Optimization Algorithms 2: RMSprop, Adadelta & Adam

ASTR 502
Maria Gabriela Cota Moreira
March 21st

Recap 1

AdaGrad

What it does?

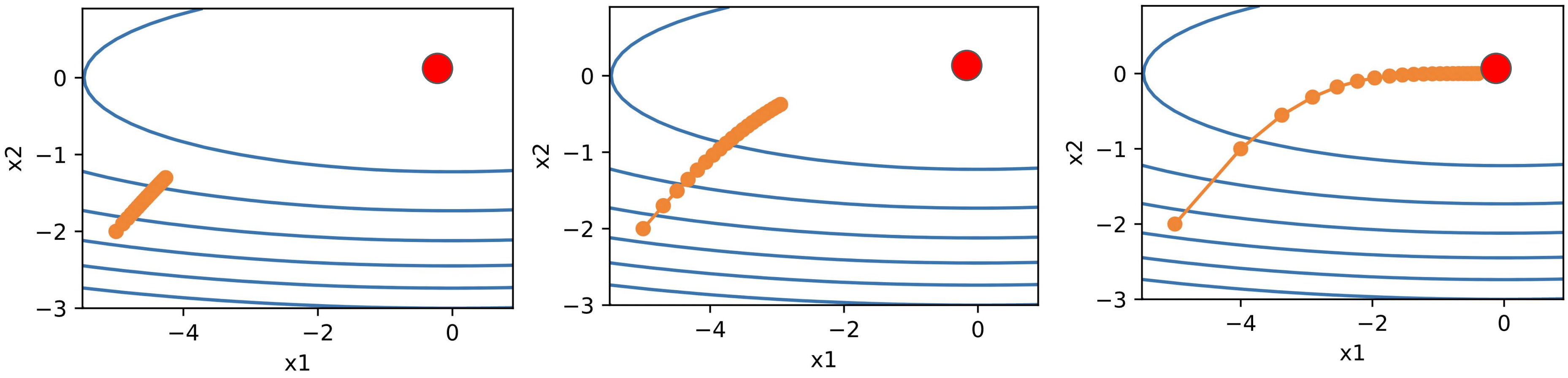
Includes an adaptive learning rate that decreases the step-size with time

1. $\boldsymbol{g}_t = \partial_w \mathcal{L}(\boldsymbol{w_t})$ — Loss function gradient
2. $\boldsymbol{s}_t = \boldsymbol{s}_{t-1} + \boldsymbol{g}_t^2$ — Learning rate scaling
3. $\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \frac{\eta}{\sqrt{\boldsymbol{s}_t + \epsilon}} \boldsymbol{g}_t$ — Parameter update

Issue: The more you update the less you update

$$\boldsymbol{s}_t = \sum_{\tau=0}^{t} \boldsymbol{g}_\tau^2 \qquad \boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \frac{\eta}{\sqrt{\boldsymbol{s}_t + \epsilon}}\boldsymbol{g}_t$$

Slow and sensitive to the learning rate!



$\eta = 0.1$ $\eta = 0.3$ $\eta = 1.0$

RMSprop

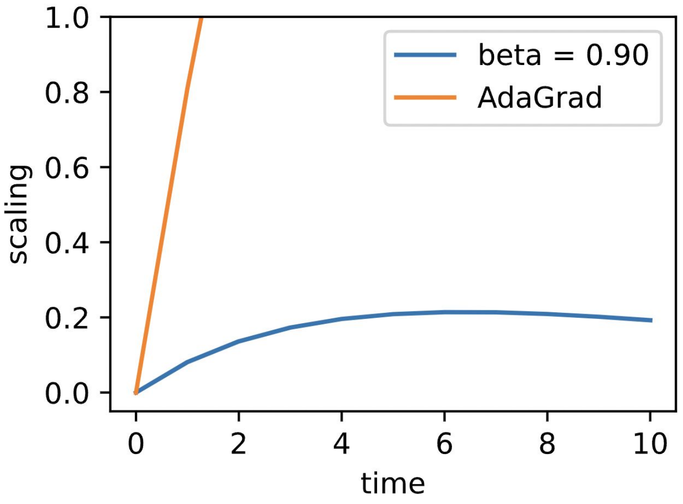
Root-Mean-Square Propagation

What's new?

