# Non-Uniform Stochastic Average Gradient Method for Training Conditional Random Fields

**Mark Schmidt, Reza Babanezhad, Mohamed Osama Ahemd**
Department of Computer Science
University of British Columbia

**Aaron Defazio**
Ambiata
Australian National University

**Ann Clifton, Anoop Sarkar**
Department of Computing Science
Simon Fraser University

## Abstract

We apply stochastic average gradient (SAG) algorithms for training conditional random fields (CRFs). We give a practical implementation that uses structure in the CRF gradient to reduce the memory requirement of this linearly-convergent stohastic gradient method, propose a non-uniform sampling scheme that substantially improves practical performance, and analyze the rate of convergence of the SAGA variant under non-uniform sampling. Our experimental results reveal that our method significantly outperforms existing methods in terms of the training objective, and performs as well or better than optimally-tuned stochastic gradient methods in terms of test error.

## 1 Introduction

Conditional random fields (CRFs) [Lafferty et al., 2001] are a ubiquitous tool in natural language processing. They are used for part-of-speech tagging [McCallum et al., 2003], semantic role labeling [Cohn and Blunsom, 2005], topic modeling [Zhu and Xing, 2010], information extraction [Peng and McCallum, 2006], shallow parsing [Sha and Pereira, 2003], named-entity recognition [Settles, 2004], as well as a host of other applications in natural language processing and in other fields such as computer vision [Nowozin and Lampert, 2011]. Similar to generative Markov random field (MRF) models, CRFs allow us to model probabilistic dependencies between output variables. The key advantage of discriminative CRF models is the ability to use a very high-dimensional feature set, without explicitly building a model for these features (as required by MRF models). Despite the widespread use of CRFs, a major disadvantage of these models is that they can be very slow to train and the time needed for numerical optimization in CRF models remains a bottleneck in many applications.

Due to the high cost of evaluating the CRF objective function on even a single training example, it is now common to train CRFs using stochastic gradient methods [Vishwanathan et al., 2006]. These methods are advantageous over deterministic methods because on each iteration they only require computing the gradient of a single example (and not *all* example as in deterministic methods). Thus, if we have a data set with $n$ training examples, the iterations of stochastic gradient methods are $n$ times faster than deterministic methods. However, the number of stochastic gradient iterations required might be very high. This has been studied in the optimization community, which considers the problem of finding the minimum number of iterations $t$ so that we can guarantee that we reach an accuracy of $\epsilon$, meaning that

$$f(w^t) - f(w^*) \leq \epsilon, \text{ and } \|w^t - w^*\|^2 \leq \epsilon,$$

where $f$ is our training objective function, $w^t$ is our parameter estimate on iteration $t$, and $w^*$ is the parameter vector minimizing the training objective function. For strongly-convex objectives like $\ell_2$-regularized CRFs, stochastic gradient methods require $O(1/\epsilon)$ iterations [Nemirovski et al., 2009]. This is in contrast to traditional deterministic methods which only require $O(\log(1/\epsilon))$ iterations [Nesterov, 2004]. However, this

Appearing in Proceedings of the $18^{th}$ International Conference on Artificial Intelligence and Statistics (AISTATS) 2015, San Diego, CA, USA. JMLR: W&CP volume 38.

much faster (polynomial) number of iterations comes at the cost of requiring us to process the entire data set on each iteration.

For problems with a finite number of training examples, Le Roux et al. [2012] recently proposed the stochastic average gradient (SAG) algorithm which combines the advantages of deterministic and stochastic methods: it only requires evaluating a single randomly-chosen training example on each iteration, and only requires $O(\log(1/\epsilon))$ iterations to reach an accuracy of $\epsilon$. Beyond this faster convergence rate, the SAG method also allows us to address two issues that have traditionally frustrated users of stochastic gradient methods: *setting the step-size* and *deciding when to stop*. Implementations of the SAG method use both an adaptive step-size procedure and a cheaply-computable criterion for deciding when to stop. Le Roux et al. [2012] show impressive empirical performance of the SAG algorithm for binary classification.

This is the first work to apply a SAG algorithm to train CRFs. We show that tracking marginals in the CRF can drastically reduce the SAG method's huge memory requirement. We also give a non-uniform sampling (NUS) strategy that adaptively estimates how frequently we should sample each data point, and we analyze show that the SAG-like algorithm of Defazio et al. [2014] converges under any NUS strategy and that a particular NUS strategy achieves a faster rate. Our experiments compare the SAG algorithm with a variety of competing deterministic, stochastic, and semi-stochastic methods on benchmark data sets for four common tasks: part-of-speech tagging, named entity recognition, shallow parsing, and optical character recognition. Our results indicate that the SAG algorithm with NUS can outperform previous methods by an order of magnitude in terms of the training objective and, despite not requiring us to tune the step-size, performs as well or better than optimally tuned stochastic gradient methods in terms of the test error.

