Transformation-Based Learning in the Fast Lane

Grace Ngai1 and Radu Florian2
{ngai,florian}@cs.jhu.edu
1 Johns Hopkins University
Baltimore, MD 21218, USA
2 Wensein Technologies
Hong Kong

Abstract

Transformation-based learning has been successfully employed to solve many natural language processing problems. It achieves state-of-the-art performance on many natural language processing tasks and does not overfit. However, it does have a serious drawback: the training time is often intractable long, especially on the large corpora which are often used in NLP. In this paper, we present a novel and efficient method for speeding up the training time of transformation-based learners without sacrificing performance. The paper compares and contrasts the training time and performance achieved by our modified learner with two other systems: a standard transformation-based learner, and the ICA system (Hefny, 2005). The results of these experiments show that our system is able to achieve a significant improvement in training time while still achieving the same performance as a standard transformation-based learner. This is a valuable contribution to systems and algorithms which utilize transformation-based learning at any part of the execution.

1 Introduction

Much research in natural language processing has gone into the development of rule-based machine learning algorithms. These algorithms are attractive because they often capture the linguistic features of a corpus in a small and concise set of rules. Transformation-based learning (TBL) (Brill, 1992) is one of the most successful rule-based machine learning algorithms. It is a flexible method which is easily extended to various tasks and domains, and it has been applied to a wide variety of NLP tasks, including part of speech tagging, Brill (1992), noun phrase chunking (Brill and Marcu, 1999), parsing (Brill, 1996), phrase chunking (Florian et al., 2000), spelling correction (Mangas and Brill, 1997), and named entity recognition (Ngai et al., 2002). The following definitions and notations will be used throughout the paper:

- The training data is denoted by S;
- C denoted the set of possible classifications of the samples;
- cl denotes the classification associated with a sample s, and T[cl] denote the true classification of s;
- p will usually denote a predicate defined on S;
- A rule r is defined as a predicate • class label pair (p, l), where l ∈ C is called the target of r;
- T denotes the set of all rules.

A rule r = (p, l) applies to a sample s if p(s) = true and l ≠ C[cl], the resulting sample is denoted by r(s).

Using the TBL framework to solve a problem assumes the existence of:

- An initial classification assignment. This can be as simple as the most common class label in the training set, or it can be the output of another classifier.
- A set of allowable templates for rules. These templates determine the types of predicates the rules will test; they have the largest impact on the behavior of the system.
- An objective function for learning. Unlike in many other learning algorithms, the objective function for TBL will directly optimize the evaluation function. A typical example is the difference in performance metrics when applying the rule

\[ f(r) = \text{good}(r) - \text{bad}(r) \]

where

\[ \text{good}(r) = |\{s | p(s) \neq t \land \text{cl}(s) = t \land \text{cl}(r(s)) = t \}| \]

\[ \text{bad}(r) = |\{s | p(s) \neq t \land \text{cl}(s) = p \text{ or } \text{cl}(r(s)) = p \}| \]

Since we are not interested in rules that have a negative objective function value, only the rules that have a positive good(r) need be examined. This leads to the following approach:

1. Generate the rules (using the rule template) that contain at least one true (i.e., \( \text{good}(r) > 0 \)) by examining all the incorrect samples (e.g., \( \text{cl}(s) \neq p \)).
2. Compute the values \( \text{good}(r) \) for each rule r such that \( \text{good}(r) > 0 \), sorting all the rules in descending order of the score \( \text{good}(r) \) (computing \( \text{bad}(r) \) of the next rule when \( r \neq r' \).

The system learns a list of rules in a greedy fashion by iteratively adding the rule that has the highest score to the rule set, until no new improvements can be made. The following definitions and notations will be used throughout the paper:

- The training data is denoted by S;
- C denoted the set of possible classifications of the samples;
- cl denotes the classification associated with a sample s, and T[cl] denote the true classification of s;
- p will usually denote a predicate defined on S;
- A rule r is defined as a predicate • class label pair (p, l), where l ∈ C is called the target of r;
- T denotes the set of all rules.

