CHAPTER 8-2. ADAPTIVE LEARNING ALGORITHMS

Yutong He Kun Yuan

November 14, 2023

1 Introduction

In this chapter, we consider adaptive optimization algorithms that are useful in deep learning scenarios. The way adaptive algorithms differ from non-adaptive algorithms (e.g. SGD, momentum SGD, etc.) lies in the fact that adaptive methods can make use of information gained through the optimization procedure, while non-adaptive ones can not. Through this chapter, we have the following notations:

Notation.

- Let $f : \mathbb{R}^d \to \mathbb{R}$ be the objective function to minimize.
- Let $f^\star := \inf_{x\in\mathbb{R}^d} f(x)$ be the optimal value of f.
- Let $\hat{\nabla} f$ denote the gradient oracle that estimates the gradient of f at given query points.
- Let $[N]$ denote the set $\{1, 2, \cdots, N\}$ for any positive integer N.
- Let $[x]_k$ denote the k-th entry of vector x.
- Let $\odot$ denote entry-wise multiplication for vectors, i.e. for vectors $x, y \in \mathbb{R}^d$, $[x \odot y]_k = [x]_k[y]_k$, $\forall k \in [d]$.
- For vector $x \in \mathbb{R}^d$ and scalar $a \in \mathbb{R}$, let $x^{\odot a}$ denote the entry-wise power of x, i.e., $[x^{\odot a}]_k = [x]_k^a$, $\forall k \in [d]$.
- By adding a vector $x \in \mathbb{R}^d$ with a scalar $a \in \mathbb{R}$, we mean the entry-wise addition, i.e., $[x + a]_k = [x]_k + a$, $\forall k \in [d]$.

- By dividing vector $x \in \mathbb{R}^d$ with vector $y \in \mathbb{R}^d$, we mean the entry-wise division, i.e., $\frac{x}{y} = x \odot y^{\odot(-1)}$.

2 Adaptive algorithms

2.1 AdaGrad

AdaGrad (Adaptive Gradient) algorithm, introduced by [1], is motivated to balance different entries in the gradients, works well with sparse gradients.

Motivation. Viewing each entry as a feature, the author aimed to normalize the gradients according to the occurring frequencies, i.e. lower learning rate for frequently occurred features, and higher learning rate for infrequent ones, with the intuition to tell the learner to "take notice" whenever an infrequent feature is seen.

Iteration. The AdaGrad algorithm iterates as follows:

$$g_k = \hat{\nabla} f(x_k), \tag{1}$$

$$G_k = G_{k-1} + g_k g_k^\top, \tag{2}$$

$$x_{k+1} = x_k - \gamma G_k^{-1/2} g_k, \tag{3}$$

where G_k initialized by $G_{-1} = 0$ tracks the sum of outer products of sampled gradients, and γ is the global learning rate.

A more practical version of AdaGrad derives from diagonalizing the above general formation, which is given by

$$g_k = \hat{\nabla} f(x_k), \tag{4}$$

$$v_k = v_{k-1} + g_k^{\odot 2}, \tag{5}$$

$$x_{k+1} = x_k - \gamma (v_k + \epsilon)^{\odot(-1/2)} \odot g_k, \tag{6}$$

where v_k initialized by $v_{-1} = 0$ tracks the diagonal of G_k, and $\epsilon > 0$ is a small constant that protect the algorithm from zero-division.

Drawback. The gradients become too small in later training stages, since the second-order information is accumulated since the very beginning iterations. The adaptive learning rate could approach zero before the algorithm arrives at good local minima. Also, AdaGrad is sensitive to the global learning rate γ.

Remark. We refer the readers to `https://pytorch.org/docs/stable/optim.html` for the pytorch implementation where learning rate decay and weight decay are added to AdaGrad.

2.2 AdaDelta

AdaDelta algorithm, introduced by [2] aims at fixing the drawbacks of AdaGrad.

