國立臺灣大學電機資訊學院資訊工程學研究所

碩士論文

Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

master thesis

應用截斷牛頓法於條件隨機場

Newton Methods for Conditional Random Fields

陳鵬仁

Peng-Jen Chen

指導教授：林智仁 博士

Advisor: Chih-Jen Lin, Ph.D.

中華民國 98 年 6 月

June, 2009

Newton Methods for Conditional Random Fields

by

Peng-Jen Chen

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Master of Science
(Computer Science and Information Engineering)
in National Taiwan University
2009

國立臺灣大學碩士學位論文
口試委員會審定書

應用截斷牛頓法於條件隨機場

Newton Method for CRF

本論文係陳鵬仁君（學號 R96922049）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 98 年 6 月 11 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

林軒田　　李育杰

（指導教授）

系主任　呂育道

摘要

條件隨機場是一個適合用來標記序列性資料的模組。由於考慮序列中所有可能的標籤組合，條件隨機場在學習及預測階段都非常耗時。牛頓法在最佳化的最後階段具有較快的收斂性質，因此我們採用牛頓法來解條件隨機場。海森矩陣向量乘積是整個計算過程中最耗時的部份。本篇論文提出一個新的動態規劃技巧，可以在多項式時間複雜度內完成海森矩陣向量乘積。

關鍵辭: 共軛梯度法、信賴區間牛頓法、最大熵值法、條件隨機場。

ABSTRACT

Conditional Random Fields (CRFs) is a useful technique to label sequential data. Due to considering all label combinations of a sequence, CRFs' training and testing are time consuming. In this work, we consider a Newton method for training CRFs because of its possible fast final convergence. The computational bottleneck is on the Hessian-vector product. We propose a novel dynamic programming technique to calculate it in polynomial time.

KEYWORDS: conjugate gradient methods, trust region Newton methods, maximum entropy, conditional random fields.

TABLE OF CONTENTS

CHAPTER I

Introduction

Conditional Random Fields (CRFs), a type of discriminative probability model, were first proposed in Lafferty et al. (2001). The linear-chain type CRFs model is especially widely used on segmenting and labeling sequential data. Some applications such as noun phrase chunking task (Sha and Pereira, 2003), part-of-speech (POS) tagging (Lafferty et al., 2001), named entity recognitions (NER) (Mccallum and Li, 2003) and Chinese segmentation (Peng et al., 2004) get good results with CRFs.

For solving a CRFs model, Lafferty et al. (2001) use two algorithms based on improved iterative scaling (IIS) to estimate the parameter, but get slow convergence. Sha and Pereira (2003) proposed two quasi-Newton methods, limited memory BFGS (L-BFGS) (Liu and Nocedal, 1989) and pre-conditioned conjugate gradient, and has better results than IIS methods. Where L-BFGS only needs gradient computation, and pre-conditioned conjugate gradient only uses diagonal elements of the Hessian matrix. Newton methods optimize object function with Hessian information but the analytical form of second derivative CRFs is too complex to compute. Without any additional derivation, it takes exponential time of complexity.

Automatic differentiation (AD) (Griewank, 2000) is a technique which can be used to calculate the gradient without having any analytical form of gradient. This idea can be also apply to Hessian-vector products. Vishwanathan et al. (2006) apply AD

to do the Hessian-vector product in a stochastic meta-descent (SMD).

Trust region Newton method (TRON) is a kind of truncated Newton methods. It uses conjugate gradient to guarantee the function value convergence. Lin et al. (2008) adopt TRON to solve large-scaled logistic regression problems, and obtain better results than L-BFGS on some data sets. A logistic regression model can be seen as a special case of CRFs model, so we expect TRON will have better performance than L-BFGS. Wang (2008) tried to use TRON on CRFs models, and used AD to do Hessian-vector products, which are needed in conjugate gradient procedure. The experiments show that the training time of TRON is slower than L-BFGS.

In this thesis, instead of using AD, we derive an analytical form of the Hessian-vector product of CRFs and give a dynamic programming technique to compute it in polynomial time complexity. We also discuss details of time/space complexity.

The chapters are organized as follows: Chapter II introduces the formula of CRF model, and explains how to apply the CRF model on solving named-entity recognition problems. Chapter III shows the TRON algorithm and gives the details of calculating gradient. Chapter IV describes how to calculate the Hessian-vector product with dynamic programming, and analyzes the time/space complexity.

CHAPTER II

Conditional Random Fields

Conditional Random Fields (CRFs), a probability graphical model for segmenting and labeling sequence data, was first introduced in Lafferty et al. (2001). Linear-chain is the most important special case of CRFs model, and it has many successful applications on natural language processing (NLP). In this chapter, we describe an NLP application called named-entity recognition (NER) problem, introduce the formula of linear-chain CRFs model and explain how to apply CRFs on NER problem.

2.1 Named-Entity Recognition Problem

Named-entities are phrases that contain the names of persons, organizations, locations, times and quantities. For example, "Knuth" is a person, "Taiwan" is a location, and "National Taiwan University" is an organization. The named-entity recognition task is to select a predefined named-entity tag for each word in given sentences.

A named-entity word will be marked *PER* as a person, *LOC* as a location and *ORG* as an organization. If a word does not belong to any named-entity, it is marked as *O* .

We take a real sentence as an example in Table 2.1. This sentence is instantiated by Conference on Computational Natural Language Learning (CoNLL) 2002 shared task website, `http://www.cnts.ua.ac.be/conll2002/ner/`.

Table 2.1: An example of tagging name-entity tags.

token	Wolff	,	currently	a	journalist	in	Argentina	,
tag	PER	O	O	O	O	O	LOC	O
	played	with	Del	Bosque	in	the	final	years
	O	O	PER		O	O	O	O
	of	the	seventies	in	Real	Madrid	.	
	O	O	O	O	ORG		O	

In this sample sentence, "Wolff" is a person name, and we give it a tag *PER* . "Argentina" is a country name and should be tagged with *LOC* . Two *PER* tags are tagged on "Del Bosque" since these two words compose a person name. "Real Madrid" is an organization name, so two *ORG* tags are given. Other words including punctuation marks are tagged with *O* .

To give a more precise formulation, we define some symbols. $D = \{\boldsymbol{x}^i, \boldsymbol{y}^i\}_{i=1}^N$ is an *i.i.d.* data set. $\boldsymbol{x}^i$ represents the ith sentence, and is composed of some sequential words $\boldsymbol{x}^i = \{x_1^i, x_2^i, \ldots, x_{T_i}^i\}$. T_i is the number of words in the ith sentence and N is the number of sentences in data set D. The task is to learn how to predict tags from the given tag sequences $\boldsymbol{y}^i = \{y_1^i, y_2^i, \ldots, y_{T_i}^i\}$, where each tag $y_t^i \in Y_D$ corresponds to a word x_t^i. Y_D is the set of predefined tags. In the above example, $Y_D = \{PER\ , LOC\ , ORG\ , O\ \}$.

In an NER problem, the difficult part is that not every named-entity term has been shown in training corpus, and sometimes, even the same word has different tags. For example, "Taiwan" is *LOC* , but "National Taiwan University" should be tagged with *ORG* . To solve this problem, linear-chain CRFs learns the information not only from context, but also from the previous tag it predicts. In the following discussions, we call linear-chain CRFs model as CRFs.

2.2 Conditional Random Fields

CRFs model follows the same assumptions as Hidden Markov model (HMM) that the tth tag is only related to two independent factors when modeling the probability distribution, $p(\boldsymbol{y}, \boldsymbol{x})$. First, y_t is related to the $(t-1)$th predicted tag, y_{t-1}. We use a probability function $p(y_t|y_{t-1})$ to model it. Second, the tth observation, x_t, is related to the tth tag, y_t. We use another probability function $p(x_t|y_t)$ to model it. Under these assumptions, the joint probability distribution of a sequence, $(\boldsymbol{y}, \boldsymbol{x})$, can be written as

$$p(\boldsymbol{y}, \boldsymbol{x}) = \prod_{t=1}^{T} p(y_t|y_{t-1})p(x_t|y_t). \tag{2.1}$$

For these two probability functions, $p(y_t|y_{t-1})$ and $p(x_t|y_t)$, CRFs uses an exponential function and a parameter, $\boldsymbol{\lambda}$, to model them, where Z' and Z'' are normalization terms.

$$p(y_t|y_{t-1}) = \frac{1}{Z'} \exp(\lambda_{y_t, y_{t-1}}), \tag{2.2}$$

$$p(x_t|y_t) = \frac{1}{Z''} \exp(\lambda_{y_t, x_t}). \tag{2.3}$$

Applying (2.2) and (2.3) to (2.1), we have

$$p(\boldsymbol{y}, \boldsymbol{x}) = \frac{1}{Z'''} \exp\left(\sum_{t=1}^{T} (\lambda_{y_t, y_{t-1}} + \lambda_{y_t, x_t})\right). \tag{2.4}$$

To easily represent equation (2.4), we introduce a concept of feature functions. For those two probability assumptions, we consider two kinds of 0/1 feature functions:

$$f_{rs}(y_t, y_{t-1}, x_t) = \mathbf{1}_{y_t=s} \mathbf{1}_{y_{t-1}=r}. \tag{2.5}$$

$$f_{os}(y_t, y_{t-1}, x_t) = \mathbf{1}_{y_t=s} \mathbf{1}_{x_t=o}, \tag{2.6}$$

The number of feature functions in (2.5) equals the number of probability values in the probability function (2.2). And (2.6) is corresponding to (2.3).

