

Optimizers in Machine Learning

Jianwei Zhang

ZJU CS

June 24, 2019



浙江大學
ZHEJIANG UNIVERSITY

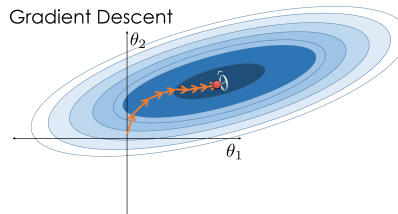
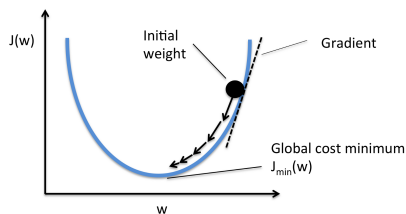
1. Gradient Descent
2. Stochastic Gradient Descent
 - SGD
 - Average SGD
 - Batch SGD
 - Challenges
3. Momentum and Nesterov
 - Exponentially Weighted Average
 - Momentum SGD
 - Nesterov SGD
4. Adaptive Learning Rate
 - AdaGrad
 - AdaDelta
 - RMSProp
5. Adaptive Momentum
 - Adam
 - AdaMax
 - NAdam

1. Gradient Descent
2. Stochastic Gradient Descent
 - SGD
 - Average SGD
 - Batch SGD
 - Challenges
3. Momentum and Nesterov
 - Exponentially Weighted Average
 - Momentum SGD
 - Nesterov SGD
4. Adaptive Learning Rate
 - AdaGrad
 - AdaDelta
 - RMSProp
5. Adaptive Momentum
 - Adam
 - AdaMax
 - NAdam

- Find a local minimizer of a function $f(x)$

Theorem (First-Order Necessary Conditions)

If x^* is a local minimizer and f is continuously differentiable in an open neighborhood of x^* , then $\nabla f(x^*) = 0$.



Algorithm 1 Gradient Descent

Require: Startint point \mathbf{x}_0

Require: Step length γ_t

- 1: Compute negative gradient direction

$$\mathbf{p}_t = -\nabla f(\mathbf{x}_t)$$

- 2: Move one step

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \gamma_t \mathbf{p}_t$$

- 3: Loop line 1 and 2 until $\mathbf{p}_t \approx \mathbf{0}$

- 4: **return** \mathbf{x}_t
-

Machine Learning Model

- Model parameter θ
- Dataset $\{\mathbf{z}_i | i = 1, 2, \dots, n\}$
- Empirical error $f(\mathbf{z}_i | \theta)$, and f is differentiable
- Goal

$$\theta^* = \min_{\theta} \frac{1}{n} \sum_{i=1}^n f(\mathbf{z}_i | \theta)$$

Algorithm 2 Gradient Descent in ML

Require: Initial parameters θ

Require: Learning rate γ_t

- 1: Compute negative gradient direction

$$\mathbf{p}_t = -\frac{1}{n} \sum_{i=1}^n \nabla_{\theta} f(\mathbf{z}_i | \theta) \quad (1)$$

- 2: Update parameters

$$\theta \leftarrow \theta + \gamma_t \mathbf{p}_t$$

- 3: Loop line 1 and 2 until $\mathbf{p}_t \approx \mathbf{0}$

- 4: **return** θ
-

1. Gradient Descent
2. Stochastic Gradient Descent
 - SGD
 - Average SGD
 - Batch SGD
 - Challenges
3. Momentum and Nesterov
 - Exponentially Weighted Average
 - Momentum SGD
 - Nesterov SGD
4. Adaptive Learning Rate
 - AdaGrad
 - AdaDelta
 - RMSProp
5. Adaptive Momentum
 - Adam
 - AdaMax
 - NAdam

- It is hard to compute gradients of the empirical error in (1) when we have a large-scale dataset. So we have **stochastic gradient descent**:
 - Estimate empirical error $E_n(f_\theta)$ by a single randomly picked example \mathbf{z}_t at step t at each iteration:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma_t \nabla_{\boldsymbol{\theta}} f(\mathbf{z}_t | \boldsymbol{\theta}) \quad (2)$$

- The stochastic process $\{\boldsymbol{\theta}_t, t = 1, 2, \dots\}$ depends on the examples randomly picked at each iteration.
- It is *hoped* that (2) behaves like its expectation (1) despite the noise introduced by this approximation.
- Convergence results usually require decreasing learning rate satisfying the conditions $\sum_t \gamma_t^2 < \infty$ and $\sum_t \gamma_t = \infty$.