Motivation. AdaGrad algoirthm's adaptive learning rate vanish too fast because of the accumulated gradient squares. AdaGrad also has a global learning rate γ to tune. AdaDelta uses the idea of moving average to reduce the influence of large gradient in the beginning steps, i.e., use $G_k = \rho G_{k-1} + (1-\rho) g_k g_k^\top$ instead of (2). In addition, the author analyzed and corrected the units with Hessian approximation. Intuitively, assuming the correctness of Newton method, the weight update satisfies $\Delta x \propto \frac{1}{\partial^2 f/\partial x^2} \cdot \frac{\partial f}{\partial x}$, indicating the use of a preconditioning matrix of unit $\frac{1}{\partial^2 f/\partial x^2} \propto \frac{\Delta x}{\partial f/\partial x}$.

Iteration. The AdaDelta algorithm iterates as follows:

$$
\begin{aligned}
g_k =& \hat{\nabla} f(x_k), && (7)\\
v_k =& \rho v_{k-1} + (1-\rho) g_k^{\odot 2}, && (8)\\
\Delta_k =& -\frac{(u_{k-1}+\epsilon)^{\odot(1/2)}}{(v_k+\epsilon)^{\odot(1/2)}} \odot g_k, && (9)\\
u_k =& \rho u_{k-1} + (1-\rho) \Delta_k^{\odot 2}, && (10)\\
x_{k+1} =& x_k + \Delta_k, && (11)
\end{aligned}
$$

where $\rho > 0$ is a constant which are set to 0.95/0.90 by default in Tensorflow/Pytorch implementation, and ϵ is a positive safeguard constant, v_k and u_k are initialized with zero.

Remark. We refer the readers to the Pytorch documentation for implementation details, where weight decay and global learning rate are considered additionally.

2.3 RMSProp

RMSProp is an unpublished method proposed by Geoff Hinton in Lecture 6e of his Coursera Class [3], which works well in on-line and non-stationary settings.

Motivation. The RMSProp algorithm is in fact a parallel work of AdaDelta, both aiming at solving the drawback of AdaGrad. RMSProp is indeed AdaDelta without the use of Δ_k's second order information.

Iteration. The RMSProp algorithm iterates as follows:

$$
\begin{aligned}
g_k =& \hat{\nabla} f(x_k), && (12)\\
v_k =& \rho v_{k-1} + (1-\rho) g_k^{\odot 2}, && (13)\\
x_{k+1} =& x_k - \frac{\gamma}{(v_k+\epsilon)^{\odot(1/2)}} \odot g_k, && (14)
\end{aligned}
$$

where $\rho > 0$ is a constant decaying factor, $\epsilon > 0$ is a safeguard constant, v_k is initialized with $v_{-1} = 0$, and $\gamma > 0$ is the global learning rate.

Remark. We refer the readers to the Pytorch documentation for implementation details, where weight decay, centered second order version, and momentum are considered additionally.

2.4 Adam

Adam (Adaptive Moment Estimation), introduced in [4], additionally stores an exponentially decaying average of past gradients.

Motivation. Adam combines the momentum acceleration mechanism with RMSProp. The momentum is updated as an exponentially decaying average of the past gradients. When computing the updates, Adam also corrected the momentum, as well as the second order momentum average to reduce the bias in the beginning stages (where historical information is identical to zero).

Iteration. The Adam algorithm iterates as follows:

$$g_k = \hat{\nabla} f(x_k), \tag{15}$$

$$m_k = \rho_1 m_{k-1} + (1-\rho_1) g_k, \tag{16}$$

$$v_k = \rho_2 v_{k-1} + (1-\rho_2) g_k^{\odot 2}, \tag{17}$$

$$\hat{m}_k = \frac{m_k}{1-\rho_1^k}, \tag{18}$$

$$\hat{v}_k = \frac{v_k}{1-\rho_2^k}, \tag{19}$$

$$x_{k+1} = x_k - \frac{\gamma}{\hat{v}_k^{\odot(1/2)} + \epsilon} \odot \hat{m}_k, \tag{20}$$

where m_k, v_k are moving average initialized with zero, ρ_1, ρ_2 are decaying factors for the first and second order momentum, respectively, and $\gamma > 0$ is the global learning rate which approximately bound the adaptive learning rates, and $\epsilon > 0$ is a safeguard constant.

