Accelerated Training of Maximum Margin Markov Models for Sequence Labeling: A Case Study of NP Chunking[*]

Xiaofeng YU **Wai LAM**

Information Systems Laboratory
Department of Systems Engineering & Engineering Management
The Chinese University of Hong Kong
{xfyu,wlam}@se.cuhk.edu.hk

Abstract

We present the first known empirical results on sequence labeling based on maximum margin Markov networks (M^3N), which incorporate both kernel methods to efficiently deal with high-dimensional feature spaces, and probabilistic graphical models to capture correlations in structured data. We provide an efficient algorithm, the stochastic gradient descent (SGD), to speedup the training procedure of M^3N. Using official dataset for noun phrase (NP) chunking as a case study, the resulting optimizer converges to the same quality of solution over an order of magnitude faster than the structured sequential minimal optimization (structured SMO). Our model compares favorably with current state-of-the-art sequence labeling approaches. More importantly, our model can be easily applied to other sequence labeling tasks.

1 Introduction

The problem of annotating or labeling observation sequences arises in many applications across a variety of scientific disciplines, most prominently in natural language processing, speech recognition, information extraction, and bioinformatics. Recently, the predominant formalism for modeling and predicting label sequences has been based on discriminative graphical models and variants.

Among such models, maximum margin Markov networks (M^3N) and variants (Taskar *et al.* (2003); Taskar (2004); Taskar *et al.* (2005)) have recently gained popularity in the machine learning community. While the M^3N framework makes extensive use of many theoretical results available for Markov networks, it largely dispenses with the probabilistic interpretation. M^3N thus combines the advantages of both worlds, the possibility to have a concise model of the relationships present in the data via log-linear Markov networks over a set of label variables and the highly accurate predictions based on maximum margin estimation of the model parameters.

Traditionally, M^3N can be trained using the structured sequential minimal optimization (structured SMO), a coordinate descent method for solving quadratic programming (QP) problems (Taskar *et al.*, 2003). Clearly, however, the polynomial number of constraints in the QP problem associated with the M^3N can still be very large, making the structured SMO algorithm slow to converge over the training data. This currently limits the scalability and applicability of M^3N to large-scale real world problems.

Stochastic gradient methods (e.g., Lecun *et al.* (1998); Bottou (2004)), on the other hand, are online and scale sub-linearly with the amount of training data, making them very attractive for large-scale datasets. In stochastic (or online) gradient descent (SGD), the true gradient is approximated by the gradient of the cost function only evaluated on a single training example. The parameters are then adjusted by an amount proportional to this approximate gradient. Therefore, the parameters of the model are updated after each training example. For large-scale datasets, online gradient descent can be much faster than standard (or batch) gradient descent.

In this paper, we marry the above two techniques and show how SGD can be used to significantly accelerate the training of M^3N. And we

[*]The work described in this paper is substantially supported by grants from the Research Grant Council of the Hong Kong Special Administrative Region, China (Project No: CUHK4128/07) and the Direct Grant of the Faculty of Engineering, CUHK (Project Codes: 2050442 and 2050476). This work is also affiliated with the Microsoft-CUHK Joint Laboratory for Human-centric Computing and Interface Technologies.

then apply our model to the well-established sequence labeling task: noun phrase (NP) chunking. Experimental results show the validity and effectiveness of our approach. We now summarize the primary contributions of this paper as follows:

- We exploit M^3N to NP chunking on the standard evaluation dataset, achieving favorable performance against recent top-performing systems. The M^3N framework allows arbitrary features of observation sequence, as well as the important benefits of kernels. To the best of our knowledge, this is the first known empirical study on NP chunking using M^3N in the NLP community.

- We provide the efficient SGD algorithm to accelerate the training procedure of M^3N, and experimental results show that it converges over an order of magnitude faster than the structured SMO without sacrificing performance.

- Our model is easily extendable to other sequence labeling tasks, such as part-of-speech tagging and named entity recognition. Based on the promising results on NP chunking, we believe that our model will significantly further the applicability of margin-based approaches to large-scale sequence labeling tasks.

2 Maximum Margin Markov Networks for Sequence Labeling

In sequence labeling, the output is a sequence of labels $\mathbf{y} = (y_1, \ldots, y_T)$ which corresponds to an observation sequence $\mathbf{x} = (x_1, \ldots, x_T)$. Suppose each individual label can take values from set Σ, then the problem can be considered as a multiclass classification problem with $|\Sigma|^T$ different classes.

