Practical very large scale CRFs

Thomas Lavergne
LIMSI – CNRS
lavergne@limsi.fr

Olivier Cappé
Télécom ParisTech
LTCI – CNRS
cappe@enst.fr

François Yvon
Université Paris-Sud 11
LIMSI – CNRS
yvon@limsi.fr

Abstract

Conditional Random Fields (CRFs) are a widely-used approach for supervised sequence labelling, notably due to their ability to handle large description spaces and to integrate structural dependency between labels. Even for the simple linear-chain model, taking structure into account implies a number of parameters and a computational effort that grows quadratically with the cardinality of the label set. In this paper, we address the issue of training very large CRFs, containing up to hundreds output labels and several billion features. Efficiency stems here from the sparsity induced by the use of a ℓ^1 penalty term. Based on our own implementation, we compare three recent proposals for implementing this regularization strategy. Our experiments demonstrate that very large CRFs can be trained efficiently and that very large models are able to improve the accuracy, while delivering compact parameter sets.

1 Introduction

Conditional Random Fields (CRFs) (Lafferty et al., 2001; Sutton and McCallum, 2006) constitute a widely-used and effective approach for supervised structure learning tasks involving the mapping between complex objects such as strings and trees. An important property of CRFs is their ability to handle large and redundant feature sets and to integrate structural dependency between output labels. However, even for simple *linear chain* CRFs, the complexity of learning and inference grows quadratically with respect to the number of output labels and so does the number of *structural features*, ie. features testing adjacent pairs of labels. Most empirical studies on CRFs thus either consider tasks with a restricted output space (typically in the order of few dozens of output labels), heuristically reduce the use of features, especially of features that test pairs of adjacent labels[1], and/or propose heuristics to simulate contextual dependencies, via extended tests on the observations (see discussions in, eg., (Punyakanok et al., 2005; Liang et al., 2008)). Limitating the feature set or the number of output labels is however frustrating for many NLP tasks, where the type and number of potentially relevant features are very large. A number of studies have tried to alleviate this problem. Pal et al. (2006) propose to use a "sparse" version of the forward-backward algorithm during training, where sparsity is enforced through beam pruning. Related ideas are discussed by Dietterich et al. (2004); by Cohn (2006), who considers "generalized" feature functions; and by Jeong et al. (2009), who use approximations to simplify the forward-backward recursions. In this paper, we show that the sparsity that is induced by ℓ^1-penalized estimation of CRFs can be used to reduce the total training time, while yielding extremely compact models. The benefits of sparsity are even greater during inference: less features need to be extracted and included in the potential functions, speeding up decoding with a lesser memory footprint. We study and compare three different ways to implement ℓ^1 penalty for CRFs that have been introduced recently: orthant-wise Quasi Newton (Andrew and Gao, 2007), stochastic gradient descent (Tsuruoka et al., 2009) and coordinate descent (Sokolovska et al., 2010), concluding that these methods have complemen-

This work was partly supported by ANR projects CroTaL (ANR-07-MDCO-003) and MGA (ANR-07-BLAN-0311-02).

[1] In CRFsuite (Okazaki, 2007), it is even impossible to jointly test a pair of labels and a test on the observation, bigrams feature are only of the form $f(y_{t-1}, y_t)$.

Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 504–513,
Uppsala, Sweden, 11-16 July 2010.

tary strengths and weaknesses. Based on an efficient implementation of these algorithms, we were able to train very large CRFs containing more than a hundred of output labels and up to several billion features, yielding results that are as good or better than the best reported results for two NLP benchmarks, text phonetization and part-of-speech tagging.

Our contribution is therefore twofold: firstly a detailed analysis of these three algorithms, discussing implementation, convergence and comparing the effect of various speed-ups. This comparison is made fair and reliable thanks to the reimplementation of these techniques in the same software package. Second, the experimental demonstration that using large output label sets is doable and that very large feature sets actually help improve prediction accuracy. In addition, we show how sparsity in *structured feature sets* can be used in incremental training regimes, where long-range features are progressively incorporated in the model insofar as the shorter range features have proven useful.

The rest of the paper is organized as follows: we first recall the basics of CRFs in Section 2, and discuss three ways to train CRFs with a ℓ^1 penalty in Section 3. We then detail several implementation issues that need to be addressed when dealing with massive feature sets in Section 4. Our experiments are reported in Section 5. The main conclusions of this study are drawn in Section 6.

2 Conditional Random Fields

In this section, we recall the basics of Conditional Random Fields (CRFs) (Lafferty et al., 2001; Sutton and McCallum, 2006) and introduce the notations that will be used throughout.