## 2 Conditional Random Fields

CRFs model the conditional probability of a structured output $y \in \mathcal{Y}$ (such as a sequence of labels) given an input $x \in \mathcal{X}$ (such as a sequence of words) based on features $F(x, y)$ and parameters $w$ using

$$p(y|x, w) = \frac{\exp(w^T F(x, y))}{\sum_{y'} \exp(w^T F(x, y'))}. \qquad (1)$$

Given $n$ pairs $\{x_i, y_i\}$ comprising our training set, the standard approach to training the CRF is to minimize the $\ell_2$-regularized negative log-likelihood,

$$\min_w f(w) = \frac{1}{n} \sum_{i=1}^{n} -\log p(y_i|x_i, w) + \frac{\lambda}{2}\|w\|^2, \qquad (2)$$

where $\lambda > 0$ is the strength of the regularization parameter. Unfortunately, evaluating $\log p(y_i|x_i, w)$ is expensive due to the summation over all possible configurations $y'$. For example, in chain-structured models the forward-backward algorithm is used to compute $\log p(y_i|x_i, w)$ and its gradient. A second problem with solving (2) is that the number of training examples $n$ in applications is constantly-growing, and thus we would like to use methods that only require a few passes through the data set.

## 3 Related Work

Lafferty et al. [2001] proposed an iterative scaling algorithm to solve problem (2), but this proved to be inferior to generic deterministic optimization strategies like the limited-memory quasi-Newton algorithm L-BFGS [Wallach, 2002, Sha and Pereira, 2003]. The bottleneck in these methods is that we must evaluate $\log p(y_i|x_i, w)$ and its gradient for all $n$ training examples on every iteration. This is be very expensive for problems where $n$ is very large, so to deal with this problem stochastic gradient methods were examined [Vishwanathan et al., 2006, Finkel et al., 2008]. However, traditional stochastic gradient methods require $O(1/\epsilon)$ iterations rather than the much smaller $O(\log(1/\epsilon))$ required by deterministic methods.

There have been several attempts at improving the cost of deterministic methods or the convergence rate of stochastic methods. For example, the exponentiated gradient method of Collins et al. [2008] processes the data online and only requires $O(\log(1/\epsilon))$ iterations to reach an accuracy of $\epsilon$ in terms of the dual objective. However, this does not guarantee good performance in terms of the primal objective or the weight vector. Although this method is highly-effective if $\lambda$ is very large, our experiments and the experiments of others show that the performance of online exponentiated gradient degrades substantially if a small value of $\lambda$ is used (which may be required to achieve the best test error), see Collins et al. [2008, Figures 5-6 and Table 3] and Lacoste-Julien et al. [2013, Figure 1]. In contrast, SAG degrades more gracefully as $\lambda$ becomes small, even achieving a convergence rate faster than classic SG methods when $\lambda = 0$ [Schmidt et al., 2013]. Lavergne et al. [2010][1] consider using multiple processors and vectorized computation to reduce the high iteration cost of quasi-Newton methods, but

[1] http://wapiti.limsi.fr/

when $n$ is enormous these methods still have a high iteration cost. Friedlander and Schmidt [2012] explore a hybrid deterministic-stochastic method that slowly grows the number of examples that are considered in order to achieve an $O(\log(1/\epsilon))$ convergence rate with a decreased cost compared to deterministic methods.

Below we state the convergence rates of different methods for training CRFs, including the fastest known rates for deterministic algorithms (like L-BFGS and accelerated gradient) [Nesterov, 2004], stochastic algorithms (like [averaged] stochastic gradient and AdaGrad) [Ghadimi and Lan, 2012], online exponentiated gradient, and SAG. Here $L$ is the Lipschitz constant of the gradient of the objective, $\mu$ is the strong-convexity constant (and we have $\lambda \leq \mu \leq L$), and $\sigma^2$ bounds the variance of the gradients.

| | | |
|---|---|---|
| Deterministic: | $O(n\sqrt{\frac{L}{\mu}}\log(1/\epsilon))$ | (primal) |
| Online EG | $O((n + \frac{L}{\lambda})\log(1/\epsilon))$ | (dual) |
| Stochastic | $O(\frac{\sigma^2}{\mu\epsilon} + \sqrt{\frac{L}{\mu\epsilon}})$ | (primal) |
| SAG | $O((n + \frac{L}{\mu})\log(1/\epsilon))$ | (primal) |

## 4 Stochastic Average Gradient

Le Roux et al. [2012] introduce the SAG algorithm, a simple method with the low iteration cost of stochastic gradient methods but that only requires $O(\log(1/\epsilon))$ iterations. To motivate this new algorithm, we write the classic gradient descent iteration as

$$w^{t+1} = w^t - \frac{\alpha}{n}\sum_{i=1}^{n} s_i^t, \tag{3}$$

where $\alpha$ is the step-size and at each iteration we set the 'slope' variables $s_i^t$ to the gradient with respect to training example $i$ at $w^t$, so that $s_i^t = -\nabla \log p(y_i|x_i, w^t) + \lambda w^t$. The SAG algorithm uses this same iteration, but instead of updating $s_i^t$ for all $n$ data points on every iterations, it simply sets $s_i^t = -\nabla \log p(y_i|x_i, w^t) + \lambda w^t$ for *one randomly chosen* data point and keeps the remaining $s_i^t$ at their value from the previous iteration. Thus the SAG algorithm is a randomized version of the gradient algorithm where we use the gradient of each example from the last iteration where it was selected. The surprising aspect of the work of Le Roux et al. [2012] is that this simple *delayed* gradient algorithm achieves a similar convergence rate to the classic full gradient algorithm despite the iterations being $n$ times faster.