Remark. We refer the readers to [4] for variant AdaMax, which uses $v_k = \max\{\rho_2 v_{k-1}, |g_k|\}$ instead to track the ℓ_∞ norm, and use it directly in computing the updates without the correction step. We also refer the readers to the Pytorch documentation for detailed implementation of Adam, AdaMax with weight decay as well as amsgrad variant.

2.5 AdamW

AdamW, introduced by [5], aims at fixing the weight decay module which is often used when training neural networks with Adam.

Motivation. Adam with weight decay had been previously implemented as ℓ_2 regularization. Although in SGD, it's easy to show that ℓ_2 regularization is equivalent to weight decay, it is not the case for adaptive learning rate methods like Adam. By decoupling the weight decay module, it can work well just as in an SGD algorithm.

Iteration. The AdamW algorithm iterates as follows:

$$
\begin{aligned}
g_k =& \hat{\nabla} f(x_k), && (21)\\
m_k =& \rho_1 m_{k-1} + (1-\rho_1) g_k, && (22)\\
v_k =& \rho_2 v_{k-1} + (1-\rho_2) g_k^{\odot 2}, && (23)\\
\hat{m}_k =& \frac{m_k}{1-\rho_1^k}, && (24)\\
\hat{v}_k =& \frac{v_k}{1-\rho_2^k}, && (25)\\
x_{k+1} =& x_k - \eta_k \left(\frac{\gamma}{\hat{v}_k^{\odot(1/2)}} \odot \hat{m}_k + \lambda x_k \right), && (26)
\end{aligned}
$$

where m_k, v_k are arrays starting from zero, γ is the global learning rate, η_k is the additional learning rate scheduler, λ is the weight decay strength.

Remark. The author of AdamW empirically suggested that the optimal weight decay strength λ should be weaker/stronger for longer/shorter runs, and suggested the use of step-drop learning rate decay or cosine annealing for the scheduler η_k's. Pytorch documentation provides a default implementation of AdamW without the additional learning rate scheduler, which differs from AdamW in the sense that it queries the gradient prior to the addition of weight decay.

2.6 More algorithms

Researchers have designed lots of adaptive gradient descent algorithms for deep learning tasks. Besides what we've mentioned, we refer the readers to the following works: Adafactor[6], CAME[7], and Shampoo[8].

3 Convergence analysis

Following [9], we list convergence analysis results for a family of Adam-style adaptive algorithms, which iterates as follows:

$$
\begin{aligned}
g_k =& \hat{\nabla} f(x_k), && (27)\\
m_k =& \rho_1 m_{k-1} + (1-\rho_1) g_k, && (28)\\
v_k =& h_k(g_0, \cdots, g_k), && (29)\\
x_{k+1} =& x_k - \frac{\gamma}{v_k^{\odot(1/2)} + G_0} m_k, && (30)
\end{aligned}
$$

where h_k outputs v_k using history gradient estimations, $G_0 > 0$ is a constant for safeguarding, and $\gamma > 0$ is the learning rate. The momentum factor $\rho_1 \in (0, 1)$.

Assumption 3.1. Assume f is L-smooth, and its gradient oracle $\hat{\nabla} f$ satisfies:

- $\mathbb{E}[\hat{\nabla} f(x)] = \nabla f(x),\ \forall x \in \mathbb{R}^d$.
- $\mathbb{E}[\|\hat{\nabla} f(x) - \nabla f(x)\|_2^2] \le \sigma^2(1 + c\|\nabla f(x)\|_2^2),\ \forall x \in \mathbb{R}^d$, for some $c > 0$.