- **Average Stochastic Gradient Descent (ASGD)** performs Normal SGD update and recursively computes the average $\bar{\theta}_t = \frac{1}{t} \sum_{i=1}^t \theta_i$:

$$\theta_{t+1} = \theta_t - \gamma_t \nabla_{\theta} f(\mathbf{z}_t | \theta), \quad (3)$$

$$\bar{\theta}_{t+1} = \frac{t}{t+1} \bar{\theta}_t + \frac{1}{t+1} \theta_{t+1}. \quad (4)$$

Notice that $\bar{\theta}$ is the desired parameter. Generally, ASGD might need a large amount of data to converge.

- A proper learning rate schedule used in ASGD:

$$\gamma_t = \gamma_0 (1 + a\gamma_0 t)^c,$$

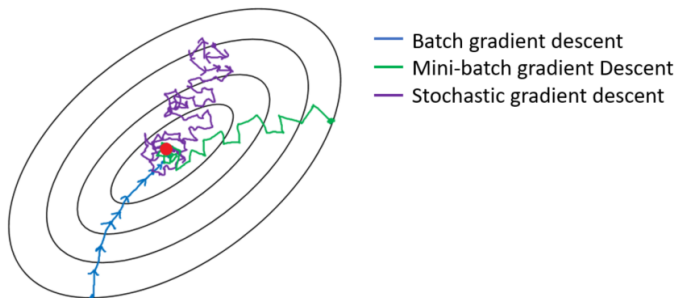
with which we can take a reasonable amount of data to reach its **asymptotic region**.

- Batch SGD is a compromise for estimating gradients.

Optimizer	Use data	Gradients
GD	all data	$\frac{1}{n} \sum_{i=1}^n \nabla f(\mathbf{z}_i)$
SGD	single data per step	$\nabla f(\mathbf{z}_t)$
Batch SGD	a batch of data	$\frac{1}{m} \sum_{i=1}^m \nabla f(\mathbf{z}_i), m < n$
Mini-Batch SGD	a mini-batch of data	$\frac{1}{m} \sum_{i=1}^m \nabla f(\mathbf{z}_i), m \ll n$

Table: GD and variants

- Although batch GD has better performance in the figure below, a huge amount of non-convex functions will make (large) batch GD perform poorly.



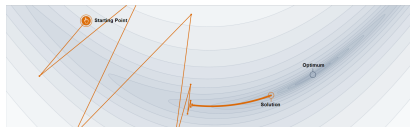
learning rate 0.0005



learning rate 0.003



learning rate 0.004



learning rate 0.005

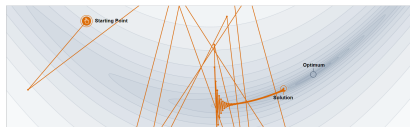
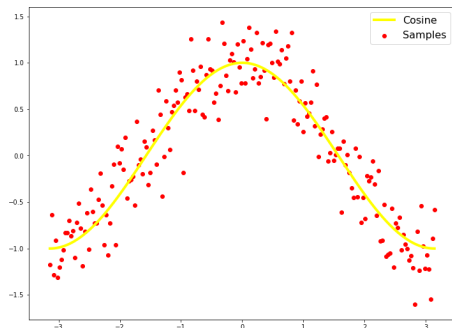


Figure: Why Momentum Really Works (<https://distill.pub/2017/momentum/>)

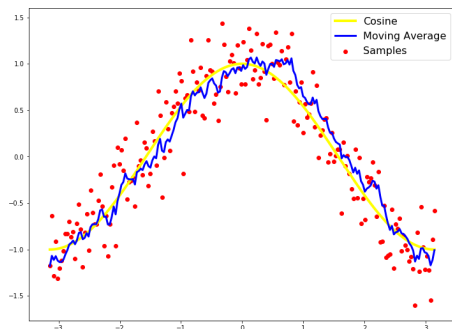
- Choose a proper learning rate is difficult.
- One learning rate schedule is unable to adapt to various datasets.
- Applying the same learning rate to all the parameters may not be the best.
- Object function is highly non-convex for neural networks, which means lost of local minima.
- Second-order optimization methods such as Newton's method are not suitable for deep learning. Because Hessian matrix is hard to compute.

