



AI and Deep Learning

2-5 Gradient Descent Algorithms and Their Variations

Zhonglei Wang

WISE, SOE, CHOW

XMU

(Version v1)

Contents

1. Gradient descent algorithms

2. Variation

3. Learning rate decay

Recall: Batch gradient descent algorithm

Step 1. Randomly initialize $\boldsymbol{\theta}^{(0)}$

Step 2. Based on the current parameter $\boldsymbol{\theta}^{(t)}$, obtain

$$\nabla \mathcal{J}(\boldsymbol{\theta}^{(t)}) = \frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}^{(t)}) = n^{-1} \sum_{i=1}^n \frac{\partial \mathcal{L}_i}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}^{(t)})$$

- $\mathcal{L}_i$: loss function associated with the i th training example

Step 3. Update parameter

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \cdot \nabla \mathcal{J}(\boldsymbol{\theta}^{(t)})$$

Step 4. Go back to Step 2 until convergence, or reaching maximum number of iterations

Recall: Batch gradient descent algorithm

1. Advantages

- “True gradient” is used

2. Disadvantages

- Computationally inefficient when n is large
- Severe memory consumption

3. Why not use part of the training examples for each iteration?

Mini-batch gradient descent

1. Idea: Use part of training examples for each iteration
2. Partition the index set of training examples: $\{1, \dots, n\} = S_1 \cup \dots \cup S_k$
 - $|S_1| = \dots = |S_{k-1}| = m$
 - $|S_k| \leq m$
 - $k = \lceil n/m \rceil$
 - m is usually a power of 2 to achieve better performance for GPU.
 - ▷ For example, $m = 32$

Mini-batch gradient descent

1. For the t th iteration, update the model parameter by

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \cdot \nabla \mathcal{J}_t(\boldsymbol{\theta}^{(t)}),$$

$$\nabla \mathcal{J}_t(\boldsymbol{\theta}^{(t)}) = |S_{t \% k + 1}|^{-1} \sum_{i \in S_{t \% k + 1}} \frac{\partial \mathcal{L}_i}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}^{(t)})$$

- Only use training examples in $S_{t \% k + 1}$ to obtain the gradients

2. Two special cases

- $m = 1$: Stochastic gradient descent
- $m = n$: Batch gradient descent

3. We finish one epoch when each training example is used

Remarks on stochastic gradient descent

1. Disadvantages:

- Noisy estimation for the batch gradient

2. Advantages:

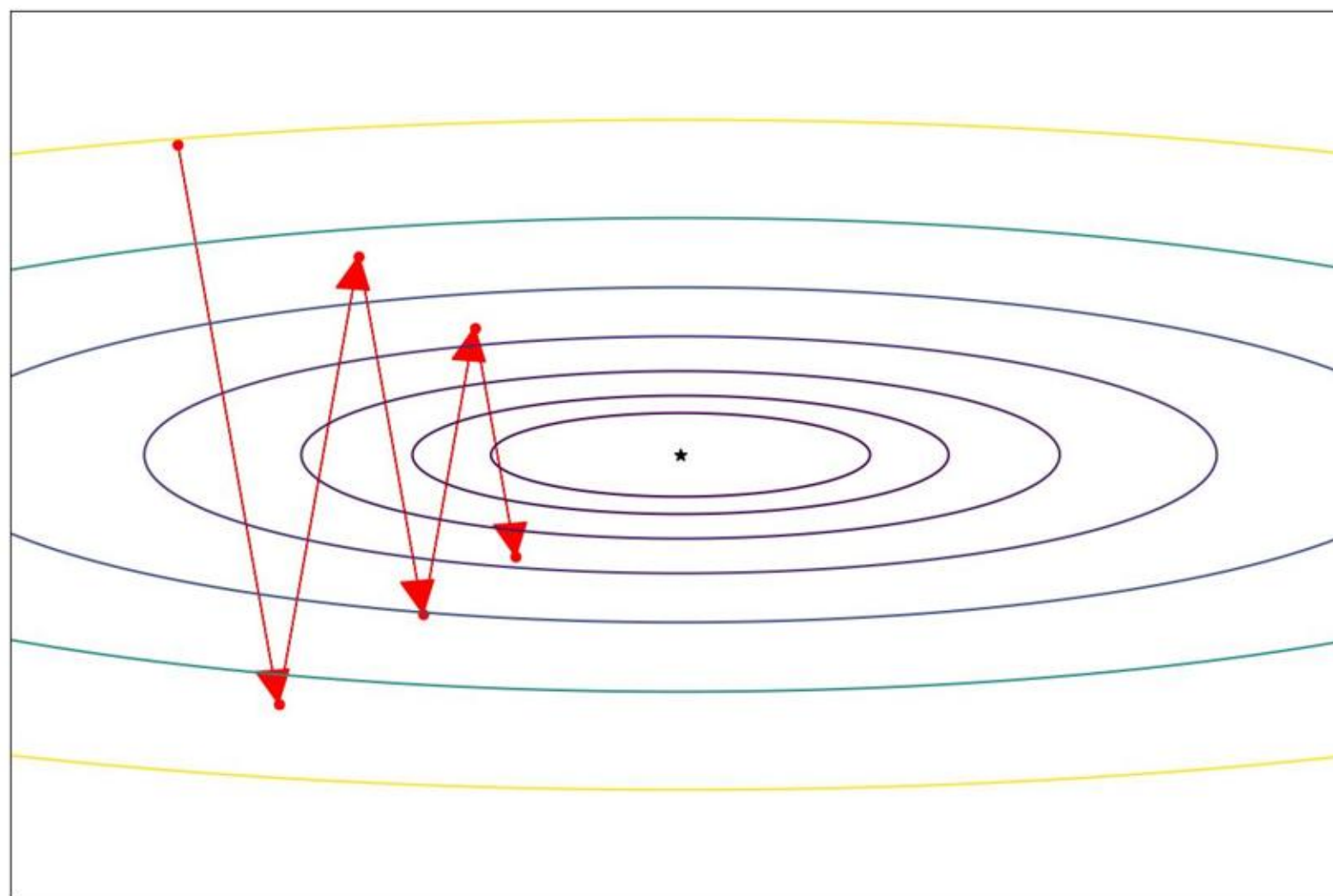
- Noisy estimation is not so bad, since it helps to avoid local minima or saddle points
- We need to adjust the learning rate to achieve convergence
- Computational efficiency is very high
- Suitable for online learning

