Parameter Optimization

F. Noé[1]

Deep Learning Classes, FU Berlin 2018

Finding optimal model parameters
Minimization of cost functions

Symbols

Meaning	Symbol	Shape
Feature vector and Matrix	$\mathbf{x}$	$\mathbb{R}^n$
Data set matrix	$\mathbf{X}$	$\mathbb{R}^{N\times n}$
Label matrix	$\mathbf{y}$	$\mathbb{R}^l$
Label matrix	$\mathbf{Y}$	$\mathbb{R}^{N\times l}$
Parameters	θ	$\mathbb{R}^d$
Loss / Cost function	$C(\mathbf{X};\theta)$ or $C(\mathbf{X},\mathbf{Y};\theta)$	$\mathbb{R}$

Cost function C quantifies how well a given model with parameters θ explains the observations $\mathbf{X}$.

Model fitting

$\hat{\theta} = \arg\min_\theta C(\mathbf{X};\theta)$

Minimimizing the *cost* C is equivalent to maximizing the *score* $-C$.

Finding optimal model parameters

Minimization of cost functions

Symbols

Meaning	Symbol	Shape
Feature vector and Matrix	$\mathbf{x}$	$\mathbb{R}^n$
Data set matrix	$\mathbf{X}$	$\mathbb{R}^{N \times n}$
Label matrix	$\mathbf{y}$	$\mathbb{R}^l$
Label matrix	$\mathbf{Y}$	$\mathbb{R}^{N \times l}$
Parameters	θ	$\mathbb{R}^d$
Loss / Cost function	$C(\mathbf{X}; \theta)$ or $C(\mathbf{X}, \mathbf{Y}; \theta)$	$\mathbb{R}$

Cost function C quantifies how well a given model with parameters θ explains the observations $\mathbf{X}$.

Model fitting

$\hat{\theta} = \arg\min_\theta C(\mathbf{X}; \theta)$

Minimimizing the *cost* C is equivalent to maximizing the *score* $-C$.

Gradient descent

- Iteratively adjust the parameters in the direction of

$$-\nabla_\theta C(\mathbf{X}; \theta)$$

  Parameters move towards a local minimum of the cost function.

- Difficulties:
  - Cost functions are usually non-convex, and often rugged
    $\rightarrow$ optimization may get stuck in local minima
  - Cost functions often involve terms for every data point
    $\rightarrow$ cost function is expensive to evaluate
  - Exact cost function often not available, must be estimated from data.
    $\rightarrow$ cost functions noisy, not all minima are meaningful.

- Further reading:
  - Y. LeCun, L. Bottou, G. B. Orr, K.-R. Müller: *Efficient backprop*, in Neural networks: Tricks of the trade (Springer) pp. 9–50 (1998).
  - L. Bottou: *Stochastic gradient descent tricks*, in Neural networks: Tricks of the trade (Springer), pp. 421–436 (2012).
  - S. Ruder: *An overview of gradient descent optimization algorithms*, arXiv:1609.04747 (2016).

Gradient descent

- Iteratively adjust the parameters in the direction of

$$-\nabla_\theta C(\mathbf{X}; \theta)$$

  Parameters move towards a local minimum of the cost function.
- Difficulties:
  - Cost functions are usually non-convex, and often rugged
    $\rightarrow$ optimization may get stuck in local minima
  - Cost functions often involve terms for every data point
    $\rightarrow$ cost function is expensive to evaluate
  - Exact cost function often not available, must be estimated from data.
    $\rightarrow$ cost functions noisy, not all minima are meaningful.
- Further reading:
  - Y. LeCun, L. Bottou, G. B. Orr, K.-R. Müller: *Efficient backprop*, in Neural networks: Tricks of the trade (Springer) pp. 9–50 (1998).
  - L. Bottou: *Stochastic gradient descent tricks*, in Neural networks: Tricks of the trade (Springer), pp. 421–436 (2012).
  - S. Ruder: *An overview of gradient descent optimization algorithms*, arXiv:1609.04747 (2016).

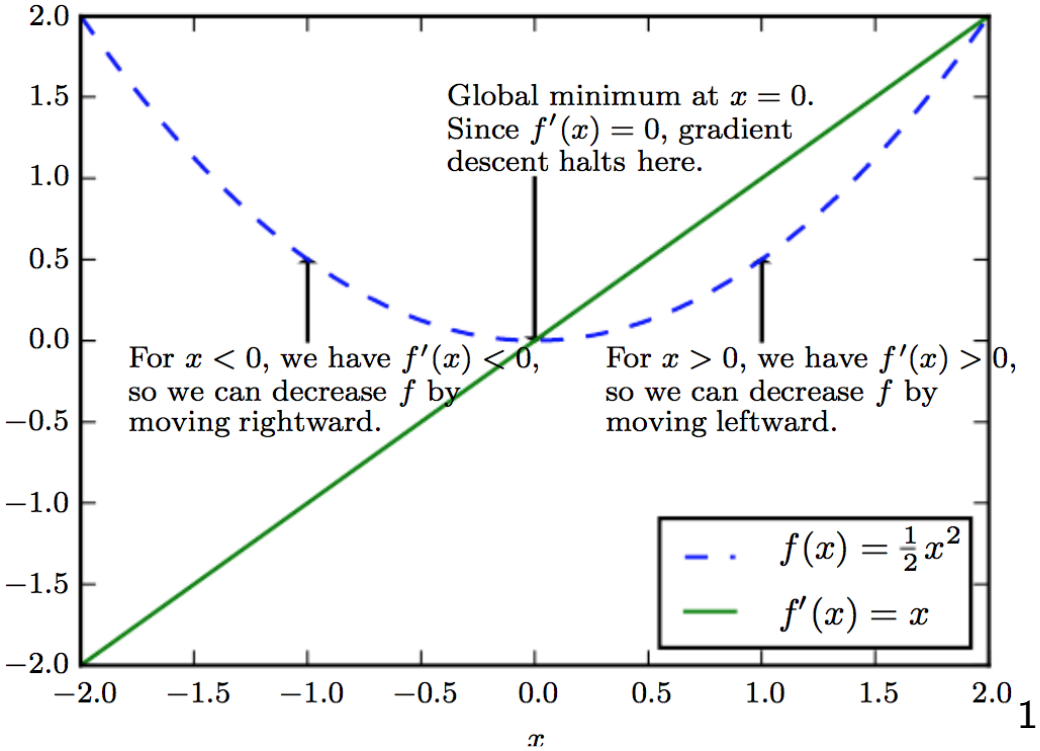
Gradient descent

- Iteratively adjust the parameters in the direction of

$$-\nabla_\theta C(\mathbf{X}; \theta)$$

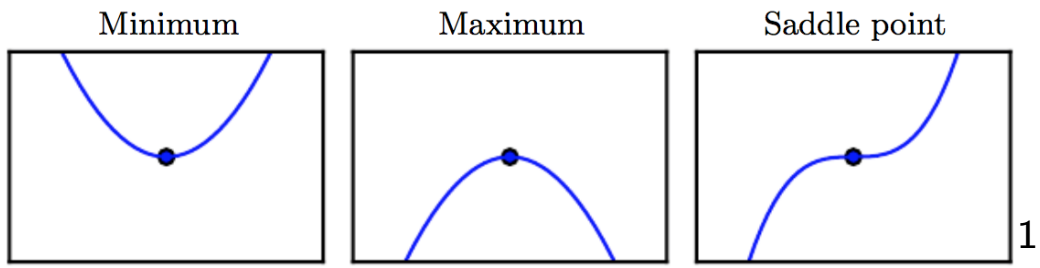
  Parameters move towards a local minimum of the cost function.
- Difficulties:
  - Cost functions are usually non-convex, and often rugged
    $\rightarrow$ optimization may get stuck in local minima
  - Cost functions often involve terms for every data point
    $\rightarrow$ cost function is expensive to evaluate
  - Exact cost function often not available, must be estimated from data.
    $\rightarrow$ cost functions noisy, not all minima are meaningful.
- Further reading:
  - Y. LeCun, L. Bottou, G. B. Orr, K.-R. Müller: *Efficient backprop*, in Neural networks: Tricks of the trade (Springer) pp. 9–50 (1998).
  - L. Bottou: *Stochastic gradient descent tricks*, in Neural networks: Tricks of the trade (Springer), pp. 421–436 (2012).
  - S. Ruder: *An overview of gradient descent optimization algorithms*, arXiv:1609.04747 (2016).

