CHAPTER 4

INCREMENTAL DERIVATIONS OF GRAPHICAL EM

In radar ES applications, new information from a battlefield or other sources often becomes available in blocks at different times. Since timely protection against threats is often of vital importance, fast *incremental* learning of SCFG probabilities is an undisputed asset. In this context, incremental learning refers to the ability to adapt SCFG probabilities from new blocks of training sequences, without requiring access to training sequences used to learn the existing SCFG. Incremental learning must also preserve previously-acquired knowledge.

From a radar ES perspective, the IO and VS algorithm have the drawback of being computationally very demanding, and not allowing for incremental learning of SCFG probabilities. If new training data becomes available, it must be added with all previous training sequences, and used to re-train the production rule probabilities from the start. This process has a high overall time complexity. The fast alternatives TS and gEM are only semi-incremental: if new data is added, the results from the pre-processing can be stored incrementally, but a new iterative process has to be redone from scratch using all cumulative data resulting from pre-processing. Faster pre-processing allows reducing the overall time complexity when new data becomes available. Only HOLA is incremental. If new data is added, one just updates the histogram $N().o$, and re-starts the iterative process using the result from previous training. This should theoretically lead to small convergence time, since the production rule probabilities incorporate a priori information on the probability distributions.

Experimental results (presented in Sub-section 6.1) indicate that since HOLA optimizes the relative entropy between two distributions, it yields lower accuracy than TS or gEM.

In addition, TS should lead to very high time complexity per iteration for ambiguous grammars. Thus, in order to achieve the best trade-off in terms of accuracy and resources requirements, one may either try to design a more accurate HOLA algorithm, or to design an incremental gEM algorithm. In light of techniques found in the literature, this last option seems more feasible and is developed in this chapter.

In literature, several variants of the EM algorithm can provide inspiration for the development of an incremental gEM algorithm. Several EM-based algorithms for on-line or sequential optimization of parameters have been proposed for different applications. Neal and Hinton (1998) proposed an incremental version of the basic EM algorithm based on the fact that some parameters are reestimated using a vector of sufficient statistics. The incremental EM algorithm consists in computing the new sufficient statistics of a selected block of data in the E-step, and reestimating the parameters by combining the old sufficient statistics (of the unused dataset) and the new ones in the M-step. With this approach, one can choose to update the parameters by selecting data blocks cyclicly, or by giving preference to some scheme for which the algorithm has not converged. The authors have observed that this leads to a shorter convergence time.

Based on this general approach, several incremental algorithms have been developed in the literature, mainly to estimate HMM probabilities. For instance, Digalakis (1999) proposed an online EM algorithm that updates the parameters on a HMM after each sentence, with only one pass through the data. Gotoh *et al.* (1998) presented an incremental estimation approach for HMM parameters in order to accelerate the reestimation of the probabilities through a process selecting different blocks of data to update the parameters, until convergence is achieved.

Sato and Ishii (2000) proposes an on-line version of the EM algorithm for normalized Gaussian networks. It is based on an approximation of the weighted means of the variables with respect to the posterior probability, from its previous value. Then these approximated

weighted means are combined during the reestimation phase of the process to lead to the new estimation of the parameters.

A different reference approach to estimate parameters using incomplete data is proposed by Titterington (1984). This recursive approach is adapted to EM technique in the following way. A classical auxiliary function is computed during the E-step, and recursion is used to approximate the re-estimation of the parameter θ during the M-step. At iteration $k+1$, the recursion computes θ_{k+1} using (a) the previous value θ_k; (b) a Fisher matrix corresponding to a complete observation of the data; (c) a vector of scores. Chung and Bohme (2003) improves this technique by proposing an adaptive procedure to determine the step size at each recursion, to reduce the convergence time. Jorgensen (1999) investigates a dynamic form of EM, based on Titterington's algorithm, to reestimate mixture proportions that require a single EM update for each observation.

Finally, Baldi and Chauvin (1994) proposes a smooth on-line algorithm to learn the parameters of a HMM. It is not based on EM or Baum-Welch algorithms as with the majority of the existing algorithms, but on a simple gradient descent to minimize negative likelihood. To avoid non-positive values, production rule probabilities are expressed using an exponential form, upon which this algorithm is applied. Considering a given sample of data, the frequency of the rules in the derivation trees are computed, and the gradient is computed by normalizing the frequencies and subtracting them from the current value of the corresponding production rule. Probabilities are reestimated based on those of the previous iteration. The authors ensure that this algorithm can either be applied in a batch, sequential, or on-line fashion.

The rest of this chapter presents two versions of gEM, named incremental gEM (igEM) and online igEM (oigEM), that are based on research by Neal and Hinton (1998), and by Sato and Ishii (2000), respectively. This work has also been introduced in (Latombe *et al.*, 2006f) and (Latombe *et al.*, 2006d). These algorithms have the advantage of al-

lowing to adapt to incremental learning as desired in this thesis. Prior knowledge of all the training sequences is not required for training, such as with Titterington's algorithm. Moreover, the re-initialization of the production rule probabilities with igEM and oigEM may help avoid getting trapped in local optima (*cf.* Section 4.3). Techniques based on gradient descent starting from the previous SCFG probabilities may encounter difficulties during incremental learning, as they are likely to provide solutions that get trapped in local optima.