2.1 Basics

CRFs are based on the following model

$$p_\theta(\mathbf{y}|\mathbf{x}) = \frac{1}{Z_\theta(\mathbf{x})} \exp\left\{\sum_{k=1}^{K} \theta_k F_k(\mathbf{x},\mathbf{y})\right\} \quad (1)$$

where $\mathbf{x} = (x_1, \ldots, x_T)$ and $\mathbf{y} = (y_1, \ldots, y_T)$ are, respectively, the input and output sequences[2], and $F_k(\mathbf{x},\mathbf{y})$ is equal to $\sum_{t=1}^{T} f_k(y_{t-1}, y_t, x_t)$, where $\{f_k\}_{1\leq k\leq K}$ is an arbitrary set of feature functions and $\{\theta_k\}_{1\leq k\leq K}$ are the associated parameter values. We denote by Y and X, respectively, the sets in which y_t and x_t take their values. The normalization factor in (1) is defined by

$$Z_\theta(\mathbf{x}) = \sum_{\mathbf{y}\in Y^T} \exp\left\{\sum_{k=1}^{K} \theta_k F_k(\mathbf{x},\mathbf{y})\right\}. \quad (2)$$

The most common choice of feature functions is to use binary tests. In the sequel, we distinguish between two types of feature functions: *unigram features* $f_{y,x}$, associated with parameters $\mu_{y,x}$, and *bigram features* $f_{y',y,x}$, associated with parameters $\lambda_{y',y,x}$. These are defined as

$$f_{y,x}(y_{t-1}, y_t, x_t) = \mathbf{1}(y_t = y, x_t = x)$$
$$f_{y',y,x}(y_{t-1}, y_t, x_t) = \mathbf{1}(y_{t-1} = y', y_t = y, x_t = x)$$

where $\mathbf{1}(\text{cond.})$ is equal to 1 when the condition is verified and to 0 otherwise. In this setting, the number of parameters K is equal to $|Y|^2 \times |X|_{\text{train}}$, where $|\cdot|$ denotes the cardinal and $|X|_{\text{train}}$ refers to the number of configurations of x_t observed during training. Thus, even in moderate size applications, the number of parameters can be very large, mostly due to the introduction of sequential dependencies in the model. This also explains why it is hard to train CRFs with dependencies spanning more than two adjacent labels. Using only unigram features $\{f_{y,x}\}_{(y,x)\in Y\times X}$ results in a model equivalent to a simple bag-of-tokens position-by-position logistic regression model. On the other hand, bigram features $\{f_{y',y,x}\}_{(y,x)\in Y^2\times X}$ are helpful in modelling dependencies between successive labels. The motivations for using simultaneously both types of feature functions are evaluated experimentally in Section 5.

2.2 Parameter Estimation

Given N independent sequences $\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{N}$, where $\mathbf{x}^{(i)}$ and $\mathbf{y}^{(i)}$ contain $T^{(i)}$ symbols, conditional maximum likelihood estimation is based on the minimization, with respect to θ, of the negated conditional log-likelihood of the observations

$$l(\theta) = -\sum_{i=1}^{N} \log p_\theta(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}) \quad (3)$$
$$= \sum_{i=1}^{N} \left\{\log Z_\theta(\mathbf{x}^{(i)}) - \sum_{k=1}^{K} \theta_k F_k(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\right\}$$

This term is usually complemented with an additional regularization term so as to avoid overfitting

[2] Our implementation also includes a special label y_0, that is always observed and marks the beginning of a sequence.

(see Section 3.1 below). The gradient of $l(\theta)$ is

$$\frac{\partial l(\theta)}{\partial \theta_k} = \sum_{i=1}^{N} \sum_{t=1}^{T^{(i)}} \mathrm{E}_{p_\theta(\mathbf{y}|\mathbf{x}^{(i)})}\, f_k(y_{t-1}, y_t, x_t^{(i)}) - \sum_{i=1}^{N} \sum_{t=1}^{T^{(i)}} f_k(y_{t-1}^{(i)}, y_t^{(i)}, x_t^{(i)}) \quad (4)$$

where $\mathrm{E}_{p_\theta(\mathbf{y}|\mathbf{x})}$ denotes the conditional expectation given the observation sequence, i.e.

$$\mathrm{E}_{p_\theta(\mathbf{y}|\mathbf{x})}\, f_k(y_{t-1}, y_t, x_t^{(i)}) = \sum_{(y',y)\in Y^2} f_k(y, y', x_t)\, \mathrm{P}_\theta(y_{t-1} = y', y_t = y|\mathbf{x}) \quad (5)$$

Although $l(\theta)$ is a smooth convex function, its optimum cannot be computed in closed form, and $l(\theta)$ has to be optimized numerically. The computation of its gradient implies to repeatedly compute the conditional expectation in (5) for all input sequences $\mathbf{x}^{(i)}$ and all positions t. The standard approach for computing these expectations is inspired by the forward-backward algorithm for hidden Markov models: using the notations introduced above, the algorithm implies the computation of the forward

$$\begin{cases} \alpha_1(y) = \exp(\mu_{y,x_1} + \lambda_{y_0,y,x_1}) \\ \alpha_{t+1}(y) = \sum_{y'} \alpha_t(y') \exp(\mu_{y,x_{t+1}} + \lambda_{y',y,x_{t+1}}) \end{cases}$$

and backward recursions

$$\begin{cases} \beta_{T_i}(y) = 1 \\ \beta_t(y') = \sum_y \beta_{t+1}(y) \exp(\mu_{y,x_{t+1}} + \lambda_{y',y,x_{t+1}}), \end{cases}$$

for all indices $1 \leq t \leq T$ and all labels $y \in Y$. Then, $Z_\theta(\mathbf{x}) = \sum_y \alpha_T(y)$ and the pairwise probabilities $\mathrm{P}_\theta(y_t = y', y_{t+1} = y|\mathbf{x})$ are given by

$$\alpha_t(y') \exp(\mu_{y,x_{t+1}} + \lambda_{y',y,x_{t+1}}) \beta_{t+1}(y) / Z_\theta(\mathbf{x})$$

These recursions require a number of operations that grows quadratically with $|Y|$.