Remark. Assumption 3.1 is weaker than the assumptions on gradient oracles we have seen in the previous chapters. It's worth noting that the randomness in the gradient oracles are also assumed to be independent.

Assumption 3.2. Assume in the iterations of the Adam-style algorithms (27)- (30), $s_k := 1/(v_k^{\odot(1/2)} + G_0)$ is lower bounded and upper bounded, *i.e.*, there exists $0 < c_l < c_u$ such that $c_l \le \|s_k\|_\infty \le c_u$.

Remark. Assumption 3.2 is a strong assumption, yet it can be satisfied in algorithms like Adam under a more traditional assumption: $\|\hat{\nabla} f(x)\|_\infty \le G$. In the theoretical analysis below, one can discover that the exact formulation of (29) does not matter, and that the convergence property only needs the final output v_k to satisfy Assumption 3.2.

Notation.

- Let $\mathcal{F}_k := \{x_k, g_{k-1}, m_{k-1}, v_{k-1}, x_{k-2}, \cdots, g_0, m_0, v_0, x_0\}$ denote the filtration containing all historical variables at and before computing x_k.
- Let $\mathbb{E}_k$ denote the expectation conditioned on $\mathcal{F}_k$, *i.e.*, $\mathbb{E}_k[\cdot] = \mathbb{E}[\cdot \mid \mathcal{F}_k]$.
- Let $\tilde{\gamma}_k := \gamma/(v_k^{\odot 1/2} + G_0)$.

Lemma 3.3. Following the momentum update rule (27)(28) and under Assumption 3.1, it holds that

$$\begin{aligned}\mathbb{E}_k[\|m_k - \nabla f(x_k)\|_2^2] \le& \rho_1\|m_{k-1} - \nabla f(x_{k-1})\|_2^2 + 2(1-\rho_1)^2\mathbb{E}_k[\|g_k - \nabla f(x_k)\|_2^2] \\ &+ \frac{L^2\|x_k - x_{k-1}\|_2^2}{1-\rho_1}. \end{aligned} \tag{31}$$

Proof. According to the recursion, we have

$$\begin{aligned} m_k - \nabla f(x_k) =& \rho_1 m_{k-1} + (1-\rho_1)g_k - \nabla f(x_k) \\ =& \rho_1(m_{k-1} - \nabla f(x_{k-1})) + (1-\rho_1)(g_k - \nabla f(x_k)) - \rho_1(\nabla f(x_k) - \nabla f(x_{k-1})), \end{aligned}$$

which is equivalent to

$$m_k - \nabla f(x_k) + \rho_1(\nabla f(x_k) - \nabla f(x_{k-1})) = \rho_1(m_{k-1} - \nabla f(x_{k-1})) + (1-\rho_1)(g_k - \nabla f(x_k)). \tag{32}$$

Using the unbiasedness of g_k, (32) indicates that

$$\begin{aligned}&\mathbb{E}_k[\|m_k - \nabla f(x_k) + \rho_1(\nabla f(x_k) - \nabla f(x_{k-1}))\|_2^2]\\ \leq&\rho_1^2\|m_{k-1} - \nabla f(x_{k-1})\|_2^2 + (1-\rho_1)^2\mathbb{E}_k[\|g_k - \nabla f(x_k)\|_2^2].\end{aligned} \tag{33}$$

Consequently, by applying L-smoothness, (33) indicates

$$\begin{aligned}&\mathbb{E}_k[\|m_k - \nabla f(x_k)\|_2^2]\\ \leq&(1+1-\rho_1)\mathbb{E}_k[\|m_k - \nabla f(x_k) + \rho_1(\nabla f(x_k) - \nabla f(x_{k-1}))\|_2^2]\\ &+\left(1+\frac{1}{1-\rho_1}\right)\mathbb{E}_k[\|\rho_1(\nabla f(x_k) - \nabla f(x_{k-1}))\|_2^2]\\ \leq&\rho_1\|m_{k-1} - \nabla f(x_{k-1})\|_2^2 + 2(1-\rho_1)^2\mathbb{E}_k[\|g_k - \nabla f(x_k)\|_2^2] + \frac{L^2\|x_k - x_{k-1}\|_2^2}{1-\rho_1},\end{aligned}$$

