FIRST ORDER OPTIMISATION METHODS

CASSANDRA DURR

15 OCTOBER 2025

Lancaster AI (LAI)

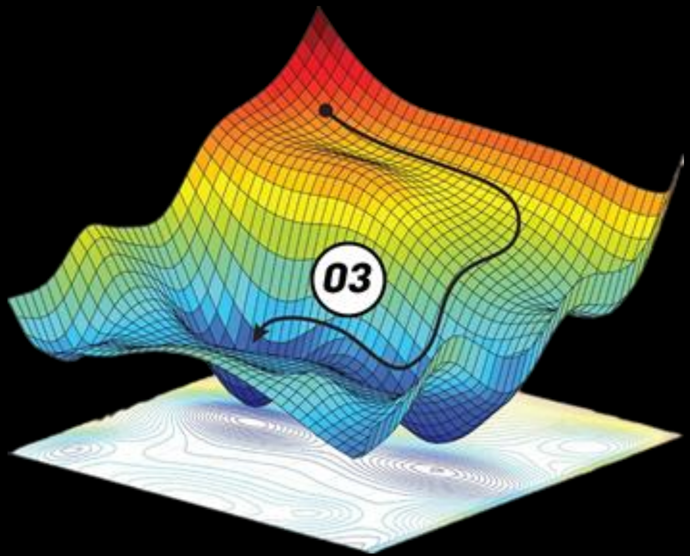
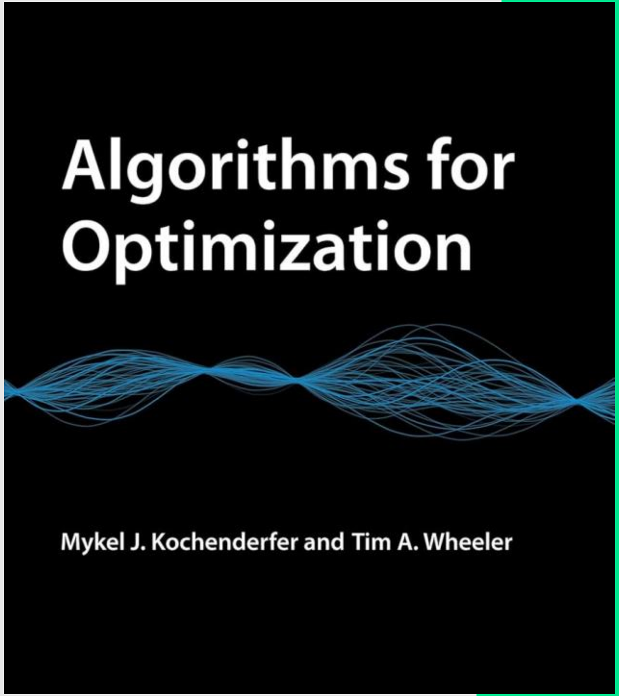


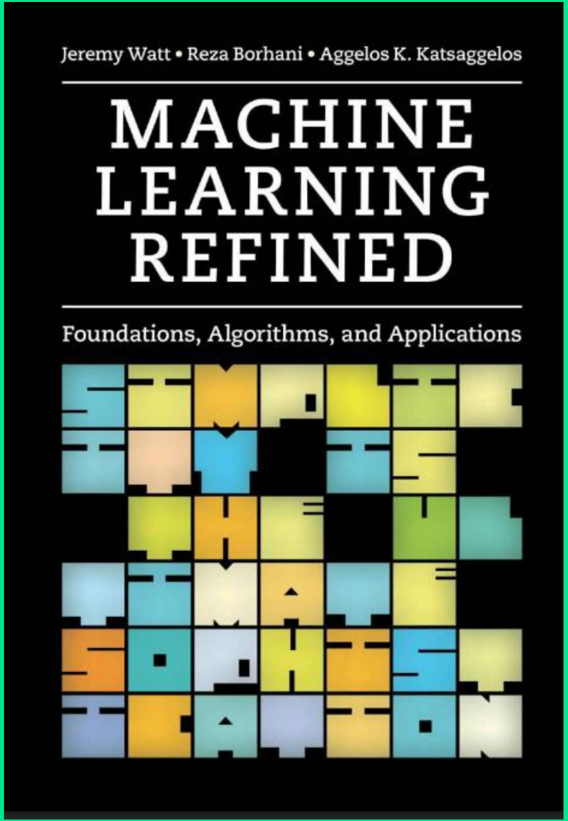
IMAGE SOURCE [0]