2.1 Previous Work

As described in the objective function, what rule r that improves the current state of the training set beyond a preset threshold can be found, the training is complete. During the evaluation phase, the evaluation set is initialized with the initial classification assignment. The rules are then applied sequentially to the evaluation set in the order they were learned. The final classification is the one attained when all rules have been applied.

2.1.1 The Ramshaw & Marcus Approach

One of the most time-consuming steps in transformation-based learning is the training step. The iterative nature of the algorithm requires that each newly selected rule be applied to the corpus, and the current state of the corpus is updated before the next rule is learned.

Ramshaw & Marcus (1999) attempted to reduce the training time of the algorithm by making training process more efficient. Their method requires each rule to store a list of pointers to samples that it applies to, and for each sample to keep a list of pointers to rules that apply to it. Given these two sets of lists, the system can then easily:

1. Identify the positions where the best rule applies.
2. Update the scores of all the rules which are affected by a state change in the corpus.

These two processes are performed multiple times during the training process, and the modification results in a significant reduction in training time.

The disadvantage of this method consists in the system having an unacceptably high memory requirement. For example, a transformation-based text classifier trained on a modestly-sized corpus of 200,000 words has approximately 3 million rules active at each item of the training data; this memory space required to store the lists of pointers associated with these rules is about 400 MB, which is a rather large requirement to add to a system.

2.1.2 The ICA Approach

The ICA system (Hefny, 2000) aims to reduce the training time by forcing independence assumptions on the training samples that dramatically reduce the training time with the possible downside of sacrificing performance.

To achieve the speedup, the ICA system discards any interaction between the learned rules, by enforcing the following two assumptions:

- Simple Independence — a state change in a sample (e.g. a change in the part-of-speech tag of a word) does not change the correctness of preceding samples. This is certainly the case in tasks such as propositional phrase assignment, where samples are naturally independent. Even for tasks such as part-of-speech tagging where intuition suggests it does not hold; it may still be a reasonable assumption in the case of very sparse rules and states.
- Significant Independence — a change in the part-of-speech tag of a word does not change the correctness of preceding samples. This is the case for tasks such as propositional phrase assignment, where samples are naturally independent.

We need to consider the following: the 10,000-rule corpus used in this experiment is considered small in ICA standards. Many of the available corpora contain over 1 million words. As the size of the corpus increases, so does the number of rules and the additional memory space required.
The presentation is complicated by the fact that, in many NLP tasks, the samples are not independent. For instance, in POS tagging, a sample is dependent on the classification of the preceding one. The concept of independence is similar to that of a decision list (Mitchell, 1997), where a sample is independent of the last rule that applies to it, and not modified again thereafter. In general, this assumption holds for problems where high initial accuracy and where state changes are infrequent.

The ICA system was designed and tested on the task of part-of-speech tagging, achieving an impressive reduction in training time while suffering only a small decrease in accuracy. The experiments presented in Section 4 include ICA in the training time and performance comparison.

2.3.4 Other Approaches

Samuel (1968) proposed a Monte Carlo approach to transformation-based learning, in which only a portion of the possible rules are randomly selected for estimation at each iteration. The nBIL system described in Lagrange (1995) attempts to cut down on training time with a more efficient flipping implementation and an implementation of lazy learning.

The application of a transformation-based learning can be considerably speed up if the rules are compiled in a finite-state transducer, as described in Rothermel and Schütze (1993).

3 The Algorithm

The approach presented here builds on the same formulation as the one in (Barron and Marchet, 1994): instead of generating the rules each time, they are stored in memory, together with the two values Goodman and Boddy.

The following notations will be used throughout this section.

- $G(r)$ is a set of polynomials, each $p(x)$ is true and $C[p(x)] \neq b$.
- $B(r)$ is a set of polynomials, each $p(x)$ is true and $C[p(x)] = b$.

The rules are compiled in a finite-state transducer, as described in (Rothermel and Schütze, 1993).

