SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107
Accredited by NAAC-UGC with 'A' Grade
Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

Department of AI &DS

Course Name – 19AD602 DEEP LEARNING

III Year / VI Semester

UNIT-4 OPTIMIZATION AND GENERALIZATION
Topic: OPTIMIZATION IN DEEP LEARNING

CASE STUDY:

After applying optimization techniques like batch normalization, learning rate scheduling, and the Adam optimizer, training time reduces to 4 hours, accuracy improves to 90%, and the model generalizes better to unseen data.

OPTIMIZATION IN DEEP LEARNING

There are various optimization techniques to change model weights and learning rates, like Gradient Descent, Stochastic Gradient Descent, Stochastic Gradient descent with momentum, Mini-Batch Gradient Descent, AdaGrad, RMSProp, AdaDelta, and Adam. These optimization techniques play a critical role in the training of neural networks, as they help improve the model by adjusting its parameters to minimize the loss of function value. Choosing the best optimizer depends on the application.

1. **The epoch** is the number of times the algorithm iterates over the entire training dataset.
2. **Batch weights** refer to the number of samples used for updating the model parameters.
3. **A sample** is a single record of data in a dataset.
4. **Learning Rate** is a parameter determining the scale of model weight updates
5. **Weights and Bias** are learnable parameters in a model that regulate the signal between two neurons.

Gradient Descent

A derivative or gradient indicates the direction of increase of the function. Thus a negative derivative or gradient would indicate the direction of decrease of the function. This fact is used to minimize the value of the function.

In gradient descent, we initialize the variables with random values.

1. We calculate the derivative/gradient for each variable.
2. We take steps in the direction of the negative derivate/gradient using a learning rate. The learning rate controls the descent. Too large learning rate may result in oscillations while a small learning rate results in slow convergence and hence the optimal value of the learning rate is critical
3. This is iteratively done until we reach a convergence criteria.

1. Formula :

$\theta(k+1)=\theta k-\alpha\nabla J(\theta k)$

where,

- $\theta(k+1)$ is the updated parameter vector at the (k+1)th iteration.
- θk is the current parameter vector at the kth iteration.
- α is the learning rate, which is a positive scalar that determines the step size for each iteration.
- $\nabla J(\theta k)$ is the gradient of the cost or loss function J with respect to the parameters θk

OPTIMIZATION IN DEEP LEARNING

Its a variant of gradient descent in which we ensure that the step size taken is sufficient enough to reduce the objective function thereby avoiding small steps. Here the step size is determined through a line search which must satisfy Armijo condition. Below is the process

1. **Initialization** : We set a initial guess for the function f(x)
2. **Gradient** : We compute the gradient of the objective function ∇f(x)
3. **Line Search** : Here we take a large step size(and check if the reduction in function value

   (using updated value and old value) satisfies below conditions know as Armjio condition ,

$$f(xt-1+\alpha\nabla f(xt-1))-f(xt-1)\geq c\alpha||\nabla f(xt-1)||$$

2

1. Here
   - We are trying to find value at x(t) at time steep t and x(t-1) is the value at step t-1
   - α is the step size
   - c is a constant between 0 to 1.
   - If we do not get the required reduction we reduce the step size by beta β ∈ (0, 1) iteratively till the above condition know as Armjio is satisfied
   - Why this value ? It has been shown mathematically through Taylor series first order expansion that the minimum decrease in f(x) should be at least "step size * ∇f(x)2 ". These theoretical value is not practically possible to achieve that's why we multiply by a fraction c.
2. **Update** : Update the solution parameters with the chosen step size.
3. **Convergence Check**: This can be done by examining the magnitude of the gradient, the change in the objective function value, or other convergence criteria

Gradient descent with Armijo Full Relaxation condition:

It is an optimization algorithm that combines the Armijo line search condition with a full Newton step. It considers both the first derivative and second derivative(Hessian) information to find a step size that ensures sufficient decrease in the objective function while incorporating information about the curvature of the function.

1. **Initialization**
2. **Gradient**
3. **Line Search**

Stochastic Gradient Descent (SGD):

It's a variation of the Gradient Descent algorithm. In Gradient Descent, we analyze the entire dataset in each step, which may not be efficient when dealing with very large datasets. To address this issue, we have Stochastic Gradient Descent (SGD). In Stochastic Gradient Descent, we process just one example at a time to perform a single step. So, if the dataset contains 10000 rows, SGD will update the model parameters 10000 times in a single cycle through the dataset, as opposed to just once in the case of Gradient Descent.

