ADASECANT: Robust Adaptive Secant Method for Stochastic Gradient

Caglar Gulcehre
Université de Montréal
caglar.gulcehre@umontreal.ca

Marcin Moczulski
University of Oxford
marcin.moczulski@stcatz.ox.ac.uk

Yoshua Bengio
Université de Montréal
bengioy@iro.umontreal.ca

Abstract

Stochastic gradient algorithms have been the main focus of large-scale learning problems and led to important successes in machine learning. The convergence of SGD depends on the careful choice of learning rate and the amount of the noise in stochastic estimates of the gradients. In this paper, we propose a new adaptive learning rate algorithm, which utilizes curvature information for automatically tuning the learning rates. The information about the element-wise curvature of the loss function is estimated from the local statistics of the stochastic first order gradients. We further propose a new variance reduction technique to speed up the convergence. In our experiments with deep neural networks, we obtained better performance compared to the popular stochastic gradient algorithms.

1 Introduction

In this paper, we develop a stochastic gradient algorithm that reduces the burden of extensive hyper-parameter search for the optimization algorithm. The proposed algorithm exploits a low variance estimator of curvature of the cost function and uses it to obtain an automatically tuned adaptive learning rate for each parameter.

In the deep learning and numerical optimization, several papers suggest using a diagonal approximation of the Hessian (second derivative matrix of the cost function with respect to parameters), in order to estimate optimal learning rates for stochastic gradient descent over high dimensional parameter spaces [3, 14, 9]. A fundamental advantage of using such approximation is that inverting it is a trivial and cheap operation. However generally, for neural networks, the inverse of the diagonal Hessian is usually a bad approximation of the diagonal of the inverse of Hessian. Examples options of obtaining a diagonal approximation of Hessian are the Gauss-Newton matrix [10] or by finite differences [13]. Such estimations may however be sensitive to noise due to the stochastic gradient. [14] suggested a reliable way to estimate the local curvature in the stochastic setting by keeping track of the variance and average of the gradients.

In this paper, we followed a different approach: instead of using a diagonal estimate of Hessian, we proposed to estimate curvature along the direction of the gradient and we apply a new variance reduction technique to compute it reliably. By using root mean square statistics, the variance of gradients are reduced adaptively with a simple transformation. We keep track of the estimation of curvature using a technique similar to that proposed by [14], which uses the variability of the expected loss. Standard adaptive learning rate algorithms only scale the gradients, but regular Newton-like second order methods, can perform more complicate transformations, e.g. rotating the gradient vector. Newton and quasi-newton methods can also be invariant to affine transformations in the parameter space. Our proposed **Adasecant** algorithm is basically a stochastic rank-1 quasi-Newton method. But in comparison with other adaptive learning algorithms, instead of just scaling the gradient of each parameter, Adasecant can also perform an affine transformation on them.

2 Directional Secant Approximation

Directional Newton is a method proposed for solving equations with multiple variables [11]. The advantage of directional Newton method proposed in [11], compared to Newton's method is that, it does not require a matrix inversion and still maintains a quadratic rate of convergence.

In this paper, we developed a second-order directional Newton method for nonlinear optimization. Step-size $\mathbf{t}^k$ of update Δ^k for step k can be written as if it was a diagonal matrix:

$$\Delta^k = \mathbf{t}^k \odot \nabla_{\boldsymbol{\theta}} \mathrm{f}(\boldsymbol{\theta}^k) = \mathrm{diag}(\mathbf{t}^k)\nabla_{\boldsymbol{\theta}} \mathrm{f}(\boldsymbol{\theta}^k) = -\,\mathrm{diag}(\mathbf{d}^k)(\mathrm{diag}(\mathbf{H}\mathbf{d}^k))^{-1}\nabla_{\boldsymbol{\theta}} \mathrm{f}(\boldsymbol{\theta}^k) \tag{1}$$