SOURCES

Chapter 5
Source [1] in references



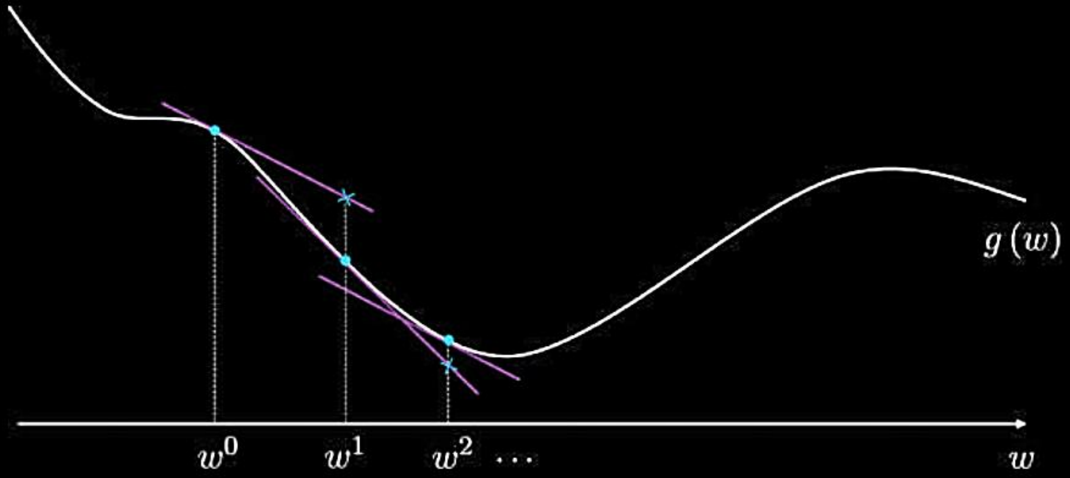
Chapter 2 & 8
Source [2] in references



Gradient Descent

First-order methods are algorithms that use the first derivative (i.e. gradient) to direct the optimisation procedure/ search towards a local minimum.

Linear model of $g(\mathbf{w})$ = the first-order Taylor Series approximation: $h(\mathbf{w}) = g(\mathbf{w}^0) + \nabla g(\mathbf{w}^0)^T(\mathbf{w} - \mathbf{w}^0)$



Update equation:

$$\mathbf{w}^{(k+1)} \leftarrow \mathbf{w}^{(k)} - \alpha_k \nabla g\left(\mathbf{w}^{(k)}\right)$$

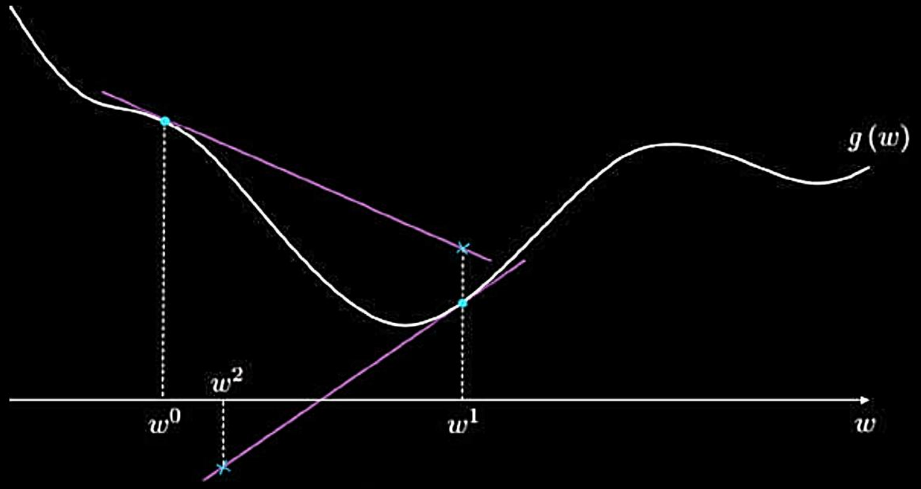
The step size/ learning rate associated with the kth descent step.

Stopping criteria: The gradient $\nabla g\left(\mathbf{w}^k\right)$ becomes sufficiently close to **0**.

IMAGE SOURCE [2]

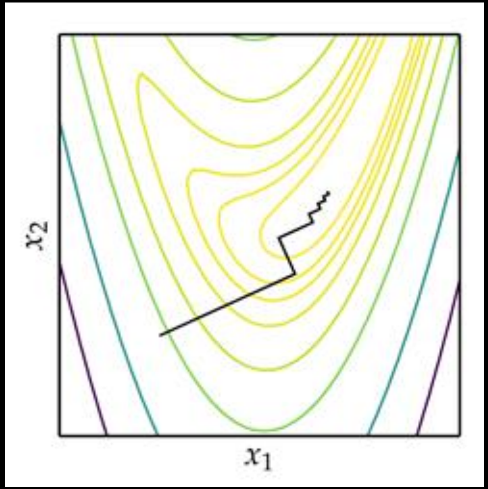
Gradient Descent: The Pitfalls

OVERSHOOTING



Too large a step length may result in overshooting the minimum.
SOURCE [2]

ZIG-ZAGGING



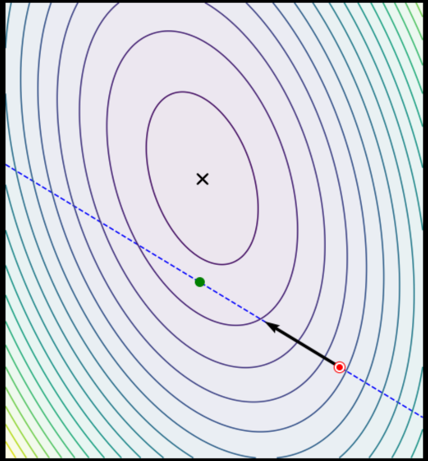
Gradient descent can result in zig-zagging in narrow valleys or troughs
SOURCE [1]