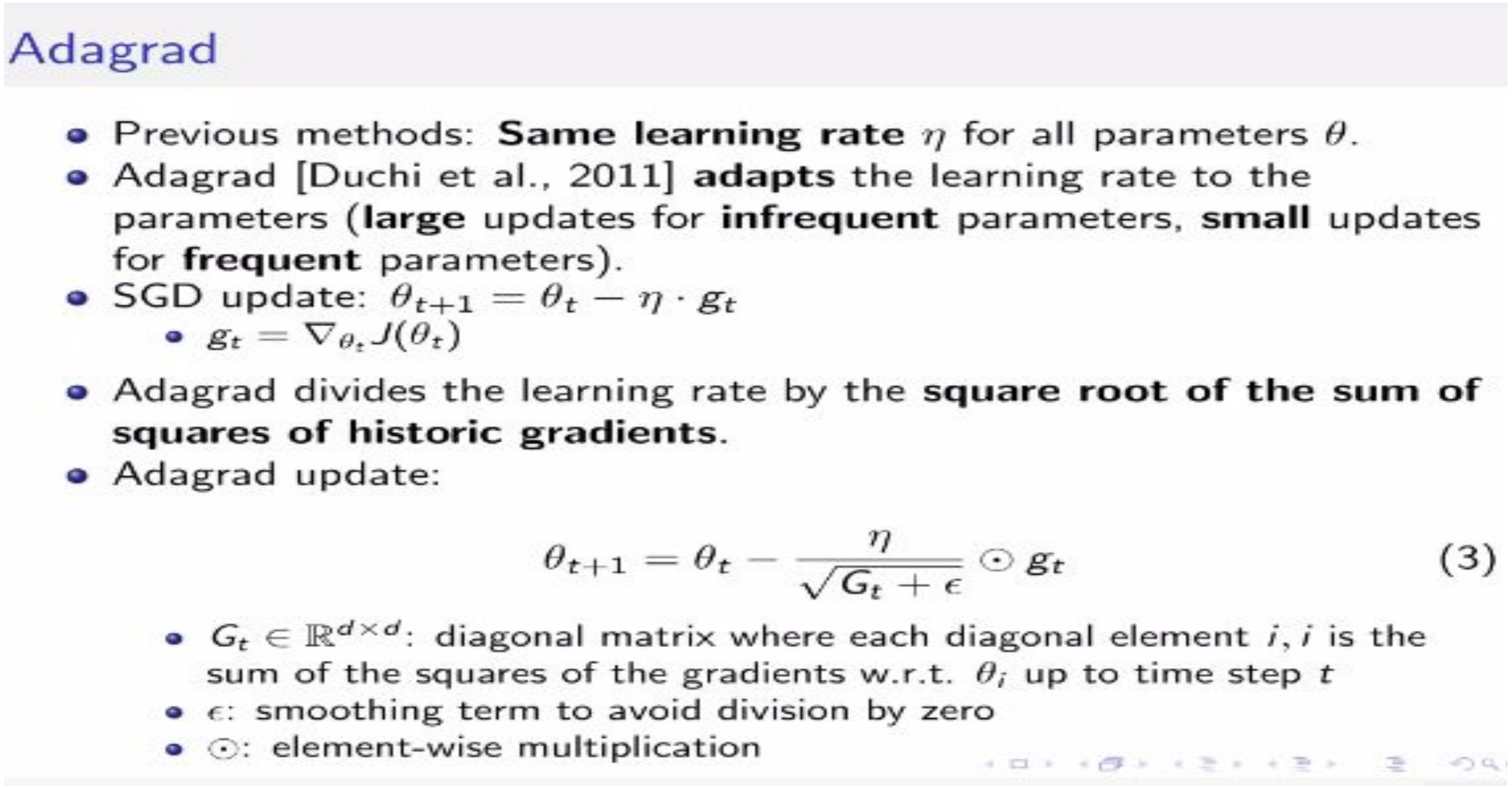
Here's the process:

1. Select an example from the dataset.
2. Calculate its gradient.
3. Utilize the calculated gradient from step 2 to update the model weights.
4. Repeat steps 1 to 3 for all examples in the training dataset.
5. Completing a full pass through all the examples constitutes one epoch.
6. Repeat this entire process for several epochs as specified during training.

Mini Batch Stochastic Gradient Descent:

We utilize a mini-batch stochastic gradient descent, which consists of a predetermined number of training examples, smaller than the full dataset. This approach combines the advantages of the previously mentioned variants. In one epoch, following the creation of fixed-size mini-batches, we execute the following steps:

1. Select a mini-batch.
2. Compute the mean gradient of the mini-batch.
3. Apply the mean gradient obtained in step 2 to update the model's weights.
4. Repeat steps 1 to 2 for all the mini-batches that have been created.