3 ℓ^1 Regularization in CRFs

3.1 Regularization

The standard approach for parameter estimation in CRFs consists in minimizing the logarithmic loss $l(\theta)$ defined by (3) with an additional ℓ^2 penalty term $\frac{\rho_2}{2}\|\theta\|_2^2$, where ρ_2 is a regularization parameter. The objective function is then a smooth convex function to be minimized over an unconstrained parameter space. Hence, any numerical optimization strategy may be used and practical solutions include limited memory BFGS (L-BFGS) (Liu and Nocedal, 1989), which is used in the popular CRF++ (Kudo, 2005) and CRFsuite (Okazaki, 2007) packages; conjugate gradient (Nocedal and Wright, 2006) and Stochastic Gradient Descent (SGD) (Bottou, 2004; Vishwanathan et al., 2006), used in CRFsgd (Bottou, 2007). The only caveat is to avoid numerical optimizers that require the full Hessian matrix (e.g., Newton's algorithm) due to the size of the parameter vector in usual applications of CRFs.

The most significant alternative to ℓ^2 regularization is to use a ℓ^1 penalty term $\rho_1\|\theta\|_1$: such regularizers are able to yield sparse parameter vectors in which many component have been zeroed (Tibshirani, 1996). Using a ℓ^1 penalty term thus implicitly performs feature selection, where ρ_1 controls the amount of regularization and the number of extracted features. In the following, we will jointly use both penalty terms, yielding the so-called elastic net penalty (Zhou and Hastie, 2005) which corresponds to the objective function

$$l(\theta) + \rho_1\|\theta\|_1 + \frac{\rho_2}{2}\|\theta\|_2^2 \quad (6)$$

The use of both penalty terms makes it possible to control the number of non zero coefficients and to avoid the numerical problems that might occur in large dimensional parameter settings (see also (Chen, 2009)). However, the introduction of a ℓ^1 penalty term makes the optimization of (6) more problematic, as the objective function is no longer differentiable in 0. Various strategies have been proposed to handle this difficulty. We will only consider here exact approaches and will not discuss heuristic strategies such as grafting (Perkins et al., 2003; Riezler and Vasserman, 2004).

3.2 Quasi Newton Methods

To deal with ℓ^1 penalties, a simple idea is that of (Kazama and Tsujii, 2003), originally introduced for maxent models. It amounts to reparameterizing θ_k as $\theta_k = \theta_k^+ - \theta_k^-$, where θ_k^+ and θ_k^- are positive. The ℓ^1 penalty thus becomes $\rho_1(\theta^+ - \theta^-)$. In this formulation, the objective function recovers its smoothness and can be optimized with conventional algorithms, subject to domain constraints. Optimization is straightforward, but the number of parameters is doubled and convergence is slow

(Andrew and Gao, 2007): the procedure lacks a mechanism for zeroing out useless parameters.

A more efficient strategy is the orthant-wise quasi-Newton (OWL-QN) algorithm introduced in (Andrew and Gao, 2007). The method is based on the observation that the ℓ^1 norm is differentiable when restricted to a set of points in which each coordinate never changes its sign (an "orthant"), and that its second derivative is then zero, meaning that the ℓ^1 penalty does not change the Hessian of the objective on each orthant. An OWL-QN update then simply consists in (i) computing the Newton update in a well-chosen orthant; (ii) performing the update, which might cause some component of the parameter vector to change sign; and (iii) projecting back the parameter value onto the initial orthant, thereby zeroing out those components. In (Gao et al., 2007), the authors show that OWL-QN is faster than the algorithm proposed by Kazama and Tsujii (2003) and can perform model selection even in very high-dimensional problems, with no loss of performance compared to the use of ℓ^2 penalty terms.

3.3 Stochastic Gradient Descent

Stochastic gradient (SGD) approaches update the parameter vector based on an crude approximation of the gradient (4), where the computation of expectations only includes a small batch of observations. SGD updates have the following form

$$\theta_k \leftarrow \theta_k + \eta \frac{\partial l(\theta)}{\partial \theta_k}, \tag{7}$$

where η is the learning rate. In (Tsuruoka et al., 2009), various ways of adapting this update to ℓ^1-penalized likelihood functions are discussed. Two effective ideas are proposed: (i) only update parameters that correspond to active features in the current observation, (ii) keep track of the cumulated penalty z_k that θ_k should have received, had the gradient been computed exactly, and use this value to "clip" the parameter value. This is implemented by patching the update (7) as follows

$$\begin{cases} \textbf{if } (\theta_k > 0)\ \theta_k \leftarrow \max(0, \theta_k - z_k) \\ \textbf{else if } (\theta_k < 0)\ \theta_k \leftarrow \min(0, \theta_k - z_k) \end{cases} \tag{8}$$

Based on a study of three NLP benchmarks, the authors of (Tsuruoka et al., 2009) claim this approach to be much faster than the orthant-wise approach and yet to yield very comparable performance, while selecting slightly larger feature sets.

