Notes: If you plan to read a print-out of this draft, please use a color-printer.

1 Introduction

The goal of this project is to implement existing stochastic optimization methods in the context of conditional random field (CRF) autoencoders [1]. CRF autoencoders are a class of probabilistic models which was designed to address unsupervised and semi-supervised problems in natural language processing. However, for concreteness, we will focus on a particular instantiation of CRF autoencoders for the classic problem of bitext word alignment [4]. In this milestone, we experimented with several variations of stochastic gradient descent (see § 3.1 for details). By the end of the semester, we plan to also experiment with stochastic expectation maximization (see § 3.1.1 for details).

2 Problem description

Statistical (in contrast to “rule-based”) machine translation systems (e.g., http://www.google.com/translate, http://www.bing.com/translate) use pairs of sentences (in a source language such as Spanish) and their translations (in a target language such as English) for training. A crucial step in building such translation systems is to identify which words in a given sentence translate into which words in a given translation [8, 6], namely the bitext word alignment problem, illustrated via an example in Figure 1.

Model. Given an observed sentence pair (s, t), we model the alignment variables (a) and a reconstruction of the target sentence (\( \hat{t} \)) as follows:

\[
p(a, \hat{t} \mid s, t) = p_\lambda(a \mid s, t)p_0(\hat{t} \mid s, a) = \frac{\exp \lambda^T \sum_{i=1}^{n_t} f(a_i, a_{i-1}, s, t)}{\sum_{a'} \exp \lambda^T \sum_{i=1}^{n_t} f(a'_i, a'_{i-1}, s, t)} \times \prod_{i=1}^{n_t} g_{l_i} | a_i
\]

\[1\]

As is typical in word alignment models, we assume that each word in the target sentence aligns to zero (when \( a_i = 0 \)) or one (when \( a_i > 0 \)) words in the source sentence. Since naturally occurring translations often include many-to-many word alignments, the standard practice is to train one model for each direction (e.g., from Spanish to English, and from English to Spanish) then combine the predicted alignments. Posterior regularization [7] provides a more disciplined approach but this is beyond the scope of this proposal.
where $\lambda \in \mathbb{R}^{n \times a}$ is a vector of feature weights and $f(a_i, a_{i-1}, s, t)$ is a vector of feature functions (also in $\mathbb{R}^{n \times a}$), and $\theta_{|s}$ are parameters of a multinomial distribution (over possible target words given the source word $(s)$) with $\sum_{t \in \mathcal{V}_t} \theta_{t|s} = 1$, $0 \leq \theta_{t|s} \leq 1, \forall s \in \mathcal{V}_s$.

Data. It is relatively easy to collect a training set $\mathcal{T} = \{(s, t)\}$ of sentences in the source language and corresponding translations in the target language. However, it is difficult and time-consuming for humans to identify the word alignments associated with those translations.

In our experiments, we used a training set of Finnish-English sentence pairs. In order to speed up our experiments, we filtered out long sentences. After filtering, our training set consists of 134,296 sentence pairs such that $|s| \leq 5, |t| \leq 5$. In the final report, in order to make sure we are not overfitting to this dataset, we plan to use a held-out development set and a separate test set. We also plan to test the best method we find on a significantly larger dataset and report end-to-end translation performance.

Objective. We optimize the parameters of the CRF autoencoder model in Eq. (1) by maximizing the conditional log-likelihood of generating the correct reconstruction of target sentences ($\hat{t}$), given a pair of source and target sentences $(s, t)$, marginalizing out the word alignment variables $(a)$, as follows:

$$\text{maximize}_{\lambda, \theta} \sum_{(s, t, \hat{t})} \log \sum_a p_{\lambda, \theta}(a | s, t) \quad \text{subject to } \sum_{t \in \mathcal{V}_t} \theta_{t|s} = 1, 0 \leq \theta_{t|s} \leq 1, \forall s \in \mathcal{V}_s$$

(2)

3 Optimization

In this section, we discuss the convexity of this problem (or lack thereof), and quickly review how it was optimized in previous work. We then discuss using stochastic optimization methods to speed up training on large datasets.