4.1 Incremental gEM

4.1.1 The Expectation-Maximization algorithm

In many real-world problems, it is impossible to obtain complete datasets, without any corrupted or even missing sample. Moreover, one often has access to data generated by a system whose corresponding state is unknown but of vital importance. The EM algorithm allows the estimation of a parameter θ by optimizing some chosen likelihood in problems depending on unobserved variables. It consists of two steps:

- **The *Expectation* step (E-step):** given a set of observed variables $x_1^n = \{x_1, ..., x_n\}$ and a set of unobserved ones $Z_1^n = \{Z_1, ..., Z_n\}$, it computes $P(Z_i|x_i, \theta)$ to get the objective function $Q(\theta, \theta^{(m)}) = E_{Z_1^n}[logP(x_1^n, Z_1^n|\theta)|x_1^n, \theta^{(m)}]$, where θ is the parameter to estimate and $\theta^{(m)}$ its estimation at iteration m;
- **The *Maximization* step (M-step):** compute $\theta^{(m+1)} = \text{argmax}_\theta\{Q(\theta, \theta^{(m)})\}$.

4.1.2 Incremental EM

Neal and Hinton (1998) propose a version of the EM algorithm based on sufficient statistics instead of the raw data. Given a sample $\{x_1, ..., x_n\}$ governed by the density function $f(x|\theta)$, the statistic $S(x)$ of x "is sufficient for θ if the conditional distribution of x given

$S(x) = s$ is independent of θ" (Scott and Nowak, 2004). This can be interpreted according to the following statement: "any inference strategy based on $f_\theta(x)$ may be replaced by a strategy based on $f_\theta(s)$" (Scott and Nowak, 2004).

Therefore, considering $\{x_1^n, Z_1^n\}$ instead of x, given the vector of sufficient statistics $s(x, Z)$, the EM algorithm for the m^{th} iteration becomes:

- **E-step:** set $\tilde{s}^{(m+1)} = E_{\tilde{P}}[s(x, Z)]$, where $\tilde{P}(Z) = P(Z|x, \theta^{(m)})$; (4.1)
- **M-step:** set $\theta^{(m+1)}$ to the θ with maximum likelihood given $\tilde{s}^{(m+1)}$.

These authors introduced an algorithm called *incremental EM* that was shown to reduce the number of iterations needed to converge. After having divided the original dataset $\Omega = \{x, Z\}$ in blocks $\{\Omega_1, ..., \Omega_n\} = \{x_1, Z_1, ..., x_n, Z_n\}$, the vector of sufficient statistics corresponding to each block Ω_j is initialized to an initial guess $s_j^{(0)}$. Thus, for the m^{th} iteration, the successive E and M steps are applied as follows:

- **E-step:** Select a block Ω_i to be updated;

  Set $\tilde{s}_j^{(m+1)} = \tilde{s}_j^{(m)}$ for every $j \neq i$; (4.2)

  Set $\tilde{s}_i^{(m+1)} = E_{\tilde{P}_i}[s(x, Z)]$, where $\tilde{P}_i(Z) = P(Z_i|x_j, \theta^{(m)})$; (4.3)

  Set $\tilde{s}^{(m+1)} = \tilde{s}_i^{(m+1)} + \tilde{s}_j$; (4.4)

- **M-step:** set $\theta^{(m+1)}$ to the θ with maximum likelihood given $\tilde{s}^{(m+1)}$;
- If convergence is reached, store $\tilde{s}_i = \tilde{s}_i^{(m+1)}$.

4.1.3 Incremental EM for SCFGs

In our context, the observed variables $\{x_i\}$ correspond to a given sequence, while the unobserved ones $\{Z_i\}$ correspond to the associated derivation trees $\{d_{x_i}\}$. Therefore,

in gEM, η (see Eq. 3.28) is a vector of sufficient statistics of θ, and $\tilde{s}$ can be identified with η. The E-step now consists in computing the vector η, while the M-step consists in reestimating the associated vector of production probabilities θ. It will be shown here how gEM can be modified to re-estimate these production probabilities incrementally.

For the m^{th} iteration, after having divided the original dataset Ω of sequences in blocks $\{\Omega_1, ..., \Omega_n\}$, each block containing a certain number of sequences, this concept may be adapted to gEM as follows:

- **E-step:** Select a block Ω_i to be updated;

  Compute $\eta^{(m+1)}$ using Alg. 18 and 19 on Ω_i (see Section 3.2.2);

  Set $\eta'^{(m+1)} = \eta^{(m+1)} + \sum_{j \neq i} \eta_j$; (4.5)

- **M-step:** re-estimate $\theta^{(m+1)}$ using Eq. 3.27 on $\eta'^{(m+1)}$;

- If convergence is reached, store $\eta_i = \eta^{(m+1)}$.