Discussion

1. Mini-batch gradient descent sacrifices accuracy when $m < n$
2. Nevertheless, it saves memory and is computationally more efficient
3. Recommendation: $m \in \{2, 4, 8, 16, 32\}$ (Masters and Luschi, 2018)
4. Commonly used in practice
5. Question: Why batch size m is not always better when larger?

Background

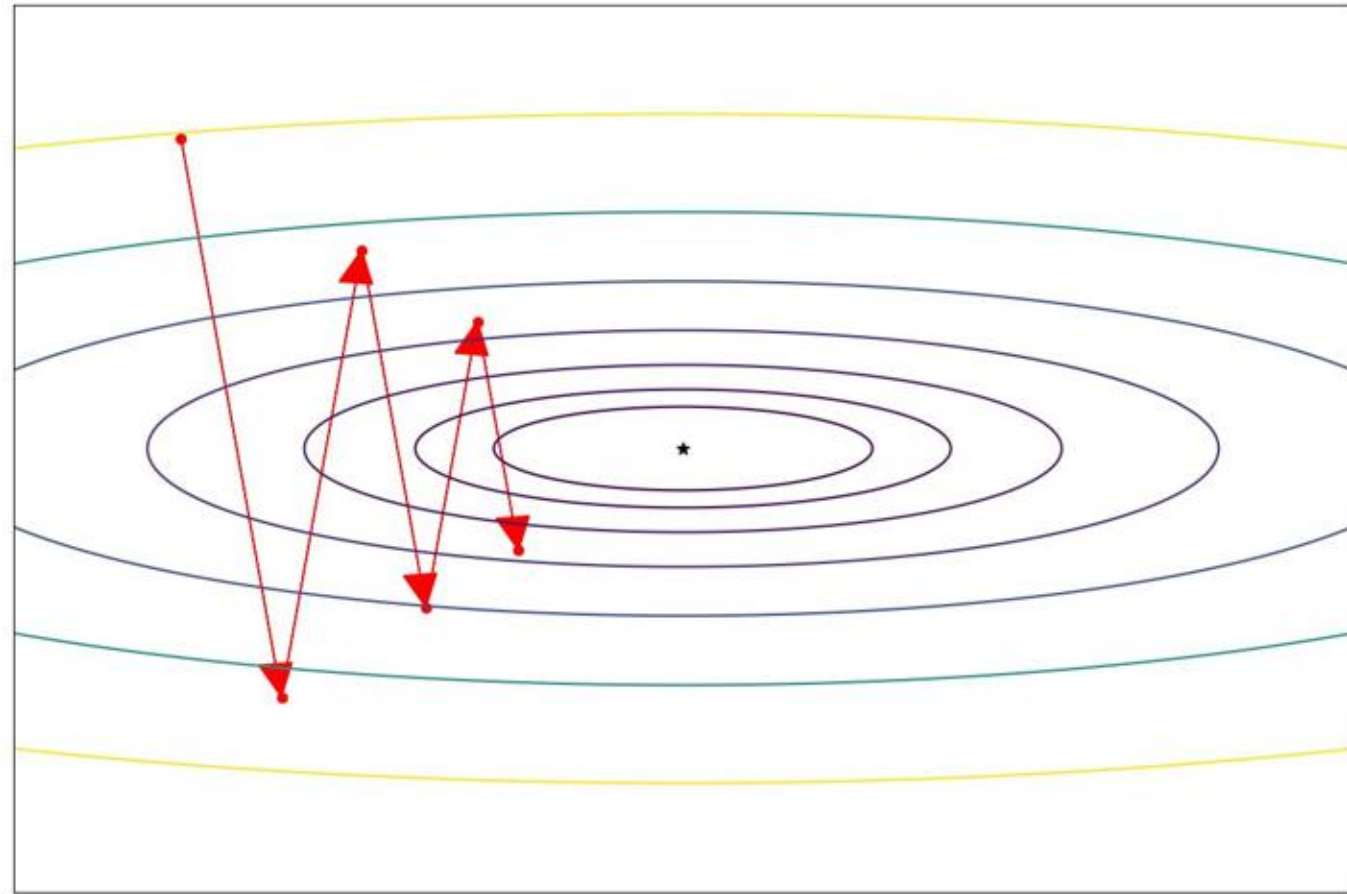
1. Gradient shows the fastest direction along which the cost function increases
2. It may causes problems, especially when the cost function is unstable



