Mr. MIRA: Open-Source Large-Margin Structured Learning on MapReduce

Vladimir Eidelman[1], **Ke Wu**[1], **Ferhan Ture**[1], **Philip Resnik**[2], **Jimmy Lin**[3]
[1] Dept. of Computer Science [2] Dept. of Linguistics [3] The iSchool
Institute for Advanced Computer Studies
University of Maryland
{eidelman,wuke,fture,resnik,jimmylin}@umd.edu

Abstract

We present an open-source framework for large-scale online structured learning. Developed with the flexibility to handle cost-augmented inference problems such as statistical machine translation (SMT), our large-margin learner can be used with any decoder. Integration with MapReduce using Hadoop streaming allows efficient scaling with increasing size of training data. Although designed with a focus on SMT, the decoder-agnostic design of our learner allows easy future extension to other structured learning problems such as sequence labeling and parsing.

1 Introduction

Structured learning problems such as sequence labeling or parsing, where the output has a rich internal structure, commonly arise in NLP. While batch learning algorithms adapted for structured learning such as CRFs (Lafferty et al., 2001) and structural SVMs (Joachims, 1998) have received much attention, online methods such as the structured perceptron (Collins, 2002) and a family of Passive-Aggressive algorithms (Crammer et al., 2006) have recently gained prominence across many tasks, including part-of-speech tagging (Shen, 2007), parsing (McDonald et al., 2005) and statistical machine translation (SMT) (Chiang, 2012), due to their ability to deal with large training sets and high-dimensional input representations.

Unlike batch learners, which must consider all examples when optimizing the objective, online learners operate in rounds, optimizing using one example or a handful of examples at a time. This online nature offers several attractive properties, facilitating scaling to large training sets while remaining simple and offering fast convergence.

Mr. MIRA, the open source system[1] described in this paper, implements an online large-margin structured learning algorithm based on MIRA (§2.1), for cost-augmented online large-scale training in high-dimensional feature spaces. Our contribution lies in providing the first published decoder-agnostic parallelization of MIRA with Hadoop for structured learning.

While the current demonstrated application focuses on large-scale discriminative training for machine translation, the learning algorithm is general with respect to the inference algorithm employed. We are able to decouple our learner entirely from the MT decoder, allowing users to specify their own inference procedure through a simple text communication protocol (§2.2). The learner only requires k-best output with feature vectors, as well as the specification of a cost function. Standard automatic evaluation metrics for MT, such as BLEU (Papineni et al., 2002) and TER (Snover et al., 2006), have already been implemented. Furthermore, our system can be extended to other structured learning problems with a minimal amount of effort, simply by implementing a task-specific cost function and specifying an appropriate decoder.

Through Hadoop streaming, our system can take advantage of commodity clusters to handle large-scale training (§3), while also being capable of running in environments ranging from a single machine to a PBS-managed batch cluster. Experimental results (§4) show that it scales linearly and makes fast parameter tuning on large tuning sets for SMT practical.

2 Learning and Inference

2.1 Online Large-Margin Learning

MIRA is a popular online large-margin structured learning method for NLP tasks (McDonald et al., 2005; Chiang et al., 2009; Chiang, 2012). The

[1]`https://github.com/kho/mr-mira`

main intuition is that we want our model to enforce a margin between the correct and incorrect outputs of a sentence that agrees with our cost function. This is done by making the smallest update we can to our parameters, $\mathbf{w}$, on every sentence, that will ensure that the difference in model scores $\delta\mathbf{f}_i(y') = \mathbf{w}^\top(\mathbf{f}(x_i, y^+) - \mathbf{f}(x_i, y'))$ between the correct output y^+ and incorrect output y' is at least as large as the cost, $\Delta_i(y')$, incurred by predicting the incorrect output:[2]

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w} - \mathbf{w}_t||^2 + C\xi_i$$
$$\text{s.t. } \forall y' \in \mathcal{Y}(x_i),\ \delta\mathbf{f}_i(y') \geq \Delta_i(y') - \xi_i$$

where $\mathcal{Y}(x_i)$ is the space of possible structured outputs we are able to produce from x_i, and C is a regularization parameter that controls the size of the update. In practice, we can define $\mathcal{Y}(x_i)$ to be the k-best output. With a passive-aggressive (PA) update, the $\forall y'$ constraint above can be approximated by selecting the single most violated constraint, which maximizes $y' \leftarrow \arg\max_{y\in\mathcal{Y}(x_i)} \mathbf{w}^\top\mathbf{f}(x_i, y) + \Delta_i(y)$. This optimization problem is attractive because it reduces to a simple analytical solution, essentially performing a subgradient descent step with the step size adjusted based on each example:

$$\alpha \leftarrow \min\left(C, \frac{\Delta_i(y') - \delta\mathbf{f}_i(y')}{\|\mathbf{f}(x_i, y^+) - \mathbf{f}(x_i, y')\|^2}\right)$$
$$\mathbf{w} \leftarrow \mathbf{w} + \eta\alpha\left(\mathbf{f}(x_i, y^+) - \mathbf{f}(x_i, y')\right)$$