In M^3N, a pairwise Markov network is defined as a graph $\mathcal{G} = (Y, E)$. Each edge $(i, j) \in E$ is associated with a potential function

$$\begin{aligned}\psi_{ij}(\mathbf{x}, y_i, y_j) &= \exp(\sum_{k=1}^{l} w_k \phi_k(\mathbf{x}, y_i, y_j)) \\ &= \exp(\mathbf{w}^\top \phi(\mathbf{x}, y_i, y_j)) \qquad (1)\end{aligned}$$

where $\phi(\mathbf{x}, y_i, y_j)$ is a pairwise basis function. All edges in the graph denote the same type of interaction, so that we can define a feature map $\phi_k(\mathbf{x}, \mathbf{y}) = \sum_{(i,j)\in E} \phi_k(\mathbf{x}, y_i, y_j)$. The network encodes the following conditional probability distribution (Taskar *et al.*, 2003):

$$P(\mathbf{y}|\mathbf{x}) \propto \prod_{(i,j)\in E} \psi_{ij}(\mathbf{x}, y_i, y_j) = \exp(\mathbf{w}^\top \phi(\mathbf{x}, \mathbf{y})) \qquad (2)$$

where $\phi(\mathbf{x}, \mathbf{y}) = [\phi_1 \phi_2 \ldots \phi_{|\Sigma|} \phi_{trans}]^\top$ is used to learn a weight vector $\mathbf{w}$. $\phi_k = \sum_{i=1}^{n} \phi_i(x) \mathcal{I}(y_i = k), \forall k \in \{1, 2, \ldots, |\Sigma|\}$ and $\phi_{trans} = [c_{11} c_{12} \ldots c_{TT}]^\top$ where c_{ij} is the number of observed transitions from the i^{th} alphabet to the j^{th} alphabet in Σ.

Similar to SVMs (Vapnik, 1995), M^3N tries to find a projection to maximize the margin γ. On the other hand, M^3N also attempts to minimize $\|\mathbf{w}\|$ to minimize the generalization error. Suppose $\Delta \mathbf{t}_\mathbf{x}(\mathbf{y}) = \sum_{i=1}^{n} \Delta \mathbf{t}_\mathbf{x}(y_i) = \sum_{i=1}^{n} I(y_i \neq (\mathbf{t}(\mathbf{x}))_i)$ where $\mathbf{t}((\mathbf{x}))_i$ is the true label of the i^{th} sequence x_i, and $\Delta \phi_\mathbf{x}(\mathbf{y}) = \phi(\mathbf{x}, \mathbf{t}(\mathbf{x})) - \phi(\mathbf{x}, \mathbf{y})$ where $\mathbf{t}(\mathbf{x})$ is the true label of the observation sequence $\mathbf{x}$. We can get a quadratic program (QP) using a standard transformation to eliminate γ as follows:

$$\begin{aligned}\min \quad & \frac{1}{2}\|\mathbf{w}\|^2; \qquad (3)\\ \text{s.t.} \quad & \mathbf{w}^\top \Delta\phi_\mathbf{x}(\mathbf{y}) \geq \Delta \mathbf{t}_\mathbf{x}(\mathbf{y}), \forall \mathbf{x} \in S, \forall \mathbf{y} \in \Sigma.\end{aligned}$$

However, the sequence data is often not separable by the defined hyperplane. In such cases, we can introduce slack variables $\xi_\mathbf{x}$ which are guaranteed to be non-negative to allow some constraints. Thus the complete primal form of the optimization problem can be formulated by:

$$\begin{aligned}\min \quad & \frac{1}{2}\|\mathbf{w}\|^2 + \mathcal{C}\sum_{\mathbf{x}} \xi_\mathbf{x}; \qquad (4)\\ \text{s.t.} \quad & \mathbf{w}^\top \Delta\phi_\mathbf{x}(\mathbf{y}) \geq \Delta \mathbf{t}_\mathbf{x}(\mathbf{y}) - \xi_\mathbf{x}, \forall \mathbf{x} \in S, \forall \mathbf{y} \in \Sigma.\end{aligned}$$

where $\mathcal{C}$ is called the capacity in the support vector literature and presents a way to trade-off the training error and margin size. One should note that the number of constraints is $\sum_{i=1}^{T} |\Sigma^i|$, an extremely large number. And the corresponding dual formulation can be defined as:

$$\max \sum_{\mathbf{x},\mathbf{y}} \alpha_{\mathbf{x}}(\mathbf{y})\Delta \mathbf{t}_{\mathbf{x}}(\mathbf{y}) - \frac{1}{2}\|\sum_{\mathbf{x},\mathbf{y}} \alpha_{\mathbf{x}}(\mathbf{y})\Delta\phi_{\mathbf{x}}(\mathbf{y})\|^2;$$
$$\text{s.t.} \sum_{\mathbf{y}} \alpha_{\mathbf{x}}(\mathbf{y}) = \mathcal{C}, \forall \mathbf{x}; \alpha_{\mathbf{x}}(\mathbf{y}) \geq 0, \forall \mathbf{x}, \mathbf{y}. \quad (5)$$

where $\alpha_{\mathbf{x}}(\mathbf{y})$ is a dual variable.

As well as loss functions, kernels might have substantial influence on the performance of a classification system. M^3N is capable of incorporating many different kinds of kernel functions to reduce computations in the high-dimensional feature space $\mathcal{H}$. This is sometimes referred to as the "kernel trick" (Schölkopf and Smola, 2002; Shawe-Taylor and Cristianini, 2004). A linear kernel can be defined as

$$\kappa((\mathbf{x}, \mathbf{y}), (\mathbf{x}', \mathbf{y}')) = \langle \phi(\mathbf{x}, \mathbf{y}), \phi(\mathbf{x}', \mathbf{y}')\rangle_{\mathcal{H}} \quad (6)$$

For a polynomial kernel,

$$\kappa((\mathbf{x}, \mathbf{y}), (\mathbf{x}', \mathbf{y}')) = (s \cdot \langle \phi(\mathbf{x}, \mathbf{y}), \phi(\mathbf{x}', \mathbf{y}')\rangle_{\mathcal{H}} + r)^d, \quad (7)$$

and for a neural kernel,

$$\kappa((\mathbf{x}, \mathbf{y}), (\mathbf{x}', \mathbf{y}')) = \tanh(s \cdot \langle \phi(\mathbf{x}, \mathbf{y}), \phi(\mathbf{x}', \mathbf{y}')\rangle_{\mathcal{H}} + r), \quad (8)$$

where s, d, and r are coefficients in kernel functions.