Gradient descent
Illustration



1

Note that $\nabla_\theta C(\mathbf{X}; \theta) = \mathbf{0}$ is fulfilled at *any* critical point. In high dimensions, most critical points are saddle points.



1

[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press 2016

Gradient descent

Abbreviation (data $\mathbf{X}$ is implicit):

$$C(\theta) \equiv C(\mathbf{X}; \theta)$$

Simple gradient descent algorithm

1. Initialize θ_0
2. For $t = 0, \ldots, T-1$ or until converged:
   1. Compute gradient $\mathbf{g}(\theta_t) = \nabla_\theta C(\theta_t)$
   2. Update parameters: $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}(\theta_t)$

- η_t is the learning rate that controls how big a step we should take in the search direction $-\mathbf{g}(\theta_t)$ at time t.
- For small η_t, this method will converge to a local minimum of $C(\theta_t)$, but involves a large computational cost.
- For large η_t, the algorithm can overshoot and even diverge.
- In practice, one decreases η_t over time or chooses it adaptively.

Gradient descent

Abbreviation (data $\mathbf{X}$ is implicit):

$$C(\theta) \equiv C(\mathbf{X}; \theta)$$

Simple gradient descent algorithm

1. Initialize θ_0
2. For $t = 0, \ldots, T-1$ or until converged:
   1. Compute gradient $\mathbf{g}(\theta_t) = \nabla_\theta C(\theta_t)$
   2. Update parameters: $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}(\theta_t)$

- η_t is the learning rate that controls how big a step we should take in the search direction $-\mathbf{g}(\theta_t)$ at time t.
- For small η_t, this method will converge to a local minimum of $C(\theta_t)$, but involves a large computational cost.
- For large η_t, the algorithm can overshoot and even diverge.
- In practice, one decreases η_t over time or chooses it adaptively.

Gradient descent

Abbreviation (data $\mathbf{X}$ is implicit):

$$C(\theta) \equiv C(\mathbf{X}; \theta)$$

Simple gradient descent algorithm

1. Initialize θ_0
2. For $t = 0, \ldots, T-1$ or until converged:
   1. Compute gradient $\mathbf{g}(\theta_t) = \nabla_\theta C(\theta_t)$
   2. Update parameters: $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}(\theta_t)$

- η_t is the learning rate that controls how big a step we should take in the search direction $-\mathbf{g}(\theta_t)$ at time t.
- For small η_t, this method will converge to a local minimum of $C(\theta_t)$, but involves a large computational cost.
- For large η_t, the algorithm can overshoot and even diverge.
- In practice, one decreases η_t over time or chooses it adaptively.

Gradient descent

Abbreviation (data $\mathbf{X}$ is implicit):

$$C(\theta) \equiv C(\mathbf{X}; \theta)$$

Simple gradient descent algorithm

1. Initialize θ_0
2. For $t = 0, \ldots, T-1$ or until converged:
   1. Compute gradient $\mathbf{g}(\theta_t) = \nabla_\theta C(\theta_t)$
   2. Update parameters: $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}(\theta_t)$

- η_t is the learning rate that controls how big a step we should take in the search direction $-\mathbf{g}(\theta_t)$ at time t.
- For small η_t, this method will converge to a local minimum of $C(\theta_t)$, but involves a large computational cost.
- For large η_t, the algorithm can overshoot and even diverge.
- In practice, one decreases η_t over time or chooses it adaptively.

Gradient descent

Abbreviation (data $\mathbf{X}$ is implicit):

$$C(\theta) \equiv C(\mathbf{X}; \theta)$$

Simple gradient descent algorithm

1. Initialize θ_0
2. For $t = 0, \ldots, T-1$ or until converged:
   1. Compute gradient $\mathbf{g}(\theta_t) = \nabla_\theta C(\theta_t)$
   2. Update parameters: $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}(\theta_t)$

- η_t is the learning rate that controls how big a step we should take in the search direction $-\mathbf{g}(\theta_t)$ at time t.
- For small η_t, this method will converge to a local minimum of $C(\theta_t)$, but involves a large computational cost.
- For large η_t, the algorithm can overshoot and even diverge.
- In practice, one decreases η_t over time or chooses it adaptively.

Gradient descent

Abbreviation (data $\mathbf{X}$ is implicit):

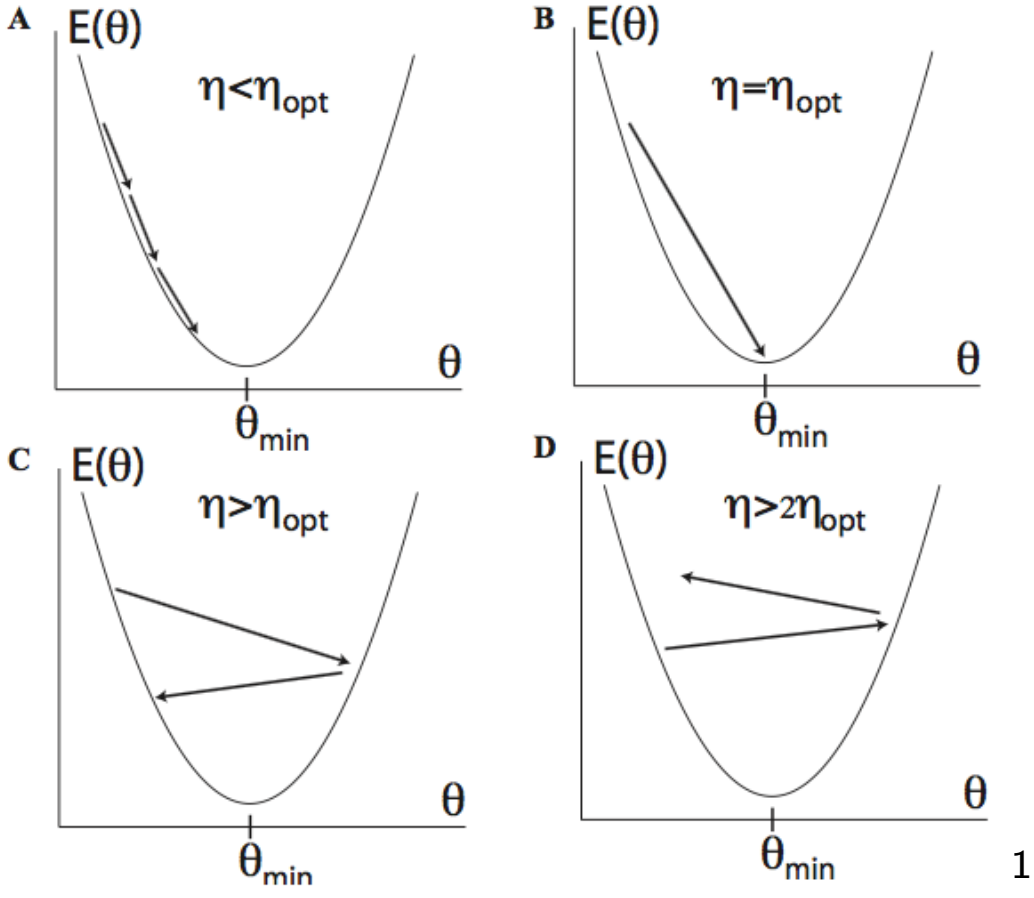
$$C(\theta) \equiv C(\mathbf{X}; \theta)$$

Simple gradient descent algorithm

1. Initialize θ_0
2. For $t = 0, \ldots, T-1$ or until converged:
   1. Compute gradient $\mathbf{g}(\theta_t) = \nabla_\theta C(\theta_t)$
   2. Update parameters: $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}(\theta_t)$

- η_t is the learning rate that controls how big a step we should take in the search direction $-\mathbf{g}(\theta_t)$ at time t.
- For small η_t, this method will converge to a local minimum of $C(\theta_t)$, but involves a large computational cost.
- For large η_t, the algorithm can overshoot and even diverge.
- In practice, one decreases η_t over time or chooses it adaptively.

Gradient descent
Choice of learning rate η_t



Convergence as a function of learning rate η_t in a quadratic potential.

[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press 2016

Gradient descent

Lessons from Newton (2nd order) for gradient descent (1st order)

Consider second-order Taylor expansion:

$$C(\theta+\mathbf{v}) \approx C(\theta)+\nabla_\theta C(\theta)\mathbf{v}+\frac{1}{2}\mathbf{v}^\top\left(\nabla_\theta\nabla_\theta^\top\right)(\theta)\mathbf{v}$$
$$= C(\theta)+\mathbf{g}(\theta)\mathbf{v}+\frac{1}{2}\mathbf{v}^\top\mathbf{H}(\theta)\mathbf{v}$$

with gradient $\mathbf{g}=\nabla_\theta C$ and Hessian matrix of second derivatives $\mathbf{H}=\nabla_\theta\nabla_\theta^\top$.