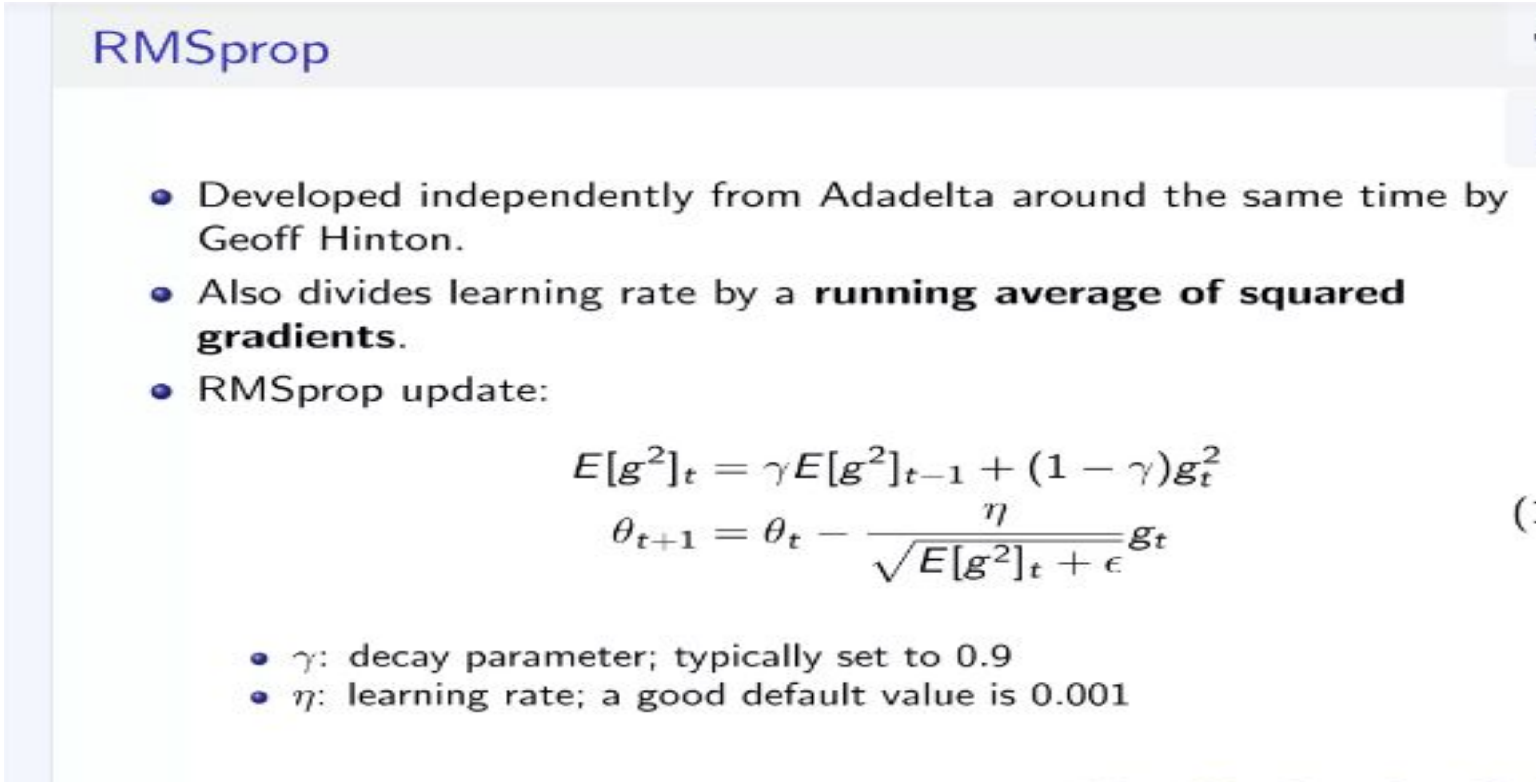


Adagrad

- Previous methods: **Same learning rate** η for all parameters θ.
- Adagrad [Duchi et al., 2011] **adapts** the learning rate to the parameters (**large** updates for **infrequent** parameters, **small** updates for **frequent** parameters).
- SGD update: $\theta_{t+1} = \theta_t - \eta \cdot g_t$
  - $g_t = \nabla_{\theta_t} J(\theta_t)$
- Adagrad divides the learning rate by the **square root of the sum of squares of historic gradients**.
- Adagrad update:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \tag{3}$$

