Posted online June 15, 2018 *bioRxiv, 2018, **1–5***

Hyperparameter-free optimizer of gradient-descent method incorporating unit correction and moment estimation

Kazunori D Yamada [1,2*]

[1]Graduate School of Information Sciences, Tohoku University, Sendai, Japan [2]Artificial Intelligence Research Center, National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan

ABSTRACT

In the deep learning era, a gradient descent method is the most common method to optimize parameters of neural networks. Among various mathematical optimization methods, a gradient descent method is the most naive method. Although controlling a learning rate of the method is necessary for quick convergence, the gradient descent method normally requires us to adjust it manually. In order to control the learning rate and accelerate converging speed of the method, a lot of optimizers were developed. These optimizers mainly adjust the learning rate automatically depending on state of learning. These optimizers have been improved little by little by incorporating merits of former developed methods. In this study, we developed a novel optimizer, YamAdam. Our optimizer is based on Adam, which utilizes the first and second moment information of previous gradients. In addition to the moment estimation system, we incorporated a merit of AdaDelta, namely a unit correction system, into YamAdam. According to benchmark tests on some popular datasets, our optimizer showed same or faster convergent performance, compared to the existing methods. Our optimizer will be an option of alternative optimizers for deep learning.

INTRODUCTION

There are a lot of iterative mathematical optimization methods such as gradient descent method, Newton method, quasi-Newton method and conjugate gradient method. The Newton method is fast convergent method, which utilizes Newton direction to update parameters. The Newton direction is calculated with gradient information and Hessian of the objective function by

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \boldsymbol{H}_t^{-1} \boldsymbol{g}_t, \tag{1}$$

where t, $\boldsymbol{\theta}_t$, $\boldsymbol{H}_t$ and $\boldsymbol{g}_t$ stands for time, parameter vector, Hessian and gradient vector at time t, respectively. The Newton method converges in quadratic manner and is of faster convergent performance, compared to first-order methods like the gradient descent method. The method is able to reach optimal value within only an iteration, if the objective function is expressed by quadratic equation. However, the Newton method has several disadvantages. For example, the convergence is not ensured, if initial value of parameter is not close to optimal value, Hessian is required to be positive definite matrix and calculation cost of Inverse of Hessian is large. To solve these problem, quasi-Newton method had been developed. The method is what attempts to approximate Hessian with an alternative matrix, rather than directly calculating it. This method is of superlinear convergent property.

Different from the above fancy methods, the gradient descent method is the most naive mathematical optimization method. Normally, the gradient descent is not fast convergent method and it converges in linear manner. On the other hand, since the implementation of gradient descent method is easy and it is of global convergence property, it is frequently used as solver of neural networks including deep learning methods. The update rule of gradient descent is formulate by

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \boldsymbol{g}_t, \tag{2}$$

where α stands for the learning rate (step size), which decides magnitude of parameter update from previous value to current value. To efficiently search an optimal value with the gradient descent method, adjustment of the learning rate is indispensable. With too large learning rate, we cannot reach an optimal value because oscillation of searching near the optimal value will occur. On the other hand, with too small learning rate, we will require tremendous time to reach the optimal value because magnitude of update is small and we needs a number of updates. Since the gradient descent method is frequently used as mathematical optimization method, various improvement strategies have been devised so far. For example, a stochastic gradient descent (SGD) method attempts to avoid falling into a local minimum by stochastically sampling data and repeatedly updating parameters with the sampled data. In recent deep learning, SGD is de facto standard of parameter updates. Further, one of the other strategies for improving the gradient descent or SGD is using an optimizer. So far, a lot of optimizing methods of SGD were developed. Basically, the optimizer attempts to control the learning rate of SGD. The most naive method is a bisection method, where the learning rate is updated by dividing previous value by two when the number of updates reaches predefined value. A Momentum method is

*To whom correspondence should be addressed. Tel: +81 22 795 5856; Email: kyamada@ecei.tohoku.ac.jp

© 2018 The Author(s)
This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/2.0/uk/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

what incorporates momentum term into parameter update. The momentum term literally works as moment and helps update direction toward previously updated direction. Further, this method was modified to incorporate acceleration term [1]. AdaGrad is adaptive method and it collects square of previous gradient information and based on the value, the learning rate is diminished gradually [2].