which justifies (31). □

Lemma 3.4. Under Assumption 3.1 and 3.2, if $\gamma L \leq c_l/(2c_u^2)$, we have

$$f(x_{k+1}) \leq f(x_k) + \frac{\gamma c_u}{2}\|\nabla f(x_k) - m_k\|_2^2 - \frac{\gamma c_l}{2}\|\nabla f(x_k)\|_2^2 - \frac{\gamma c_l}{4}\|m_k\|_2^2. \tag{34}$$

Proof. By L-smoothness, we have

$$\begin{aligned}f(x_{k+1}) \leq&f(x_k) + \langle\nabla f(x_k), x_{k+1} - x_k\rangle + \frac{L}{2}\|x_{k+1} - x_k\|_2^2,\\ =&f(x_k) - \langle\nabla f(x_k), \tilde{\gamma}_k \odot m_k\rangle + \frac{L}{2}\|\tilde{\gamma}_k \odot m_k\|_2^2,\\ \leq&f(x_k) + \frac{1}{2}\|\sqrt{\tilde{\gamma}_k} \odot (\nabla f(x_k) - m_k)\|_2^2 - \frac{1}{2}\|\sqrt{\tilde{\gamma}_k} \odot \nabla f(x_k)\|_2^2\\ &+\left(\frac{L}{2}\|\tilde{\gamma}_k \odot m_k\|_2^2 - \frac{1}{2}\|\sqrt{\tilde{\gamma}_k} \odot m_k\|_2^2\right)\\ \leq&f(x_k) + \frac{\gamma c_u}{2}\|\nabla f(x_k) - m_k\|_2^2 - \frac{\gamma c_l}{2}\|\nabla f(x_k)\|_2^2 + \frac{\gamma^2 c_u^2 L - \gamma c_l}{2}\|m_k\|_2^2\\ \leq&f(x_k) + \frac{\gamma c_u}{2}\|\nabla f(x_k) - m_k\|_2^2 - \frac{\gamma c_l}{2}\|\nabla f(x_k)\|_2^2 - \frac{\gamma c_l}{4}\|m_k\|_2^2.\end{aligned}$$

□

Theorem 3.5. Let $\mathcal{M}_k = \|m_k - \nabla f(x_k)\|_2^2$ and $f(x_0) - f^\star \leq \Delta_f$, under assumption 3.1 and 3.2, if we let $\rho_1 \in [1 - \frac{c_l}{(12/\epsilon+8c)\sigma^2 c_u}, 1)$, $\gamma \leq \min\left\{\frac{1-\rho_1}{2Lc_u^{3/2}}, \frac{1}{\sqrt{2}Lc_u}, \frac{c_l}{2c_u^2 L}\right\}$, $K \geq \max\left\{\frac{6\mathcal{M}_0 c_u}{(1-\rho_1)\epsilon c_l}, \frac{12\Delta_f}{\gamma\epsilon c_l}\right\}$, it holds that

$$\mathbb{E}\left[\frac{1}{K+1}\sum_{k=0}^{K}\|\nabla f(x_k)\|_2^2\right] \leq \epsilon. \tag{35}$$

Proof. By Lemma 3.3 and Assumption 3.1 we have

$$\mathbb{E}[\mathcal{M}_{k-1}] \leq \frac{1}{1-\rho_1}\mathbb{E}[\mathcal{M}_{k-1} - \mathcal{M}_k] + 2(1-\rho_1)\sigma^2(1 + c\|\nabla f(x_k)\|_2^2) + \frac{L^2\|x_k - x_{k-1}\|_2^2}{(1-\rho_1)^2}. \tag{36}$$

