Online EM for Unsupervised Models

Percy Liang **Dan Klein**
Computer Science Division, EECS Department
University of California at Berkeley
Berkeley, CA 94720
{pliang,klein}@cs.berkeley.edu

Abstract

The (batch) EM algorithm plays an important role in unsupervised induction, but it sometimes suffers from slow convergence. In this paper, we show that online variants (1) provide significant speedups and (2) can even find *better* solutions than those found by batch EM. We support these findings on four unsupervised tasks: part-of-speech tagging, document classification, word segmentation, and word alignment.

1 Introduction

In unsupervised NLP tasks such as tagging, parsing, and alignment, one wishes to induce latent linguistic structures from raw text. Probabilistic modeling has emerged as a dominant paradigm for these problems, and the EM algorithm has been a driving force for learning models in a simple and intuitive manner.

However, on some tasks, EM can converge slowly. For instance, on unsupervised part-of-speech tagging, EM requires over 100 iterations to reach its peak performance on the Wall-Street Journal (Johnson, 2007). The slowness of EM is mainly due to its batch nature: Parameters are updated only once after each pass through the data. When parameter estimates are still rough or if there is high redundancy in the data, computing statistics on the entire dataset just to make one update can be wasteful.

In this paper, we investigate two flavors of online EM—*incremental EM* (Neal and Hinton, 1998) and *stepwise EM* (Sato and Ishii, 2000; Cappé and Moulines, 2009), both of which involve updating parameters after each example or after a mini-batch (subset) of examples. Online algorithms have the potential to speed up learning by making updates more frequently. However, these updates can be seen as noisy approximations to the full batch update, and this noise can in fact impede learning.

This tradeoff between speed and stability is familiar to online algorithms for convex supervised learning problems—e.g., Perceptron, MIRA, stochastic gradient, etc. Unsupervised learning raises two additional issues: (1) Since the EM objective is nonconvex, we often get convergence to different local optima of varying quality; and (2) we evaluate on accuracy metrics which are at best loosely correlated with the EM likelihood objective (Liang and Klein, 2008). We will see that these issues can lead to surprising results.

In Section 4, we present a thorough investigation of online EM, mostly focusing on stepwise EM since it dominates incremental EM. For stepwise EM, we find that choosing a good stepsize and mini-batch size is important but can fortunately be done adequately without supervision. With a proper choice, stepwise EM reaches the same performance as batch EM, but much more quickly. Moreover, it can even surpass the performance of batch EM. Our results are particularly striking on part-of-speech tagging: Batch EM crawls to an accuracy of 57.3% after 100 iterations, whereas stepwise EM shoots up to 65.4% after just two iterations.

2 Tasks, models, and datasets

In this paper, we focus on unsupervised induction via probabilistic modeling. In particular, we define a probabilistic model $p(\mathbf{x}, \mathbf{z}; \theta)$ of the input $\mathbf{x}$ (e.g., a sentence) and hidden output $\mathbf{z}$ (e.g., a parse tree) with parameters θ (e.g., rule probabilities). Given a set of unlabeled examples $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}$, the standard training objective is to maximize the marginal log-likelihood of these examples:

$$\ell(\theta) = \sum_{i=1}^{n} \log p(\mathbf{x}^{(i)}; \theta). \tag{1}$$

A trained model $\hat{\theta}$ is then evaluated on the accuracy of its predictions: $\operatorname{argmax}_{\mathbf{z}} p(\mathbf{z} \mid \mathbf{x}^{(i)}; \hat{\theta})$ against the true output $\mathbf{z}^{(i)}$; the exact evaluation metric depends on the task. What makes unsupervised induction hard at best and ill-defined at worst is that the training objective (1) does not depend on the true outputs at all.

We ran experiments on four tasks described below. Two of these tasks—part-of-speech tagging and document classification—are "clustering" tasks. For these, the output $\mathbf{z}$ consists of labels; for evaluation, we map each predicted label to the true label that maximizes accuracy. The other two tasks—segmentation and alignment—only involve unlabeled combinatorial structures, which can be evaluated directly.

Part-of-speech tagging For each sentence $\mathbf{x} = (x_1, \dots, x_\ell)$, represented as a sequence of words, we wish to predict the corresponding sequence of part-of-speech (POS) tags $\mathbf{z} = (z_1, \dots, z_\ell)$. We used a simple bigram HMM trained on the Wall Street Journal (WSJ) portion of the Penn Treebank (49208 sentences, 45 tags). No tagging dictionary was used. We evaluated using per-position accuracy.

Document classification For each document $\mathbf{x} = (x_1, \dots, x_\ell)$ consisting of ℓ words,[1] we wish to predict the document class $\mathbf{z} \in \{1, \dots, 20\}$. Each document $\mathbf{x}$ is modeled as a bag of words drawn independently given the class $\mathbf{z}$. We used the 20 Newsgroups dataset (18828 documents, 20 classes). We evaluated on class accuracy.

