535520: Optimization Algorithms

Lecture 14: Optimizers for Neural Networks

Ping-Chun Hsieh (謝秉均)

December 18, 2024

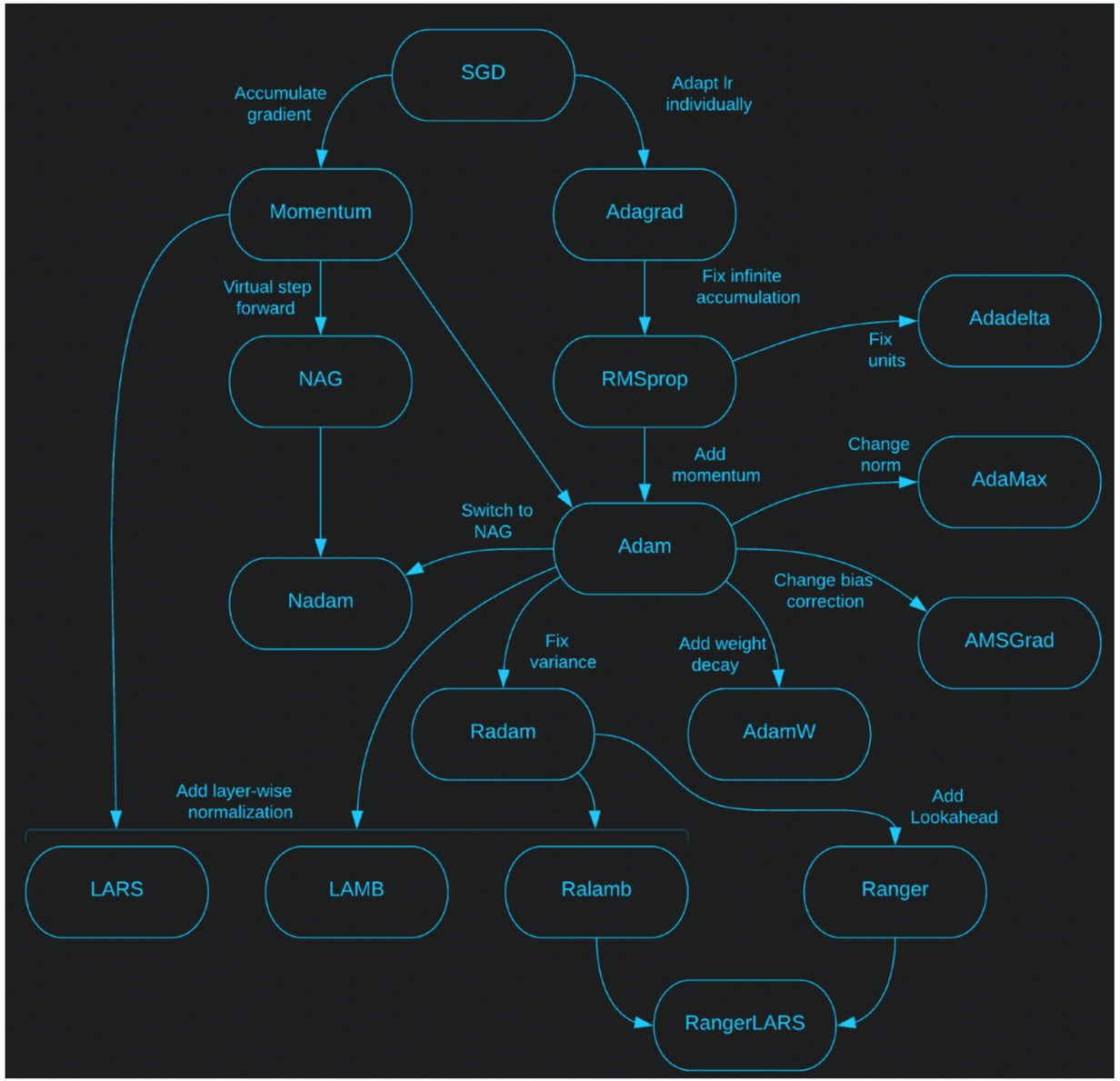
This Lecture

1. AdaGrad

2. RMSProp, Adam, and Beyond

- Reading Material:
  - Duchi and Hazan, “Adaptive subgradient methods for online learning and stochastic optimization,” COLT 2011.
  - Kings and Ba, “Adam: A Method for Stochastic Optimization,” ICLR 2015.
  - Part of the material is adapted from Prof. Gabriele Farina’s lecture notes (https://www.mit.edu/~gfarina/2024/67220s24_L13_adagrad/L13.pdf)

A Whole Family of Optimizers



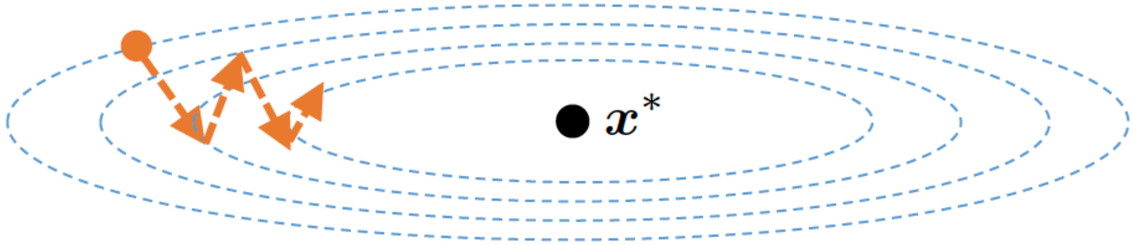
- A lot of variants of SGD for optimization in practice
  - Learning rate
  - Momentum

- Today: We focus on the most widely used variants
  - AdaGrad
  - RMSProp
  - Adam
  - NAdam

Figure Source: https://darel13712.github.io/ml/optimizers.html

Recall: Non-Homogeneity and "Scaled" Gradients

$$\text{minimize}_{x \in \mathbb{R}^2} \quad f(x) := \frac{1}{2}(x - x^*)^\top Q (x - x^*)$$



- Suppose $Q = [Q_{11}, 0; 0, Q_{22}]$ is a diagonal matrix with $Q_{11} \gg Q_{22}$

- Idea: Accelerate GD by *scaling the gradient*

$$x_{t+1} = x_t - \eta_t Q^{-1} \nabla f(x_t) = \_\_\_\_\_$$

- Another interpretation: Scale the learning rate of each input variable separately
  (in other words, $\eta_t Q^{-1}$ shall be jointly chosen to tackle non-homogeneity)

- Question: In practice, how to jointly choose $\eta_t Q^{-1}$?