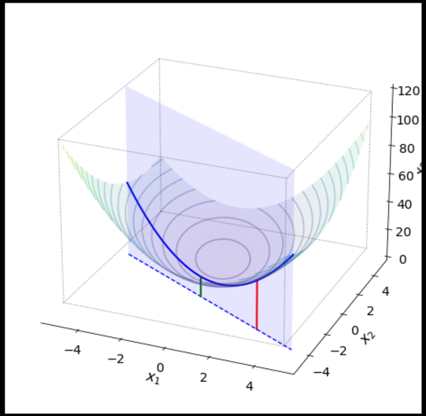
Gradient Descent: Zig-Zagging

Gradient descent tells you **which direction** to go in, but not **how far** to go.

STEEPEST GRADIENT DESCENT

Step size/ learning rate is optimally selected to produce the largest gain along the negative gradient direction.





1. Gradient descent provides the direction.
2. Isolate the hyperplane cutting through the function along the selected direction.
3. The intersection of the original function and the hyperplane is a parabola.
4. The minimum of the parabola indicates the optimal step length.

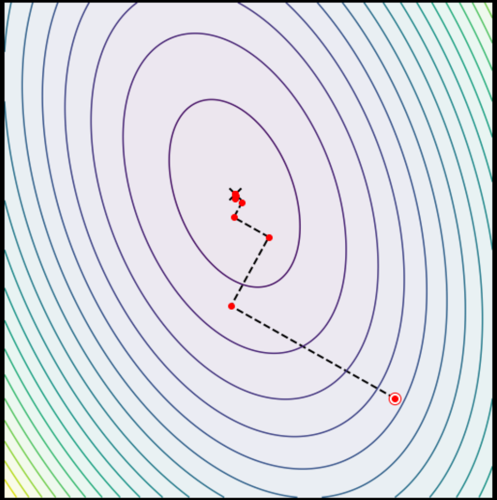
IMAGE SOURCE [3]

Gradient Descent: Zig-Zagging

Gradient descent tells you **which direction** to go in, but not **how far** to go.

STEEPEST GRADIENT DESCENT

Step size/ learning rate is optimally selected to produce the largest gain along the negative gradient direction.



However, steepest gradient descent can result in zig-zagging in narrow valleys or troughs.

The steps resulting from steepest gradient descent are **orthogonal**.

If you took a step that was not orthogonal to your last step, it would mean that there is some **shared direction** between your last and next step. This implies you should have gone further in your previous step.

The resulting zig-zagging is **not the most efficient** search through the space.

IMAGE SOURCE [3]

IMPROVING GRADIENT DESCENT

Conjugate Gradient

Momentum

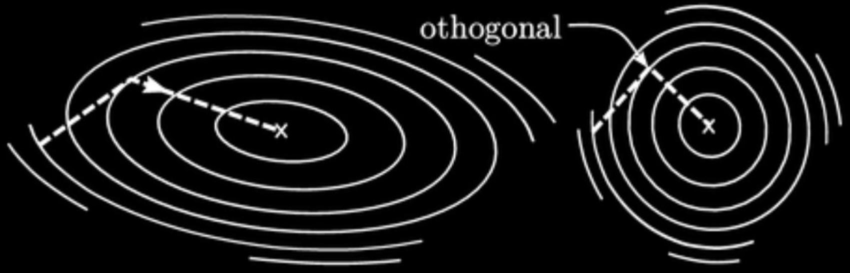
Nesterov Momentum

This section covers methods designed to overcome the limitations of vanilla gradient descent by incorporating memory of past steps.

Conjugate Gradient Descent

CONJUGATE GRADIENT DESCENT

- Balance between **gradient descent** and **Newton's method** (second-order method)
- Steepest gradient descent is **slow** and **zig-zags** in narrow valleys
- Newton's Method is **computationally expensive** (requires the inverse Hessian)



EXAMPLE

Assuming an n-dimensional **quadratic** function

- **Quadratic function:** $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} + c$
- Directions d_i & d_j are **mutually conjugate with A** if:

$$\boldsymbol{d}_i^T A\boldsymbol{d}_j = 0 \; \forall \; i \neq j$$

- The conjugate vectors form a basis of A.
- Min can be found in n steps with CG.