where $\boldsymbol{\theta}^k$ is the parameter vector at update k, f is the objective function and $\mathbf{d}^k$ is a unit vector of direction that the optimization algorithm should follow. Denoting by $\mathbf{h}_i = \nabla_{\boldsymbol{\theta}} \frac{\partial \mathrm{f}(\boldsymbol{\theta}^k)}{\partial \theta_i}$ the ith row of the Hessian matrix $\mathbf{H}$ and by $\nabla_{\boldsymbol{\theta}_i} f(\boldsymbol{\theta}^k)$ the ith element of the gradient vector at update k, a reformulation of Equation 1 for each diagonal element of the step-size $\mathrm{diag}(\mathbf{t}^k)$ is:

$$\Delta_i^k = -t_i^k \nabla_{\boldsymbol{\theta}_i} \mathrm{f}(\boldsymbol{\theta}^k) = -d_i^k \frac{\nabla_{\boldsymbol{\theta}_i} \mathrm{f}(\boldsymbol{\theta}^k)}{\mathbf{h}_i^k \mathbf{d}^k}, \text{ so effectively } t_i^k = \frac{d_i^k}{\mathbf{h}_i^k \mathbf{d}^k} \tag{2}$$

We can approximate the per-parameter learning rate t_i^k following [1]:

$$t_i^k = \frac{d_i^k}{\mathbf{h}_i^k \mathbf{d}^k} \approx \frac{t_i^k d_i^k}{\nabla_{\boldsymbol{\theta}_i} \mathrm{f}(\boldsymbol{\theta}^k + \mathbf{t}^k \mathbf{d}^k) - \nabla_{\boldsymbol{\theta}_i} f(\boldsymbol{\theta}^k)} \tag{3}$$

Please note that alternatively one might use the R-op to compute the Hessian-vector product for the denominator in Equation 3 [15].

To choose a good direction $\mathbf{d}^k$ in the stochastic setting, we use a block-normalized gradient vector for each weight matrix $\mathbf{W}_k^i$ and bias vector $\mathbf{b}_k^i$ where $\boldsymbol{\theta} = \left\{\mathbf{W}_k^i, \mathbf{b}_k^i\right\}_{i=1\cdots k}$ at each layer i and update k, i.e. $\mathbf{d}_{\mathbf{W}_k^i}^k = \frac{\nabla_{\mathbf{W}_k^i} \mathrm{f}(\boldsymbol{\theta})}{||\nabla_{\mathbf{W}_k^i} \mathrm{f}(\boldsymbol{\theta})||_2}$ and $\mathbf{d}_k = \left[\mathbf{d}_{\mathbf{W}_k^0}^k \mathbf{d}_{\mathbf{b}_k^0}^k \cdots \mathbf{d}_{\mathbf{b}_k^l}^k\right]$ for a neural network with l layers. Block normalization of the gradient adds an additional noise, but in practice we did not observe any negative impact of it. We conjecture that this is due to the angle between the stochastic gradient and the block-normalized gradient still being less than 90 degrees. The update step is defined as $\Delta_i^k = t_i^k d_i^k$. The per-parameter learning rate t_i^k can be estimated with the finite difference approximation,

$$t_i^k = \frac{\Delta_i^k}{\nabla_{\boldsymbol{\theta}_i} \mathrm{f}(\boldsymbol{\theta}^k + \Delta^k) - \nabla_{\boldsymbol{\theta}_i} \mathrm{f}(\boldsymbol{\theta}^k)}, \tag{4}$$

since, in the vicinity of the quadratic local minima,

$$\nabla_{\boldsymbol{\theta}} \mathrm{f}(\boldsymbol{\theta}^k + \Delta^k) - \nabla_{\boldsymbol{\theta}} \mathrm{f}(\boldsymbol{\theta}^k) \approx \mathbf{H}^k \Delta^k \tag{5}$$

We can therefore recover $\mathbf{t}^k$ as

$$\mathbf{t}^k = \mathrm{diag}(\Delta^k)(\mathrm{diag}(\mathbf{H}^k \Delta^k))^{-1}. \tag{6}$$

The directional secant method basically scales the gradient of each parameter with the curvature along the direction of the gradient vector and it is numerically stable.