3 Stochastic Gradient Descent

For M^3N optimization, Taskar *et al.* (2003) has proposed a reparametrization of the dual variables to take advantage of the network structure of the labeling sequence problem. The dual QP is then solved using the structured sequential minimal optimization (structured SMO) analogous to the SMO used for SVMs (Platt, 1998). However, the resulting number of constraints in the QP make the structured SMO algorithm slow to converge, or even prohibitively expensive for large-scale real world problems. In this section we will present stochastic gradient descent (SGD) method, and show SGD can significantly speedup the training of M^3N.

3.1 Regularized Loss Minimization

Recall that for M^3N, the goal is to find a linear hypothesis $h_{\mathbf{w}}$ such that $h_{\mathbf{w}}(\mathbf{x}) = \arg\max_{\mathbf{y}\in\Sigma} \mathbf{w}^\top \phi(\mathbf{x}, \mathbf{y})$. The parameters $\mathbf{w}$ are learned by minimizing a regularized loss

$$\mathcal{L}(\mathbf{w}; \{(x_i, y_i)\}_{i=1}^T, C) = \sum_{i=1}^m \ell(\mathbf{w}, x_i, y_i) + \frac{C}{2}\|\mathbf{w}\|^2. \quad (9)$$

The function ℓ measures the loss incurred in using $\mathbf{w}$ to predict the label of x_i. Following (Taskar *et al.*, 2003), $\ell(\mathbf{w}, x_i, y_i)$ is a variant of the hinge loss, and can be defined as follows:

$$\ell(\mathbf{w}, x_i, y_i) = \max_{\mathbf{y}\in\Sigma}[e(x_i, y_i, \mathbf{y}) - \mathbf{w} \cdot (\phi(x_i, y_i) - \phi(x_i, \mathbf{y}))], \quad (10)$$

where $e(x_i, y_i, \mathbf{y})$ is some non-negative measure of the error incurred in predicting y instead of y_i as the label of x_i. We assume that $e(x_i, y_i, \mathbf{y}) = 0$ for all i, so that no loss is incurred for correct prediction, and therefore $\ell(\mathbf{w}, x_i, y_i)$ is always non-negative. This loss function corresponds to the M^3N approach, which explicitly penalizes training examples for which, for some $y \neq y_i$, $\mathbf{w} \cdot (\phi(x_i, y_i) - \phi(x_i, \mathbf{y})) < e(x_i, y_i, \mathbf{y})$. And the function $\mathcal{L}$ is convex in $\mathbf{w}$ for $\ell(\mathbf{w}, x_i, y_i)$. Therefore, minimization of $\mathcal{L}$ can be re-cast as optimization of the following dual convex problem:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_i \max_{\mathbf{y}\in\Sigma}[e(x_i, y_i, \mathbf{y}) - \mathbf{w} \cdot (\phi(x_i, y_i) - \phi(x_i, \mathbf{y}))] + \frac{C}{2}\|\mathbf{w}\|^2. \quad (11)$$

3.2 The SGD Algorithm

To perform parameter estimation, we need to minimize $\mathcal{L}(\mathbf{w}; \{(x_i, y_i)\}_{i=1}^T, C)$. For this purpose we compute its gradient $\mathbf{G}(\mathbf{w})$:

$$\mathbf{G}(\mathbf{w}) = \frac{\partial}{\partial \mathbf{w}}(\mathcal{L}(\mathbf{w}; \{(x_i, y_i)\}_{i=1}^T, C)) = \frac{\partial}{\partial \mathbf{w}}(\sum_{i=1}^m \ell(\mathbf{w}, x_i, y_i) + \frac{C}{2}\|\mathbf{w}\|^2) \quad (12)$$

In addition to the gradient, second-order methods based on Newton steps also require computation and inversion of the Hessian $\mathbf{H}(\mathbf{w})$. Taking the gradient of Equation 12 *wrt.* $\mathbf{w}$ yields:

$$\mathbf{H}(\mathbf{w}) = \frac{\partial}{\partial \mathbf{w}}\mathbf{G}(\mathbf{w}) = \frac{\partial^2}{\partial \mathbf{w}^2}\mathcal{L} \qquad (13)$$

Explicitly computing the full Hessian is time consuming. Instead we can make use of the differential

$$\mathrm{d}\mathbf{G}(\mathbf{w}) = \mathbf{H}(\mathbf{w})\mathrm{d}\mathbf{w} \qquad (14)$$

to efficiently compute the product of the Hessian with a chosen vector $\mathbf{v} =: \mathrm{d}\mathbf{w}$ by forward-mode algorithmic differentiation (Pearlmutter, 1994). These Hessian-vector products can be computed along with the gradient at only 2-3 times the cost of the gradient computation alone. We denote $\mathbf{G}(\mathbf{w}) = \nabla_{\mathbf{w}}\mathcal{L}$, and each iteration of the SGD algorithm consists in drawing an example (x_i, y_i) at random and applying the parameter update rule (Robbins and Monroe, 1951):

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \cdot \nabla_{\mathbf{w}}\mathcal{L} \qquad (15)$$

where η is the learning rate in the algorithm.