### 4.1 Implementation for CRFs

Unfortunately, a major problem with applying (3) to CRFs is the requirement to store $s_i^t$. While the CRF gradients $\nabla \log p(y_i|x_i, w^t)$ for all $i$ have a nice structure (see Section 4.2), $s_i^t$ includes $\lambda w^t$ for some previous $t$, which is dense and unstructured. To get around this issue, instead of using (3) we use the following SAG-like update [Le Roux et al., 2012, Section 4]

$$\begin{aligned} w^{t+1} &= w^t - \alpha(\frac{1}{m}\sum_{i=1}^{n} g_i^t + \lambda w^t) \\ &= w^t - \alpha(\frac{1}{m}d + \lambda w^t) \\ &= (1 - \alpha\lambda)w^t - \frac{\alpha}{m}d, \end{aligned} \tag{4}$$

where $g_i^t$ is the value of $-\nabla \log p(y_i|x_i, w^k)$ for the last iteration $k$ where $i$ was selected and $d$ is the sum of the $g_i^t$ over all $i$. Thus, this update uses the exact gradient of the regularizer and only uses an approximation for the (structured) CRF log-likelihood gradients. Since we don't yet have any information about these log-likelihoods at the start, we initialize the algorithm by setting $g_i^0 = 0$. But to compensate for this, we track the number of examples seen $m$, and normalize $d$ by $m$ in the update (instead of $n$). In Algorithm 1, we summarize this variant of the SAG algorithm for training CRFs.[2] In many applications of CRFs the $g_i^t$ are very sparse, and we would like to take advantage of this as in stochastic gradient methods. Fortunately, we can implement (4) without using dense vector operations by using the representation $w^t = \beta^t v^t$ for a

**Algorithm 1** SAG algorithm for training CRFs

```
Require: ${x_i, y_i}$, $\lambda$, $w$, $\delta$
$m \leftarrow 0$, $g_i \leftarrow 0$ for $i = 1, 2, \ldots, n$
$d \leftarrow 0$, $L_g \leftarrow 1$
while $m < n$ and $\|\frac{1}{n}d + \lambda w\|_\infty \geq \delta$ do
    Sample $i$ from ${1, 2, \ldots, n}$
    $f \leftarrow -\log p(y_i|x_i, w)$
    $g \leftarrow -\nabla \log p(y_i|x_i, w)$
    if this is the first time we sampled $i$ then
        $m \leftarrow m + 1$
    end if
    Subtract old gradient $g_i$, add new gradient $g$:
    $d \leftarrow d - g_i + g$
    Replace old gradient of example $i$:
    $g_i \leftarrow g$
    if $\|g_i\|^2 > 10^{-8}$ then
        $L_g \leftarrow$lineSearch$(x_i, y_i, f, g_i, w, L_g)$
    end if
    $\alpha \leftarrow 1/(L_g + \lambda)$
    $w \leftarrow (1 - \alpha\lambda)w - \frac{\alpha}{m}d$
    $L_g \leftarrow L_g \cdot 2^{-1/n}$
end while
```

[2] If we solve the problem for a sequence of regularization parameters, we can obtain better performance by warm-starting $g_i^0$, $d$, and $m$.

scalar $\beta^t$ and a vector $v^t$, and using 'lazy updates' that apply $d$ repeatedly to an individual variable when it is needed [Le Roux et al., 2012].

Also following Le Roux et al. [2012], we set the step-size to $\alpha = 1/L$, where $L$ is an approximation to the maximum Lipschitz constant of the gradients. This is the smallest number $L$ such that

$$\|\nabla f_i(w) - \nabla f_i(v)\| \leq L\|w - v\|, \tag{5}$$

for all $i$, $w$, and $v$. This quantity is a bound on how fast the gradient can change as we change the weight vector. The Lipschitz constant with respect to the gradient of the regularizer is simply $\lambda$. This gives $L = L_g + \lambda$, where $L_g$ is the Lipschitz constant of the gradient of the log-likelihood. Unfortunately, $L_g$ depends on the covariance of the CRF and is typically too expensive to compute. To avoid this computation, as in Le Roux et al. [2012] we approximate $L_g$ in an online fashion using the standard backtracking line-search given by Algorithm 2 [Beck and Teboulle, 2009]. The test used in this algorithm is faster than testing (5), since it uses function values (which only require the forward algorithm for CRFs) rather than gradient values (which require the forward and backward steps). Algorithm 2 monotonically increases $L_g$, but we also slowly decrease it in Algorithm 1 in order to allow the possibility that we can use a more aggressive step-size as we approach the solution.

**Algorithm 2** Lipschitz line-search algorithm

```
Require: $x_i, y_i, f, g_i, w, L_g$.
$f' = -\log p(y_i|x_i, w - \frac{1}{L_g} g_i)$
while $f' \geq f - \frac{1}{2L_g}\|g_i\|^2$ do
    $L_g = 2L_g$
    $f' = -\log p(y_i|x_i, w - \frac{1}{L_g} g_i)$
end while
return $L_g$.
```

Since the solution is the only stationary point, we must have $\nabla f(w^t) = 0$ at the solution. Further, the value $\frac{1}{n}d + \lambda w^t$ converges to $\nabla f(w^t)$ so we can use the size of this value to decide when to stop the algorithm (although we also require that $m = n$ to avoid premature stopping before we have seen the full data set). This is in contrast to classic stochastic gradient methods, where the step-size must go to zero and it is therefore difficult to decide if the algorithm is close to the optimal value or if we simply require a small step-size to continue making progress.

