HadoopPerceptron: a Toolkit for Distributed Perceptron Training and Prediction with MapReduce

Andrea Gesmundo
Computer Science Department
University of Geneva
Geneva, Switzerland
andrea.gesmundo@unige.ch

Nadi Tomeh
LIMSI-CNRS and
Université Paris-Sud
Orsay, France
nadi.tomeh@limsi.fr

Abstract

We propose a set of open-source software modules to perform structured Perceptron Training, Prediction and Evaluation within the Hadoop framework. Apache Hadoop is a freely available environment for running distributed applications on a computer cluster. The software is designed within the Map-Reduce paradigm. Thanks to distributed computing, the proposed software reduces substantially execution times while handling huge data-sets. The distributed Perceptron training algorithm preserves convergence properties, thus guaranties same accuracy performances as the serial Perceptron. The presented modules can be executed as stand-alone software or easily extended or integrated in complex systems. The execution of the modules applied to specific NLP tasks can be demonstrated and tested via an interactive web interface that allows the user to inspect the status and structure of the cluster and interact with the MapReduce jobs.

1 Introduction

The Perceptron training algorithm (Rosenblatt, 1958; Freund and Schapire, 1999; Collins, 2002) is widely applied in the Natural Language Processing community for learning complex structured models. The non-probabilistic nature of the perceptron parameters makes it possible to incorporate arbitrary features without the need to calculate a partition function, which is required for its discriminative probabilistic counterparts such as CRFs (Lafferty et al., 2001). Additionally, the Perceptron is robust to approximate inference in large search spaces.

Nevertheless, Perceptron training is proportional to inference which is frequently non-linear in the input sequence size. Therefore, training can be time-consuming for complex model structures. Furthermore, for an increasing number of tasks is fundamental to leverage on huge sources of data as the World Wide Web. Such difficulties render the scalability of the Perceptron a challenge.

In order to improve scalability, Mcdonald et al. (2010) propose a distributed training strategy called *iterative parameter mixing*, and show that it has similar convergence properties to the standard perceptron algorithm; it finds a separating hyperplane if the training set is separable; it produces models with comparable accuracies to those trained serially on all the data; and reduces training times significantly by exploiting computing clusters.

With this paper we present the HadoopPerceptron package. It provides a freely available open-source implementation of the iterative parameter mixing algorithm for training the structured perceptron on a generic sequence labeling tasks. Furthermore, the package provides two additional modules for prediction and evaluation. The three software modules are designed within the MapReduce programming model (Dean and Ghemawat, 2004) and implemented using the Apache Hadoop distributed programming Framework (White, 2009; Lin and Dyer, 2010). The presented HadoopPerceptron package reduces execution time significantly compared to its serial counterpart while maintaining comparable performance.

Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics, pages 97–101,
Avignon, France, April 23 - 27 2012.

PerceptronIterParamMix($\mathcal{T} = \{(\mathbf{x}_t, \mathbf{y}_t)\}_{t=1}^{|\mathcal{T}|}$)
1. Split $\mathcal{T}$ into $\mathcal{S}$ pieces $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_\mathcal{S}\}$
2. $\mathbf{w} = \mathbf{0}$
3. for $n : 1..N$
4. $\quad \mathbf{w}^{(i,n)} =$ OneEpochPerceptron($\mathcal{T}_i, \mathbf{w}$)
5. $\quad \mathbf{w} = \sum_i \mu_{i,n} \mathbf{w}^{(i,n)}$
6. return $\mathbf{w}$

OneEpochPerceptron($\mathcal{T}_i, \mathbf{w}^*$)
1. $\mathbf{w}^{(0)} = \mathbf{w}^*; k = 0$
2. for $n : 1..T$
3. Let $\mathbf{y}' = \arg\max_{\mathbf{y}'} \mathbf{w}^{(k)}.\mathbf{f}(\mathbf{x}_t, \mathbf{y}'_t)$
4. if $\mathbf{y}' \neq \mathbf{y}_t$
5. $\quad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{f}(\mathbf{x}_t, \mathbf{y}_t) - \mathbf{f}(\mathbf{x}_t, \mathbf{y}'_t)$
6. $\quad k = k + 1$
7. return $\mathbf{w}^{(k)}$

Figure 1: Distributed perceptron with iterative parameter mixing strategy. Each $\mathbf{w}^{(i,n)}$ is computed in parallel. $\boldsymbol{\mu}_n = \{\mu_{1,n}, \ldots, \mu_{\mathcal{S},n}\}, \forall \mu_{i,n} \in \boldsymbol{\mu}_n : \mu_{i,n} \geq 0$ and $\forall n : \sum_i \mu_{i,n} = 1$.