Differentiating this equation and seeking the optimum via $\nabla_\theta C(\theta+\mathbf{v}_{\text{opt}})=0$ yields:

$$\mathbf{g}(\theta)+\mathbf{H}(\theta)\mathbf{v}_{\text{opt}}=0$$

Rearranging yields the Newton update rules:

$$\mathbf{v}_t=-\mathbf{H}^{-1}(\theta_t)\mathbf{g}(\theta_t)$$
$$\theta_{t+\tau}=\theta_t+\mathbf{v}_t$$

Often the Hessian is poorly conditioned $\rightarrow$ replace inverse by regularized version, e.g. $[\mathbf{H}(\theta_t)+\varepsilon I]^{-1}$ with small parameter ε.

Gradient descent

Lessons from Newton (2nd order) for gradient descent (1st order)

Consider second-order Taylor expansion:

$$C(\theta+\mathbf{v}) \approx C(\theta)+\nabla_\theta C(\theta)\mathbf{v}+\frac{1}{2}\mathbf{v}^\top\left(\nabla_\theta\nabla_\theta^\top\right)(\theta)\mathbf{v}$$
$$= C(\theta)+\mathbf{g}(\theta)\mathbf{v}+\frac{1}{2}\mathbf{v}^\top\mathbf{H}(\theta)\mathbf{v}$$

with gradient $\mathbf{g}=\nabla_\theta C$ and Hessian matrix of second derivatives $\mathbf{H}=\nabla_\theta\nabla_\theta^\top$.
Differentiating this equation and seeking the optimum via $\nabla_\theta C(\theta+\mathbf{v}_{\text{opt}})=0$ yields:

$$\mathbf{g}(\theta)+\mathbf{H}(\theta)\mathbf{v}_{\text{opt}}=0$$

Rearranging yields the Newton update rules:

$$\mathbf{v}_t=-\mathbf{H}^{-1}(\theta_t)\mathbf{g}(\theta_t)$$
$$\theta_{t+\tau}=\theta_t+\mathbf{v}_t$$

Often the Hessian is poorly conditioned $\rightarrow$ replace inverse by regularized version, e.g. $[\mathbf{H}(\theta_t)+\varepsilon I]^{-1}$ with small parameter ε.

Gradient descent

Lessons from Newton (2nd order) for gradient descent (1st order)

Consider second-order Taylor expansion:

$$C(\theta+\mathbf{v}) \approx C(\theta)+\nabla_\theta C(\theta)\mathbf{v}+\frac{1}{2}\mathbf{v}^\top\left(\nabla_\theta\nabla_\theta^\top\right)(\theta)\mathbf{v}$$
$$= C(\theta)+\mathbf{g}(\theta)\mathbf{v}+\frac{1}{2}\mathbf{v}^\top\mathbf{H}(\theta)\mathbf{v}$$

with gradient $\mathbf{g}=\nabla_\theta C$ and Hessian matrix of second derivatives $\mathbf{H}=\nabla_\theta\nabla_\theta^\top$.
Differentiating this equation and seeking the optimum via $\nabla_\theta C(\theta+\mathbf{v}_{\text{opt}})=0$ yields:

$$\mathbf{g}(\theta)+\mathbf{H}(\theta)\mathbf{v}_{\text{opt}}=0$$

Rearranging yields the Newton update rules:

$$\mathbf{v}_t=-\mathbf{H}^{-1}(\theta_t)\mathbf{g}(\theta_t)$$
$$\theta_{t+\tau}=\theta_t+\mathbf{v}_t$$

Often the Hessian is poorly conditioned $\rightarrow$ replace inverse by regularized version, e.g. $[\mathbf{H}(\theta_t)+\varepsilon I]^{-1}$ with small parameter ε.

Gradient descent

Lessons from Newton (2nd order) for gradient descent (1st order)

- Neural networks often use $d > 10^6$ parameters $\rightarrow$ Newtons method unpractical, because calculating (d^2) and inverting (d^3) the Hessian is too expensive.
- But: we can gain insights from the Newton update for gradient descent.
- Consider the one-dimensional quadratic function

$$C(\theta) = \frac{1}{2} k\theta^2 \qquad g(\theta) = k\theta \qquad H(\theta) = k$$

Inserting into the Newton update leads to:

$$\begin{aligned} \theta_{t+\tau} &= \theta_t - H^{-1}(\theta_t)\mathbf{g}(\theta_t) \\ &= \theta_t - k^{-1}\mathbf{g}(\theta_t) \end{aligned}$$

- Compare to gradient descent update $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}(\theta_t) \rightarrow$ learning rate $\eta_t = k^{-1}$. The stiffer the potential (large k), the lower the learning rate must be chosen.

Gradient descent

Lessons from Newton (2nd order) for gradient descent (1st order)

- Neural networks often use $d > 10^6$ parameters $\rightarrow$ Newtons method unpractical, because calculating (d^2) and inverting (d^3) the Hessian is too expensive.
- But: we can gain insights from the Newton update for gradient descent.
- Consider the one-dimensional quadratic function

$$C(\theta) = \frac{1}{2} k \theta^2 \qquad g(\theta) = k\theta \qquad H(\theta) = k$$

Inserting into the Newton update leads to:

$$\begin{aligned} \theta_{t+\tau} &= \theta_t - H^{-1}(\theta_t)\mathbf{g}(\theta_t) \\ &= \theta_t - k^{-1}\mathbf{g}(\theta_t) \end{aligned}$$

- Compare to gradient descent update $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}(\theta_t) \rightarrow$ learning rate $\eta_t = k^{-1}$. The stiffer the potential (large k), the lower the learning rate must be chosen.

- Neural networks often use $d > 10^6$ parameters $\rightarrow$ Newtons method unpractical, because calculating (d^2) and inverting (d^3) the Hessian is too expensive.
- But: we can gain insights from the Newton update for gradient descent.
- Consider the one-dimensional quadratic function

$$C(\theta) = \frac{1}{2} k\theta^2 \qquad g(\theta) = k\theta \qquad H(\theta) = k$$

Inserting into the Newton update leads to:

$$\begin{aligned} \theta_{t+\tau} &= \theta_t - H^{-1}(\theta_t)\mathbf{g}(\theta_t) \\ &= \theta_t - k^{-1}\mathbf{g}(\theta_t) \end{aligned}$$

- Compare to gradient descent update $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}(\theta_t) \rightarrow$ learning rate $\eta_t = k^{-1}$. The stiffer the potential (large k), the lower the learning rate must be chosen.

Gradient descent

Lessons from Newton (2nd order) for gradient descent (1st order)

- Neural networks often use $d > 10^6$ parameters $\rightarrow$ Newtons method unpractical, because calculating (d^2) and inverting (d^3) the Hessian is too expensive.
- But: we can gain insights from the Newton update for gradient descent.
- Consider the one-dimensional quadratic function

$$C(\theta) = \frac{1}{2} k\theta^2 \qquad g(\theta) = k\theta \qquad H(\theta) = k$$

Inserting into the Newton update leads to:

$$\begin{aligned} \theta_{t+\tau} &= \theta_t - H^{-1}(\theta_t)\mathbf{g}(\theta_t) \\ &= \theta_t - k^{-1}\mathbf{g}(\theta_t) \end{aligned}$$

- Compare to gradient descent update $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}(\theta_t) \rightarrow$ learning rate $\eta_t = k^{-1}$. The stiffer the potential (large k), the lower the learning rate must be chosen.

Gradient descent

Lessons from Newton (2nd order) for gradient descent (1st order)

- For multidimensional case, the optimal Newton learning rate is

$$\eta = \sigma_{\max}^{-1}$$

where $\sigma_{\max}$ is the maximal singular value of the Hessian

- Singular value decomposition:

$$\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\top}$$

with U and V are unitary matrices of singular vectors and $\Sigma = \mathrm{diag}(\sigma_1, ..., \sigma_d)$ are the singular values.

- Time to converge close to the minimum (ε-ball around the minimum in a quadradic expansion) is proportional to the condition number:

$$\kappa = \frac{\sigma_{\max}}{\sigma_{\min}},$$

i.e. convergence is slow for anisotropic minima (with steep and shallow directions). *LeCun et al., 1998*.