The SGD algorithm has been shown to be fast, reliable, and less prone to reach bad local minima. In this algorithm, the weights are updated after the presentation of each example, according to the gradient of the loss function (Lecun *et al.*, 1998). The convergence is very fast when the training examples are redundant since only a few examples are needed to perform. This algorithm can get a good estimation after considerably few iterations.

3.3 Choosing Learning Rate η

The learning rate η is crucial to the speed of SGD algorithm. Ideally, each parameter weight w_i should have its own learning rate η_i. Because of possible correlations between input variables, the learning rate of a unit should be inversely proportional to the square root of the number of inputs to the unit. If shared weights are used, the learning rate of a weight should be inversely proportional to the square root of the number of connection sharing that weight.

For one-dimensional sequence labeling task, the optimal learning rate yields the fastest convergence in the direction of highest curvature is (Bottou, 2004):

$$\eta_{\text{opt}} = (\frac{\partial^2\mathcal{L}}{\partial \mathbf{w}^2})^{-1} = (\mathbf{H}(\mathbf{w}))^{-1}, \qquad (16)$$

and the maximum learning rate is $\eta_{\max} = 2\eta_{\text{opt}}$.

The simple SGD update offers lots of engineering opportunities. In practice, however, at any moment during the training procedure, we can select a small subset of training examples and try various learning rates on the subset, then pick the one that most reduces the cost and use it on the full dataset.

3.4 The SGD Convergence

The convergence of stochastic algorithms actually has been studied for a long time in adaptive signal processing. Given a suitable choice of the learning rate η_t, the standard (batch) gradient descent algorithm is known to converge to a local minimum of the cost function. However, the random noise introduced by SGD disrupts this deterministic picture and the specific study of SGD convergence usually is fairly complex (Benveniste *et al.*, 1987).

It is reported that for the convex case, if several assumptions and conditions are valid, then the SGD algorithm converges almost surely to the optimum $\mathbf{w}^*$ [1]. For the general case where the cost function is non-convex and has both local and global minima, if four assumptions and two learning rate assumptions hold, it is guaranteed that the gradient $\nabla_{\mathbf{w}}\mathcal{L}$ converges almost surely to zero (Bottou, 2004). We omit the details of the convergence theorem and corresponding proofs due to space limitation.

3.5 SGD Speedup

Unfortunately, many of sophisticated gradient methods are not robust to noise, and scale badly with the number of parameters. The plain SGD algorithm can be very slow to converge. Inspired by stochastic meta-descent (SMD) (Schraudolph, 1999), the convergence speed of SGD can be further improved with gradient step size adaptation by using second-order information. SMD is a highly scalable local optimizer. It shines when gradients are stochastically approximated.

In SMD, the learning rate η is simultaneously

[1] One may argue that SGD on many architectures does not result in a global optima. However, our goal is to obtain good performance on future examples in learning rather than achieving a global optima on the training set.

```
INPUT: training set $S\{(x_1,y_1),\ldots,(x_\mathbb{T},y_\mathbb{T})\}$;
        factor $\lambda$; number of iterations $N$.
INITIALIZE: $\mathbf{w}_0$, $\mathbf{v}_0 = 0$, $\eta_0$.
FOR $t = 1,2,\ldots,N$
    Choose a random example $(x_i,y_i) \in S$
    Compute the gradient $\nabla_t = \mathbf{G}_t$ and $\mathbf{H}_t\mathbf{v}_t$
    Set $\mathbf{v}_{t+1} = \lambda\mathbf{v}_t - \eta_t \cdot (\mathbf{G}_t + \lambda\mathbf{H}_t\mathbf{v}_t)$
    Update the parameter vector:
    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \cdot \nabla_t$
    Adapt the gradient step size:
    $\eta_{t+1} = \eta_t \cdot \max(\frac{1}{2}, 1 - \mu\mathbf{G}_{t+1} \cdot \mathbf{v}_{t+1})$
OUTPUT: $\mathbf{w}_{N+1}$
```

Figure 1: Pseudo-code for the SGD algorithm.

adapted via a multiplicative update with μ:

$$\eta_{t+1} = \eta_t \cdot \max(\frac{1}{2}, 1 - \mu\mathbf{G}_{t+1} \cdot \mathbf{v}_{t+1}), \quad (17)$$

where the vector $\mathbf{v}$ ($\mathbf{v} =:$ d$\mathbf{w}$) captures the long-term dependencies of parameters. $\mathbf{v}$ can be computed by the simple iterative update:

$$\mathbf{v}_{t+1} = \lambda\mathbf{v}_t - \eta_t \cdot (\mathbf{G}_t + \lambda\mathbf{H}_t\mathbf{v}_t), \quad (18)$$

where the factor $0 \le \lambda \le 1$ governs the time scale over which long-term dependencies are taken into account, and $\mathbf{H}_t\mathbf{v}_t$ can be calculated efficiently alongside the gradient by forward-mode algorithmic differentiation via Equation 14. This Hessian-vector product is computed implicitly and it is the key to SMD's efficiency. The pseudo-code for the SGD algorithm is shown in Figure 1.