### 4.2 Reducing the Memory Requirements

Even if the gradients $g_i^t$ are not sparse, we can often reduce the memory requirements of Algorithm 1 because it is known that the CRF gradients only depend on $w$ through marginals of the features. Specifically, the gradient of the log-likelihood under model (1) with respect to feature $j$ is given by

$$\begin{aligned}\nabla_j \log p(y|x,w) &= F_j(x,y) - \frac{\sum_{y'} \exp(F(x,y'))}{\sum_{y'} F(x,y')} F_j(x,y') \\ &= F_j(x,y) - \sum_{y'} p(y'|x,w) F_j(x,y') \\ &= F_j(x,y) - \mathbb{E}_{y'|x,w}[F_j(x,y')]\end{aligned}$$

Typically, each feature $j$ only depends on a small 'parts' of $y$. For example, we typically include features of the form $F_j(x,y) = F(x)\mathbb{I}[y_k = s]$ for some function $F$, where $k$ is an element of $y$ and $s$ is a discrete state that $y_k$ can take. In this case, the gradient can be written in terms of the marginal probability of element $y_k$ taking state $s$,

$$\begin{aligned}\nabla_j \log p(y|x,w) &= F(x)\mathbb{I}[y_k = s] - \mathbb{E}_{y'|x,w}[F(x)\mathbb{I}[y_k = s]] \\ &= F(x)(\mathbb{I}[y_k = s] - \mathbb{E}_{y'|x,w}[\mathbb{I}[y_k = s]) \\ &= F(x)(\mathbb{I}[y_k = s] - p(y_k = s|x,w)).\end{aligned}$$

Notice that Algorithm 1 only depends on the old gradient through its difference with the new gradient (line 10), which in this example gives

$$\nabla_j \log p(y|x,w) - \nabla_j \log p(y|x,w_{\text{old}}) = \\ F(x)(p(y_k = s|x,w_{\text{old}}) - p(y_k = s|x,w)),$$

where $w$ is the current parameter vector and $w_{\text{old}}$ is the old parameter vector. Thus, to perform this calculation the only thing we need to know about $w_{\text{old}}$ is the unary marginal $p(y_k = s|x,w_{\text{old}})$, which will be *shared* across features that only depend on the event that $y_k = s$. Similarly, features that depend on pairs of values in $y$ will need to store pairwise marginals, $p(y_k = s, y'_k = s'|x,w_{\text{old}})$. For general pairwise graphical model structures, the memory requirements to store these marginals will thus be $O(VK + EK^2)$, where $V$ is the number of vertices and $E$ is the number of edges. This can be an enormous reduction since *it does not depend on the number of features.* Further, since computing these marginals is a by-product of computing the gradient, this potentially-enormous reduction in the memory requirements comes at no extra computational cost.

## 5 Non-Uniform Sampling

Recently, several works show that we can improve the convergence rates of randomized optimization algorithms by using non-uniform sampling (NUS) schemes. This inculdes randomized Kaczmarz [Strohmer and Vershynin, 2009], randomized coordinate descent [Nesterov, 2012], and stochastic gradient methods [Needell et al., 2014]. The key idea behind all of these NUS strategies is to *bias the sampling towards the Lipschitz constants of the gradients*, so that gradients that change quickly get sampled more often and gradients that change slowly get sampled less often. Specifically, we maintain a Lipschitz constant $L_i$ for each training example $i$ and, instead of the usual sampling strategy $p(i) = 1/n$, we bias towards the distribution $p(i) = L_i / \sum_j L_j$. In these various contexts, NUS allows us to improve the dependence on the values $L_i$ in the convergence rate, since the NUS methods depend on $\bar{L} = (1/n)\sum_j L_j$, which may be substantially smaller than the usual dependence on $L = \max_j\{L_j\}$. Schmidt et al. [2013] argue that faster convergence rates might be achieved with NUS for SAG since it allows a larger step size $\alpha$ that depends on $\bar{L}$ instead of $L$.[3]

The scheme for SAG proposed by Schmidt et al. [2013, Section 5.5] uses a fairly complicated adaptive NUS scheme and step-size, but the key ingredient is estimating each constant $L_i$ using Algorithm 2. Our experiments show this method already improves on state of the art methods for training CRFs by a substantial margin, but we found we could obtain improved performance for training CRFs using the following simple NUS scheme for SAG: as in Needell et al. [2014], with probability 0.5 choose $i$ uniformly and with probability 0.5 sample $i$ with probability $L_i/(\sum_j L_j)$ (restricted to the examples we have previously seen).[4] We also use a step-size of $\alpha = \frac{1}{2}\left(1/L + 1/\bar{L}\right)$, since the faster convergence rate with NUS is due to the ability to use a larger step-size than $1/L$. This simple step-size and sampling scheme contrasts with the more complicated choices described by Schmidt et al. [2013, Section 5.5], that make the degree of non-uniformity grow with the number of examples seen $m$. This prior work initializes each $L_i$ to 1, and updates $L_i$ to $0.5L_i$ each subsequent time an example is chosen. In the context of CRFs, this leads to a large number of expensive backtracking iterations. To avoid this, we initialize $L_i$ with $0.5\bar{L}$ the first time an example is chosen, and decrease $L_i$ to $0.9L_i$ each time it is subsequently chosen.

### 5.1 Convergence Analysis under NUS

Schmidt et al. [2013] give an intuitive but non-rigorous motivation for using NUS in SAG. More recently, Xiao and Zhang [2014] show that NUS gives a dependence on $\bar{L}$ in the context of a related algorithm that uses occasional full passes through the data (which substantially simplifies the analysis). Below, we consider a NUS extension of the SAGA algorithm of Defazio et al. [2014], which does not require full passes through the data and has similar performance to SAG in practice but is much easier to analyze. This result shows that SAGA has (a) a linear convergence rate for any NUS scheme where $p_j > 0$ for all $i$, and (b) a rate depending on $\bar{L}$ by sampling proportional to the Lipschitz constants and also generating a uniform sample. However, (a) achieves the fastest rate when $p_j = 1/n$ while (b) requires two samples on each iteration. We were not able to show a faster rate using only one sample on each iteration as used in our implementation.