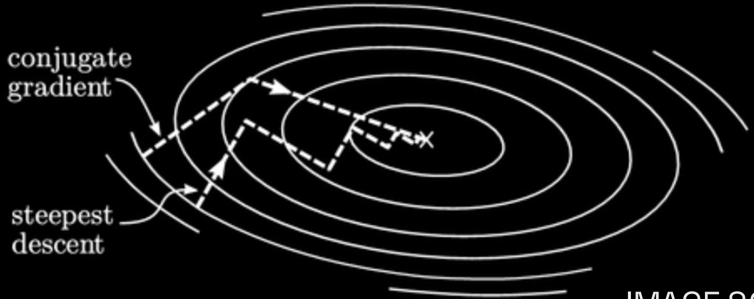


IMAGE SOURCE [7]

Conjugate Gradient Descent

ALGORITHM

1. First step: use steepest descent
   - Direction $\boldsymbol{d}_1 = -\boldsymbol{g}_1$
   - Steepest descent step size α_1
   - Update position $\boldsymbol{x}_2 = \boldsymbol{x}_1 + \alpha_1 \boldsymbol{d}_1$
2. Subsequent steps:
   - Direction: $\boldsymbol{d}_k = -\boldsymbol{g}_k + \beta_k \boldsymbol{d}_{k-1}$

Current gradient

Previous search direction

DERIVING β_k

$$\boldsymbol{d}_k^T A \boldsymbol{d}_{k-1} = \boldsymbol{0}$$

$$(-\boldsymbol{g}_k + \beta_k \boldsymbol{d}_{k-1})^T A \boldsymbol{d}_{k-1} = \boldsymbol{0}$$

$$-\boldsymbol{g}_k^T A \boldsymbol{d}_{k-1} + \beta_k \boldsymbol{d}_{k-1}^T A \boldsymbol{d}_{k-1} = \boldsymbol{0}$$

$$\beta_k = \frac{\boldsymbol{g}_k^T A \boldsymbol{d}_{k-1}}{\boldsymbol{d}_{k-1}^T A \boldsymbol{d}_{k-1}}$$

Assuming an n-dimensional **quadratic** function

Conjugate Gradient Descent

NON-QUADRATIC FUNCTIONS

- Smooth functions behave like quadratic functions close to a local minimum.
- Difficult to estimate/ approximate Hessian in a local region.
- Require “Hessian-less” approximations.

FLETCHER-REEVES UPDATE β_k

$$\beta_k = \frac{\boldsymbol{g}_k^T \boldsymbol{g}_k}{\boldsymbol{g}_{k-1}^T \boldsymbol{g}_{k-1}}$$

Assuming an n-dimensional **non- quadratic** function

POLAK-RIBIERE UPDATE β_k

$$\beta_k = \frac{\boldsymbol{g}_k^T (\boldsymbol{g}_k - \boldsymbol{g}_{k-1})}{\boldsymbol{g}_{k-1}^T \boldsymbol{g}_{k-1}}$$

$$\beta \leftarrow \max(\beta, 0)$$

IMAGE SOURCE [7]

Momentum

Momentum remembers the previous update and computes the next update as a linear combination of the current gradient and the previous update.

Gradient descent takes a long time to traverse surfaces which are almost flat → momentum helps to speed up descent over flat surfaces.

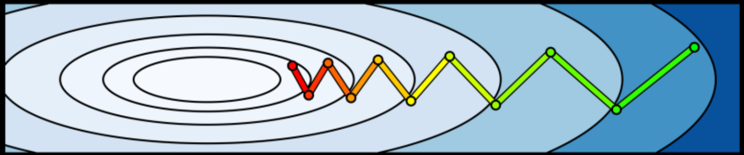
Momentum also mitigates the zig-zagging effect associated with steepest gradient descent.

UPDATE EQUATIONS

Velocity vector: $\boldsymbol{v}^{(k+1)} \leftarrow \beta \boldsymbol{v}^{(k)} - \alpha^{(k)} \boldsymbol{g}^{(k)}$

- Inertia term: $\beta \boldsymbol{v}^{(k)}$
- The gradient term: $-\alpha^{(k)} \boldsymbol{g}^{(k)}$
- β is the momentum decay coefficient $\epsilon\ (0,1)$

Without momentum



With momentum

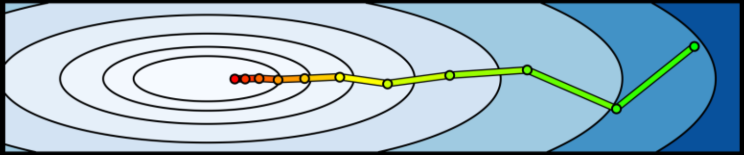


IMAGE SOURCE [4]

Momentum

Momentum remembers the previous update and computes the next update as a linear combination of the current gradient and the previous update.

Gradient descent takes a long time to traverse surfaces which are almost flat → momentum helps to speed up descent over flat surfaces.

Momentum also mitigates the zig-zagging effect associated with steepest gradient descent.

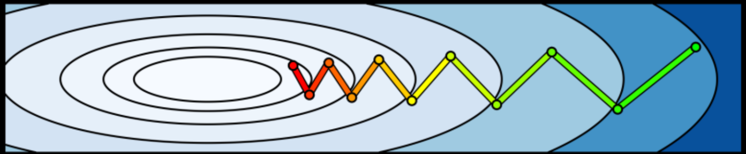
UPDATE EQUATIONS

Velocity vector: $\boldsymbol{v}^{(k+1)} \leftarrow \beta \boldsymbol{v}^{(k)} - \alpha^{(k)} \boldsymbol{g}^{(k)}$

Position vector: $\boldsymbol{x}^{(k+1)} \leftarrow \boldsymbol{x}^{(k)} + \boldsymbol{v}^{(k+1)}$

Initialise $\boldsymbol{v}^{(0)}$ as $\mathbf{0}$ → first step is vanilla gradient descent.