AdaGrad: Adaptive Gradients

Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*

John Duchi JDUCHI@CS.BERKELEY.EDU
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720 USA

Elad Hazan EHAZAN@IE.TECHNION.AC.IL
Technion - Israel Institute of Technology
Technion City
Haifa, 32000, Israel

Yoram Singer SINGER@GOOGLE.COM
Google
1600 Amphitheatre Parkway
Mountain View, CA 94043 USA

(COLT 2011)

This work has been cited for more than 14,000 times

John Duchi
(Professor @ Stanford)

Elad Hazan
(Professor @ Princeton)

AdaGrad: Adaptive Gradients (Formally)

$x_t, s_t \in \mathbb{R}^d$

- Main idea: Scale the learning rate of each input variable (i.e., per-dimension learning rate) based on the sum of squared gradients accumulated over time

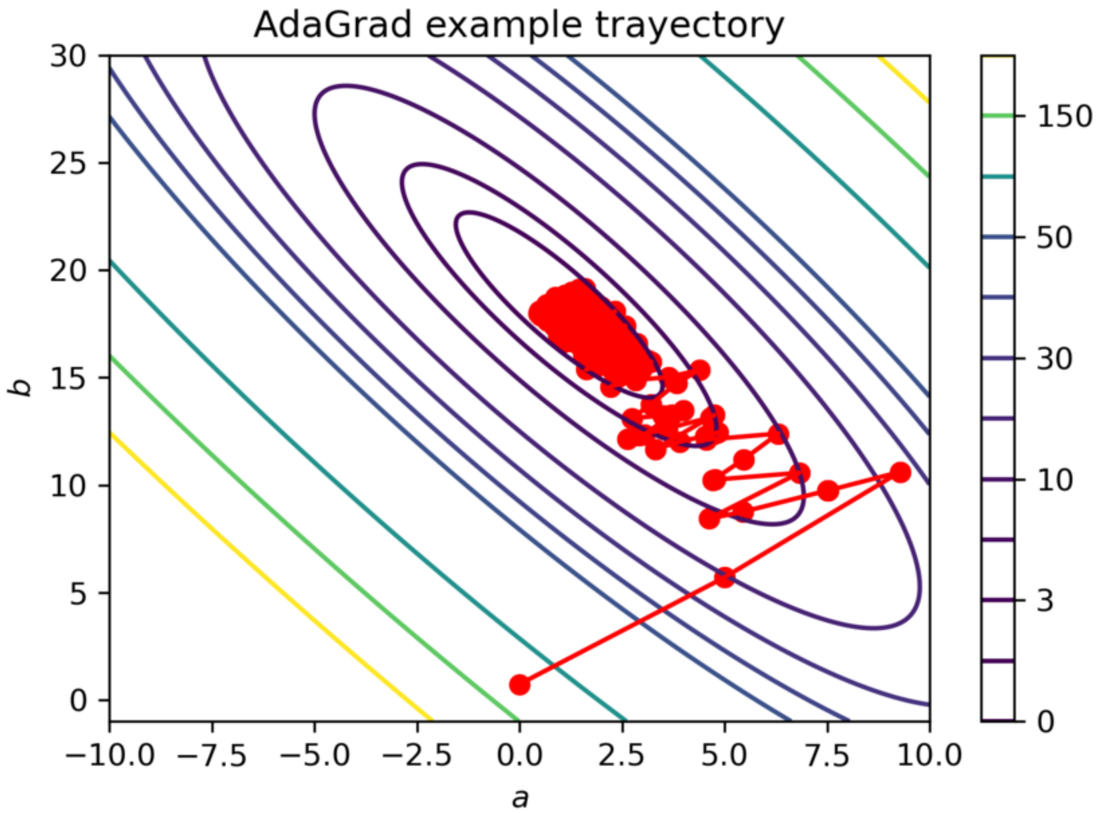
AdaGrad

$$x_{t+1} = x_t - \eta M_t^{-1} \nabla f(x_t), \quad \text{where } M_t = \text{diag}\left([s_t]_i := \sqrt{\sum_{\tau=0}^{t} [\nabla f(x_\tau)]_i^2} \right)$$

i-th component of the gradient at time τ

- Remark: We assume $[\nabla f(x_0)]_i \neq 0$ for all i, and hence M_t is invertible
- Remark: In the stochastic setting, we can replace the true gradient $\nabla f(x_\tau)$ with the stochastic gradient $\hat{\nabla} f(x_\tau)$
- Remark: Clearly $[s_t]_i$ is monotonically increasing w.r.t. iteration t

AdaGrad: Numerical Example



Training dataset: Randomly generated data samples as $z = 20y + \varepsilon$

Learned predictor: $z = a + by$

Loss function: $f(a,b) = \frac{1}{n}\sum_{i=1}^{n}[(a + by_i) - z_i]^2$

Initial parameters: $x_0 \equiv (a,b) = (0,0)$

- If the first sample is $(y_1, z_1) = (0.39, 9.84)$, what is x_1 under AdaGrad?

$$x_{t+1} = x_t - \eta M_t^{-1} \nabla f(x_t)$$

$$M_t = \text{diag}\left([s_t]_i := \sqrt{\sum_{\tau=0}^{t} [\nabla f(x_\tau)]_i^2} \right)$$

Example: https://optimization.cbe.cornell.edu/index.php?title=AdaGrad

Intuition Behind AdaGrad

- Imagine a high-dimensional problem of $d \gg 1$
- Each gradient update can be “sparse” (only a few entries dominate the magnitude)
- Question: Is it good to use only one learning rate for all input variables?