To simplify the notation, we use integer $k = 1, 2, \ldots, d$ to index all possible rs and os. d is the total number of features which is equal to the number of all possible rs

and *os* indices. We further define $\boldsymbol{f}(\boldsymbol{y}, \boldsymbol{x})$ as a column vector function, where

$$f_k(\boldsymbol{y}, \boldsymbol{x}) = \sum_{t=1}^{T} f_k(y_t, y_{t-1}, x_t), \qquad \forall k = 1, 2, \ldots, d. \tag{2.7}$$

With the same indexing method as the feature function (2.7), we also use a column vector $\boldsymbol{\lambda} \in R^d$ to represent all parameters $\lambda_{y_t, y_{t-1}}$ and λ_{y_t, x_t}. Thus, (2.4) can be rewritten as

$$p(\boldsymbol{y}, \boldsymbol{x}) = \frac{1}{Z'''} \exp\left(\boldsymbol{f}(\boldsymbol{y}, \boldsymbol{x})^T \boldsymbol{\lambda}\right). \tag{2.8}$$

To avoid a label bias problem (Lafferty et al., 2001), CRFs models the conditional probability, $p(\boldsymbol{y}|\boldsymbol{x})$, instead of the joint probability, $p(\boldsymbol{y}, \boldsymbol{x})$. We derive (2.9) with simple probability law,

$$\begin{aligned} p(\boldsymbol{y}|\boldsymbol{x}) &= \frac{p(\boldsymbol{y}, \boldsymbol{x})}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{y}, \boldsymbol{x})}{\sum_{\boldsymbol{y'} \in {Y_D}^T} p(\boldsymbol{y'}, \boldsymbol{x})} \\ &= \frac{\exp\left(\boldsymbol{f}(\boldsymbol{y}, \boldsymbol{x})^T \boldsymbol{\lambda}\right)}{\sum_{\boldsymbol{y'}} \exp(\boldsymbol{f}(\boldsymbol{y'}, \boldsymbol{x})^T \boldsymbol{\lambda})}. \end{aligned} \tag{2.9}$$

Intuitively, when observing a sequence $\boldsymbol{x}, \boldsymbol{y}$, the optimal parameter is $\boldsymbol{\lambda}^*$ which makes the probability function (2.9) has the maximum value.

2.3 Applying CRFs on NER Problems

For a given iid data set, $D = \{\boldsymbol{x}^i, \boldsymbol{y}^i\}_{i=1}^N$, the conditional probability of $\boldsymbol{y}$ given $\boldsymbol{x}$ is

$$p(\boldsymbol{y}^1, \boldsymbol{y}^2, \ldots, \boldsymbol{y}^N | \boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^N) = \prod_{i=1}^{N} p(\boldsymbol{y}^i | \boldsymbol{x}^i). \tag{2.10}$$

To find the most possible model on the given data set, we want to maximize the probability function or equally minimize the negative log-likelihood formulation. To avoid overfitting, an regularization term, $\frac{||\boldsymbol{\lambda}||^2}{2\sigma^2}$ is usually added to the object function.

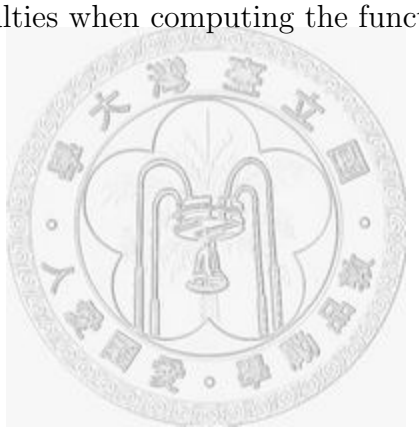
The constant $2\sigma^2$ is specified by users. The object function we want to minimize is

$$\begin{aligned}\min_{\boldsymbol{\lambda}} L(\boldsymbol{\lambda}) &= \frac{||\boldsymbol{\lambda}||^2}{2\sigma^2} - \log \prod_{i=1}^{N} p(\boldsymbol{y}^i|\boldsymbol{x}^i) \\ &= \frac{||\boldsymbol{\lambda}||^2}{2\sigma^2} - \sum_{i=1}^{N} \log p(\boldsymbol{y}^i|\boldsymbol{x}^i) \\ &= \frac{||\boldsymbol{\lambda}||^2}{2\sigma^2} - \sum_{i=1}^{N} \left(\boldsymbol{f}(\boldsymbol{y}^i, \boldsymbol{x}^i)^T \boldsymbol{\lambda} - \log Z(\boldsymbol{x}^i) \right). \end{aligned} \tag{2.11}$$

The normalization term is defined as

$$Z(\boldsymbol{x}^i) = \sum_{\boldsymbol{y'} \in {Y_D}^{T_i}} \exp(\boldsymbol{f}(\boldsymbol{y'}, \boldsymbol{x})^T \boldsymbol{\lambda}). \tag{2.12}$$

Note that $\boldsymbol{y'}$ here considers all exponential possibly tag sequences on the given $\boldsymbol{x}^i$. This will cause some difficulties when computing the function value and the gradient.



CHAPTER III

Trust Region Newton Methods

According to the discussion in Chapter II, we have an unconstrained convex optimization problem.

$$\min_{\boldsymbol{\lambda}} L(\boldsymbol{\lambda}) = \frac{||\boldsymbol{\lambda}||^2}{2\sigma^2} - \sum_{i=1}^{N} \left(\boldsymbol{f}(\boldsymbol{y}^i, \boldsymbol{x}^i)^T \boldsymbol{\lambda} - \log Z(\boldsymbol{x}^i) \right). \tag{3.1}$$

Trust region Newton method (TRON) (Lin and Moré, 1999) is a kind of Newton methods, which needs exact Hessian-vector products during the algorithm. Lin et al. (2008) use TRON to solve large-scale logistic regression and get better performance than LBFGS. Since logistic regression can be viewed as an special case of CRFs models, we attempt to apply TRON on solving CRFs.

3.1 A Trust Region Newton Method

A trust region Newton method minimizes the object function, $L(\boldsymbol{\lambda})$, iteratively. The parameter $\boldsymbol{\lambda}$ is updated sequentially with a sequence $\{\boldsymbol{\lambda}^0, \boldsymbol{\lambda}^1, \ldots, \boldsymbol{\lambda}^m, \ldots\}$. At the mth iteration, it uses a quadratic function

$$q_m(\boldsymbol{v}) = \nabla L(\boldsymbol{\lambda}^m)^T \boldsymbol{v} + \frac{1}{2} \boldsymbol{v}^T \nabla^2 L(\boldsymbol{\lambda}^m) \boldsymbol{v} \tag{3.2}$$

to approximate the function decrease $L(\boldsymbol{\lambda}^m + \boldsymbol{v}^m) - L(\boldsymbol{\lambda}^m)$ in a trust region Δ_m. The task of each iteration becomes solving a sub-problem

$$\boldsymbol{v}^m = \arg\min_{\boldsymbol{v}} \quad q_m(\boldsymbol{v}), \qquad \text{such that } \|\boldsymbol{v}\| \leq \Delta_m. \tag{3.3}$$

A ratio

$$\rho_m = \frac{L(\boldsymbol{\lambda}^m + \boldsymbol{v}^m) - L(\boldsymbol{\lambda}^m)}{q_m(\boldsymbol{v}^m)} \tag{3.4}$$

of the actual function value reduction to the approximate reduction is used to estimate the performance of the approximation. The mth step is accepted if ρ_m is larger than a given constant $\eta_0 > 0$, and the parameter is updated by

$$\boldsymbol{\lambda}^{m+1} = \begin{cases} \boldsymbol{\lambda}^m + \boldsymbol{v}^m & \text{if } \rho_m > \eta_0, \\ \boldsymbol{\lambda}^m & \text{if } \rho_m \leq \eta_0. \end{cases} \tag{3.5}$$