3 Variance Reduction for Robust Stochastic Gradient Descent

Variance reduction techniques for stochastic gradient estimators have been well-studied in the machine learning literature. Both [16] and [8] proposed new ways of dealing with this problem. In this paper, we proposed a new variance reduction technique for stochastic gradient descent that relies only on basic statistics related to the gradient. Let g_i refer to the ith element of the gradient vector $\mathbf{g}$ with respect to the parameters $\boldsymbol{\theta}$ and $\mathrm{E}[\cdot]$ be an expectation taken over minibatches and different trajectories of parameters.

We propose to apply the following transformation to reduce the variance of the stochastic gradients:

$$\tilde{g}_i = \frac{g_i + \gamma_i \mathrm{E}[g_i]}{1 + \gamma_i} \tag{7}$$

where γ_i is strictly a positive real number. Let us note that:

$$\mathrm{E}[\tilde{g}_i] = \mathrm{E}[g_i] \text{ and } \mathrm{Var}(\tilde{g}_i) = \frac{1}{(1+\gamma_i)^2}\,\mathrm{Var}(g_i) \tag{8}$$

So variance is reduced by a factor of $(1+\gamma_i)^2$ compared to $\mathrm{Var}(g_i)$. In practice we do not have access to $\mathrm{E}[g_i]$, therefore a biased estimator $\overline{g_i}$ based on past values of g_i will be used instead. We can rewrite the $\tilde{g}_i$ as:

$$\tilde{g}_i = \frac{1}{1+\gamma_i} g_i + (1 - \frac{1}{1+\gamma_i})\mathrm{E}[g_i] \tag{9}$$

After substitution $\beta_i = \frac{1}{1+\gamma_i}$, we will have:

$$\tilde{g}_i = \beta_i g_i + (1-\beta_i)\mathrm{E}[g_i] \tag{10}$$

By adapting γ_i or β_i, it is possible to control the influence of high variance, unbiased g_i and low variance, biased $\overline{g_i}$ on $\tilde{g}_i$. Denoting by $\mathbf{g}'$ the stochastic gradient obtained on the next minibatch, the γ_i that well balances those two influences is the one that keeps the $\tilde{g}_i$ as close as possible to the true gradient $\mathrm{E}[g'_i]$ with g'_i being the only sample of $\mathrm{E}[g'_i]$ available:

$$\arg\min_{\beta_i} \mathrm{E}[||\tilde{g}_i - g'_i||_2^2] \tag{11}$$

It can be shown that this a convex problem in β_i with a closed-form solution (details in appendix) and we can obtain the γ_i from it:

$$\gamma_i = \frac{\mathrm{E}[(g_i - g'_i)(g_i - \mathrm{E}[g_i])]}{\mathrm{E}[(g_i - \mathrm{E}[g_i])(g^{i\prime} - \mathrm{E}[g_i]))]} \tag{12}$$

As a result, in order to estimate γ for each dimension, we keep track of a estimation of $\frac{\mathrm{E}[(g_i - g'_i)(g_i - \mathrm{E}[g_i])]}{\mathrm{E}[(g_i - \mathrm{E}[g_i])(g'_i - \mathrm{E}[g_i]))]}$ during training. The necessary and sufficient condition here, for the variance reduction is to keep γ positive, to achieve a positive estimate of γ we used the root mean square statistics for the expectations.

4 Adaptive Step-size in Stochastic Case

In the stochastic gradient case, the step-size of the directional secant can be computed by using an expectation over the minibatches:

$$\mathrm{E}_k[t_i] = \mathrm{E}_k[\frac{\Delta_i^k}{\nabla_{\boldsymbol{\theta}_i}\mathrm{f}(\boldsymbol{\theta}^k + \Delta^k) - \nabla_{\boldsymbol{\theta}_i}\mathrm{f}(\boldsymbol{\theta}^k)}] \tag{13}$$

The $E_k[\cdot]$ that is used to compute the secant update, is taken over the minibatches at the past values of the parameters.