- For multidimensional case, the optimal Newton learning rate is

$$\eta = \sigma_{\max}^{-1}$$

where $\sigma_{\max}$ is the maximal singular value of the Hessian

- Singular value decomposition:

$$\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\top}$$

with U and V are unitary matrices of singular vectors and $\Sigma = \mathrm{diag}(\sigma_1, ..., \sigma_d)$ are the singular values.

- Time to converge close to the minimum (ε-ball around the minimum in a quadradic expansion) is proportional to the condition number:

$$\kappa = \frac{\sigma_{\max}}{\sigma_{\min}},$$

i.e. convergence is slow for anisotropic minima (with steep and shallow directions). *LeCun et al., 1998*.

- For multidimensional case, the optimal Newton learning rate is

$$\eta = \sigma_{\max}^{-1}$$

where $\sigma_{\max}$ is the maximal singular value of the Hessian

- Singular value decomposition:

$$\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\top}$$

with U and V are unitary matrices of singular vectors and $\Sigma = \mathrm{diag}(\sigma_1, ..., \sigma_d)$ are the singular values.

- Time to converge close to the minimum (ε-ball around the minimum in a quadradic expansion) is proportional to the condition number:

$$\kappa = \frac{\sigma_{\max}}{\sigma_{\min}},$$

i.e. convergence is slow for anisotropic minima (with steep and shallow directions). *LeCun et al., 1998*.

- GD finds only local minima of $C(\theta) \rightarrow$ Optimization may get stuck.
  $\rightarrow$ **Idea**: Use stochastic perturbations.
- GD is sensitive to initial conditions $\rightarrow$ Minimum depends on θ_0.
  $\rightarrow$ **Idea**: Try different θ_0
- Many cost functions are of the form

$$C(\mathbf{X}, \theta) = \sum_{i=1}^{N} c_i(\mathbf{x}_i, \theta)$$

  with data samples $\mathbf{x}_i$. Computing the gradients is expensive, $\mathcal{O}(N)$.
  $\rightarrow$ **Idea**: Compute sample expectation of the gradient.
- GD is very sensitive to learning rate η_t. Small η_t involve high computational cost, large η_t may result in divergence.
  $\rightarrow$ **Idea**: Choose η_t adaptively.
- GD has the same learning rate in all directions. $\rightarrow$ Poor convergence with $\kappa = \sigma_{\max}/\sigma_{\min}$, close to the optimum.
- GD can take exponential time to escape saddle points. Boost methods exist (not treated here)

- GD finds only local minima of $C(\theta) \rightarrow$ Optimization may get stuck.
  $\rightarrow$ **Idea**: Use stochastic perturbations.
- GD is sensitive to initial conditions $\rightarrow$ Minimum depends on θ_0.
  $\rightarrow$ **Idea**: Try different θ_0
- Many cost functions are of the form

$$C(\mathbf{X}, \theta) = \sum_{i=1}^{N} c_i(\mathbf{x}_i, \theta)$$

  with data samples $\mathbf{x}_i$. Computing the gradients is expensive, $\mathcal{O}(N)$.
  $\rightarrow$ **Idea**: Compute sample expectation of the gradient.
- GD is very sensitive to learning rate η_t. Small η_t involve high computational cost, large η_t may result in divergence.
  $\rightarrow$ **Idea**: Choose η_t adaptively.
- GD has the same learning rate in all directions. $\rightarrow$ Poor convergence with $\kappa = \sigma_{\max}/\sigma_{\min}$, close to the optimum.
- GD can take exponential time to escape saddle points. Boost methods exist (not treated here)

- GD finds only local minima of $C(\theta) \rightarrow$ Optimization may get stuck.
  $\rightarrow$ **Idea**: Use stochastic perturbations.
- GD is sensitive to initial conditions $\rightarrow$ Minimum depends on θ_0.
  $\rightarrow$ **Idea**: Try different θ_0
- Many cost functions are of the form

$$C(\mathbf{X}, \theta) = \sum_{i=1}^{N} c_i(\mathbf{x}_i, \theta)$$

  with data samples $\mathbf{x}_i$. Computing the gradients is expensive, $\mathcal{O}(N)$.
  $\rightarrow$ **Idea**: Compute sample expectation of the gradient.
- GD is very sensitive to learning rate η_t. Small η_t involve high computational cost, large η_t may result in divergence.
  $\rightarrow$ **Idea**: Choose η_t adaptively.
- GD has the same learning rate in all directions. $\rightarrow$ Poor convergence with $\kappa = \sigma_{\max}/\sigma_{\min}$, close to the optimum.
- GD can take exponential time to escape saddle points. Boost methods exist (not treated here)

- GD finds only local minima of $C(\theta) \rightarrow$ Optimization may get stuck.
  $\rightarrow$ **Idea**: Use stochastic perturbations.
- GD is sensitive to initial conditions $\rightarrow$ Minimum depends on θ_0.
  $\rightarrow$ **Idea**: Try different θ_0
- Many cost functions are of the form

$$C(\mathbf{X}, \theta) = \sum_{i=1}^{N} c_i(\mathbf{x}_i, \theta)$$

  with data samples $\mathbf{x}_i$. Computing the gradients is expensive, $\mathcal{O}(N)$.
  $\rightarrow$ **Idea**: Compute sample expectation of the gradient.
- GD is very sensitive to learning rate η_t. Small η_t involve high computational cost, large η_t may result in divergence.
  $\rightarrow$ **Idea**: Choose η_t adaptively.
- GD has the same learning rate in all directions. $\rightarrow$ Poor convergence with $\kappa = \sigma_{\max}/\sigma_{\min}$, close to the optimum.
- GD can take exponential time to escape saddle points. Boost methods exist (not treated here)

- GD finds only local minima of $C(\theta) \rightarrow$ Optimization may get stuck.
  $\rightarrow$ **Idea**: Use stochastic perturbations.
- GD is sensitive to initial conditions $\rightarrow$ Minimum depends on θ_0.
  $\rightarrow$ **Idea**: Try different θ_0
- Many cost functions are of the form

$$C(\mathbf{X}, \theta) = \sum_{i=1}^{N} c_i(\mathbf{x}_i, \theta)$$

  with data samples $\mathbf{x}_i$. Computing the gradients is expensive, $\mathcal{O}(N)$.
  $\rightarrow$ **Idea**: Compute sample expectation of the gradient.
- GD is very sensitive to learning rate η_t. Small η_t involve high computational cost, large η_t may result in divergence.
  $\rightarrow$ **Idea**: Choose η_t adaptively.
- GD has the same learning rate in all directions. $\rightarrow$ Poor convergence with $\kappa = \sigma_{\max}/\sigma_{\min}$, close to the optimum.
- GD can take exponential time to escape saddle points. Boost methods exist (not treated here)

- GD finds only local minima of $C(\theta) \rightarrow$ Optimization may get stuck.
  $\rightarrow$ **Idea**: Use stochastic perturbations.
- GD is sensitive to initial conditions $\rightarrow$ Minimum depends on θ_0.
  $\rightarrow$ **Idea**: Try different θ_0
- Many cost functions are of the form

$$C(\mathbf{X}, \theta) = \sum_{i=1}^{N} c_i(\mathbf{x}_i, \theta)$$

  with data samples $\mathbf{x}_i$. Computing the gradients is expensive, $\mathcal{O}(N)$.
  $\rightarrow$ **Idea**: Compute sample expectation of the gradient.
- GD is very sensitive to learning rate η_t. Small η_t involve high computational cost, large η_t may result in divergence.
  $\rightarrow$ **Idea**: Choose η_t adaptively.
- GD has the same learning rate in all directions. $\rightarrow$ Poor convergence with $\kappa = \sigma_{\max}/\sigma_{\min}$, close to the optimum.
- GD can take exponential time to escape saddle points. Boost methods exist (not treated here)

Stochastic Gradient Descent

Speeding up gradient descent (Bottou, 2012; Williams and Hinton, 1986)

Many cost functions and their gradients are of the form

$$C(\mathbf{X}, \theta) = \frac{1}{N} \sum_{i=1}^{N} c_i(\mathbf{x}_i, \theta) \qquad \nabla_\theta C(\mathbf{X}, \theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta c_i(\mathbf{x}_i, \theta)$$

with data samples $\mathbf{x}_i$. Computing such gradients is expensive, involving $\mathscr{O}(N)$ operations, where N may be millions.