3.4 Block Coordinate Descent

The coordinate descent approach of Dudík et al. (2004) and Friedman et al. (2008) uses the fact that optimizing a mono-dimensional quadratic function augmented with a ℓ^1 penalty can be performed analytically. For arbitrary functions, this idea can be adapted by considering quadratic approximations of the objective around the current value $\bar{\theta}$

$$l_{k,\bar{\theta}}(\theta_k) = \frac{\partial l(\bar{\theta})}{\partial \theta_k}(\theta_k - \bar{\theta}_k) + \frac{1}{2}\frac{\partial^2 l(\bar{\theta})}{\partial \theta_k^2}(\theta_k - \bar{\theta}_k)^2 + \rho_1|\theta_k| + \frac{\rho_2}{2}\theta_k^2 + C^{st} \tag{9}$$

The minimizer of the approximation (9) is simply

$$\theta_k = \frac{s\left\{\frac{\partial^2 l(\bar{\theta})}{\partial \theta_k^2}\bar{\theta}_k - \frac{\partial l(\bar{\theta})}{\partial \theta_k}, \rho_1\right\}}{\frac{\partial^2 l(\bar{\theta})}{\partial \theta_k^2} + \rho_2} \tag{10}$$

where s is the soft-thresholding function

$$s(z, \rho) = \begin{cases} z - \rho & \text{if } z > \rho \\ z + \rho & \text{if } z < -\rho \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

Coordinate descent is ported to CRFs in (Sokolovska et al., 2010). Making this scheme practical requires a number of adaptations, including (i) approximating the second order term in (10), (ii) performing updates *in block*, where a block contains the $|Y| \times |Y + 1|$ features $\nu_{y',y,x}$ and $\lambda_{y,x}$ for a fixed test x on the observation sequence and (iii) approximating the Hessian for a block by its diagonal terms. (ii) is specially critical, as repeatedly cycling over individual features to perform the update (10) is only possible with restricted sets of features. The *block update* schemes uses the fact that all features within a block appear in the same set of sequences, which means that most of the computations needed to perform theses updates can be shared within the block. One advantage of the resulting algorithm, termed BCD in the following, is that the update of θ_k only involves carrying out the forward-backward recursions for the set of sequences that contain symbols x such that at least one $\{f_k(y', y, x)\}_{(y,y') \in Y^2}$ is non null, which can be much smaller than the whole training set.

4 Implementation Issues

Efficiently processing very-large feature and observation sets requires to pay attention to many implementation details. In this section, we present several optimizations devised to speed up training.

4.1 Sparse Forward-Backward Recursions

For all algorithms, the computation time is dominated by the evaluations of the gradient: our implementation takes advantage of the sparsity to accelerate these computations. Assume the set of bigram features $\{\lambda_{y',y,x_{t+1}}\}_{(y',y)\in Y^2}$ is sparse with only $r(x_{t+1}) \ll |Y|^2$ non null values and define the $|Y| \times |Y|$ sparse matrix

$$M_t(y', y) = \exp(\lambda_{y',y,x_t}) - 1.$$

Using M, the forward-backward recursions are

$$\alpha_t(y) = \sum_{y'} u_{t-1}(y') + \sum_{y'} u_{t-1}(y') M_t(y', y)$$
$$\beta_t(y') = \sum_{y} v_{t+1}(y) + \sum_{y} M_{t+1}(y', y) v_{t+1}(y)$$

with $u_{t-1}(y) = \exp(\mu_{y,x_t})\alpha_{t-1}(y)$ and $v_{t+1}(y) = \exp(\mu_{y,x_{t+1}})\beta_{t+1}(y)$. (Sokolovska et al., 2010) explains how computational savings can be obtained using the fact that the vector/matrix products in the recursions above only involve the sparse matrix $M_{t+1}(y', y)$. They can thus be computed with exactly $r(x_{t+1})$ multiplications instead of $|Y|^2$. The same idea can be used when the set $\{\mu_{y,x_{t+1}}\}_{y\in Y}$ of unigram features is sparse. Using this implementation, the complexity of the forward-backward procedure for $\mathbf{x}^{(i)}$ can be made proportional to the average number of active features per position, which can be much smaller than the number of potentially active features.

For BCD, forward-backward can even be made slightly faster. When computing the gradient wrt. features $\lambda_{y,x}$ and $\mu_{y',y,x}$ (for all the values of y and y') for sequence $\mathbf{x}^{(i)}$, assuming that x only occurs once in $\mathbf{x}^{(i)}$ at position t, all that is needed is $\alpha'_t(y), \forall t' \leq t$ and $\beta'_t(y), \forall t' \geq t$. $Z_\theta(\mathbf{x})$ is then recovered as $\sum_y \alpha_t(y)\beta_t(y)$. Forward-backward recursions can thus be truncated: in our experiments, this divided the computational cost by 1,8 on average.

Note finally that forward-backward is performed on a per-observation basis and is easily parallelized (see also (Mann et al., 2009) for more powerful ways to distribute the computation when dealing with very large datasets). In our implementation, it is distributed on all available cores, resulting in significant speed-ups for OWL-QN and L-BFGS; for BCD the gain is less acute, as parallelization only helps when updating the parameters for a block of features that are occur in many sequences; for SGD, with batches of size one, this parallelization policy is useless.