AdaDelta [3] is more sophisticated method and one of the best optimizers in current, which improved the performance of AdaGrad. The most basic idea of AdaDelta is incorporating a correction of mismatch of unit between update values and original parameters. The match of unit is naturally ensured by the second-order method like Newton method, though vanilla SGD, AdaGrad and other methods basically do not have the system. The update rule of AdaDelta is described by algorithm 1. AdaDelta does not have the learning rate anymore.

Algorithm 1 Update rule of AdaDelta

Require: Objective function $f(\theta)$, Initial parameter θ_0
$\beta = 0.95$, $\epsilon = 10^{-6}$

1: $v_0 = 0$, $s_0 = 0$, $h_0 = 0$
2: **for** $t = 0 : t < T$ **do**
3: $g_t \leftarrow \nabla f(\theta_t)$
4: $v_{t+1} \leftarrow \beta v_t + (1-\beta) g_t^2$
5: $s_{t+1} \leftarrow \beta s_t + (1-\beta) h_t^2$
6: $h_{t+1} \leftarrow \frac{\sqrt{s_{t+1}+\epsilon}}{\sqrt{v_{t+1}+\epsilon}} g_t$
7: $\theta_{t+1} \leftarrow \theta_t - h_{t+1}$
8: **end for**
9: **return** θ_T

Here, ϵ is small value to avoid zero division and β is hyperparameter to control parameter update. In the context of unit correction, the most important variable is s. As shown in line 5, the unit of s corresponds to square root of h^2, namely parameter update, h. Also, the unit of h is same as that of θ. Thus, there is no mismatch of unit between the update and parameters.

On the other hand, Adam [4] is one of the other best optimizers, which also improved the AdaGrad. The update rule of Adam was described in algorithm 2. Here, ϵ is small value to avoid zero division, α stands for the learning rate and β_1 and β_2 are hyperparameter to decide smoothing coefficient of exponential moving average. The fundamental idea of Adam is adjusting the learning rate depending on a progress of learning. In the update rule, m_t in line 4 and v_t in line 5 approximates first moment and second moment, namely mean and variance of gradient from first update to time t. Namely, by dividing mean by variance of conserved gradient information, when the gradient fluctuates on near a local minimum, update size will be smaller to be able to fall into the minimum. On the contrary situation, magnitude of update will be larger, resulting in acceleration of convergence. This idea is similar to that of Newton method, in which if an alteration of gradient, namely Hessian, is larger, the magnitude of update will be smaller. The convergence performance of SGD with Adam is quite fast and Adam is one of the most popular optimizer in current.

As just described, the performance of optimizers have been improved little by little, incorporating features and

Algorithm 2 Update rule of Adam

Require: Objective function $f(\theta)$, Initial parameter θ_0
$\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-6}$

1: $m_0 = 0$, $v_0 = 0$
2: **for** $t = 0 : t < T$ **do**
3: $g_t \leftarrow \nabla f(\theta_t)$
4: $m_{t+1} \leftarrow \beta_1 m_t + (1-\beta_1) g_t$
5: $v_{t+1} \leftarrow \beta_2 v_t + (1-\beta_2) g_t^2$
6: $\hat{m} \leftarrow \frac{m_{t+1}}{1-\beta_1^{t+1}}$
7: $\hat{v} \leftarrow \frac{v_{t+1}}{1-\beta_2^{t+1}}$
8: $h = \alpha \frac{\hat{m}}{\sqrt{\hat{v}-\epsilon}}$
9: $\theta_{t+1} \leftarrow \theta_t - h$
10: **end for**
11: **return** θ_T

merits of previously developed optimizers. In this study, we developed a novel optimizer by incorporating a merit of AdaDelta, the unit correction system, to Adam. In addition, we introduced an adjusting system of a smoothing coefficient of exponential moving average of update information, based on previous update information, resulting on dissipating all hyperparameters.

ALGORITHM

The Algorithm 3 includes update rule of our developed optimizer, YamAdam (Yamada modified Adam). YamAdam does not have any hyperparameters.