4 Experiments: A Case Study of NP Chunking

4.1 Data

Our data comes from the CoNLL 2000 shared task (Sang and Buchholz, 2000). The dataset is divided into a standard training set of 8,936 sentences and a testing set of 2,012 sentences. This data consists of the same partitions of the Wall Street Journal corpus (WSJ) as the widely used data for NP chunking: sections 15-18 as training data (211,727 tokens) and section 20 as test data (47,377 tokens). And the annotation of the data has been derived from the WSJ corpus.

$w_{t-\delta} = w$
w_t matches `[A-Z]`
w_t matches `[A-Z]+`
w_t matches `[A-Z][a-z]+`
w_t matches `[A-Z]+[a-z]+[A-Z]+[a-z]`
w_t matches `.*[0-9].*`
w_t contains dash "-" or dash-based "-based"
w_t is capitalized, all-caps, single capital letter, or mixed capitalization
w_t contains years, year-spans or fractions
w_t is contained in a lexicon of words with POS p (from the Brill tagger)
$p_t = p$
$q_k(x, t+\delta)$ for all k and $\delta \in [-3, 3]$

Table 1: Input feature template $q_k(x,t)$ for NP chunking. In this table w_t is the token (word) at position t, p_t is the POS tag at position t, w ranges over all words in the training data, and p ranges over all POS tags.

4.2 Features

We follow some top-performing NP chunking systems and perform holdout methodology to design features for our model, resulting in a rich feature set including POS features provided in the official CoNLL 2000 dataset (generated by the Brill tagger (Brill, 1995), with labeling accuracy of around 95-97%), some contextual and morphological features. Table 1 lists our feature set for NP chunking.

4.3 Experimental Results

We trained linear-chain conditional random fields (CRFs) (Lafferty *et al.*, 2001) as the baseline. The well known limited memory quasi-Newton BFGS algorithm (L-BFGS) (Liu and Nocedal, 1989) was applied to learn the parameters for CRFs. To avoid over-fitting, we penalized the log-likelihood by the commonly used zero-mean Gaussian prior over the parameters. This gives us a competitive baseline CRF model for NP chunking. To make fair and accurate comparison, we used the same set of features listed in Table 1 for both M^3N and CRFs. All experiments were performed on the Linux platform, with a 3.2GHz Pentium 4 CPU and 4 GB of memory.

Model	Training Method	Kernel Function	Iteration	Training Time(s)	P(%)	R(%)	$F_{\beta=1}$
M^3N	structured SMO	linear kernel: $\langle a, b\rangle_{\mathcal{H}}$	100	1176	94.59	94.22	94.40
M^3N	structured SMO	polynomial(quadratic): $(\langle a, b\rangle_{\mathcal{H}} + 1)^2$	100	30792	94.88	94.49	**94.68**
M^3N	structured SMO	polynomial(cubic): $(\langle a, b\rangle_{\mathcal{H}} + 1)^3$	100	30889	94.47	94.01	94.24
M^3N	structured SMO	polynomial(biquadratic): $(\langle a, b\rangle_{\mathcal{H}} + 1)^4$	100	31556	93.90	93.77	93.83
M^3N	structured SMO	neural kernel: $\tanh(0.1 \cdot \langle a, b\rangle_{\mathcal{H}})$	20	7395	94.42	94.02	94.22
CRFs	L-BFGS	—	100	352	94.55	94.09	94.32

Table 2: M^3N vs. CRFs: Performance and training time comparison for NP chunking on the CoNLL 2000 official dataset. M^3N was trained using the structured SMO algorithm.

Model	Training Method	Kernel Function	Iteration	Training Time(s)	P(%)	R(%)	$F_{\beta=1}$
M^3N	SGD	linear kernel: $\langle a, b\rangle_{\mathcal{H}}$	100	89	94.58	94.21	94.39
M^3N	SGD	polynomial(quadratic): $(\langle a, b\rangle_{\mathcal{H}} + 1)^2$	100	1820	94.89	94.50	**94.69**
M^3N	SGD	polynomial(cubic): $(\langle a, b\rangle_{\mathcal{H}} + 1)^3$	100	1831	94.47	94.01	94.24
M^3N	SGD	polynomial(biquadratic): $(\langle a, b\rangle_{\mathcal{H}} + 1)^4$	100	1857	93.91	93.76	93.83
M^3N	SGD	neural kernel: $\tanh(0.1 \cdot \langle a, b\rangle_{\mathcal{H}})$	20	477	94.40	94.01	94.20
CRFs	L-BFGS	—	100	352	94.55	94.09	94.32

Table 3: M^3N vs. CRFs: Performance and training time comparison for NP chunking on the CoNLL 2000 official dataset. M^3N was trained using the SGD algorithm.