4.2 Scaling

Most existing implementations of CRFs, eg. CRF++ and CRFsgd perform the forward-backward recursions in the log-domain, which guarantees that numerical over/underflows are avoided no matter the length $T^{(i)}$ of the sequence. It is however very inefficient from an implementation point of view, due to the repeated calls to the $\exp()$ and $\log()$ functions. As an alternative way of avoiding numerical problems, our implementation, like crfSuite's, resorts to "scaling", a solution commonly used for HMMs. Scaling amounts to normalizing the values of α_t and β_t to one, making sure to keep track of the cumulated normalization factors so as to compute $Z_\theta(\mathbf{x})$ and the conditional expectations $\mathrm{E}_{p_\theta(\mathbf{y}|\mathbf{x})}$. Also note that in our implementation, all the computations of $\exp(x)$ are vectorized, which provides an additional speed up of about 20%.

4.3 Optimization in Large Parameter Spaces

Processing very large feature vectors, up to billions of components, is problematic in many ways. Sparsity has been used here to speed up forward-backward, but we have made no attempt to accelerate the computation of the OWL-QN updates, which are linear in the size of the parameter vector. Of the three algorithms, BCD is the most affected by increases in the number of features, or more precisely, in the number of *features blocks*, where one block correspond to a specific test of the observation. In the worst case scenario, each block may require to visit all the training instances, yielding terrible computational wastes. In practice though, most blocks only require to process a small fraction of the training set, and the actual complexity depends on the average number of blocks per observations. Various strategies have been tried to further accelerate BCD, such as processing blocks that only visit one observation in parallel and updating simultaneously all the blocks that visit all the training instances, leading to a small speed-up on the POS-tagging task.

Working with billions of features finally requires to worry also about memory usage. In this respect, BCD is the most efficient, as it only requires to store one K-dimensional vector for the parameter itself. SGD requires two such vectors, one for the parameter and one for storing the z_k (see Eq. (8)). In comparison, OWL-QN requires much more memory, due to the internals of the update routines, which require several histories of the parameter vector and of its gradient. Typically, our implementation necessitates in the order of a dozen K-dimensional vectors. Parallelization only makes things worse, as each core will also need to maintain its own copy of the gradient.

5 Experiments

Our experiments use two standard NLP tasks, phonetization and part-of-speech tagging, chosen here to illustrate two very different situations, and to allow for comparison with results reported elsewhere in the literature. Unless otherwise mentioned, the experiments use the same protocol: 10 fold cross validation, where eight folds are used for training, one for development, and one for testing. Results are reported in terms of phoneme error rates or tag error rates *on the test set*.

Comparing run-times can be a tricky matter, especially when different software packages are involved. As discussed above, the observed run-times depend on many small implementation details. As the three algorithms share as much code as possible, we believe the comparison reported hereafter to be fair and reliable. All experiments were performed on a server with 64G of memory and two Xeon processors with 4 cores at 2.27 Ghz. For comparison, all measures of run-times include the cumulated activity of all cores and give very pessimistic estimates of the wall time, which can be up to 7 times smaller. For OWL-QN, we use 5 past values of the gradient to approximate the inverse of the Hessian matrix: increasing this value had no effect on accuracy or convergence and was detrimental to speed; for SGD, the learning rate parameter was tuned manually.

Note that we have not spent much time optimizing the values of ρ_1 and ρ_2. Based on a pilot study on Nettalk, we found that taking $\rho_1 = .5$ and ρ_2 in the order of 10^{-5} to yield nearly optimal performance, and have used these values throughout.

5.1 Tasks and Settings

5.1.1 Nettalk

Our first benchmark is the word phonetization task, using the Nettalk dictionary (Sejnowski and Rosenberg, 1987). This dataset contains approximately 20,000 English word forms, their pronunciation, plus some prosodic information (stress markers for vowels, syllabic parsing for consonants). Grapheme and phoneme strings are aligned at the character level, thanks to the use of a "null sound" in the latter string when it is shorter than the former; likewise, each prosodic mark is aligned with the corresponding letter. We have derived two test conditions from this database. The first one is standard and aims at predicting the pronunciation information only. In this setting, the set of observations (X) contains 26 graphemes, and the output label set contains $|Y| = 51$ phonemes.

The second condition aims at *jointly predicting phonemic and prosodic information*[3]. The reasons for designing this new condition are twofold: firstly, it yields a large set of composite labels ($|Y| = 114$) and makes the problem computationally challenging. Second, it allows to quantify how much the information provided by the prosodic marks help predict the phonemic labels. Both information are quite correlated, as the stress mark and the syllable openness, for instance, greatly influence the realization of some archi-phonemes.

The features used in Nettalk experiments take the form $f_{y,w}$ (unigram) and $f_{y',y,w}$ (bigram), where w is a n-gram of letters. The n-grm feature sets ($n = \{1, 3, 5, 7\}$) includes all features testing embedded windows of k letters, for all $0 \leq k \leq n$; the n-grm- setting is similar, but only includes the window of length n; in the n-grm+ setting, we add features for odd-size windows; in the n-grm++ setting, we add all sequences of letters up to size n occurring in current window. For instance, the active bigram features at position $t = 2$ in the sequence **x**='*lemma*' are as follows: the 3-grm feature set contains $f_{y,y'}$, $f_{y,y',e}$ and $f_{y',y,lem}$; only the latter appears in the 3-grm- setting. In the 3-grm+ feature set, we also have $f_{y',y,le}$ and $f_{y',y,em}$. The 3-grm++ feature set additionally includes $f_{y',y,l}$ and $f_{y',y,m}$. The number of features ranges from 360 thousands (1-grm setting) to 1.6 billion (7-grm).