AdaGrad uses separate counters for different variables

1. If a variable $x^{(i)}$ has been updated many times, then the learning rate shall be made smaller

2. To “count” the number of updates, AdaGrad considers $[s_t]_i := \sqrt{\sum_{\tau=0}^{t} [\nabla f(x_\tau)]_i^2}$

AdaGrad is a Mirror Descent Algorithm

Theorem: AdaGrad's update rule is equivalent to the mirror descent update with Bregman divergence $D_\phi(y\|x) = \frac{1}{2}(x-y)^\top M_t (x-y)$, where $M_t := \text{diag}(s_t)$

Proof: Recall the "proximal viewpoint" of MD

$$x_{t+1} = \arg\min_x \left\{ \eta \nabla f(x_t)^\top x + D_\phi(x\|x_t) \right\} = \arg\min_x \left\{ \eta \nabla f(x_t)^\top x + \frac{1}{2}(x-y)^\top M_t (x-y) \right\}$$

By setting the gradient of the above to 0 and solving for x:

$$\eta \nabla f(x_t) + M_t(x - x_t) = 0 \quad \Leftrightarrow$$

Remark: AdaGrad captures local geometry via M_t

Review: Mirror Descent (MD)

Key Idea: Generalize the L_2 proximal term to other distance measures

$$x_{t+1} = \arg\min_{x \in C} \left\{ \underbrace{f(x_t) + \nabla f(x_t)^\top (x - x_t)}_{\text{first-order approximation}} + \frac{1}{\eta_t} \underbrace{D_\phi(x \| x_t)}_{\text{Bregman divergence}} \right\}$$

where $D_\phi(y \| x) := \phi(y) - \phi(x) - \nabla \phi(x)^\top (y - x)$

($\phi(\cdot)$ is a strictly convex and differentiable "distance-generating" function)

- Remark: Bregman divergence is meant to capture "local geometry" of objective function

Review: Bregman Divergence as Mahalanobis Distance

Suppose $\phi(x) = x^\top A x$ (where A is a positive definite $d \times d$ matrix)

- The resulting Bregman divergence is $D_\phi(y\|x) = (x-y)^\top A(x-y)$

(aka "Mahalanobis distance")

$$D_\phi(y\|x) := \phi(y) - \big(\phi(x) + \nabla\phi(x)^\top(y-x)\big)$$

$$= y^\top A y - \big(x^\top A x + 2(Ax)^\top(y-x)\big)$$

$$= y^\top A y + x^\top A x - 2x^\top A y$$

$$=$$

Performance Guarantee of AdaGrad

Theorem (Duchi et al., 2011): Let f be a convex and differentiable function. For any choice of coefficients $\lambda_i \geq 0$, AdaGrad with step size $\eta = D/\sqrt{2}$ satisfies

$$\frac{1}{T}\sum_{t=0}^{T-1} f(x_t) \leq f(x_*) + \frac{\sqrt{2d}D}{T}\sqrt{\min_{\lambda \in \mathbb{R}^n_{\geq 0}, \|\lambda\|_1 = n} \sum_{t=0}^{T-1} \nabla f\left(x_t\right)^\top \operatorname{diag}(\lambda)^{-1} \nabla f\left(x_t\right)},$$

where $D := \max_{t \in [0,T]} \|x_t - x^*\|_\infty$ is the maximum distance from the optimizer

Moreover, with bounded gradients, AdaGrad achieves a regret bound of

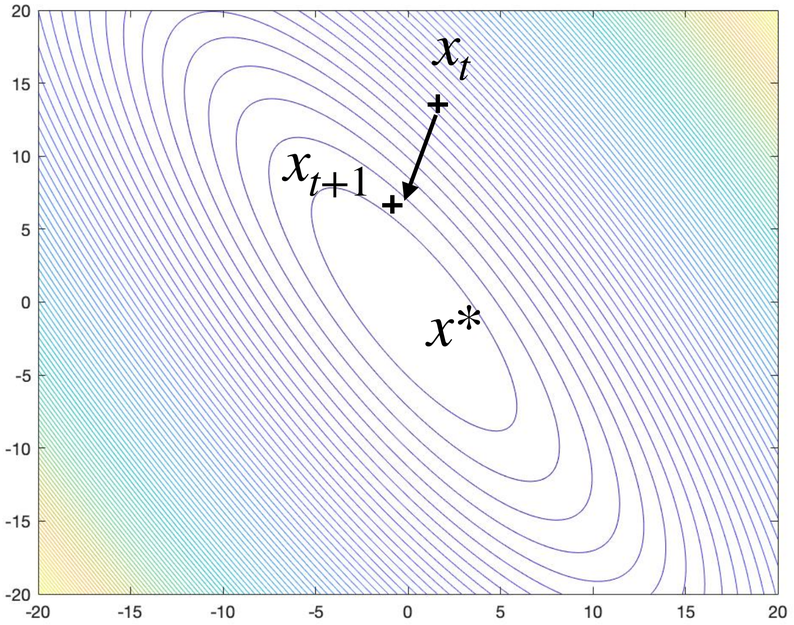
$$R(T) := \sum_{t=1}^{T} \left(f(x_t) - f(x^*)\right) = O(\sqrt{T})$$

Recall: Mirror Descent Lemma

Lemma: Let f be a convex function. Under Mirror descent update, we have

$$f(x_t) \leq f(x^*) + \frac{1}{\eta}\left(D_\phi\left(x^* \| x_t\right) - D_\phi\left(x^* \| x_{t+1}\right) + D_\phi\left(x_{t+1} \| x_t\right)\right).$$

- What's the intuition?



Convergence Analysis for AdaGrad

$$f(x_t) \leq f(x^*) + \frac{1}{\eta}\left(D_{\phi_t}\left(x^* \| x_t\right) - D_{\phi_t}\left(x^* \| x_{t+1}\right) + D_{\phi_t}\left(x_{t+1} \| x_t\right) \right).$$

By taking the summation of the above over $t = 0, 1, \cdots, T-1$, we have

$$\sum_{t=0}^{T-1} f(x_t) \leq T \cdot f(x^*) + \frac{1}{\eta}\left(\underbrace{\sum_{t=0}^{T-2} \left(D_{\phi_{t+1}}\left(x^* \| x_{t+1}\right) - D_{\phi_t}\left(x^* \| x_{t+1}\right) \right)}_{\text{(A)}} + \underbrace{\sum_{t=0}^{T-1} D_{\phi_t}\left(x_{t+1} \| x_t\right)}_{\text{(B)}} \right).$$

Bounding the "Almost-telescopic" term (A)

Lemma: Let T be arbitrary and assume that $D := \max_{0 \le t \le T} \| X_t - X^* \|_\infty$ is finite.