Convexity analysis. For each $s \in \mathcal{V}_s$, the constraints on $\theta_{t|s}$, i.e., $\sum_{t \in \mathcal{V}_t} \theta_{t|s} = 1, 0 \leq \theta_{t|s} \leq 1$ describe a probability simplex (a special case of polyhedra) and therefore constitutes a feasible set. Taking the constraints for all $s \in \mathcal{V}_s$ into account, the feasible set is the intersection of convex sets, and therefore is also convex.

Given the convex feasible set, in order to show that Eq. (2) is a convex optimization problem, it suffices to show that $h_{s, t}(\lambda, \theta) = -\log \sum_a p(a | s, t)$ is convex. We can rewrite this as:

$$h_{s, t}(\lambda, \theta) = \log \sum_a e^{\lambda^T F(a, s, t)} - \log \sum_a e^{\lambda^T F(a, s, t)} \times \prod_{i=1}^{|t|} \theta_{t_i|s_{a_i}}$$

(3)

The first term is convex in $(\lambda, \theta)$, since it is a log-sum-exp function (with an affine transformation of $\lambda$). However, the second term is non-convex in $(\lambda, \theta)$ since multiplication does not preserve convexity. We note that, even if we hold either set of parameters constant (i.e., $h_{s, t}(\lambda, \theta = \hat{\theta})$ or $h_{s, t}(\lambda = \hat{\lambda}, \theta)$), the function $h$ is still non-convex in general.

We identified one exception which makes $h_{s, t}$ convex in $\theta$. When $\lambda = 0$, $p_{\lambda}(a | s, t) = \frac{1}{|t|^{|s|} |s|^{|t|}}$; i.e., the prior (and discrete) distribution over possible alignments is uniform. The second term of $h$ reduces to $-\log \sum_a \prod_{i=1}^{|t|} \theta_{t_i|s_{a_i}}$, which can be re-written as $-\sum_{i=1}^{|t|} \log \sum_{a_i=0}^{|s|} \theta_{t_i|s_{a_i}}$. This is a non-positive combination of concave functions and is therefore convex.

---

2 A relatively small number of translations is often manually aligned by bilingual speakers for the purpose of intrinsic evaluation of word alignment predictions. It is also common to evaluate word alignment predictions by using them to train an end-to-end translation system and evaluate the output translations using BLEU [11].

3 It is worth noting that other unsupervised bitext word alignment models with a convex objective, such as the IBM Model 1 [4], assume a uniform prior distribution over possible alignments, and are never used in practice since they result in poor predictions of word alignments.
Although this optimization problem is non-convex, locally-optimal solutions have been found to be useful in practice, provided that we start with a good initialization for model parameters. To that end, Ammar et al. \cite{1} used block-coordinate descent, iteratively alternating between the two optimization problems in Eq. \ref{eq:4} (to optimize $\lambda$) and Eq. \ref{eq:10} (to optimize $\theta$), using L-BFGS \cite{10} and batch expectation-maximization (EM), respectively.

3.1 Optimizing w.r.t. $\lambda$ using L-BFGS vs. stochastic gradient descent

In this section, we are only concerned about the following problem:

$$\min_{\lambda} \ell(\lambda) = \sum_{(s, t, \hat{t})} \log \sum_a p(a, \hat{t} | s, t) \quad (4)$$

The sufficient statistics needed for one iteration of using L-BFGS for this objective require expensive computations and abundant memory since the training sets for word alignments tend to be large. For example, it takes our efficient C++ implementation approximately $4$ minutes to compute the objective value and the gradient with respect to $\lambda$ with a training set of $134296$ short Finnish-English sentence pairs. In other words, it takes $4$ minutes for L-BFGS to propose the next update for $\lambda$.

Since we use L-BFGS to solve for the optimal $\lambda$ in Eq. \ref{eq:4} inside the outer loop of block-coordinate descent, we cannot afford to spend too much time optimizing $\lambda$.

Instead, we proposed to use stochastic gradient descent (SGD) and update $\lambda$ according to an approximation of the gradient based on a few sentence pairs. Our intuition is that one epoch (i.e., full pass over the training set) of SGD constitutes many updates of $\lambda$, and incurs the same runtime cost as one update of L-BFGS. Since we will be optimizing $\lambda$ again in the following block-coordinate descent iteration, we are willing to sacrifice the precision of our solution of $\lambda$ in each block-coordinate descent iteration.