Computing the expectation in Eq 13 was numerically unstable in stochastic setting. We decided to use a more stable second order Taylor approximation of Equation 13 around $(\sqrt{\mathrm{E}_k[(\alpha_i^k)^2]}, \sqrt{\mathrm{E}_k[(\Delta_i^k)^2]})$, with $\alpha_i^k = \nabla_{\boldsymbol{\theta}_i}\mathrm{f}(\boldsymbol{\theta}^k + \Delta^k) - \nabla_{\boldsymbol{\theta}_i}\mathrm{f}(\boldsymbol{\theta}^k)$. Assuming $\sqrt{\mathrm{E}_k[(\alpha_i^k)^2]} \approx \mathrm{E}_k[\alpha_i^k]$ and $\sqrt{\mathrm{E}_k[(\Delta_i^k)^2]} \approx \mathrm{E}_k[\Delta_i^k]$ we obtain always non-negative approximation of $\mathrm{E}_k[t_i]$:

$$\mathrm{E}_k[t_i] \approx \frac{\sqrt{\mathrm{E}_k[(\Delta_i^k)^2]}}{\sqrt{\mathrm{E}_k[(\alpha_i^k)^2]}} - \frac{\mathrm{Cov}(\alpha_i^k, \Delta_i^k)}{\mathrm{E}_k[(\alpha_i^k)^2]} \tag{14}$$

In our experiments, we used a simpler approximation, which in practice worked as well as formulations in Eq 14:

$$\mathrm{E}_k[t_i] \approx \frac{\sqrt{\mathrm{E}_k[(\Delta_i^k)^2]}}{\sqrt{\mathrm{E}_k[(\alpha_i^k)^2]}} - \frac{\mathrm{E}_k[\alpha_i^k \Delta_i^k]}{\mathrm{E}_k[(\alpha_i^k)^2]} \tag{15}$$

5 Algorithmic Details

5.1 Approximate Variability

To compute the moving averages as also adopted by [14], we used an algorithm to dynamically decide the time constant based on the step size being taken. As a result algorithm that we used will give bigger weights to the updates that have large step-size and smaller weights to the updates that have smaller step-size.

By assuming that $\bar{\Delta}_i[k] \approx \mathrm{E}[\Delta_i]_k$, the moving average update rule for $\bar{\Delta}_i[k]$ can be written as,

$$\bar{\Delta}_i^2[k] = (1 - \tau_i^{-1}[k])\bar{\Delta}_i^2[k-1] + \tau_i^{-1}[k](t_i^k \tilde{\mathbf{g}}_i^k), \text{ and } \bar{\Delta}_i[k] = \sqrt{\bar{\Delta}_i^2[k]} \tag{16}$$

This rule for each update assigns a different weight to each element of the gradient vector . At each iteration a scalar multiplication with τ_i^{-1} is performed and τ_i is adapted using the following equation:

$$\tau_i[k] = (1 - \frac{\mathrm{E}[\Delta_i]_{k-1}^2}{\mathrm{E}[(\Delta_i)^2]_{k-1}})\tau_i[k-1] + 1 \tag{17}$$

5.2 Outlier Gradient Detection

Our algorithm is very similar to [13], but instead of incrementing $\tau_i[t+1]$ when an outlier is detected, the time-constant is reset to 2.2. Note that when $\tau_i[t+1] \approx 2$, this assigns approximately the same amount of weight to the current and the average of previous observations. This mechanism made learning more stable, because without it outlier gradients saturate τ_i to a large value.

5.3 Variance Reduction

The correction parameters γ_i (Eq 12) allows for a fine-grained variance reduction for each parameter independently. The noise in the stochastic gradient methods can have advantages both in terms of generalization and optimization. It introduces an exploration and exploitation trade-off, which can be controlled by upper bounding the values of γ_i with a value ρ_i, so that thresholded $\gamma_i' = \min(\rho_i, \gamma_i)$.

We block-wise normalized the gradients of each weight matrix and bias vectors in $\mathbf{g}$ to compute the $\tilde{\mathbf{g}}$ as described in Section 2. That makes Adasecant scale-invariant, thus more robust to the scale of the inputs and the number of the layers of the network. We observed empirically that it was easier to train very deep neural networks with block normalized gradient descent.