Background

1. For gradient descent algorithms, replace gradients by possibly more efficient vectors
2. To achieve this goal, we introduce
 - Momentum: Smooth gradients by taking “average”
 - RMSprop: Adaptively modify learning rates for each model parameter
 - Adam: Integrate advantages of momentum and RMSprop

Momentum



1. Ideally, we want to

- Decrease the up-down effect
- Increase the left-right effect

Momentum

1. Consider **EWMA** (Exponential Weighted Moving Average)

- Original sequence: $\{\mathbf{g}^{(t)} : t = 1, 2, \dots\}$
 - ▷ It can be $d\mathbf{b}^{[l](t)}$ or $d\mathbf{W}^{[l](t)}$ evaluated at the current parameter $\boldsymbol{\theta}^{(t)}$
- EWMA sequence: $\{\mathbf{v}^{(t)} : t = 1, 2, \dots\}$
$$\mathbf{v}^{(t)} = \beta_1 \cdot \mathbf{v}^{(t-1)} + (1 - \beta_1) \cdot \mathbf{g}^{(t)} \quad (t = 1, 2, \dots)$$
 - ▷ $\mathbf{v}^{(0)} = 0$
 - ▷ $\beta_1 \in (0, 1)$: Controls how much we stay with the “momentum” $\mathbf{v}^{(t-1)}$
 - ▷ $\beta_1 = 0.9$: Hyperparameter, but seldom tuned

Remark

1. A fact: For $\beta_1 \in (0, 1)$, we have

$$\frac{1}{(1 - \beta_1)^{-1}} \sum_{i=0}^{\infty} \beta_1^i = 1$$

2. More details

$$\begin{aligned} \mathbf{v}^{(t)} &= \beta_1 \cdot \mathbf{v}^{(t-1)} + (1 - \beta_1) \cdot \mathbf{g}^{(t)} \\ &= (1 - \beta_1) \cdot \beta_1^{t-1} \cdot \mathbf{g}^{(1)} + (1 - \beta_1) \cdot \beta_1^{t-2} \cdot \mathbf{g}^{(2)} + \cdots + (1 - \beta_1) \cdot \mathbf{g}^{(t)} \\ &= \frac{1}{(1 - \beta_1)^{-1}} \sum_{i=1}^t \beta_1^{t-i} \cdot \mathbf{g}^{(i)} \end{aligned}$$

- When t is large, it is approximately weighted average
- $(1 - \beta_1)^{-1}$: “Effective sample size” for EMWA

Momentum-based gradient descent algorithm

Step 1. Randomly initialize $\theta^{(0)}$

Step 2. Based on the current model parameter $\theta^{(t)}$, obtain $\text{d}\mathbf{b}^{[1](t)}$

- Take the update procedure for $\mathbf{b}^{[1]}$ as an example
- The procedure applies to other parameters as well