The user-defined cost function is a task-specific external measure of quality that relays how bad selecting y' truly is on the task we care about. The cost can take any form as long as it decomposes across the local parts of the structure, just as the feature functions. For instance, it could be the Hamming loss for sequence labeling, F-score for parsing, or an approximate BLEU score for SMT.

Cost-augmented Inference For most structured prediction problems in machine learning, $y_i \in \mathcal{Y}(x_i)$, that is, the model is able to produce, and thus score, the correct output structure, meaning $y^+ = y_i$. However, for certain NLP problems this may not be the case. For instance in SMT, our model may not be able to produce or reach the correct reference translation, which prohibits our model from scoring it. This problem necessitates cost-augmented inference, where we select $y^+ \leftarrow \arg\max_{y\in\mathcal{Y}(x_i)} \mathbf{w}^\top\mathbf{f}(x_i, y) - \Delta_i(y)$ from the space of structures our model can produce, to stand in for the correct output in optimization. Our system was developed to handle both cases, with the decoder providing the k-best list to the learner, specifying whether to perform cost-augmented selection.

Sparse Features While utilizing sparse features is a primary motivation for performing large-scale discriminative training, which features to use and how to learn their weights can have a large impact on the potential benefit. To this end, we incorporate ℓ_1/ℓ_2 regularization for joint feature selection in order to improve efficiency and counter overfitting effects (Simianer et al., 2012). Furthermore, the PA update has a single learning rate η for all features, which specifies how much the feature weights can change at each update. However, since dense features (e.g., language model) are observed far more frequently than sparse features (e.g., rule id), we may instead want to use a per-feature learning rate that allows larger steps for features that do not have much support. Thus, we allow setting an adaptive per-feature learning rate (Green et al., 2013; Crammer et al., 2009; Duchi et al., 2011).

2.2 Learner/Decoder Communication

Training requires communication between the decoder and the learner. The decoder needs to receive weight updates and the input sentence from the learner; and the learner needs to receive k-best output with feature vectors from the decoder. This is essentially all the required communication between the learner and the decoder. Below, we describe a simple line-based text protocol.

Input sentence and weight updates Following common practice in machine translation, the learner encodes each input sentence as a single-line SGML entry named `seg` and sends it to the decoder. The first line of Figure 1 is an example sentence in this format. In addition to the required sentence ID (useful in parallel processing), an optional `delta` field is used to encode the weight updates, as a sparse vector indexed by feature names. First, for each name and update pair, a binary record consisting of a null-terminated string (name) and a double-precision floating point number in native byte order (update) is created. Then, all binary records are con-

[2]For a more formal description we refer the reader to (Crammer et al., 2006; Chiang, 2012).

```
<seg id="123" delta="TE0AexSuR+F6hD8="> das ist ein kleine haus </seg>
<seg id="124"> ein kleine haus </seg>\tein kleine ||| a small\thaus ||| house
```

Figure 1: Example decoder input in SGML

```
5
123 ||| 5 ||| this is a small house ||| TE0AAAAA... <base64> ||| 120.3
123 ||| 5 ||| this is the small house ||| <base64> ||| 118.4
123 ||| 5 ||| this was small house ||| <base64> ||| 110.5
<empty>
<empty>
```

Figure 2: Example k-best output

catenated and encoded in base64. In the example above, the value of `delta` is the base64 encoding of `0x4c 0x4d 0x00 0x7b 0x14 0xae 0x47 0xe1 0x7a 0x84 0x3f`. The first 3 bytes store the feature name (`LM`) and the next 8 bytes is its update (0.01), to be added to the decoder's current value of the corresponding feature weight.