6 Improving Convergence

Classical convergence results for SGD are based on the conditions:

$$\sum_i (\eta^{(i)})^2 < \infty \text{ and } \sum_i \eta^{(i)} = \infty \tag{18}$$

such that the learning rate $\eta^{(i)}$ should decrease [12]. Due to the noise in the estimation of adaptive step-sizes for Adasecant, the convergence would not be guaranteed. To ensure it, we developed a new variant of Adagrad [4] with thresholding, such that each scaling factor is lower bounded by 1. Assuming a_i^k is the accumulated norm of all past gradients for i^{th} parameter at update k, it is thresholded from below ensuring that the algorithm will converge:

$$a_i^k = \sqrt{\sum_{j=0}^{k} (g_i^j)^2} \text{ and } \rho_i^k = \text{maximum}(1, a_i^k) \text{ giving } \Delta_i^k = \frac{1}{\rho_i} \eta_i^k \tilde{\mathbf{g}}_i^k \tag{19}$$

In the initial stages of training, accumulated norm of the per-parameter gradients can be less than 1. If the accumulated per-parameter norm of a gradient is less than 1, Adagrad will augment the learning-rate determined by Adasecant for that update, i.e. $\frac{\eta_i^k}{\rho_i^k} > \eta_i^k$ where $\eta_i^k = \mathrm{E}_k[t_i^k]$ is the per-parameter learning rate determined by Adasecant. This behavior tends to create unstabilities during the training with Adasecant. Our modification of the Adagrad algorithm is to ensure that, it will reduce the learning rate determined by the Adasecant algorithm at each update, i.e. $\frac{\eta_i^k}{\rho_i^k} \leq \eta_i^k$ and the learning rate will be bounded. At the beginning of the training, parameters of a neural network can get 0-valued gradients, e.g. in the existence of dropout and ReLU units. However this phenomena can cause the per-parameter learning rate scaled by Adagrad to be unbounded.

In Algorithm 1, we provide a simple pseudo-code of the Adasecant algorithm.

7 Experiments

We have performed our experiments on MNIST with Maxout Networks [6] comparing Adasecant with popular stochastic gradient learning algorithms: Adagrad, Rmsprop [7], Adadelta [17] and SGD+momentum (with linearly decaying learning rate). Results are summarized in Figure 1 at the appendinx and we showed that Adasecant converges as fast or faster than other techniques, including the use of hand-tuned global learning rate and momentum for SGD, RMSprop, and Adagrad.

8 Conclusion

We described a new stochastic gradient algorithm with adaptive learning rates that is fairly insensitive to the tuning of the hyper-parameters and doesn't require tuning of learning rates. Furthermore, the variance reduction technique we proposed improves the convergence when the stochastic gradients have high variance. According to preliminary experiments presented, we were able to obtain a better training performance compared to other popular, well-tuned stochastic gradient algorithms. As a future work, we should do a more comprehensive analysis, which will help us to better understand the algorithm both analytically and empirically.

Acknowledgments

We thank the developers of Theano [2] and Pylearn2 [5] and the computational resources provided by Compute Canada and Calcul Québec. This work has been partially supported by NSERC, CIFAR, and Canada Research Chairs, Project TIN2013-41751, grant 2014-SGR-221. We would like to thank Tom Schaul for the valuable discussions. We also thank Kyunghyun Cho and Orhan Firat for proof-reading and giving feedbacks on the paper.

References

[1] Heng-Bin An and Zhong-Zhi Bai. Directional secant method for nonlinear equations. *Journal of computational and applied mathematics*, 175(2):291–304, 2005.

[2] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio. Theano: new features and speed improvements. *Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop*, 2012.

[3] Sue Becker and Yann Le Cun. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school*, pages 29–37. San Matteo, CA: Morgan Kaufmann, 1988.

[4] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[5] I. J. Goodfellow, D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, and Y. Bengio. Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*, 2013.

[6] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[7] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[8] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323, 2013.

[9] Yann LeCun, Patrice Y Simard, and Barak Pearlmutter. Automatic learning rate maximization by on-line estimation of the hessian's eigenvectors. *Advances in neural information processing systems*, 5:156–163, 1993.

[10] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[11] Yuri Levin and Adi Ben-Israel. Directional newton methods in n variables. *Mathematics of Computation*, 71(237):251–262, 2002.

[12] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[13] Tom Schaul and Yann LeCun. Adaptive learning rates and parallelization for stochastic, sparse, non-smooth gradients. *arXiv preprint arXiv:1301.3764*, 2013.