**Proposition 1** *Let the sequences $\{w^t\}$ and $\{s^t_j\}$ be defined by*

$$w^{t+1} = w^t - \alpha\left[\frac{1}{p_j n}(\nabla f_{j_t}(w^t) - s^t_{j_t}) + \frac{1}{n}\sum_{i=1}^n s^t_i\right],$$

$$s^{t+1}_j = \begin{cases}\nabla f_{r_t}(w^t) & \text{if } j = r_t,\\ s^t_j & \text{otherwise.}\end{cases}$$

*where $j_t$ is chosen with probability $p_j$.*

*(a) If $r_t$ is set to $j_t$, then with $\alpha = \frac{np_{min}}{4L+n\mu}$ we have*

$$\mathbb{E}[\|w^t - w^*\|^2] \leq (1-\mu\alpha)^t\left[\|x^0 - x^*\| + C_a\right],$$

*where $p_{min} = \min_i\{p_i\}$ and*

$$C_a = \frac{2p_{min}}{(4L+n\mu)^2}\sum_{i=1}^n \frac{1}{p_i}\|\nabla f_i(x^0) - \nabla f_i(x^*)\|^2.$$

*(b) If $p_j = \frac{L_j}{\sum_{i=1}^n L_i}$ and $r_t$ is chosen uniformly at random, then with $\alpha = \frac{1}{4\bar{L}}$ we have*

$$\mathbb{E}[\|w^t - w^*\|^2] \leq \left(1 - \min\left\{\frac{1}{3n}, \frac{\mu}{8\bar{L}}\right\}\right)^t\left[\|x^0 - x^*\| + C_b\right],$$

*where:*

$$C_b = \frac{n}{2\bar{L}}\left[f(x^0) - f(x^*)\right]$$

## 6 Experiments

We compared a wide variety of approaches on four CRF training tasks: the optical character recognition (OCR) dataset of Taskar et al. [2003], the CoNLL-2000 shallow parse chunking dataset,[5] the CoNLL-2002 Dutch named-entity recognition dataset,[6] and a part-of-speech (POS) tagging task using the Penn Treebank Wall Street Journal data (POS-WSJ). The

[3] An interesting difference between the SAG update with NUS and NUS for stochastic gradient methods is that the SAG update does not seem to need to decrease the step-size for frequently-sampled examples (since the SAG update does not rely on using an unbiased gradient estimate).

[4] Needell et al. [2014] only analyze the basic stochastic gradient method and thus require $O(1/\epsilon)$ iterations.

[5] http://www.cnts.ua.ac.be/conll2000/chunking

[6] http://www.cnts.ua.ac.be/conll2002/ner

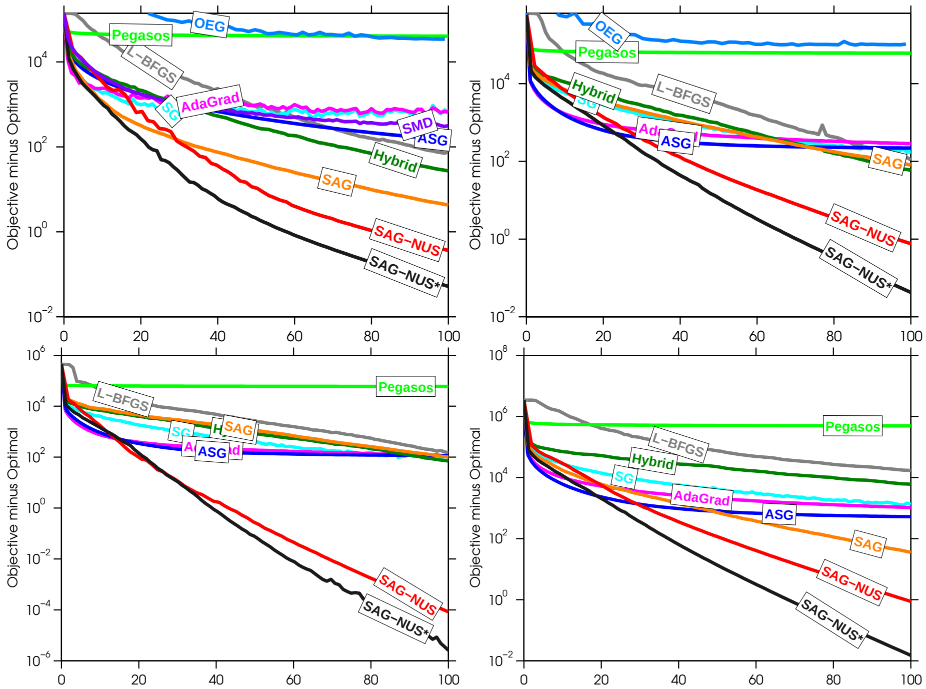

Figure 1: Objective minus optimal objective value against effective number of passes for different deterministic, stochastic, and semi-stochastic optimization strategies. Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ.