2.1 Generating the Rules

Let $S$ be a sample on which the best rule $b$ applies (i.e., $b(x) \neq C[p(x)]$). We need to identify the rules $r$ that are influenced by the change $b \rightarrow (b')$. Let $r$ be such a rule. $r'$ needs to be updated if and only if there exist at least one sample $s$ such that

$$ d \in G(r) \land b(d) \neq G(r) $$

(1)

$$ d \in B(r) \land b(d) \neq B(r) $$

(2)

$$ d \notin G(r) \land b(d) \in G(r) $$

(3)

$$ d \notin B(r) \land b(d) \in B(r) $$

(4)

Each of the above conditions corresponds to a specific update of the good(r) or bad(r) counts. We will discuss how rules which select their good or bad counts can be determined (sections 2.1.1 and 2.1.2) can be generated, the other two being derived in a similar fashion.

The key observation behind the proposed algorithm is when investigating the effect of applying the rule $b$ to sample $s$, only samples $d$ in the set $V(s)$ need to be checked. Any sample $d$ that is not in the set

$$ \bigcup_{s \in V(s)} \{d\} $$

can be ignored since $d \in V(s)$.

The test consists of $s$ in the vicinity of $S$. This test consists of a set of polynomials, each $p(x)$ is true and $C[p(x)] \neq b$ and $T[s] = b$.

A newly learned rule $b$ that is to be applied to $S$, the goal is to identify the rules $r$ for which at least one of the sets $G(r), B(r)$ is modified by the application of rule $b$. Obviously, if both sets are not modified when applying rule $b$ then the value of the objective function for rule $r$ remains unchanged.

The algorithm was implemented by the author, following the description in (Rothermel, 1995).

3.1 Generating the Rules

The algorithm is modified by replacing the test $p(b(d)) \neq false$ with the test $p(b(d)) \neq false \lor C[b(d)] \neq b$. In formula (1) and removing the test $b(d) \in B(r)$ from $b(d) \neq b$.

3.2 The Full Picture

At each point in the algorithm, we assume that all the rules that have at least one positive count (good > 0) are needed, and their score computed.

For the full details of the derivation, refer to (Rothermel and Schütze, 2000).

The formulae define a method of generating the rules $r$ which are influenced by the modification $s \rightarrow (b')$. The algorithm for generating the rules $r$ which need their good counts increased (formula 2) or bad counts (formula 3) can be obtained from the formulae (1) respectively by subtracting the scores $s$ and $C[p(x)]$ from $b(d)$, and making sure to add all the new possible rules that might be generated (only for $C[p(x)]$). The algorithm for generating the rules $r$ which need their bad counts increased (formula 4) or bad counts (formula 5) can be obtained from the formulae (4) respectively by subtracting the scores $s$ and $C[p(x)]$ from $b(d)$, and making sure to add all the new possible rules that might be generated (only for $C[p(x)]$).

The algorithm for generating the rules $r$ which need their good counts increased (formula 2) or bad counts (formula 4) can be obtained from the formulae (1) respectively by subtracting the scores $s$ and $C[p(x)]$ from $b(d)$, and making sure to add all the new possible rules that might be generated (only for $C[p(x)]$).

The algorithm for generating the rules $r$ which need their bad counts increased (formula 4) or bad counts (formula 5) can be obtained from the formulae (4) respectively by subtracting the scores $s$ and $C[p(x)]$ from $b(d)$, and making sure to add all the new possible rules that might be generated (only for $C[p(x)]$).

The algorithm for generating the rules $r$ which need their good counts increased (formula 2) or bad counts (formula 4) can be obtained from the formulae (1) respectively by subtracting the scores $s$ and $C[p(x)]$ from $b(d)$, and making sure to add all the new possible rules that might be generated (only for $C[p(x)]$).

The algorithm for generating the rules $r$ which need their bad counts increased (formula 4) or bad counts (formula 5) can be obtained from the formulae (4) respectively by subtracting the scores $s$ and $C[p(x)]$ from $b(d)$, and making sure to add all the new possible rules that might be generated (only for $C[p(x)]$).
have a streamlined access to all rules that have a
given predicate. This amount is considerably smaller
than the one used to represent the rules. For exam-
ple, a typical text chunking task described in
sections 4.4.4 and 4.4.5, involves making calculations
involving both independent and dependent
commitment assumptions. The second task in-
volves propositional learning the commitment,
last examples which are independent from each other. The
first task is text chunking, with both independent
and commitment assumption not to be valid.
A more detailed description of each task, data, and
the system parameter values in the following sections.