As the parameter changes, the trust region should also be adjusted to fit the object function. In Lin and Moré (1999), updating rules of the trust region size, Δ_m, depend on positive constant thresholds η_1 and η_2, where $0 < \eta_1 < \eta_2 < 1$, and positive updating rates $\sigma_1, \sigma_2, \sigma_3$, where $0 < \sigma_1 < \sigma_2 < 1 < \sigma_3$. After solving the mth sub-problem (3.3), Δ_m is updated by the rules

$$\begin{aligned} &\Delta_{m+1} \in [\sigma_1 \min\{\|\boldsymbol{v}^m\|, \Delta_m\}, \sigma_2 \Delta_m] && \text{if} \quad \rho_m \leq \eta_1, \\ &\Delta_{m+1} \in [\sigma_1 \Delta_m, \sigma_3 \Delta_m] && \text{if} \quad \rho_m \in (\eta_1, \eta_2), \\ &\Delta_{m+1} \in [\Delta_m, \sigma_3 \Delta_m] && \text{if} \quad \rho_m \geq \eta_2. \end{aligned} \tag{3.6}$$

A description of the trust region algorithm is given in Algorithm 1.

At the minimum of the quadratic function (3.2), we have a necessary condition

$$\nabla L(\boldsymbol{\lambda}^m)^T + \nabla^2 L(\boldsymbol{\lambda}^m) \boldsymbol{v}^* = \mathbf{0}. \tag{3.11}$$

A standard conjugate gradient method can solve such a linear system iteratively. But in (3.11), the trust region constraint should be considered. A modified conjugate gradient

Algorithm 1 A trust region algorithm for CRF

1. Given $\boldsymbol{\lambda}^0$.
2. For $m = 0, 1, \ldots$ (outer iterations)
   - If $\nabla L(\boldsymbol{\lambda}^m) = \mathbf{0}$, stop.
   - Find an approximate solution $\boldsymbol{v}^m$ of the trust region sub-problem

$$\min_{\boldsymbol{v}} \ q_m(\boldsymbol{v}), \quad \text{subject to } \|\boldsymbol{v}\| \le \Delta_m. \tag{3.7}$$

   - Compute ρ_m via (3.8):

$$\rho_m = \frac{L(\boldsymbol{\lambda}^m + \boldsymbol{v}^m) - L(\boldsymbol{\lambda}^m)}{q_m(\boldsymbol{v}^m)} \tag{3.8}$$

   - Update $\boldsymbol{\lambda}^m$ to $\boldsymbol{\lambda}^{m+1}$ according to (3.9):

$$\boldsymbol{\lambda}^{m+1} = \begin{cases} \boldsymbol{\lambda}^m + \boldsymbol{v}^m & \text{if } \rho_m > \eta_0, \\ \boldsymbol{\lambda}^m & \text{if } \rho_m \le \eta_0. \end{cases} \tag{3.9}$$

   - Obtain Δ_{m+1} according to (3.10).

$$\begin{array}{lll} \Delta_{m+1} \in [\sigma_1 \min\{\|\boldsymbol{v}^m\|, \Delta_m\}, \sigma_2 \Delta_m] & \text{if} & \rho_m \le \eta_1, \\ \Delta_{m+1} \in [\sigma_1 \Delta_m, \sigma_3 \Delta_m] & \text{if} & \rho_m \in (\eta_1, \eta_2), \\ \Delta_{m+1} \in [\Delta_m, \sigma_3 \Delta_m] & \text{if} & \rho_m \ge \eta_2. \end{array} \tag{3.10}$$

method is given in Algorithm 2. It approximately solves the trust region sub-problem (3.7). At the early iterations, the updating step $\boldsymbol{d}^i$ will follow the steepest decent direction $\boldsymbol{r}^0 = -\nabla L(\boldsymbol{\lambda}^m)$ to have more function value decrease. To cost less time, a given constant $\xi_m < 1$ is used in an early stopping condition (3.12). (3.14) projects the solution in the trust region such that $\boldsymbol{v}^m$ is a feasible solution of (3.2).

3.2 Hessian-vector Product in Conjugate Gradient

When adapting TRON to solve the minimization problem $L(\boldsymbol{\lambda})$, in addition to the gradient, the Hessian-vector product is needed at each iteration; see (3.13) and (3.15).

Algorithm 2 Conjugate gradient procedure for approximately solving the trust region sub-problem (3.7)

1. Given $\xi_m < 1, \Delta_m > 0$. Let $\bar{\boldsymbol{v}}^0 = \mathbf{0}, \boldsymbol{r}^0 = -\nabla L(\boldsymbol{\lambda}^m)$, and $\boldsymbol{d}^0 = \boldsymbol{r}^0$.
2. For $i = 0, 1, \ldots$ (inner iterations)
   - If
   $$\|\boldsymbol{r}^i\| \leq \xi_m \|\nabla L(\boldsymbol{\lambda}^m)\|, \tag{3.12}$$
   then output $\boldsymbol{v}^m = \bar{\boldsymbol{v}}^i$ and stop.
   - $$\alpha_i = \|\boldsymbol{r}^i\|^2 / ((\boldsymbol{d}^i)^T \nabla^2 L(\boldsymbol{\lambda}^m) \boldsymbol{d}^i). \tag{3.13}$$
   - $\bar{\boldsymbol{v}}^{i+1} = \bar{\boldsymbol{v}}^i + \alpha_i \boldsymbol{d}^i$.
   - If $\|\bar{\boldsymbol{v}}^{i+1}\| \geq \Delta_m$, compute τ such that
   $$\|\bar{\boldsymbol{v}}^i + \tau \boldsymbol{d}^i\| = \Delta_m, \tag{3.14}$$
   then output $\boldsymbol{v}^m = \bar{\boldsymbol{v}}^i + \tau \boldsymbol{d}^i$ and stop.
   - $$\boldsymbol{r}^{i+1} = \boldsymbol{r}^i - \alpha_i \nabla^2 L(\boldsymbol{\lambda}^m) \boldsymbol{d}^i. \tag{3.15}$$
   - $\beta_i = \|\boldsymbol{r}^{i+1}\|^2 / \|\boldsymbol{r}^i\|^2$.
   - $\boldsymbol{d}^{i+1} = \boldsymbol{r}^{i+1} + \beta_i \boldsymbol{d}^i$.

We denote $\boldsymbol{H}$ as $\nabla^2 L(\boldsymbol{\lambda})$, so

$$\boldsymbol{H}\boldsymbol{v} = \nabla^2 L(\boldsymbol{\lambda}) \boldsymbol{v}.$$

The Hessian matrix is a d by d matrix, where d is the number of parameters in $\boldsymbol{\lambda}$. In CRFs problems, d is usually large, so it is almost impossible to calculate and store the whole matrix in the memory.

In this thesis, we describe a polynomial time complexity algorithm to calculate Hessian-vector product of CRFs, so that it can be applied on the TRON algorithm directly without using automatic differentiation. We give the well-known procedure to calculate gradient in the next section, and details of the Hessian-vector product procedure in Chapter IV.

3.3 Gradient Calculation

The partial derivative of (3.1) respects to λ_k is

$$\begin{aligned}&\frac{\partial}{\partial\lambda_k}L(\boldsymbol{\lambda})\\ =&\frac{\lambda_k}{\sigma^2}-\sum_{i=1}^{N}f_k(\boldsymbol{y}^i,\boldsymbol{x}^i)+\sum_{i=1}^{N}\sum_{\boldsymbol{y}\in\{Y_D\}^{T_i}}f_k(\boldsymbol{y},\boldsymbol{x}^i)\frac{\exp(\boldsymbol{f}(\boldsymbol{y},\boldsymbol{x}^i)^T\boldsymbol{\lambda})}{Z(\boldsymbol{x}^i)}\end{aligned}\tag{3.16}$$

The above formula involves exponentially many possible y. Fortunately, a dynamic programming technique has been introduced to efficiently calculate (3.16) (Rabiner, 1989). We have