The learner also allows the user to pass any additional information to the decoder, as long as it can be encoded as a single-line text string. Such information, if given, is appended after the `seg` entry, with a leading tab character as the delimiter. For example, the second line of Figure 1 passes two phrase translation rules to the decoder.

k-best output The decoder reads from standard input and outputs the k-best output for one input sentence before consuming the next line. For the k-best output, the decoder first outputs to standard output a line consisting of a single integer N. Next the decoder outputs N lines where each line can be either empty or an actual hypothesis. When the line is an actual hypothesis, it consists of the following parts:

```
SID ||| LEN ||| TOK ||| FEAT [ REST ]
```

`SID` is the sentence ID of the corresponding input; `LEN` is the length of source sentence;[3] `TOK` contains the tokens of the hypothesis sentence separated by spaces; `FEAT` is the feature vector, encoded in the same way as the weight updates, delimited by a whitespace. Everything after `FEAT` until the end of the line is discarded. See Figure 2 for an example of k-best output. Note the scores after the last `|||` are discarded by the learner.

Overall workflow The learner reads lines from standard input in the following tab-delimited format:

```
SRC<tab>REF<tab>REST
```

`SRC` is the actual input sentence as a `seg` entry; `REF` is the gold output for the input sentence, for example, reference translations in MT;[4] `REST` is the additional information that will be appended after the `seg` entry and passed to the decoder.

The learner creates a sub-process for the decoder and connects to the sub-process' standard input and output with pipes. Then it processes the input lines one by one. For each line, it first sends a composed input message to the decoder, combining the input sentence, weight updates, and user-supplied information. Next it collects the k-best output from the decoder, solves the QP problem to obtain weight updates and repeats.

The learner produces two types of output. First, the 1-best hypothesis for each input sentence, in the following format:

```
SID<tab>TOK
```

Second, when there are no more input lines, the learner outputs final weights and the number of lines processed, in the following format:

```
-1<tab>NUM ||| WEIGHTS
```

The 1-best hypotheses can be scored against references to obtain an estimate of cost. The final weights are stored in a way convenient for averaging in a parallel setting, as we shall discuss next.

3 Large-Scale Discriminative Training

3.1 MapReduce

With large amounts of data available today, distributed computations have become essential. MapReduce (Dean and Ghemawat, 2004) has emerged as a popular distributed processing framework for commodity clusters that has gained widespread adoption in both industry and academia, thanks to its simplicity and the availability of the Hadoop open-source implementation. MapReduce provides a higher level of

[3]This is used in computing the smoothed cost. Usually this is identical for all hypotheses if the input is a plain sentence. But in applications such as lattice-based translation, each hypothesis can be produced from different source sentences, resulting in different lengths.

[4]There can be multiple references, separated by `|||`.

abstraction for designing distributed algorithms compared to, say, MPI or pthreads, by hiding system-level details (e.g., deadlock, race conditions, machine failures) from the developer.

A single MapReduce program begins with a *map* phase, where mapper processes input key-value pairs to produce an arbitrary number of intermediate key-value pairs. The mappers execute in parallel, consuming data splits independently. Following the map phase, all key-value pairs emitted by the mappers are sorted by key and distributed to the reducers, such that all pairs sharing the same key are guaranteed to arrive at the same reducer. Finally, in the *reduce* phase, each reducer processes the intermediate key-value pairs it receives and emits final output key-value pairs.

3.2 System Architecture

Algorithm design We use Hadoop streaming to parallelize the training process. Hadoop streaming allows any arbitrary executable to serve as the mapper or reducer, as long as it handles key-value pairs properly.[5] One iteration of training is implemented as a single Hadoop streaming job. In the map step, our learner can be directly used as the mapper. Each mapper loads the same initial weights, processes a single split of data and produces key-value pairs: the one-best hypothesis of each sentence is output with the sentence ID as the key (non-negative); the final weights with respect to the split are output with a special negative key. In the reduce step, a single reducer collects all key-value pairs, grouped and sorted by keys. The one-best hypotheses are output to disk in the order they are received, so that the order matches the reference translation set. The reducer also computes the feature selection and weighted average of final weights received from all of the mappers. Assuming mapper i produces the final weights $\mathbf{w}_i$ after processing n_i sentences, the weighted averaged is defined as $\mathbf{w}^* = \frac{\sum_i \mathbf{w}_i \times n_i}{\sum_i n_i}$. Although averaging yields different result from running a single learner over the entire data, we have found the difference to be quite small in terms of convergence and quality of tuned weights in practice.

After the reducer finishes, the averaged weights are extracted and used as the initial weights for the next iteration; the emitted hypotheses are scored against the references, which allows us to track the learning curve and the progress of convergence.

