Optimization and Learning in FACTORIE

Alexandre Passos, Luke Vilnis, Andrew McCallum
Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA, 01003
{apassos,luke,mccallum}@cs.umass.edu

Abstract

We present the optimization package in the FACTORIE library for machine learning, graphical models, and natural language processing in Scala. The library is designed with an eye towards the way practitioners naturally break down the structure of a learning problem, giving first-class status to, for example, training examples, update rules, and regularization. It supports flexible mixing and matching of parameter representation, update rules, loss functions, online-vs-batch training schedules, and serial-vs-parallel processing.

1 Introduction

Most optimization problems encountered in machine learning fall into two categories: inference, in which the goal is to make a prediction given a model and some observations, and learning, where the goal is to parametrize a model so that it generalizes well.

Surprisingly, there are few open source optimization packages that are tuned to solve the specific family of problems arising from supervised learning (notable exceptions being Theano and Pylearn2 [8]).

We present a novel architecture for an optimization package that solves the learning problem, as implemented in our FACTORIE toolkit for graphical models and machine learning. Since it is used both for research in machine learning as well as for applied systems, the package has to balance customizability and efficiency, allowing users to decide how to deal with parallelism, high-dimensional data, online vs batch, efficient online regularization, and other concerns. FACTORIE, implemented in the Scala programming language [17], is open-source under the Apache 2.0 license.

Formally, the goal of supervised learning is to find a parameter vector θ with low generalization error with respect to a distribution over data examples (x, y):

$$\theta^* = \arg\min_\theta E[\ell(x, y, \theta)]. \tag{1}$$

Many learning algorithms find such a θ by minimizing regularized loss over the observed training data $\{(x_i, y_i)\}$:

$$\theta^* = \arg\min_\theta \sum_i \ell(x_i, y_i, \theta) + R(\theta), \tag{2}$$

where the regularizer R penalizes model complexity and encourages generalization. Because data is often high-dimensional and sparse, most practical algorithms are gradient-based and avoid explicitly computing higher-order derivatives of the objective function.

While each learning problem involves a specific loss function ℓ and its gradient, all algorithms share common concerns that correspond to different parts of equation (2) : how to share work

between threads and aggregate losses and gradients over examples (the Σ), how to incorporate the regularizer R into the solution, and how to efficiently compute scores and update the parameters θ during learning. A well-designed optimization library should implement strategies for handling these issues somewhat independently, and allow for arbitrary mixing and overriding as desired by the user.

2 Design

We represent the components of equation (2) with the following interfaces:

- *Weights* are mutable slots containing parameter tensors θ. Optimizers can select appropriate representations for the weights, allowing e.g. efficient online regularization or averaging.
- *Examples* use Weights to calculate the gradient of the loss function ℓ for a particular labeled training instance.
- *Optimizers* take a gradient and use it to update Weights, optionally applying regularization R.
- *Trainers* coordinate between sets of Examples and Optimizers, deciding how many Examples to evaluate between calls to the Optimizer, and in which threads to evaluate Examples and call the Optimizer.

3 Functionality

FACTORIE implements many state-of-the-art optimization algorithms, including LBFGS, conjugate gradient, and many online algorithms, both regularized and unregularized. The regularized online algorithms include variants of Pegasos [20], regularized dual averaging [22], and AdaGrad dual averaging [4]. The unregularized algorithms include variants of MIRA and passive-aggressive algorithms [3], exponentiated gradient, and the averaged perceptron [7], and most of these are compatible with adaptive learning rates from the AdaGrad algorithm [4] and parameter averaging. An example of the power of this approach is our support for combining MIRA-style updates with AdaGrad's adaptive learning rates, an approach which is not mentioned in the literature but gives good empirical performance.

FACTORIE implements three batch Trainers: a single-core trainer, a parallel trainer with a single shared gradient vector and a parallel trainer with per-thread gradients. Each of the parallel strategies can be optimal in different scenarios, depending on the relative cost of computing a gradient on one example versus adding it to the single shared gradient vector. Likewise, there are many online trainers, including a single-core trainer, a parallel trainer which locks each weight tensor individually, a trainer which locks the optimizer (but not the weights tensor), and a fully hogwild [16] trainer which uses no locks.

It is important that Weights are slots rather than mutable tensors, because the optimization package must be able to swap in alternate tensor implementations in order to efficiently implement some optimization algorithms. An example is using a combination of a tensor and a scalar mutliplier for implementing ℓ_2 regularization, Pegasos, or exponentiated gradient. Algorithms such as RDA [22] require storing a tensor of dual averages and shrinking and thresholding this tensor on the fly when computing scores.