4.1.4 Incremental gEM (igEM)

Neal and Hinton's incremental algorithm updates the parameters by presenting the data sequentially, or by presenting the data in order to give preference to a block for which the algorithm has not yet stabilized. However this algorithm is not exactly incremental as defined in Chapter 2. Indeed, at every iteration, all the data is considered to compute $\tilde{s}^{(m+1)}$, while the purpose of this work is to learn new data not present at the beginning of the training.

The following approximation of Neal's algorithm is proposed. Consider that the SCFG probabilities have previously been learned from a block of sequences Ω_1, and that the final value of η, referred to as η_1, is stored. Then, to learn a new block Ω_2, the m^{th} iteration of the E and M-steps of the incremental EM algorithm, become:

- **E-step:** Compute $\eta^{(m+1)}$ using Alg. 18 and 19 on Ω_2;

$$\text{Set } \eta'^{(m+1)} = \eta^{(m+1)} + \eta_1; \tag{4.6}$$

- **M-step:** re-estimate θ using Eq. 3.27;

- If convergence is reached, store the new $\eta_2 = \eta'^{(m+1)}$.

The only difference with Neal's algorithm lays in the fact that Ω_2 was not considered during the initial estimation of θ. If another dataset Ω_3 is available after training has been led on $\{\Omega_1, ..., \Omega_2\}$, the procedure remains the same, using η_2 and applying the E-step on Ω_3.

A version of Alg. 17 that allows for incremental learning is given in Alg. 26. Note that in contrast to the original gEM, `Get-Inside-Probs()` is performed only on new data Ω_{i+1}, and `Get-Expectation()` produces η^{m+1}. The framed numbers highlight the difference with Alg. 17.

Algorithm 26: `Incremental gEM()`

1- load η_i;
2- `Get-Inside-Probs()` on Ω_{i+1};
3- $loglikelihood(0) = \sum_{x\in\Omega} \alpha_x(0, L(x)|Start)$;
4- **while** $loglikelihood^{(m)} - loglikelihood^{(m-1)} > \varepsilon$ **do**
  5- `Get-Expectation()`;
  6- $\eta'^{(m+1)} = \eta_i + \eta^{(m+1)}$;
  7- **foreach** $(A \rightarrow \lambda)$ **do**
    8- $\theta'(A \rightarrow \lambda) = \frac{\eta'^{(m+1)}(A\rightarrow\lambda)}{\sum_\mu \eta'^{(m+1)}(A\rightarrow\lambda')}$;
  9- $m = m + 1$;
  10- `Get-Inside-Probs()`;
  11- $loglikelihood(m) = \sum_{x\in\Omega} \alpha_x(0, L(x)|Start)$;
12- save $\eta_{i+1} = \eta^{(m)}$;

Suppose for example that training was completed successfully for the Mercury MFR (Annex 3.1) on a training block Ω_1 of 20 sequences, and that the final values $\eta_1(Na \rightarrow$

$S1\ T6) = 132.69$ and $\eta_1(Na \rightarrow Q6\ Q6) = 197.57$ are stored. Suppose that an iteration on a new block Ω_2 of 20 sequences gives $\eta^{(1)}(Na \rightarrow S1\ T6) = 83.38$ and $\eta^{(1)}(Na \rightarrow Q6\ Q6) = 148.49$. Thus, $\theta'(Na \rightarrow S1\ T6)$ is computed as follows:

$$\begin{aligned} \theta'(Na \rightarrow S1\,T6) &= \frac{\eta'^{(1)}(Na \rightarrow S1\,T6)}{\eta'^{(1)}(Na)} = \frac{\eta_1(Na \rightarrow S1\,T6) + \eta^{(1)}(Na \rightarrow S1\,T6)}{\eta_1(Na) + \eta^{(1)}(Na)} \\ &= \frac{132.69 + 83.38}{(132.69 + 197.57) + (83.38 + 148.49)} = 0.384 \end{aligned} \tag{4.7}$$

Suppose then that the next iteration on Ω_2 gives $\eta^{(2)}(Na \rightarrow S1\ T6) = 155.01$ and $\eta^{(2)}(Na \rightarrow Q6\ Q6) = 163.09$. Thus, $\theta'(Na \rightarrow S1\ T6)$ is now computed as follows:

$$\begin{aligned} \theta'(Na \rightarrow S1\,T6) &= \frac{\eta'^{(2)}(Na \rightarrow S1\,T6)}{\eta'^{(2)}(Na)} = \frac{\eta_1(Na \rightarrow S1\,T6) + \eta^{(2)}(Na \rightarrow S1\,T6)}{\eta_1(Na) + \eta^{(2)}(Na)} \\ &= \frac{132.69 + 155.01}{(132.69 + 197.57) + (155.01 + 163.09)} = 0.444 \end{aligned} \tag{4.8}$$

This process is repeated until convergence, and the final value $\eta_2(Na \rightarrow S1\ T6) = \eta'^{(convergence)}(Na \rightarrow S1\ T6) = \eta_1(Na \rightarrow S1\ T6) + \eta^{(convergence)}(Na \rightarrow S1\ T6)$ is stored for the next block of data.