  - $G_t \in \mathbb{R}^{d \times d}$: diagonal matrix where each diagonal element i, i is the sum of the squares of the gradients w.r.t. θ_i up to time step t
  - ϵ: smoothing term to avoid division by zero
  - $\odot$: element-wise multiplication

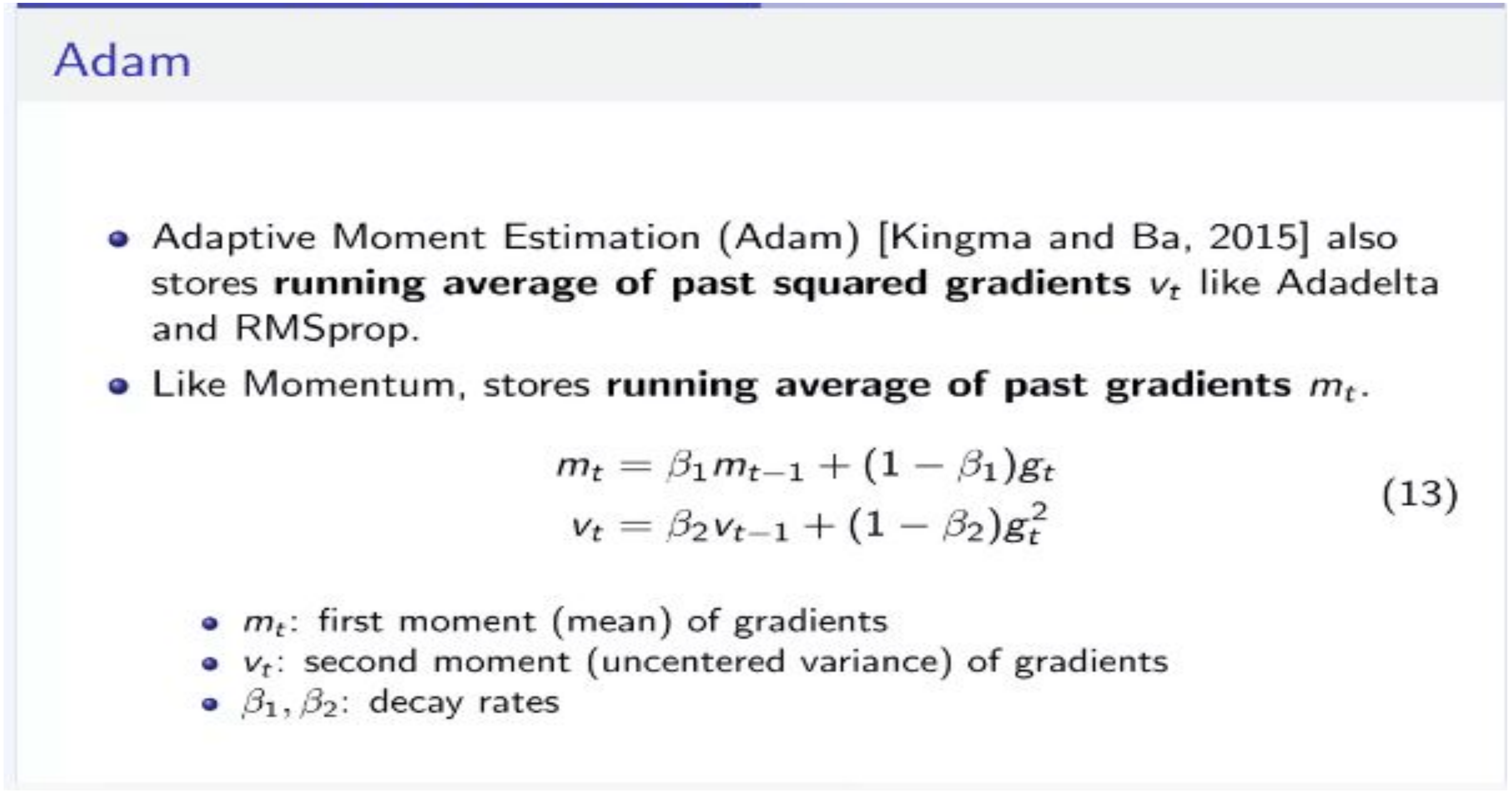


RMSprop

- Developed independently from Adadelta around the same time by Geoff Hinton.
- Also divides learning rate by a **running average of squared gradients**.
- RMSprop update:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