(3.16)

$$=\frac{\lambda_k}{\sigma^2}-\sum_{i=1}^{N}f_k(\boldsymbol{y}^i,\boldsymbol{x}^i)+\sum_{i=1}^{N}\sum_{\boldsymbol{y}\in\{Y_D\}^{T_i}}\sum_{t=1}^{T_i}f_k(y_t,y_{t-1},x_t^i)\frac{\exp(\boldsymbol{f}(\boldsymbol{y},\boldsymbol{x}^i)^T\boldsymbol{\lambda})}{Z(\boldsymbol{x}^i)}$$

$$=\frac{\lambda_k}{\sigma^2}-\sum_{i=1}^{N}f_k(\boldsymbol{y}^i,\boldsymbol{x}^i)+\sum_{i=1}^{N}\frac{1}{Z(\boldsymbol{x}^i)}\sum_{t=1}^{T_i}\sum_{\boldsymbol{y}\in\{Y_D\}^{T_i}}f_k(y_t,y_{t-1},x_t^i)\prod_{t'=1}^{T_i}\exp(\boldsymbol{f}(y_{t'},y_{t'-1},x_{t'}^i)^T\boldsymbol{\lambda})$$

$$\begin{aligned}=&\frac{\lambda_k}{\sigma^2}-\sum_{i=1}^{N}f_k(\boldsymbol{y}^i,\boldsymbol{x}^i)+\\ &\sum_{i=1}^{N}\frac{1}{Z(\boldsymbol{x}^i)}\sum_{t=1}^{T_i}\sum_{y_{t-1},y_t}f_k(y_t,y_{t-1},x_t^i)\sum_{y_1,y_2,\ldots,y_{t-2},y_{t+1},\ldots,y_{T_i}}\prod_{t'=1}^{T_i}\exp(\boldsymbol{f}(y_{t'},y_{t'-1},x_{t'}^i)^T\boldsymbol{\lambda}).\end{aligned}$$

(3.17)

Define a new function Ψ to simplify the exponential term:

$$\Psi(y_t,y_{t-1},x_t^i)=\exp(\boldsymbol{f}(y_t,y_{t-1},x_t^i)^T\boldsymbol{\lambda}).\tag{3.18}$$

The last summation of (3.17) can be split to three terms. Given t, y_{t-1}, y_t:

$$\sum_{y_1,y_2,\ldots,y_{t-2},y_{t+1},\ldots,y_{T_i}} \prod_{t'=1}^{T_i} \exp(\boldsymbol{f}(y_{t'}, y_{t'-1}, x_{t'}^i)^T \boldsymbol{\lambda}) \tag{3.19}$$

$$= \Psi(y_t, y_{t-1}, x_t^i) \sum_{y_1,y_2,\ldots,y_{t-2},y_{t+1},\ldots,y_{T_i}} \left(\prod_{t'=1}^{t-1} \Psi(y_{t'}, y_{t'-1}, x_{t'}^i) \right) \left(\prod_{t'=t+1}^{T_i} \Psi(y_{t'}, y_{t'-1}, x_{t'}^i) \right)$$

$$= \Psi(y_t, y_{t-1}, x_t^i) \left(\sum_{y_1,y_2,\ldots,y_{t-2}} \prod_{t'=1}^{t-1} \Psi(y_{t'}, y_{t'-1}, x_{t'}^i) \right) \left(\sum_{y_{t+1},y_{t+2},\ldots,y_{T_i}} \prod_{t'=t+1}^{T_i} \Psi(y_{t'}, y_{t'-1}, x_{t'}^i) \right). \tag{3.20}$$

The second term considers all possible combinations of tag sequences with length $t-1$ which end at a fixed tag y_{t-1}. In these sequences, there are $|Y_D|$ possible tags at position $t-2$, i.e., y_{t-2}. We can split all sequences to $|Y_D|$ cases according to y_{t-2}, and combine them to the desired solution. The detailed procedure can be shown by a recursive formula.

Define a function, $\alpha_t^i(y_t)$ on a given data $\boldsymbol{x}^i$:

$$\begin{aligned} \alpha_t^i(y) &\equiv \sum_{y_1,y_2,\ldots,y_{t-1} \in Y_D, y_t = y} \prod_{t'=1}^{t} \Psi(y_{t'}, y_{t'-1}, x_{t'}^i) \\ &= \sum_{y_{t-1} \in Y_D} \Psi(y, y_{t-1}, x_t^i) \sum_{y_1,y_2,\ldots,y_{t-2} \in Y_D} \prod_{t'=1}^{t-1} \Psi(y_{t'}, y_{t'-1}, x_{t'}^i) \\ &= \sum_{y_{t-1} \in Y_D} \Psi(y, y_{t-1}, x_t^i) \alpha_{t-1}^i(y_{t-1}). \end{aligned} \tag{3.21}$$

By the definition, $\alpha_{t-1}^i(y_{t-1})$ is the second term of (3.20). The third term of (3.20) has a similar recursive formula. Define

$$\begin{aligned} \beta_t^i(y) &\equiv \sum_{y_t = y, y_{t+1}, y_{t+2}, \ldots, y_{T_i} \in Y_D} \prod_{t'=t+1}^{T_i} \Psi(y_{t'}, y_{t'-1}, x_{t'}^i) \\ &= \sum_{y_{t+1} \in Y_D} \Psi(y_{t+1}, y, x_{t+1}^i) \sum_{y_{t+2}, y_{t+3}, \ldots, y_{T_i} \in Y_D} \prod_{t'=t+2}^{T_i} \Psi(y_{t'}, y_{t'-1}, x_{t'}^i) \\ &= \sum_{y_{t+1} \in Y_D} \Psi(y_{t+1}, y, x_{t+1}^i) \beta_{t+1}^i(y_{t+1}). \end{aligned} \tag{3.22}$$

And $\beta_t^i(y_t)$ is the third term of (3.20).

We set the initial conditions as

$$\alpha_0^i(y) = 1, \qquad \forall y \in Y_D$$

$$\beta_{T_i+1}^i(y) = 1, \qquad \forall y \in Y_D.$$

If all $\alpha_t^i(y_t)$ and $\beta_t^i(y_t)$ are available and i, t, y_t, y_{t-1} are given, we can directly have

$$(3.20)$$

$$= \alpha_{t-1}^i(y_{t-1})\Psi(y_t, y_{t-1}, x_t^i)\beta_t^i(y_t). \tag{3.23}$$

We also have

$$Z(\boldsymbol{x}^i) = \sum_{y \in Y_D} \alpha_{T_i}^i(y) = \sum_{y \in Y_D} \beta_1^i(y). \tag{3.24}$$

Thus we can use (3.20), (3.23) and (3.24) to calculate the gradient. The detailed procedure is given in Algorithm 3.

We analyze the total time complexity of this procedure. To calculate each $\alpha_t^i(y_t)$ or $\beta_t^i(y_t)$, it takes $O(|Y_D|)$ by the recursive formula. There are $T_i|Y_D|$ elements in $\alpha_t^i(y_t)$ and $\beta_t^i(y_t)$, so building the dynamic table needs $O(T_i|Y_D|^2)$. $Z(\boldsymbol{x}^i)$ takes $O(|Y_D|)$, and (3.19) takes $O(1)$ by (3.23), if we already have every value of $\alpha_t^i(y_t)$ and $\beta_t^i(y_t)$. There are N training sentences in the data. Totally, the gradient of λ_i needs $O(T_i|Y_D|^2 dN)$ time complexity.

For the space complexity, $\alpha_t^i(y_t)$ and $\beta_t^i(y_t)$ both have $O(T_i|Y_D|^2)$ fields. $\Psi(y_t, y_{t-1}, x_t^i)$ needs $O(T_i|Y_D|^2 N)$ spaces. The total space complexity to calculate the gradient is $O(T_i|Y_D|^2 N)$.