To simplify the exposition, we rewrite the objective in Eq. \ref{eq:4} as

$$\min_{\lambda} \ell(\lambda) = \sum_{i=1}^{N} \log p(\hat{t}_i | s_i, t_i) = \sum_{i=1}^{N} f_i(\lambda); \text{ where } N \text{ is the number of sentence pairs, and } (s_i, t_i, \hat{t}_i) \text{ refer to the } i-th \text{ source sentence, target sentence, and target sentence construction, respectively. The } t-th \text{ SGD update takes the form:}$$

$$\lambda^{(t)} = \lambda^{(t-1)} - \gamma_t \nabla \lambda f_i(\lambda^{(t-1)}) \quad \text{ where } t = 1, \ldots, \infty; i \in \{1, \ldots, N\} \quad (5)$$

3.1.1 Learning rate

In this section, we explore several variations of the learning rate $\gamma_t$ (also known as step size). We quantify the progress each variation makes and compare it to the batch L-BFGS baseline. We quantify progress by reporting the attained value of the objective function (to be minimized) after each epoch. For practical purposes, we are primarily interested in the progress made by each optimization method for one or two epochs, but we plot the objective values after $k = 1, \ldots, 32$ epochs to also see how each method converges. We emphasize that all optimization methods are initialized with the same value of $\lambda^0 = 0$.

Fixed learning rate. Here, the learning rate is constant throughout the experiment. We experimented with two values $\gamma_t = 0.03$ and $\gamma_t = 0.01$, and compared the objective values across epochs with L-BFGS. As shown in Fig. \ref{fig:2} (left), the progress made by SGD far exceeds that of L-BFGS for the first few epochs. While it would be interesting to study the effect of changing the fixed value more carefully, our preliminary results confirm the intuition that when solving for an approximate solution with large data sets, a simple stochastic optimization method is preferred to advanced batch optimization methods such as L-BFGS. That said, if we can afford running many training epochs, L-BFGS eventually outperforms SGD.

\footnote{Industrial machine translation systems are often trained with millions of longer sentence pairs.}

\footnote{This is a much stronger baseline than, say, gradient descent.}

3
Diminishing learning rate. Here, we use variations of the learning rate recommended by Bottou [3]. In particular, we use the following learning rates:

\[ \gamma_t = \gamma_0 (1 + \gamma_0 \eta t)^{-1} \]  
\[ \gamma_t = \gamma_{t-1} (1 + \gamma_{t-1} \eta t)^{-1} \]  
\[ \gamma_t = \gamma_{t-1} (1 + \eta)^{-1} \]

where \( \gamma_0 \) is the initial learning rate, and \( \eta > 0 \) is a decay hyper-parameter (larger values of \( \eta \) result in faster decay). In both Eq. 6 and Eq. 7, the decay rate increases with larger values of \( t \). Eq. 8, on the other hand, uses a constant decay rate.

We compare each variant with L-BFGS in Fig. 2 (right). In the first few epochs, Eq. 6 with \( \eta = 0.001 \) performs best (pink), closely followed by Eq. 6 with \( \eta = 0.025 \) (red). Eq. 7 with \( \eta = 0.001 \) follows with a large margin (green), and then Eq. 8 with \( \eta = 0.025 \) (blue).

Epoch-fixed learning rate. Here, we update the learning rate at \( t = N, t = 2N, \ldots \). The intuition is that, in §3.1.1 when \( N \) is large, the training examples used in the last few iterations in each epoch will tend to have significantly smaller update values. At the same time, we know that SGD may not converge without diminishing the learning rate. This update strategy provides for a trade-off between fixed learning rates and diminishing learning rates. Hence,

\[ \gamma_{\{\ell : (k-1)N \leq \ell \leq kN\}} = \frac{1}{k}, \text{ where } k = 1, 2, \ldots \text{ is the epoch index.} \]

The learning rate starts at 1 in the first epoch, and diminishes as \( \frac{1}{2}, \frac{1}{3}, \ldots \) in following epochs.

Fig. 3 (left) contrasts the objective values with this learning strategy with L-BFGS. Despite having no hyper-parameters, this update strategy for epoch-fixed learning rate appears to be quite effective. In addition to having a significant head-start compared to L-BFGS, it also converges to approximately the same objective value after 32 training epochs.