4.2 On-line Incremental gEM

4.2.1 EM for Gaussian networks

Sato and Ishii (2000) proposes an on-line version of the EM algorithm for normalized gaussian networks. For this application, the batch EM algorithm can be adapted in the following way: while the E-step remains the same as for the general EM, the M-step

computes the weighed mean, with respect to the probability computed during the E-step, of functions that allow re-estimating the parameter of the network.

Imagine that we deal with an application in which parameters can be re-estimated using the weighted mean of a given function $f()$, that depends on both observable and hidden variables. The batch EM algorithm for such an application is the following:

- **The *Expectation* step (E-step):** given a set of observed variables $\{x_1, ..., x_n\}$ and a set of unobserved ones $\{Z_1, ..., Z_n\}$, it computes $P(Z_i|x_i, \theta)$, where θ is the parameter to estimate;
- **M-step:** it computes the weighted mean over n examples of a function of parameters x with respect to the posterior probability:

$$f^*(x, Z) = \frac{1}{n} \sum_{i=1}^{n} f(x_i) P(Z_i|x_i, \theta) \tag{4.9}$$

and it re-estimates the parameters using $f^*(x, Z)$.

4.2.2 On-line EM

The principle of the Sato's on-line algorithm consists in computing iteratively an estimate of $f^*(x, Z)$, that will be denoted herein $\tilde{f}(x, Z)$. The estimate $\tilde{f}(x, Z)$ can be derived from $f(x, Z)$ as shown in Eq. 4.10.

$$\begin{aligned} \tilde{f}(x, Z) &= \chi \sum_{i=1}^{n} \left(\prod_{j=i+1}^{n} \xi(j) \right) f(x_i, Z_i) P(Z_i|x_i, \theta(i-1)) \\ where \ \chi &= \left(\sum_{i=1}^{n} \prod_{j=i+1}^{n} \xi(j) \right)^{-1} \end{aligned} \tag{4.10}$$

$0 \leq \xi(j) \leq 1$ is a time dependent discount factor, χ is a normalization coefficient that plays the role of a learning rate, and $\theta(i-1)$ is the estimator of the parameter after the $i-1^{th}$ observation x_{i-1}.

This modified weighted mean can be computed in an iterative way, by presenting the examples x_i one after the other, using the Eq. 4.11. Consider that $\tilde{f}(x_1^i, Z_1^i)$ has already been computed on $\{x_1, ..., x_i, Z_1, ..., Z_i\}$:

$$\tilde{f}(x_1^{i+1}, Z_1^{i+1}) = \tilde{f}(x_1^i, Z_1^i) + \chi(i+1)\left(f(x_{i+1}, Z_{i+1})P(Z_{i+1}|x_{i+1}, \theta(i)) - \tilde{f}(x_1^i, Z_1^i)\right) \tag{4.11}$$

It has been shown that, with an appropriate value of χ, the modified weighted mean is equal to the classical mean, and that this on-line EM algorithm is equivalent to the batch version. It has also been proved that this on-line algorithm converges to the maximum likelihood estimator.

Considering a dataset $\Omega = x_1, ..., x_n$,and the fact that training has already been completed on $\{x_1, ...x_i\}$ the on-line EM algorithms can be applied as follows:

- **E-step:** compute $P(Z_{i+1} = j|x_{i+1}, \theta(i))$;
- **M-step:** compute $\tilde{f}(x_1^{i+1}, Z_1^{i+1})$ using Eq. 4.11 and re-estimate the parameters.

4.2.3 On-line gEM for SCFGs

The on-line EM method can be applied to gEM, where x_i is a given sequence, and where Z_i corresponds to d_{x_i} in the following way. The weighting probability $P(d_x|x, \theta)$ then corresponds to the probability of having a particular derivation tree d_x, given x and θ, and

f is now identified with the frequency of the rules, N. Indeed, Bayes' theorem gives the relation between N^* and η for a particular production rule $A \rightarrow \lambda$ and a set of derivation trees $d_\Omega = \{d_x\}$:

$$\begin{aligned} N^*(A \rightarrow \lambda, d_\Omega) &= \frac{1}{|\Omega|} \sum_{x \in \Omega} N(A \rightarrow \lambda, d_x) P(d_x | x, G_s) \\ &= \frac{1}{|\Omega|} \sum_{x \in \Omega} N(A \rightarrow \lambda, d_x) \frac{P(x, d_x | G_s)}{P(x | G_s)} \\ &= \frac{1}{|\Omega|} \sum_{x \in \Omega} N(A \rightarrow \lambda, d_x) \frac{P(x, d_x | G_s)}{P(x, \Delta_x | G_s)} \\ &= \frac{\eta(A \rightarrow \lambda, d_\Omega)}{|\Omega|} \end{aligned} \tag{4.12}$$