System	$F_{\beta=1}$
SVMs (polynomial kernel) (Kudo and Matsumoto, 2000)	93.79
SVM combination (Kudo and Matsumoto, 2001)	94.39
Generalized winnow (Zhang *et al.*, 2002)	94.38
Voted perceptron (Collins, 2002)	94.09
CRFs (Sha and Pereira, 2003)	94.38
Second order CRFs (McDonald *et al.*, 2005)	94.29
Chunks from the Charniak Parser (Hollingshead *et al.*, 2005)	94.20
Second order latent-dynamic CRFs + improved A* search based inference (Sun *et al.*, 2008)	94.34
Our approach	94.69

Table 4: NP chunking: Comparison with some existing state-of-the-art systems.

Similar to other discriminative graphical models such as CRFs, the modeling flexibility of M^3N permits the feature functions to be complex, arbitrary, nonindependent, and overlapping features, allowing the multiple features described in Table 1 to be directly exploited. Moreover, M^3N is capable of incorporating multiple kernel functions (see Section 2) which allow the efficient use of high-dimensional feature spaces during the experiments.

The resulting number of features is 7,835,439, and both M^3N and CRFs were trained to predict 47,366 tokens with 12,422 noun phrases in the testing set. For simplicity, we denote $a = \phi(\mathbf{x}, \mathbf{y})$, and $b = \phi(\mathbf{x}', \mathbf{y}')$, and the linear kernel can be rewritten as $\kappa(a, b) = \langle a, b\rangle_{\mathcal{H}}$. We performed holdout methodology to find optimal values for coefficients s, d, and r in M^3N kernel functions. For polynomial kernels, we varied d from 2 to 4, resulting in quadratic, cubic, and biquadratic kernels, respectively. Finally, we chose optimized values: $s = 1, r = 1$ for polynomial kernels, and $s = 0.1, r = 0$ for neural kernels. The capacity $\mathcal{C}$ for M^3N was set to 1 in our experiments.

Table 2 shows comparative performance and training time for M^3N (trained with structured SMO) and CRFs, while Table 3 shows comparative performance and training time for M^3N (trained with SGD) and CRFs [2]. For M^3N, when trained with quadratic kernel and structured SMO, the best F-measure of 94.68 was achieved, leading to an improvement of 0.36 compared to the CRF baseline. What follows is the linear kernel that obtained 94.40 F-measure. The cubic and neural kernels obtained close performance, while the biquadratic kernel led to the worst performance. However, the structured SMO is very computationally intensive, especially for polynomial kernels. For example, CRFs converged in 352 sec-

[2]We used Taku Kudo's CRF++ toolkit (available at http://crfpp.sourceforge.net/) in our experiments. The M^3N model, and the structured SMO and SGD training algorithms were also implemented using C++.

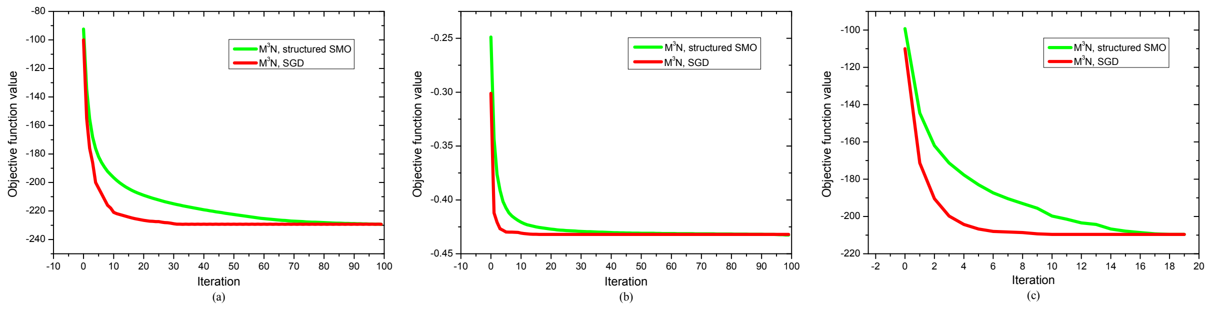

Figure 2: Convergence speed comparison for structured SMO and SGD algorithms. The X axis shows number of training iterations, and the Y axis shows objective function value. (a) The M^3N model was trained using linear kernel. (b) The M^3N model was trained using polynomial(quadratic) kernel. (c) The M^3N model was trained using neural kernel.

onds, while M^3N (polynomial kernels) took more than 8.5 hours to finish training.

As can be seen in Table 3, the SGD algorithm significantly accelerated the training procedure of M^3N without sacrificing performance. When the linear kernel was used, M^3N finished training in 89 seconds, more than 13 times faster than the model trained with structured SMO. And it is even much faster than the CRF model trained with L-BFGS. More importantly, SGD obtained almost the same performance as structured SMO with all M^3N kernel functions.

Table 4 gives some representative NP chunking results for previous work and for our best model on the same dataset. These results showed that our model compares favorably with existing state-of-the-art systems [3].