For any $T \in \mathbb{N}$, the term (A) satisfies the inequality

$$\sum_{t=0}^{T-2} \left(D_{\phi_{t+1}}(X^* \| X_{t+1}) - D_{\phi_t}(X^* \| X_{t+1}) \right) \le \frac{D^2}{2} \| S_{T-1} \|_1$$

Proof: **Step 1:** It is easy to verify that

$$D_{\phi_{t+1}}(X^* \| X_{t+1}) - D_{\phi_t}(X^* \| X_{t+1}) = \frac{1}{2} (X_{t+1} - X^*)^T (M_{t+1} - M_t)(X_{t+1} - X^*)$$

$$\le \frac{1}{2} \| X_{t+1} - X^* \|_\infty^2 \| S_{t+1} - S_t \|_1 \quad \cdots (\quad)$$

$$\le \frac{1}{2} D^2 \cdot \| S_{t+1} - S_t \|_1 \quad \cdots (\quad)$$

(Cont.).

Step2: By the fact that $[S_{t+1}]_i \geq [S_t]_i \geq 0$ for all $i \in \{1, \cdots, d\}$,

then we have $$\|S_{t+1} - S_t\|_1 = \|S_{t+1}\|_1 - \|S_t\|_1,$$

This implies that

$$\sum_{t=0}^{T-2} \left(D_{\phi_{t+1}}(x^* \| x_{t+1}) - D_{\phi_t}(x^* \| x_{t+1}) \right) \leq \frac{D^2}{2} \sum_{t=0}^{T-2} \left(\|S_{t+1}\|_1 - \|S_t\|_1 \right)$$

$$= \frac{D^2}{2} \left(\|S_{T-1}\|_1 - \|S_0\|_1 \right)$$

$$\leq \frac{D^2}{2} \|S_{T-1}\|_1$$

Bounding the summation (B)

Lemma: For any $T \in \mathbb{N}$, we have $\sum_{t=0}^{T-1} D_{\phi_t}(x_t \| x_{t+1}) \le \eta^2 \cdot \|s_{T-1}\|_1$

Pf: Recall that $D_{\phi_t}(x_t \| x_{t+1}) = \frac{1}{2}(x_{t+1} - x_t)^T M_t (x_{t+1} - x_t)$

$$= \frac{\eta^2}{2} \nabla f(x_t)^T M_t^{-1} \nabla f(x_t) \quad \cdots (\quad)$$

By the definition of $M_t := \mathrm{diag}(s_t)$ and $[s_t]_i = \sqrt{\sum_{\tau=0}^{t} [\nabla f(x_\tau)]_i^2}$

$$\nabla f(x_t)^T M_t^{-1} \nabla f(x_t) = \sum_{i=1}^{d} \frac{[\nabla f(x_t)]_i^2}{[s_t]_i} = \sum_{i=1}^{d} \frac{[s_t]_i^2 - [s_{t-1}]_i^2}{[s_t]_i} \le 2 \sum_{i=1}^{d} \frac{[s_t]_i^2 - [s_{t-1}]^2}{[s_t]_i + [s_{t-1}]_i}$$

By taking the summation over $t = 0, \cdots, T-1$,

$$= 2(\|s_t\|_1 - \|s_{t-1}\|_1)$$

we get the desired result.

Bounding the norm $\|s_{T-1}\|_1$

Lemma: $$\|s_{T-1}\|_1 \le \sqrt{n} \cdot \sqrt{\min_{\lambda \in \mathbb{R}^d, \|\lambda\|_1 = d} \sum_{t=0}^{T-1} \nabla f(x_t)^T \operatorname{diag}(\lambda)^{-1} \nabla f(x_t)}$$

Pf: $$\|s_{T-1}\|_1^2 = \left(\sum_{i=1}^{d} \sqrt{\sum_{t=0}^{T-1} [\nabla f(x_t)]_i^2} \right)^2 \quad \cdots \left(\right)$$

Pick any $\lambda \in \mathbb{R}^d$ with $\|\lambda\|_1 = d$

$$= \left(\sum_{i=1}^{d} \left[\sqrt{\lambda_i} \cdot \left(\frac{1}{\sqrt{\lambda_i}} \sqrt{\sum_{t=0}^{T-1} [\nabla f(x_t)]_i^2} \right) \right] \right)^2 \quad \cdots \left(\right)$$

$$\le \left(\sum_{i=1}^{d} \left(\sqrt{\lambda_i}\right)^2 \right) \left(\sum_{i=1}^{d} \left(\frac{1}{\sqrt{\lambda_i}} \cdot \sqrt{\sum_{t=0}^{T-1} [\nabla f(x_t)]_i^2} \right)^2 \right) \quad \cdots \left(\right)$$

$$= n \cdot \left(\sum_{t=0}^{T-1} \nabla f(x_t)^T \operatorname{diag}(\lambda)^{-1} \nabla f(x_t) \right).$$

Pros and Cons of AdaGrad

Pros:

1. No manual tuning of learning rate

2. Memory-efficient

3. Provable regret guarantee

4. Originally designed for “sparse gradient” regime (and hence good for this setting)

Cons:

1. AdaGrad can decrease the learning rate too fast in practice (why?)

2. Sensitive to the initialization (e.g., if $\nabla f(x_0)$ is large, then the learning rate can be get too small very quickly at later iterations)

RMSProp

- RMSProp = **R**oot **M**ean **S**quared **Prop**agation
- Main idea: RMSProp uses a less aggressive learning rate decay by considering the Root Mean Square of recent (stochastic) gradients