Summing (36) from 1 to $K+1$ yields

$$\begin{aligned}
&\mathbb{E}\left[\sum_{k=0}^{K}\mathcal{M}_k\right] \\
\leq&\mathbb{E}\left[\sum_{k=0}^{K}\frac{\mathcal{M}_k - \mathcal{M}_{k+1}}{1-\rho_1} + 2(1-\rho_1)\sigma^2(K+1) + 2(1-\rho_1)\sigma^2 c\sum_{k=1}^{K+1}\|\nabla f(x_k)\|_2^2\right. \\
&\left. + \sum_{k=0}^{K}\frac{L^2\gamma^2 c_u^2\|m_k\|_2^2}{(1-\rho_1)^2}\right].
\end{aligned} \tag{37}$$

Applying Lemma 3.4 from 0 to K, we have

$$\frac{\gamma c_l}{2}\mathbb{E}\left[\sum_{k=0}^{K}\|\nabla f(x_k)\|_2^2\right] \leq f(x_0) - f(x_{K+1}) + \frac{\gamma c_u}{2}\mathbb{E}\left[\sum_{k=0}^{K}\mathcal{M}_k\right] - \frac{\gamma c_l}{4}\mathbb{E}\left[\sum_{k=0}^{K}\|m_k\|_2^2\right]. \tag{38}$$

Applying (37) to(38), we obtain

$$\begin{aligned}
&\frac{\gamma c_l}{2}\mathbb{E}\left[\sum_{k=0}^{K}\|\nabla f(x_k)\|_2^2\right] \\
\leq&\Delta_f - \frac{\gamma c_l}{4}\mathbb{E}\left[\sum_{k=0}^{K}\|m_k\|_2^2\right] + \frac{\gamma c_u}{2}\mathbb{E}\left[\sum_{k=0}^{K}\frac{\mathcal{M}_k - \mathcal{M}_{k+1}}{1-\rho_1} + 2(1-\rho_1)\sigma^2(K+1)\right. \\
&\left. + 2(1-\rho_1)\sigma^2 c\sum_{k=1}^{K+1}\|\nabla f(x_k)\|_2^2 + \sum_{k=0}^{K}\frac{L^2\gamma^2 c_u^2\|m_k\|_2^2}{(1-\rho_1)^2}\right] \\
\leq&\Delta_f - \frac{\gamma c_l}{4}\mathbb{E}\left[\sum_{k=0}^{K}\|m_k\|_2^2\right] + \frac{\gamma c_u}{2}\mathbb{E}\left[\sum_{k=0}^{K}\frac{\mathcal{M}_k - \mathcal{M}_{k+1}}{1-\rho_1} + 2(1-\rho_1)\sigma^2(K+1)\right. \\
&\left. + 4(1-\rho_1)\sigma^2 c\sum_{k=0}^{K}(\|\nabla f(x_k)\|_2^2 + L^2\gamma^2 c_u^2\|m_k\|_2^2) + \sum_{k=0}^{K}\frac{L^2\gamma^2 c_u^2\|m_k\|_2^2}{(1-\rho_1)^2}\right].
\end{aligned} \tag{39}$$

Let $L^2\gamma^2 c_u^3/(2(1-\rho_1)^2) \leq c_l/8$ $\left(i.e.,\ \gamma \leq \frac{(1-\rho_1)\sqrt{c_l}}{2Lc_u^{3/2}}\right)$, $2(1-\rho_1)\sigma^2 c \leq c_l/(4c_u)$ $\left(i.e.,\ \rho_1 \geq 1 - \frac{c_l}{8\sigma^2 c c_u}\right)$, $2(1-\rho_1)\sigma^2 c L^2\gamma^2 c_u^3 \leq c_l/8$ $\left(i.e.,\ \gamma L \leq \frac{1}{\sqrt{2}c_u}\right)$, (39) indicates