Idea: rewrite gradient as an expectation $\mathbb{E}_{\mathbf{x}}$:

$$\nabla_\theta C(\mathbf{X}, \theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta c_i(\mathbf{x}_i, \theta) \approx \mathbb{E}_{\mathbf{x}}[\nabla_\theta c(\mathbf{x}, \theta)]$$

Subdivide the N-sample average into M averages over n samples each. Call index sets B_m:

$$\nabla_\theta C(\mathbf{X}, \theta) = \frac{1}{M} \sum_{m=1}^{M} \frac{1}{n} \sum_{i \in B_m} \nabla_\theta c_i(\mathbf{x}_i, \theta) \approx \mathbb{E}_{\mathbf{x}}[\nabla_\theta c(\mathbf{x}, \theta)]$$

Stochastic Gradient Descent

Speeding up gradient descent (Bottou, 2012; Williams and Hinton, 1986)

Many cost functions and their gradients are of the form

$$C(\mathbf{X}, \theta) = \frac{1}{N} \sum_{i=1}^{N} c_i(\mathbf{x}_i, \theta) \qquad \nabla_\theta C(\mathbf{X}, \theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta c_i(\mathbf{x}_i, \theta)$$

with data samples $\mathbf{x}_i$. Computing such gradients is expensive, involving $\mathscr{O}(N)$ operations, where N may be millions.

Idea: rewrite gradient as an expectation $\mathbb{E}_{\mathbf{x}}$:

$$\nabla_\theta C(\mathbf{X}, \theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta c_i(\mathbf{x}_i, \theta) \approx \mathbb{E}_{\mathbf{x}} [\nabla_\theta c(\mathbf{x}, \theta)]$$

Subdivide the N-sample average into M averages over n samples each. Call index sets B_m:

$$\nabla_\theta C(\mathbf{X}, \theta) = \frac{1}{M} \sum_{m=1}^{M} \frac{1}{n} \sum_{i \in B_m} \nabla_\theta c_i(\mathbf{x}_i, \theta) \approx \mathbb{E}_{\mathbf{x}} [\nabla_\theta c(\mathbf{x}, \theta)]$$

Stochastic Gradient Descent

Speeding up gradient descent (Bottou, 2012; Williams and Hinton, 1986)

Many cost functions and their gradients are of the form

$$C(\mathbf{X}, \theta) = \frac{1}{N} \sum_{i=1}^{N} c_i(\mathbf{x}_i, \theta) \qquad \nabla_\theta C(\mathbf{X}, \theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta c_i(\mathbf{x}_i, \theta)$$

with data samples $\mathbf{x}_i$. Computing such gradients is expensive, involving $\mathscr{O}(N)$ operations, where N may be millions.

Idea: rewrite gradient as an expectation $\mathbb{E}_{\mathbf{x}}$:

$$\nabla_\theta C(\mathbf{X}, \theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta c_i(\mathbf{x}_i, \theta) \approx \mathbb{E}_{\mathbf{x}} [\nabla_\theta c(\mathbf{x}, \theta)]$$

Subdivide the N-sample average into M averages over n samples each. Call index sets B_m:

$$\nabla_\theta C(\mathbf{X}, \theta) = \frac{1}{M} \sum_{m=1}^{M} \frac{1}{n} \sum_{i \in B_m} \nabla_\theta c_i(\mathbf{x}_i, \theta) \approx \mathbb{E}_{\mathbf{x}} [\nabla_\theta c(\mathbf{x}, \theta)]$$

Ignore constant M (we can incorporate it into learning rate η_t). Change gradient descent algorithm as follows:

Stochastic gradient descent

1. Initialize θ_0
2. For *epoch* $e = 1, ..., E$ or until converged:
   1. For *minibatch* $m = 1, ..., M$:
      1. Compute minibatch gradient $\mathbf{g}_m(\theta) = \frac{1}{n} \sum_{i \in B_m} \nabla_\theta c_i(\mathbf{x}_i, \theta)$
      2. Update parameters: $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}_m(\theta_t)$

- SGD replaces the gradient over the full data by an approximation to the gradient computed using a minibatch.
- Introduces stochasticity and decreases probability of getting stuck in local minima (Bishop, 1995a; Keskar et al., 2016).
- Calculation in each step is much cheaper ($M \ll N$ evaluations). In practice, also much less total evaluations are needed in total to achieve similar loss values.

Ignore constant M (we can incorporate it into learning rate η_t). Change gradient descent algorithm as follows:

Stochastic gradient descent

1. Initialize θ_0
2. For *epoch* $e = 1, ..., E$ or until converged:
   1. For *minibatch* $m = 1, ..., M$:
      1. Compute minibatch gradient $\mathbf{g}_m(\theta) = \frac{1}{n} \sum_{i \in B_m} \nabla_\theta c_i(\mathbf{x}_i, \theta)$
      2. Update parameters: $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}_m(\theta_t)$

- SGD replaces the gradient over the full data by an approximation to the gradient computed using a minibatch.
- Introduces stochasticity and decreases probability of getting stuck in local minima (Bishop, 1995a; Keskar et al., 2016).
- Calculation in each step is much cheaper ($M \ll N$ evaluations). In practice, also much less total evaluations are needed in total to achieve similar loss values.

Ignore constant M (we can incorporate it into learning rate η_t). Change gradient descent algorithm as follows:

Stochastic gradient descent

1. Initialize θ_0
2. For *epoch* $e = 1, ..., E$ or until converged:
   1. For *minibatch* $m = 1, ..., M$:
      1. Compute minibatch gradient $\mathbf{g}_m(\theta) = \frac{1}{n} \sum_{i \in B_m} \nabla_\theta c_i(\mathbf{x}_i, \theta)$
      2. Update parameters: $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}_m(\theta_t)$

- SGD replaces the gradient over the full data by an approximation to the gradient computed using a minibatch.
- Introduces stochasticity and decreases probability of getting stuck in local minima (Bishop, 1995a; Keskar et al., 2016).
- Calculation in each step is much cheaper ($M \ll N$ evaluations). In practice, also much less total evaluations are needed in total to achieve similar loss values.

Ignore constant M (we can incorporate it into learning rate η_t). Change gradient descent algorithm as follows:

Stochastic gradient descent

1. Initialize θ_0
2. For *epoch* $e = 1, ..., E$ or until converged:
   1. For *minibatch* $m = 1, ..., M$:
      1. Compute minibatch gradient $\mathbf{g}_m(\theta) = \frac{1}{n} \sum_{i \in B_m} \nabla_\theta c_i(\mathbf{x}_i, \theta)$
      2. Update parameters: $\theta_{t+1} = \theta_t - \eta_t \mathbf{g}_m(\theta_t)$

- SGD replaces the gradient over the full data by an approximation to the gradient computed using a minibatch.
- Introduces stochasticity and decreases probability of getting stuck in local minima (Bishop, 1995a; Keskar et al., 2016).
- Calculation in each step is much cheaper ($M \ll N$ evaluations). In practice, also much less total evaluations are needed in total to achieve similar loss values.

"Momentum" term: Memory of the direction we are moving.

SGD update with Momentum

Using the momentum parameter $0 \leq \gamma \leq 1$, we can implement SGD as:

$$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \eta_t \nabla_\theta C(\theta_t)$$
$$\theta_{t+1} = \theta_t - \mathbf{v}_t$$

or, equivalently, written with parameter updates $\Delta\theta_t = \theta_t - \theta_{t-1}$:

$$\Delta\theta_{t+1} = \gamma \Delta\theta_t - \eta_t \mathbf{g}(\theta_t)$$

- $\mathbf{v}_t$ is running average of recent gradients
- $(1-\gamma)^{-1}$ is characteristic time scale of the averaging memory.
- Momentum helps SGD to gain speed in directions with persistent gradients and suppresses oscillations in high curvature directions. Useful when $C(\theta_t)$ has flat and steep directions.

"Momentum" term: Memory of the direction we are moving.

SGD update with Momentum

Using the momentum parameter $0 \leq \gamma \leq 1$, we can implement SGD as:

$$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \eta_t \nabla_\theta C(\theta_t)$$
$$\theta_{t+1} = \theta_t - \mathbf{v}_t$$

or, equivalently, written with parameter updates $\Delta\theta_t = \theta_t - \theta_{t-1}$:

$$\Delta\theta_{t+1} = \gamma \Delta\theta_t - \eta_t \mathbf{g}(\theta_t)$$

- $\mathbf{v}_t$ is running average of recent gradients
- $(1-\gamma)^{-1}$ is characteristic time scale of the averaging memory.
- Momentum helps SGD to gain speed in directions with persistent gradients and suppresses oscillations in high curvature directions. Useful when $C(\theta_t)$ has flat and steep directions.

"Momentum" term: Memory of the direction we are moving.

