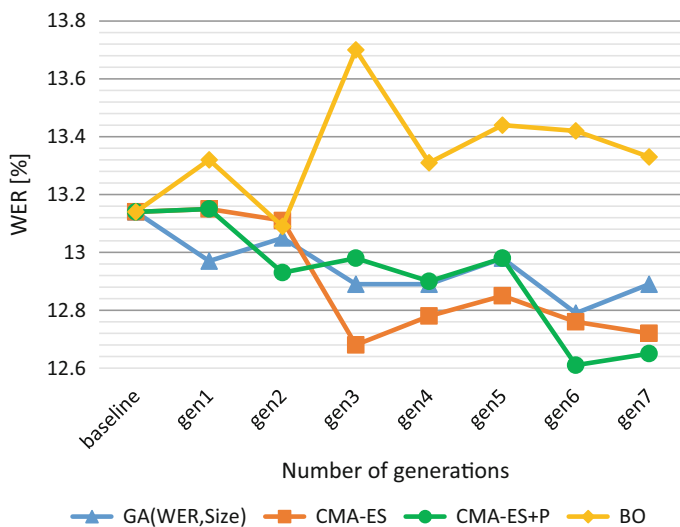


**Fig. 4.13** Results of BO when the 100-h training data were used



scatter plot of the GA(WER, Size). The distribution is oriented to the left side of the plot with the progress of generations, but the WER reduction was relatively small. Figure 4.11 presents the results of the single-objective CMA-ES. The distribution shifted towards lower WERs and lower DNN file sizes from the baseline with the progress of generations. The reason that it trended to a lower DNN size was probably due to the time limit imposed on the DNN training. In the evolution process, the ratio of individuals that hit the limit was approximately 35%. If an individual has a large DNN size, then it is likely that it hits the limit. Then, the WER is evaluated using a DNN at that time before the back-propagation converges, which is a disadvantage for that individual. Figure 4.12 presents the results of the multi-objective CMA-ES+P. The result is similar to that produced by using CMA-ES, but the distribution is oriented more to the lower-left side of the plot.

Figure 4.13 presents the results using BO for the optimization. In this case, the initial configuration is not directly specified, but the ranges of the meta-parameters are specified. We found that specifying a proper range was actually not straightforward and required knowledge of the problem. That is, if the ranges are too wide, then the initial samples are coarsely distributed in the space, and it is likely that the systems have lower performance. Meanwhile, if the ranges are too narrow, then it is likely that the optimal configuration is not included in the search space. Consequently, the improvement by BO was smaller than that by the CMA-



**Fig. 4.14** Number of generations and evaluation set WER. At each condition, the best system was chosen by using the development set

**Table 4.3** WER and DNN size of the best system when the 100-h training data was used

Opt. method	WER [%]		DNN size [MB]
	Dev	Eval	
Baseline	14.4	13.1	161.8
GA(WER, Size)	14.1	13.0	234.5
CMA-ES	14.0	12.7	225.5
CMA-ES+P	14.0	12.7	202.4
BO	14.2	13.1	110.6

ES. Carefully setting the ranges might solve the problem but would again assume expert human knowledge.

Figure 4.14 shows the WER of the evaluation set based on the best systems chosen by using the development set at each generation. CMA-ES evolved more efficiently than GA(WER, Size) and BO. Table 4.3 shows the evaluation results of the best systems chosen by the development set WER through all the generations. The evaluation set WERs by CMA-ES and CMA-ES+P were both 12.7%.<sup>6</sup> However, a smaller DNN model size was obtained by using CMA-ES with Pareto. The DNN model size by CMA-ES was 225.5 Mb, whereas it was 202.4 Mb when CMA-ES+P was used, which was 89.8% of the former. The selected feature type was all MFCC except for the 7th generation, which was PLP.

<sup>6</sup>In the table, we scored the evaluation set WERs of systems that gave the lowest development set WER through all the generations. Therefore, they were not necessarily the same as the minimum of the generation wise evaluation set WERs shown in Fig. 4.14.



Figure 4.15 shows the results of CMA-ES+P using the 240-h training data. Approximately 70% of the individuals completed the training within the limit of 4 days. This figure shows that the distributions shifted towards lower WERs and lower DNN file sizes with the progress of generations.