Figure 2 compares the convergence speed of structured SMO and SGD algorithms for the M^3N model. Linear (Figure 2 (a)), polynomial(quadratic) (Figure 2 (b)) and neural kernels (Figure 2 (c)) were used [4]. We calculated objective function values during effective training iterations. It can be seen that both structured SMO and SGD algorithms converge to the same objective function value for different kernels, but SGD converges considerably faster than the structured SMO.

Figure 3 (a) demonstrates the effect of training set size on performance for NP chunking. We increased the training set size from 1,000 sentences to 8,000 sentences, with an incremental step of 1,000. And the testing set was fixed to be 2,012 sentences. The M^3N models (with different kernels) were trained using the SGD algorithm. It is particularly interesting to know that the performance boosted for all the models when increasing the training set size. Using linear and quadratic kernels, M^3N model significantly and consistently outperforms the CRF model for different training set sizes. The cubic and neural kernels lead to almost the same performance for M^3N, which is slightly lower than the CRF baseline. As illustrated by the curves, M^3N (trained with quadratic kernel) achieved the best performance and larger training set size leads to better improvement for this model when compared to the CRF model, while M^3N (trained with biquadratic kernel) obtained the worst performance among all the models.

Accordingly, Figure 3 (b) shows the impact of increasing the training set size on training time for NP chunking. Increasing training set size leads to an increase in the computational complexity of training procedure for all models. For the M^3N model, it is faster when trained with linear kernel than the CRF model. And the three polynomial kernels (quadratic, cubic and biquadratic) have roughly the same training time. For CRFs and (M^3N, neural kernel), the training time is close to each other. For example, when the training set contains 1,000 sentences, the training time for CRFs, (M^3N, linear kernel), (M^3N, quadratic kernel), (M^3N, cubic kernel), (M^3N, biquadratic kernel), and (M^3N, neural kernel) is 24s, 7s, 72s,

[3]Note that it is difficult to compare strictly, since reported results sometimes leave out details (e.g., feature sets, significance tests, etc) needed for accurate comparison.

[4]For cubic and biquadratic kernels, the curves are very similar to that of quadratic kernel, and we omitted them for space.

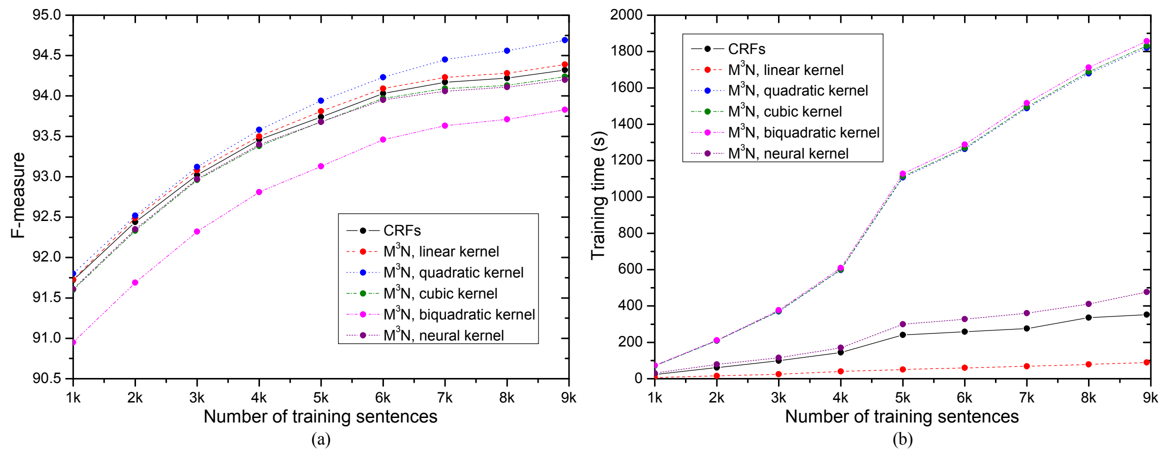

Figure 3: (a) Effect of training set size on performance for NP chunking. The training set size was increased from 1,000 sentences to 8,000 sentences, with an incremental step of 1,000. The testing set contains 2,012 sentences. All the M^3N models (with different kernels) were trained using the SGD algorithm. (b) Effect of training set size on training time for NP chunking.

72s, 74s, and 30s. When trained on 8,000 sentences, the numbers become 336s, 79s, 1679s, 1689s, 1712s, and 411s, respectively.

5 Related Work

The M^3N framework and its variants have generated much interest and great progress has been made, as evidenced by their promising results evaluated in handwritten character recognition, collective hypertext classification (Taskar *et al.*, 2003), parsing (Taskar *et al.*, 2004), and XML tag relabeling (Spengler, 2005). However, all the above mentioned research work used structured SMO algorithm for parameter learning, which can be computationally intensive, especially for very large datasets.

Recently, similar stochastic gradient methods have been applied to train log-linear models such as CRFs (Vishwanathan *et al.*, 2006). However, the maximum margin loss has a discontinuity in its derivative, making optimization of such models somewhat more involved than log-linear ones. We first exploit SGD method for fast parameter learning of M^3N and achieve state-of-the-art performance on the NP chunking task in the NLP community.