  - γ: decay parameter; typically set to 0.9
  - η: learning rate; a good default value is 0.001

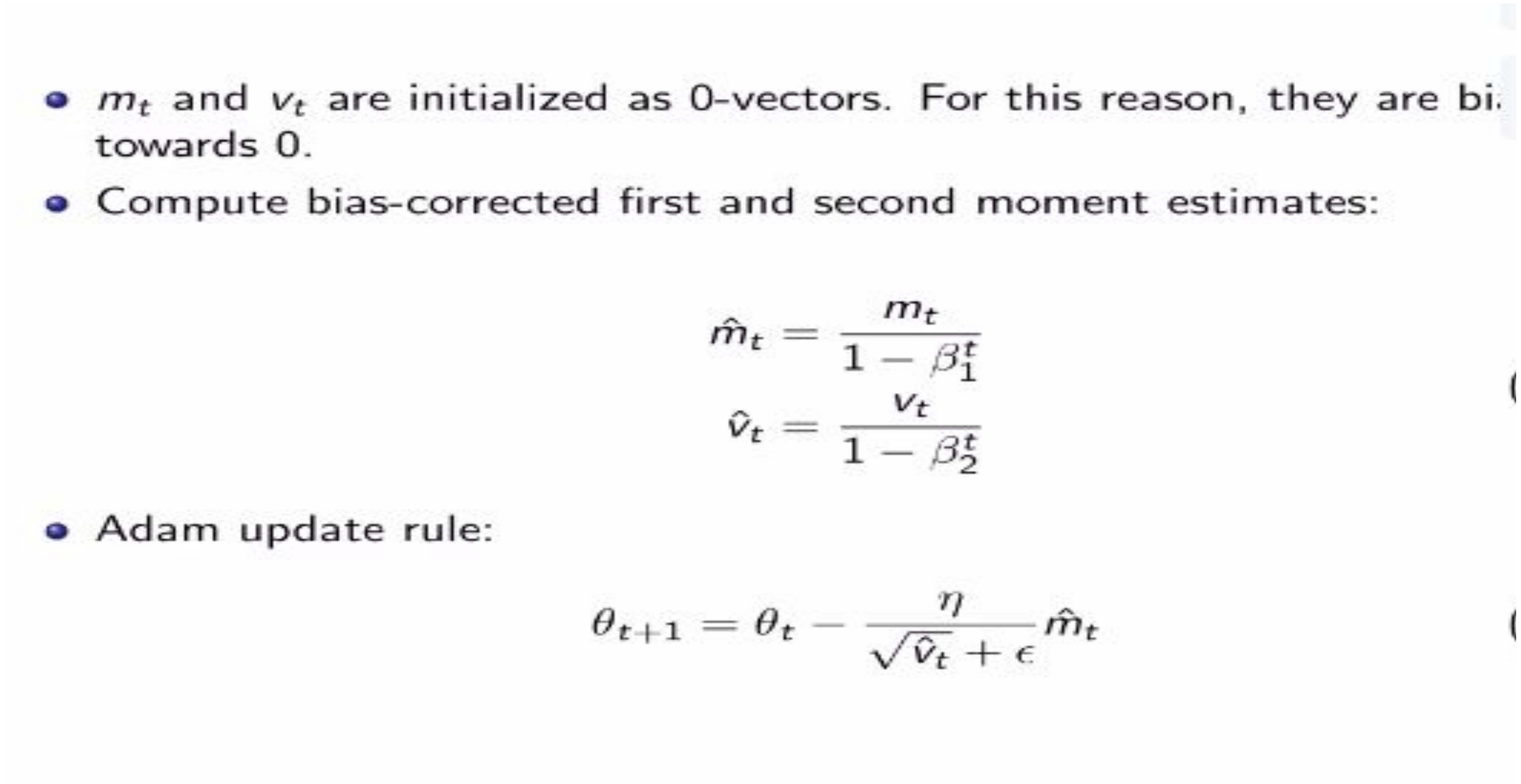


Adam

- Adaptive Moment Estimation (Adam) [Kingma and Ba, 2015] also stores **running average of past squared gradients** v_t like Adadelta and RMSprop.
- Like Momentum, stores **running average of past gradients** m_t.

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \qquad (13)$$

  - m_t: first moment (mean) of gradients
  - v_t: second moment (uncentered variance) of gradients
  - β_1, β_2: decay rates



- m_t and v_t are initialized as 0-vectors. For this reason, they are bi towards 0.
- Compute bias-corrected first and second moment estimates:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

- Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$$

THANK YOU