Figure 4.16 shows the WERs of the best systems selected at each generation based on the development set WER when the 240-h training set was used. Although the development set error rate monotonically decreased with the number of the generation, the evaluation set error rate appeared to be saturated after the fourth

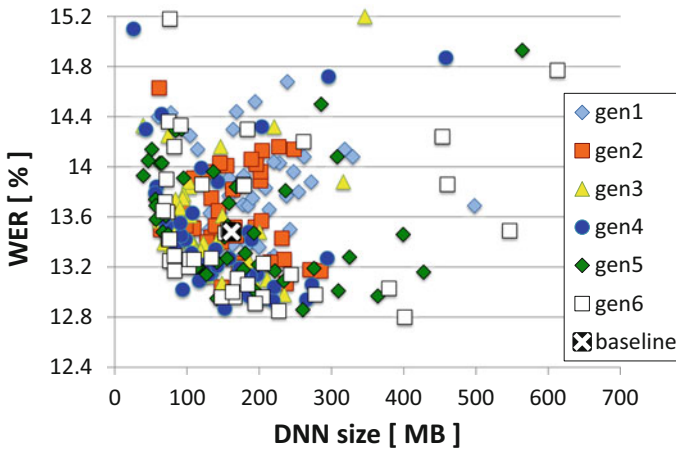


Fig. 4.15 The DNN model size and the development set WER when the 240-h training set was used with CMA-ES+P. The results of the  $n$ -th generation are denoted as “gen  $n$ ”

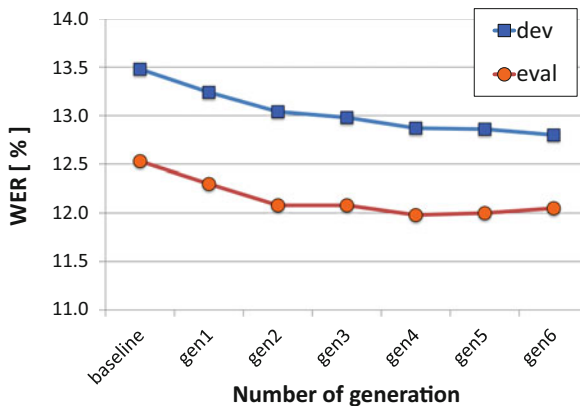
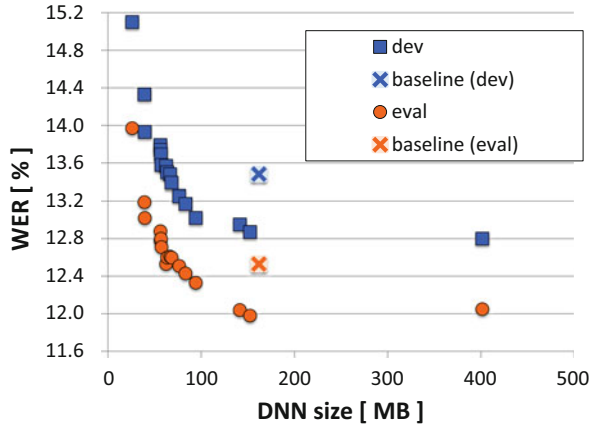


Fig. 4.16 The development and evaluation set WERs of the best systems at each generation when the 240-h training set was used with CMA-ES+P. The systems were chosen by the development set WER. In the figure “dev” and “eval” indicate the results of the development and the evaluation sets, respectively

**Fig. 4.17** Pareto frontier derived from the results from the initial to the 6th generation using the 240-h training data. In the figure “dev” and “eval” indicate the results of the development and the evaluation sets, respectively

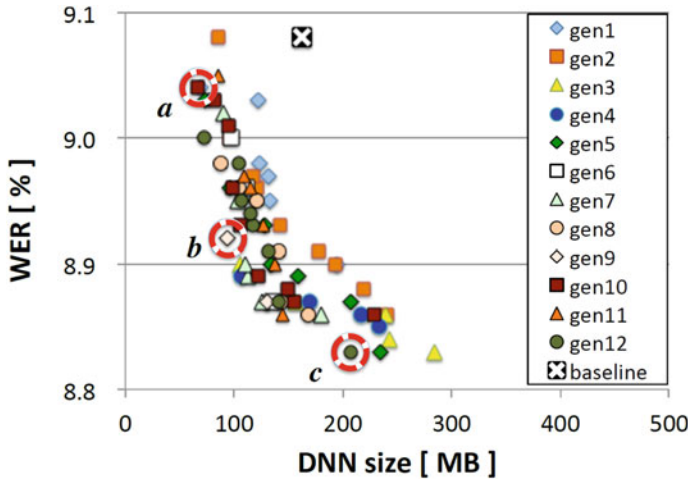


generation, which might have resulted from overfitting to the development set because we used the same development set for all the generations. The lowest WER of the development set was obtained at the 6th generation. The corresponding evaluation set error rate was 12.1%. The difference in the evaluation set WERs between the baseline (12.5%) and the optimized system (12.1%) was 0.48%, and this was statistically significant under the MAPSSWE significance test [33]. The relative WER was 3.8%.