Our framework is designed to be not only customizable but also user-friendly. To this end, FACTORIE provides helper functions to train models and classifiers without requiring that the user explicitly selects examples and optimizers, but allowing customization as-needed. The level of parallelism can also always be controlled by the user, which is helpful in shared computing environments such as grid engines.

All of this learning infrastructure is automatically available for any Example a user can implement. FACTORIE also has predefined Examples for many common learning problems, such as classification, regression, learning to rank, matrix factorization, structured linear models with arbitrary inference (including CRF, structured perceptron, structured SVM), pseudolikelihood, and many others.

4 Related work

Most machine learning libraries, such as scikit-learn [19], Apache Mahout [6], OpenNLP [1], Weka [10], Liblinear [5], Leon Bottou's SGD package [2], and CRFSuite [18] provide efficient specialized implementations of some models. These implementations tend to be disconnected, do not share much learning code between them, and are not built on top of an optimization package for arbitrary loss functions.

A few libraries, however, do provide generic optimization infrastructures designed for machine learning. Unlike FACTORIE, they take a more black-box approach. Optimizers are usually unaware of the way the loss breaks down over examples, delegating the choice between batch and online optimization to the implementation of the loss function. Weights and gradients are represented concretely as sparse or dense arrays of floating point numbers, which prohibits the implementation of regularized stochastic algorithms without modifying the way that models compute scores.

Pylearn2 [8] is an example of a machine learning library with a generic optimization infrastructure. It has a generic model interface, and allows users to choose between many pre-packaged batch and online optimizers. Pylearn2 does not distinguish between Trainers and Optimizers. It is not agnostic to the representation for the Weights, forcing models to be aware of representations designed for specific learning algorithms. On the other hand, Pylearn2 can transparently delegate optimization to the GPU, which FACTORIE does not support.

The MALLET package for conditional random fields and classifiers [15] has an interface for generic optimization methods, but it does not support the efficient implementation of online algorithms, and it also does not distinguish between Trainers and Optimizers.

ScalaNLP [9] has a very general framework for linear algebra and optimization, but has not so far been able to make this efficient and relies on special-purpose solvers such as Liblinear to practically solve e.g. classification problems.

Vowpal Wabbit (VW) [13] falls into neither category. At its core, VW is a solver for binary classification problems. It solves more structured problems, such as multiclass classification, neural networks, or structured prediction, using reductions to binary classification. However, there is no functionality for directly optimizing arbitrary loss functions.

5 Benchmarks

While flexibility is a goal of the FACTORIE optimization package, so is efficiency. We measure its performance on two simple yet representative benchmarks: binary classification in the relatively large RCV1 dataset, and structured prediction in the CoNLL 2000 chunking shared task [2, 21].

For each task we report the following metrics. We report two accuracy numbers: when applicable, accuracy after the first pass through the data, as well as final accuracy of the trained model. We measure time both to do one pass through each task's training set as well as total time, which includes multiple passes over the training set, reading the data, and measuring accuracy.

The two timing numbers are necessary because the benchmarked packages vary greatly in their stopping criteria and speed of parsing the data files and extracting features. While overall time to learn a model is a good metric for real-world applications, time to perform one iteration is a better measure of optimization performance.

All experiments are run on a single core of a circa 2013 MacBook Pro. All hyperparameters are left to their default values.

5.1 Binary classification

For this experiment we follow the setup of Bottou et al [2] and use a version of the RCV1 dataset [14] which has been processed into a binary classification task for the class CCAT. The training file is 423mb long in gzipped svmlight [11] format, and is comprised of over 781k training examples. The test file is 12mb long and has 23k test examples. FACTORIE, Bottou SGD, and MALLET optimize the log loss. Scikit-learn optimizes the hinge loss. VW optimizes a squared loss variant.

Software	Algorithm	Acc. (1 pass)	Time (1 pass)	Acc. (final)	Time (final)
FACTORIE	AdaGrad RDA SGD	93.95	1.89	94.34	67.33
	L2 SGD	94.26	1.46	94.57	74.99
	LBFGS	N/A	1.07	94.79	101.30
	Avg. SGD	94.69	1.66	94.80	65.84
VW	Ada-Inv-Norm SGD*	94.23	9.03	94.23	14.78
	LBFGS*	N/A	10.07	94.72	217.17
Bottou SGD	Log SGD	94.40	0.25	94.82	11.27
MALLET	LBFGS	N/A	0.93	94.59	45.27
scikit-learn	SGD	93.94	0.91	93.99	205.31
	Liblinear SVM	N/A	13.73	94.77	216.20