1. Gradient Descent
2. Stochastic Gradient Descent
 - SGD
 - Average SGD
 - Batch SGD
 - Challenges
3. Momentum and Nesterov
 - Exponentially Weighted Average
 - Momentum SGD
 - Nesterov SGD
4. Adaptive Learning Rate
 - AdaGrad
 - AdaDelta
 - RMSProp
5. Adaptive Momentum
 - Adam
 - AdaMax
 - NAdam

- Samples from cosine function with Gaussian noise



- We want some kind of 'moving' average which would 'denoise' the data and bring it closer to original function.
- Exponentially weighted averages define a new sequence

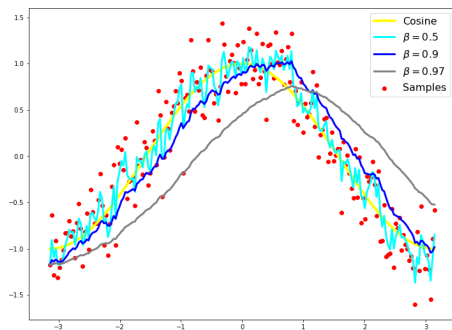


- Exponentially weighted average formula:

$$\mathbf{m}_t = \beta \mathbf{m}_{t-1} + (1 - \beta) \hat{\mathbf{m}}_t, \quad (5)$$

where $\{\mathbf{m}_t, t = 1, 2, \dots\}$ is a new moving-averaged sequence.

- Larger β will give smoother sequences but cause more seriously 'decay'.



- SGD has trouble navigating ravines(narrow valley), i.e. areas where the surface curves much more steeply in one dimension than in another, which are common around local optima.
- The system will oscillate back and forth in the direction of short axis and move very slowly along the long axis of the valley.
- How to accelerate SGD? Momentum!

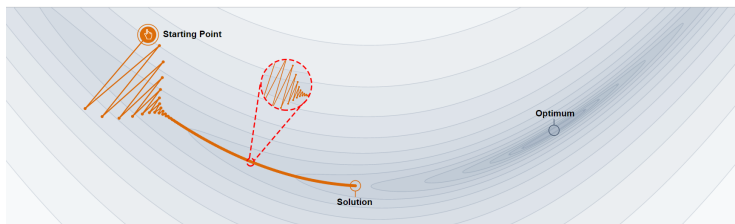


Figure: SGD without momentum, learning rate 0.003

Figure: Why Momentum Really Works (<https://distill.pub/2017/momentum/>)

- Recall exponentially weighted average:

$$\mathbf{m}_t = \beta \mathbf{m}_{t-1} + (1 - \beta) \hat{\mathbf{m}}_t.$$

- Remove $1 - \beta$ and let $\hat{\mathbf{m}}_t = \gamma \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t)$. Then we get

Momentum SGD

$$\mathbf{m}_t = \beta \mathbf{m}_{t-1} + \gamma \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t), \quad 0 < \beta < 1 \quad (6)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{m}_t \quad (7)$$

- Usually momentum parameter β is set to 0.9 or 0.99 that is close to 1.

- The modification of the parameters θ at the current times step depends on both the current gradient $\nabla_{\theta} f(\theta_t)$ and the parameters change \mathbf{m}_t of the previous step.



Figure: SGD with momentum 0.85, learning rate 0.003

Qian N. On the momentum term in gradient descent learning algorithms[J]. Neural networks, 1999, 12(1): 145-151.

Figure: Why Momentum Really Works (<https://distill.pub/2017/momentum/>)

- We will use momentum term $\beta \mathbf{m}_t$ to move parameter θ . So computing a "look-ahead" $\theta - \beta \mathbf{m}_t$ will give us an approximation of the next position of the parameters, which we can use to estimate gradients.