If desired, we can choose a system from the Pareto frontier that best matches the required balance of the WER and the model size. Figure 4.17 shows the Pareto frontier derived from the results from the initial to the 6th generation using the 240-h training data. This figure shows that if we choose a system with approximately the same WER as the initial model, then we can obtain a reduced model size that is only 41% of the baseline. That is, the model size was reduced by 59%. The decoding time of the evaluation set by the reduced model was 79.5 min, which was 85.4% of the 93.5 min by the baseline. Similarly, the training time of the reduced model was 54.3% of that of the baseline model.

Columns 4 to 9 of Table 4.1 show the meta-parameter configurations obtained as the result of evolution using the 240-h training set. These are the configurations that yielded the lowest development set WERs at each generation. When we analyze the obtained meta-parameters, although the changes were not monotonic for most of the meta-parameters, we found that splice size was increased by more than three times from the initial model. We also note that the learning rate decreased by more than half from the initial condition.

As a supplementary experiment, sequential training [34] was performed using the best model at the 4th generation as an initial model. Because the sequential training is computationally intensive, it took an additional 7 days. After the training, the WER was further reduced, and a WER of 10.9% was obtained for the evaluation set. This value was lower than the WER of 11.2% obtained with sequential training using the baseline as the initial model. The difference was statistically significant, which confirms the effectiveness of the proposed method.



**Fig. 4.18** Evolution result of the nnet1 based system. The CMA-ES with Pareto based evolution (CMA-ES+P) was applied to nnet1 of the newer version of Kaldi with reduced tuning meta-parameters. The baseline is the initial model of the evolution. Only individuals on the Pareto frontier at each generation are plotted for visibility

Figure 4.18 shows the result of the additional experiments of nnet1 using the newer version of Kaldi with the reduced meta-parameters. The figure plots the development set WER and DNN model size. The evolution was performed by CMA-ES with Pareto (CMA-ES+P) and the process was repeated for 12 generations. Approximately 77% of the individuals had completed the training within the 24-h limit. In the figure, only the results of genes on the Pareto frontier at each generation were plotted for visibility. The gene marked as “a” gave the smallest DNN size, while the gene marked as “c” gave the lowest WER (There were three genes with the smallest WER and c was the one with the smallest DNN size.). Gene “b” gave both smaller DNN size and smaller WER than the initial system. Table 4.4 describes properties of these representative genes. In this experiment, the improvement in the evaluation set WER from the baseline initial configuration was minor even when

**Table 4.4** Summary of three representative genes in the additional nnet1 experiment with CMA-ES+P

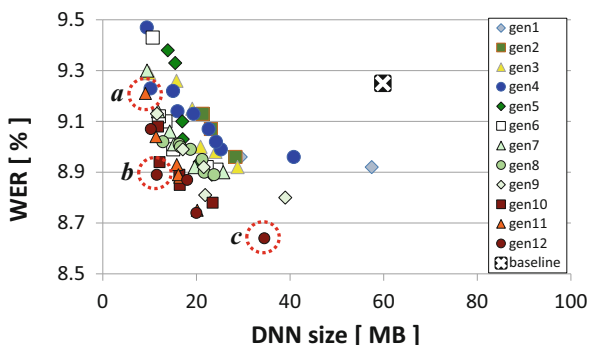
Gene	Generation	WER [%]		DNN size [MB]	Decoding time [min]
		Dev	Eval		
Baseline	0	9.1	11.9	161.0	90.5
<i>a</i>	10	9.0	12.0	66.5	70.9
<i>b</i>	9	8.9	11.8	93.6	80.4
<i>c</i>	12	8.8	11.8	207.3	99.4

Gene *a* gave the smallest model size, and gene *c* gave the lowest development set WER. Gene *b* balances the model size and WER reductions. See Fig. 4.18 for their positions