Algorithm 3 Calculating gradient of (3.1) according to (3.17)

1. Given

$$\{\boldsymbol{x}^i, \boldsymbol{y}^i\}_{i=1}^N, \boldsymbol{\lambda}, \boldsymbol{g} = \frac{\boldsymbol{\lambda}}{\sigma^2}.$$

2. For $i = 1, 2, \ldots, N$

   - Calculate and store all

$$\Psi(y_t, y_{t-1}, x_t^i) = \exp(\boldsymbol{f}(y_t, y_{t-1}, x_t^i)^T \boldsymbol{\lambda}), \qquad \forall y_t, y_{t-1} \in Y_D, 1 \le t \le T_i.$$

   - Use (3.21), (3.22) and (3.24) to calculate

$$\alpha_t^i(y_t), \beta_t^i(y_t), \text{ and } Z(\boldsymbol{x}^i) \qquad \forall 1 \le i \le N, 1 \le t \le T_i.$$

   - For $t = 1, 2, \ldots, T_i$
     - For $k = 1, 2, \ldots, d$, where $f_k(y_t, y_{t-1}, x_t^i) = 1$

$$g_k \leftarrow g_k - 1.$$

   - For $t = 1, 2, \ldots, T_i$
     - For $y_{t-1}, y_t \in Y_D$
       * For $k = 1, 2, \ldots, d$, where $f_k(y_t, y_{t-1}, x_t^i) = 1$

$$g_k \leftarrow g_k + \alpha_{t-1}^i(y_{t-1}) \frac{\Psi(y_t, y_{t-1}, x_t^i)}{Z(\boldsymbol{x}^i)} \beta_t^i(y_t).$$

3. Return gradient vector $\boldsymbol{g}$.

CHAPTER IV

Dynamic Programming Formula for Hessian-vector Product

The second derivative of CRF is

$$\begin{aligned}
&\frac{\partial^2}{\partial\lambda_k\partial\lambda_{k'}}L(\boldsymbol{\lambda}) \\
=&\frac{I(k=k')}{\sigma^2}+\sum_{i=1}^{N}\sum_{\boldsymbol{y}\in Y_D^{T_i}} f_k(\boldsymbol{y},\boldsymbol{x}^i)f_{k'}(\boldsymbol{y},\boldsymbol{x}^i)\frac{\exp(\boldsymbol{f}(\boldsymbol{y},\boldsymbol{x}^i)^T\boldsymbol{\lambda})}{Z(\boldsymbol{x}^i)} \\
&-\sum_{i=1}^{N}(\sum_{\boldsymbol{y}\in Y_D^{T_i}} f_k(\boldsymbol{y},\boldsymbol{x}^i)\frac{\exp(\boldsymbol{f}(\boldsymbol{y},\boldsymbol{x}^i)^T\boldsymbol{\lambda})}{Z(\boldsymbol{x}^i)})(\sum_{\boldsymbol{y}\in Y_D^{T_i}} f_{k'}(\boldsymbol{y},\boldsymbol{x}^i)\frac{\exp(\boldsymbol{f}(\boldsymbol{y},\boldsymbol{x}^i)^T\boldsymbol{\lambda})}{Z(\boldsymbol{x}^i)}) \qquad (4.1)
\end{aligned}$$

There are d^2 elements in the Hessian matrix. But in TRON, only d elements of the Hessian-vector product are needed. The kth element is

$$(\boldsymbol{H}\boldsymbol{v})_k=\sum_{k'=1}^{d}\frac{\partial^2}{\partial\lambda_k\partial\lambda_{k'}}L(\boldsymbol{\lambda})v_{k'}.$$

In previous research works, AD is applied to calculate the Hessian-vector product. This thesis provides another dynamic programming approach.

The same as in calculating gradient of CRFs, a dynamic programming technique can be used to calculate all the k elements in polynomial time.

Define two new functions

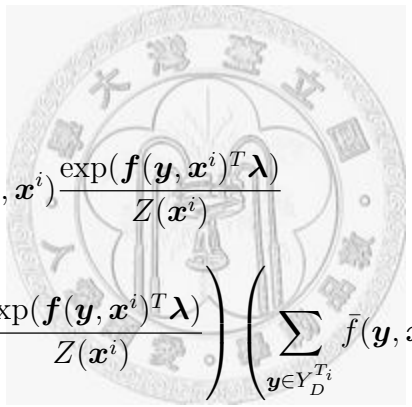
$$\bar{f}(\boldsymbol{y}, \boldsymbol{x}^i) \equiv \sum_{k'=1}^{d} f_{k'}(\boldsymbol{y}, \boldsymbol{x}^i) v_{k'}, \text{ and}$$
$$\bar{f}(y_t, y_{t-1}, x_t^i) \equiv \sum_{k'=1}^{d} f_{k'}(y_t, y_{t-1}, x_t^i) v_{k'}.$$

By the properties of $\boldsymbol{f}(\boldsymbol{y}, \boldsymbol{x}^i)$,

$$\begin{aligned}\bar{f}(\boldsymbol{y}, \boldsymbol{x}^i) &= \sum_{k'=1}^{d} f_{k'}(\boldsymbol{y}, \boldsymbol{x}^i) v_{k'} = \sum_{k'=1}^{d} \sum_{t=1}^{T_i} f_{k'}(y_t, y_{t-1}, x_t^i) v_{k'} \\ &= \sum_{t=1}^{T_i} \bar{f}(y_t, y_{t-1}, x_t^i). \end{aligned} \tag{4.2}$$

With (4.1) and (4.2),

$$\begin{aligned}&(\boldsymbol{H}\boldsymbol{v})_k \\ =\ & \frac{v_k}{\sigma^2} \\ &+ \sum_{i=1}^{N} \sum_{\boldsymbol{y} \in Y_D^{T_i}} f_k(\boldsymbol{y}, \boldsymbol{x}^i) \bar{f}(\boldsymbol{y}, \boldsymbol{x}^i) \frac{\exp(\boldsymbol{f}(\boldsymbol{y}, \boldsymbol{x}^i)^T \boldsymbol{\lambda})}{Z(\boldsymbol{x}^i)} \qquad (4.3) \\ &- \sum_{i=1}^{N} \left(\sum_{\boldsymbol{y} \in Y_D^{T_i}} f_k(\boldsymbol{y}, \boldsymbol{x}^i) \frac{\exp(\boldsymbol{f}(\boldsymbol{y}, \boldsymbol{x}^i)^T \boldsymbol{\lambda})}{Z(\boldsymbol{x}^i)} \right) \left(\sum_{\boldsymbol{y} \in Y_D^{T_i}} \bar{f}(\boldsymbol{y}, \boldsymbol{x}^i) \frac{\exp(\boldsymbol{f}(\boldsymbol{y}, \boldsymbol{x}^i)^T \boldsymbol{\lambda})}{Z(\boldsymbol{x}^i)} \right). \qquad (4.4)\end{aligned}$$



The first term can be easily calculated. Subsequently, we describe details of calculating the second and the third terms.

4.1 Calculation of (4.3)

We define two new symbols.

$$\bar{\alpha}_t^i(y) \tag{4.5}$$

$$\equiv \sum_{\boldsymbol{y}=(y_1,y_2,\ldots,y_{t-1}\in Y_D,\text{ and } y_t=y)} \bar{f}(\boldsymbol{y},\boldsymbol{x}^i)\exp(\boldsymbol{f}(\boldsymbol{y},\boldsymbol{x}^i)^T\boldsymbol{\lambda})$$

$$= \sum_{y_1,y_2,\ldots,y_{t-1}\in Y_D,\text{ and } y_t=y} \left(\sum_{j=1}^{t}\bar{f}(y_j,y_{j-1},x_j^i)\right)\left(\prod_{t'=1}^{t}\exp(\boldsymbol{f}(y_{t'},y_{t'-1},x_{t'}^i)^T\boldsymbol{\lambda})\right). \tag{4.6}$$

$$\bar{\beta}_t^i(y) \tag{4.7}$$

$$\equiv \sum_{\boldsymbol{y}=(y_t=y\text{ and }y_{t+1},y_{t+2},\ldots,y_{T_i}\in Y_D)} \bar{f}(\boldsymbol{y},\boldsymbol{x}^i)\exp(\boldsymbol{f}(\boldsymbol{y},\boldsymbol{x}^i)^T\boldsymbol{\lambda})$$

$$= \sum_{y_t=y,\text{ and }y_{t+1},y_{t+2},\ldots,y_{T_i}\in Y_D} \left(\sum_{j=t}^{T_i}\bar{f}(y_{j+1},y_j,x_{j+1}^i)\right)\left(\prod_{t'=t}^{T_i}\exp(\boldsymbol{f}(y_{t'+1},y_{t'},x_{t'+1}^i)^T\boldsymbol{\lambda})\right). \tag{4.8}$$