3.1.2 Cyclic vs. randomized order

Here, we consider two ways of choosing \( i \) (i.e., the sentence pair index in Eq. 5) for the \( t \)-th update (i.e., the SGD iteration index). We empirically test the theoretical results in [2] which favor the randomized order to the cyclic order. In the cyclic order, we choose \( i_t = 1, 2, \ldots, N, 1, 2, \ldots \). While using randomized order, every \( N \) consecutive indices \( \{i_{(k-1)N}, i_kN, \ldots, i_{k+1}N\} \) is a uniform random permutation of \( \{1, \ldots, N\} \), for \( k = 1, 2, \ldots \).

As shown in Fig. 3 (left), our experiments showed no noticeable difference between the cyclic vs. randomized order.
3.1.3 Averaged stochastic gradient descent

Instead of the SGD update in Eq. 5, the averaged stochastic gradient descent (ASGD) algorithm updates the parameters $\lambda$ as follows:

$$
\hat{\lambda} = \lambda^{(t-1)} - \gamma t \nabla f_i(\lambda^{(t-1)})
$$

$$
\lambda^{(t)} = \frac{1}{t} (\hat{\lambda} + \sum_{j=1}^{t-1} \lambda^{(j)})
$$

where $t = 1, \ldots, \infty; i \in \{1, \ldots, N\}$

ASGD enjoys an optimal (local) convergence rate of $O\left(\frac{1}{t}\right)$, assuming the learning rates decrease slower than $t^{-1}$. However, our empirical results suggest that there is no practical difference between the objective values obtained with SGD vs. ASGD.

3.1.4 Regularization

$L_p$-norm regularization modifies the criterion such that the gradient approximation based on each example is no longer sparse. In this work, we use an algorithm described in a blog post by Alex Smola\footnote{http://blog.smola.org/post/940672544/fast-quadratic-regularization-for-online-learning} that preserves the sparsity of the updates with $L_2$ regularization. The main idea is to store values of the variables being optimized (i.e., $\lambda$) in a scaled form (i.e., using a positive scalar and an array). When performing an SGD update, we update the scalar value to reflect the shrinkage due to the $L_2$ term, and only update values of the active variables (i.e., variables for which the unregularized criterion has non-zero derivatives) in the array.

3.1.5 Parallelization

In order to fully leverage the hardware in modern computers, we would like to perform SGD updates in parallel, on multiple processors on the same machine. We experimented asynchronous SGD updates with 1, 2, 4, and 8 processors using the Message Passing Interface (MPI) architecture\footnote{We omit engineering details about how we use MPI to parallelize SGD and L-BFGS to save space.}. We found that runtime cost of processing each epoch shrinks linearly with the number of processors, while the reduction in the criterion value is indistinguishable from using a single-core (see Fig. 3). As a result, in this work, SGD can be easily scaled to multiple cores asynchronously, with almost no loss of accuracy.

3.2 Optimizing w.r.t. $\theta$ using batch vs. online EM

In this section, we are only concerned about the following problem:

$$
\min_{\theta} \ell(\theta) = \sum_{(s,t,\hat{t})} \log \sum_{a} p(a, \hat{t} | s, t) \text{ s.t. } \sum_{t \in \mathcal{V}_s} \theta_{t|s} = 1, 0 \leq \theta_{t|s} \leq 1, \forall s \in \mathcal{V}_s
$$
**Batch EM.** Expectation Maximization (EM) is a popular method for optimizing parameters of generative models ($\theta$) with latent (i.e., unobserved) variables ($a$). In each iteration of batch EM, we update $\theta$ by solving: $\min_{\theta} E_{\theta^{old}}[\log p_{\theta}(a, t | s, t)]$ subject to the multinomial distribution constraints on $\theta$. Each iteration consists of two steps:

- E-step: compute the sufficient statistics ($\mu$) for estimating $\theta$ given $\theta^{old}$. The sufficient statistic for each parameter in $\theta$ turn out to be the expected number of times that parameter is being used to generate an observation.
- M-step: estimate $\theta$ given $\mu$ (by projecting to the probability simplex).

In practice, in our problem, the first iteration of batch EM achieves the most progress towards a local optimum value of the optimization criterion (see Fig. 4).

**Online EM.** Inspired by the high efficacy of using SGD, we were hoping to achieve more progress (in criterion value) by updating $\theta$ more frequently. In our experiments, we use an online EM variant proposed by Capp and Moulines [5], also known as “stepwise EM”. The batch and online EM algorithms are outlined below.