Word segmentation For each sentence $\mathbf{x} = (x_1, \dots, x_\ell)$, represented as a sequence of English phonemes or Chinese characters without spaces separating the words, we would like to predict a segmentation of the sequence into words $\mathbf{z} = (z_1, \dots, z_{|\mathbf{z}|})$, where each segment (word) z_i is a contiguous subsequence of $1, \dots, \ell$. Since the naïve unigram model has a degenerate maximum likelihood solution that makes each sentence a separate word, we incorporate a penalty for longer segments: $p(\mathbf{x}, \mathbf{z}; \theta) \propto \prod_{k=1}^{|\mathbf{z}|} p(\mathbf{x}_{z_k}; \theta) e^{-|z_k|^\beta}$, where $\beta > 1$ determines the strength of the penalty. For English, we used $\beta = 1.6$; Chinese, $\beta = 2.5$. To speed up inference, we restricted the maximum segment length to 10 for English and 5 for Chinese.

We applied this model on the Bernstein-Ratner corpus from the CHILDES database used in Goldwater et al. (2006) (9790 sentences) and the Academia Sinica (AS) corpus from the first SIGHAN Chinese word segmentation bakeoff (we used the first 100K sentences). We evaluated using F_1 on word tokens.

To the best of our knowledge, our penalized unigram model is new and actually beats the more complicated model of Johnson (2008) 83.5% to 78%, which had been the best published result on this task.

Word alignment For each pair of translated sentences $\mathbf{x} = (e_1, \dots, e_{n_e}, f_1, \dots, f_{n_f})$, we wish to predict the word alignments $\mathbf{z} \in \{0, 1\}^{n_e n_f}$. We trained two IBM model 1s using agreement-based learning (Liang et al., 2008). We used the first 30K sentence pairs of the English-French Hansards data from the NAACL 2003 Shared Task, 447+37 of which were hand-aligned (Och and Ney, 2003). We evaluated using the standard alignment error rate (AER).

3 EM algorithms

Given a probabilistic model $p(\mathbf{x}, \mathbf{z}; \theta)$ and unlabeled examples $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}$, recall we would like to maximize the marginal likelihood of the data (1). Let $\phi(\mathbf{x}, \mathbf{z})$ denote a mapping from a fully-labeled example $(\mathbf{x}, \mathbf{z})$ to a vector of sufficient statistics (counts in the case of multinomials) for the model. For example, one component of this vector for HMMs would be the number of times state 7 emits the word "house" in sentence $\mathbf{x}$ with state sequence $\mathbf{z}$. Given a vector of sufficient statistics μ, let $\theta(\mu)$ denote the maximum likelihood estimate. In our case, $\theta(\mu)$ are simply probabilities obtained by normalizing each block of counts. This closed-form

[1]We removed the 50 most common words and words that occurred fewer than 5 times.

```
Batch EM
$\mu \leftarrow$ initialization
for each iteration $t = 1, \dots, T$:
  $\mu' \leftarrow 0$
  for each example $i = 1, \dots, n$:
    $s'_i \leftarrow \sum_{\mathbf{z}} p(\mathbf{z} \mid \mathbf{x}^{(i)}; \theta(\mu))\, \phi(\mathbf{x}^{(i)}, \mathbf{z})$    [inference]
    $\mu' \leftarrow \mu' + s'_i$    [accumulate new]
  $\mu \leftarrow \mu'$    [replace old with new]
```

```
Incremental EM (iEM)
$s_i \leftarrow$ initialization for $i = 1, \dots, n$
$\mu \leftarrow \sum_{i=1}^n s_i$
for each iteration $t = 1, \dots, T$:
  for each example $i = 1, \dots, n$ in random order:
    $s'_i \leftarrow \sum_{\mathbf{z}} p(\mathbf{z} \mid \mathbf{x}^{(i)}; \theta(\mu))\, \phi(\mathbf{x}^{(i)}, \mathbf{z})$    [inference]
    $\mu \leftarrow \mu + s'_i - s_i$; $s_i \leftarrow s'_i$    [replace old with new]
```

```
Stepwise EM (sEM)
$\mu \leftarrow$ initialization; $k = 0$
for each iteration $t = 1, \dots, T$:
  for each example $i = 1, \dots, n$ in random order:
    $s'_i \leftarrow \sum_{\mathbf{z}} p(\mathbf{z} \mid \mathbf{x}^{(i)}; \theta(\mu))\, \phi(\mathbf{x}^{(i)}, \mathbf{z})$    [inference]
    $\mu \leftarrow (1-\eta_k)\mu + \eta_k s'_i$; $k \leftarrow k+1$    [towards new]
```

solution is one of the features that makes EM (both batch and online) attractive.

3.1 Batch EM

In the (batch) EM algorithm, we alternate between the E-step and the M-step. In the E-step, we compute the expected sufficient statistics μ' across all the examples based on the posterior over $\mathbf{z}$ under the current parameters $\theta(\mu)$. In all our models, this step can be done via a dynamic program (for example, forward-backward for POS tagging).