Scalability In an application such as SMT, the decoder requires access to the translation grammar and language model to produce translation hypotheses. For small tuning sets, which have been typical in MT research, having these files transferred across the network to individual servers (which then load the data into memory) is not a problem. However, for even modest input on the order of tens of thousands of sentences, this creates a challenge. For example, distributing thousands of per-sentence grammar files to all the workers in a Hadoop cluster is time-consuming, especially when this needs to be performed prior to every iteration.

To benefit from MapReduce, it is essential to avoid dependencies on "side data" as much as possible, due to the challenges explained above with data transfer. To address this issue, we append the per-sentence translation grammar as user-supplied additional information to each input sentence. This results in a large input file (e.g., 75 gigabytes for 50,000 sentences), but this is not an issue since the data reside on the Hadoop distributed file system and MapReduce optimizes for data locality when scheduling mappers.

Unfortunately, it is much more difficult to obtain per-sentence language models that are small enough to handle in this same manner. Currently, the best solution we have found is to use Hadoop's distributed cache to ship the single large language model to each worker.

4 Evaluation

We evaluated online learning in Hadoop MapReduce by applying it to German-English machine translation, using our hierarchical phrase-based translation system with `cdec` as the decoder (Dyer et al., 2010). The parallel training data consist of the Europarl and News Commentary corpora from the WMT12 translation task,[6] containing 2.08M sentences. A 5-gram language model was trained on the English side of the bitext along with 750M words of news using SRILM with modified Kneser-Ney smoothing (Chen and Goodman, 1996).

We experimented with two feature sets: (1) a small set with standard MT features, including

[5]By default, each line is treated as a key-value pair encoded in text, where the key and the value are separated by a `<tab>`.

[6]http://www.statmt.org/wmt12/translation-task.html

Tuning set size		Time/iteration	# splits	# features	Tuning BLEU	Test	
(corpus)	(on disk, GB)	(in seconds)				BLEU	TER
dev	3.3	119	120	16	22.38	22.69	60.61
5k	7.8	289	120	16	32.60	22.14	59.60
10k	15.2	432	120	16	33.16	22.06	59.43
25k	37.2	942	300	16	32.48	22.21	59.54
50k	74.5	1802	600	16	32.21	22.21	59.39
dev	3.3	232	120	85k	23.08	23.00	60.19
5k	7.8	610	120	159k	33.70	22.26	59.26
10k	15.2	1136	120	200k	34.00	22.12	59.24
25k	37.2	2395	300	200k	32.96	22.35	59.29
50k	74.5	4465	600	200k	32.86	22.40	59.15

Table 1: Evaluation of our Hadoop implementation of MIRA, showing running time as well as BLEU and TER values for tuning and testing data.

	dev	test	5k	10k	25k	50k
Sentences	3003	3003	5000	10000	25000	50000
Tokens en	75k	74k	132k	255k	634k	1258k
Tokens de	74k	73k	133k	256k	639k	1272k

Table 2: Corpus statistics

phrase and lexical translation probabilities in both directions, word and arity penalties, and language model scores; and (2) a large set containing the top 200k sparse features that might be useful to train on large numbers of instances: rule id and shape, target bigrams, insertions and deletions, and structural distortion features.

All experiments were conducted on a Hadoop cluster (running Cloudera's distribution, CDH 4.2.1) with 16 nodes, each with two quad-core 2.2 GHz Intel Nehalem Processors, 24 GB RAM, and three 2 TB drives. In total, the cluster is configured for a capacity of 128 parallel workers, although we do not have direct control over the number of simultaneous mappers, which depends on the number of input splits. If the number of splits is smaller than 128, then the cluster is under-utilized. To note this, we report the number of splits for each setting in our experimental results (Table 1).

We ran MIRA on a number of tuning sets, described in Table 2, in order to test the effectiveness and scalability of our system. First, we used the standard development set from WMT12, consisting of 3,003 sentences from news domain. In order to show the scaling characteristics of our approach, we then used larger portions of the training bitext directly to tune parameters. In order to avoid overfitting, we used a jackknifing method to split the training data into $n = 10$ folds, and built a translation system on $n - 1$ folds, while adjusting the sampling rate to sample sentences from the other fold to obtain tuning sets ranging from 5,000 sentences to 50,000 sentences. Table 1 shows details of experimental results for each setting. The second column shows the space each tuning set takes up on disk when we include reference translations and grammar files along with the sentences. The reported tuning BLEU is from the iteration with best performance, and running times are reported from the top-scoring iteration as well.

Even though our focus in this evaluation is to show the scalability of our implementation to large input and feature sets, it is also worthwhile to mention the effectiveness aspect. As we increase the tuning set size by sampling sentences from the training data, we see very little improvement in BLEU and TER with the smaller feature set. This is not surprising, since sparse features are more likely to gain from additional tuning instances. Indeed, tuning scores for all sets improve substantially with sparse features, accompanied by small increases on test.