### Batch EM:

- $\mu :=$ initialize
- for each EM iteration $t = 1, \ldots, T$ :
  - $\mu' := 0$
  - for each example $i : (s, t, \hat{t})$
    - $m'_i := \sum_a p(a | s, t, \hat{t}; \theta(\mu)) \phi(a, s, t, \hat{t})$ [inference]
    - $\mu' := \mu' + m'_i$ [accumulate new]
    - $\mu := \mu'$ [replace old with new]

### Online EM:

- $\mu :=$ initialize, $k := 0$
- for each EM iteration $t = 1, \ldots, T$ :
  - for each example $i : (s, t, \hat{t})$ in random order
    - $m'_i := \sum_a p(a | s, t, \hat{t}; \theta(\mu)) \phi(a, s, t, \hat{t})$ [inference]
    - $\mu := (1 - \eta_k)\mu + \eta_k m'_i; k := k + 1$ [interpolate]

In the outlines above, $\mu$ is a vector of expected counts for corresponding parameters in $\theta$, $\phi$ is a function that maps a sentence pair and its alignment to a vector of sufficient statistics, $m'_i$ are the expected counts for a given sentence pair, and $\theta(\mu)$ is shorthand for the parameter values $\theta$ projected from sufficient statistics $\mu$ as in the M-step. Note that a projection is only necessary when $\mu$ changes though. So this operation is only performed once in batch EM, but many times in online EM.

### Learning rate.** The main idea behind the online EM algorithm is to interpolate between the sufficient statistics inferred from the current example (with weight $\eta_k$) and the accumulated sufficient statistics inferred from previous iterations of the algorithm (with weight $1 - \eta_k$). Following Liang and Klein [9], we use the following formula to control the interpolation parameter $\eta_k$ in the $k$th iteration of online EM:

$$\eta_k = (k + 2)^{-\alpha}$$  \hspace{1cm} (11)

It is instructive to consider extreme values of $\alpha$ and how they affect the interpolation. When $\alpha = 0$, the interpolation puts all the weight on the current example. When $\alpha = \infty$, the interpolation puts all the weight on previous examples. It is therefore appropriate to regard $\alpha$ as an (inverse) learning rate, or rather a “stickiness” rate. Our empirical results in Fig. 4 confirm the significant effect of $\alpha$. However, none of the learning rates we tried resulted in faster convergence than batch EM.

---

8 The expectation here is governed by $p_{\theta^{old}}(a | s, t, \hat{t})$.
9 The pseudocode has been adapted from [9].
Mini-batch size. Due to the projection step (M-step) in expectation maximization with multinomial-constrained parameters, the stochastic updates inevitably affects all parameters, including those with zero expected counts in the new example. As a result, it is critical for online EM to use more than one example (i.e., mini-batches) to compute the new sufficient statistics $m'_i$. In Fig. 4, we experiment with mini-batches of size $10^2$, $10^3$, $10^4$. At the limit, when the mini-batch size is $\infty$, we recover batch EM. None of the mini-batch sizes we tried resulted in faster convergence than batch EM.

![Figure 4: Left: Batch vs. online EM with different values of $\alpha$ (stickiness parameter). Right: Batch EM vs. online EM with mini-batch sizes of $10^2$, $10^3$, $10^4$.](image)

4 Conclusions

In this work, we implement stochastic optimization methods in the context of conditional random field (CRF) autoencoders. There are two variables $\lambda$ and $\theta$ in the objective function. We use stochastic gradient descent to optimize $\lambda$, and compare its performance to that of L-BFGS. In the experiments, we study the effect of different learning rate decay strategies. From the experiment results, SGD that uses epoch-fixed learning rate is the most effective, and it has a significant head-start compared to L-BFGS. In addition, SGD can be easily scaled to multiple cores asynchronously, with almost no loss of accuracy.

We use online EM to optimize $\theta$, and compare its performance to that of batch EM. In the experiments, we explore the effect of different interpolation parameters and mini-batch sizes. However, the online EM algorithm does not outperform the batch EM. As for future work, our plan includes trying alternative method to online EM; using the stochastic optimization implementation in an end-to-end machine translation program.

References