In the M-step, we use these sufficient statistics μ' to re-estimate the parameters. Since the M-step is trivial, we represent it implicitly by $\theta(\cdot)$ in order to concentrate on the computation of the sufficient statistics. This focus will be important for online EM, so writing batch EM in this way accentuates the parallel between batch and online.

3.2 Online EM

To obtain an online EM algorithm, we store a single set of sufficient statistics μ and update it after processing each example. For the i-th example, we compute sufficient statistics s'_i. There are two main variants of online EM algorithms which differ in exactly how the new s'_i is incorporated into μ.

The first is *incremental EM* (iEM) (Neal and Hinton, 1998), in which we not only keep track of μ but also the sufficient statistics $s_1, \dots, s_n$ for each example ($\mu = \sum_{i=1}^n s_i$). When we process example i, we subtract out the old s_i and add the new s'_i.

Sato and Ishii (2000) developed another variant, later generalized by Cappé and Moulines (2009), which we call *stepwise EM* (sEM). In sEM, we interpolate between μ and s'_i based on a stepsize η_k (k is the number of updates made to μ so far).

The two algorithms are motivated in different ways. Recall that the log-likelihood can be lower bounded as follows (Neal and Hinton, 1998):

$$\ell(\theta) \geq \mathcal{L}(q_1, \dots, q_n, \theta) \tag{2}$$
$$\stackrel{\text{def}}{=} \sum_{i=1}^{n} \Big[\sum_{\mathbf{z}} q_i(\mathbf{z} \mid \mathbf{x}^{(i)}) \log p(\mathbf{x}^{(i)}, \mathbf{z}; \theta) + H(q_i) \Big],$$

where $H(q_i)$ is the entropy of the distribution $q_i(\mathbf{z} \mid \mathbf{x}^{(i)})$. Batch EM alternates between optimizing $\mathcal{L}$ with respect to $q_1, \dots, q_n$ in the E-step (represented implicitly via sufficient statistics μ') and with respect to θ in the M-step. Incremental EM alternates between optimizing with respect to a single q_i and θ.

Stepwise EM is motivated from the stochastic approximation literature, where we think of approximating the update μ' in batch EM with a single sample s'_i. Since one sample is a bad approximation, we interpolate between s'_i and the current μ. Thus, sEM can be seen as stochastic gradient in the space of sufficient statistics.

Stepsize reduction power α Stepwise EM leaves open the choice of the stepsize η_k. Standard results from the stochastic approximation literature state that $\sum_{k=0}^{\infty} \eta_k = \infty$ and $\sum_{k=0}^{\infty} \eta_k^2 < \infty$ are sufficient to guarantee convergence to a local optimum. In particular, if we take $\eta_k = (k+2)^{-\alpha}$, then any $0.5 < \alpha \leq 1$ is valid. The smaller the α, the larger the updates, and the more quickly we forget (decay) our old sufficient statistics. This can lead to swift progress but also generates instability.

Mini-batch size m We can add some stability to sEM by updating on multiple examples at once instead of just one. In particular, partition the n examples into mini-batches of size m and run sEM, treating each mini-batch as a single example. Formally, for each $i = 0, m, 2m, 3m, \dots$, first compute the sufficient statistics $s'_{i+1}, \dots, s'_{i+m}$ on $\mathbf{x}^{(i+1)}, \dots, \mathbf{x}^{(i+m)}$ and then update μ using $s'_{i+1} + \dots + s'_{i+m}$. The larger the m, the less frequent the updates, but the more stable they are. In this way, mini-batches interpolate between a pure online ($m = 1$) and a pure batch ($m = n$) algorithm.[2]

Fast implementation Due to sparsity in NLP, the sufficient statistics of an example s'_i are nonzero for a small fraction of its components. For iEM, the time required to update μ with s'_i depends only on the number of nonzero components of s'_i. However, the sEM update is $\mu \leftarrow (1-\eta_k)\mu + \eta_k s'_i$, and a naïve implementation would take time proportional to the total number of components. The key to a more efficient solution is to note that $\theta(\mu)$ is invariant to scaling of μ. Therefore, we can store $S = \frac{\mu}{\prod_{j<k}(1-\eta_j)}$ instead of μ and make the following *sparse* update: $S \leftarrow S + \frac{\eta_k}{\prod_{j \le k}(1-\eta_j)} s'_i$, taking comfort in the fact that $\theta(\mu) = \theta(S)$.

For both iEM and sEM, we also need to efficiently compute $\theta(\mu)$. We can do this by maintaining the normalizer for each multinomial block (sum of the components in the block). This extra maintenance only doubles the number of updates we have to make but allows us to fetch any component of $\theta(\mu)$ in constant time by dividing out the normalizer.