Without momentum



With momentum

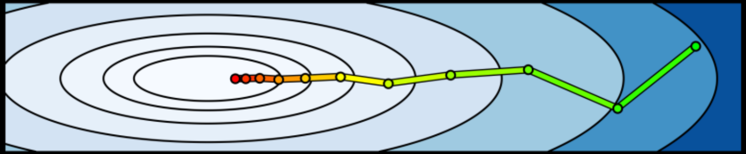
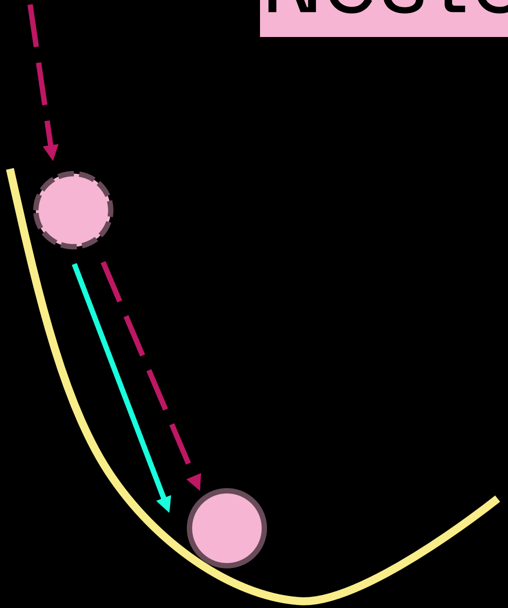


IMAGE SOURCE [4]

Nesterov Momentum

ISSUE WITH MOMENTUM

Momentum does not slow down sufficiently at the minimum, resulting in overshooting.



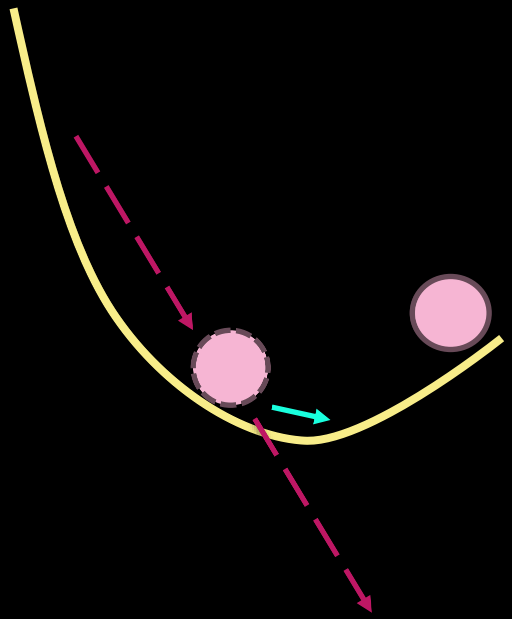
APPROACHING THE MINIMUM

The gradient is in the same direction as the velocity. The **inertia term** ($\beta \boldsymbol{v}^{(k)}$) and the **gradient term** ($-\alpha^{(k)} \boldsymbol{g}^{(k)}$) work together, increasing velocity.

Nesterov Momentum

ISSUE WITH MOMENTUM

Momentum does not slow down sufficiently at the minimum, resulting in overshooting.



NEAR THE MINIMUM

The **inertia term** ($\beta \boldsymbol{v}^{(k)}$) and the **gradient term** ($-\alpha^{(k)} \boldsymbol{g}^{(k)}$) oppose. Near the minimum, the gradient approaches **0**, so the inertia drives the velocity update. This will push the parameters up the other side of the slope, overshooting the minimum.

Nesterov Momentum

SOLUTION

- Modifies the momentum update by “looking ahead”.
- Uses the projected gradient.

UPDATE EQUATIONS

Velocity vector:

$$\boldsymbol{v}^{(k+1)} \leftarrow \beta \boldsymbol{v}^{(k)} - \alpha^{(k)} \boxed{\nabla f\left(\boldsymbol{x}^{(k)} + \beta \boldsymbol{v}^{(k)}\right)}$$

Look-ahead gradient

Position vector:

$$\boldsymbol{x}^{(k+1)} \leftarrow \boldsymbol{x}^{(k)} + \boldsymbol{v}^{(k+1)}$$

Initialise $\boldsymbol{v}^{(0)}$ as $\mathbf{0}$ → first step is vanilla gradient descent.

WHY IT WORKS?

By “looking ahead” you can see that the slope starting to flatten out/ increase, so reduce velocity to prevent overshooting the minimum.

ADAPTIVE METHODS

AdaGrad

RMSProp

Adadelta

Adam

This section explores a family of algorithms that adapt the learning rate for each parameter individually.

