Faster training for machine learning

Edo Collins
LIONS, DATA, EDIC, EPFL

***Abstract*—In this report I will describe some of the challenges that face machine learning in light of ever growing datasets. I will make the case that in addition to the recent improvements in computer hardware, adoption of more efficient training algorithms could have a big impact on the feasibility of training and using large-scale models. The first paper serves to illustrate the need for using large, deep models, with many degrees-of-freedom, while the other two propose ways with which such models could be trained more efficiently. I conclude with a look at current work done to extend and combine these methods, as well as directions for future research.**

***Index Terms*—thesis proposal, candidacy exam write-up, EDIC, EPFL**

I. Introduction

MUCH has been said about the new age of “big data”. For machine learning, large-scale datasets, large both in number of samples as well as in the dimensionality of the data, call for the use of large-scale models.

For a model to have sufficient capacity to capture the properties of objects in the data and the potentially complex relationships between them, it must have a large enough number of trainable parameters, or degrees-of-freedom, but not so many as to overfit the data, i.e. fit the sampling noise in the training data to the point of poor generalization to unseen data.

Proposal submitted to committee: June 29th, 2015; Candidacy exam date: July 6th, 2015; Candidacy exam committee: Sabine Süsstrunk, Volkan Cevher, Christoph Koch, Aude Billard.

The high overhead associated with training, tuning and evaluating large-scale models has been somewhat alleviated with improvements in computer hardware, such as ever cheaper memory and ever faster computation. At the same time, major saving could be achieved by using more efficient training algorithms that cut down on running times. The development of such algorithms is the focus of this report.

In what follows, I review three papers. The first, “ImageNet Classification with Deep Convolutional Neural Networks”, Krizhevsky et al., 2012 [1], is a motivating example of the successful application of large, deep models to large-scale machine learning tasks. While performance is strong, reported training times are very long, making model selection a tedious, almost impractical task.

The two following papers demonstrate how training could be sped-up by changing some of the basic assumptions made by the age-old gradient descent procedure. More specifically, the first paper, “Stochastic Spectral Descent for Restricted Boltzmann Machines”, Carlson et al., 2015 [2], involves substituting a quadratic prior on the geometry of the error surface by a different prior, argued to be more suitable in many machine learning settings. The second paper, “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”, Duchi et al., 2011 [3], relaxes the assumption that the error is isotropic, and allows for adaptation to local curvature in a way that minimizes a bound on the empirical *regret*.

II. ImageNet classification with deep convolutional neural networks

In [1], the authors train a deep convolutional neural net (CNN) for object recognition on the ImageNet dataset. This dataset contains over 15 million high-resolution images, labeled with about 22,000 categories. Out of these, the authors used the subset provided for the 2010 and 2012 ImageNet Large-Scale Visual Recognition Challenge, which contains about 1.2 million images for training, 50,000 images for validation, and an additional 150,000 for testing.

For this dataset, the authors trained a deep CNN made up eight layers - five convolutional layers followed by three fully-connected layers. The use of convolution is well motivated in this context, as it greatly reduces the number of parameters, compared to fully-connected layers, at the price of assumptions that are quite reasonable for image data, namely the spatial organization of pixels on a 2-D grid (per channel) and the restricted, local dependence of pixel values on their neighbors. As a result, the convolutional layers account for about 5% of their 60 million model parameters, the other 95% consisting of the three fully-connected layers that follow.

To prevent overfitting, data augmentation was used, where five patches were extracted from each image, resulting in a five-fold increase in training data. Additionally, a technique called Dropout was implemented for the first two fully-connected layers [4]. With Dropout, half the units in each layer are stochastically excluded from a training iteration. According to the authors, this has the affect of preventing complex co-adaptations between units, encouraging them to individually capture more robust features. In fact, it has been shown that Dropout acts as data-dependent regularization [5], which similar to AdaGrad (section V), is more favorable to the learning of indicative yet sparse features.