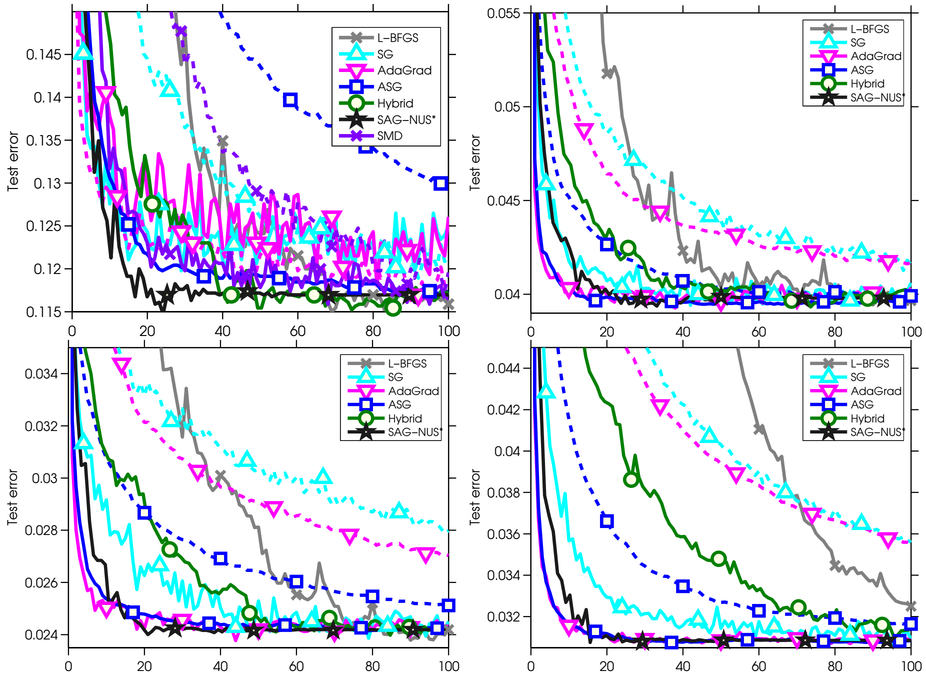

Figure 2: Test error against effective number of passes for different deterministic, stochastic, and semi-stochastic optimization strategies. Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ. The dotted lines show the performance of the classic stochastic gradient methods when the optimal step-size is not used. *Note that the performance of all classic stochastic gradient methods is much worse when the optimal step-size is not used*, whereas the SAG methods have an adaptive step-size so are not sensitive to this choice.

optimal character recognition dataset labels the letters in images of words. Chunking segments a sentence into syntactic chunks by tagging each sentence token with a chunk tag corresponding to its constituent type (e.g., ‘NP’, ‘VP’, etc.) and location (e.g., beginning, inside, ending, or outside any constituent). We use standard n-gram and POS tag features [Sha and Pereira, 2003]. For the named-entity recognition task, the goal is to identify named entities and correctly classify them as persons, organizations, locations, times, or quantities. We again use standard n-gram and POS tag features, as well as word shape features over the case of the characters in the token. The POS-tagging task assigns one of 45 syntactic tags to each token in each of the sentences in the data. For this data, we follow the standard division of the WSJ data given by Collins [2002], using sections 0-18 for training, 19-21 for development, and 22-24 for testing. We use the standard set of features following Ratnaparkhi [1996] and Collins [2002]: n-gram, suffix, and shape features. As is common on these tasks, our pairwise features do not depend on $x$.

To quantify the memory savings given by the choices in Section 4, below we report the size of the memory required for these datasets under different memory-saving strategies divided by the memory required by the naive SAG algorithm. *Sparse* refers to only storing non-zero gradient values, *Marginals* refers to storing all unary and pairwise marginals, and *Mixed* refers to storing node marginals and the gradient with respect to pairwise features.

| Dataset | Sparse | Marginals | Mixed |
|---|---|---|---|
| OCR | $7.8 \times 10^{-1}$ | $1.1 \times 10^{0}$ | $2.1 \times 10^{-1}$ |
| CoNLL-2000 | $4.8 \times 10^{-3}$ | $7.0 \times 10^{-3}$ | $6.1 \times 10^{-4}$ |
| CoNLL-2002 | $6.4 \times 10^{-4}$ | $3.8 \times 10^{-4}$ | $7.0 \times 10^{-5}$ |
| POS-WJ | $1.3 \times 10^{-3}$ | $5.5 \times 10^{-3}$ | $3.6 \times 10^{-4}$ |

On these datasets we compared the performance of a set of competitive methods, including five variants on classic stochastic gradient methods: *Pegasos* which is a standard stochastic gradient method with a step-size of $\alpha = \eta/\lambda t$ on iteration $t$ [Shalev-Shwartz et al., 2011],[7] a basic stochastic gradient (*SG*) method where we use a constant $\alpha = \eta$, an averaged stochastic gradient (*ASG*) method where we use a constant step-size $\alpha = \eta$ and average the iterations,[8] *AdaGrad* where we use the per-variable $\alpha_j = \eta/(\delta + \sqrt{\sum_{i=1}^{t} \nabla_j \log p(y_i|x_i, w^i)^2})$ and the proximal-step with respect to the $\ell_2$-regularizer [Duchi et al., 2011], and stochastic meta-descent (*SMD*) where we initialize with $\alpha_j = \eta$ and dynamically update the step-size [Vishwanathan et al., 2006]. Since setting the step-size is a notoriously hard problem when applying stochastic gradient methods, we let these classic stochastic gradient methods cheat by choosing the $\eta$ which gives the best performance among powers of 10 on the training data (for SMD we additionally tested the four choices among the paper and associated code of Vishwanathan et al. [2006], and we found $\delta = 1$ worked well for *AdaGrad*). Our comparisons also included a deterministic *L-BFGS* algorithm and the *Hybrid* L-BFGS/stochastic algorithm of Friedlander and Schmidt [2012]. We also included the online exponentiated gradient *OEG* method of [Collins et al., 2008], but we found it had very poor performance for the small values of $\lambda$ that approach the optimal test error.[9] Finally, we included the *SAG* algorithm as described in Section 4, the *SAG-NUS* variant of Schmidt et al. [2013], and our proposed *SAG-NUS** strategy from Section 5.[10]