Using these new notations and properties, for any given i, (4.3) can be derived to a new form,

$$
\begin{aligned}
&\sum_{\boldsymbol{y}\in Y_D^{T_i}} f_k(\boldsymbol{y},\boldsymbol{x}^i)\bar{f}(\boldsymbol{y},\boldsymbol{x}^i)\frac{\exp(\boldsymbol{f}(\boldsymbol{y},\boldsymbol{x}^i)^T\boldsymbol{\lambda})}{Z(\boldsymbol{x}^i)} \\
=&\frac{1}{Z(\boldsymbol{x}^i)}\sum_{\boldsymbol{y}\in Y_D^{T_i}}\left(\sum_{t=1}^{T_i} f_k(y_t,y_{t-1},x_t^i)\right)\bar{f}(\boldsymbol{y},\boldsymbol{x}^i)\exp(\boldsymbol{f}(\boldsymbol{y},\boldsymbol{x}^i)^T\boldsymbol{\lambda}) \\
=&\frac{1}{Z(\boldsymbol{x}^i)}\sum_{t=1}^{T_i}\sum_{y_{t-1},y_t\in Y_D} f_k(y_t,y_{t-1},x_t^i)\sum_{\substack{\boldsymbol{y}\in\{\boldsymbol{y}|\boldsymbol{y}\in {Y_D}^{T_i},\\ y_{t-1}=\bar{y}_{t-1},y_t=\bar{y}_t\}}}\bar{f}(\boldsymbol{y},\boldsymbol{x}^i)\exp(\boldsymbol{f}(\boldsymbol{y},\boldsymbol{x}^i)^T\boldsymbol{\lambda}) \\
=&\frac{1}{Z(\boldsymbol{x}^i)}\sum_{t=1}^{T_i}\sum_{y_{t-1},y_t} f_k(y_t,y_{t-1},x_t^i)\sum_{y_1,y_2,\ldots,y_{t-2},y_{t+1},\ldots,y_{T_i}} \\
&(\sum_{j=1}^{t-1}\bar{f}(y_j,y_{j-1},x_j^i)+\bar{f}(y_t,y_{t-1},x_t^i)+\sum_{j=t+1}^{T_i}\bar{f}(y_j,y_{j-1},x_j^i)) \\
&\left(\prod_{j=1}^{t-1}\exp(\boldsymbol{f}(y_j,y_{j-1},x_j^i)^T\boldsymbol{\lambda})\right)\exp(\boldsymbol{f}(y_t,y_{t-1},x_t^i)^T\boldsymbol{\lambda})\left(\prod_{j=t+1}^{T_i}\exp(\boldsymbol{f}(y_j,y_{j-1},x_j^i)^T\boldsymbol{\lambda})\right) \\
=&\frac{1}{Z(\boldsymbol{x}^i)}\sum_{t=1}^{T_i}\sum_{y_{t-1},y_t} f_k(y_t,y_{t-1},x_t^i)\exp(\boldsymbol{f}(y_t,y_{t-1},x_t^i)^T\boldsymbol{\lambda})\sum_{y_1,y_2,\ldots,y_{t-2},y_{t+1},\ldots,y_{T_i}} \\
&(\sum_{j=1}^{t-1}\bar{f}(y_j,y_{j-1},x_j^i)+\bar{f}(y_t,y_{t-1},x_t^i)+\sum_{j=t+1}^{T_i}\bar{f}(y_j,y_{j-1},x_j^i)) \\
&\left(\prod_{j=1}^{j=t-1}\exp(\boldsymbol{f}(y_j,y_{j-1},x_j^i)^T\boldsymbol{\lambda})\right)\left(\prod_{j=t+1}^{T_i}\exp(\boldsymbol{f}(y_j,y_{j-1},x_j^i)^T\boldsymbol{\lambda})\right) \\
=&\frac{1}{Z(\boldsymbol{x}^i)}\sum_{t=1}^{T_i}\sum_{y_{t-1},y_t} f_k(y_t,y_{t-1},\boldsymbol{x}^i)\exp(\boldsymbol{f}(y_t,y_{t-1},x_t^i)^T\boldsymbol{\lambda}) \\
&(\bar{\alpha}_{t-1}^i(y_{t-1})\beta_t^i(y_t)+\bar{f}(y_t,y_{t-1},x_t^i)\alpha_{t-1}^i(y_{t-1})\beta_t^i(y_t)+\alpha_{t-1}^i(y_{t-1})\bar{\beta}_t^i(y_t)). \qquad (4.9)
\end{aligned}
$$

To calculate (4.9), we need to obtain $\bar{\alpha}^i$ and $\bar{\beta}^i$ in advance.

As α^i, $\bar{\alpha}^i$ has a recursive formula.

$$
\begin{aligned}
&\bar{\alpha}_t^i(y) \\
&= \sum_{y_1,y_2,\ldots,y_{t-1},y_t=y} \left(\sum_{j=1}^{t} \bar{f}(y_j, y_{j-1}, x_j^i) \right) \left(\prod_{t'=1}^{t} \exp(\boldsymbol{f}(y_{t'}, y_{t'-1}, x_{t'}^i)^T \boldsymbol{\lambda}) \right) \\
&= \sum_{y_{t-1},y_t=y} \sum_{y_1,y_2,\ldots,y_{t-2}} \left(\sum_{j=1}^{t} \bar{f}(y_j, y_{j-1}, x_j^i) \right) \left(\prod_{t'=1}^{t} \exp(\boldsymbol{f}(y_{t'}, y_{t'-1}, x_{t'}^i)^T \boldsymbol{\lambda}) \right) \\
&= \sum_{y_{t-1},y_t=y} \exp(\boldsymbol{f}(y_t, y_{t-1}, x_t^i)^T \boldsymbol{\lambda}) \sum_{y_1,y_2,\ldots,y_{t-2}} \\
&\quad (\bar{f}(y_t, y_{t-1}, x_t^i) + \sum_{j=1}^{t-1} \bar{f}(y_j, y_{j-1}, x_{j-1}^i)) \left(\prod_{t'=1}^{t-1} \exp(\boldsymbol{f}(y_{t'}, y_{t'-1}, x_{t'}^i)^T \boldsymbol{\lambda}) \right) \\
&= \sum_{y_{t-1},y_t=y} \exp(\boldsymbol{f}(y_t, y_{t-1}, x_t^i)^T \boldsymbol{\lambda}) (\bar{f}(y_t, y_{t-1}, x_t^i) \alpha_{t-1}^i(y_{t-1}) + \bar{\alpha}_{t-1}^i(y_{t-1})). \qquad (4.10)
\end{aligned}
$$

Similarly,

$$
\begin{aligned}
&\bar{\beta}_t^i(y) \\
&= \sum_{y_t=y,y_{t+1},y_{t+2},\ldots,y_{T_i} \in Y_D} \left(\sum_{j=t}^{T_i} \bar{f}(y_{j+1}, y_j, x_j^i) \right) \left(\prod_{t'=t}^{T_i} \exp(\boldsymbol{f}(y_{t'+1}, y_{t'}, x_{t'+1}^i)^T \boldsymbol{\lambda}) \right) \\
&= \sum_{y_t=y,y_{t+1}} \sum_{y_{t+2},y_{t+3},\ldots,y_{T_i}} \left(\sum_{j=t}^{T_i} \bar{f}(y_{j+1}, y_j, x_{j+1}^i) \right) \left(\prod_{t'=t}^{T_i} \exp(\boldsymbol{f}(y_{t'+1}, y_{t'}, x_{t'+1}^i)^T \boldsymbol{\lambda}) \right) \\
&= \sum_{y_t=y,y_{t+1}} \exp(\boldsymbol{f}(y_{t+1}, y_t, x_{t+1}^i)^T \boldsymbol{\lambda}) \sum_{y_{t+2},y_{t+3},\ldots,y_{T_i}} \\
&\quad (\bar{f}(y_{t+1}, y_t, x_{t+1}^i) + \sum_{j=t+1}^{T_i} \bar{f}(y_{j+1}, y_j, x_{j+1}^i)) \left(\prod_{t'=t+1}^{T_i} \exp(\boldsymbol{f}(y_{t'+1}, y_{t'}, x_{t'+1}^i)^T \boldsymbol{\lambda}) \right) \\
&= \sum_{y_t=y,y_{t+1}} \exp(\boldsymbol{f}(y_{t+1}, y_t, x_{t+1}^i)^T \boldsymbol{\lambda}) (\bar{f}(y_{t+1}, y_t, x_{t+1}^i) \beta_{t+1}^i(y_{t+1}) + \bar{\beta}_{t+1}^i(y_{t+1})). \qquad (4.11)
\end{aligned}
$$

To conduct a recursive calculation, we set the initial values of $\bar{\alpha}_0^i$ and $\bar{\beta}_{T_i+1}^i$ as

$$\bar{\alpha}_0^i(y) = 0, \quad \forall y \in Y_D,$$

$$\bar{\beta}_{T_i+1}^i(y) = 0. \quad \forall y \in Y_D.$$

Algorithm 4 Using $\boldsymbol{v}$ to calculate $\bar{f}(y_t, y_{t-1}, x_t^i)$, $\forall 1 \le t \le T_i$, $y_{t-1}, y_t \in Y_D$