[3]Given the design of the Nettalk dictionary, this experiment required to modify the original database so as to reassign prosodic marks to phonemes, rather than to letters.

Features	With		Without	
Nettalk				
3-grm	10.74%	$14.3M$	14.59%	$0.3M$
5-grm	8.48%	$132.5M$	11.54%	$2.5M$
POS tagging				
base	2.91%	$436.7M$	3.47%	$70.2M$

Table 1: Features jointly testing label pairs and the observation are useful (error rates and features counts.)

	ℓ^2	ℓ^1-sparse	ℓ^1	% zero
1-grm	84min	41min	57min	44.6%
3-grm-	65min	16min	44min	99.6%
3-grm	72min	48min	58min	19.9%

Table 2: Sparse vs standard forward-backward (training times and percentages of sparsity of M)

5.1.2 Part-of-Speech Tagging

Our second benchmark is a part-of-speech (POS) tagging task using the PennTreeBank corpus (Marcus et al., 1993), which provides us with a quite different condition. For this task, the number of labels is smaller ($|Y| = 45$) than for Nettalk, and the set of observations is much larger ($|X| = 43207$). This benchmark, which has been used in many studies, allows for direct comparisons with other published work. We thus use a standard experimental set-up, where sections 0-18 of the Wall Street Journal are used for training, sections 19-21 for development, and sections 22-24 for testing.

Features are also standard and follow the design of (Suzuki and Isozaki, 2008) and test the current words (as written and lowercased), prefixes and suffixes up to length 4, and typographical characteristics (case, etc.) of the words. Our baseline feature set also contains tests on individual and pairs of words in a window of 5 words.

5.2 Using Large Feature Sets

The first important issue is to assess the benefits of using large feature sets, notably including features testing both a bigram of labels and an observation. Table 1 compares the results obtained with and without these features for various setting (using OWL-QN to perform the optimization), suggesting that for the tasks at hand, these features are actually helping.

	ℓ^2	ℓ^1	Elastic-net
1-grm	17.81%	17.86%	17.79%
3-grm	10.62%	10.74%	10.70%
5-grm	8.50%	8.45%	8.48%

Table 3: Error rates of the three regularizers on the Nettalk task.

5.3 Speed, Sparsity, Convergence

The training speed depends of two main factors: the number of iterations needed to achieve convergence and the computational cost of one iteration. In this section, we analyze and compare the runtime efficiency of the three optimizers.

5.3.1 Convergence

As far as convergence is concerned, the two forms of regularization (ℓ^2 and ℓ^1) yield the same performance (see Table 3), and the three algorithms exhibit more or less the same behavior. They quickly reach an acceptable set of active parameters, which is often several orders of magnitude smaller than the whole parameter set (see results below in Table 4 and 5). Full convergence, reflected by a stabilization of the objective function, is however not so easily achieved. We have often observed a slow, yet steady, decrease of the log-loss, accompanied with a diminution of the number of active features as the number of iterations increases. Based on this observation, we have chosen to stop all algorithms based on their performance on an independent development set, allowing a fair comparison of the overall training time; for OWL-QN, it allowed to divide the total training time by almost 2.

It has finally often been found useful to fine tune the non-zero parameters by running a final handful of L-BFGS iterations using only a small ℓ^2 penalty; at this stage, all the other features are removed from the model. This had a small impact BCD and SGD's performance and allowed them to catch up with OWL-QN's performance.

5.3.2 Sparsity and the Forward-Backward

As explained in section 4.1, the forward-backward algorithm can be written so as to use the sparsity of the matrix $M_{y,y',x}$. To evaluate the resulting speed-up, we ran a series of experiments using Nettalk (see Table 2). In this table, the 3-grm- setting corresponds to maximum sparsity for M, and training with the sparse algorithm is three times faster than with the non-sparse version. Throwing

	Method	Iter.	# Feat.	Error	Time
OWL-QN	1-grm	63.4	4684	17.79%	11min
	7-grm	140.2	38214	8.12%	1h02min
	5-grm+	141.0	43429	7.89%	1h37min
SGD	1-grm	21.4	3540	18.21%	9min
	5-grm+	28.5	34319	8.01%	45min
BCD	1-grm	28.2	5017	18.27%	27min
	7-grm	9.2	3692	8.21%	1h22min
	5-grm+	8.7	47675	7.91%	2h18min

Table 4: Performance on Nettalk

	Method	Error	Time
SGD	5-grm	14.71% / 8.11%	55min
	5-grm+	13.91% / 7.51%	2h45min
BCD	5-grm	14.57% / 8.06%	2h46min
	7-grm	14.12% / 7.86%	3h02min
	5-grm+	13.85% / 7.47%	7h14min
	5-grm++	13.69% / 7.36%	16h03min

Table 5: Performance on Nettalk+prosody. Error is given for both joint labels and phonemic labels.

in more features has the effect of making M much more dense, mitigating the benefits of the sparse recursions. Nevertheless, even for very large feature sets, the percentage of zeros in M averages 20% to 30%, and the sparse version remains 10 to 20% faster than the non-sparse one. Note that the non-sparse version is faster with a ℓ^1 penalty term than with only the ℓ^2 term: this is because $\exp(0)$ is faster to evaluate than $\exp(x)$ when $x \neq 0$.