Includes a damping parameter to AdaGrad, β w/ same $\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \frac{\eta}{\sqrt{\boldsymbol{s}_t + \epsilon}} \boldsymbol{g}_t$

Leaky average of the squared gradients

$$\boldsymbol{s}_t = \beta \boldsymbol{s}_{t-1} + (1 - \beta) \boldsymbol{g}_t^2$$



<u>Keeps the squares under a manageable size the whole time !</u>

<u>Initial values and params</u>

$\boldsymbol{s}_0 = 0$

$\beta = 0.9$

$\epsilon = 10^{-8}$

How it works? We give more importance to recent gradients

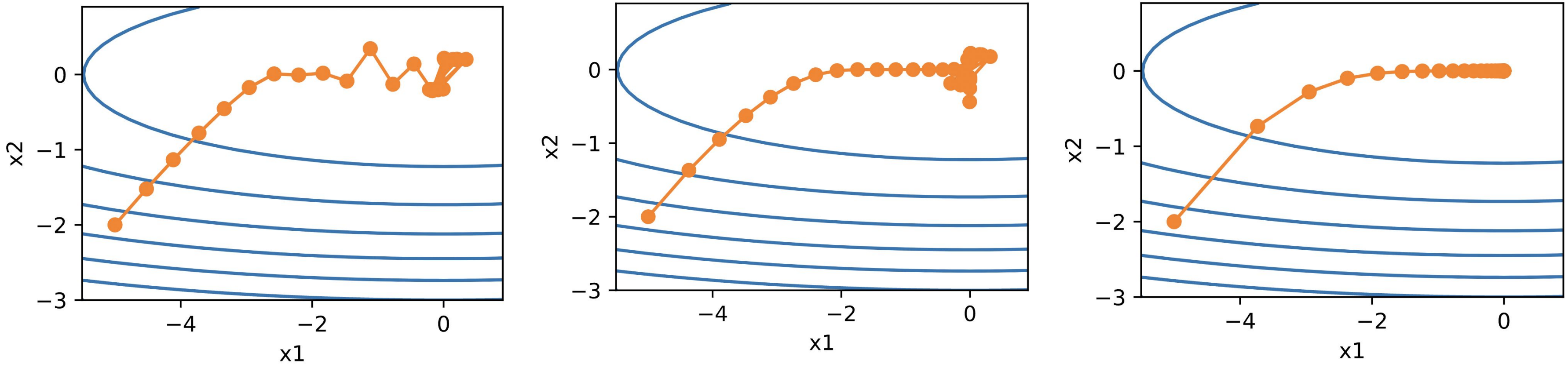
$$\boxed{0 < \beta < 1}$$

$$\beta = 0.9$$

$$\boldsymbol{s}_t = \beta \boldsymbol{s}_{t-1} + (1-\beta)\boldsymbol{g}_t^2$$

$$\boldsymbol{s}_t = (1-\beta)(\boldsymbol{g}_t^2 + \beta \boldsymbol{g}_{t-1}^2 + \beta^2 \boldsymbol{g}_{t-2}^2 + \beta^3 \boldsymbol{g}_{t-3}^2 + \dots)$$

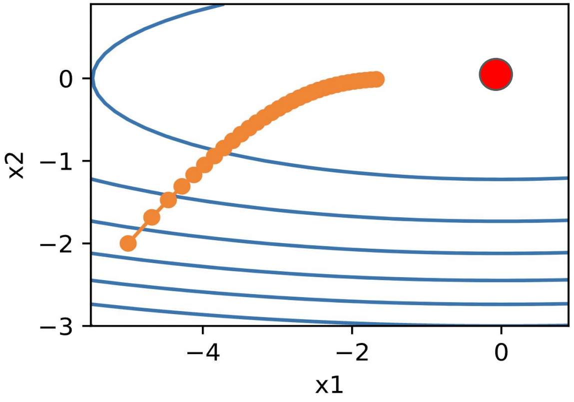
lower weight to previous gradients variance



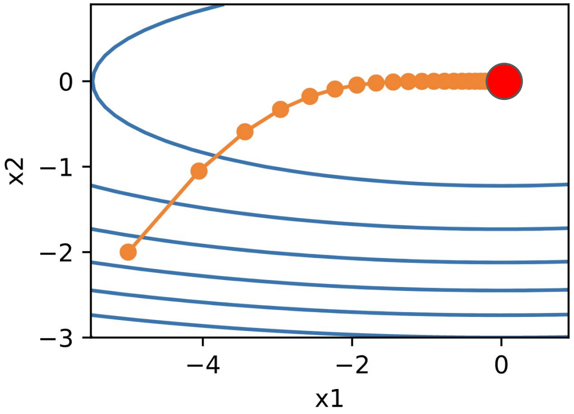
$\beta = 0.3$ $\beta = 0.6$ $\beta = 0.9$