Yurii Nesterov

- We slightly adjust the formulas of Momentum SGD to get NAG:

Nesterov accelerated gradient (NAG)

$$\mathbf{m}_t = \beta \mathbf{m}_{t-1} + \gamma \nabla_{\theta} f(\theta_t - \beta \mathbf{m}_{t-1}), \quad 0 < \beta < 1 \quad (8)$$

$$\theta_{t+1} = \theta_t - \mathbf{m}_t \quad (9)$$

Nesterov Y. A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$ [C]//Doklady AN USSR. 1983, 269: 543-547.

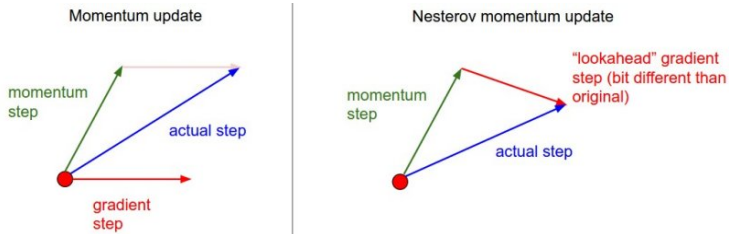


Figure: Momentum vs. Nesterov momentum

1. Gradient Descent
2. Stochastic Gradient Descent
 - SGD
 - Average SGD
 - Batch SGD
 - Challenges
3. Momentum and Nesterov
 - Exponentially Weighted Average
 - Momentum SGD
 - Nesterov SGD
4. Adaptive Learning Rate
 - AdaGrad
 - AdaDelta
 - RMSProp
5. Adaptive Momentum
 - Adam
 - AdaMax
 - NAdam

- **Adaptive gradient (AdaGrad)** adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters.

Adaptive gradient (AdaGrad)

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\gamma}{\sqrt{G_t + \epsilon}} \odot \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t), \quad (10)$$

where \odot is element-wise product, ϵ is a small number to prevent dividing by zero and $G_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix where each diagonal element (i, i) is the sum of the squares of the gradients w.r.t. $\boldsymbol{\theta}^{(i)}$ up to time step t :

$$G_t^{(i,i)} = \sum_{\tau=1}^t \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_{\tau})^2$$

- Advantages
 - It is well suited for sparse data.
 - It basically eliminates the need to tune the learning rate.
- Disadvantages
 - It is sensitive to the initial gradients. Large initial gradients result in lower learning rate and make training slower.
 - Due to the continual accumulation of squared gradients in the denominator, the learning rate will continue to decrease throughout training, eventually decreasing to zero and stop training.

- **AdaDelta** is an extension of AdaGrad for reducing its aggressive, monotonically decreasing learning rate. A natural idea is to restrict the accumulation window of the squared gradients. However, instead of storing a series of gradients which is not an efficient choice, AdaDelta use an exponentially weighted average (\mathbb{E}) of the square gradients:

$$\mathbb{E}[\nabla_{\theta} f(\theta)^2]_t = \rho \mathbb{E}[\nabla_{\theta} f(\theta)^2]_{t-1} + (1 - \rho) \nabla_{\theta} f(\theta_t)^2. \quad (11)$$

- Denote $\text{RMS}[\nabla_{\theta} f(\theta)]_t$ (**R**oot **M**ean **S**quare) the square root of \mathbb{E} :

$$\text{RMS}[\nabla_{\theta} f(\theta)]_t = \sqrt{\mathbb{E}[\nabla_{\theta} f(\theta)^2]_t + \epsilon}. \quad (12)$$

- But actually RMS represents Root Exponentially Weighted Average Square here.

- Then we get parameter update:

AdaDelta (Primary)

$$\theta_{t+1} = \theta_t - \frac{\gamma}{\text{RMS}[\nabla_{\theta} f(\theta)]_t} \odot \nabla_{\theta} f(\theta_t), \quad (13)$$

- Notice that if the optimization problem (such as a physical problem) has some hypothetical units, then the update formula above will lead to wrong units (comparing to SGD, here exists an extra unit of gradients in denominator). So as to AdaGrad.