1. For $t = 1, 2, \ldots, T_i$
   - Initialize $\bar{f}(y_t, y_{t-1}, x_t^i) \leftarrow 0$ in memory.
   - For $y_{t-1}, y_t \in Y_D$
     - For $k \in \{1, 2, \ldots, d\}$ with $f_k(y_t, y_{t-1}, x_t^i) = 1$

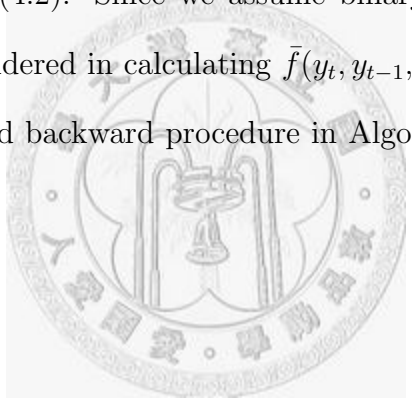
$$\bar{f}(y_t, y_{t-1}, x_t^i) \leftarrow \bar{f}(y_t, y_{t-1}, x_t^i) + f_k(y_t, y_{t-1}, x_t^i) v_k$$

In (4.9), (4.10) and (4.11), we need

$$\bar{f}(y_t, y_{t-1}, x_t^i), \quad \forall y_t, y_{t-1} \in Y_D, t = 1, 2, \ldots, d.$$

They must be calculated and stored. Algorithm 4 describes the procedure to calculate $\bar{f}(y_t, y_{t-1}, x_t^i)$ using (4.2). Since we assume binary feature values, only those $f_k(y_t, y_{t-1}, x_t^i) = 1$ are considered in calculating $\bar{f}(y_t, y_{t-1}, x_t^i)$.

We give the forward and backward procedure in Algorithm 5 to show how to calculate $\bar{\alpha}^i$, $\bar{\beta}^i$.



Algorithm 5 Forward and backward procedure to calculate $\bar{\alpha}_t^i(y)$ and $\bar{\beta}_t^i(y)$ by (4.10) and (4.11)

1. For $y \in Y_D$

$$\bar{\alpha}_0^i(y) = 0 \text{ and } \bar{\beta}_{T_i+1}^i(y) = 0$$

2. For $t = 1, 2, \ldots, T_i$

   - For $y_{t-1}, y_t \in Y_D$
     - Use (4.10) to calculate $\bar{\alpha}_t^i(y)$, where

$$\Psi(y_t, y_{t-1}, x_t^i) = \exp(\boldsymbol{f}(y_t, y_{t-1}, x_t^i)^T \boldsymbol{\lambda})$$

       is calculated in Algorithm 3 and stored in memory.

$$\bar{\alpha}_t^i(y_t) \leftarrow \bar{\alpha}_t^i(y_t) + \Psi(y_t, y_{t-1}, x_t^i)(\bar{f}(y_t, y_{t-1}, x_t^i)\alpha_{t-1}^i(y_{t-1}) + \bar{\alpha}_{t-1}^i(y_{t-1}))$$

3. For $t = T_i, T_i - 1, \ldots, 2, 1$

   - For $y_{t-1}, y_t \in Y_D$
     - Use (4.11) to calculate $\bar{\beta}_t^i(y)$, where

$$\Psi(y_{t+1}, y_t, x_{t+1}^i) = \exp(\boldsymbol{f}(y_{t+1}, y_t, x_t^i)^T \boldsymbol{\lambda})$$

       is calculated in Algorithm 3 and stored in memory.

$$\bar{\beta}_t^i(y_t) \leftarrow \bar{\beta}_t^i(y_t) + \Psi_{t+1}(y_{t+1}, y_t, x_{t+1}^i)(\bar{f}(y_{t+1}, y_t, x_{t+1}^i)\beta_{t+1}^i(y_{t+1}) + \bar{\beta}_{t+1}^i(y_{t+1}))$$

4.2 Calculation of (4.4)

Eq. (4.4) can be seen as the multiplication of two terms, where one is

$$\sum_{\boldsymbol{y}\in {Y_D}^{T_i}} f_k(\boldsymbol{y},\boldsymbol{x}^i)\frac{\exp(\boldsymbol{f}(\boldsymbol{y},\boldsymbol{x}^i)^T\boldsymbol{\lambda})}{Z(\boldsymbol{x}^i)}$$

$$=\frac{1}{Z(\boldsymbol{x}^i)}\sum_{\boldsymbol{y}\in Y_D^{T_i}}\sum_{t=1}^{T_i} f_k(y_t,y_{t-1},x_t^i)\prod_{j=1}^{T_i}\exp(\boldsymbol{f}(y_j,y_{j-1},x_j^i)^T\boldsymbol{\lambda}) \tag{4.12}$$

$$=\frac{1}{Z(\boldsymbol{x}^i)}\sum_{t=1}^{T_i}\sum_{y_{t-1},y_t} f_k(y_t,y_{t-1},x_t^i)\sum_{y_1,y_2,\ldots,y_{t-2},y_{t+1},\ldots,y_{T_i}}\prod_{j=1}^{T_i}\exp(\boldsymbol{f}(y_j,y_{j-1},x_j^i)^T\boldsymbol{\lambda})$$

$$=\frac{1}{Z(\boldsymbol{x}^i)}\sum_{t=1}^{T_i}\sum_{y_{t-1},y_t} f_k(y_t,y_{t-1},x_t^i)\exp(\boldsymbol{f}(y_t,y_{t-1},x_t^i)^T\boldsymbol{\lambda})$$

$$\left(\sum_{y_1,y_2,\ldots,y_{t-2}}\prod_{j=1}^{t-1}\exp(\boldsymbol{f}(y_j,y_{j-1},x_j^i)^T\boldsymbol{\lambda})\right)\left(\sum_{y_{t+1},y_{t+2},\ldots,y_{T_i}}\prod_{j=t+1}^{T_i}\exp(\boldsymbol{f}(y_j,y_{j-1},x_j^i)^T\boldsymbol{\lambda})\right)$$

$$=\frac{1}{Z(\boldsymbol{x}^i)}\sum_{t=1}^{T_i}\sum_{y_{t-1},y_t} f_k(y_t,y_{t-1},x_t^i)\exp(\boldsymbol{f}(y_t,y_{t-1},x_t^i)^T\boldsymbol{\lambda})\alpha_{t-1}^i(y_{t-1})\beta_t^i(y_t). \tag{4.13}$$

The other is

$$\sum_{\boldsymbol{y}\in {Y_D}^{T_i}} \bar{f}(\boldsymbol{y},\boldsymbol{x}^i)\frac{\exp(\boldsymbol{f}(\boldsymbol{y},\boldsymbol{x}^i)^T\boldsymbol{\lambda})}{Z(\boldsymbol{x}^i)}$$

$$=\frac{1}{Z(\boldsymbol{x}^i)}\sum_{t=1}^{T_i}\sum_{y_{t-1},y_t} \bar{f}(y_t,y_{t-1},x_t^i)\sum_{y_1,y_2,\ldots,y_{t-2},y_{t+1},\ldots,y_{T_i}}\prod_{j=1}^{T_i}\exp(\boldsymbol{f}(y_j,y_{j-1},x_j^i)^T\boldsymbol{\lambda})$$

$$=\frac{1}{Z(\boldsymbol{x}^i)}\sum_{t=1}^{T_i}\sum_{y_{t-1},y_t} \bar{f}(y_t,y_{t-1},x_t^i)\exp(\boldsymbol{f}(y_t,y_{t-1},x_t^i)^T\boldsymbol{\lambda})$$

$$\left(\sum_{y_1,y_2,\ldots,y_{t-2}}\prod_{j=1}^{t-1}\exp(\boldsymbol{f}(y_j,y_{j-1},x_j^i))\right)\left(\sum_{y_{t+1},y_{t+2},\ldots,y_{T_i}}\prod_{l=t+1}^{T_i}\exp(\boldsymbol{f}(y_l,y_{l-1},x_l^i)^T\boldsymbol{\lambda})\right)$$

$$=\frac{1}{Z(\boldsymbol{x}^i)}\sum_{t=1}^{T_i}\sum_{y_{t-1},y_t} \bar{f}(y_t,y_{t-1},x_t^i)\exp(\boldsymbol{f}(y_t,y_{t-1},x_t^i)^T\boldsymbol{\lambda})\alpha_{t-1}^i(y_{t-1})\beta_t^i(y_t). \tag{4.14}$$

We give the procedure to calculate (4.13) and (4.14) in Algorithm 6.