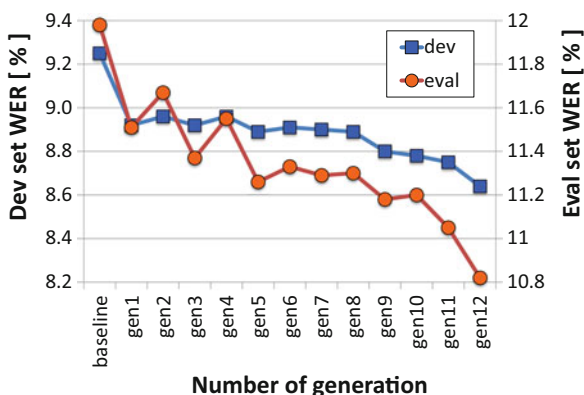
choosing the gene with the lowest WER in the development set. We conjecture this was probably because the initial meta-parameters were already close to optimal in terms of WER. The reduction of the number of meta-parameters might also have limited the room for improvement though we chose the ones that we thought important based on our previous experiments. However, the evolution had an effect of reducing the DNN size. When the gene “*b*” was chosen, it gave slightly lower WER on the evaluation set and largely reduced DNN size of 93.6 (MB), which was 58% of the initial model of 161.0 (MB). If the gene “*a*” was chosen, the WER of the evaluation set slightly increased from 11.9 to 12.0%, but the model size reduced to 66.5 (MB), which was only 40% of the initial model. Accordingly, the decoding time of the evaluation set was reduced from 90.5 to 70.9 min.

Figure 4.19 shows the result of the evolution based optimization of the chain script. Approximately 63% of the individuals completed the training within the 24-h limit. In this case, larger improvement than nnet1 was obtained both in reducing the WER and the model size. Figure 4.20 shows the WERs of the best systems selected at each generation based on the development set WER. While there was a little random behavior in the evaluation set WER, overall, a consistent trend of WER reduction was observed both in the development and the evaluation set. Table 4.5 shows corresponding changes of the meta-parameters. Different from the changes

**Fig. 4.19** Evolution result of the chain based system. The CMA-ES with Pareto based evolution (CMA-ES+P) was applied to the chain script of the newer version of Kaldi. Only individuals on the Pareto frontier at each generation are plotted for visibility



**Fig. 4.20** The development and evaluation set WERs of the best TDNN-based systems at each generation with CMA-ES+P. The systems were chosen by the development set WER. The results of the development and the evaluation sets are indicated by “dev” and “eval,” respectively



**Table 4.5** Meta-parameters subject to optimization and their automatically tuned results for the TDNN-based system

Description of meta-parameters	Baseline	Values obtained by evolution using 240h training data											
		gen1	gen2	gen3	gen4	gen5	gen6	gen7	gen8	gen9	gen10	gen11	gen12
Number of hidden layers	9	8	12	9	8	11	9	8	9	10	9	12	10
Units per layer	625	641	308	518	320	354	384	349	461	311	282	427	427
Learning rate during the initial iteration	1.0E-3	1.1E-3	1.0E-3	1.5E-3	1.2E-3	1.9E-3	1.4E-3	1.8E-3	3.0E-3	1.3E-3	7.0E-3	1.1E-2	1.1E-2
Learning rate during the final iteration	1.0E-4	0.7E-4	1.4E-4	0.8E-4	1.1E-4	0.9E-4	0.9E-4	0.6E-4	0.6E-4	0.7E-4	0.4E-4	0.9E-4	0.9E-4
Weight of cross-entropy cost	1.0E-1	1.1E-1	0.7E-1	1.6E-1	1.3E-1	1.4E-1	1.3E-1	1.4E-1	0.9E-1	1.5E-1	1.2E-1	1.4E-1	1.4E-1
Coefficient for l2 norm	5.0E-5	3.5E-5	5.5E-5	4.4E-5	3.0E-5	2.1E-5	2.7E-5	1.7E-5	1.5E-5	2.9E-5	0.5E-5	0.7E-5	0.7E-5
Coefficient for leaky hmn	1.0E-1	1.5E-1	1.1E-1	1.0E-1	1.9E-1	2.1E-1	1.6E-1	3.0E-1	4.7E-1	1.0E-1	4.0E-1	4.0E-1	4.0E-1

CMA-ES with Pareto (CMA-ES+P) was used for the automatic tuning. A gene was selected at each generation that gave the best development set WER

**Table 4.6** Summary of three representative genes in the additional chain experiment

Gene	Generation	WER [%]		DNN size [MB]	Decoding time [min]
		Dev	Eval		
CSJ default		9.3	11.7	53.7	22.6
Baseline	0	9.3	12.0	59.7	23.0
<i>a</i>	11	9.2	11.9	9.1	13.1
<i>b</i>	12	8.9	11.5	11.5	13.4
<i>c</i>	12	8.6	10.8	34.5	16.1

Gene *a* gave the smallest model size, and gene *c* gave the lowest development set WER. Gene *b* balances the model size and WER reductions. See Fig. 4.19 for their positions

of the WERs, it is seen that none of their changes was monotonic revealing their complex mutual interactions. A remarkable change after 12 generations was the large reduction of units in the hidden layers (units per layer) from 625 of the baseline to 427.