Adaptive Methods

PROBLEM WITH FIXED METHODS

$$\boldsymbol{x}^{(k+1)} \leftarrow \boldsymbol{x}^{(k)} - \alpha^{(k)} \nabla g\left(\boldsymbol{x}^{(k)}\right)$$

- All parameters are affected by the same learning rate.
- “One-size-fits-all” solution.
- Steep valley: smaller learning rate to prevent oscillations.
- Plateau: large learning rate to keep moving.

ADAPTIVE METHODS

$$\boldsymbol{x}^{(k+1)} \leftarrow \boldsymbol{x}^{(k)} - \boldsymbol{\alpha}^{(k)} \odot \nabla g\left(\boldsymbol{x}^{(k)}\right)$$

- Personalised learning rate per parameter that adapts over training.
- Faster convergence.
- More stable.
- Larger computational cost.

AdaGrad

$$\boldsymbol{x}^{(k+1)} \leftarrow \boldsymbol{x}^{(k)} - \boldsymbol{\alpha}^{(k)} \odot \nabla g\left(\boldsymbol{x}^{(k)}\right)$$

PARAMETER LEARNING RATE

$$\alpha_i^{(k)} = \frac{\alpha}{\epsilon + \sqrt{s_i^{(k)}}}$$

SUM OF SQUARES

$$s_i^{(k)} = \sum_{j=1}^{k} \left(g_i^{(j)}\right)^2$$

- α: baseline learning rate
- ϵ: small value to prevent division by 0
- $g_i^{(j)}$, j $\in \{1, \ldots, k\}$: history of gradients for i^{th} parameter (up to most recent step k)

Intuition:
Sum of squares term will grow as training progresses, increasing the denominator of $\alpha_i^{(k)}$, so the learning rate decreases over time.

❌ **Disadvantage:**
Monotonically decreasing learning rate → sometimes premature stopping.

RMSProp

Extends AdaGrad to **prevent monotonically decreasing** learning rates.

$$\boldsymbol{x}^{(k+1)} \leftarrow \boldsymbol{x}^{(k)} - \boldsymbol{\alpha}^{(k)} \odot \nabla g\left(\boldsymbol{x}^{(k)}\right)$$

- α: baseline learning rate
- ϵ: small value to prevent division by 0
- γ: decay rate $\epsilon\ (0, 1)$

PARAMETER LEARNING RATE

$$\alpha_i^{(k)} = \frac{\alpha}{\epsilon + \sqrt{s_i^{(k)}}}$$

SQUARED GRADIENTS

$$s_i^{(k)} = \gamma s_i^{(k-1)} + (1-\gamma)\left(g_i^{(k)}\right)^2$$

Moving average of squared gradients.

$$s_i^{(k)} = \boxed{\gamma s_i^{(k-1)}} + \boxed{(1-\gamma)\left(g_i^{(k)}\right)^2}$$

Decay previous squared gradients.

Add new information about the current gradient.

RMSProp

Extends AdaGrad to **prevent monotonically decreasing** learning rates.

$$\boldsymbol{x}^{(k+1)} \leftarrow \boldsymbol{x}^{(k)} - \boldsymbol{\alpha}^{(k)} \odot \nabla g\left(\boldsymbol{x}^{(k)}\right)$$

PARAMETER LEARNING RATE

$$\alpha_i^{(k)} = \frac{\alpha}{\epsilon + \sqrt{s_i^{(k)}}}$$

SQUARED GRADIENTS

$$s_i^{(k)} = \gamma s_i^{(k-1)} + (1-\gamma)\left(g_i^{(k)}\right)^2$$

Moving average of squared gradients.

ADVANTAGES

- Adaptive learning rate.
- Non-monotonically decreasing learning rates.
- Gives more weight to recent gradients than older gradients.

DISADVANTAGES

- Additional hyperparameter to tune, γ.

Adadelta

Extends AdaGrad to prevent **monotonically decreasing** learning rates.
Improves on RMSProp by **not requiring a global learning rate**, α.

1. ACCUMULATE GRADIENTS

$$s_i^{(k)} = \gamma s_i^{(k-1)} + (1-\gamma)\left(g_i^{(k)}\right)^2$$

2. CALCULATE PARAMETER UPDATE

$$\Delta x_i^{(k)} = -\frac{\epsilon + \sqrt{u_i^{(k-1)}}}{\epsilon + \sqrt{s_i^{(k)}}} g_i^{(k)}$$

$$u_i^{(0)} = s_i^{(0)} = 0$$

3. ACCUMULATE PARAMETER UPDATES

$$u_i^{(k)} = \gamma u_i^{(k-1)} + (1-\gamma)\left(\Delta x_i^{(k)}\right)^2$$

4. APPLY PARAMETER UPDATE

$$x_i^{(k)} = x_i^{(k-1)} + \Delta x_i^{(k)}$$

The moving average of parameter updates $u_i^{(k)}$ learns an appropriate scale for the updates.
Self-adjusting per-parameter rate: no need for global alpha value.

Adam

Initialising the gradient and squared gradient to zero introduces a bias. A bias correction step alleviates this issue.

Momentum-component

$$v_i^{(k)} = \rho v_i^{(k-1)} + (1-\rho) g_i^{(k)}$$

Biased decaying momentum/ moving average of gradient.