To speed up training, the net was implemented to run on two GPUs in parallel. As in [6], the net parallelizes over convolutional layers by sharing parameters and splitting the data, and parallelizes over fully-connected layers by sharing the data and splitting parameters. This efficient parallelization keeps resource-sharing to a minimum and results in faster run times.

Another considerable speed up is reported to be due to using rectified-linear units (ReLU), i.e. $f(x) = \max(0, x)$, in place of traditional sigmoidal activation functions, e.g. logisitc or hyperbolic tangent. This is because the gradient value of the ReLU is always 1 (or zero), as opposed to the infinitesimal gradient value of a sigmoidal unit once it has 'saturated' i.e. moved away from it's near-linear regime onto a plateau. By completely avoiding the problem of saturation, learning proceeds faster.

In spite of these efforts, training is reported to have taken up to six days for a single instance of the model. The authors also report that without exception, more training iterations resulted in better performance. In other words, training was halted prematurely, before full convergence was achieved, making the case for faster converging training procedures.

The work presented in that paper shows state-of-the-art results, by a significant margin: from the previous state-of-the-art of 26,2% classification error, the deep net achieved 15.3% error. By speeding up taining time, this type of model could be used more widely, and better tuning schemes could be considered which would lead to better results.

III. Beyond gradient descent

Gradient descent (GD) is likely the most widely used optimization algorithm in machine learning. It is also the simplest and most intuitive: the negative gradient is the direction of steepest descent, and moving in that direction is guaranteed to find a minimum of the objective function, provided an appropriate step size is used. If the objective is convex, the solution is optimal globally. However, gradient descent makes some assumption that often lead to sub-optimal performance on many real-world tasks.

A. Majorization minimization

Consider the problem of linear regression:

$$\beta^* = \arg\min_{\beta} \ell(\beta), \tag{1}$$

$$\ell(\beta) = \frac{1}{N}\sum_{i=1}^{N}(\beta^\top x - y_i)^2. \tag{2}$$

where, $\beta, x_i \in \mathcal{R}^n$ and $y_i \in \mathcal{R}$. When the matrix $\boldsymbol{X} = [x_1^\top, \cdots, x_N^\top]$ is full-rank, the optimality condition for this problem leads to a closed-form solution for β^*:

$$\nabla\ell(\beta^*) = X^\top(X\beta^* - y) = 0 \rightarrow \tag{3}$$

$$\beta^* = (X^\top X)^{-1}X^\top y \tag{4}$$

In other words, the minimum is attained in one "step". For slightly more complicated problems however, e.g. logistic regression, such a solution cannot be derived:

$$\beta^* = \arg\min_{\beta}\sum_{i=1}^{N} y_i \log\sigma(\beta^\top x_i) + (1-y_i)\log(1-\sigma(\beta^\top x)) \tag{5}$$

In this case, the optimality condition, $\nabla\ell(\beta^*) = X^\top(\sigma(X\beta^*) - y) = 0$, does *not* yield a closed-form solution for the optimal model parameters.

In such cases a majorization-minimization approach is adopted, where at every iteration an upper-bound to the true function is established, whose minimum *is* known analytically. Consider the familiar gradient descent update:

$$\beta_{k+1} = \beta_k - \alpha\nabla\ell(\beta_k) \tag{6}$$

This update is in fact the minimizer of the following bound around β_k:

$$\ell(\beta) \leq \ell(\beta_k) + \langle\nabla\ell(\beta_k), \beta - \beta_k\rangle + \frac{1}{2\alpha}\|\beta - \beta_k\|_2^2 \tag{7}$$

The use of the Euclidean norm in the above bound imposes a set of assumptions, that often don't hold in real-world tasks, leading to sub-optimal performance.

The first assumption is that a quadratic upper-bound is appropriate for the function $\ell(\beta)$ which is not necessarily the case. In fact, as will be discussed in section IV, for many typical machine learning problem settings, the infinity norm leads to tighter bounds and algorithms that perform better in practice.