Potential issue

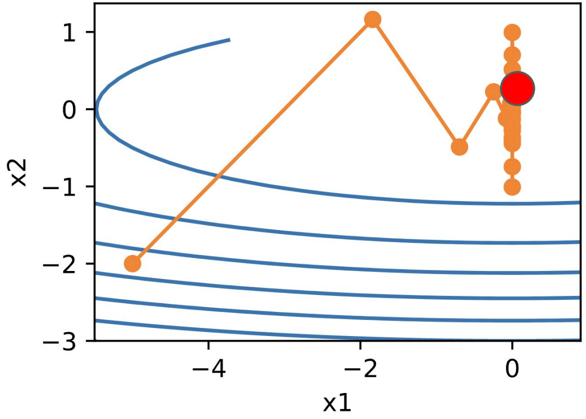
Learning rate is still manual, because the suggested value is not always appropriate for every task.



$\eta = 0.1$



$\eta = 0.3$



$\eta = 1.0$

ADADELTA

An Adaptive Learning Rate Method

What's new?

I) Also Includes a damping parameter to AdaGrad

II) Address the need for a manually selected global learning rate (kinda…)

1. $\boldsymbol{g}_t = \partial_w \mathcal{L}(\boldsymbol{w_t})$

2. $\boldsymbol{s}_t = \beta \boldsymbol{s}_{t-1} + (1-\beta)\boldsymbol{g}_t^2$

→ Same as RMSprop

Initial values

$\Delta \boldsymbol{w}_0 = 0$

$\boldsymbol{s}_0 = 0$

3. $\boldsymbol{g}'_t = \frac{\sqrt{\Delta \boldsymbol{w}_{t-1} + \epsilon}}{\sqrt{\boldsymbol{s}_t + \epsilon}} \odot \boldsymbol{g}_t$ → Step-size

4. $\Delta \boldsymbol{w}_t = \beta \Delta \boldsymbol{w}_{t-1} + (1-\beta)\boldsymbol{g}_t'^2$ → Leaky average of the step-size

5. $\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \boldsymbol{g}'_t$ → Parameter update

$\beta = 0.9$

Motivation/How it works

Numerator arises from unit correction.

In AdaGrad,

$$\boldsymbol{g}_t' = -\frac{\eta}{\sqrt{\boldsymbol{s}_t + \epsilon}} \odot \boldsymbol{g}_t \quad \text{is } \textbf{unitless} \quad \boldsymbol{g}_t' \propto \frac{\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}}{\sqrt{\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}\right)^2}}$$

But it should have units of **w** since $\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \boldsymbol{g}_t'$. Then,

$$\boldsymbol{g}_t' \propto \frac{\boxed{\Delta \boldsymbol{w}_t} \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}}{\sqrt{\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}\right)^2}} \propto \text{units of } \boldsymbol{w}_t$$

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \frac{\boxed{\sqrt{\Delta \boldsymbol{w}_{t-1} + \epsilon}}}{\sqrt{\boldsymbol{s}_t + \epsilon}} \odot \boldsymbol{g}_t$$