While tuning on dev data results in better BLEU on test data than when tuning on the larger sets, it is important to note that although we are able to tune more features on the larger bitext tuning sets, they are not composed of the same genre as the dev and test sets, resulting in a domain mismatch.

Therefore, we are actually comparing a smaller in-domain tuning set with a larger out-of-domain set. While this domain adaptation is problematic (Haddow and Koehn, 2012), the ability to discriminatively tune on larger sets remains highly desirable.

In terms of running time, we observe that the algorithm scales linearly with respect to input size, regardless of the feature set. With more features, running time increases due to a more complex translation model, as well as larger intermediate output (i.e., amount of information passed from mappers to reducers). The scaling characteristics point out the strength of our system: our scalable MIRA implementation allows one to tackle learning problems where there are many parameters, but also many training instances.

Comparing the wall clock time of parallelization with Hadoop to the standard mode of 10–20 learner parallelization (Haddow et al., 2011; Chiang et al., 2009), for the small 25k feature setting, after one iteration, which takes 4625 seconds using 15 learners on our PBS cluster, the tuning score is 19.5 BLEU, while in approximately the same time, we can perform five iterations with Hadoop and obtain 30.98 BLEU. While this is not a completely fair comparison, as the two clusters utilize different resources and the number of learners, it suggests the practical benefits that Hadoop can provide. Although increasing the number of learners on our PBS cluster to the number of mappers used in Hadoop would result in roughly equivalent performance, arbitrarily scaling out learners on the PBS cluster to handle larger training sets can be challenging since we'd have to manually coordinate the parallel processes in an ad-hoc manner. In contrast, Hadoop provides scalable parallelization in a manageable framework, providing data distribution, synchronization, fault tolerance, as well as other features, "for free".

5 Conclusion

In this paper, we presented an open-source framework that allows seamlessly scaling structured learning to large feature-rich problems with Hadoop, which lets us take advantage of large amounts of data as well as sparse features. The development of Mr. MIRA has been motivated primarily by application to SMT, but we are planning to extend our system to other structured prediction tasks in NLP such as parsing, as well as to facilitate its use in other domains.

Acknowledgments

This research was supported in part by the DARPA BOLT program, Contract No. HR0011-12-C-0015; NSF under awards IIS-0916043 and IIS-1144034. Vladimir Eidelman is supported by a NDSEG Fellowship. Any opinions, findings, conclusions, or recommendations expressed are those of the authors and do not necessarily reflect views of the sponsors.

References

S. Chen and J. Goodman. 1996. An empirical study of smoothing techniques for language modeling. In *ACL*.

D. Chiang, K. Knight, and W. Wang. 2009. 11,001 new features for statistical machine translation. In *NAACL-HLT*.

D. Chiang. 2012. Hope and fear for discriminative training of statistical translation models. *JMLR*, 13:1159–1187.

M. Collins. 2002. Ranking algorithms for named-entity extraction: boosting and the voted perceptron. In *ACL*.

K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. 2006. Online passive-aggressive algorithms. *JMLR*, 7:551–585.

K. Crammer, A. Kulesza, and M. Dredze. 2009. Adaptive regularization of weight vectors. In *NIPS*.

J. Dean and S. Ghemawat. 2004. MapReduce: Simplified data processing on large clusters. In *OSDI*.

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

C. Dyer, A. Lopez, J. Ganitkevitch, J. Weese, F. Ture, P. Blunsom, H. Setiawan, V. Eidelman, and P. Resnik. 2010. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *ACL System Demonstrations*.

S. Green, S. Wang, D. Cer, and C. Manning. 2013. Fast and adaptive online training of feature-rich translation models. In *ACL*.

B. Haddow and P. Koehn. 2012. Analysing the effect of out-of-domain data on smt systems. In *WMT*.

B. Haddow, A. Arun, and P. Koehn. 2011. SampleRank training for phrase-based machine translation. In *WMT*.

T. Joachims. 1998. Text categorization with support vector machines: Learning with many relevant features. In *ECML*.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *ACL*.

K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *ACL*.

L. Shen. 2007. Guided learning for bidirectional sequence classification. In *ACL*.

P. Simianer, S. Riezler, and C. Dyer. 2012. Joint feature selection in distributed stochastic learning for large-scale discriminative training in SMT. In *ACL*.

M. Snover, B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *AMTA*.