$$\begin{aligned}
&\frac{1}{K+1}\mathbb{E}\left[\sum_{k=0}^{K}\|\nabla f(x_k)\|_2^2\right] \\
\leq&\frac{\mathcal{M}_0 c_u}{(1-\rho_1)Kc_l} + \frac{2\Delta_f}{\gamma c_l K} + 2(1-\rho_1)\sigma^2\frac{c_u}{c_l} + \frac{1}{2(K+1)}\mathbb{E}\left[\sum_{k=0}^{K}\|\nabla f(x_k)\|_2^2\right],
\end{aligned}$$

which implies

$$\frac{1}{K+1}\mathbb{E}\left[\sum_{k=0}^{K}\|\nabla f(x_k)\|_2^2\right] \leq \frac{2\mathcal{M}_0 c_u}{(1-\rho_1)Kc_l} + \frac{4\Delta_f}{\gamma c_l K} + 4(1-\rho_1)\sigma^2\frac{c_u}{c_l}. \tag{40}$$

Following (40) with $1-\rho_1 \leq \frac{\epsilon c_l}{12\sigma^2 c_u}$ and $K \geq \max\left\{\frac{6\mathcal{M}_0 c_u}{1-\rho_1 \epsilon c_l}, \frac{12\Delta_f}{\gamma \epsilon c_l}\right\}$, we conclude the proof. □

Remark. In addition, [9] also provides convergence results for the case where $1-\rho_1$ diminishes to zero as the iteration step k increases.

4 Experiments

In this section, we selected some interesting experimental results done by the optimizer designers, which hopefully provide some intuition on the performance of different optimizers. Fig. 1 compared Adam optimizer with momentum SGD, AdaGrad, AdaDelta and RMSProp, details of which can be found in [4]. Fig. 2 compared AdamW with Adam (with ℓ_2 regularization), details of which can be found in [5].