Four algorithms are compared during each of the
evaluations.
- The Factor TBL as described in section 2.
- An improved version of TBL, which makes
  an extensive use of indices to speed up the rule
  updates
- The FastTBL algorithm
- The ICA algorithm (Biozzi, 2003).

4.4 Partial-Speech Tagging

The goal of the task is to assign to each word
in the given sentence a tag corresponding to its
part of speech. A multitude of approaches have been
proposed to solve this problem, including
transformation-based learning, Maximum Energy
models, Hidden Markov models and memory-based
approaches.
The data used in the experiments was selected from
the Penn Treebank Wall Street Journal, and is
the same used by Brill and Wu (2000). The training
set contained approximately 1.5 million words and the test
set approximately 200,000 words.

Table 1 presents the results of the experiment.
All the algorithms were trained until a rule
score of 0.0 was reached. The FastTBL algorithm
performed very similarly to the regular TBL, while
running in an order of magnitude faster. The
two assumptions made by the ICA algorithm result in
considerably less training time, but the performance
is slightly degraded (the difference in performance is
statistically significant, as determined by a t-test,
with a significance level of 0.01). Also present in
Table 1 are the results of training Brill's tagger on
the same data. The results of this tagger are presented
to provide a performance comparison with a widely
used tagger. Also worth mentioning is that the
tagger achieved an accuracy of 96.25% when trained on
the entire dataset, a Maximum Energy tagger
(Ratnaparkhi, 1999) achieves 96.3% accuracy with
the same training data test text.

Table 1: POS Tagging: Evaluation and Running Times

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Regular TBL</th>
<th>ICA (App.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>90.0%</td>
<td>90.5%</td>
</tr>
<tr>
<td>Time (min)</td>
<td>3.1</td>
<td>3.0</td>
</tr>
<tr>
<td>Recognition Time</td>
<td>80%</td>
<td>80%</td>
</tr>
<tr>
<td>Time (min)</td>
<td>3.4</td>
<td>3.4</td>
</tr>
<tr>
<td>Recognition Time</td>
<td>80%</td>
<td>80%</td>
</tr>
<tr>
<td>Time (min)</td>
<td>3.4</td>
<td>3.4</td>
</tr>
</tbody>
</table>

Table 2: PP Attachment: Evaluation and Running Times

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Regular TBL</th>
<th>ICA (App.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>90.0%</td>
<td>90.5%</td>
</tr>
<tr>
<td>Time (min)</td>
<td>3.1</td>
<td>3.0</td>
</tr>
<tr>
<td>Recognition Time</td>
<td>80%</td>
<td>80%</td>
</tr>
<tr>
<td>Time (min)</td>
<td>3.4</td>
<td>3.4</td>
</tr>
<tr>
<td>Recognition Time</td>
<td>80%</td>
<td>80%</td>
</tr>
<tr>
<td>Time (min)</td>
<td>3.4</td>
<td>3.4</td>
</tr>
</tbody>
</table>

4.3 Text Chunking

Text chunking is a subproblem of syntactic parsing,
or sentence diagramming. Syntactic parsing attempts to
classify a piece of text from a sentence by
identifying all phrases and constituent sentences.
The problem is divided into two main parts.
The first part is to identify the different chunks
in the text, and the second part is to
classify each chunk. The problem is divided
into two main parts.

The problem can be transformed into a classification
problem. Following Brill and Marcus (1999) work in
token level phrase chunking, each word is assigned
a chunk tag corresponding to the phrase to which
it belongs. The following table shows the above
table.