3.3 Incremental versus stepwise EM

Incremental EM increases $\mathcal{L}$ monotonically after each update by virtue of doing coordinate-wise ascent and thus is guaranteed to converge to a local optimum of both $\mathcal{L}$ and ℓ (Neal and Hinton, 1998). However, ℓ is not guaranteed to increase after each update. Stepwise EM might not increase either $\mathcal{L}$ or ℓ after each update, but it is guaranteed to converge to a local optimum of ℓ given suitable conditions on the stepsize discussed earlier.

Incremental and stepwise EM actually coincide under the following setting (Cappé and Moulines, 2009): If we set $(\alpha, m) = (1, 1)$ for sEM and initialize all $s_i = 0$ for iEM, then both algorithms make the same updates on the first pass through the data. They diverge thereafter as iEM subtracts out old s_is, while sEM does not even remember them.

One weakness of iEM is that its memory requirements grow linearly with the number of examples due to storing $s_1, \dots, s_n$. For large datasets, these s_is might not even fit in memory, and resorting to physical disk would be very slow. In contrast, the memory usage of sEM does not depend on n.

The relationship between iEM and sEM (with $m = 1$) is analogous to the one between exponentiated gradient (Collins et al., 2008) and stochastic gradient for supervised learning of log-linear models. The former maintains the sufficient statistics of each example and subtracts out old ones whereas the latter does not. In the supervised case, the added stability of exponentiated gradient tends to yield better performance. For the unsupervised case, we will see empirically that remembering the old sufficient statistics offers no benefit, and much better performance can be obtained by properly setting (α, m) for sEM (Section 4).

4 Experiments

We now present our empirical results for batch EM and online EM (iEM and sEM) on the four tasks described in Section 2: part-of-speech tagging, document classification, word segmentation (English and Chinese), and word alignment.

We used the following protocol for all experiments: We initialized the parameters to a neutral setting plus noise to break symmetries.[3] Training was performed for 20 iterations.[4] No parameter smoothing was used. All runs used a fixed random seed for initializing the parameters and permuting the examples at the beginning of each iteration. We report two performance metrics: log-likelihood normalized by the number of examples and the task-specific accuracy metric (see Section 2). All numbers are taken from the final iteration.

[2]Note that running sEM with $m = n$ is similar but not equivalent to batch EM since old sufficient statistics are still interpolated rather than replaced.

[3]Specifically, for each block of multinomial probabilities $\theta_1, \dots, \theta_K$, we set $\theta_k \propto \exp\{10^{-3}(1 + a_k)\}$, where $a_k \sim U[0, 1]$. Exception: for batch EM on POS tagging, we used 1 instead of 10^{-3}; more noise worked better.

[4]Exception: for batch EM on POS tagging, 100 iterations was needed to get satisfactory performance.

Stepwise EM (sEM) requires setting two optimization parameters: the stepsize reduction power α and the mini-batch size m (see Section 3.2). As Section 4.3 will show, these two parameters can have a large impact on performance. As a default rule of thumb, we chose $(\alpha, m) \in \{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \times \{1, 3, 10, 30, 100, 300, 1\text{K}, 3\text{K}, 10\text{K}\}$ to maximize log-likelihood; let sEM$_\ell$ denote stepwise EM with this setting. Note that this setting requires no labeled data. We will also consider fixing $(\alpha, m) = (1, 1)$ (sEM$_i$) and choosing (α, m) to maximize accuracy (sEM$_a$).

In the results to follow, we first demonstrate that online EM is faster (Section 4.1) and sometimes leads to higher accuracies (Section 4.2). Next, we explore the effect of the optimization parameters (α, m) (Section 4.3), briefly revisiting the connection between incremental and stepwise EM. Finally, we show the stability of our results under different random seeds (Section 4.4).

4.1 Speed

One of the principal motivations for online EM is speed, and indeed we found this motivation to be empirically well-justified. Figure 1 shows that, across all five datasets, sEM$_\ell$ converges to a solution with at least comparable log-likelihood and accuracy with respect to batch EM, but sEM$_\ell$ does it anywhere from about 2 (word alignment) to 10 (POS tagging) times faster. This supports our intuition that more frequent updates lead to faster convergence. At the same time, note that the other two online EM variants in Figure 1 (iEM and sEM$_i$) are prone to catastrophic failure. See Section 4.3 for further discussion on this issue.

4.2 Performance

It is fortunate but perhaps not surprising that stepwise EM is faster than batch EM. But Figure 1 also shows that, somewhat surprisingly, sEM$_\ell$ can actually converge to a solution with higher accuracy, in particular on POS tagging and document classification. To further explore the accuracy-increasing potential of sEM, consider choosing (α, m) to maximize accuracy (sEM$_a$). Unlike sEM$_\ell$, sEM$_a$ does require labeled data. In practice, (α, m) can be tuned on a small labeled set along with any model hyperparameters.