Algorithm 6 Calculating (4.4) by (4.13) and (4.14)

1. Fix i.
2. Initialize
$$\boldsymbol{p} \leftarrow \mathbf{0}, w \leftarrow 0$$
3. Calculate (4.13)
   - For $t = 1, 2, \ldots, T_i$
     - For $y_{t-1}, y_t \in Y_D$
       - For $k \in \{1, 2, \ldots, d\}$ with $f_k(y_t, y_{t-1}, x_t^i) = 1$
$$p_k \leftarrow p_k + f_k(y_t, y_{t-1}, x_t^i) \exp(\boldsymbol{f}(y_t, y_{t-1}, x_t^i)^T \boldsymbol{\lambda}) \alpha_{t-1}^i(y_{t-1}) \beta_t^i(y_t).$$
4. $\boldsymbol{p} \leftarrow \boldsymbol{p}/Z(\boldsymbol{x}^i)$.
5. Calculate (4.14)
   - For $t = 1, 2, \ldots, T_i$
     - For $y_{t-1}, y_t \in Y_D$
$$w \leftarrow w + \bar{f}(y_t, y_{t-1}, x_t^i) \exp(\boldsymbol{f}(y_t, y_{t-1}, x_t^i)^T \boldsymbol{\lambda}) \alpha_{t-1}^i(y_{t-1}) \beta_t^i(y_t).$$
6. $w \leftarrow w/Z(\boldsymbol{x}^i)$

4.3 Overall Procedure and Time/Memory Analysis

Using Algorithm 4, Algorithm 5 and Algorithm 6, the procedure to calculate the Hessian-vector product is in Algorithm 7.

We discuss the time complexity of Algorithm 7. To calculate $\bar{f}(y_t, y_{t-1}, x_t^i)$ in Algorithm 4, t is from 1 to T_i, y_{t-1} and y_t both have $|Y_D|$ choices, and k is from 1 to d. The time complexity is $O(T_i|Y_D|^2 d)$. With a similar analysis, the forward and backward procedure in Algorithm 5 needs $O(T_i|Y_D|^2)$. Calculating (4.9) needs $O(T_i|Y_D|^2)$ under fixed k and i, so the cost over all k is $O(T_i|Y_D|^2 d)$. In Algorithm 6, step 3 needs $O(T_i|Y_D|^2 d)$, and step 5 needs $O(T_i|Y_D|^2)$. By summing the cost over all $i = 1, 2, \ldots, N$, the total cost of Algorithm 7 is $O(T_i|Y_D|^2 dN)$.

To discuss the space complexity, $\bar{f}(y_t, y_{t-1}, x_t^i)$ has $O(T_i|Y_D|^2)$ different values. Similarly, $\Psi(y_t, y_{t-1}, x_t^i)$ needs $O(T_i|Y_D|^2)$ spaces. For $\alpha_t^i(y)$, $\beta_t^i(y)$, $\bar{\alpha}_t^i(y)$ and $\bar{\beta}_t^i(y)$ they all have $O(T_i|Y_D|)$ fields. Totally, the space complexity to calculate the Hessian-vector product is $O(T_i|Y_D|^2)$.

Algorithm 7 Use (4.9) to calculate Hessian vector products (3.13), (3.15) in conjugate gradient subroutine

1. Given

$$\{\boldsymbol{x}^i, \boldsymbol{y}^i\}_{i=1}^N, \boldsymbol{\lambda}, \boldsymbol{v}, \boldsymbol{Hv} = \frac{\boldsymbol{v}}{\sigma^2}$$

2. For $i = 1, 2, \ldots, N$

   - Use $\boldsymbol{v}$ to calculate $\bar{f}(y_t, y_{t-1}, x_t^i), 1 \le t \le T_i, y_{t-1}, y_t \in Y_D$ by Algorithm 4
   - Calculate $\bar{\alpha}_t^i(y_t), t = 0, 1, \ldots, T_i$ and $\bar{\beta}_t^i(y_t), t = 0, 1, \ldots, T_i$ by Algorithm 5.
   - Initialize $\boldsymbol{d}^i \leftarrow \boldsymbol{0}$.
   - For $t = 1, 2, \ldots, T_i$
     - For $y_{t-1}, y_t \in Y_D$
       - For $k = 1, 2, \ldots, d$, where $f_k(y_t, y_{t-1}, x_t^i) = 1$

$$d_k^i \leftarrow d_k^i + f_k(y_t, y_{t-1}, x_t^i)\Psi(y_t, y_{t-1}, x_t^i)$$
$$(\bar{\alpha}_{t-1}^i(y_{t-1})\beta_t^i(y_t) + \bar{f}(y_t, y_{t-1}, x_t^i)\alpha_{t-1}^i(y_{t-1})\beta_t^i(y_t)) + \alpha_{t-1}^i(y_{t-1})\bar{\beta}_t^i(y_t))$$

   - $\boldsymbol{d}^i \leftarrow \boldsymbol{d}^i / Z(\boldsymbol{x}^i)$
   - Initialize

$$\boldsymbol{p} \leftarrow \boldsymbol{0}, w \leftarrow 0$$

   - Calculate (4.13)
     - For $t = 1, 2, \ldots, T_i$
       - For $y_{t-1}, y_t \in Y_D$
         - For $k \in \{1, 2, \ldots, d\}$ with $f_k(y_t, y_{t-1}, x_t^i) = 1$

$$p_k \leftarrow p_k + f_k(y_t, y_{t-1}, x_t^i) \exp(\boldsymbol{f}(y_t, y_{t-1}, x_t^i)^T \boldsymbol{\lambda}) \alpha_{t-1}^i(y_{t-1}) \beta_t^i(y_t).$$

   - $\boldsymbol{p} \leftarrow \boldsymbol{p} / Z(\boldsymbol{x}^i)$.
   - Calculate (4.14)
     - For $t = 1, 2, \ldots, T_i$
       - For $y_{t-1}, y_t \in Y_D$

$$w \leftarrow w + \bar{f}(y_t, y_{t-1}, x_t^i) \exp(\boldsymbol{f}(y_t, y_{t-1}, x_t^i)^T \boldsymbol{\lambda}) \alpha_{t-1}^i(y_{t-1}) \beta_t^i(y_t).$$

   - $w \leftarrow w / Z(\boldsymbol{x}^i)$
   - $\boldsymbol{Hv} \leftarrow \boldsymbol{Hv} + \boldsymbol{d}^i - w\boldsymbol{p}$

3. Return $\boldsymbol{Hv}$.

CHAPTER V

Conclusions

Conditional random fields are popular in the natural language process area, and have good performances on applications such as labeling sequential data and segmenting sentences. Many optimization methods have been applied to solve CRFs. They focus on gradient based methods and quasi-newton methods. In this thesis we consider a Newton method for training CRFs because of its possible fast final convergence. However, Newton methods need Hessian information which is hard to calculate. We focus on a trust region Newton method using conjugate gradient methods, because it only needs Hessian-vector products.

Previous works show that automatic differentiation can be applied to Hessian-vector product in some optimization methods such as stochastic meta decent methods and trust region Newton methods. This thesis provides another approach to do Hessian-vector product. We derive an analytical Hessian-vector formula of CRFs, and propose a novel dynamic programming technique which has the same time complexity as the gradient calculation.

BIBLIOGRAPHY

A. Griewank. Evaluating derivatives: principles and techniques of algorithmic differentiation. SIAM, 2000.

J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Proceedings of the 18th International Conference on Machine Learning (ICML), pages 282–289, 2001.

C.-J. Lin and J. J. Moré. Newton's method for large-scale bound constrained problems. SIAM Journal on Optimization, 9:1100–1127, 1999.

C.-J. Lin, R. C. Weng, and S. S. Keerthi. Trust region Newton method for large-scale logistic regression. Journal of Machine Learning Research, 9:627–650, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/logistic.pdf`.

D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. Mathematical Programming, 45(1):503–528, 1989.

A. Mccallum and W. Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In Seventh Conference on Natural Language Learning (CoNLL), 2003.

F. Peng, F. Feng, and A. McCalum. Chinese segmentation and new word detection using conditional random fields. In Proceedings of The 20th International Conference on Computational Linguistics (COLING), pages 562–568, 2004.

L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE, 77(2):257–285, 1989.

F. Sha and F. C. N. Pereira. Shallow parsing with conditional random fields. In HLT-NAACL, 2003.

S. Vishwanathan, N. N. Schraudolph, M. W. Schmidt, and K. Murphy. Accelerated training of conditional random fields with stochastic gradient methods. In Proceedings of the 23rd International Conference on Machine Learning (ICML), pages 969–976, 2006.

H.-J. Wang. Applying automatic di entiation and truncated newton methods to conditional random fields. Master's thesis, Department of Computer Science and Information Engineering, National Taiwan University, 2008.