Several algorithms have been proposed to train max-margin models, including cutting plane SMO (Tsochantaridis *et al.*, 2005), exponentiated gradient (Bartlett *et al.*, 2004; Collins *et al.*, 2008), extragradient (Taskar *et al.*, 2006), and subgradient (Shalev-Shwartz *et al.*, 2007). Some methods are similar to SGD in that they all process a single training example at a time. The SGD methods directly optimize the primal problem, and at each update use a single example to approximate the gradient of the primal objective function. Some of the proposed algorithms, such as exponentiated gradient corresponds to block-coordinate descent in the dual, and uses the exact gradient with respect to the block being updated. We plan to implement and compare some of these algorithms with SGD for M^3N.

6 Conclusion and Future Work

We have presented the first known empirical study on sequence labeling based on M^3N. We have also provided the efficient SGD algorithm and shown how it can be applied to significantly speedup the training procedure of M^3N. As a case study, we performed extensive experiments on standard dataset for NP chunking, showing the promising and competitiveness of our approach. Several interesting issues, such as the convergence speed of the SGD algorithm, the effect of training set size on performance for NP chunking, and the effect of training set size on training time, were also investigated in our experiments. For the future work, we plan to further the scalability and applicability of our approach and evaluate it on other large-scale real world sequence labeling tasks, such as POS tagging and NER.

References

Peter L. Bartlett, Ben Taskar, Michael Collins, and David Mcallester. Exponentiated gradient algorithms for large-margin structured classification. In *Proceedings of NIPS-04*, pages 113–120. MIT Press, 2004.

A. Benveniste, M. Metivier, and P. Priouret. Algorithmes adaptatifs et approximations stochastiques. *Masson*, 1987.

Léon Bottou. 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, 2004.

Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–565, 1995.

Michael Collins, Amir Globerson, Terry Koo, Xavier Carreras, and Peter L. Bartlett. Exponentiated gradient algorithms for conditional random fields and Max-margin Markov networks. *Journal of Machine Learning Research*, 9:1775–1822, 2008.

Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of HLT/EMNLP-02*, pages 1–8, 2002.

Kristy Hollingshead, Seeger Fisher, and Brian Roark. Comparing and combining finite-state and context-free parsers. In *Proceedings of HLT/EMNLP-05*, pages 787–794, Vancouver, British Columbia, Canada, 2005.

Taku Kudo and Yuji Matsumoto. Use of support vector learning for chunk identification. In *Proceedings of CoNLL-2000 and LLL-2000*, pages 142–144, Lisbon, Portugal, 2000.

Taku Kudo and Yuji Matsumoto. Chunking with support vector machines. In *Proceedings of HLT/NAACL-01*, pages 1–8, 2001.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML-01*, pages 282–289, 2001.

Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

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

Ryan McDonald, Koby Crammer, and Fernando Pereira. Flexible text segmentation with structured multilabel classification. In *Proceedings of HLT/EMNLP-05*, pages 987–994, Vancouver, British Columbia, Canada, 2005.

Barak A. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1):147–160, 1994.

John C. Platt. Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods: Support Vector Learning*, pages 41–64, 1998.

H. Robbins and S. Monroe. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

Erik Tjong Kim Sang and Sabine Buchholz. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of CoNLL-2000*, pages 127–132, Lisbon, Portugal, 2000.

Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

Nicol N. Schraudolph. Local gain adaptation in stochastic gradient descent. In *Proceedings of the 9th International Conference on Artificial Neural Networks*, pages 569–574, 1999.

Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Proceedings of HLT/NAACL-03*, pages 213–220, 2003.

Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-GrAdient SOlver for SVM. In *Proceedings of ICML-07*, pages 807–814, New York, NY, USA, 2007.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, UK, 2004.

Alex Spengler. Maximum margin Markov networks for XML tag relabelling. Master's thesis, University of Karlsruhe, 2005.

Xu Sun, Louis-Philippe Morency, Daisuke Okanohara, and Jun'ichi Tsujii. Modeling latent-dynamic in shallow parsing: A latent conditional model with improved inference. In *Proceedings of COLING-08*, pages 841–848, Manchester, UK, 2008.

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin Markov networks. In *Proceedings of NIPS-03*. MIT Press, 2003.

Ben Taskar, Dan Klein, Michael Collins, Daphne Koller, and Christopher Manning. Max-margin parsing. In *Proceedings of HLT/EMNLP-04*, pages 1–8, 2004.

Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. Learning structured prediction models: A large margin approach. In *Proceedings of ICML-05*, pages 896–903, Bonn, Germany, 2005.

Ben Taskar, Simon Lacoste-Julien, and Michael I. Jordan. Structured prediction via the extragradient method. In *Proceedings of NIPS-06*. MIT Press, 2006.

Ben Taskar. *Learning Structured Prediction Models: A Large Margin Approach*. PhD thesis, Stanford University, December 2004.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, Inc., New York, USA, 1995.

S. V. N. Vishwanathan, Nicol N. Schraudolph, Mark W. Schmidt, and Kevin P. Murphy. Accelerated training of conditional random fields with stochastic gradient methods. In *Proceedings of ICML-06*, pages 969–976, Pittsburgh, Pennsylvania, 2006.

Tong Zhang, Fred Damerau, and David Johnson. Text chunking based on a generalization of winnow. *Journal of Machine Learning Research*, 2:615–637, 2002.