RMSProp

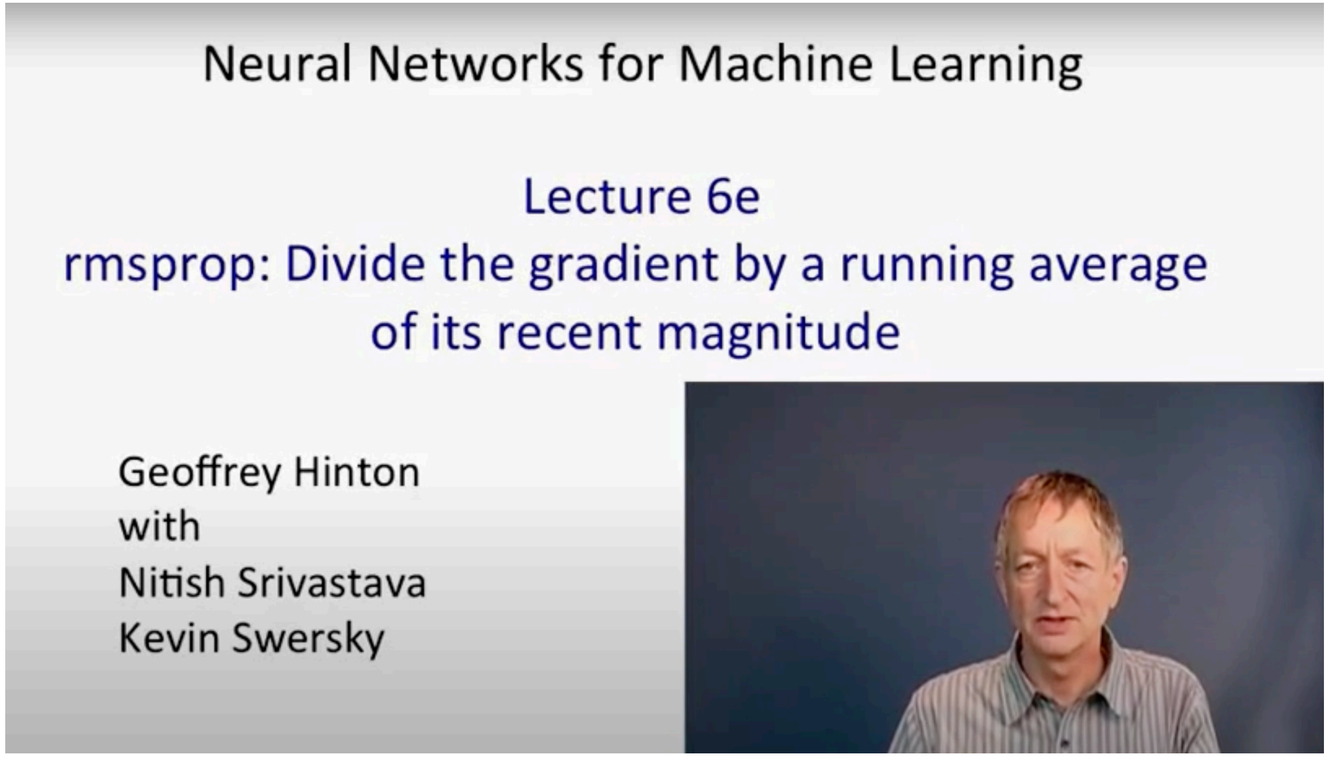
$$x_{t+1} = x_t - \eta M_t^{-1} \nabla f(x_t),$$

where $[s_t]_i^2 = \beta [s_{t-1}]_i^2 + (1-\beta)[\nabla f(x_t)]_i^2$

$$M_t = \text{diag}([s_t])$$

- Remark: Now $[s_t]_i$ is no longer monotonically increasing

RMSProp



- Interestingly, RMSProp has never been formally published in any conference or journal
- In fact, RMSProp came from a Coursera lecture by Jeff Hinton!

Lecture Video: https://www.youtube.com/watch?v=defQQqkXEfE

Jeff Hinton's slides: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Adadelta Optimizer

- Adadelta = **Ada**ptive with Unit Correction via Δx_t
- Main idea: Adadelta enhances AdaGrad by incorporating two techniques

1. Use exponential moving averages to estimate M_t (same as RMSProp)
2. Apply exponential moving averages of Δx_t for unit correction

Adadelta

$$x_{t+1} = x_t - \eta M_t^{-1} \nabla f(x_t),$$

where $[s_t]_i^2 = \beta [s_{t-1}]_i^2 + (1-\beta)[\nabla f(x_t)]_i^2$

$$[\Delta_t]_i^2 = \beta [\Delta_{t-1}]_i^2 + (1-\beta)[(x_t - x_{t-1})]_i^2$$

$$M_t = \text{diag}([s_t / \Delta_t])$$

Matthew Zieler, “An Adaptive Learning Rate Method,” arXiv 2012.

Adam Optimizer

- Adam = **Ada**ptive **M**oment Estimation
- Main idea: Adam combines the adaptive learning rate of AdaGrad and RMSProp with the idea of momentum
  1. Use exponential moving averages to estimate gradients (aka momentum)
  2. Use exponential moving averages to estimate $\text{diag}(s_t)$