2 Distributed Structured Perceptron

The structured perceptron (Collins, 2002) is an online learning algorithm that processes training instances one at a time during each training epoch. In sequence labeling tasks, the algorithm predicts a sequence of labels (an element from the structured output space) for each input sequence. Prediction is determined by linear operations on high-dimensional feature representations of candidate input-output pairs and an associated weight vector. During training, the parameters are updated whenever the prediction that employed them is incorrect.

Unlike many batch learning algorithms that can easily be distributed through the gradient calculation, the perceptron online training is more subtle to parallelize. However, Mcdonald et al. (2010) present a simple distributed training through a parameter mixing scheme.

The Iterative Parameter Mixing is given in Figure 2 (Mcdonald et al., 2010). First the training data is divided into disjoint splits of example pairs $(\mathbf{x}_t, \mathbf{y}_t)$ where $\mathbf{x}_t$ is the observation sequence and $\mathbf{y}_t$ is the associated labels. The algorithm proceeds to train a single epoch of the perceptron algorithm for each split in parallel, and mix the local models weights $\mathbf{w}^{(i,n)}$ to produce the global weight vector $\mathbf{w}$. The mixed model is then passed to each split to reset the perceptron local weights, and a new iteration is started. Mcdonald et al. (2010) provide bound analysis for the algorithm and show that it is guaranteed to converge and find a seperation hyperplane if one exists.

3 MapReduce and Hadoop

Many algorithms need to iterate over number of records and 1) perform some calculation on each of them and then 2) aggregate the results. The MapReduce programming model implements a functional abstraction of these two operations called respectively Map and Reduce. The Map function takes a value-key pairs and produces a list of key-value pairs: map$(k, v) \rightarrow (k', v')^*$; while the input the Reduce function is a key with all the associated values produced by all the mappers: reduce$(k', (v')^*) \rightarrow (k'', v'')^*$. The model requires that all values with the same key are reduced together.

Apache Hadoop is an open-source implementation of the MapReduce model on cluster of computers. A cluster is composed by a set of computers (nodes) connected into a network. One node is designated as the Master while other nodes are referred to as Worker Nodes. Hadoop is designed to scale out to large clusters built from commodity hardware and achieves seamless scalability. To allow rapid development, Hadoop hides system-level details from the application developer.The MapReduce runtime automatically schedule worker assignment to mappers and reducers;handles synchronization required by the programming model including gathering, sorting and shuffling of intermediate data across the network; and provides robustness by detecting worker failures and managing restarts. The framework is built on top of he Hadoop Distributed File System (HDFS), which allows to distribute the data across the cluster nodes. Network traffic is minimized by moving the process to the node storing the data. In Hadoop terminology an entire MapReduce program is called a *job* while individual mappers and reducers are called *tasks*.

4 HadoopPerceptron Implementation

In this section we give details on how the training, prediction and evaluation modules are implemented for the Hadoop framework using the

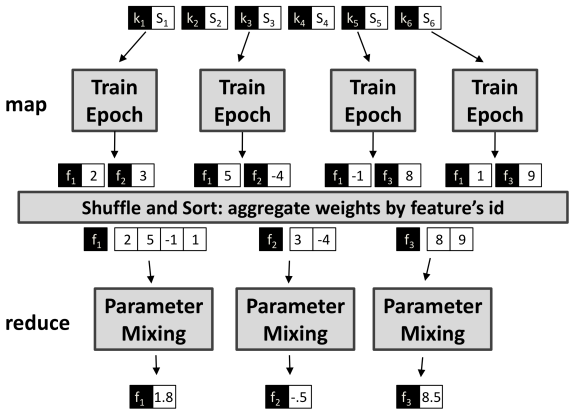

Figure 2: HadoopPerceptron in MapReduce.

MapReduce programming model[1].