In the this application, however, all the derivation trees have to be considered to re-estimate the production rule probabilities. Indeed, for each sequence, we have to sum the weighted frequency of the rules over all the corresponding derivation trees, as follows:

$$\begin{aligned} N^*(A \rightarrow \lambda) &= \sum_{d_\Omega \in \Delta_\Omega} N^*(A \rightarrow \lambda, d_\Omega) \\ &= \sum_{d_\Omega \in \Delta_\Omega} \frac{1}{|\Omega|} \sum_{x \in \Omega} N(A \rightarrow \lambda, d_x) \frac{P(x, d_x | G_s)}{P(x, \Delta_x | G_s)} \\ &= \frac{1}{|\Omega|} \sum_{x \in \Omega} \sum_{d_x \in \Delta_x} N(A \rightarrow \lambda, d_x) \frac{P(x, d_x | G_s)}{P(x, \Delta_x | G_s)} \\ &= \frac{1}{|\Omega|} \sum_{x \in \Omega} \frac{1}{P(x, \Delta_x | G_s)} \sum_{d_x \in \Delta_x} N(A \rightarrow \lambda, d_x) P(x, d_x | G_s) \\ &= \frac{\eta(A \rightarrow \lambda)}{|\Omega|} \end{aligned} \tag{4.13}$$

Thus, the re-estimation equation (Eq. 3.27) can be modified according to Eq. 4.13:

$$\theta(A \rightarrow \lambda) = \frac{\eta(A \rightarrow \lambda)}{\eta(A)} = \frac{\eta(A \rightarrow \lambda)/|\Omega|}{\eta(A)/|\Omega|} = \frac{N^*(A \rightarrow \lambda)}{N^*(A)}$$

$$\text{where} \qquad N^*(A) = \sum_{\lambda} N^*(A \rightarrow \lambda) \tag{4.14}$$

Suppose that $\tilde{N}_i(A \rightarrow \lambda)$ (obtained from $\{x_1, ..., x_i\}$) has already been computed. Using Eq. 4.13, it is now possible to compute $\tilde{N}_{i+1}(A \rightarrow \lambda)$ iteratively, and the modified version of $N^*(A \rightarrow \lambda)$ is computed on $\{x_1, ..., x_i\}$, as follows:

$$\begin{aligned}
\tilde{N}_{i+1}(A \rightarrow \lambda) &= \tilde{N}_i(A \rightarrow \lambda) \\
&\quad +\chi(i+1)[\sum_{d_{x_{i+1}} \in \Delta_{x_{i+1}}} N(A \rightarrow \lambda, d_{x_{i+1}}) P(d_{x_{i+1}} | x_{i+1}, G_s) \\
&\quad -\tilde{N}_i(A \rightarrow \lambda)] \\
&= \tilde{N}_i(A \rightarrow \lambda) \\
&\quad +\chi(i+1)[\eta(A \rightarrow \lambda, x_{i+1}) - \tilde{N}_i(A \rightarrow \lambda)]
\end{aligned} \tag{4.15}$$

Based on Eq. 4.15, the on-line EM algorithm can be adapted to gEM in the following way. Assuming that training has already been completed on $\{x_1, ...x_i\}$, the m^{th} iteration gives:

- **E-step:** Compute η_{i+1} using Alg. 18 and 19 on x_{i+1};
  Set $\tilde{N}_{i+1} = \tilde{N}_i + \chi(i+1)[\eta_{i+1} - \tilde{N}_i]$; (4.16)

- **M-step:** re-estimate $\theta^{(i+1)}(A \rightarrow \lambda) = \frac{\tilde{N}_{i+1}(A \rightarrow \lambda)}{\tilde{N}_{i+1}(A)}$; (4.17)

- If convergence is reached, store $\tilde{N}_{i+1}$.

4.2.4 On-line incremental gEM (oigEM)

The algorithm described until now is said to be on-line algorithm because it consists in presenting the examples of the training dataset one after the other, from the first to the last one, and looping until convergence. However, once more, it is not incremental as defined in Sec. 2. First, this on-line algorithm can be modified in order to make it sequential for blocks of data instead of only single examples. Consider two data blocks $\{\Omega_1, ..., \Omega_i\}$ and Ω_{i+1} of sizes $|\Omega_{1,...,i}|$ and $|\Omega_{i+1}|$ and suppose that $\tilde{N}_i(A \rightarrow \lambda)$, after training on $\{\Omega_1, ..., \Omega_i\}$, was already computed.

$$\begin{aligned}
\tilde{N}_{i+1}(A \rightarrow \lambda) &= \tilde{N}_i(A \rightarrow \lambda) \\
&\quad +\chi(i+1)[\frac{\sum_{x \in \Omega_{i+1}} \sum_{d_x \in \Delta_x} N(A \rightarrow \lambda, d_x) P(d_x|x, G_s)}{|\Omega_{i+1}|} \\
&\quad -\tilde{N}_i(A \rightarrow \lambda)] \\
&= \tilde{N}_{i+1}(A \rightarrow \lambda) \\
&\quad +\chi(i+1)[\frac{\sum_{x \in \Omega_{i+1}} \eta(A \rightarrow \lambda, x)}{|\Omega_{i+1}|} - \tilde{N}_i(A \rightarrow \lambda)] \\
&= \tilde{N}_{i+1}(A \rightarrow \lambda) + \chi(i+1)[\frac{\eta_{i+1}(A \rightarrow \lambda)}{|\Omega_{i+1}|} - \tilde{N}_i(A \rightarrow \lambda)] \quad (4.18)
\end{aligned}$$