Adam

$$x_{t+1} = x_t - \eta M_t^{-1} g_t,$$

where $g_t = \gamma g_{t-1} + (1-\gamma)\nabla f(x_t)$

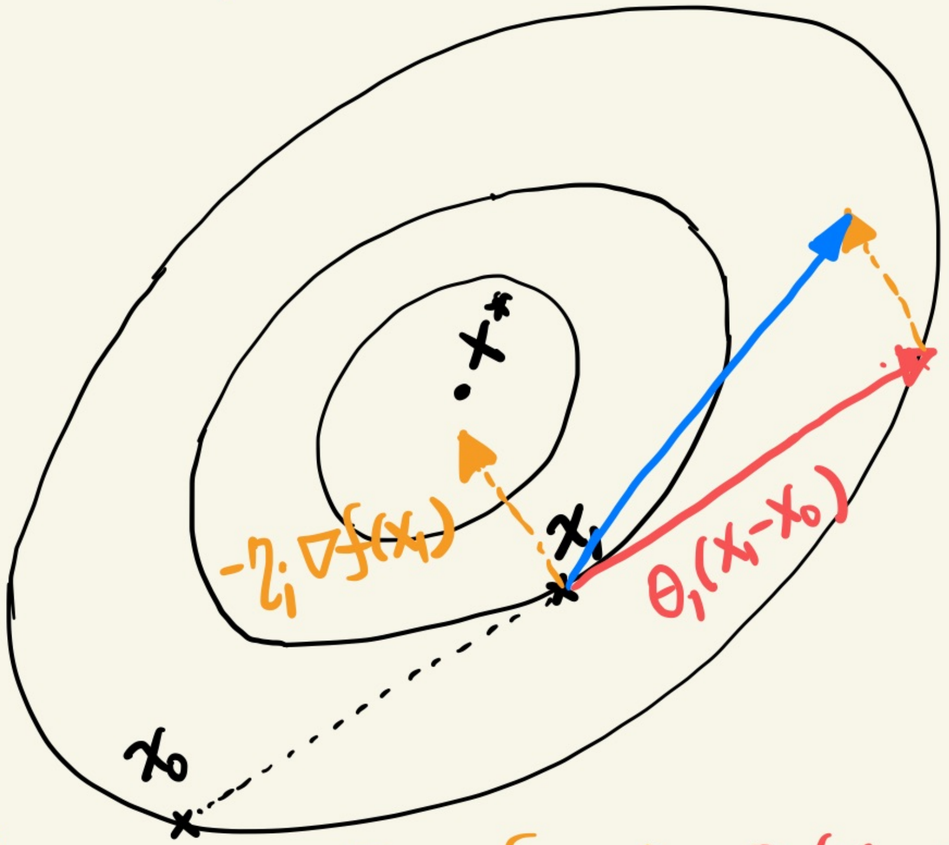
$$[s_t]_i^2 = \beta [s_{t-1}]_i^2 + (1-\beta)[\nabla f(x_t)]_i^2$$

$$M_t = \text{diag}([s_t]) + \varepsilon \mathbb{I}_d$$

- Remark: Typically, $\gamma = 0.9$ and $\beta = 0.999$ (e.g., PyTorch)
- Question: What method does Adam reduce to if $\gamma = 0$?

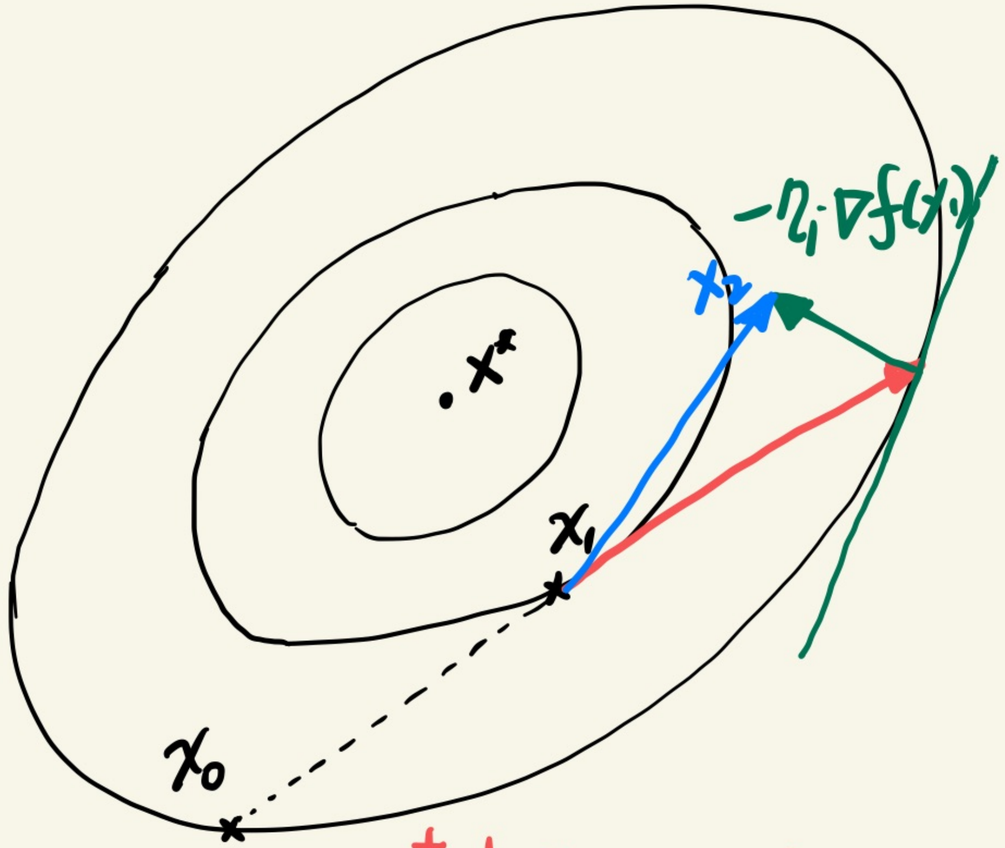
Visualization: Polyak's Momentum vs Nesterov's Method

Polyak's Momentum



$$x_{t+1} = x_t - \eta_t \nabla f(x_t) + \theta_t (x_t - x_{t-1})$$

Nesterov's Method



$$x_{t+1} = \underbrace{x_t + \frac{t-1}{t+2}(x_t - x_{t-1})}_{=y_t} - \eta_t \nabla f(y_t)$$

Adam Optimizer

ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION

Diederik P. Kingma*
University of Amsterdam, OpenAI
dpkingma@openai.com

Jimmy Lei Ba*
University of Toronto
jimmy@psi.utoronto.ca

(ICLR 2015)

ABSTRACT

We introduce *Adam*, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning. Some connections to related algorithms, on which *Adam* was inspired, are discussed. We also analyze the theoretical convergence properties of the algorithm and provide a regret bound on the convergence rate that is comparable to the best known results under the online convex optimization framework. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods. Finally, we discuss *AdaMax*, a variant of *Adam* based on the infinity norm.