It acts as a "momentum" term

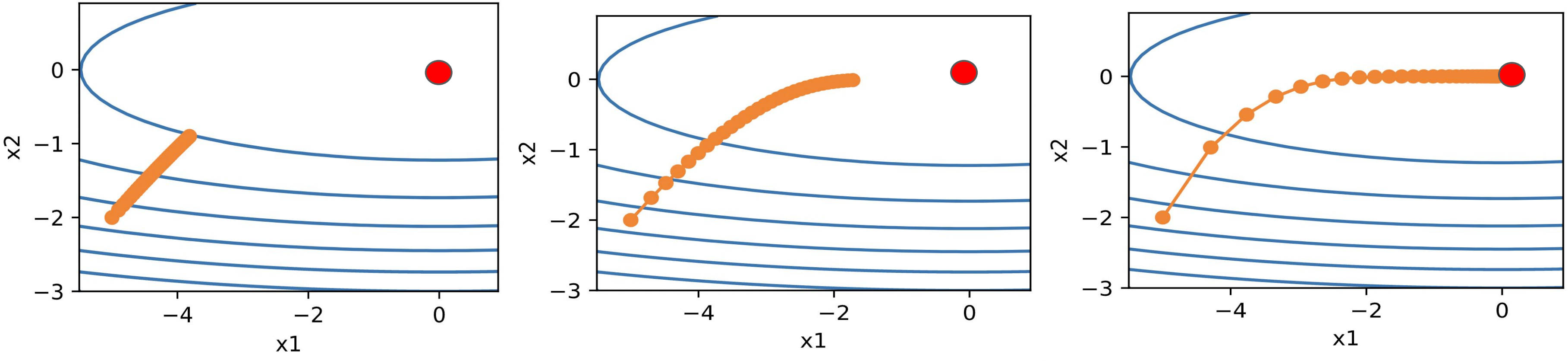
Potential issues *(that I ran into)*

When I chose a small ϵ the algorithm converges slowly

$$\boldsymbol{g}'_t = \frac{\sqrt{\Delta \boldsymbol{w}_{t-1} + \epsilon}}{\sqrt{\boldsymbol{s}_t + \epsilon}} \odot \boldsymbol{g}_t$$

Is it really "learning-rate free"?

Changing the initial Δw_0 also leads to a similar effect: https://akyrillidis.github.io/notes/AdaDelta



$\epsilon = 0.001$ | $\epsilon = 0.01$ | $\epsilon = 0.1$

Potential issues *(that I ran into)*

ϵ is behaving as a **learning rate**. But why?

$$\boldsymbol{g}'_t = \frac{\sqrt{\Delta \boldsymbol{w}_{t-1} + \epsilon}}{\sqrt{\boldsymbol{s}_t + \epsilon}} \odot \boldsymbol{g}_t$$

at the first iteration/epoch (t=1) you have

$$\boldsymbol{s}_1 = (1-\beta)\boldsymbol{g}_1^2$$

$$\Delta \boldsymbol{w}_0 = 0$$

then,

$$\boldsymbol{g}'_1 = \frac{\sqrt{\epsilon}}{\sqrt{\boldsymbol{s}_1 + \epsilon}} \odot \boldsymbol{g}_1 \Longrightarrow \Delta \boldsymbol{w}_1 = (1-\beta)\boldsymbol{g}_1'^2$$

