Interactive machine learning using BIDMach

Biye Jiang
Computer Science Division
University of California
Berkeley, CA 94720
`bjiang@cs.berkeley.edu`

John Canny
Computer Science Division
University of California
Berkeley, CA 94720
`jfc@cs.berkeley.edu`

Abstract

Machine learning is growing in importance in industry, the sciences, and many other fields. In many and perhaps most of these applications, users need to trade off competing goals and build different model prototypes rapidly, which requires much human intelligence and is time consuming. Therefore, *interactive customization and optimization* aims to help expert incorporate secondary criteria into the model-generation process in an interactive way.

In this paper we describe the design of BIDMach machine learning system, with an emphasis of performing customized and interactive model optimization. The keys of the design are (i) a machine learning architecture which is modular, and support primary and secondary loss functions (ii) high-performance training so that non-trivial models can be trained in real-time (using roofline design and GPU hardware) (iii) highly-interactive visualization tools that support dynamic creation of visualizations and controls to match the bespoke criteria being optimized.

Also, when we later turn to deep neural networks which require hours or days for training, we discuss how our current framework could be extended to support monitoring and optimizing learning schedule.

1 BIDMach: high-performance, customized machine learning framework

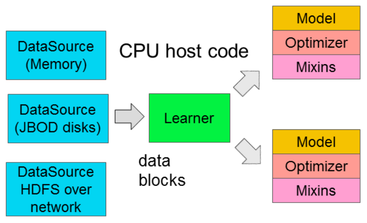

Figure 1: BIDMach's Architecture

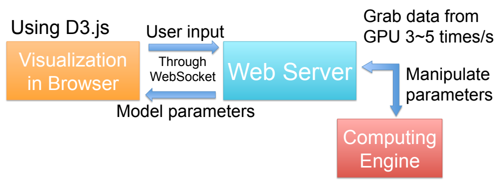

Figure 2: Visualization Architecture

The first key to interactive, customized machine learning is an architecture which supports it. BIDMach [5] is a new machine learning toolkit which has demonstrated extremely high performance with modest hardware (single computers with GPUs), and which has the modular design shown in Fig 1. BIDMach uses minibatch updates, typically many per second, so that models are being updated continuously. This is a good match to interactive modeling, since the effects of analysts actions will be seen quickly. Rather than a single model class, models comprise first a primary model, which typically outputs the model loss on a minibatch and a derivative or other update (Gibbs sampling for example) for it. Next an optimizer is responsible for updating the model given gradients. Finally, "mixin" represents secondary constraints or likelihoods. Primary models and mixins are combined with a weighted sum. In our interactive context, these weights are set interactively. Details of the mixins will be described in the next section.

BIDMach supports a variety of machine learning models, and new models or components could be easily implemented in high level Scala language. Besides productivity, high performance is very important for interactivity. Using roofline design to optimize computational kernels toward hardware limits and to fully leverage the performance of single machine, BIDMach has reduced the running time of many non-trivial ML tasks from hours to minutes. And even for models that take minutes to train fully, the effects of parameter changes are typically visible in seconds.

2 Architecture for interactive modeling

2.1 Mixin functions as secondary criteria

Model customization is useful for both supervised and unsupervised problems. Unsupervised learning involves a certain amount of arbitrariness in the criteria for the best latent state [6]. But even fully supervised models (e.g. click prediction models) may be subject to a variety of secondary constraints or "business logic" desiderata.

Therefore, different evaluation criteria are often used to measure model performance. For example, clustering usually uses a measure of model/centroid similarity, and may use intra-cluster coherence measures or inter-cluster distance as well. ROC AUC, DCG, and precision-recall measures are also commonly used to evaluate supervised models.

But usually the model itself optimizes a much simpler criterion, which follows by some filtering steps in order to fit the model into evaluation criteria or constrains. Clearly one should get better scores for these criteria if they were directly optimized as part of training. In BIDMach, we deal with this problem by associating a variety of secondary or "mixin" criteria as part of the learning process. In our design, mixin is a built-in component which is separated from the model class. Therefore it can also be shared across different models. We use a linear combination of cost functions for primary and secondary criteria:

$$arg \min_{\theta} f(\theta, d) + \sum_{i} \lambda_i * g_i(\theta)$$

Where θ is the model parameters, d is data, f is the primary cost function defined by the model and g_i are the user-defined mixin functions. The weights λ_i are "controls" that could be dynamically adjusted by the analyst as part of training. One simple example of a mixin function is just the L1-regularization which can enforce sparsity.