In Fig. 4.19, three representative genes are marked as in the `nnet1` results. Table 4.6 describes their details. The evaluation set WER of the gene *b* was 11.5% and it was 0.5% lower than the baseline initial structure. While the improvement was only 0.2% when compared to the CSJ default (11.7%), the model size reduction was significant from 53.7 (MB) to 11.5 (MB). When the gene *c* was used, evaluation set WER was 10.8% and the relative reduction was 7.8 and 9.7% compared to the CSJ default and the baseline initial configuration, respectively. Their differences were both statistically significant by the MAPSSWE test. Moreover, the model size was reduced to 57.7% of the original size. The decoding time of 22.6 min of the CSJ default settings was reduced to 16.1 min.

## 4.6 Conclusion

In this chapter, we have introduced the basic principles of spoken language processing, focusing on speech recognition. We have performed an automatic optimization of the meta-parameters by using evolutionary algorithms without human expert elaboration. In the experiments using the 100-h training set, multi-objective GA, CMA-ES, CMA-ES with Pareto (CMA-ES+P) and BO were compared. Both of the CMA-ES methods and GA yielded lower WERs than the baseline. Among them, CMA-ES and CMA-ES+P provided lower WERs than GA. By using CMA-ES+P to jointly minimize the WER and the DNN model size, a smaller DNN size than single-objective CMA-ES was obtained while keeping the WER. CMA-ES was more convenient for optimizing speech recognition systems than BO, which requires the ranges of the meta-parameters to be specified. Moreover, we ran additional experiments using the newer version of the Kaldi toolkit and demonstrated the consistent effectiveness of the CMA-ES+P based approach. Especially, the tuned

chain system was significantly superior to the default system both in WER and the model size. Other than experiments introduced here, we have also applied CMA-ES to language modeling and neural machine translation and have achieved automatic performance improvements [35, 36].

When the meta-parameter tuning is applied to the neural network training, there is a double structure of learning; one is the estimation of the neural network connection weights, and the other is the meta-parameter tuning of the network structure and the learning conditions. Currently, the tuning process only uses the performance score, and the learned network weight parameters are all discarded at each generation. Future work includes improving the optimization efficiency by introducing a mechanism to transmit knowledge learned by ancestors to descendants.

## References

1. Davis, S.B., Mermelstein, P.: Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Trans. Acoust. Speech Signal Process.* **28**(4), 357–366 (1980)
2. Hermansky, H.: Perceptual linear predictive (PLP) analysis of speech. *J. Acoust. Soc. Am.* **87**(4), 1738–1752 (1990)
3. Odell, J.J.: The use of context in large vocabulary speech recognition, Ph.D. Thesis, Cambridge University (1995)
4. Chorowski, J.K., Bahdanau, D., Serdyuk, D., Cho, K., Bengio, Y.: Attention-based models for speech recognition. In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 577–585 (2015)
5. Graves, A., Mohamed, A.-R., Hinton, G.: Speech recognition with deep recurrent neural networks. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6645–6649. IEEE, Piscataway (2013)
6. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3104–3112 (2014)
7. Vinyals, O., Le, Q.: A neural conversational model. Preprint. arXiv:1506.05869 (2015)
8. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. Preprint. arXiv:1409.0473 (2014)
9. Bellman, R.E., Dreyfus, S.E.: *Applied Dynamic Programming*. Princeton University Press, Princeton (1962)
10. Papineni, K., Roukos, S., Ward, T., Zhu, W.-J.: BLEU: a method for automatic evaluation of machine translation. In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, Stroudsburg, ACL '02*, pp. 311–318. Association for Computational Linguistics, Stroudsburg (2002)
11. Hansen, N., Müller, S.D., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evol. Comput.* **11**(1), 1–18 (2003)
12. Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J., Schmidhuber, J.: Natural evolution strategies. *J. Mach. Learn. Res.* **15**(1), 949–980 (2014)
13. Akimoto, Y., Nagata, Y., Ono, I., Kobayashi, S.: Bidirectional relation between CMA evolution strategies and natural evolution strategies. In: *Proceedings of Parallel Problem Solving from Nature (PPSN)*, pp. 154–163 (2010)

14. Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Proceedings of the 12th International Conference on Neural Information Processing Systems, NIPS'99, pp. 1057–1063 (1999)
15. Brochu, E., Cora, V.M., De Freitas, N.: A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. Preprint. arXiv:1012.2599 (2010)
16. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Advances in Neural Information Processing Systems 25 (2012)
17. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press, Cambridge (2006)
18. Miettinen, K.: Nonlinear Multiobjective Optimization. Springer, Berlin (1998)
19. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **6**(2), 182–197 (2002)
20. Deb, K., Kalyanmoy, D.: Multi-Objective Optimization Using Evolutionary Algorithms. John Wiley & Sons, Inc., New York (2001)
21. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Trans. Evol. Comput.* **3**(4), 257–271 (1999)
22. David Schaffer, J.: Multiple objective optimization with vector evaluated genetic algorithms. In: Proceedings of the 1st International Conference on Genetic Algorithms, Hillsdale, pp. 93–100. L. Erlbaum Associates Inc., Mahwah (1985)
23. Hajela, P., Lin, C.Y.: Genetic search strategies in multicriterion optimal design. *Struct. Optim.* **4**(2), 99–107 (1992)
24. Hernandez-Lobato, D., Hernandez-Lobato, J., Shah, A., Adams, R.: Predictive entropy search for multi-objective Bayesian optimization. In: Balcan, M.F., Weinberger, K.Q. (eds.) Proceedings of The 33rd International Conference on Machine Learning, New York, 20–22 Jun. Proceedings of Machine Learning Research, vol. 48, pp. 1492–1501 (2016)
25. Knowles, J.: ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Trans. Evol. Comput.* **10**(1), 50–66 (2006)
26. Moriya, T., Tanaka, T., Shinozaki, T., Watanabe, S., Duh, K.: Evolution-strategy-based automation of system development for high-performance speech recognition. *IEEE/ACM Trans. Audio Speech Lang. Process.* **27**(1), 77–88 (2019)
27. Furui, S., Maekawa, K., Isahara, H.: A Japanese national project on spontaneous speech corpus and processing technology. In: Proceedings of ASR'00, pp. 244–248 (2000)
28. Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., Mohri, M.: OpenFST: a general and efficient weighted finite-state transducer library. In: Implementation and Application of Automata, pp. 11–23. Springer, Berlin (2007)
29. Furui, S.: Speaker independent isolated word recognition using dynamic features of speech spectrum. *IEEE Trans. Acoustics Speech Signal Process.* **34**, 52–59 (1986)
30. Haeb-Umbach, R., Ney, H.: Linear discriminant analysis for improved large vocabulary continuous speech recognition. In: Proceedings of International Conference on Acoustics, Speech, and Signal Processing, vol. 1, pp. 13–16 (1992)
31. Gales, M.J.F.: Maximum likelihood linear transformations for HMM-based speech recognition. *Comput. Speech Lang.* **12**, 75–98 (1998)
32. Povey, D., Peddinti, V., Galvez, D., Ghahremani, P., Manohar, V., Na, X., Wang, Y., Khudanpur, S.: Purely sequence-trained neural networks for ASR based on lattice-free MMI. In: Interspeech, pp. 2751–2755 (2016)
33. Gillick, L., Cox, S.: Some statistical issues in the comparison of speech recognition algorithms. In: Proceedings of International Conference on Acoustics, Speech, and Signal Processing, pp. 532–535 (1989)
34. Vesely, K., Ghoshal, A., Burget, L., Povey, D.: Sequence-discriminative training of deep neural networks. In: Proceedings of Interspeech, pp. 2345–2349 (2013)



35. Tanaka, T., Moriya, T., Shinozaki, T., Watanabe, S., Hori, T., Duh, K.: Automated structure discovery and parameter tuning of neural network language model based on evolution strategy. In: Proceedings of the 2016 IEEE Workshop on Spoken Language Technology, pp. 665–671 (2016)
36. Qin, H., Shinozaki, T., Duh, K.: Evolution strategy based automatic tuning of neural machine translation systems. In: Proceeding of International Workshop on Spoken Language Translation (IWSLT), pp. 120–128 (2017)