$$\eta = \sqrt{\epsilon}$$

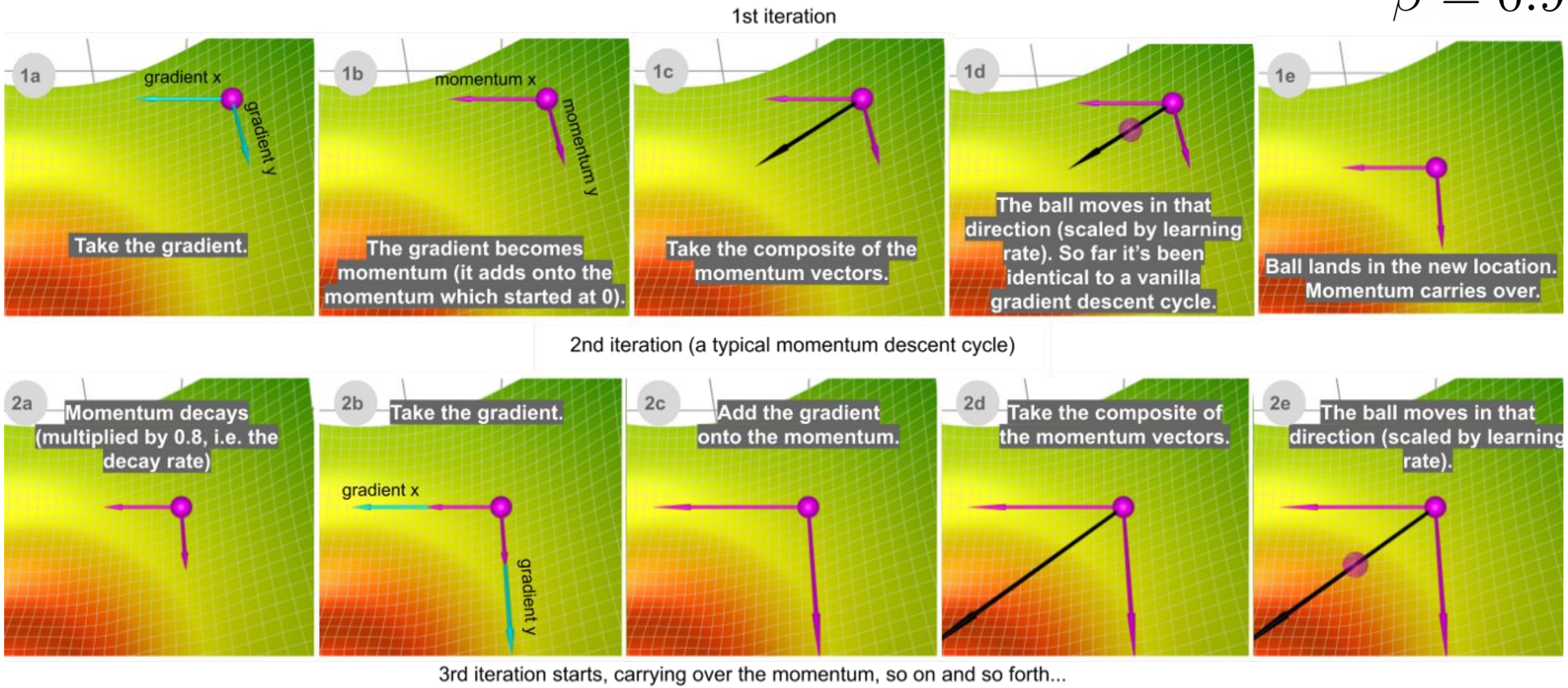
Recap 2

Momentum

Momentum accelerates motion towards the minima

$$\boldsymbol{v}_t = \beta \boldsymbol{v}_{t-1} + (1-\beta)\boldsymbol{g}_t \qquad \boldsymbol{w}_t = \boldsymbol{w}_{t-1} + \eta \boldsymbol{v}_t$$

$$\beta = 0.9$$



Adam

Adaptive Moment Estimation

What's new?

Utilize the **momentum** concept and **adaptive learning rate** from RMSprop

Momentum

$$\boldsymbol{v}_t = \beta_1 \boldsymbol{v}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$$

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} + \eta \boldsymbol{v}_t$$

+

RMSprop

$$\boldsymbol{s}_t = \beta_2 \boldsymbol{s}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2$$

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \frac{\eta}{\sqrt{\boldsymbol{s}_t + \epsilon}}\boldsymbol{g}_t$$

=

Adam

bias correction

$$\hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1-\beta_1^t} \qquad \hat{\boldsymbol{s}}_t = \frac{\boldsymbol{s}_t}{1-\beta_2^t}$$

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \frac{\eta}{\sqrt{\hat{\boldsymbol{s}}_t + \epsilon}} \odot \hat{\boldsymbol{v}}_t$$

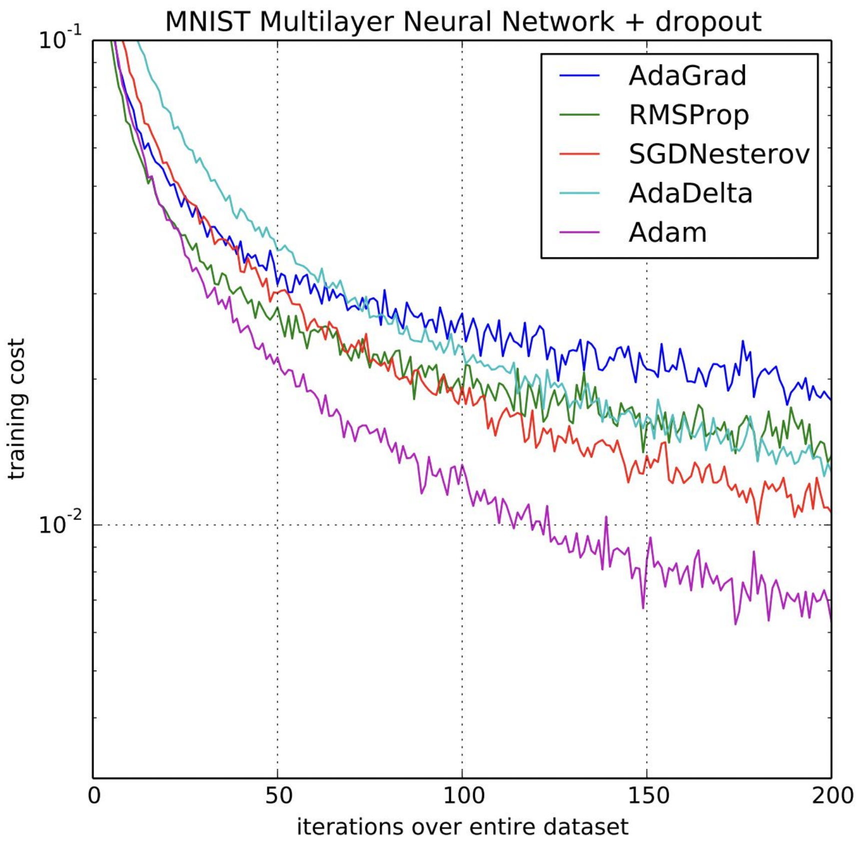
$\beta_1 = 0.9 \quad \beta_2 = 0.999 \quad \epsilon = 10^{-8}$

How it works?