Such design makes it very easy to implement new customized model, which is very important for iteratively prototyping. After building the model, for the primary criterion f and each secondary criteria g_i there should be at least one dynamic graphic that captures changes in that criterion in an intuitive way, as shown in Fig 3. The analyst watches these as each of the controls are adjusted to monitor the tradeoff between them.

2.2 Controlling the learning schedule

Besides adding mixin functions, the BIDMach engine also provides a very convenient way for users to specify customized learning schedule, like setting different learning rates in different learning stages. Also, for models like LDA [3], instead of computing the gradient of the model, Gibbs sampling is used for optimizing the likelihood. To be specific, Gibbs sampling is used to estimate hidden state and model parameters in a joint distribution $P(X, Z, \Theta)$, where X is the observed data, Z is the hidden state and θ is the model parameters. In each iteration, a new model estimation will be computed, and merged into the old model parameters in an online way [9].

Learning rates could be used to control such learning procedure, and users can also control the sampling variance by using SAME sampling [8], which draw K independent copies rather than one from the hidden state distribution Z. Setting different K will lead to very different sampling behavior. This gives us extra flexibility to control the learning procedure. As discussed in [15], using annealing schedule for K can lead to a better convergence rate.

2.3 Client-Server architecture

In order to interact with the users, we use a client-server architecture with 3 components as shown in Fig 2: a computing (BIDMach) engine, a web server, and a web based front end. As the model is being trained in the engine, the front end will receive real-time updates for each minibatch via the web server. User can also change weights of the mixin functions via the interface and observe the effects on the fly. Therefore we use WebSockets for bi-directional communication between client and server. And for simplicity and extensibility, we use JSON as the over-the-wire exchange format. Such architecture is well-suited for the cloud environment, where computation is usually done remotely.

3 Visual interface

In the client side, we implement a web based interface which uses D3.js [4] for data visualization. D3.js is very flexible and has good support for animation. We provide several different kinds of visualization that users could use them to customize their dashboard, as shown in Fig 3. Since we are optimizing an additive function which consists of a main loss term as well as several Mixin terms. The value of the cost functions can reflect how model behaves under each criteria. Therefore visualizing the main loss function as well as other Mixin functions as streaming data is very useful. Especially when we change the control parameters, it will reflect how algorithm responses to the user control and whether the tradeoff for Mixin functions may affect the general model performance.

Besides, we also have model-specific visualizations to provide a natural interpretation of the model directly. For image models, we can directly visualize the image patches. And for topic models like LDA, we visualize the weights of the model matrix using similar designs from [7], as show in Fig 3.

Also, users can create sliders in the dashboard, which are used to change hyper-parameters of the model. As described above, all the changes will be send back to the engine immediately and the effect will be taken in the next mini-batch.

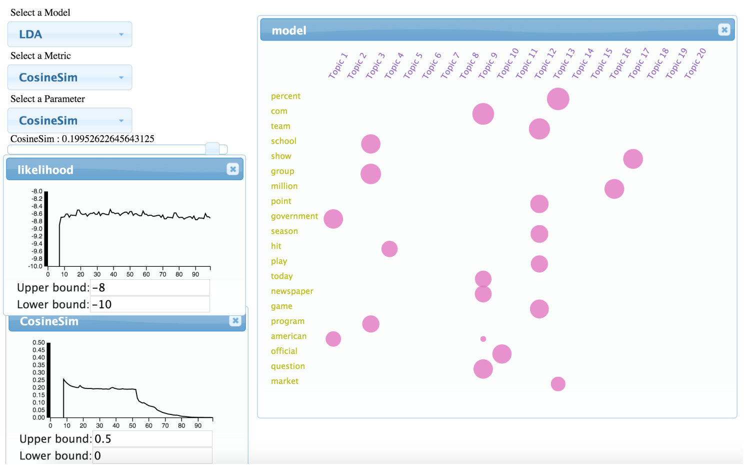
Select a Model: LDA
Select a Metric: CosineSim
Select a Parameter: CosineSim
CosineSim : 0.19952622645643125
Upper bound: -8
Lower bound: -10
Upper bound: 0.5
Lower bound: 0

Figure 3: Visualization for LDA

4 Use Case

Fig 3 demonstrates a real use case of our toolkit. When training a LDA [3] topic model on UCI NYTimes dataset [11], we at the same time minimize its topic-wised cosine similarity metric:

$$g(T) = \sum_{1 \le i,j \le N} T_i \cdot T_j$$

Where T_i is the ith Topic vector (probability distribution of the words), and N is the number of topics.