Figure 1 shows the result of our experiments on the training objective and Figure 2 shows the result of tracking the test error. Here we measure the number of ‘effective passes’, meaning $(1/n)$ times the number of times we performed the bottleneck operation of computing $\log p(y_i|x_i, w)$ and its gradient. This is an implementation-independent way to compare the convergence of the different algorithms (whose runtimes differ only by a small constant), but we have included the performance in terms of runtime as supplementary material. For the different *SAG* methods that use a line-search we count the extra ‘forward’ operations used by the line-search as full evaluations of $\log p(y_i|x_i, w)$ and its gradient, even though these operations are cheaper because they do not require the backward pass nor computing the gradient. In these experiments we used $\lambda = 1/n$, which yields a value close to the optimal test error across all data sets. The objective is strongly-convex and thus has a unique minimum value. We approximated this value by running L-BFGS for up to 1000 iterations, which always gave a value of $w$ satisfying $\|\nabla f(w)\|_\infty \leq 1.4 \times 10^{-7}$, indicating that this is a very accurate approximation of the true solution. In the test error plots, we have excluded the *SAG* and *SAG-NUS* methods to keep the plots interpretable (while Pegasos and OEG do not appear becuase they perform very poorly), but the supplemental material includes these plots with all meth-

[7] We also tested *Pegasos* with averaging but it always performed worse than the non-averaged version.

[8] We also tested *SG* and *ASG* with decreasing step-sizes of either $\alpha_t = \eta/\sqrt{t}$ or $\alpha_t = \eta/(\delta + t)$, but these gave worse performance than using a constant step size.

[9] Because of the extra implementation effort required to implement them efficiently, we did not test OEG and SMD on all datasets. However, they clearly do not achieve the best performance across all dataests.

[10] We also tested *SG* with the proposed NUS scheme, but the performance was similar to the regular SG method. This is consistent with the analysis of Needell et al. [2014, Corollary 3.1] showing that NUS for regular *SG* only improves the non-dominant term.

ods added. In the test error plots, we have also plotted as dotted lines the performance of the classic stochastic gradient methods when the second-best step-size is used.

We make several observations based on these experiments:

- *SG* outperformed *Pegasos.* Pegasos is known to move exponentially away from the solution in the early iterations [Bach and Moulines, 2011], meaning that $\|w^t - w^*\| \geq \rho^t \|w^0 - w^*\|$ for some $\rho > 1$, while SG moves exponentially towards the solution ($\rho < 1$) in the early iterations [Nedic and Bertsekas, 2000].
- *ASG* outperformed *AdaGrad* and *SMD.* ASG methods are known to achieve the same asymptotic efficiency as an optimal stochastic Newton method [Polyak and Juditsky, 1992], while AdaGrad and SMD can be viewed as approximations to a stochastic Newton method.
- *Hybrid* outperformed *L-BFGS.* The hybrid algorithm processes fewer data points in the early iterations, leading to cheaper iterations.
- None of the three competitive algorithms *ASG*/*Hybrid*/*SAG* dominated the others: the relative ranks of these methods changed based on the data set and whether we could choose the optimal step-size.
- Both *SAG-NUS* methods outperform all other methods by a substantial margin based on the training objective, and are always among the best methods in terms of the test error. Further, our proposed *SAG-NUS** always performed as well or better than *SAG-NUS.*

On three of the four data sets, the best classic stochastic gradient methods (*AdaGrad* and *ASG*) seem to reach the optimal test error with a similar speed to the *SAG-NUS** method, although they require many passes to reach the optimal test error on the OCR data. Further, we see that the good test error performance of the *AdaGrad* and *ASG* methods *is very sensitive to choosing the optimal step-size*, as the methods perform much worse if we don't use the optimal step-size (dashed lines in Figure 2). In contrast, SAG uses an adaptive step-size and has virtually identical performance even if the initial value of $L_g$ is too small by several orders of magnitude (the line-search will quickly increase $L_g$ to a reasonable value on the first training example, so the dashed black line in Figure 2 would be on top of the solid line).

# 7 Discussion

Due to its memory requirements, it may be difficult to apply the SAG algorithm for natural language applications involving complex features that depend on a large number of labels. However, grouping training examples into mini-batches can also reduce the memory requirement (since only the gradients with respect to the mini-batches would be needed). An alternative strategy for reducing the memory is to use the algorithm of Johnson and Zhang [2013] or Zhang et al. [2013]. These require evaluating the chosen training example twice on each iteration, and occasionally require full passes through the data, but do not have the memory requirements of SAG (in our experiments, these performed similar to or slightly worse than running SAG at half speed).