Algorithm 3 Update rule of YamAdam

Require: Objective function $f(\theta)$, Initial parameter θ_0
Sigmoid function $\phi(x)$, $\epsilon = 10^{-6}$

1: $m_0 = 0$, $v_0 = 0$, $s_0 = 0$, $h_0 = 0$, $\beta_0 = 0$
2: **for** $t = 0 : t < T$ **do**
3: $g_t \leftarrow \nabla f(\theta_t)$
4: $m_{t+1} \leftarrow \beta_t m_t + (1-\beta_t) g_t$
5: $v_{t+1} \leftarrow \beta_t v_t + (1-\beta_t)(g_t - m_t)^2$
6: $s_{t+1} \leftarrow \beta_t s_t + (1-\beta_t) h_t^2$
7: $h_{t+1} \leftarrow \frac{\sqrt{s_{t+1}+\epsilon}}{\sqrt{v_{t+1}+\epsilon}} m_{t+1}$
8: $\beta_{t+1} \leftarrow \phi\left(\frac{\|h_t\|_1 + \epsilon}{\|h_{t+1}\|_1 + \epsilon}\right) - \epsilon$
9: $\theta_{t+1} \leftarrow \theta_t - h_{t+1}$
10: **end for**
11: **return** θ_T

In the algorithm, the first and second moment information was stored as exponential moving average with smoothing coefficient, β_t in line 4 and 5. These values were estimation of mean and variance of gradient respectively. In line 6, square of update value is stored in similar manner as above. Using these values, a novel update value is calculated by the equation in line 7. Further, smoothing coefficient of the exponential moving average is calculated based on current and

previous update value in line 8. If next update size is smaller, β_t will larger and thus next gradient information will be considered to be less important. Conversely, if next update size is larger, the next gradient information will be regard as more important. Namely, in this system, the smoothing coefficient act as momentum, where the contribution of current gradient information on the accumulated gradient information will change depending on update intensity. Finally, the parameters are updated by the equation in line 9.

SIMULATION EXPERIMENTS

To benchmark the performance of our optimizer, we conducted two simulation experiments, in which the convergent performance of logistic regression and four layer multilayer perceptron (MLP) were tested on various dataset.

Methods

Benchmark datasets As benchmark datasets, we utilized three datasets such as Modified National Institute of Standards and Technology (MNIST) [5], Letter Recognition (LR) and Sensorless Drive Diagnosis (SDD) dataset [6]. The number of attributes of each instance on the datasets is 784, 16 and 48, the dimension of output vector is 10, 26 and 11 and the number of instances is 70,000, 20,000 and 58,509 respectively. From the datasets, first 60,000, 15,000 and 40,000 instances were used as training dataset and remaining 10,000, 5,000 and 18,509 instances were used as validation dataset respectively.

Network architechture As a simulation experiment, we designed a logistic regression model. In addition, we designed four layer MLP, consisting of an input layer, two middle layers and an output layer. The number of unit of the middle layers was set to 500 respectively. For both architecture, the output vector was solution of softmax function and these models were for multi-class classification. We used ReLU as an activation function of middle layers and for that, the suitable optimized initial parameters [7] were used. As a framework to implement the architecture, we utilized Theano version 0.8.2 (University of Montreal) with Python version 3.5.2.

Prameters of compared optimizers For vanilla SGD and AdaGrad, the learning rate α was set to 0.01. For AdaDelta, the smoothing coefficient of exponential moving average, β was set to 0.95. For Adam, the learning rate α, the smoothing coefficient of the exponential moving average β_1 and β_2 was set to 0.001, 0.9 and 0.999 respectively.

Benchmark procedure For each method, training loss from 1 to 50 epoch was recorded repeatedly 10 times changing initial parameters at random. The mean training loss value of the 10 trials was plotted on a scatterplot with line representation.

Computational environment The benchmark was conducted on an Intel(R) Xeon(R) CPU E5-2680 v2 @2.80 GHz with 64GB RAM with Tesla K20m (NVIDIA) at the NIG supercomputer at ROIS National Institute of Genetics in Japan.