Our implementation of the iterative parameter mixing algorithm is sketched in Figure 2. At the beginning of each iteration, the training data is split and distributed to the worker nodes. The set of training examples in a data split is streamed to map workers as pairs $(\text{sentence-id}, (\mathbf{x}_t, \mathbf{y}_t))$. Each map worker performs a standard perceptron training epoch and outputs a pair $(\text{feature-id}, w_{i,f})$ for each feature. The set of such pairs emitted by a map worker represents its local weight vector. After map workers have finished, the MapReduce framework guarantees that all local weights associated with a given feature are aggregated together as input to a distinct reduce worker. Each reduce worker produces as output the average of the associated feature weight. At the end of each iteration, the reduce workers outputs are aggregated into the global averaged weight vector. The algorithm iterates N times or until convergence is achieved. At the beginning of each iteration the weight vector of each distinct model is initialized with the global averaged weight vector resultant from the previous iteration. Thus, for all the iterations except for the first, the global averaged weight vector resultant from the previous iteration needs to be provided the map workers. In Hadoop it is possible to pass this information via the Distributed Cache System.

In addition to the training module, the HadoopPerceptron package provides separate modules for prediction and evaluation both of them are designed as MapReduce programs. The evaluation module output the accuracy measure computed against provided gold standards. Prediction and evaluation modules are independent from the training modules, the weight vector given as input could have been computed with any other system using any other training algorithm as long as they employ the same features.

The implementation is in Java, and we interface with the Hadoop cluster via the native Java API. It can be easily adapted to a wide range of NLP tasks. Incorporating new features by modifying the extensible feature extractor is straightforward. The package includes the implementation of the basic feature set described in (Suzuki and Isozaki, 2008).

5 The Web User Interface

Hadoop is bundled with several web interfaces that provide concise tracking information for jobs, tasks, data nodes, etc. as shown in Figure 3. These web interfaces can be used to demonstrate the HadoopPerceptron running phases and monitor the distributed execution of the training, prediction and evaluation modules for several sequence labeling tasks including part-of-speech tagging and named entity recognition.

6 Experiments

We investigate HadoopPerceptron training time and prediction accuracy on a part-of-speech (POS) task using the PennTreeBank corpus (Marcus et al., 1994). We use sections 0-18 of the Wall Street Journal for training, and sections 22-24 for testing.

We compare the regular percepton trained serially on all the training data with the distributed perceptron trained with iterative parameter mixing with variable number of splits $\mathcal{S} \in \{10, 20\}$. For each system, we report the prediction accuracy measure on the final test set to determine if any loss is observed as a consequence of distributed training.

For each system, Figure 4 plots accuracy results computed at the end of every training epoch against consumed wall-clock time. We observe that iterative mixing parameter achieves comparable performance to its serial counterpart while converging orders of magnitude faster.

Furthermore, we note that the distributed algorithm achieves a slightly higher final accuracy

[1]The Hadoop Perceptron toolkit is available from https://github.com/agesmundo/HadoopPerceptron .