	EM	sEM$_\ell$	sEM$_a$	α_ℓ	m_ℓ	α_a	m_a
POS	57.3	59.6	**66.7**	0.7	3	0.5	3
DOC	39.1	47.8	**49.9**	0.8	1K	0.5	3K
SEG(en)	80.5	80.7	**83.5**	0.7	1K	1.0	100
SEG(ch)	**78.2**	77.2	78.1	0.6	10K	1.0	10K
ALIGN	78.8	78.9	**78.9**	0.7	10K	0.7	10K

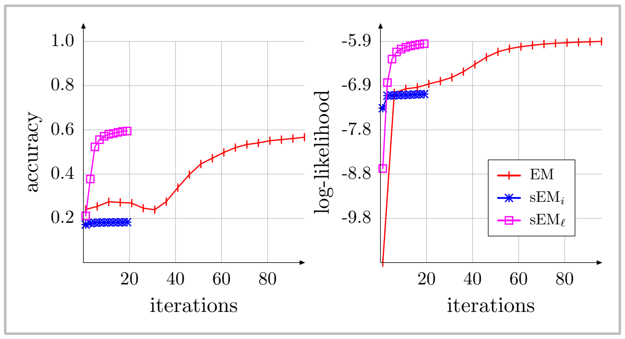
Table 1: Accuracy of batch EM and stepwise EM, where the optimization parameters (α, m) are tuned to either maximize log-likelihood (sEM$_\ell$) or accuracy (sEM$_a$). With an appropriate setting of (α, m), stepwise EM outperforms batch EM significantly on POS tagging and document classification.

Table 1 shows that sEM$_a$ improves the accuracy compared to batch EM even more than sEM$_\ell$. The result for POS is most vivid: After one iteration of batch EM, the accuracy is only at 24.0% whereas sEM$_a$ is already at 54.5%, and after two iterations, at 65.4%. Not only is this orders of magnitude faster than batch EM, batch EM only reaches 57.3% after 100 iterations.

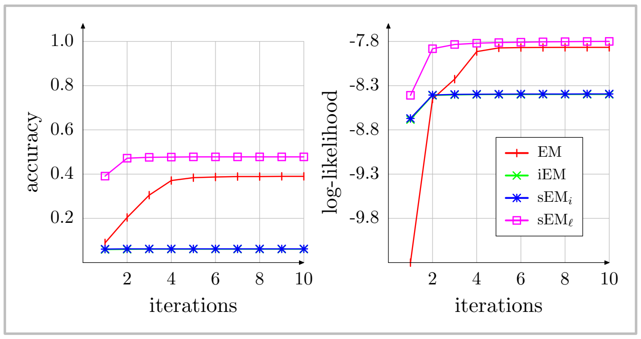
We get a similarly striking result for document classification, but the results for word segmentation and word alignment are more modest. A full understanding of this phenomenon is left as an open problem, but we will comment on one difference between the tasks where sEM improves accuracy and the tasks where it doesn't. The former are "clustering" tasks (POS tagging and document classification), while the latter are "structural" tasks (word segmentation and word alignment). Learning of clustering models centers around probabilities over words given a latent cluster label, whereas in structural models, there are no cluster labels, and it is the combinatorial structure (the segmentations and alignments) that drives the learning.

Likelihood versus accuracy From Figure 1, we see that stepwise EM (sEM$_\ell$) can outperform batch EM in both likelihood and accuracy. This suggests that stepwise EM is better at avoiding local minima, perhaps leveraging its stochasticity to its advantage.

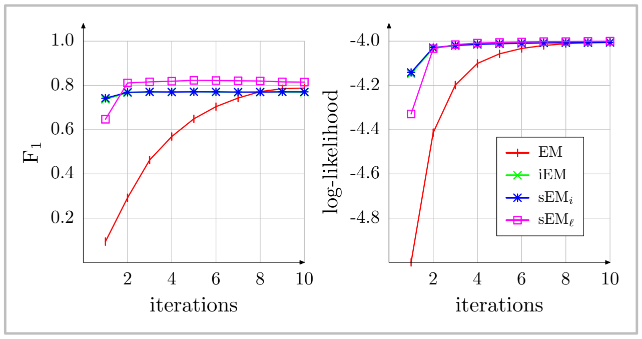
However, on POS tagging, tuning sEM to maximize accuracy (sEM$_a$) results in a slower increase in likelihood: compare sEM$_a$ in Figure 2 with sEM$_\ell$ in Figure 1(a). This shouldn't surprise us too much given that likelihood and accuracy are only loosely