5.3.3 Training Speed and Test Accuracy

Table 4 displays the results achieved on the Nettalk task. The three algorithms yield very comparable accuracy results, and deliver compact models: for the 5-gram+ setting, only 50,000 out of 250 million features are selected. SGD is the fastest of the three, up to twice as fast as OWL-QN and BCD depending on the feature set. The performance it achieves are consistently slightly worst than the other optimizers, and only catch up when the parameters are fine-tuned (see above). There are not so many comparisons for Nettalk with CRFs, due to the size of the label set. Our results compare favorably with those reported in (Pal et al., 2006), where the accuracy attains 91.7% using 19075 examples for training and 934 for testing, and with those in (Jeong et al., 2009) (88.4% accuracy with 18,000 (2,000) training (test) instances). Table 5 gives the results obtained for the larger Nettalk+prosody task. Here, we only report the results obtained with SGD and BCD. For OWL-QN, the largest model we could handle was the 3-grm model, which contained 69 million features, and took 48min to train. Here again, performance steadily increase with the number of features, showing the benefits of large-scale models. We lack comparisons for this task, which seems considerably harder than the sole phonetization task, and all systems seem to plateau around 13.5% accuracy. Interestingly, simultaneously predicting the phoneme and its prosodic markers allows to improve the accuracy on the prediction of phonemes, which improves of almost a half point as compared to the best Nettalk system.

For the POS tagging task, BCD appears to be unpractically slower to train than the others approaches (SGD takes about 40min to train, OWL-QN about 1 hour) due the simultaneous increase in the sequence length and in the number of observations. As a result, one iteration of BCD typically requires to repeatedly process over and over the same sequences: on average, each sequence is visited 380 times when we use the baseline feature set. This technique should reserved for tasks where the number of blocks is small, or, as below, when memory usage is an issue.

5.4 Structured Feature Sets

In many tasks, the ambiguity of tokens can be reduced by looking up increasingly large windows of local context. This strategy however quickly runs into a combinatorial increase of the number of features. A side note of the Nettalk experiments is that when using embedded features, the active feature set tends to reflect this hierarchical organization. This means that when a feature testing a n-gram is active, in most cases, the features for all embedded k-grams are also selected.

Based on this observation, we have designed an incremental training strategy for the POS tagging task, where more specific features are progressively incorporated into the model if the corresponding less specific feature is active. This experiment used BCD, which is the most memory efficient algorithm. The first iteration only includes tests on the current word. During the second iteration, we add tests on bigram of words, on suffixes and prefixes up to length 4. After four iterations, we throw in features testing word trigrams, subject to the corresponding unigram block being active. After 6 iterations, we finally augment the

model with windows of length 5, subject to the corresponding trigram being active. After 10 iterations, the model contains about 4 billion features, out of which 400,000 are active. It achieves an error rate of 2.63% (resp. 2.78%) on the development (resp. test) data, which compares favorably with some of the best results for this task (for instance (Toutanova et al., 2003; Shen et al., 2007; Suzuki and Isozaki, 2008)).

6 Conclusion and Perspectives

In this paper, we have discussed various ways to train extremely large CRFs with a ℓ^1 penalty term and compared experimentally the results obtained, both in terms of training speed and of accuracy. The algorithms studied in this paper have complementary strength and weaknesses: OWL-QN is probably the method of choice in small or moderate size applications while BCD is most efficient when using very large feature sets combined with limited-size observation alphabets; SGD complemented with fine tuning appears to be the preferred choice in most large-scale applications. Our analysis demonstrate that training large-scale sparse models can be done efficiently and allows to improve over the performance of smaller models. The CRF package developed in the course of this study implements many algorithmic optimizations and allows to design innovative training strategies, such as the one presented in section 5.4. This package is released as open-source software and is available at `http://wapiti.limsi.fr`.

In the future, we intend to study how sparsity can be used to speed-up training in the face of more complex dependency patterns (such as higher-order CRFs or hierarchical dependency structures (Rozenknop, 2002; Finkel et al., 2008). From a performance point of view, it might also be interesting to combine the use of large-scale feature sets with other recent improvements such as the use of semi-supervised learning techniques (Suzuki and Isozaki, 2008) or variable-length dependencies (Qian et al., 2009).

References

Galen Andrew and Jianfeng Gao. 2007. Scalable training of l1-regularized log-linear models. In *Proceedings of the International Conference on Machine Learning*, pages 33–40, Corvalis, Oregon.

Léon Bottou. 2004. Stochastic learning. In Olivier Bousquet and Ulrike von Luxburg, editors, *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146–168. Springer Verlag, Berlin.

Léon Bottou. 2007. Stochastic gradient descent (sgd) implementation. http://leon.bottou.org/projects/sgd.

Stanley Chen. 2009. Performance prediction for exponential language models. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 450–458, Boulder, Colorado, June.

Trevor Cohn. 2006. Efficient inference in large conditional random fields. In *Proceedings of the 17th European Conference on Machine Learning*, pages 606–613, Berlin, September.