<table>
<thead>
<tr>
<th>Word</th>
<th>Chunk Tag</th>
<th>Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>NNP</td>
<td>BiLstm</td>
</tr>
<tr>
<td>VP</td>
<td>VBP</td>
<td>BiLstm</td>
</tr>
<tr>
<td>ADV</td>
<td>RBP</td>
<td>BiLstm</td>
</tr>
<tr>
<td>PP</td>
<td>INP</td>
<td>BiLstm</td>
</tr>
<tr>
<td>shares</td>
<td>NNS</td>
<td>BiLstm</td>
</tr>
<tr>
<td>NP</td>
<td>NNP</td>
<td>BiLstm</td>
</tr>
<tr>
<td>VP</td>
<td>VBP</td>
<td>BiLstm</td>
</tr>
<tr>
<td>ADV</td>
<td>RBP</td>
<td>BiLstm</td>
</tr>
<tr>
<td>PP</td>
<td>INP</td>
<td>BiLstm</td>
</tr>
</tbody>
</table>

The data used in this experiment is the CoNLL-
2000 phrase chunking corpus (Dong, Roesner, and
Buchholz, 2001). The training corpus consists of
sections 15-18 of the Penn Treebank (Marcus et al.,
1993), section 2 was used as the test set. The chunk
tags are derived from the parse tree constituents,
and the part-of-speech tags were generated by Brill's tagger (Brill, 1995). All the systems are trained to completion (until all the rules are learned).

Table 3 shows the results of the test-chaining experiment. The performance of the Fast TBL algorithm is the best of the three TBL systems and is within an order of magnitude factor. The ICA algorithm again runs considerably faster, but at a cost of a significant performance hit. Here are 2 reasons that contribute to this behavior:

1. The initial state has a lower performance than the ones in tagged training; therefore the independence assumption might not hold. 20% of the samples are changed by at least one rule as opposed to POS tagging, where only 25% of the samples are changed by a rule.

2. The commitment assumption might also not hold. For this test, 20% of the samples that were modified by a rule are also changed again by another rule.

4.4 Training Data Size Scalability

A question usually asked about a machine learning algorithm is how well it adapts to larger amounts of training data. Since the performance of the Fast TBL algorithm is identical to that of regular TBL, the learning time increases two orders of magnitude between the running time of the algorithm and the amount of training data.

The experiment was performed with the part-of-speech data set. The four algorithms were trained on training sets of different size training sets were recorded and averaged over 4 trials. The sample size is presented in Figure 2(a). It is obvious that the Fast TBL algorithm is much more scalable than the regular TBL — displaying a linear dependency on the amount of training data, while the regular TBL has an almost quadratic dependency. The explanation for this behavior has been given in Section 3.3.

Figure 2(b) shows the time spent at each iteration versus the iteration number, for the original TBL and Fast TBL systems. It can be observed that as the time taken per iteration increases dramatically with the iteration number for the regular TBL, while for the Fast TBL, the situation is reversed. The consequence is that, over a certain threshold has been reached, the training time needed to train the Fast-TBL system is comparable.

5 Conclusions

We have presented in this paper a new and improved method of computing the objective function for transformation-based learning. This method allows a transformation-based algorithm to train an observed 12 to 120 times faster than the original one, while preserving the final performance of the algorithm. The method was tested in three different domains, each one having different characteristics: part-of-speech tagging, prepositional phrase attachment and text chaining. The results obtained indicate that the algorithm improvement generated by our method is not linked to a particular task, but extends to any classification task where transformation-based learning can be applied. Furthermore, our algorithm scales better with training data size, therefore the relative speedup will increase when more samples are available for training, making the procedure a good candidate for large corpus tasks.

The increased speed of the Fast TBL algorithm also enables its usage in higher level machine learning algorithms, such as adjective-boosting, model combination and active learning. Recent work (Florian et al., 2003) has shown how a TBL framework can be adopted to generate confidence on the output, and our algorithm is compatible with that framework. The stability, resilience to overtraining, the existence of probability estimates and, reasonable speed make TBL an excellent candidate for solving classification tasks in general.

6 Acknowledgements

The authors would like to thank David Jernovetz for his advice and guidance, Eric Brill and John C. Henderson for discussions on the initial ideas of the material presented in the paper, and the anonymous reviewers of the conference submissions, observations, and connections with other published material. The work presented here was supported by NSF grants BES-9633754, BES-9633814 and DMI-9908088.

References