First moment of gradient

Squared gradients from RMSProp

$$s_i^{(k)} = \gamma s_i^{(k-1)} + (1-\gamma)\left(g_i^{(k)}\right)^2$$

Biased decaying squared gradients/ moving average of squared gradient.

Second moment of gradient

$$\hat{v}_i^{(k)} = \frac{v_i^{(k)}}{1-\rho^k}$$

Corrected decaying momentum

$$\hat{s}_i^{(k)} = \frac{s_i^{(k)}}{1-\gamma^k}$$

Corrected decaying squared gradients

Bias correction

$$x_i^{(k)} = x_i^{(k-1)} - \alpha \frac{\hat{v}_i^{(k)}}{\epsilon + \sqrt{\hat{s}_i^{(k)}}}$$

Parameter update

Adam

Recommended hyperparameters

$$v_i^{(k)} = \rho v_i^{(k-1)} + (1-\rho) g_i^{(k)}$$

Biased decaying momentum/ moving average of gradient.

$$s_i^{(k)} = \gamma s_i^{(k-1)} + (1-\gamma)\left(g_i^{(k)}\right)^2$$

Biased decaying squared gradients/ moving average of squared gradient.

$\gamma = 0.999, \rho = 0.9$

$$\hat{v}_i^{(k)} = \frac{v_i^{(k)}}{1-\rho^k}$$

Corrected decaying momentum

$$\hat{s}_i^{(k)} = \frac{s_i^{(k)}}{1-\gamma^k}$$

Corrected decaying squared gradients

$$x_i^{(k)} = x_i^{(k-1)} - \alpha \frac{\hat{v}_i^{(k)}}{\epsilon + \sqrt{\hat{s}_i^{(k)}}}$$

Parameter update

$\alpha = 0.001, \epsilon = 10^{-8}$

Adam: Bias correction

$$v_i^{(k)} = \rho v_i^{(k-1)} + (1-\rho) g_i^{(k)}$$

Moving average of gradient

$$v_i^{(k)} = (1-\rho)\sum_{j=1}^{k} \rho^{k-j}\, g_i^{(j)}$$

Equivalent expression

Take the expectation on both sides

$$E\left(v_i^{(k)}\right) = E\left(g_i^{(k)}\right)(1-\rho)\sum_{j=1}^{k}\rho^{k-j} + \delta$$

(Almost) stationary expectation of gradient

$$E\left(v_i^{(k)}\right) = (1-\rho)\sum_{j=1}^{k}\rho^{k-j}\, E\left(g_i^{(j)}\right)$$

$$E\left(g_i^{(j)}\right) = E\left(g_i^{(k)}\right)\ \forall\, j \in \{1, \dots, k\}$$

$$E\left(v_i^{(k)}\right) = E\left(g_i^{(k)}\right)(1-\rho)\sum_{j=1}^{k}\rho^{k-j}$$

$$E\left(v_i^{(k)}\right) = E\left(g_i^{(k)}\right)\left(1-\rho^k\right) + \delta$$

$$E\left(g_i^{(k)}\right) \cong \frac{E\left(v_i^{(k)}\right)}{\left(1-\rho^k\right)}$$

Finite geometric series

$$\sum_{j=1}^{k}\rho^{k-j} = \sum_{j=0}^{k-1}\rho^{k} = \frac{1-\rho^k}{1-\rho} \therefore (1-\rho)\sum_{j=1}^{k}\rho^{k-j} = \left(1-\rho^k\right)$$

$$\hat{v}_i^{(k)} = \frac{v_i^{(k)}}{1-\rho^k}$$

SOURCE [5]

ENHANCING FIRST ORDER METHODS

Hypergradient descent

This section explores enhancements to first order methods.

Hypergradient descent

Dynamically **update the global learning rate** associated with gradient descent methods.

Algorithm 2 SGD with Nesterov (SGDN)

Require: μ: momentum

$t, v_0 \leftarrow 0, 0$ ▷ Initialization

Update rule:

$v_t \leftarrow \mu\, v_{t-1} + g_t$ ▷ "Velocity"

$u_t \leftarrow -\alpha\,(g_t + \mu\, v_t)$ ▷ Parameter update

Algorithm 5 SGDN with hyp. desc. (SGDN-HD)

Require: μ: momentum

$t, v_0, \nabla_\alpha u_0 \leftarrow 0, 0, 0$ ▷ Initialization

Update rule:

$v_t \leftarrow \mu\, v_{t-1} + g_t$ ▷ "Velocity"

$u_t \leftarrow -\alpha_t\,(g_t + \mu\, v_t)$ ▷ Parameter update

$\nabla_\alpha u_t \leftarrow -g_t - \mu\, v_t$

Algorithm 3 Adam

Require: $\beta_1, \beta_2 \in [0, 1)$: decay rates for Adam

$t, m_0, v_0 \leftarrow 0, 0, 0$ ▷ Initialization

Update rule:

$m_t \leftarrow \beta_1\, m_{t-1} + (1 - \beta_1)\, g_t$ ▷ 1st mom. estimate

$v_t \leftarrow \beta_2\, v_{t-1} + (1 - \beta_2)\, g_t^2$ ▷ 2nd mom. estimate

$\widehat{m}_t \leftarrow m_t/(1 - \beta_1^t)$ ▷ Bias correction

$\widehat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ ▷ Bias correction

$u_t \leftarrow -\alpha\, \widehat{m}_t/(\sqrt{\widehat{v}_t} + \epsilon)$ ▷ Parameter update

Algorithm 6 Adam with hyp. desc. (Adam-HD)

Require: $\beta_1, \beta_2 \in [0, 1)$: decay rates for Adam

$t, m_0, v_0, \nabla_\alpha u_0 \leftarrow 0, 0, 0, 0$ ▷ Initialization

Update rule:

$m_t \leftarrow \beta_1\, m_{t-1} + (1 - \beta_1)\, g_t$ ▷ 1st mom. estimate

$v_t \leftarrow \beta_2\, v_{t-1} + (1 - \beta_2)\, g_t^2$ ▷ 2nd mom. estimate

$\widehat{m}_t \leftarrow m_t/(1 - \beta_1^t)$ ▷ Bias correction

$\widehat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ ▷ Bias correction

$u_t \leftarrow -\alpha_t\, \widehat{m}_t/(\sqrt{\widehat{v}_t} + \epsilon)$ ▷ Parameter update

$\nabla_\alpha u_t \leftarrow -\widehat{m}_t/(\sqrt{\widehat{v}_t} + \epsilon)$

SOURCE [6]

Hypergradient descent

Dynamically **update the global learning rate** associated with gradient descent methods.

PROBLEM

Gradient descent methods are often quite sensitive to the choice of global learning rate/ step length.

WHAT IS A HYPERGRADIENT?

The derivative with respect to a hyperparameter (such as the global learning rate).

SOLUTION

Apply gradient descent to the hyperparameter (global LR) of an underlying descent method.

WHY IT WORKS?

Hypergradient algorithms reduce sensitivity to the hyperparameter, allowing it to adapt faster.

Hypergradient descent

Dynamically **update the global learning rate** associated with gradient descent methods.

Basic gradient descent update

$$x_i^{(k)} \leftarrow x_i^{(k-1)} - \alpha \nabla g\left(x_i^{(k-1)}\right)$$

Derivative of the objective function w.r.t. the hyperparameter

(requires storing an extra copy of the gradient)

$$\frac{\partial g\left(x_i^{(k-1)}\right)}{\partial \alpha} = \nabla g\left(x_i^{(k-1)}\right) \cdot \frac{\partial x_i^{(k-1)}}{\partial \alpha}$$

$$= \nabla g\left(x_i^{(k-1)}\right) \cdot \frac{\partial}{\partial \alpha}\left(x_i^{(k-2)} - \alpha \nabla g\left(x_i^{(k-2)}\right)\right)$$

$$= \nabla g\left(x_i^{(k-1)}\right) \cdot -\nabla g\left(x_i^{(k-2)}\right)$$

Hyperparameter update equation

$$\alpha^{(k)} \leftarrow \alpha^{(k-1)} - \beta \frac{\partial x_i^{(k)}}{\partial \alpha}$$

Hyperparameter learning rate

$$\alpha^{(k)} \leftarrow \alpha^{(k-1)} + \beta \nabla g\left(x_i^{(k-1)}\right) \cdot \nabla g\left(x_i^{(k-2)}\right)$$

LINKS AND RESOURCES

[0] Gad, A. F. (2019). Implementing Gradient Descent in Python, Part 3: Adding a Hidden Layer. DigitalOcean. https://blog.paperspace.com/part-3-generic-python-implementation-of-gradient-descent-for-nn-optimization/

[1] Kochenderfer, M. J., & Wheeler, T. A. (2019). Algorithms for Optimization. MIT Press. https://algorithmsbook.com/optimization/files/optimization.pdf

[2] Watt, J., Borhani, R., & Katsaggelos, A. K. (2020). *Machine learning refined: Foundations, algorithms, and applications*. Cambridge University Press. https://www.r-5.org/files/books/computers/algo-list/statistics/data-mining/Jeremy_Watt-Machine_Learning_Refined-EN.pdf

[3] Gundersen, G. (2022) Conjugate Gradient Descent. https://gregorygundersen.com/blog/2022/03/20/conjugate-gradient-descent/

[4] Watt, J., Borhani, R. (2025) 13.4 Momentum methods. https://kenndanielso.github.io/mlrefined/blog_posts/13_Multilayer_perceptrons/13_4_Momentum_methods.html

[5] Kingma, D. P. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[6] Baydin, A. G., Cornish, R., Rubio, D. M., Schmidt, M., & Wood, F. (2017). Online learning rate adaptation with hypergradient descent. *arXiv preprint arXiv:1703.04782*.

[7] Greenshields, C., Weller, H. (2022). Conjugate gradient method. https://doc.cfd.direct/notes/cfd-general-principles/conjugate-gradient-method

THANK YOU!

Contact: c.durr@lancaster.ac.uk