[14] Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. *arXiv preprint arXiv:1206.1106*, 2012.

[15] Nicol N Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738, 2002.

[16] Chong Wang, Xi Chen, Alex Smola, and Eric Xing. Variance reduction for stochastic gradient optimization. In *Advances in Neural Information Processing Systems*, pages 181–189, 2013.

[17] Matthew D Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

A Appendix

A.1 Derivation of Eq 11

$$\frac{\partial \mathrm{E}[(\beta_i g_i + (1-\beta_i)\mathrm{E}[g_i] - g_i')^2]}{\partial \beta_i} = 0 \tag{20}$$

$$\mathrm{E}[(\beta_i g_i + (1-\beta_i)\mathrm{E}[g_i] - g_i')\frac{\partial(\beta_i g_i + (1-\beta_i)\mathrm{E}[g_i] - g_i')}{\partial \beta_i}] = 0 \tag{21}$$

$$\mathrm{E}[(\beta_i g_i + (1-\beta_i)\mathrm{E}[g_i] - g_i')(g_i - \mathrm{E}[g_i])] = 0 \tag{22}$$

$$\mathrm{E}[(\beta_i g_i(g_i - \mathrm{E}[g_i]) + (1-\beta_i)\mathrm{E}[g_i](g_i - \mathrm{E}[g_i]) - \mathrm{E}[g_i](g_i - \mathrm{E}[g_i])] = 0 \tag{23}$$

$$\beta_i = \frac{\mathrm{E}[(g_i - \mathrm{E}[g_i])(g_i' - \mathrm{E}[g_i])]}{\mathrm{E}[(g_i - \mathrm{E}[g_i])(g_i - \mathrm{E}[g_i])]} = \frac{\mathrm{E}[(g_i - \mathrm{E}[g_i])(g_i' - \mathrm{E}[g_i])]}{\mathrm{Var}(g_i)} \tag{24}$$

Algorithm 1: Adasecant: minibatch-Adasecant for adaptive learning rates with variance reduction

repeat

- draw n samples, compute the gradients $\mathbf{g}^{(j)}$ where $\mathbf{g}^{(j)} \in \mathcal{R}^n$ for each minibatch j, $\mathbf{g}^{(j)}$ is computed as, $\frac{1}{n}\sum_{k=1}^{n} \nabla_{\boldsymbol{\theta}}^{(k)} \mathrm{f}(\boldsymbol{\theta})$
- block-wise normalize gradients of each weight matrix and bias vector
- **for** *parameter* $i \in \{1, \ldots, n\}$ **do**
  - compute the correction term by using, $\gamma_i^k = \frac{\mathrm{E}[(g_i - g_i')(g_i - \mathrm{E}[g_i])]_k}{\mathrm{E}[(g_i - \mathrm{E}[g_i])(g_i' - \mathrm{E}[g_i]))]_k}$
  - compute corrected gradients $\tilde{g}_i = \frac{g_i + \gamma_\mathrm{i} \mathrm{E}[g_i]}{1 + \gamma_\mathrm{i}}$
  - **if** $|g_i^{(j)} - \mathrm{E}[g_i]| > 2\sqrt{\mathrm{E}[(g_i)^2] - (\mathrm{E}[g_i])^2}$ or $\left|\alpha_i^{(j)} - \mathrm{E}[\alpha_i]\right| > 2\sqrt{\mathrm{E}[(\alpha_i)^2] - (\mathrm{E}[\alpha_i])^2}$ **then**
    - reset the memory size for outliers $\tau_i \leftarrow 2.2$
  - **end**
  - update moving averages according to Equation 16
  - estimate learning rate $\eta_i^{(j)} \leftarrow \frac{\sqrt{\mathrm{E}_k[(\Delta_i^{(k)})^2]}}{\sqrt{\mathrm{E}_k[(\alpha_i^k)^2]}} - \frac{\mathrm{E}_k[\alpha_i^k \Delta_i^k]}{\mathrm{E}_k[(\alpha_i^k)^2]}$
  - update memory size as in Equation 17
  - update parameter $\theta_i^j \leftarrow \theta_i^{j-1} - \eta_i^{(j)} \cdot \tilde{g}_i^{(j)}$
- **end**