Thomas G. Dietterich, Adam Ashenfelter, and Yaroslav Bulatov. 2004. Training conditional random fields via gradient tree boosting. In *Proceedings of the International Conference on Machine Learning*, Banff, Canada.

Miroslav Dudík, Steven J. Phillips, and Robert E. Schapire. 2004. Performance guarantees for regularized maximum entropy density estimation. In John Shawe-Taylor and Yoram Singer, editors, *Proceedings of the 17th annual Conference on Learning Theory*, volume 3120 of *Lecture Notes in Computer Science*, pages 472–486. Springer.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 959–967, Columbus, Ohio.

Jerome Friedman, Trevor Hastie, and Rob Tibshirani. 2008. Regularization paths for generalized linear models via coordinate descent. Technical report, Department of Statistics, Stanford University.

Jianfeng Gao, Galen Andrew, Mark Johnson, and Kristina Toutanova. 2007. A comparative study of parameter estimation methods for statistical natural language processing. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 824–831, Prague, Czech republic.

Minwoo Jeong, Chin-Yew Lin, and Gary Geunbae Lee. 2009. Efficient inference of crfs for large-scale natural language data. In *Proceedings of the Joint Conference of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing*, pages 281–284, Suntec, Singapore.

Jun'ichi Kazama and Jun'ichi Tsujii. 2003. Evaluation and extension of maximum entropy models with inequality constraints. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pages 137–144.

Taku Kudo. 2005. CRF++: Yet another CRF toolkit. http://crfpp.sourceforge.net/.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA.

Percy Liang, Hal Daumé, III, and Dan Klein. 2008. Structure compilation: trading structure for features. In *Proceedings of the 25th international conference on Machine learning*, pages 592–599.

Dong C. Liu and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45:503–528.

Gideon Mann, Ryan McDonald, Mehryar Mohri, Nathan Silberman, and Dan Walker. 2009. Efficient large-scale distributed training of conditional maximum entropy models. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A.Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1231–1239.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2):313–330.

Jorge Nocedal and Stephen Wright. 2006. *Numerical Optimization*. Springer.

Naoaki Okazaki. 2007. CRFsuite: A fast implementation of conditional random fields (CRFs). http://www.chokkan.org/software/crfsuite/.

Chris Pal, Charles Sutton, and Andrew McCallum. 2006. Sparse forward-backward using minimum divergence beams for fast training of conditional random fields. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, Toulouse, France.

Simon Perkins, Kevin Lacker, and James Theiler. 2003. Grafting: Fast, incremental feature selection by gradient descent in function space. *Journal of Machine Learning Research*, 3:1333–1356.

Vasin Punyakanok, Dan Roth, Wen tau Yih, and Dav Zimak. 2005. Learning and inference over constrained output. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1124–1129.

Xian Qian, Xiaoqian Jiang, Qi Zhang, Xuanjing Huang, and Lide Wu. 2009. Sparse higher order conditional random fields for improved sequence labeling. In *Proceedings of the Annual International Conference on Machine Learning*, pages 849–856.

Stefan Riezler and Alexander Vasserman. 2004. Incremental feature selection and l1 regularization for relaxed maximum-entropy modeling. In Dekang Lin and Dekai Wu, editors, *Proceedings of the conference on Empirical Methods in Natural Language Processing*, pages 174–181, Barcelona, Spain, July.

Antoine Rozenknop. 2002. *Modèles syntaxiques probabilistes non-génératifs*. Ph.D. thesis, Dpt. d'informatique, École Polytechnique Fédérale de Lausanne.

Terrence J. Sejnowski and Charles R. Rosenberg. 1987. Parallel networks that learn to pronounce english text. *Complex Systems*, 1.

Libin Shen, Giorgio Satta, and Aravind Joshi. 2007. Guided learning for bidirectional sequence classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 760–767, Prague, Czech Republic.

Nataliya Sokolovska, Thomas Lavergne, Olivier Cappé, and François Yvon. 2010. Efficient learning of sparse conditional random fields for supervised sequence labelling. *IEEE Selected Topics in Signal Processing*.

Charles Sutton and Andrew McCallum. 2006. An introduction to conditional random fields for relational learning. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*, Cambridge, MA. The MIT Press.

Jun Suzuki and Hideki Isozaki. 2008. Semi-supervised sequential labeling and segmentation using gigaword scale unlabeled data. In *Proceedings of the Conference of the Association for Computational Linguistics on Human Language Technology*, pages 665–673, Columbus, Ohio.

Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *J.R.Statist.Soc.B*, 58(1):267–288.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 173–180.

Yoshimasa Tsuruoka, Jun'ichi Tsujii, and Sophia Ananiadou. 2009. Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty. In *Proceedings of the Joint Conference of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing*, pages 477–485, Suntec, Singapore.

S. V. N. Vishwanathan, Nicol N. Schraudolph, Mark Schmidt, and Kevin Murphy. 2006. Accelerated training of conditional random fields with stochastic gradient methods. In *Proceedings of the 23th International Conference on Machine Learning*, pages 969–976. ACM Press, New York, NY, USA.

Hui Zhou and Trevor Hastie. 2005. Regularization and variable selection via the elastic net. *J. Royal. Stat. Soc. B.*, 67(2):301–320.