Momentum accelerates motion

RMSprop controls oscillations

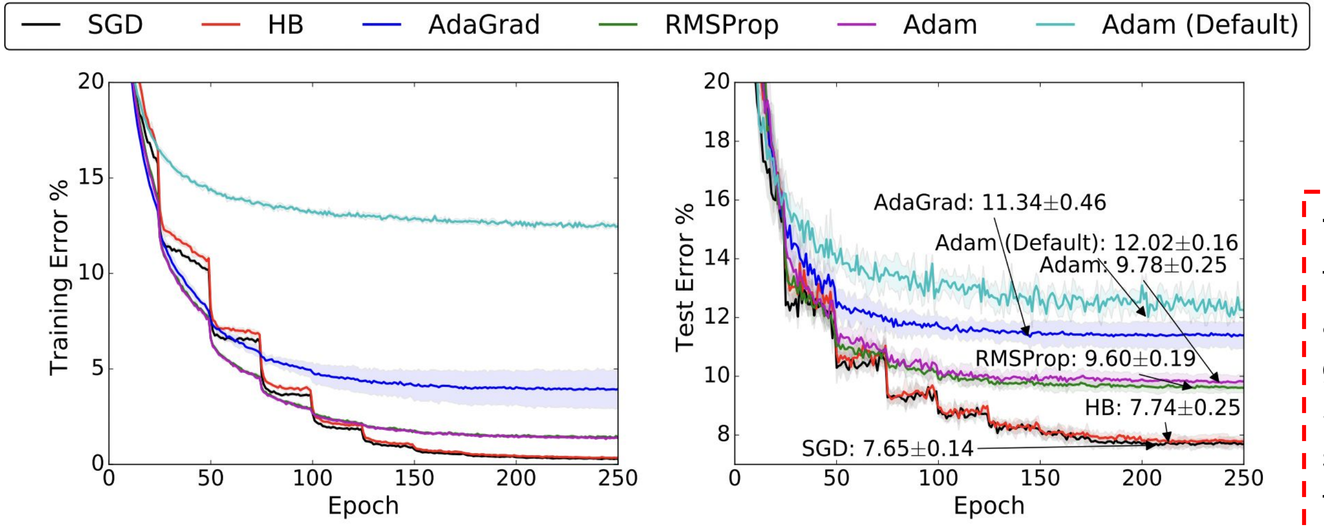
Taking advantage of both is what makes Adam powerful and popular



https://arxiv.org/pdf/1412.6980.pdf

Potential issues Don't trust on the default settings too much

Many hyperparameters to tune: $\beta_1, \beta_2, \eta, \epsilon, \boldsymbol{s}_0$ and $\boldsymbol{v}_0$



(a) CIFAR-10 (Train) **(b)** CIFAR-10 (Test)

Figure 1: Training (left) and top-1 test error (right) on CIFAR-10. The annotations indicate where the best performance is attained for each method. The shading represents ± one standard deviation computed across five runs from random initial starting points. In all cases, adaptive methods are performing worse on both train and test than non-adaptive methods.

https://arxiv.org/pdf/1705.08292.pdf

The authors observe that the solutions found by adaptive methods generalize worse than SGD, even when these solutions have better training performance

References

Dive into Deep Learning, Ch.11.8-11.10 (https://d2l.ai/)

https://akyrillidis.github.io/notes/AdaDelta

M. D. Zeiler “ADADELTA: An adaptive learning rate method” (arXiv:1212.5701, 2012)

D. P. Kingman and J. Lei Ba∗“Adam: A method for stochastic optimization” (arXiv:1412.6980)

https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

Gradient Descent Visualization (https://github.com/lilipads/gradient_descent_viz)

A. C. Wilson et al. “The Marginal Value of Adaptive Gradient Methods in Machine Learning” (arXiv:1705.08292)