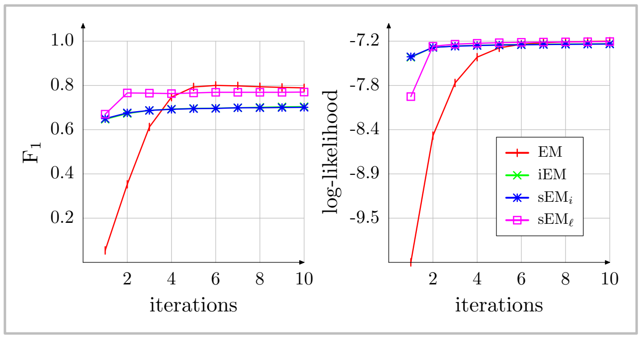
(a) POS tagging



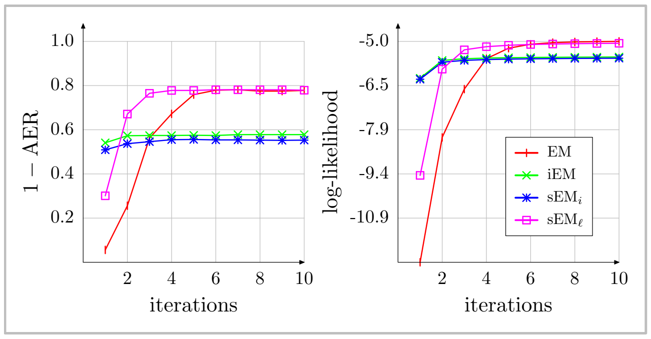
(b) Document classification



(c) Word segmentation (English)



(d) Word segmentation (Chinese)



(e) Word alignment

	accuracy		log-likelihood	
	EM	sEM$_\ell$	EM	sEM$_\ell$
POS	57.3	59.6	-6.03	-6.08
DOC	39.1	47.8	-7.96	-7.88
SEG(en)	80.5	80.7	-4.11	-4.11
SEG(ch)	78.2	77.2	-7.27	-7.28
ALIGN	78.8	78.9	-5.05	-5.12

(f) Results after convergence

Figure 1: Accuracy and log-likelihood plots for batch EM, incremental EM, and stepwise EM across all five datasets. sEM$_\ell$ outperforms batch EM in terms of convergence speed and even accuracy and likelihood; iEM and sEM$_i$ fail in some cases. We did not run iEM on POS tagging due to memory limitations; we expect the performance would be similar to sEM$_i$, which is not very encouraging (Section 4.3).

correlated (Liang and Klein, 2008). But it does suggest that stepwise EM is injecting a bias that favors accuracy over likelihood—a bias not at all reflected in the training objective.

We can create a hybrid (sEM$_a$+EM) that combines the strengths of both sEM$_a$ and EM: First run sEM$_a$ for 5 iterations, which quickly takes us to a part of the parameter space yielding good accuracies; then run EM, which quickly improves the likelihood. Fortunately, accuracy does not degrade as likelihood increases (Figure 2).

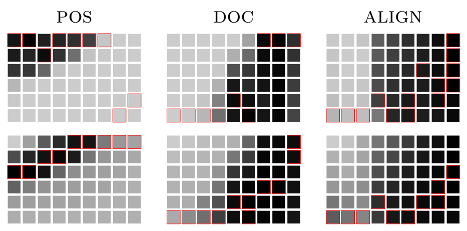
4.3 Varying the optimization parameters

Recall that stepwise EM requires setting two optimization parameters: the stepsize reduction power α and the mini-batch size m. We now explore the effect of (α, m) on likelihood and accuracy.

As mentioned in Section 3.2, larger mini-batches (increasing m) stabilize parameter updates, while larger stepsizes (decreasing α) provide swifter

DOC accuracy

$\alpha \backslash m$	1	3	10	30	100	300	1K	3K	10K
0.5	5.4	5.4	5.5	5.6	6.0	25.7	**48.8**	$\boxed{\mathbf{49.9}}$	**44.6**
0.6	5.4	5.4	5.6	5.6	22.3	36.1	48.7	49.3	44.2
0.7	5.5	5.5	5.6	11.1	39.9	43.3	48.1	49.0	43.5
0.8	5.6	5.6	6.0	21.7	47.3	45.0	47.8	49.5	42.8
0.9	5.8	6.0	13.4	32.4	**48.7**	48.4	46.4	49.4	42.4
1.0	**6.2**	**11.8**	**19.6**	**35.2**	47.6	**49.5**	47.5	49.3	41.7



DOC log-likelihood

$\alpha \backslash m$	1	3	10	30	100	300	1K	3K	10K
0.5	-8.875	-8.71	-8.61	-8.555	-8.505	-8.172	-7.92	-7.906	**-7.916**
0.6	-8.604	-8.575	-8.54	-8.524	-8.235	-8.041	-7.898	-7.901	**-7.916**
0.7	-8.541	-8.533	-8.531	-8.354	-8.023	-7.943	-7.886	-7.896	-7.918
0.8	-8.519	-8.506	-8.493	-8.228	-7.933	-7.896	$\boxed{\mathbf{-7.883}}$	**-7.89**	-7.922
0.9	-8.505	-8.486	-8.283	-8.106	**-7.91**	**-7.889**	-7.889	-7.891	-7.927
1.0	**-8.471**	**-8.319**	**-8.204**	**-8.052**	-7.919	**-7.889**	-7.892	-7.896	-7.937

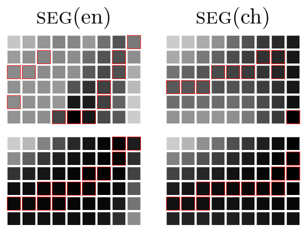


Figure 3: Effect of optimization parameters (stepsize reduction power α and mini-batch size m) on accuracy and likelihood. Numerical results are shown for document classification. In the interest of space, the results for each task are compressed into two gray scale images, one for accuracy (top) and one for log-likelihood (bottom), where darker shades represent larger values. Bold (red) numbers denote the best α for a given m.