Step 3. Update

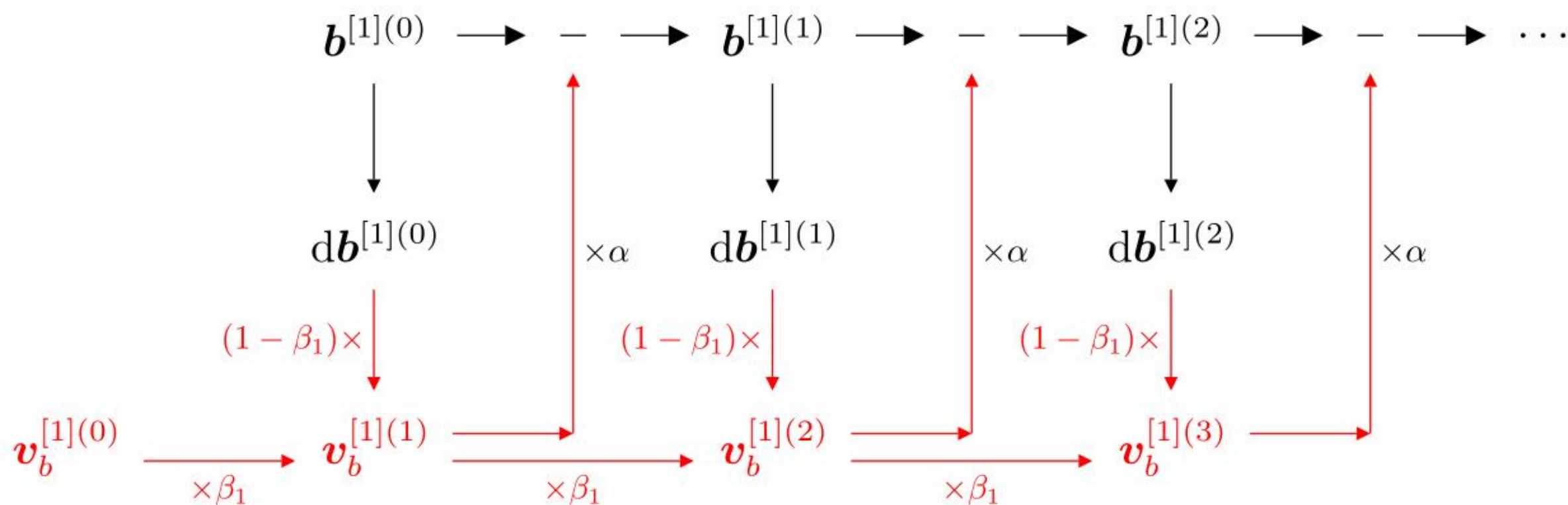
$$\mathbf{b}^{[1](t+1)} = \mathbf{b}^{[1](t)} - \alpha \cdot \mathbf{v}_b^{[1](t+1)}$$

- $\mathbf{v}_b^{[1](t+1)} = \beta_1 \cdot \mathbf{v}_b^{[1](t)} + (1 - \beta_1) \cdot \text{d}\mathbf{b}^{[1](t)}$
- $\mathbf{v}_b^{[1](0)} = 0$
- $\beta_1 = 0.9$ by default

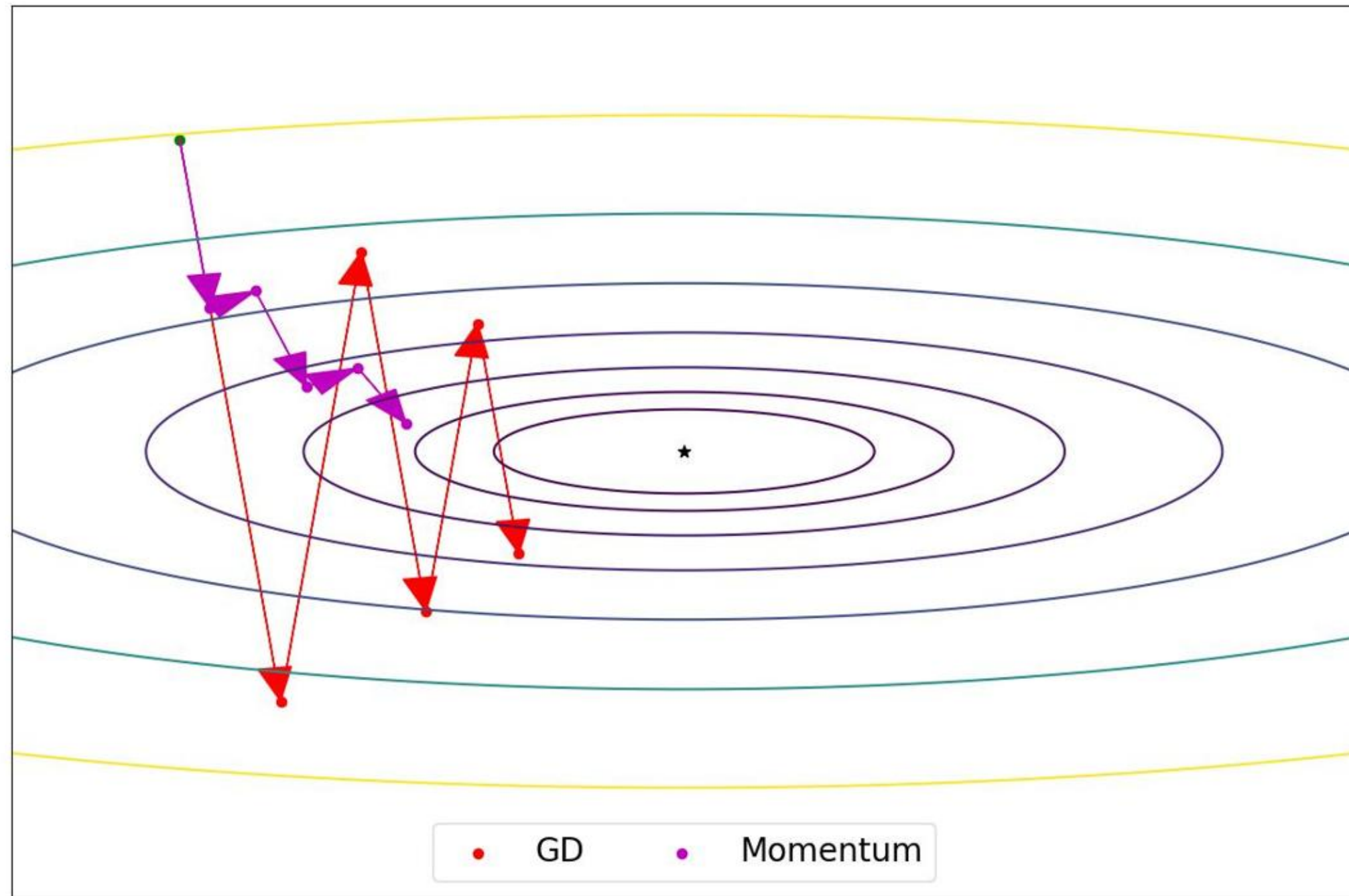
Step 4. Go back to Step 2 until convergence, or reaching maximum number of iterations