The second assumption is that the curvature of the error is isotropic, i.e. changes at the same rate in all directions. When working with real data, however, the error surface often displays varying curvature in different directions. What this means in practice is that using a global step-size could lead to poor performance, as for some directions it would be too big or too small. By considering a bound that captures some curvature information, better performance could be attained. This idea will be reviewed in section V.

IV. Stochastic spectral descent for restricted Boltzmann machines

Stochastic spectral descent (SSD) is motivated by the observation that the ubiquitous log-sum-exp function, i.e. $\mathrm{lse}_\omega(\beta) = \log \sum_i \omega_i e^{\beta_i}$, suggests a geometry that is better approximated by the infinity- or max-norm, than by the Euclidean norm. In particular, the lse function has the following bound:

$$\mathrm{lse}_\omega(\beta) \leq \mathrm{lse}\,\beta_k + \langle \nabla \mathrm{lse}_\omega(\beta_k), \beta - \beta_k \rangle + \frac{1}{2}\|\beta - \beta_k\|_\infty^2 \tag{8}$$

using this bound, an upper bound for the restricted Boltzmann machine (RBM) loss function is established, with respect to the matrix of connection weights $\boldsymbol{W}$, expressed in the Schatten-infinity norm:

$$\begin{aligned} F(\boldsymbol{W}) \leq F(\boldsymbol{W}_k)\mathrm{tr}((\nabla F(\boldsymbol{W}_k))(\nabla F(\boldsymbol{W} - \nabla F(\boldsymbol{W}_k)) \quad (9) \\ + \frac{MJ}{2}\|\boldsymbol{W} - \nabla F(\boldsymbol{W}_k)\|_{S^\infty}^2 \end{aligned}$$

Recall that Schatten matrix norms operate on the singular values of the matrix. Therefore, the Schatten-infinity norm, also called the spectral norm, is equal to the largest singular value of the matrix. Unlike gradient descent, minimizing this bound results in a step which is not in the direction of the negative gradient. It can be shown that for an arbitrary norm, the minimizer:

$$\min_y F(x)\langle \nabla F(x), y - x \rangle + \frac{1}{2\alpha}\|y - x\|^2 \tag{10}$$

is given by:

$$y = x - \alpha \left[\nabla F(x)\right]^{\#} \tag{11}$$

where $s^{\#} = \arg\min_x \left\{\langle s, x \rangle - \frac{1}{2}\|x\|^2\right\}$ is the sharp-operator. Notice the solution of the sharp-operator need not be unique, in which case any of its solutions may be chosen. When shown above, when the norm is the Euclidean norm, Eq. (11) leads to gradient descent. When the norm is the spectral norm, the solution to the sharp-operator leads to the following update:

$$\nabla F(\boldsymbol{W}) = U\Sigma V^\top, \tag{12}$$

$$\boldsymbol{W}_{k+1} = \boldsymbol{W} - \alpha\|\Sigma\|_{S^1} UV^\top. \tag{13}$$

where $U\Sigma V^\top$ is the singular-value decomposition (SVD) of the gradient, and the step size α can be seen as the inverse of the Lipschitz constant under which $\nabla F(\boldsymbol{W})$ is Lipschitz-continuous, e.g. for the RBM bound we could set $\alpha = \frac{1}{L} = \frac{1}{MJ}$.

Applying this update iteratively results in the SSD algorithm, which has been shown to achieve state-of-the-art performance for RBMs on such tasks as hand-written digit classification on the MNIST dataset, and object recognition with the Caltech-101 Silhouette dataset (see Figure 1).

V. Adaptive subgradient methods for online learning and stochastic optimization

AdaGrad is an algorithm that addresses the anisotropic curvature imposed by real-world data sets. The intuition given by the authors is that certain features or dimension of the input are rare but highly indicative. As a result, the objective is not highly-curved with respect to model parameters that correspond to those rare directions, and big steps could be taken in those direction - steps that would otherwise be small if determined by a global step size that accommodates the direction of highest curvature.