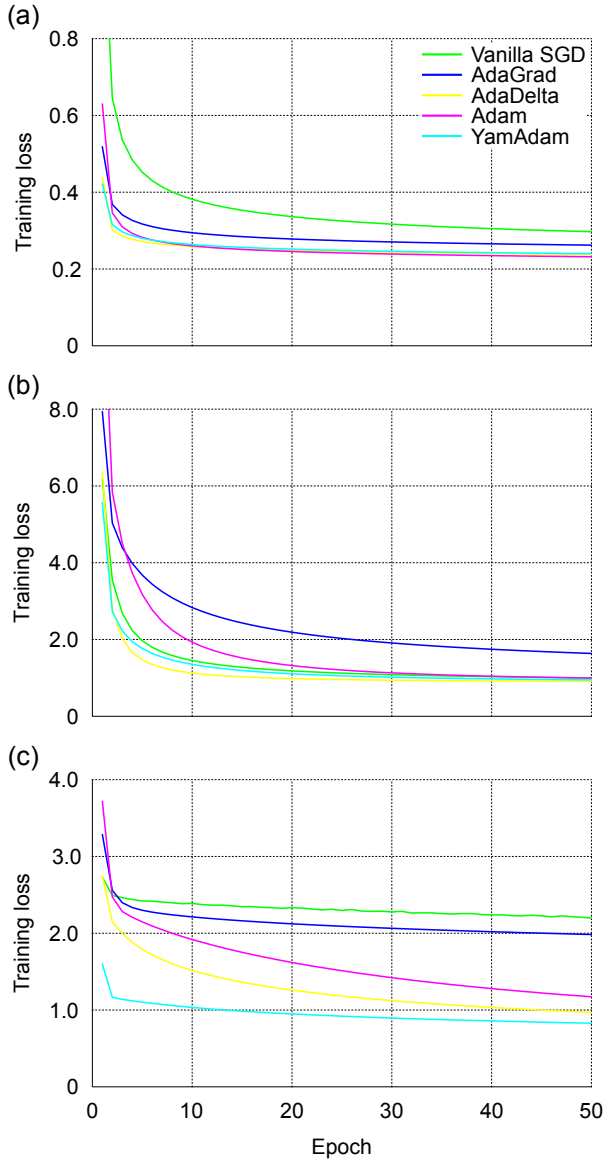

Figure 1. Comparison of optimizers on logistic regression. The performance of each optimizer was benchmarked on (a) MNIST, (b) LR and (c) SDD dataset.

Results and discussion

Logistic regression As shown in Figure 1, we conducted a performance benchmark of five methods including vanilla SGD (no optimizer), AdaGrad, AdaDelta, Adam and YamAdam on MNIST (a), LR (b) and SDD (c) dataset with logistic regression method. In the results on all dataset, Adam and AdaDelta had relatively rapid convergence performance compared to the other existing methods. These results were reasonable based on the previous reports [3, 4]. On the other hand, YamAdam has same or better convergent performance than Adam and same or a little better performance than AdaDelta. The difference of the performances was clear when

the methods was compared on SDD. As to the results on SDD, the best and second best method was YamAdam and AdaDelta. Since YamAdam incorporates the unit correction system stemmed from AdaDelta, this feature may have positive effects on the performance on SDD. One of possible factors for the advantage of unit correction system may related to difficulty of problem. MNIST is relatively easy problem, compared to the other two problem. Actually, accuracy of each method in the study was higher than about 0.9, even though the architecture was the normal logistic regression. On the other hand, the accuracy of the methods on SDD dataset was almost 0.30 – 0.75 and therefore the dataset was thought to be more difficult problem than MNIST. Although there would be lots of factors which decide difficulty of problems, a mismatch of unit of each attribute on instances of the dataset will be one of possible factors. For example, a value of first and 38th element in the first instance of SDD dataset is -6.4707×10^{-6} and 40.409. This mismatch of magnitude of values, namely mismatch of unit is the most different feature of SDD compared the other dataset. This property may be related to the higher performance of YamAdam on SDD. Actually, if we benchmark the methods on standardized SDD dataset, the difference of performances diminished.