Algorithm demonstration

1. We demonstrate momentum-based GD algorithm for $\mathbf{b}^{[1]}$
2. Other model parameters are updated in the same way at the same iteration

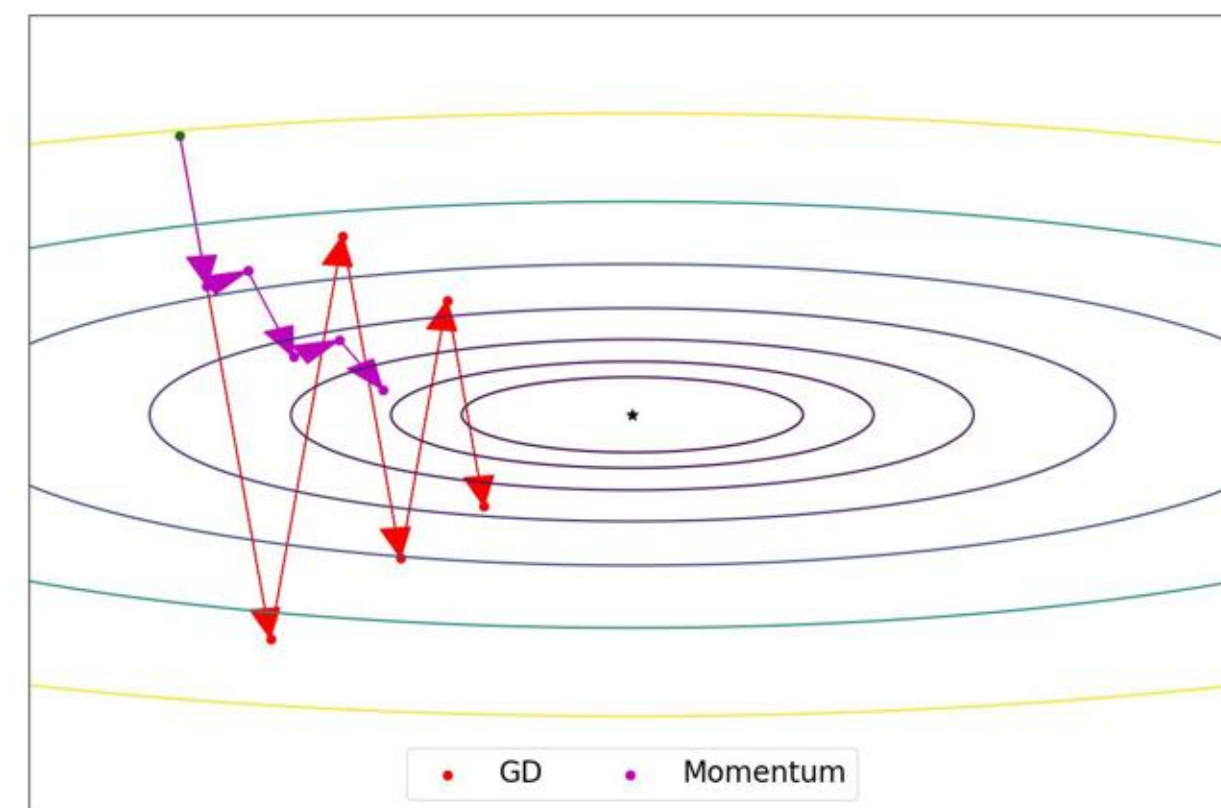


Comparison



Comparison

1. Momentum indeed decreases the variability in the up-down direction
2. Generally, given a learning rate, momentum performs better than gradient descent algorithms
3. One possible solution: Different learning rates for different directions
 - **Low** learning rate for the **up-down** direction
 - **High** learning rate for the **left-right** direction