Thus, inspiring from the incremental EM (Neal and Hinton, 1998) and taking in account the fact that some scheme for which the algorithm has not yet stabilized can be privileged, the following incremental algorithm can be defined. For each block, the SCFG is trained over several iterations until convergence. Suppose that training has already been successfully performed on the first block Ω_1, and that, for each rule $A \rightarrow \lambda$, $\tilde{N}_1(A \rightarrow \lambda)$ – the final value of $\tilde{N}(A \rightarrow \lambda)$ – is stored. In order to learn a new block Ω_2, $\tilde{N}_2(A \rightarrow \lambda)$ can be computed for the m^{th} iteration according to:

- **E-step:** Compute $\eta^{(m+1)}$ using Alg. 18 and 19 on Ω_2;

$$\text{Set } \tilde{N}^{(m+1)} = \tilde{N}_1 + \chi(m+1)[\frac{\eta_2^{(m+1)}}{|\Omega_2|} - \tilde{N}_1]; \tag{4.19}$$

- **M-step:** re-estimate $\theta'(A \rightarrow \lambda) = \dfrac{\tilde{N}^{(m+1)}(A \rightarrow \lambda)}{\tilde{N}^{(m+1)}(A)}$; (4.20)

- If convergence is reached, store $\tilde{N}_2 = \tilde{N}^{(m+1)}$.

A version of Alg. 17 that allows for on-line incremental learning is given in Alg. 27. In this algorithm, steps allow to manage updates to the value $\tilde{N}$. Note that in contrast to the original gEM, `Get-Inside-Probs()` is performed only on new data Ω_{i+1}, and `Get-Expectation()` produces $\eta^{(m+1)}$. The framed numbers highlight the difference with Alg. 17.

Algorithm 27: `On-line incremental gEM()`

1- load $\tilde{N}_i$;
2- `Get-Inside-Probs()` on Ω_{i+1};
3- $loglikelihood(0) = \sum_{x \in \Omega} \alpha_x(0, L(x)|Start)$;
4- **while** $cond^{(m)} - cond^{(m-1)} > \varepsilon$ **do**
  5- `Get-Expectation()` ;
  6- **foreach** $(A \rightarrow \lambda)$ **do**
    7- $\tilde{N}^{(m+1)}(A \rightarrow \lambda) = \tilde{N}_i(A \rightarrow \lambda) + \chi(m+1)[\frac{\eta^{(m+1)}(A \rightarrow \lambda)}{|\Omega_{i+1}|} - \tilde{N}_i(A \rightarrow \lambda)]$;
    8- $\theta(A \rightarrow \lambda) = \frac{\tilde{N}^{(m+1)}(A \rightarrow \lambda)}{\tilde{N}^{(m+1)}(A)}$;
  9- $m = m + 1$;
  10- `Get-Inside-Probs()`;
  11- $loglikelihood(m) = \sum_{x \in \Omega} \alpha_x(0, L(x)|Start)$;
13- save $\tilde{N}_{i+1}$;

Suppose that training was completed successfully for the Mercury MFR (Annex 3.1) on a training block Ω_1 of 20 sequences, and that the final values $\tilde{N}_1(Na \rightarrow S1\ T6) = \eta_1(Na \rightarrow S1\ T6)/20 = 132.69/20 = 6.63$ and $\tilde{N}_1(Na \rightarrow Q6\ T6) = \eta_1(Na \rightarrow$

$Q6\ Q6)/20 = 197.57/20 = 9.88$ are stored. Suppose that an iteration on a new block Ω_2 of 20 sequences gives $\eta^{(1)}(Na \rightarrow S1\ T6)/20 = 83.38$ and $\eta^{(1)}(Na \rightarrow Q6\ Q6)/20 = 148.49$. Thus, for $\chi = 0.25$, $\tilde{N}^{(1)}(Na \rightarrow S1\ T6)$ and $\tilde{N}^{(1)}(Na \rightarrow Q6\ T6)$ are computed as follows:

$$\begin{aligned}
\tilde{N}^{(1)}(Na \rightarrow S1\ T6) &= \tilde{N}_1(Na \rightarrow S1\ T6) \\
&\quad +0.25 \cdot (\eta^{(1)}(Na \rightarrow S1\ T6)/20 - \tilde{N}_1(Na \rightarrow S1\ T6)) \\
&= 6.63 + 0.25 \cdot (83.38/20 - 6.63) = 6.01 \\
\tilde{N}^{(1)}(Na \rightarrow Q6\ T6) &= \tilde{N}_1(Na \rightarrow Q6\ Q6) \\
&\quad +0.25 \cdot (\eta^{(1)}(Na \rightarrow Q6\ Q6)/20 - \tilde{N}_1(Na \rightarrow Q6\ Q6)) \\
&= 9.88 + 0.25 \cdot (148.49/20 - 9.88) = 9.27
\end{aligned} \tag{4.21}$$

$\theta'(Na \rightarrow S1\ T6)$ is computed as follows:

$$\theta'(Na \rightarrow S1\ T6) = \frac{\tilde{N}^{(1)}(Na \rightarrow S1\ T6)}{\tilde{N}^{(1)}(Na)} = \frac{6.01}{6.01 + 9.27} = 0.393 \tag{4.22}$$