SGD update with Momentum

Using the momentum parameter $0 \leq \gamma \leq 1$, we can implement SGD as:

$$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \eta_t \nabla_\theta C(\theta_t)$$
$$\theta_{t+1} = \theta_t - \mathbf{v}_t$$

or, equivalently, written with parameter updates $\Delta\theta_t = \theta_t - \theta_{t-1}$:

$$\Delta\theta_{t+1} = \gamma \Delta\theta_t - \eta_t \mathbf{g}(\theta_t)$$

- $\mathbf{v}_t$ is running average of recent gradients
- $(1-\gamma)^{-1}$ is characteristic time scale of the averaging memory.
- Momentum helps SGD to gain speed in directions with persistent gradients and suppresses oscillations in high curvature directions. Useful when $C(\theta_t)$ has flat and steep directions.

SGD with momentum is equivalent to numerically solving a dynamical equation of a particle with mass m and position θ moving in a viscous medium under the dimensionless potential $C(\theta)$:

$$\underbrace{m\frac{d^2\theta}{dt^2}}_{\text{total force}} = \underbrace{-\nabla_\theta C(\theta)}_{\text{force from potential}} \underbrace{-\mu\frac{d\theta}{dt}}_{\text{drag force}}. \tag{1}$$

Derivation: Finite differences $\frac{d\theta}{dt} = \frac{\theta_{t+\Delta t}-\theta_t}{\Delta t}$, $\frac{d^2\theta}{dt^2} = \frac{\theta_{t+\Delta t}-2\theta_t+\theta_{t-\Delta t}}{(\Delta t)^2}$. Define:

$$\gamma = \frac{m}{m+\mu\Delta t}, \qquad \eta = \frac{(\Delta t)^2}{m+\mu\Delta t}$$

In the limit of small learning rate, the momentum memory time scales as

$$(1-\gamma)^{-1} \approx m/\mu\Delta t.$$

Langevin form: When approximating the gradient $\nabla_\theta C(\theta)$ with minibatches, it can be written as a deterministic part (gradient in the limit $M \to N$), plus a noise term depending on minibatch size. Then, Eq. (1) is a Langevin equation.

SGD with momentum is equivalent to numerically solving a dynamical equation of a particle with mass m and position θ moving in a viscous medium under the dimensionless potential $C(\theta)$:

$$\underbrace{m\frac{d^2\theta}{dt^2}}_{\text{total force}} = \underbrace{-\nabla_\theta C(\theta)}_{\text{force from potential}} \underbrace{-\mu\frac{d\theta}{dt}}_{\text{drag force}}. \tag{1}$$

Derivation: Finite differences $\frac{d\theta}{dt} = \frac{\theta_{t+\Delta t}-\theta_t}{\Delta t}$, $\frac{d^2\theta}{dt^2} = \frac{\theta_{t+\Delta t}-2\theta_t+\theta_{t-\Delta t}}{(\Delta t)^2}$.
Define:

$$\gamma = \frac{m}{m+\mu\Delta t}, \qquad \eta = \frac{(\Delta t)^2}{m+\mu\Delta t}$$

In the limit of small learning rate, the momentum memory time scales as

$$(1-\gamma)^{-1} \approx m/\mu\Delta t.$$

Langevin form: When approximating the gradient $\nabla_\theta C(\theta)$ with minibatches, it can be written as a deterministic part (gradient in the limit $M \to N$), plus a noise term depending on minibatch size. Then, Eq. (1) is a Langevin equation.

SGD with momentum is equivalent to numerically solving a dynamical equation of a particle with mass m and position θ moving in a viscous medium under the dimensionless potential $C(\theta)$:

$$\underbrace{m\frac{d^2\theta}{dt^2}}_{\text{total force}} = \underbrace{-\nabla_\theta C(\theta)}_{\text{force from potential}} \underbrace{-\mu\frac{d\theta}{dt}}_{\text{drag force}}. \tag{1}$$

Derivation: Finite differences $\frac{d\theta}{dt} = \frac{\theta_{t+\Delta t}-\theta_t}{\Delta t}$, $\frac{d^2\theta}{dt^2} = \frac{\theta_{t+\Delta t}-2\theta_t+\theta_{t-\Delta t}}{(\Delta t)^2}$.
Define:

$$\gamma = \frac{m}{m+\mu\Delta t}, \qquad \eta = \frac{(\Delta t)^2}{m+\mu\Delta t}$$

In the limit of small learning rate, the momentum memory time scales as

$$(1-\gamma)^{-1} \approx m/\mu\Delta t.$$

Langevin form: When approximating the gradient $\nabla_\theta C(\theta)$ with minibatches, it can be written as a deterministic part (gradient in the limit $M \to N$), plus a noise term depending on minibatch size. Then, Eq. (1) is a Langevin equation.

SGD with momentum is equivalent to numerically solving a dynamical equation of a particle with mass m and position θ moving in a viscous medium under the dimensionless potential $C(\theta)$:

$$\underbrace{m\frac{d^2\theta}{dt^2}}_{\text{total force}} = \underbrace{-\nabla_\theta C(\theta)}_{\text{force from potential}} \underbrace{-\mu\frac{d\theta}{dt}}_{\text{drag force}}. \tag{1}$$

Derivation: Finite differences $\frac{d\theta}{dt} = \frac{\theta_{t+\Delta t}-\theta_t}{\Delta t}$, $\frac{d^2\theta}{dt^2} = \frac{\theta_{t+\Delta t}-2\theta_t+\theta_{t-\Delta t}}{(\Delta t)^2}$.
Define:

$$\gamma = \frac{m}{m+\mu\Delta t}, \qquad \eta = \frac{(\Delta t)^2}{m+\mu\Delta t}$$

In the limit of small learning rate, the momentum memory time scales as

$$(1-\gamma)^{-1} \approx m/\mu\Delta t.$$

Langevin form: When approximating the gradient $\nabla_\theta C(\theta)$ with minibatches, it can be written as a deterministic part (gradient in the limit $M \to N$), plus a noise term depending on minibatch size. Then, Eq. (1) is a Langevin equation.

- Desirable to adapt the learning rate by taking large steps in shallow, flat directions
- Second-order methods do this by computing the Hessian matrix, but this is computationally expensive.
- Alternative: methods that track not only the gradient but also its second moment.
- Examples:
  - AdaGrad (Duchi et al., 2011)
  - AdaDelta (Zeiler, 2012)
  - RMS-Prop (Tieleman and Hinton, 2012)
  - ADAM (Kingma and Ba, 2014).

- Desirable to adapt the learning rate by taking large steps in shallow, flat directions
- Second-order methods do this by computing the Hessian matrix, but this is computationally expensive.
- Alternative: methods that track not only the gradient but also its second moment.
- Examples:
  - AdaGrad (Duchi et al., 2011)
  - AdaDelta (Zeiler, 2012)
  - RMS-Prop (Tieleman and Hinton, 2012)
  - ADAM (Kingma and Ba, 2014).

- Desirable to adapt the learning rate by taking large steps in shallow, flat directions
- Second-order methods do this by computing the Hessian matrix, but this is computationally expensive.
- Alternative: methods that track not only the gradient but also its second moment.
- Examples:
  - AdaGrad (Duchi et al., 2011)
  - AdaDelta (Zeiler, 2012)
  - RMS-Prop (Tieleman and Hinton, 2012)
  - ADAM (Kingma and Ba, 2014).

- Desirable to adapt the learning rate by taking large steps in shallow, flat directions
- Second-order methods do this by computing the Hessian matrix, but this is computationally expensive.
- Alternative: methods that track not only the gradient but also its second moment.
- Examples:
  - AdaGrad (Duchi et al., 2011)
  - AdaDelta (Zeiler, 2012)
  - RMS-Prop (Tieleman and Hinton, 2012)
  - ADAM (Kingma and Ba, 2014).

RMS prop

Methods using the second moment of the gradient

Idea: keep track of the second moment denoted by $\mathbf{s}_t = \mathbb{E}\left[\mathbf{g}_t^2\right]$

RMS prop update

$$\mathbf{g}_t = \nabla_\theta C(\theta_t)$$
$$\mathbf{s}_t = \beta \mathbf{s}_{t-1} + (1-\beta)\mathbf{g}_t^2$$
$$\theta_{t+1} = \theta_t - \eta_t \frac{\mathbf{g}_t}{\sqrt{\mathbf{s}_t} + \varepsilon}$$

where $\mathbf{g}_t/(\sqrt{\mathbf{s}_t} + \varepsilon)$ and $\mathbf{g}_t^2$ are element-wise operations.