RMSprop-based gradient descent algorithm

Step 1. Randomly initialize $\theta^{(0)}$

Step 2. Based on the current model parameter $\theta^{(t)}$, obtain $\text{d}\mathbf{b}^{[1](t)}$

- Take the update procedure for $\mathbf{b}^{[1]}$ as an example

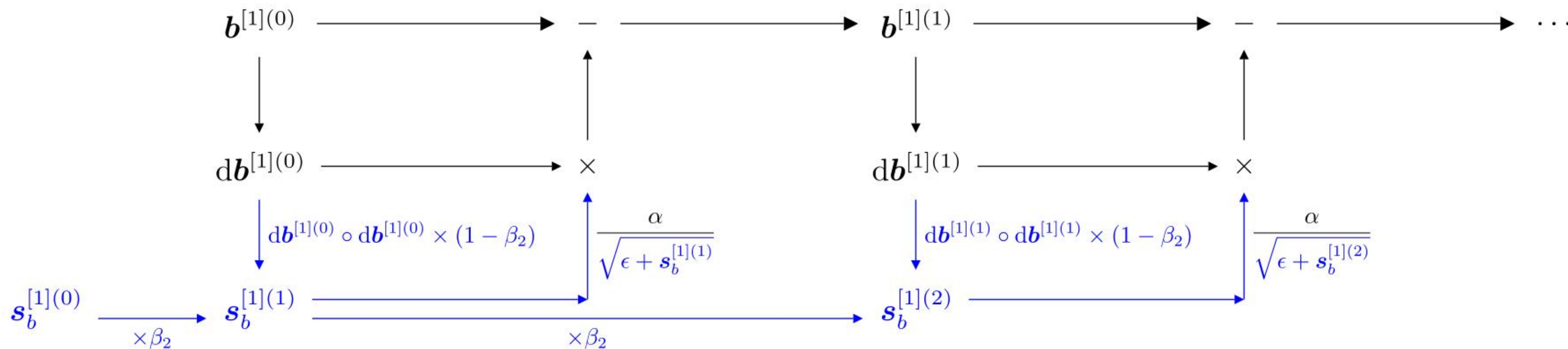
Step 3. Update

$$\mathbf{b}^{[1](t+1)} = \mathbf{b}^{[1](t)} - \frac{\alpha}{\sqrt{\epsilon + \mathbf{s}_b^{[1](t+1)}}} \cdot \text{d}\mathbf{b}^{[1](t)}$$

- $\epsilon = 10^{-8}$ by default
- $\mathbf{s}_b^{[1](t+1)} = \beta_2 \cdot \mathbf{s}_b^{[1](t)} + (1 - \beta_2) \cdot \text{d}\mathbf{b}^{[1](t)} \circ \text{d}\mathbf{b}^{[1](t)}$
- $\mathbf{s}_b^{[1](0)} = 0$
- $\beta_2 = 0.99$ by default

Algorithm demonstration

1. We demonstrate RMSprop-based GD algorithm for $\mathbf{b}^{[1]}$
2. Other model parameters are updated in the same way at the same iteration



Algorithms

1. Classical gradient descent algorithms

$$\mathbf{b}^{[1](t+1)} = \mathbf{b}^{[1](t)} - \alpha \cdot d\mathbf{b}^{[1](t)}$$

2. Momentum only modifies the gradient part

$$\mathbf{b}^{[1](t+1)} = \mathbf{b}^{[1](t)} - \alpha \cdot \mathbf{v}_b^{[1](t+1)}$$

3. RMSprop only modifies the learning rate part

$$\mathbf{b}^{[1](t+1)} = \mathbf{b}^{[1](t)} - \frac{\alpha}{\sqrt{\epsilon + \mathbf{s}_b^{[1](t+1)}}} \cdot d\mathbf{b}^{[1](t)}$$

4. Why not combine those two together?

Adam (Adaptive moment estimation)

Step 1. Randomly initialize $\boldsymbol{\theta}^{(0)}$

Step 2. Based on the current model parameter $\boldsymbol{\theta}^{(t)}$, obtain $d\mathbf{b}^{[1](t)}$

- Take the update procedure for $\mathbf{b}^{[1]}$ as an example

Step 3. Update

$$\mathbf{b}^{[1](t+1)} = \mathbf{b}^{[1](t)} - \frac{\alpha}{\epsilon + \sqrt{\hat{\mathbf{s}}_b^{[1](t+1)}}} \cdot \hat{\mathbf{v}}_b^{[1](t+1)}$$