We believe linearly-convergent stochastic gradient algorithms with non-uniform sampling could give a substantial performance improvement in a large variety of CRF training problems, and we emphasize that the method likely has extensions beyond what we have examined. For example, we have focused on the case of $\ell_2$-regularization but for large-scale problems there is substantial interest in solving large $\ell_1$-regularized CRFs [Tsuruoka et al., 2009, Lavergne et al., 2010, Zhou et al., 2011]. Fortunately, such non-smooth regularizers can be handled with a proximal-gradient variants of the method, see [Defazio et al., 2014]. While we have considered chain-structured data the algorithm applies to general graph structures, and any method for computing/approximating the marginals could be adopted. Finally, the SAG algorithm could be modified to use multi-threaded computation as in the algorithm of Lavergne et al. [2010], and indeed might be well-suited to massively distributed parallel implementations.

# Acknowledgments

We would like to thank the reviewers for their helpful comments. This research was partially supported by the Natural Sciences and Engineering Research Council of Canada (RGPIN 262313, RGPAS 446348, and CRDPJ 412844 which was made possible by a generous contribution from The Boeing Company and AeroInfo Systems). Travel support to attend the conference was provided by the Institute for Computing, Information and Cognitive Systems.

## References

F. Bach and E. Moulines. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. *Advances in Neural Information Processing Systems*, 2011.

A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

T. Cohn and P. Blunsom. Semantic role labelling with tree conditional random fields. *Conference on Computational Natural Language Learning*, 2005.

M. Collins. Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. *Conference on Empirical Methods in Natural Language Processing*, 2002.

M. Collins, A. Globerson, T. Koo, X. Carreras, and P. Bartlett. Exponentiated gradient algorithms for conditional random fields and max-margin Markov networks. *The Journal of Machine Learning Research*, 9:1775–1822, 2008.

A. Defazio, F. Bach, and S. Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. *Advances in Neural Information Processing Systems*, 2014.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

J. R. Finkel, A. Kleeman, and C. D. Manning. Efficient, feature-based, conditional random field parsing. *Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2008.

M. P. Friedlander and M. Schmidt. Hybrid deterministic-stochastic methods for data fitting. *SIAM Journal of Scientific Computing*, 34(3): A1351–A1379, 2012.

S. Ghadimi and G. Lan. Optimal stochastic approximation algorithms for strongly convex stochastic composite optimization i: A generic algorithmic framework. *SIAM Journal on Optimization*, 22(4): 1469–1492, 2012.

R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in Neural Information Processing Systems*, 2013.

S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher. Block-coordinate frank-wolfe optimization for structural svms. *International Conference on Machine Learning*, 2013.

J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *International Conference on Machine Learning*, 2001.

T. Lavergne, O. Cappé, and F. Yvon. Practical very large scale CRFs. In *Proceedings the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 504–513, 2010.

N. Le Roux, M. Schmidt, and F. Bach. A stochastic gradient method with an exponential convergence rate for strongly-convex optimization with finite training sets. *Advances in Neural Information Processing Systems*, 2012.

A. McCallum, K. Rohanimanesh, and C. Sutton. Dynamic conditional random fields for jointly labeling multiple sequences. In *NIPS Workshop on Syntax, Semantics, Statistics*, 2003.

A. Nedic and D. Bertsekas. Convergence rate of incremental subgradient algorithms. In *Stochastic Optimization: Algorithms and Applications*, pages 263–304. Kluwer Academic, 2000.

D. Needell, N. Srebro, and R. Ward. Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. *Advances in Neural Information Processing Systems*, 2014.

A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.

Y. Nesterov. *Introductory lectures on convex optimization: A basic course*. Springer, 2004.

Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM J. Optim.*, 22(2):341–362, 2012.

S. Nowozin and C. H. Lampert. Structured learning and prediction in computer vision. *Foundation and Trends in Computer Vision*, 6, 2011.

F. Peng and A. McCallum. Information extraction from research papers using conditional random fields. *Information Processing & Management*, 42 (4):963–979, 2006.

B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

A. Ratnaparkhi. A maximum entropy model for part-of-speech tagging. *Conference on Empirical Methods in Natural Language Processing*, 1996.

M. Schmidt, N. Le Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. *arXiv preprint*, 2013.

B. Settles. Biomedical named entity recognition using conditional random fields and rich feature sets. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, 2004.

F. Sha and F. Pereira. Shallow parsing with conditional random fields. *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technology*, 2003.

S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

T. Strohmer and R. Vershynin. A randomized Kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15(2):262–278, 2009.

B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. *Advances in Neural Information Processing Systems*, 2003.

Y. Tsuruoka, J. Tsujii, and S. Ananiadou. Stochastic gradient descent training for L1-regularized log-linear models with cumulative penalty. *Annual Meeting of the Association for Computational Linguisitics*, pages 477–485, 2009.

S. Vishwanathan, N. N. Schraudolph, M. W. Schmidt, and K. P. Murphy. Accelerated training of conditional random fields with stochastic gradient methods. *Iinternational conference on Machine learning*, 2006.

H. Wallach. Efficient training of conditional random fields. Master's thesis, University of Edinburgh, 2002.

L. Xiao and T. Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(2):2057–2075, 2014.

L. Zhang, M. Mahdavi, and R. Jin. Linear convergence with condition number independent access of full gradients. *Advances in Neural Information Processing Systems*, 2013.

J. Zhou, X. Qiu, and X. Huang. A fast accurate two-stage training algorithm for L1-regularized CRFs with heuristic line search strategy. *International Joint Conference on Natural Language Processing*, 2011.

J. Zhu and E. Xing. Conditional Topic Random Fields. In *International Conference on Machine Learning*, 2010.