- β controls the averaging time of second moment. Typically $\beta = 0.9$.
- η_t is the learning rate, typically $\eta_t = 10^{-3}$
- ε is a small regularization constant, typically $\varepsilon = 10^{-8}$.
- Learning rate is reduced/increased in directions when gradient fluctuates much/little.
- Speeds up convergence by larger learning rates in flat directions.

RMS prop

Methods using the second moment of the gradient

Idea: keep track of the second moment denoted by $\mathbf{s}_t = \mathbb{E}\left[\mathbf{g}_t^2\right]$

RMS prop update

$$\mathbf{g}_t = \nabla_\theta C(\theta_t)$$
$$\mathbf{s}_t = \beta \mathbf{s}_{t-1} + (1-\beta)\mathbf{g}_t^2$$
$$\theta_{t+1} = \theta_t - \eta_t \frac{\mathbf{g}_t}{\sqrt{\mathbf{s}_t} + \varepsilon}$$

where $\mathbf{g}_t/(\sqrt{\mathbf{s}_t} + \varepsilon)$ and $\mathbf{g}_t^2$ are element-wise operations.

- β controls the averaging time of second moment. Typically $\beta = 0.9$.
- η_t is the learning rate, typically $\eta_t = 10^{-3}$
- ε is a small regularization constant, typically $\varepsilon = 10^{-8}$.
- Learning rate is reduced/increased in directions when gradient fluctuates much/little.
- Speeds up convergence by larger learning rates in flat directions.

RMS prop

Methods using the second moment of the gradient

Idea: keep track of the second moment denoted by $\mathbf{s}_t = \mathbb{E}\left[\mathbf{g}_t^2\right]$

RMS prop update

$$\mathbf{g}_t = \nabla_\theta C(\theta_t)$$
$$\mathbf{s}_t = \beta \mathbf{s}_{t-1} + (1-\beta)\mathbf{g}_t^2$$
$$\theta_{t+1} = \theta_t - \eta_t \frac{\mathbf{g}_t}{\sqrt{\mathbf{s}_t} + \varepsilon}$$

where $\mathbf{g}_t/(\sqrt{\mathbf{s}_t} + \varepsilon)$ and $\mathbf{g}_t^2$ are element-wise operations.

- β controls the averaging time of second moment. Typically $\beta = 0.9$.
- η_t is the learning rate, typically $\eta_t = 10^{-3}$
- ε is a small regularization constant, typically $\varepsilon = 10^{-8}$.
- Learning rate is reduced/increased in directions when gradient fluctuates much/little.
- Speeds up convergence by larger learning rates in flat directions.

RMS prop

Methods using the second moment of the gradient

Idea: keep track of the second moment denoted by $\mathbf{s}_t = \mathbb{E}\left[\mathbf{g}_t^2\right]$

RMS prop update

$$\mathbf{g}_t = \nabla_\theta C(\theta_t)$$
$$\mathbf{s}_t = \beta \mathbf{s}_{t-1} + (1-\beta)\mathbf{g}_t^2$$
$$\theta_{t+1} = \theta_t - \eta_t \frac{\mathbf{g}_t}{\sqrt{\mathbf{s}_t} + \varepsilon}$$

where $\mathbf{g}_t/(\sqrt{\mathbf{s}_t} + \varepsilon)$ and $\mathbf{g}_t^2$ are element-wise operations.

- β controls the averaging time of second moment. Typically $\beta = 0.9$.
- η_t is the learning rate, typically $\eta_t = 10^{-3}$
- ε is a small regularization constant, typically $\varepsilon = 10^{-8}$.
- Learning rate is reduced/increased in directions when gradient fluctuates much/little.
- Speeds up convergence by larger learning rates in flat directions.

RMS prop

Methods using the second moment of the gradient

Idea: keep track of the second moment denoted by $\mathbf{s}_t = \mathbb{E}\left[\mathbf{g}_t^2\right]$

RMS prop update

$$\mathbf{g}_t = \nabla_\theta C(\theta_t)$$
$$\mathbf{s}_t = \beta \mathbf{s}_{t-1} + (1-\beta)\mathbf{g}_t^2$$
$$\theta_{t+1} = \theta_t - \eta_t \frac{\mathbf{g}_t}{\sqrt{\mathbf{s}_t} + \varepsilon}$$

where $\mathbf{g}_t/(\sqrt{\mathbf{s}_t} + \varepsilon)$ and $\mathbf{g}_t^2$ are element-wise operations.

- β controls the averaging time of second moment. Typically $\beta = 0.9$.
- η_t is the learning rate, typically $\eta_t = 10^{-3}$
- ε is a small regularization constant, typically $\varepsilon = 10^{-8}$.
- Learning rate is reduced/increased in directions when gradient fluctuates much/little.
- Speeds up convergence by larger learning rates in flat directions.

RMS prop

Methods using the second moment of the gradient

Idea: keep track of the second moment denoted by $\mathbf{s}_t = \mathbb{E}\left[\mathbf{g}_t^2\right]$

RMS prop update

$$\mathbf{g}_t = \nabla_\theta C(\theta_t)$$
$$\mathbf{s}_t = \beta \mathbf{s}_{t-1} + (1-\beta)\mathbf{g}_t^2$$
$$\theta_{t+1} = \theta_t - \eta_t \frac{\mathbf{g}_t}{\sqrt{\mathbf{s}_t} + \varepsilon}$$

where $\mathbf{g}_t/(\sqrt{\mathbf{s}_t} + \varepsilon)$ and $\mathbf{g}_t^2$ are element-wise operations.

- β controls the averaging time of second moment. Typically $\beta = 0.9$.
- η_t is the learning rate, typically $\eta_t = 10^{-3}$
- ε is a small regularization constant, typically $\varepsilon = 10^{-8}$.
- Learning rate is reduced/increased in directions when gradient fluctuates much/little.
- Speeds up convergence by larger learning rates in flat directions.

Idea: running average of 1^{st} and 2^{nd} moment of gradient, $\mathbf{m}_t = \mathbb{E}[\mathbf{g}_t]$ and $\mathbf{s}_t = \mathbb{E}\left[\mathbf{g}_t^2\right]$ and correct for bias of using running averages.

ADAM update

Gradient and moment update:

$$\mathbf{g}_t = \nabla_\theta C(\theta_t)$$
$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t$$
$$\mathbf{s}_t = \beta_2 \mathbf{s}_{t-1} + (1-\beta_2)\mathbf{g}_t^2$$

Bias correction for moments:

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

Parameter update:

$$\theta_{t+1} = \theta_t - \eta_t \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \varepsilon}$$

where $\hat{\mathbf{m}}_t/(\sqrt{\hat{\mathbf{s}}_t} + \varepsilon)$ and $\mathbf{g}_t^2$ are element-wise operations.

- $\beta_1 \sim 0.9$ and $\beta_2 \sim 0.99$ set the memory lifetime of the first and second moment.
- η_t and ε are equivalent to the RMSprop parameters.
- Learning rate is reduced in directions where the norm of the gradient is consistently large.
- Greatly speeds up the convergence by allowing us to use a larger learning rate for flat directions.

- $\beta_1 \sim 0.9$ and $\beta_2 \sim 0.99$ set the memory lifetime of the first and second moment.
- η_t and ε are equivalent to the RMSprop parameters.
- Learning rate is reduced in directions where the norm of the gradient is consistently large.
- Greatly speeds up the convergence by allowing us to use a larger learning rate for flat directions.

- $\beta_1 \sim 0.9$ and $\beta_2 \sim 0.99$ set the memory lifetime of the first and second moment.
- η_t and ε are equivalent to the RMSprop parameters.
- Learning rate is reduced in directions where the norm of the gradient is consistently large.
- Greatly speeds up the convergence by allowing us to use a larger learning rate for flat directions.

Interpretation: Consider the update of a single parameter θ_t and rewrite the second moment in terms of the variance:

$$\sigma_t^2 = \hat{s}_t - \hat{m}_t^2.$$

The update rule is then:

$$\theta_{t+1} = \theta_t - \eta_t \frac{\hat{m}_t}{\sqrt{\sigma_t^2 + \hat{m}_t^2} + \varepsilon}$$

Consider limiting cases:

1. Gradient estimates are consistent with a small variance:
   1. Update rule tends to $\Delta\theta_{t+1} \to -\eta_t$.
   2. Cuts off large persistent gradients at 1 and limit the maximum step size in steep directions.
2. Gradient fluctuates strongly between gradient descent steps:
   1. $\sigma^2 \gg \hat{m}_t^2$ and update becomes $\Delta\theta_{t+1} \to -\eta_t \frac{\hat{m}_t}{\sigma_t}$
   2. Makes learning rate equal to signal-to-noise ratio.