Suppose then that the next iteration on Ω_2 gives $\eta^{(2)}(Na \rightarrow S1\ T6) = 161.51$ and $\eta^{(2)}(Na \rightarrow Q6\ Q6) = 156.99$. Thus, for $\chi = 0.25$, $\tilde{N}^{(1)}(Na \rightarrow S1\ T6)$ and $\tilde{N}^{(1)}(Na \rightarrow Q6\ T6)$ are computed as follows:

$$\begin{aligned}
\tilde{N}^{(1)}(Na \rightarrow S1\ T6) &= \tilde{N}_1(Na \rightarrow S1\ T6) \\
&\quad +0.25 \cdot (\eta^{(2)}(Na \rightarrow S1\ T6)/20 - \tilde{N}_1(Na \rightarrow S1\ T6)) \\
&= 6.63 + 0.25 \cdot (161.51/20 - 6.63) = 6.99
\end{aligned}$$

$$\begin{aligned}\tilde{N}^{(1)}(Na \rightarrow Q6\,T6) &= \tilde{N}_1(Na \rightarrow Q6\,Q6) \\ &\quad +0.25 \cdot (\eta^{(2)}(Na \rightarrow Q6\,Q6)/20 - \tilde{N}_1(Na \rightarrow Q6\,Q6)) \\ &= 9.88 + 0.25 \cdot (156.99/20 - 9.88) = 9.37 \end{aligned} \quad (4.23)$$

$\theta'(Na \rightarrow S1\,T6)$ is computed as follows:

$$\theta'(Na \rightarrow S1\,T6) = \frac{\tilde{N}^{(2)}(Na \rightarrow S1\,T6)}{\tilde{N}^{(2)}(Na)} = \frac{6.99}{6.99 + 9.37} = 0.427 \quad (4.24)$$

This process is repeated until convergence, and the final value $\tilde{N}_2(Na \rightarrow S1\,T6) = \tilde{N}^{(convergence)}(Na \rightarrow S1\,T6)$ is stored for the next block of data.

4.3 Comparison of igEM and oigEM

The main difference between igEM and oigEM lies in the fact that parameter χ must be set in the second one. Parameter χ allows giving more importance either to the new dataset, or the old one, and thereby tuning the algorithm. A brief calculation easily show that setting $\chi(i) = \frac{\chi(i-1)}{1+\chi(i-1)}$ for the i^{th} block of sequences, make the two algorithms identical. In the original on-line EM algorithm, in which the examples are presented one after the other, χ is supposed to decrease, in order to give less importance to the new data. Indeed, if learning has already been performed on a lot of examples, the new one should not have as much importance as the old data. However, in radar ES applications, radar sequences can be obtained in different conditions and environment, and one could therefore have more or less confidence in a new block of sequences. That is why, if one has very good confidence in a new block, he could decide to increase χ.

The main advantage of these two algorithms lies in the fact that local optima may be avoided thanks to the re-initialization of the production probabilities of the SCFG when

new data is added. Suppose that training was previously performed on an original dataset Ω_1 and that a new one, Ω_2 becomes available. Both igEM and oigEM use only the support graphs from Ω_2 to compute η_2, while information on Ω_1 is already stored in η_1. Consider that each dataset is not fully representative of the whole problem, and that their distributions can even be disjoined in the space of features. In this case, as illustrated in Fig 14, the production rule probabilities obtained after training on Ω_1 should not serve as the starting point for the incremental learning process on Ω_2. In this case, it can lead to being trapped in a local minimum of the new cost function associated with $\Omega_1 + \Omega_2$, unless probabilities are re-initialized. Suppose that the plain curve represents the cost function associated with a system trained on block Ω_1, and that the dotted curved represents the cost function associated with a system first trained on block Ω_1 and then incrementally on a new block Ω_2. Point (1) represents the idéal solution of an optimization performed on Ω_1. It appears clearly that if this point is used as a starting point for training on Ω_2 (point (2)), then it will lead to the local optimum at point (3). On the other hand, if the probabilities are randomly re-initialized before training on Ω_2, allowing, for instance, to start at point (4), the optimization would lead to point (5), and a local optimum would be avoided. Other approaches, such as the gradient descent technique of Baldi and Chauvin (1994), do not re-initialize the probabilities prior to learning new training sequences, and were therefore not considered.

The HOLA algorithm uses Ω_2 to update a histogram created using Ω_1. In this case, only one distribution is used during the whole process. Note that this distribution is the only one to be modified when new data is learned, and that it has a fixed size, making HOLA very efficient for incremental learning.