Given a matrix $\boldsymbol{A} \succeq 0$, consider the weighted dot product $\langle \cdot, \cdot \rangle_{\boldsymbol{A}} = \langle \cdot, \boldsymbol{A} \cdot \rangle$ and the corresponding weighted Euclidean norm $\|\cdot\|_{\boldsymbol{A}} = \sqrt{\langle \cdot, \cdot \rangle_{\boldsymbol{A}}}$.

When $\boldsymbol{A} = \nabla^2 \ell(\beta)$, the Hessian, minimizing the bound gotten by substituting the Euclidean norm in Eq. (7) with the scaled norm retrieves the well-known Newton method. Since the Hessian matrix is often expensive to compute or store in memory, various proposals have been made to instead use a quantity with Hessian-like properties, giving rise to so-called Quasi-Newton methods.

AdaGrad proposes choosing $\boldsymbol{A}$ such as to minimize the *regret*. The regret of the training procedure at time T is defined as:

$$R(T) = \sum_{t=1}^{T} \ell_t(\beta_t) - \ell_t \beta^* \tag{14}$$

where β_t is the approximation to β^*, the optimal model parameters, at iteration t. Notice the loss ℓ is also indexed by t to reflect the common practice of using different mini-batches data at every iteration to estimate the gradient over the whole data set.

Using the weighted Euclidean norm, the regret can be bounded as:

$$R(T) \leq \frac{1}{2\alpha}\|\beta_1 - \beta^*\|_{\boldsymbol{A}}^2 + \frac{\alpha}{2}\sum_{t=1}^{\top}\|\nabla \ell_t(\beta_t)\|_{\boldsymbol{A}^{-1}}^2 \tag{15}$$

Consider the problem of minimizing $\sum_{t=1}^{T}\|\nabla \ell_t(\beta_t)\|_{\boldsymbol{A}^{-1}}$ such that $\boldsymbol{A} \succeq 0$ and $\mathrm{tr}(\boldsymbol{A}) \leq c$, where the last constraint is intended to control the tradeoff between the norm of $\boldsymbol{A}$ and its transpose in Eq. (15). The solution to this problem is in fact:

$$\boldsymbol{A} = c\left(\sum_{t=1}^{T} \nabla \ell(\beta)_t \nabla \ell(\beta)_t^\top\right)^{\frac{1}{2}}. \tag{16}$$

Since the dimensions of $\boldsymbol{A}$ are the same as those of the Hessian, a less computation- and memory-intensive version considers only the diagonal of this matrix:

$$\boldsymbol{A} = c\mathrm{diag}\left(\sum_{t=1}^{T} \nabla \ell(\beta)_t \nabla \ell(\beta)_t^\top\right)^{\frac{1}{2}}. \tag{17}$$

This leads to the AdaGrad algorithm in which every iteration follows the update:

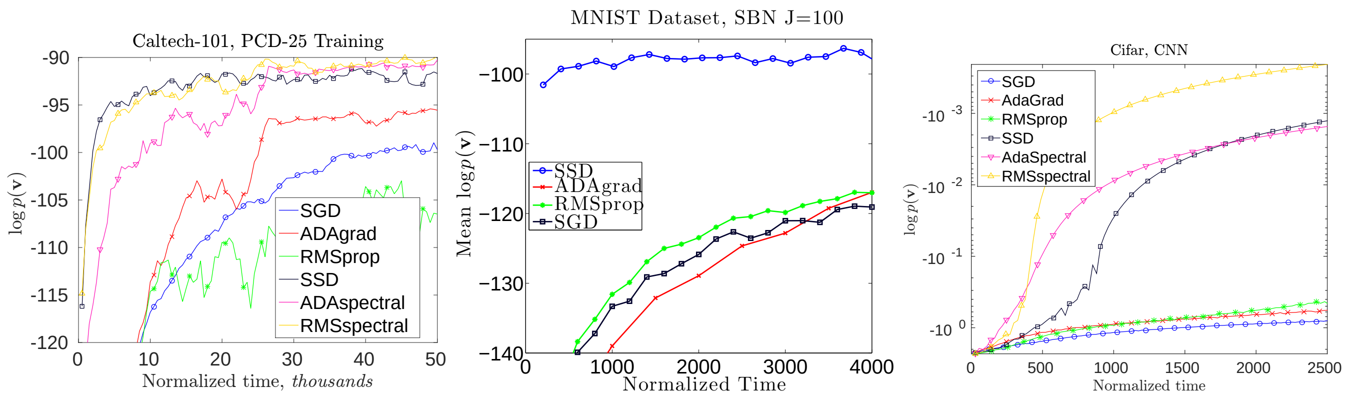

Fig. 1. A normalized time unit is 1 stochastic gradient descent (SGD) iteration (Left) Log-likelihood of training on Caltech-101 Silhouettes with RBM (Middle) Log-likelihood of training on MNIST digits with SBN (Right) Log-likelihood of the current training batch on CIFAR-10 images with a CNN

$$\beta_{k+1} = \beta_k - \alpha(\epsilon \boldsymbol{I} + \boldsymbol{A}^{-1})\nabla\ell(\beta_k) \tag{18}$$

where ϵ is a damping factor that controls oscillations in convergence and has to be chosen with cross-validation.

AdaGrad has been shown to outperform non-adaptive methods on a variety of datasets, including image ranking with ImageNet, digit classification on MNIST and text classification on the Reuters RCV1 corpus.

VI. Current and Future Work

The SSD algorithm described above was originally introduced for RBMs, by deriving an appropriate upper bound to the RBM loss function. In a recent collaboration, I and others have extended SSD to other popular models, such as sigmoid belief nets (SBN), replicated-softmax RBMs, replicated-softamx belief nets and neural networks. Empirical results show considerable improvement over GD, both in terms of convergence speed as well as in model performance.

Another natural extension was to combine the non-Euclidean bounds of SSD with the locally adaptive updates of AdaGrad, by considering updates based on a weighted infinity norm bound. Updates for adaptive spectral descent take the form:

$$A^{-\frac{1}{2}}\nabla F(\boldsymbol{W}) = U\Sigma V^{\top}, \tag{19}$$

$$\boldsymbol{W}_{k+1} = \boldsymbol{W} - \alpha\|\Sigma\|_{S^1} A^{-\frac{1}{2}} U V^{\top}. \tag{20}$$

This has led us to propose algorithms such as AdaSpectral and RMS-Spectral, based on RMSprop. These algorithms have already exhibited an order of magnitude speed up compared to GD, AdaGrad and RMSprop. This holds in spite of the additional overhead associated with performing SVD on the gradient. Some of these results are shown in Figure 1.

Future work will include experimenting with other ways of adapting to local geometry, e.g. quasi-Newton methods, applying suitable regularization schemes for adaptive spectral descent methods, as well considering more complex machine learning tasks. For instance, a well-known issue with learning deep or recurrent neural networks is that of vanishing or exploding gradients, i.e., gradients tend to grow or diminish exponentially as they are backpropogated through network layers. We would like to examine our methods, both theoretically and empirically, to see if they offer similar improvements in this setting.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] D. Carlson, V. Cevher, and L. Carin, “Stochastic spectral descent for restricted boltzmann machines,” in *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, 2015, pp. 111–119.

[3] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[5] S. Wager, S. Wang, and P. S. Liang, “Dropout training as adaptive regularization,” in *Advances in Neural Information Processing Systems 26*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 351–359. [Online]. Available: http://papers.nips.cc/paper/4882-dropout-training-as-adaptive-regularization.pdf

[6] A. Krizhevsky, “One weird trick for parallelizing convolutional neural networks,” *arXiv preprint arXiv:1404.5997*, 2014.