- See next slide for $\hat{\mathbf{s}}_b^{[1](t+1)}$ and $\hat{\mathbf{v}}_b^{[1](t+1)}$
- $\epsilon = 10^{-8}$ by default

Step 4. Go back to Step 2 until convergence, or reaching maximum number of iterations

Adam (Adaptive moment estimation)

1. Computation details for $\hat{\mathbf{s}}_b^{[1](t+1)}$ and $\hat{\mathbf{v}}_b^{[1](t+1)}$

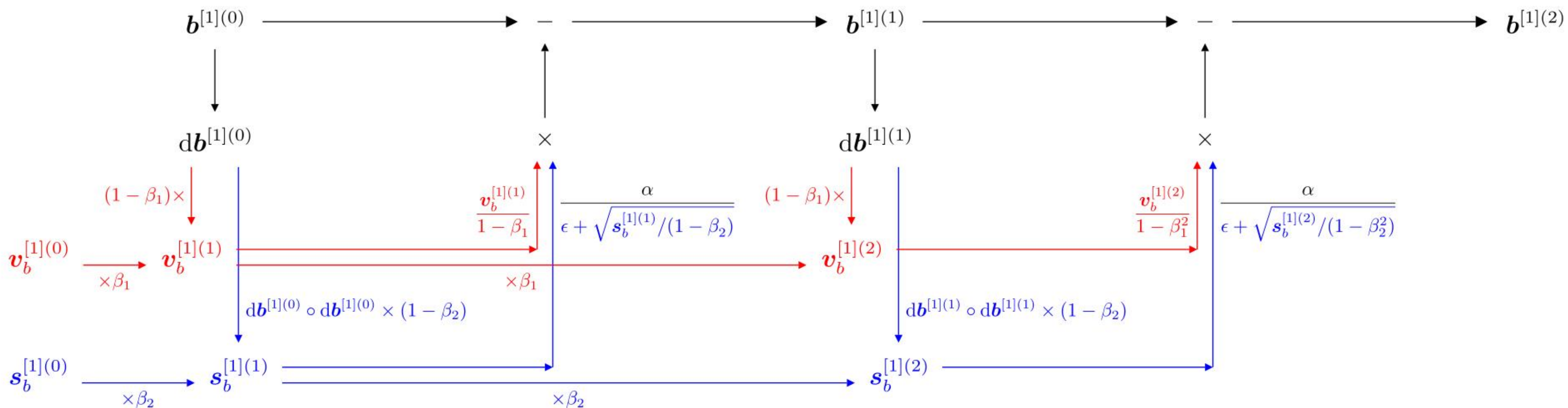
$$\mathbf{s}_b^{[1](t+1)} = \beta_2 \cdot \mathbf{s}_b^{[1](t)} + (1 - \beta_2) \cdot \mathrm{d}\mathbf{b}^{[1](t)} \circ \mathrm{d}\mathbf{b}^{[1](t)}, \quad \hat{\mathbf{s}}_b^{[1](t+1)} = \frac{\mathbf{s}_b^{[1](t+1)}}{1 - \beta_2^{t+1}},$$

$$\mathbf{v}_b^{[1](t+1)} = \beta_1 \cdot \mathbf{v}_b^{[1](t)} + (1 - \beta_1) \cdot \mathrm{d}\mathbf{b}^{[1](t)}, \quad \hat{\mathbf{v}}_b^{[1](t+1)} = \frac{\mathbf{v}_b^{[1](t+1)}}{1 - \beta_1^{t+1}}$$