An incremental version of gEM(VS) is straightforward – gEM(VS) would require deriving new routines `Get-Inside-Probs` and `Get-Expectation` (`Get-Inside-Probs-VS` and `Get-Expectation-VS`) in order to compute $\hat{\eta}$ for each sequence of the dataset,. The value $\hat{\eta}$ is the value η corresponding to the

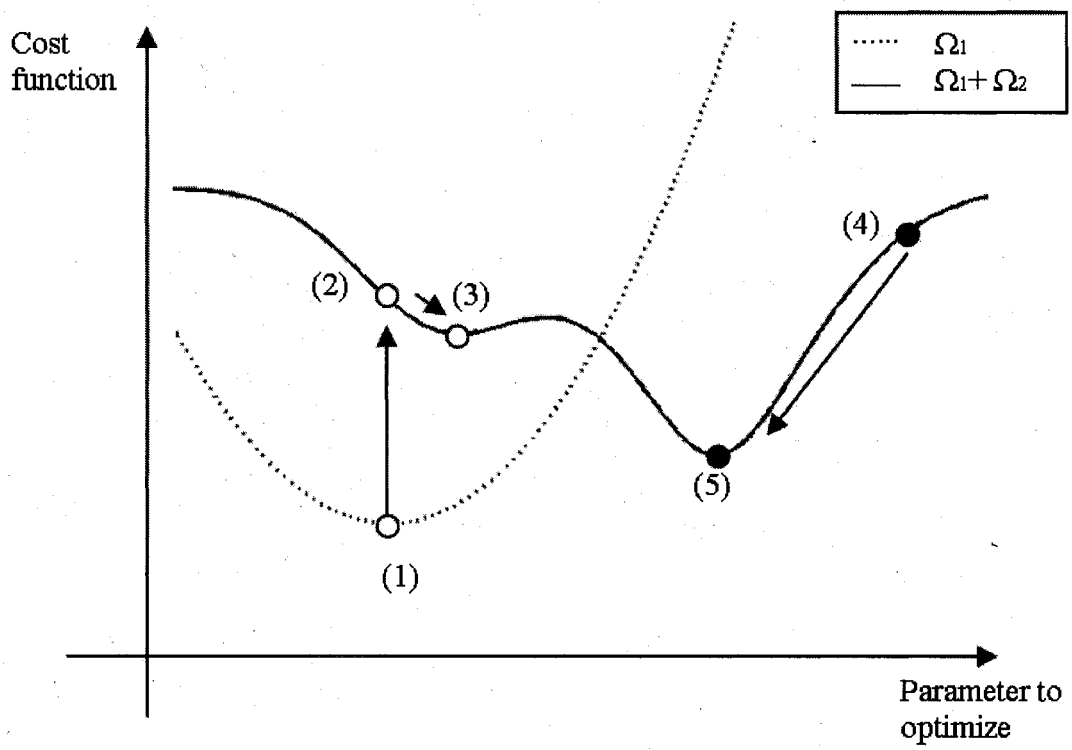

Figure 14 Illustration of the importance of re-initializing probabilities when training on a new block of sequences.

derivation tree of maximum likelihood. Therefore, adapting igEM and oigEM to VS would only involve $\hat{\eta}$ instead of η in Eqs. 3.27 and 4.20 respectively. Finally, one can see that igEM and oigEM can directly be adapted to TS. Indeed, TS computes the different elements of Eq. 3.2, which are also the elements of η.

Table 4.3 displays a summary of the re-estimation formulas of the incremental learning techniques, discussed in this chapter, and their relation with the global re-estimation formula of Eq.3.2. This table also contains definitions of the symbols used in each reestimation formula. The relationship between igEM and oigEM appears clearly in this table.

Table II

Summary of re-estimation formulas of techniques for incremental learning of SCFGs.

Method	Reestimation Formula
General	$\theta'(A \to \lambda) = \frac{\sum_{x \in \Omega} \frac{\sum_{d_x \in \Delta_x} N(A \to \lambda, d_x) P(x, d_x \mid G_s)}{P(x, \Delta_x \mid G_s)}}{\sum_{x \in \Omega} \frac{\sum_{d_x \in \Delta_x} N(A, d_x) P(x, d_x \mid G_s)}{P(x, \Delta_x \mid G_s)}}$ $N(A \to \lambda, d_x)$ = frequency of the rule $A \to \lambda$ in the derivation tree d_x
igEM	$\theta'(A \to \lambda) = \frac{\eta_0(A \to \lambda) + \eta_1(A \to \lambda)}{\sum_\lambda \eta_0(A \to \lambda) + \eta_1(A \to \lambda)}$ $\eta_0(A \to \lambda) = \eta(A \to \lambda)$ computed on the previous dataset and stored $\eta_1(A \to \lambda) = \eta(A \to \lambda)$ computed on the new dataset and updated
oigEM	$\theta'(A \to \lambda) = \frac{\tilde{N}(A \to \lambda)}{\tilde{N}(A)}$ $\tilde{N}(A \to \lambda) = \tilde{N}_0(A \to \lambda) + \chi[\frac{\eta_1(A \to \lambda)}{\lvert\Omega_i\rvert} - \tilde{N}_0(A \to \lambda)]$ $\tilde{N}(A \to \lambda)$ = balanced frequency of the rule $A \to \lambda$
HOLA	$\theta'(A \to \lambda) = \theta(A \to \lambda) \cdot (1 + \chi \cdot (N(A \to \lambda).o - N(A \to \lambda).s)$ $N(A \to \lambda).o$ = frequency of the rule $A \to \lambda$ over the training set $N(A \to \lambda).s$ = frequency of the rule $A \to \lambda$ over the generated set