- Considering the units of SGD:

$$\text{units of } \Delta\theta \propto \boxed{\text{units of } \nabla_{\theta} f(\theta) \propto \frac{\partial f}{\partial \theta}} \propto \frac{1}{\text{units of } \theta}, \quad (14)$$

assuming the cost function f is unitless.

- The SGD method will result in wrong units, while the second order methods such as Newton's method using Hessian information H do have the correct units for the parameter update:

$$\Delta\theta \propto H^{-1} \nabla_{\theta} f(\theta) \propto \boxed{\frac{\frac{\partial f}{\partial \theta}}{\frac{\partial^2 f}{\partial \theta^2}} \propto \text{units of } \theta}. \quad (15)$$

- We rearrange Newton's method (assuming a diagonal Hessian) for the inverse of the second derivative to determine the quantities involved:

$$\Delta\theta = \frac{\frac{\partial f}{\partial \theta}}{\frac{\partial^2 f}{\partial \theta^2}} \Rightarrow \boxed{\frac{1}{\frac{\partial^2 f}{\partial \theta^2}} = \frac{\Delta\theta}{\frac{\partial f}{\partial \theta}}}. \quad (16)$$

- Use equation(13) and boxed part of (14),(15), we can know that unit of $\frac{\Delta\theta}{\frac{\partial f}{\partial \theta}}$ in equation (16) is correct. So after applying some transformation, $\frac{\Delta\theta}{\frac{\partial f}{\partial \theta}}$ will be a proper substitute of $\frac{\gamma}{\text{RMS}[\nabla_{\theta} f(\theta)]_t}$.
- In denominator, $\frac{\partial f}{\partial \theta}$ is related to $\text{RMS}[\nabla_{\theta} f(\theta)]_t$, so in numerator, we can similarly replace γ with $\text{RMS}[\Delta\theta]_t$. But we still do not have $\Delta\theta_t$, so we use a delayed version $\text{RMS}[\Delta\theta]_{t-1}$.
- Finally we have final AdaDelta algorithm

AdaDelta (Final)

$$\theta_{t+1} = \theta_t - \frac{\text{RMS}[\Delta\theta]_{t-1}}{\text{RMS}[\nabla_{\theta} f(\theta)]_t} \odot \nabla_{\theta} f(\theta_t), \quad (17)$$

Algorithm 3 AdaDelta

Require: Decay rate ρ , Constant ϵ

Require: Initial parameter θ_0

- 1: Initialize accumulation variables $\mathbb{E}[\nabla_{\theta} f(\theta)^2]_0 = 0, \mathbb{E}[\Delta\theta^2]_0 = 0$
 - 2: **for** $t = 1 : T$ Loop over # of updates **do**
 - 3: Compute Gradient: $\nabla_{\theta} f(\theta_t)$
 - 4: Accumulate Gradient: $\mathbb{E}[\nabla_{\theta} f(\theta)^2]_t = \rho \mathbb{E}[\nabla_{\theta} f(\theta)^2]_{t-1} + (1 - \rho) \nabla_{\theta} f(\theta_t)^2$
 - 5: Compute Update: $(\Delta\theta)_t^2 = -\frac{\text{RMS}[\Delta\theta]_{t-1}}{\text{RMS}[\nabla_{\theta} f(\theta)]_t} \nabla_{\theta} f(\theta_t)$
 - 6: Accumulate Updates: $\mathbb{E}[(\Delta\theta)^2]_t = \rho \mathbb{E}[(\Delta\theta)^2]_{t-1} + (1 - \rho) (\Delta\theta_t)^2$
 - 7: Apply Update: $\theta_{t+1} = \theta_t + \Delta\theta_t$
 - 8: **end for**
-

- **RMSPProp**, proposed by Geoffery Hinton, is just the primary version of AdaDelta.

RMSPProp

$$\theta_{t+1} = \theta_t - \frac{\gamma}{\text{RMS}[\nabla_{\theta} f(\theta)]_t} \odot \nabla_{\theta} f(\theta_t), \quad (18)$$

- A typical choice of ρ in *RMSPProp* is 0.9.