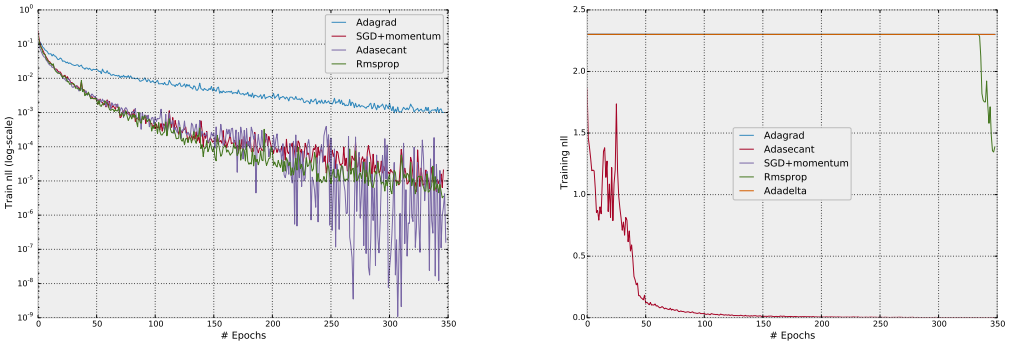
until *stopping criterion is met*;

A.2 Algorithm pseudo-code

Algorithm 1 contains the pseudo-code of the Adasecant algorithm.

A.3 Further Experimental Details

In our experiments with Adasecant algorithm, adaptive momentum term γ_i^k was clipped at 1.8. In 2-layer Maxout network experiments for SGD-momentum experiments, we used the best hyper-parameters reported by [6], for Rmsprop and Adagrad, we crossvalidated learning rate for 15 different learning rates sampled uniformly from the log-space. We crossvalidated 30 different pairs of momentum and learning rate for SGD+momentum, for Rmsprop and Adagrad, we crossvalidated 15 different learning rates sampled them from log-space uniformly for deep maxout experiments. In Figure 2, we analyzed the effect of using different minibatch sizes for Adasecant and compared its convergence with Adadelta in wall-clock time. For minibatch size 100 Adasecant was able to reach the almost same training negative log-likelihood as Adadelta after the same amount of time, but its convergence took much longer. With minibatches of size 500 Adasecant was able to converge faster in wallclock time to a better local minima.



(a) 2 layer Maxout Network

(b) 16 layer Maxout Network

Figure 1: Comparison of different stochastic gradient algorithms on MNIST with Maxout Networks. Both a) and b) are trained with dropout and maximum column norm constraint regularization on the weights. Networks are initialized with weights sampled from a Gaussian distribution with 0 mean and standard deviation of 0.05. In both experiments, the proposed algorithm, Adasecant, seems to be converging faster and arrives to a better minima in training set. We trained both networks for 350 epochs over the training set.

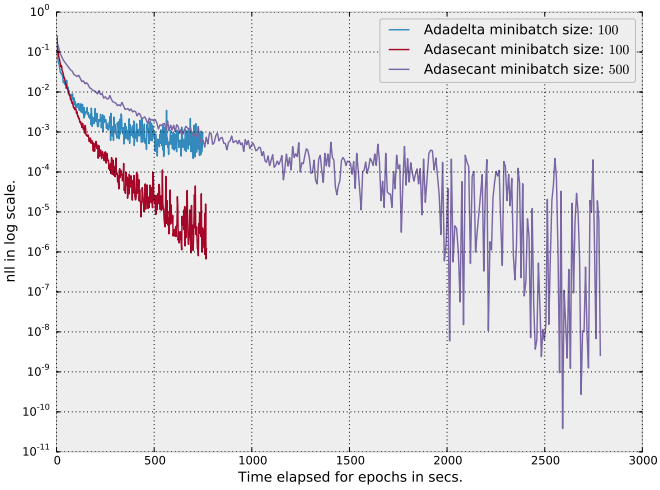


Figure 2: In this plot, we compared adasecant trained by using minibatch size of 100 and 500 with adadelta using minibatches of size 100. We performed these experiments on MNIST with 2-layer maxout MLP using dropout.