Diederik Kingma
(Anthropic)

Jimmy Ba
(University of Toronto)

This work has been cited for more than 200,000 times

Issue #1: Adam Optimizer with Initialization Bias Correction

- Adam usually initializes $g_0 = 0$ and $s_0 = 0$
- This can cause a little bias due to the exponential moving average

Adam with bias correction

$$x_{t+1} = x_t - \eta M_t^{-1} g_t,$$

where $g_t = \gamma g_{t-1} + (1-\gamma)\nabla f(x_t)$

$$[s_t]_i^2 = \beta [s_{t-1}]_i^2 + (1-\beta)[\nabla f(x_t)]_i^2$$

$$g_t \leftarrow g_t / (1-\gamma^t)$$

$$s_t \leftarrow s_t / (1-\beta^t)$$

$$M_t = \text{diag}([s_t])$$

Issue #2: Adam Can Have "Non-Monotonic" Learning Rates

Adam

$$x_{t+1} = x_t - \eta M_t^{-1} g_t,$$

where $g_t = \gamma g_{t-1} + (1-\gamma)\nabla f(x_t)$

$$[s_t]_i^2 = \beta [s_{t-1}]_i^2 + (1-\beta)[\nabla f(x_t)]_i^2$$

$$M_t = \text{diag}([s_t])$$

- **Question**: Does Adam always have "non-increasing" learning rates? (i.e., the learning rate strictly decays?)

- **Question**: How about AdaGrad and RMSProp?

Adam Can Possibly Suffer From Linear Regret

ON THE CONVERGENCE OF ADAM AND BEYOND

Sashank J. Reddi, Satyen Kale & Sanjiv Kumar
Google New York
New York, NY 10011, USA
{sashank,satyenkale,sanjivk}@google.com

ABSTRACT

Several recently proposed stochastic optimization methods that have been successfully used in training deep networks such as RMSPROP, ADAM, ADADELTA, NADAM are based on using gradient updates scaled by square roots of exponential moving averages of squared past gradients. In many applications, e.g. learning with large output spaces, it has been empirically observed that these algorithms fail to converge to an optimal solution (or a critical point in nonconvex settings). We show that one cause for such failures is the exponential moving average used in the algorithms. We provide an explicit example of a simple convex optimization setting where ADAM does not converge to the optimal solution, and describe the precise problems with the previous analysis of ADAM algorithm. Our analysis suggests that the convergence issues can be fixed by endowing such algorithms with "long-term memory" of past gradients, and propose new variants of the ADAM algorithm which not only fix the convergence issues but often also lead to improved empirical performance.

- $[s_t]_i^2 = \beta[s_{t-1}]_i^2 + (1-\beta)[\nabla f(x_t)]_i^2$ is not monotonically increasing with t
- One can construct an online learning problem such that Adam can have a regret $\Theta(\sqrt{T})$

Reddi et al., "On the convergence of Adam and beyond," ICLR 2018.

AMSGrad: Adam With Monotonic Learning Rates

Adam with monotonicity

$$x_{t+1} = x_t - \eta M_t^{-1} g_t,$$

where $g_t = \gamma g_{t-1} + (1-\gamma)\nabla f(x_t)$

$$[s_t]_i^2 = \beta [s_{t-1}]_i^2 + (1-\beta)[\nabla f(x_t)]_i^2$$

$$[s_t]_i \leftarrow \max\{[s_t]_i, [s_{t-1}]_i\}$$

$$M_t = \text{diag}([s_t])$$

- **Intuition**: Use a simple max operation to ensure monotonicity in $[s_t]_i$

Reddi et al., “On the convergence of Adam and beyond,” ICLR 2018.

PyTorch Implementation of Adam Optimizer

```
[docs]class Adam(Optimizer):
    def __init__(
        self,
        params: ParamsT,
        lr: Union[float, Tensor] = 1e-3,
        betas: Tuple[float, float] = (0.9, 0.999),
        eps: float = 1e-8,
        weight_decay: float = 0,
        amsgrad: bool = False,
        *,
        foreach: Optional[bool] = None,
        maximize: bool = False,
        capturable: bool = False,
        differentiable: bool = False,
        fused: Optional[bool] = None,
    ):
        if isinstance(lr, Tensor):
            if foreach and not capturable:
                raise ValueError(
                    "lr as a Tensor is not supported for capturable=False and
foreach=True"
                )
            if lr.numel() != 1:
                raise ValueError("Tensor lr must be 1-element")
        if not 0.0 <= lr:
            raise ValueError(f"Invalid learning rate: {lr}")
        if not 0.0 <= eps:
            raise ValueError(f"Invalid epsilon value: {eps}")
        if not 0.0 <= betas[0] < 1.0:
            raise ValueError(f"Invalid beta parameter at index 0: {betas[0]}")
        if not 0.0 <= betas[1] < 1.0:
            raise ValueError(f"Invalid beta parameter at index 1: {betas[1]}")
        if not 0.0 <= weight_decay:
            raise ValueError(f"Invalid weight_decay value: {weight_decay}")

        defaults = dict(
            lr=lr,
            betas=betas,
            eps=eps,
            weight_decay=weight_decay,
            amsgrad=amsgrad,
            maximize=maximize,
            foreach=foreach,
            capturable=capturable,
            differentiable=differentiable,
            fused=fused,
        )
```

- A quite involved implementation (~1000 lines of python code)
- Can you identify some practical considerations not mentioned here? (Hint: “AMSGrad”)

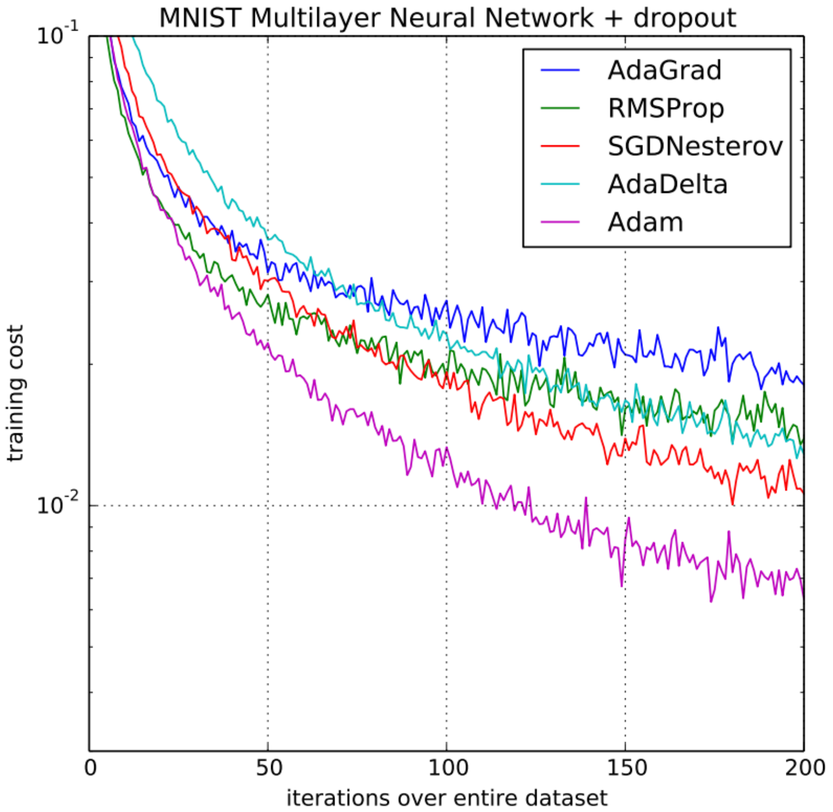
https://pytorch.org/docs/stable/_modules/torch/optim/adam.html#Adam