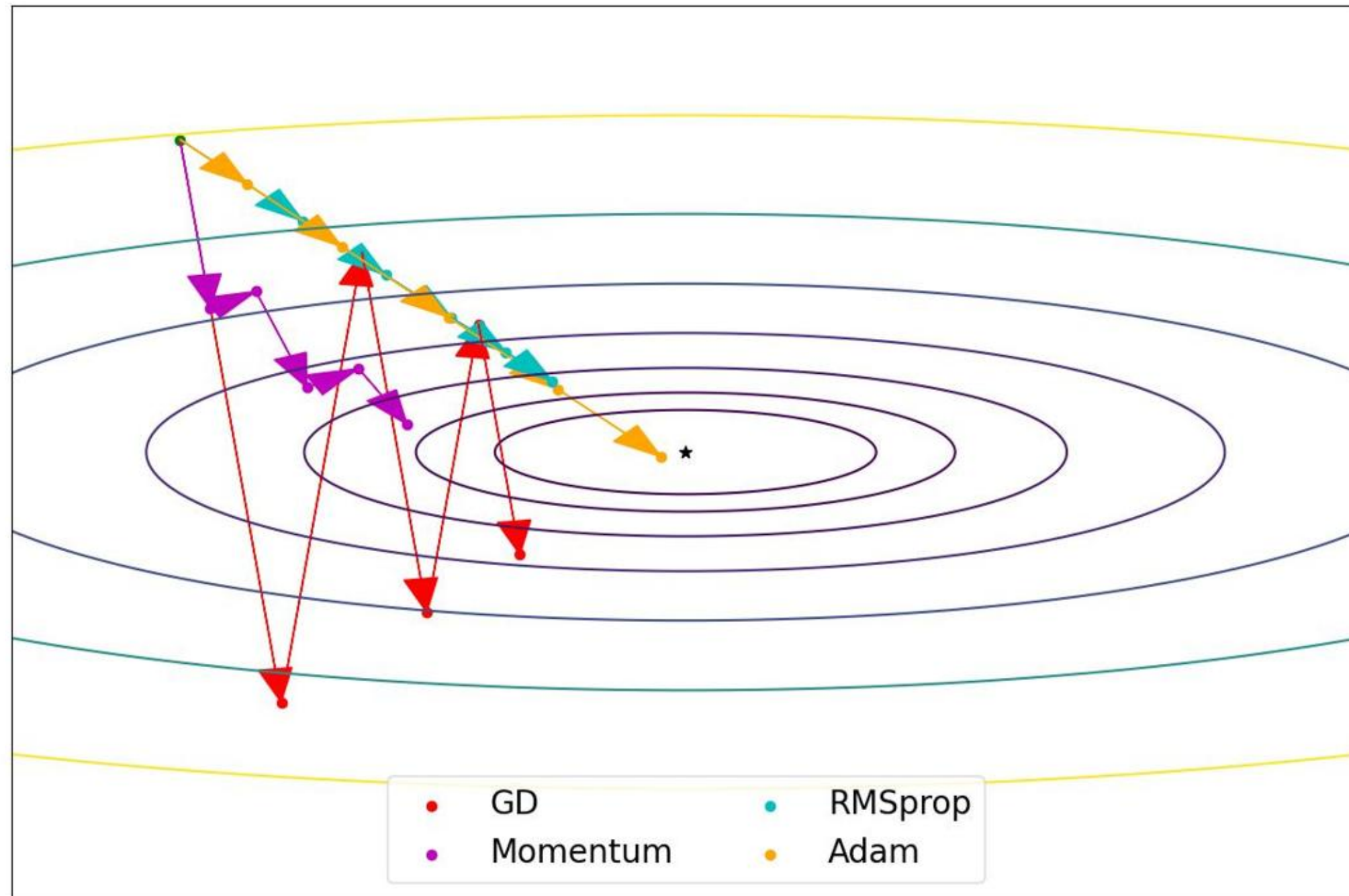
- $\mathbf{v}_b^{[1](0)} = \mathbf{s}_b^{[1](0)} = 0$
- $\beta_1 = 0.9$ by default
- $\beta_2 = 0.99$ by default

Algorithm demonstration

1. We demonstrate Adam-based GD algorithm for $\mathbf{b}^{[1]}$
2. Other model parameters are updated in the same way at the same iteration



Comparison



Comparison

1. Adam (Kingma and Ba, 2014) is commonly used in practice
 - Adam performs well even using default hyperparameters
 - Converge fast
 - Automatically adjust to features with different scales
2. Its variation, AdamW, was proposed by Loshchilov and Hutter (2019)
 - Incorporating weights decay
 - Default optimizor for deep learning models
 - To be discussed later in the section of regularization
3. Adam may fail; see Zaheer et al. (2018) and Reddi et al. (2019) for details

Learning rate decay

1. Intuition

- A large learning rate is needed for initial iterations
- As iteration goes, the parameters should get closer to the theoretical values
- It is inefficient to use the SAME learning rate for all iterations
 - ▷ [Inefficient] Small learning rate guarantees good performance, but it takes longer to get convergence
 - ▷ [Unstable] A large learning rate leads to instability when iteration index is large
- Actually, we should decrease the learning rate in a reasonable manner

Learning rate decay

1. Denote

- t : Iteration index
- $epoch$: Epoch index

2. Several possible ways to decay the learning rate

$$\alpha_t = \frac{\alpha_0}{1 + \gamma \cdot epoch}, \quad \alpha_t = \frac{\alpha_0}{\sqrt{epoch}},$$

$$\alpha_t = 0.95^{epoch} \cdot \alpha_0, \quad \alpha_t = \frac{1}{\sqrt{t}} \cdot \alpha_0,$$

$$\alpha_t = 0.95^t \cdot \alpha_0$$

- $\alpha_0 = 5 \times 10^{-3}$ for example
- $\gamma = 1$ (by default)

Summary

1. Three aspects for model training

- How many samples to use for each update?
- Whether the update vector needs smoothing or scaling?
- Whether the learning rate needs to be adjusted strategically?

2. Start from Adam