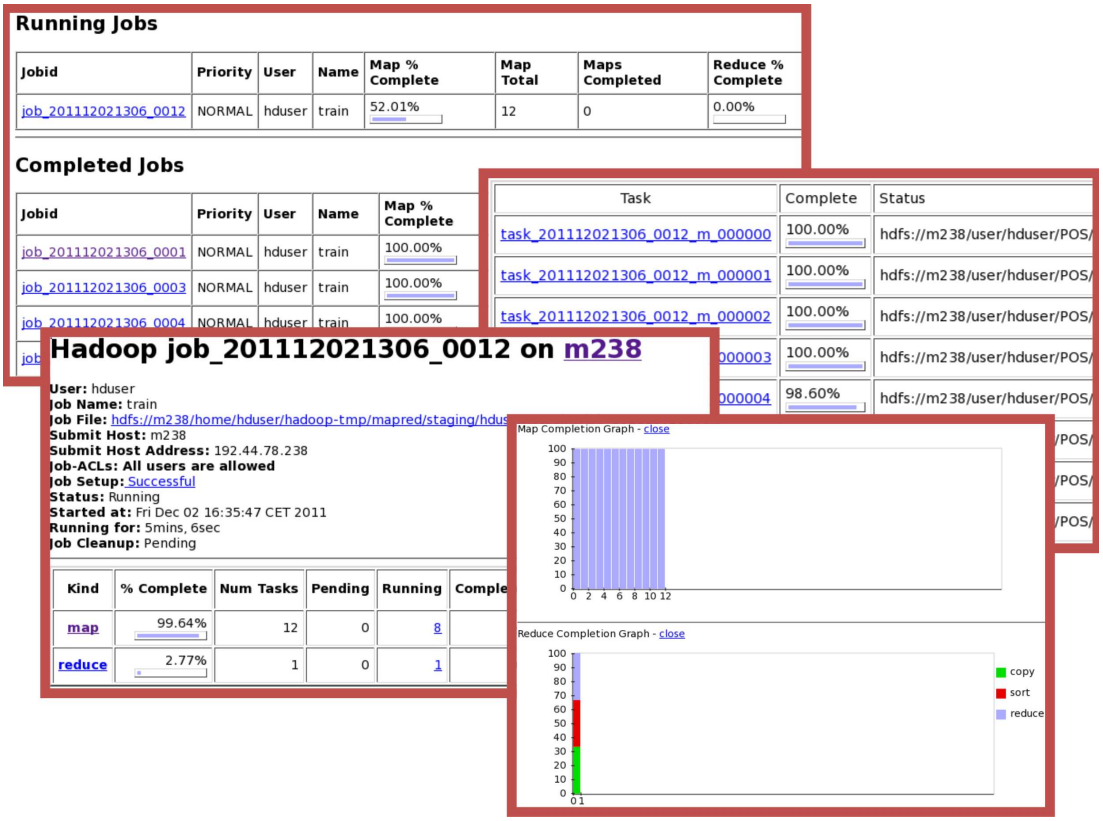

Figure 3: Hadoop interfaces for HadoopPerceptron.

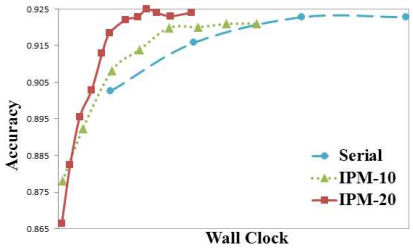

Figure 4: Accuracy vs. training time. Each point corresponds to a training epoch.

than serial training. Mcdonald et al. (2010) suggest that this is due to the bagging effect that the distributed training has, and due to parameter mixing that is similar to the averaged perceptron.

We note also that increasing the number of splits increases the number of epoch required to attain convergence, while reducing the time required per epoch. This implies a trade-off between slower convergence and quicker epochs when selecting a larger number of splits.

7 Conclusion

The HadoopPerceptron package provides the first freely-available open-source implementation of iterative parameter mixing Perceptron Training, Prediction and Evaluation for a distributed Map-Reduce framework. It is a versatile stand alone software or building block, that can be easily extended, modified, adapted, and integrated in broader systems.

HadoopPerceptron is a useful tool for the increasing number of applications that need to perform large-scale structured learning. This is the first freely available implementation of an approach that has already been applied with success in private sectors (e.g. Google Inc.). Making it possible for everybody to fully leverage on huge data sources as the World Wide Web, and develop structured learning solutions that can scale keeping feasible execution times and cluster-network usage to a minimum.

Acknowledgments

This work was funded by Google and The Scottish Informatics and Computer Science Alliance (SICSA). We thank Keith Hall, Chris Dyer and Miles Osborne for help and advice.

References

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *EMNLP '02: Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, Philadelphia, PA, USA.

Jeffrey Dean and Sanjay Ghemawat. 2004. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Opearting Systems Design and Implementation*, San Francisco, CA, USA.

Yoav Freund and Robert E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

John Lafferty, Andrew Mccallum, and Fernando Pereira. 2001. John lafferty and andrew mccallum and fernando pereira. In *Proceedings of the International Conference on Machine Learning*, Williamstown, MA, USA.

Jimmy Lin and Chris Dyer. 2010. *Data-Intensive Text Processing with MapReduce*. Morgan & Claypool Publishers.

Mitchell P. Marcus, Beatrice Santorini, and Mary A. Marcinkiewicz. 1994. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

Ryan Mcdonald, Keith Hall, and Gideon Mann. 2010. Distributed training strategies for the structured perceptron. In *NAACL '10: Proceedings of the 11th Conference of the North American Chapter of the Association for Computational Linguistics*, Los Angeles, CA, USA.

Frank Rosenblatt. 1958. The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Jun Suzuki and Hideki Isozaki. 2008. Semi-supervised sequential labeling and segmentation using giga-word scale unlabeled data. In *ACL '08: Proceedings of the 46th Conference of the Association for Computational Linguistics*, Columbus, OH, USA.

Tom White. 2009. *Hadoop: The Definitive Guide*. O'Reilly Media Inc.