1. Gradient Descent
2. Stochastic Gradient Descent
 - SGD
 - Average SGD
 - Batch SGD
 - Challenges
3. Momentum and Nesterov
 - Exponentially Weighted Average
 - Momentum SGD
 - Nesterov SGD
4. Adaptive Learning Rate
 - AdaGrad
 - AdaDelta
 - RMSProp
5. Adaptive Momentum
 - Adam
 - AdaMax
 - NAdam

- Inspired by AdaDelta storing exponentially weighted average of previous squared gradients(denote \mathbf{m}_t), **Adaptive Momentum(Adam)** Estimation also keeps an exponentially weighted average of previous gradients(denote \mathbf{v}_t), similar to momentum:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \nabla_{\theta} f(\theta_t) \quad (19)$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \nabla_{\theta} f(\theta_t)^2 \quad (20)$$

where \mathbf{m}_t and \mathbf{v}_t can also be regarded as the first moment(the mean) and the second moment(the uncentered variance) of the gradients, respectively.

- β_1 and β_2 are close to 1 just like Momentum, and RMSProp.
- Notice that if we initialize $\mathbf{m}_0 = \mathbf{0}$ and $\mathbf{v}_0 = \mathbf{0}$, then the gradients are biased towards zero, especially in initial steps.

- We need the algorithm have larger moments at the beginning of training to counteract the biases mentioned above:

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t}, \quad (21)$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t}. \quad (22)$$

And we give a brief derivation of equation (21) and (22) in next slide.

- Finally we get Adam update rule:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\gamma}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \hat{\mathbf{m}}_t. \quad (23)$$

- Let $\mathbf{g} = \nabla_{\theta} f(\theta)$. For the exponentially weighted average, we have

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) g_t^2 = (1 - \beta_2) \sum_{i=1}^t \beta_2^{t-i} \mathbf{g}_i^2. \quad (24)$$

We wish to know how $\mathbb{E}[\mathbf{v}_t]$, the expected value of the exponentially weighted average at timestep t , relates to the true second order moment $\mathbb{E}[\mathbf{g}_t^2]$, so we can correct for the discrepancy between the two. Taking expectations of equation (24):

$$\begin{aligned} \mathbb{E}[\mathbf{v}_t] &= \mathbb{E} \left[(1 - \beta_2) \sum_{i=1}^t \beta_2^{t-i} \mathbf{g}_i^2 \right] \\ &= \mathbb{E}[\mathbf{g}_t^2] \cdot (1 - \beta_2) \sum_{i=1}^t \beta_2^{t-i} + \zeta \\ &= \mathbb{E}[\mathbf{g}_t^2] \cdot (1 - \beta_2^t) + \zeta \end{aligned} \quad (25)$$

where $\zeta = 0$ if $\mathbb{E}[\mathbf{g}_t^2]$ is stationary; otherwise just choice β_2 to make ζ close to zero.

- Adam = $\underbrace{\text{Momentum}}_{\text{Weighted Average Gradients}} \oplus \underbrace{\text{RMSProp}}_{\text{Adaptive Learning Rate}} \oplus \text{InitialAdjustment}$.
- We display complete Adam routine here:

Adam

$$\gamma_t = \gamma \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t} \quad (26)$$

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t) \quad (27)$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t)^2 \quad (28)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\gamma_t}{\sqrt{\mathbf{v}_t} + \epsilon} \odot \mathbf{m}_t. \quad (29)$$

- The authors propose default values $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