Interpretation: Consider the update of a single parameter θ_t and rewrite the second moment in terms of the variance:

$$\sigma_t^2 = \hat{s}_t - \hat{m}_t^2.$$

The update rule is then:

$$\theta_{t+1} = \theta_t - \eta_t \frac{\hat{m}_t}{\sqrt{\sigma_t^2 + \hat{m}_t^2} + \varepsilon}$$

Consider limiting cases:

1. Gradient estimates are consistent with a small variance:
   1. Update rule tends to $\Delta\theta_{t+1} \to -\eta_t$.
   2. Cuts off large persistent gradients at 1 and limit the maximum step size in steep directions.
2. Gradient fluctuates strongly between gradient descent steps:
   1. $\sigma^2 \gg \hat{m}_t^2$ and update becomes $\Delta\theta_{t+1} \to -\eta_t \frac{\hat{m}_t}{\sigma_t}$
   2. Makes learning rate equal to signal-to-noise ratio.

Interpretation: Consider the update of a single parameter θ_t and rewrite the second moment in terms of the variance:

$$\sigma_t^2 = \hat{s}_t - \hat{m}_t^2.$$

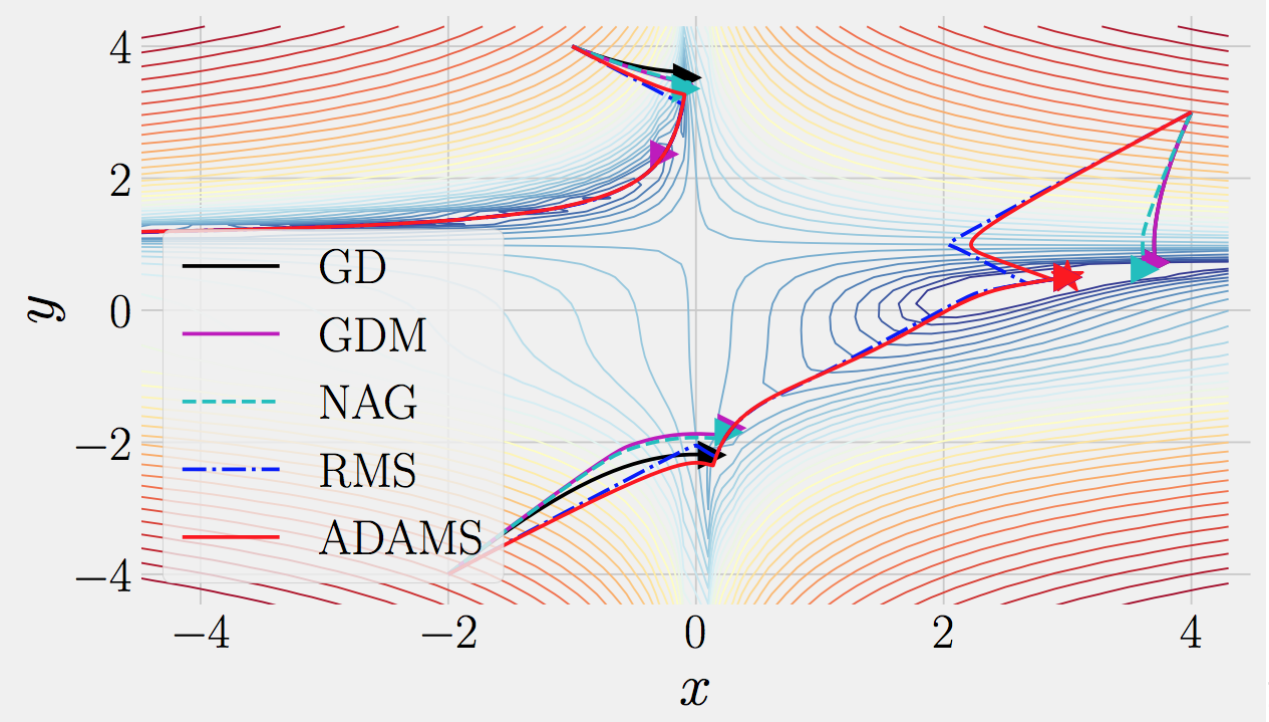
The update rule is then:

$$\theta_{t+1} = \theta_t - \eta_t \frac{\hat{m}_t}{\sqrt{\sigma_t^2 + \hat{m}_t^2} + \varepsilon}$$

Consider limiting cases:

1. Gradient estimates are consistent with a small variance:
   1. Update rule tends to $\Delta\theta_{t+1} \to -\eta_t$.
   2. Cuts off large persistent gradients at 1 and limit the maximum step size in steep directions.
2. Gradient fluctuates strongly between gradient descent steps:
   1. $\sigma^2 \gg \hat{m}_t^2$ and update becomes $\Delta\theta_{t+1} \to -\eta_t \frac{\hat{m}_t}{\sigma_t}$
   2. Makes learning rate equal to signal-to-noise ratio.

Comparison



[1]

Comparison of gradient descent methods (10^4 steps) for Beale's function.

- Gradient descent (GD; black line), $\eta = 10^{-6}$
- Gradient descent with momentum (GDM; magenta line), $\eta = 10^{-6}$
- NAG (cyan-dashed line), $\eta = 10^{-6}$
- RMSprop (blue dash-dot line), $\eta = 10^{-3}$, $\beta = 0.9$, $\varepsilon = 10^{-8}$
- ADAM (red line), $\eta = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\varepsilon = 10^{-8}$.

[1] From Mehta et al, arXiv:1803.08823v1

Best practices

1. *Randomize (shuffle) the data when making mini-batches*, to avoid spurious correlations from the data sequence.
2. *Transform inputs to avoid mix of steep and shallow directions*. A simple trick is to subtracting the mean and normalizing the variance (e.g. using a batch normalization layer). Whenever possible, also decorrelate the inputs (PCA, whitening).
3. *Monitor the out-of-sample performance (validation set)*. If the validation error starts increasing, then the model is beginning to overfit.
4. Adaptive optimization methods (ADAM, RMSprop) don't always have good generalization compared to SGD or SGD with momentum, particularly when the number of parameters exceeds the number of data points (Wilson et al., 2017).

See: Bottou, 2012; LeCun et al., 1998b; Tieleman and Hinton, 2012.

Best practices

1. *Randomize (shuffle) the data when making mini-batches*, to avoid spurious correlations from the data sequence.
2. *Transform inputs to avoid mix of steep and shallow directions*. A simple trick is to subtracting the mean and normalizing the variance (e.g. using a batch normalization layer). Whenever possible, also decorrelate the inputs (PCA, whitening).
3. *Monitor the out-of-sample performance (validation set)*. If the validation error starts increasing, then the model is beginning to overfit.
4. Adaptive optimization methods (ADAM, RMSprop) don't always have good generalization compared to SGD or SGD with momentum, particularly when the number of parameters exceeds the number of data points (Wilson et al., 2017).

See: Bottou, 2012; LeCun et al., 1998b; Tieleman and Hinton, 2012.

Best practices

1. *Randomize (shuffle) the data when making mini-batches*, to avoid spurious correlations from the data sequence.
2. *Transform inputs to avoid mix of steep and shallow directions*. A simple trick is to subtracting the mean and normalizing the variance (e.g. using a batch normalization layer). Whenever possible, also decorrelate the inputs (PCA, whitening).
3. *Monitor the out-of-sample performance (validation set)*. If the validation error starts increasing, then the model is beginning to overfit.
4. Adaptive optimization methods (ADAM, RMSprop) don't always have good generalization compared to SGD or SGD with momentum, particularly when the number of parameters exceeds the number of data points (Wilson et al., 2017).

See: Bottou, 2012; LeCun et al., 1998b; Tieleman and Hinton, 2012.

Best practices

1. *Randomize (shuffle) the data when making mini-batches*, to avoid spurious correlations from the data sequence.
2. *Transform inputs to avoid mix of steep and shallow directions*. A simple trick is to subtracting the mean and normalizing the variance (e.g. using a batch normalization layer). Whenever possible, also decorrelate the inputs (PCA, whitening).
3. *Monitor the out-of-sample performance (validation set)*. If the validation error starts increasing, then the model is beginning to overfit.
4. Adaptive optimization methods (ADAM, RMSprop) don't always have good generalization compared to SGD or SGD with momentum, particularly when the number of parameters exceeds the number of data points (Wilson et al., 2017).

See: Bottou, 2012; LeCun et al., 1998b; Tieleman and Hinton, 2012.