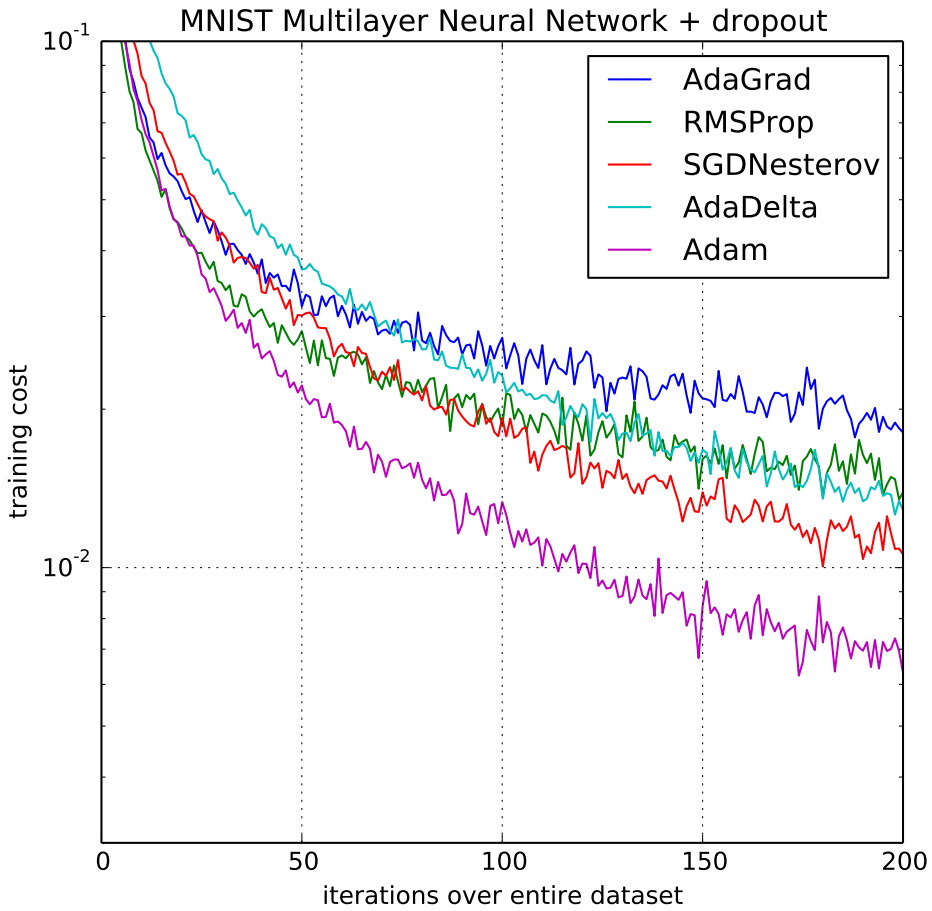

Figure 1: Adam v.s. momentum SGD/AdaGrad/AdaDelta/RMSProp

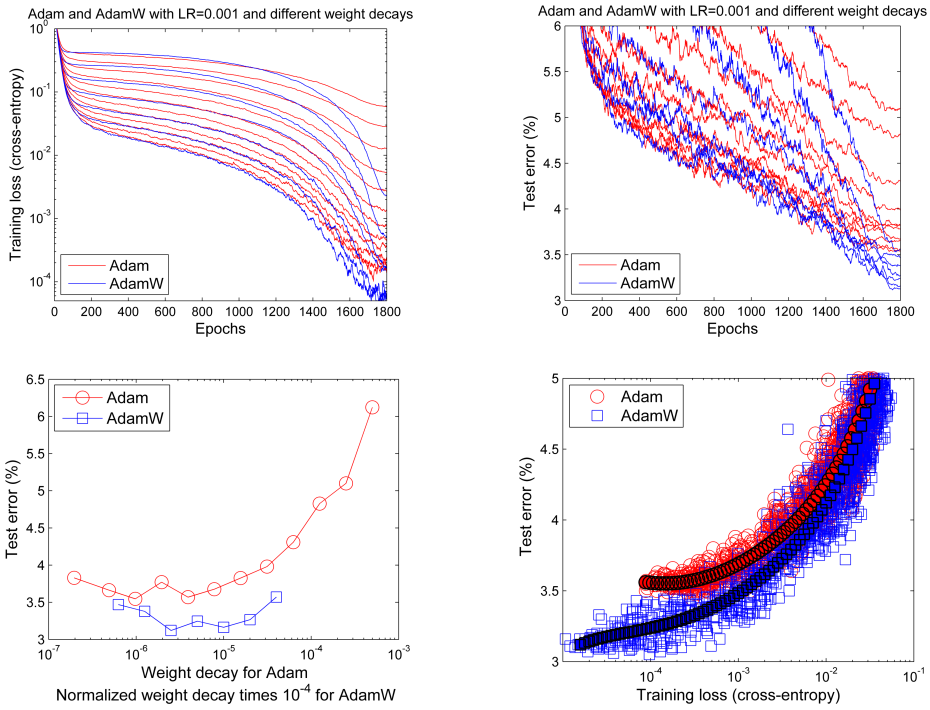

Figure 2: AdamW v.s. Adam

References

[1] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

[2] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[3] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, vol. 14, no. 8, p. 2, 2012.

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[5] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[6] N. Shazeer and M. Stern, "Adafactor: Adaptive learning rates with sublinear memory cost," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4596–4604.

[7] Y. Luo, X. Ren, Z. Zheng, Z. Jiang, X. Jiang, and Y. You, "Came: Confidence-guided adaptive memory efficient optimization," *arXiv preprint arXiv:2307.02047*, 2023.

[8] V. Gupta, T. Koren, and Y. Singer, "Shampoo: Preconditioned stochastic tensor optimization," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1842–1850.

[9] Z. Guo, Y. Xu, W. Yin, R. Jin, and T. Yang, "A novel convergence analysis for algorithms of the adam family," *arXiv preprint arXiv:2112.03459*, 2021.