Optimizing this mixin criteria is to make topics become less similar to each other. However, it is not easy to set a good weight for this mixin function. Typically, the mixin function will not affect the model at all if the weight is too small, while a larger weight may lead to significant drop in model likelihood. Therefore, by observing the model likelihood as well as the secondary criteria at the same time, we are able to tradeoff between them based on the real-time feedback. After some tuning, we can drop the cosine similarity to be almost 0, while the model likelihood is almost the same as the origin one.

For model in such a scale, BIDMach takes around 30s for one entire pass. However, as show in Fig 3, the model converges in only several seconds, and all the responses to user control happen in around one second as well. On the other hand, the model matrix visualization helps user confirm model correctness and also shows that optimizing the topic-wised cosine similarity will actually lead to a sparse model.

5 Scaling to deep neural network

Deep neural networks, on the other hand, usually take hours or even days to train. And people tend to use global learning rate schedule for different layers. This is because tuning hyper-parameters either heavily relies on human expert heuristic [1] or uses gradient-free optimization techniques [2]. Both of them could not scale well as the number of hyper-parameters increase.

Recently, Maclaurin et al [12] use the whole training history (model parameters in each epoch) to compute the exact gradient for the hyper-parameters regarding to the final loss, therefore make it possible to directly optimize hyper-parameters in different epoch/layers. Their proof of concept experiments found that a better learning schedule may have different learning rate for different layer and the learning rates are not always decreasing. These shed light on the possibility to use a more complex learning schedule and find better hyper-parameters using history of learning procedure.

But their current approach is still limited to small scale networks. For large scale networks, logging the entire model is not practical and not efficient. Much information is actually redundant. Therefore, logging aggregated data via user defined functions for each layer is a better choice. However, few work have been done to really investigate the dynamics of the training procedure. Several work like [14, 10] use visualization to interpret the results and model parameters of deep neural networks, but they didn't analyze the training procedure. On the other hand, most deep learning toolkit provide the function to monitor training progress. [13] also describes adding logging and visualization into the framework. But most logging functions they provide are designed for checkpoint or model selection, while in our case, the dynamics of the learning procedure such as magnitude of the gradient flow or variation or model parameters is more important. Once such aggregated data becomes available, online/offline exploration tools could be built to support analysis and optimization of the learning procedure. This is still an ongoing work and we will describe it in details in future papers.

6 Conclusion

We describe the design of BIDMach which supports customized and interactive model optimization and demonstrate its function via a real use case. The use of mixin functions, learning schedule control and visual interface provide great flexibilities for model prototyping. We also describe the ongoing work about finding better learning schedule for deep neural network. A better logging component will be designed to support analysis and optimization of training procedure.

References

[1] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, pages 437–478. Springer, 2012.

[2] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[4] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2301–2309, 2011.

[5] J. Canny and H. Zhao. Big data analytics with small footprint: Squaring the cloud. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 95–103. ACM, 2013.

[6] J. Chang, S. Gerrish, C. Wang, J. L. Boyd-graber, and D. M. Blei. Reading tea leaves: How humans interpret topic models. In *Advances in neural information processing systems*, pages 288–296, 2009.

[7] J. Chuang, C. D. Manning, and J. Heer. Termite: Visualization techniques for assessing textual topic models. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pages 74–77. ACM, 2012.

[8] A. Doucet, S. J. Godsill, and C. P. Robert. Marginal maximum a posteriori estimation using markov chain monte carlo. *Statistics and Computing*, 12(1):77–84, 2002.

[9] M. Hoffman, F. R. Bach, and D. M. Blei. Online learning for latent dirichlet allocation. In *advances in neural information processing systems*, pages 856–864, 2010.

[10] A. Karpathy, J. Johnson, and F.-F. Li. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[11] M. Lichman. UCI machine learning repository, 2013.

[12] D. Maclaurin, D. Duvenaud, and R. P. Adams. Gradient-based hyperparameter optimization through reversible learning. *arXiv preprint arXiv:1502.03492*, 2015.

[13] B. van Merriënboer, D. Bahdanau, V. Dumoulin, D. Serdyuk, D. Warde-Farley, J. Chorowski, and Y. Bengio. Blocks and fuel: Frameworks for deep learning. *CoRR*, abs/1506.00619, 2015.

[14] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014*, pages 818–833. Springer, 2014.

[15] H. Zhao, B. Jiang, J. F. Canny, and B. Jaros. Same but different: Fast and high quality gibbs parameter estimation. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1495–1502, New York, NY, USA, 2015. ACM.