MLP with four layers As shown in Figure 2, we conducted similar experiments with MLP consisting of four layers. As a result, there were no qualitative differences with the results on the logistic regression as to the performance of the methods, even though more layers were stacked and the parameters were increased in the architecture.

CONCLUSION

We developed a novel optimizer, YamAdam. YamAdam is hyperparameter-free optimizer and includes features of unit correction system and moment estimation, derived from AdaDelta and Adam respectively. Our method had same or faster convergent performance compared to the existing optimizers, especially on computation on unit-mismatched dataset. There are various dataset all over the world, our optimizer will be an option of alternative optimizers for deep learning.

ADDITIONAL INFORMATION

Acknowledgements

Computations were partially performed on the NIG supercomputer at ROIS National Institute of Genetics.

Funding

This work was supported in part by the Top Global University Project from the Ministry of Education, Culture, Sports, Science, and Technology of Japan (MEXT).

Availability of data and material

The source code of YamAdam are available at GitHub, git@github.com:yamada-kd/YamAdam.git.

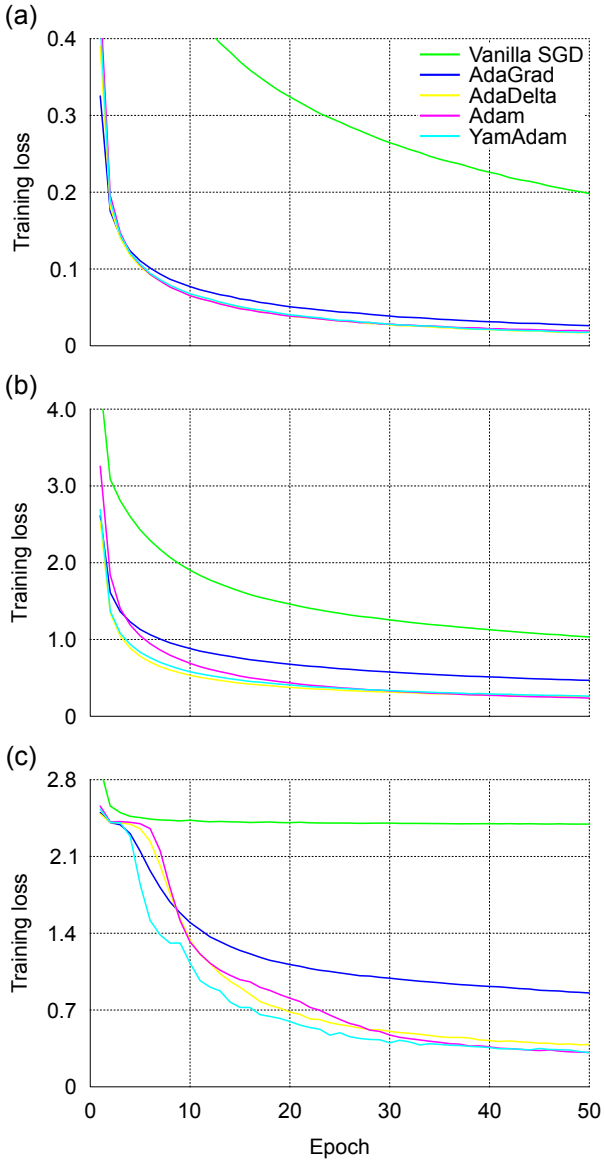

Figure 2. Comparison of optimizers on MLP. The performance of each optimizer was benchmarked on (a) MNIST, (b) LR and (c) SDD dataset.

Abbreviations

LR: Letter Recognition; MLP: multilayer perceptron; MNIST: Modified National Institute of Standards and Technology; ReLU: rectified linear unit; SDD: Sensorless Drive Diagnosis; SGD: stochastic gradient descent

Competing interests

The authors declare that they have no competing interests.

REFERENCES

1. Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady AN USSR*, volume 269,

pages 543–547, 1983.
2. John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.
3. Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.
4. Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
5. Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.
6. Dheeru Dua and Efi Karra Taniskidou. Uci machine learning repository. 2017.
7. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.