Some Remarks on Adam

1. Adam enjoys a nice $R(T) := \sum_{t=1}^{T} (f(x_t) - f(x^*)) = O(\sqrt{T})$ regret bound in the convex setting (similar to AdaGrad)

2. Adam is widely used mainly due to its strong empirical convergence and a fairly low complexity

3. Adam can also be combined with Nesterov momentum (and hence called "**Nadam**")



Timothy Dozat, "Incorporating Nesterov Momentum into Adam," ICLR 2016 Workshop.

Figure source: Original ADAM paper

A General Form of Adaptive First-Order Methods

Adaptive First-Order Method

$$x_{t+1} = x_t - \eta V_t^{-1/2} m_t,$$

where

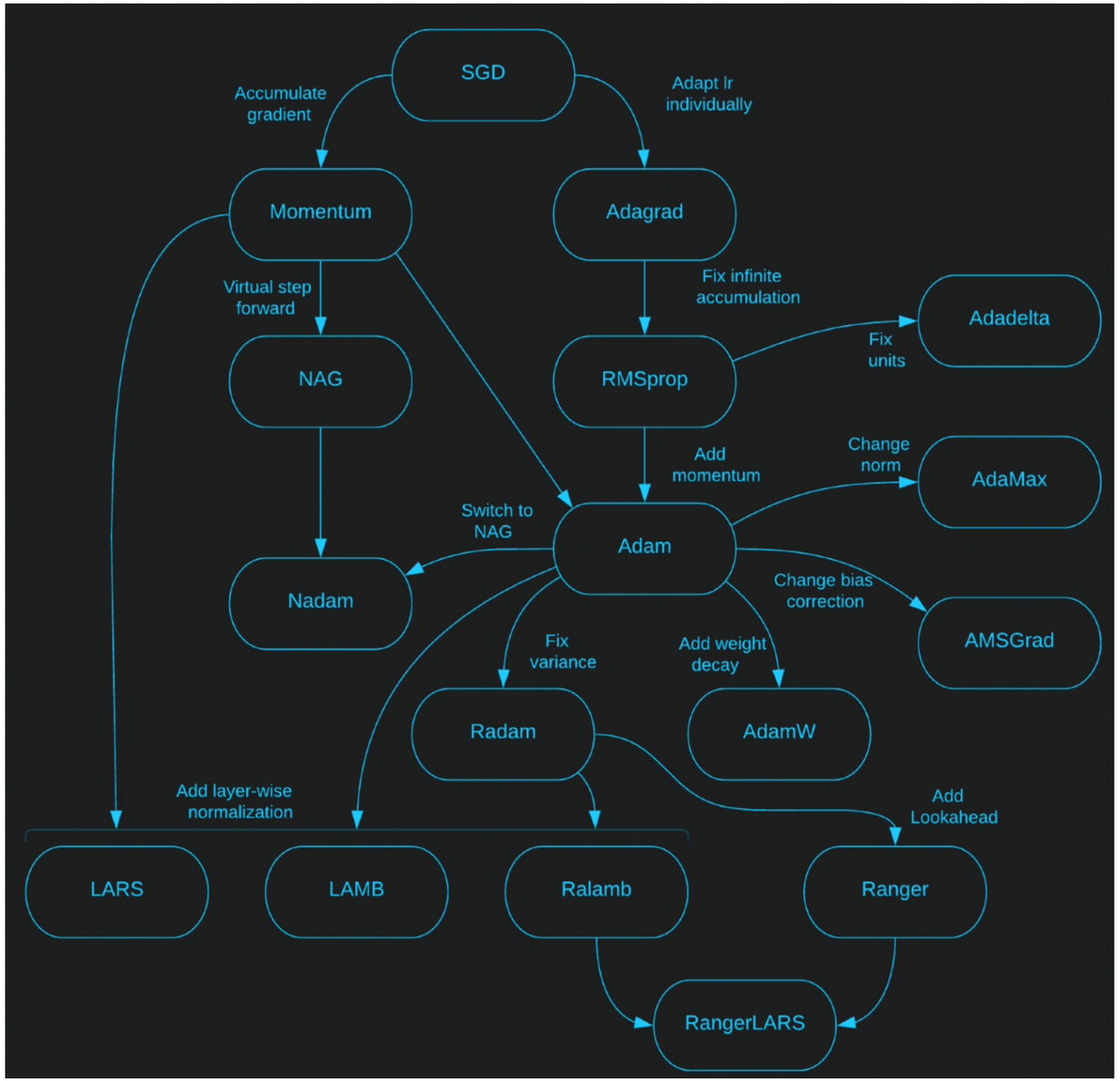
(1) $g_t \leftarrow \nabla f(x_t)$ or $\hat{\nabla} f(x_t)$

(2) $m_t = \phi(g_1, \cdots, g_t)$

(3) $V_t = \psi(g_1, \cdots, g_t)$

- What are the ϕ, ψ of GD (and SGD)?
- What are the ϕ, ψ of AdaGrad?
- What are the ϕ, ψ of Adam?

Summary: Adaptive First-Order Optimizers



- AdaGrad (2011)
- RMSProp (2012)
- Adadelta (2012)
- Adam (2015)
- Nadam (2016)
- AMSGrad (2018)

Figure Source: https://darel13712.github.io/ml/optimizers.html