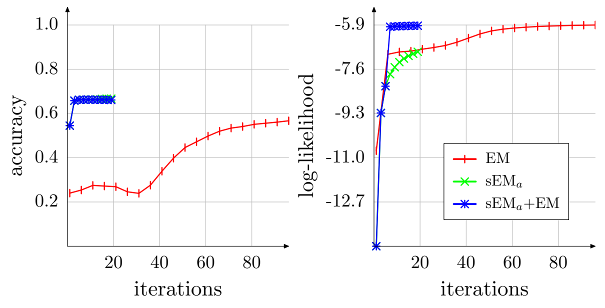


Figure 2: sEM$_a$ quickly obtains higher accuracy than batch EM but suffers from a slower increase in likelihood. The hybrid sEM$_a$+EM (5 iterations of EM$_a$ followed by batch EM) increases both accuracy and likelihood sharply.

progress. Remember that since we are dealing with a nonconvex objective, the choice of stepsize not only influences how fast we converge, but also the quality of the solution that we converge to.

Figure 3 shows the interaction between α and m in terms of likelihood and accuracy. In general, the best (α, m) depends on the task and dataset. For example, for document classification, larger m is critical for good performance; for POS tagging, it is better to use smaller values of α and m.

Fortunately, there is a range of permissible settings (corresponding to the dark regions in Figure 3) that lead to reasonable performance. Furthermore, the settings that perform well on likelihood generally correspond to ones that perform well on accuracy, which justifies using sEM$_\ell$.

A final observation is that as we use larger mini-batches (larger m), decreasing the stepsize more gradually (smaller α) leads to better performance. Intuitively, updates become more reliable with larger m, so we can afford to trust them more and incorporate them more aggressively.

Stepwise versus incremental EM In Section 3.2, we mentioned that incremental EM can be made equivalent to stepwise EM with $\alpha = 1$ and $m = 1$ (sEM$_i$). Figure 1 provides the empirical support: iEM and sEM$_i$ have very similar training curves. Therefore, keeping around the old sufficient statistics does not provide any advantage and still requires a substantial storage cost. As mentioned before, setting (α, m) properly is crucial. While we could simulate mini-batches with iEM by updating multiple coordinates simultaneously, iEM is not capable of exploiting the behavior of $\alpha < 1$.

4.4 Varying the random seed

All our results thus far represent single runs with a fixed random seed. We now investigate the impact of randomness on our results. Recall that we use randomness for two purposes: (1) initializing the parameters (affects both batch EM and online EM),

	accuracy		log-likelihood	
	EM	sEM$_\ell$	EM	sEM$_\ell$
POS	56.2 ± 1.36	58.8 $\pm 0.73, 1.41$	-6.01	-6.09
DOC	41.2 ± 1.97	51.4 $\pm 0.97, 2.82$	-7.93	-7.88
SEG(en)	80.5 ± 0.0	81.0 $\pm 0.0, 0.42$	-4.1	-4.1
SEG(ch)	78.2 ± 0.0	77.2 $\pm 0.0, 0.04$	-7.26	-7.27
ALIGN	79.0 ± 0.14	78.8 $\pm 0.14, 0.25$	-5.04	-5.11

Table 2: Mean and standard deviation over different random seeds. For EM and sEM, the first number after $\pm$ is the standard deviation due to different initializations of the parameters. For sEM, the second number is the standard deviation due to different permutations of the examples. Standard deviation for log-likelihoods are all < 0.01 and therefore left out due to lack of space.

and (2) permuting the examples at the beginning of each iteration (affects only online EM).

To separate these two purposes, we used two different seeds, $S_i \in \{1, 2, 3, 4, 5\}$ and $S_p \in \{1, 2, 3, 4, 5\}$ for initializing and permuting, respectively. Let X be a random variable denoting either log-likelihood or accuracy. We define the variance due to initialization as $\text{var}(\mathbb{E}(X \mid S_i))$ ($\mathbb{E}$ averages over S_p for each fixed S_i) and the variance due to permutation as $\mathbb{E}(\text{var}(X \mid S_i))$ ($\mathbb{E}$ averages over S_i). These two variances provide an additive decomposition of the total variance: $\text{var}(X) = \text{var}(\mathbb{E}(X \mid S_i)) + \mathbb{E}(\text{var}(X \mid S_i))$.

Table 2 summarizes the results across the 5 trials for EM and 25 for sEM$_\ell$. Since we used a very small amount of noise to initialize the parameters, the variance due to initialization is systematically smaller than the variance due to permutation. sEM$_\ell$ is less sensitive to initialization than EM, but additional variance is created by randomly permuting the examples. Overall, the accuracy of sEM$_\ell$ is more variable than that of EM, but not by a large amount.