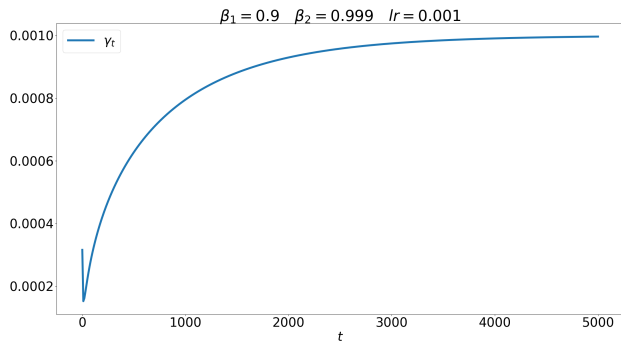


Figure: Adjusted learning rate (i.e. equation (26)) at beginning 5000 steps

- **AdaMax** generalize the L_2 norm of gradients of Adam in equation (28) to p norm L_p .
- With some slight modification, we have

$$\begin{aligned}\mathbf{v}_t &= \beta_2^p \mathbf{v}_{t-1} + (1 - \beta_2^p) \|\mathbf{g}_t\|^p \\ &= (1 - \beta_2^p) \sum_{i=1}^t \beta_2^{p(t-i)} \cdot \|\mathbf{g}_i\|^p.\end{aligned}\tag{30}$$

Note that we replace β_2 with β_2^p .

- Let $p \rightarrow \infty$, then:

$$\begin{aligned}
 \mathbf{u}_t &:= \lim_{p \rightarrow \infty} (\mathbf{v}_t)^{1/p} = \lim_{p \rightarrow \infty} \left((1 - \beta_2^p) \sum_{i=1}^t \beta_2^{p(t-i)} \cdot \|\mathbf{g}_i\|^p \right)^{1/p} \\
 &= \lim_{p \rightarrow \infty} (1 - \beta_2^p)^{1/p} \left(\sum_{i=1}^t \beta_2^{p(t-i)} \cdot \|\mathbf{g}_i\|^p \right)^{1/p} \\
 &= \lim_{p \rightarrow \infty} \left(\sum_{i=1}^t (\beta_2^{t-i} \cdot \|\mathbf{g}_i\|)^p \right)^{1/p} \\
 &= \max(\beta_2^{t-1} \|\mathbf{g}_1\|, \beta_2^{t-2} \|\mathbf{g}_2\|, \dots, \beta_2 \|\mathbf{g}_{t-1}\|, \|\mathbf{g}_t\|). \quad (31)
 \end{aligned}$$

which can be write into a recursive formula:

$$\mathbf{u}_t = \max(\beta_2 \cdot \mathbf{u}_{t-1}, \|\mathbf{g}_t\|) \quad (32)$$

with initial value $\mathbf{u}_0 = \mathbf{0}$.

- Now we have AdaMax algorithm

AdaMax

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t) \quad (33)$$

$$\mathbf{u}_t = \max(\beta_2 \cdot \mathbf{u}_{t-1}, \|\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t)\|) \quad (34)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\gamma}{(1 - \beta_1^t)(\mathbf{u}_t + \epsilon)} \odot \mathbf{m}_t. \quad (35)$$

- We know that Adam \approx Momentum \oplus RMSProp and Nesterov is an improved version of Momentum. So we can replace Momentum with a modified version of Nesterov in Adam and get **Nesterov Adam(NAdam)**.
- For Momentum update rule we can put them into one line:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - (\beta \mathbf{m}_{t-1} + \gamma \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t)). \quad (36)$$

- And for Nesterov update rule:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - (\beta \mathbf{m}_{t-1} + \gamma \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t - \beta \mathbf{m}_{t-1})). \quad (37)$$

- In NAdam, we replace \mathbf{m}_{t-1} in Momentum with \mathbf{m}_t directly, which can be viewed as a weighted momentum:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - (\beta^2 \mathbf{m}_{t-1} + (\beta + 1) \gamma \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t)). \quad (38)$$

- We can formalize Adam to a single line:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\gamma_t}{\sqrt{\mathbf{v}_t} + \epsilon} \odot (\beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t)) \quad (39)$$

Replace \mathbf{m}_{t-1} with $\tilde{\mathbf{m}}_t$ and we get NAdam

NAdam

$$\gamma_t = \gamma \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t} \quad (40)$$

$$\tilde{\mathbf{m}}_t = \beta_1 \mathbf{m}_t + (1 - \beta_1) \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t) \quad (41)$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t)^2 \quad (42)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\gamma_t}{\sqrt{\mathbf{v}_t} + \epsilon} \odot \tilde{\mathbf{m}}_t. \quad (43)$$

Thank You!