Table 1: Results for the binary classification benchmark

Software	Algorithm	Acc. (1 pass)	Time (1 pass)	Acc. (final)	Time (final)
FACTORIE	AdaGrad RDA SGD	95.65	22.68	95.91	76.88
	Avg. SGD*	95.45	6.98	95.85	30.38
	L2 SGD	95.01	11.63	95.52	44.02
	LBFGS	N/A	9.20	96.11	1845.81
	Avg. Perceptron	95.34	1.07	95.77	12.42
Bottou SGD	L2 SGD	95.26	7.78	95.93	145.26
	Avg. L2 SGD	95.27	8.37	96.00	140.57
CRFSuite	L2 SGD	95.01	1.09	95.97	278.42
	Avg. Perceptron	95.28	0.57	95.75	68.18
	LBFGS	N/A	0.92	95.99	179.81
MALLET	LBFGS	N/A	42.93	95.85	6396.70

Table 2: Results for the chunking benchmark

Table 1 shows the timing and accuracy numbers for this experiment. The results indicate that FACTORIE is competitive even with highly optimized native code libraries like VW and Bottou SGD. Note that VW's per-iteration performance necessarily includes time to read the data cache, as it does not store examples in memory. Also note that MALLET does not support reading files in the svmlight format, instead parsing its own binary format, which saves a substantial amount of time.

5.2 Chunking

For this experiment we train linear-chain conditional random fields [12] on the data from the CoNLL 2000 chunking shared task [21]. All libraries use feature templates obtained from the CRFSuite benchmarks [18]. The training file is 597kb long in gzipped text format, with 8936 training examples, and the test file is 136kb long gzipped with 2012 test examples. Most algorithms optimize ℓ_2-regularized log loss and compute gradients with the forward-backward algorithm, except for the perceptron variants that optimize perceptron loss and compute gradients with Viterbi, and FACTORIE 's averaged SGD which uses unregularized log loss.

The results indicate FACTORIE is competitive with special-purpose C++ libraries even when using its generic optimization routines. CRFSuite in particular is considered to be the fastest CRF implementation, so our performance is very impressive in this light.

6 Conclusion

We present the design of the FACTORIE optimization package. We show how it can be used to efficiently implement modular and reusable learning algorithms. The package separates concerns such as threading and batching from both the gradient-computation code and the optimizer code. Our architecture allows users to choose between many algorithms with different performance characteristics, and replace individual components as they see fit. We show in some simple benchmarks that this generality does not come at a loss of efficiency.

References

[1] Jason Baldridge. The opennlp project, 2005.

[2] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, August 2010. Springer.

[3] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585, 2006.

[4] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2010.

[5] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

[6] Apache Software Foundation, Isabel Drost, Ted Dunning, Jeff Eastman, Otis Gospodnetic, Grant Ingersoll, Jake Mannix, Sean Owen, and Karl Wettin. Apache mahout, 2010. `http://mloss.org/software/view/144/`.

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

[8] Ian J Goodfellow, David Warde-Farley, Pascal Lamblin, Vincent Dumoulin, Mehdi Mirza, Razvan Pascanu, James Bergstra, Frédéric Bastien, and Yoshua Bengio. Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*, 2013.

[9] David Hall and Daniel Ramage. Scalanlp. http://www.scalanlp.org/.

[10] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[11] Thorsten Joachims. Making large scale svm learning practical. 1999.

[12] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning (ICML-2001)*. Morgan Kaufmann, 2001.

[13] J Langford, L Li, and A Strehl. Vowpal wabbit online learning project, 2007.

[14] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397, 2004.

[15] Andrew Kachites McCallum. Mallet: A machine learning for language toolkit. http://www.cs.umass.edu/ mccallum/mallet, 2002.

[16] Feng Niu, Benjamin Recht, Christopher Ré, and Stephen J Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *arXiv preprint arXiv:1106.5730*, 2011.

[17] Martin Odersky, Philippe Altherr, Vincent Cremet, Burak Emir, Sebastian Maneth, Stéphane Micheloud, Nikolay Mihaylov, Michel Schinz, Erik Stenman, and Matthias Zenger. An overview of the scala programming language. Technical report, Citeseer, 2004.

[18] Naoaki Okazaki. Crfsuite: a fast implementation of conditional random fields (crfs), 2007.

[19] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[20] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical Programming*, 127(1):3–30, 2011.

[21] Erik F Tjong Kim Sang and Sabine Buchholz. Introduction to the conll-2000 shared task: Chunking. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*, pages 127–132. Association for Computational Linguistics, 2000.

[22] Lin Xiao. Dual averaging methods for regularized stochastic learning and online optimization. *The Journal of Machine Learning Research*, 9999:2543–2596, 2010.