5 Discussion and related work

As datasets increase in size, the demand for online algorithms has grown in recent years. One sees this clear trend in the supervised NLP literature—examples include the Perceptron algorithm for tagging (Collins, 2002), MIRA for dependency parsing (McDonald et al., 2005), exponentiated gradient algorithms (Collins et al., 2008), stochastic gradient for constituency parsing (Finkel et al., 2008), just to name a few. Empirically, online methods are often faster by an order of magnitude (Collins et al., 2008), and it has been argued on theoretical grounds that the fast, approximate nature of online methods is a good fit given that we are interested in test performance, not the training objective (Bottou and Bousquet, 2008; Shalev-Shwartz and Srebro, 2008).

However, in the unsupervised NLP literature, online methods are rarely seen,[5] and when they are, incremental EM is the dominant variant (Gildea and Hofmann, 1999; Kuo et al., 2008). Indeed, as we have shown, applying online EM does require some care, and some variants (including incremental EM) can fail catastrophically in face of local optima. Stepwise EM provides finer control via its optimization parameters and has proven quite successful.

One family of methods that resembles incremental EM includes collapsed samplers for Bayesian models—for example, Goldwater et al. (2006) and Goldwater and Griffiths (2007). These samplers keep track of a sample of the latent variables for each example, akin to the sufficient statistics that we store in incremental EM. In contrast, stepwise EM does not require this storage and operates more in the spirit of a truly online algorithm.

Besides speed, online algorithms are of interest for two additional reasons. First, in some applications, we receive examples sequentially and would like to estimate a model in real-time, e.g., in the clustering of news articles. Second, since humans learn sequentially, studying online EM might suggest new connections to cognitive mechanisms.

6 Conclusion

We have explored online EM on four tasks and demonstrated how to use stepwise EM to overcome the dangers of stochasticity and reap the benefits of frequent updates and fast learning. We also discovered that stepwise EM can actually improve accuracy, a phenomenon worthy of further investigation. This paper makes some progress on elucidating the properties of online EM. With this increased understanding, online EM, like its batch cousin, could become a mainstay for unsupervised learning.

[5]Other types of learning methods have been employed successfully, for example, Venkataraman (2001) and Seginer (2007).

References

L. Bottou and O. Bousquet. 2008. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems (NIPS)*.

O. Cappé and E. Moulines. 2009. Online expectation-maximization algorithm for latent data models. *Journal of the Royal Statistics Society: Series B (Statistical Methodology)*, 71.

M. Collins, A. Globerson, T. Koo, X. Carreras, and P. Bartlett. 2008. Exponentiated gradient algorithms for conditional random fields and max-margin Markov networks. *Journal of Machine Learning Research*, 9.

M. Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with Perceptron algorithms. In *Empirical Methods in Natural Language Processing (EMNLP)*.

J. R. Finkel, A. Kleeman, and C. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Human Language Technology and Association for Computational Linguistics (HLT/ACL)*.

D. Gildea and T. Hofmann. 1999. Topic-based language models using EM. In *Eurospeech*.

S. Goldwater and T. Griffiths. 2007. A fully Bayesian approach to unsupervised part-of-speech tagging. In *Association for Computational Linguistics (ACL)*.

S. Goldwater, T. Griffiths, and M. Johnson. 2006. Contextual dependencies in unsupervised word segmentation. In *International Conference on Computational Linguistics and Association for Computational Linguistics (COLING/ACL)*.

M. Johnson. 2007. Why doesn't EM find good HMM POS-taggers? In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP/CoNLL)*.

M. Johnson. 2008. Using adaptor grammars to identify synergies in the unsupervised acquisition of linguistic structure. In *Human Language Technology and Association for Computational Linguistics (HLT/ACL)*, pages 398–406.

J. Kuo, H. Li, and C. Lin. 2008. Mining transliterations from web query results: An incremental approach. In *Sixth SIGHAN Workshop on Chinese Language Processing*.

P. Liang and D. Klein. 2008. Analyzing the errors of unsupervised learning. In *Human Language Technology and Association for Computational Linguistics (HLT/ACL)*.

P. Liang, D. Klein, and M. I. Jordan. 2008. Agreement-based learning. In *Advances in Neural Information Processing Systems (NIPS)*.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *Association for Computational Linguistics (ACL)*.

R. Neal and G. Hinton. 1998. A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*.

F. J. Och and H. Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29:19–51.

M. Sato and S. Ishii. 2000. On-line EM algorithm for the normalized Gaussian network. *Neural Computation*, 12:407–432.

Y. Seginer. 2007. Fast unsupervised incremental parsing. In *Association for Computational Linguistics (ACL)*.

S. Shalev-Shwartz and N. Srebro. 2008. SVM optimization: Inverse dependence on training set size. In *International Conference on Machine Learning (ICML)*.

A. Venkataraman. 2001. A statistical model for word discovery in transcribed speech. *Computational Linguistics*, 27:351–372.