

Machine Learning & Pattern Recognition

SONG Xuemeng


sxmustc@gmail.com

<http://xuemeng.bitcron.com/>

Iterative Optimization Technique

Optimization Methods

- Optimization: either minimize or maximize some function $f(x)$ by altering x .
- In most cases, optimization refers to the minimization of $f(x)$.

Maximization $f(x)$  **Minimization** $-f(x)$

- $f(x)$: objective function, cost function, loss function, error function.
- The value that minimize $f(x)$: $x^* = \arg \min f(x)$.

Optimization Methods

- **Deterministic Optimization**
 - The data for the given problem are known accurately.
- **Stochastic Optimization**
 - Refers to a collection of methods for minimizing or maximizing an objective function when randomness is present.

Deterministic Optimization

- First-order methods: methods that use only the gradient.
- Second-order methods: methods that also use the Hessian matrix.

$$\mathbf{H}(f)_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\mathbf{x})$$

\mathbf{x} : multiple input dimensions.

Gradient Descent [Cauchy 1847]

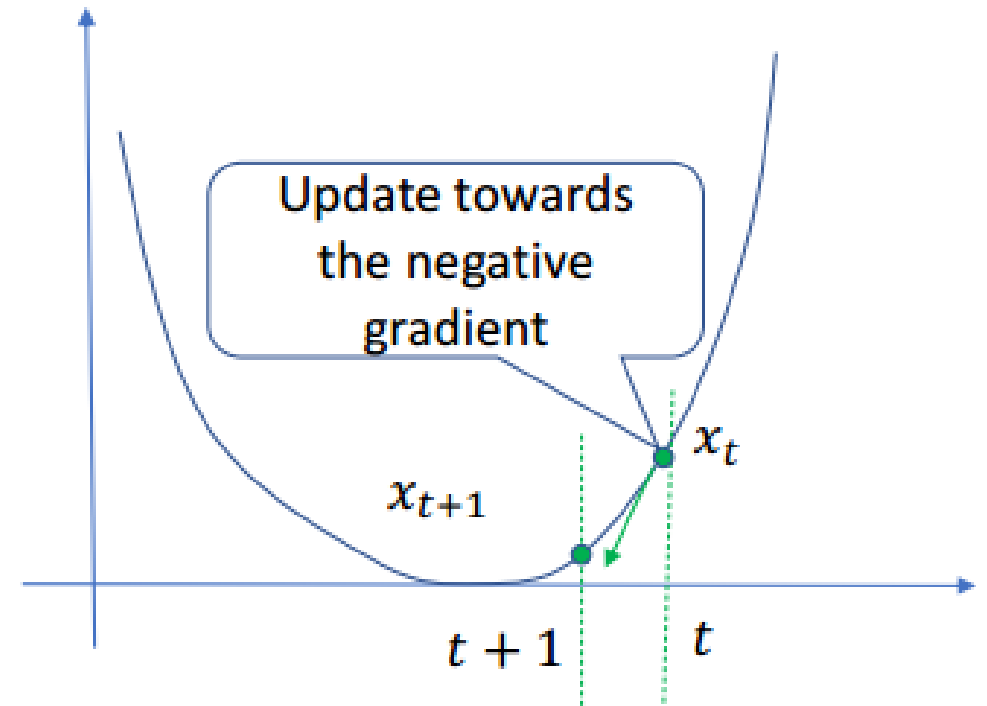
- Motivation: to **minimize** the local **first-order Taylor approximation** of f

$$\min_x f(x) \approx \min_x f(x_t) + \nabla f(x_t)^T (x - x_t)$$

- Update rule:

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

Where $\eta_t > 0$ is the step-size (learning rate).



Interpretation1

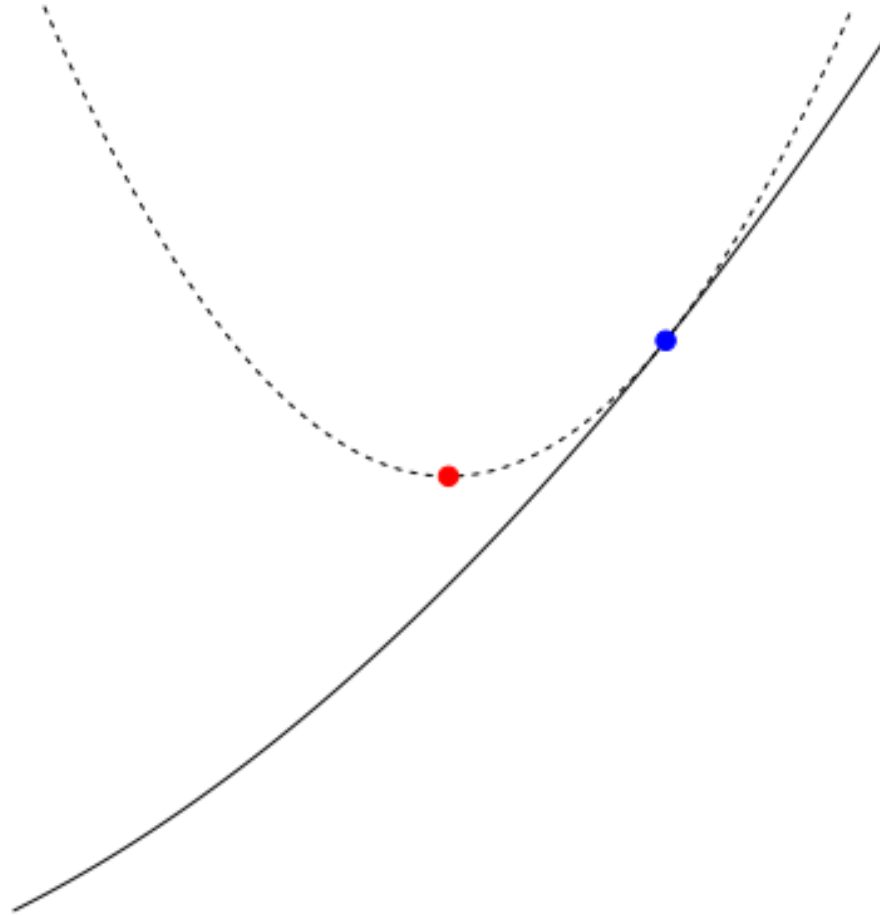
- At each iteration, consider the expansion

$$f(y) \approx \underbrace{f(x_t) + \nabla f(x_t)^T (y - x_t)}_{\text{Linear approximation of } f} + \underbrace{\frac{1}{2\eta_t} \|y - x_t\|^2}_{\text{Proximity term with weight } \frac{1}{2\eta_t}}$$

- Quadratic approximation, replacing usual $\nabla^2 f(x)$ by $\frac{1}{\eta_t} I$:

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

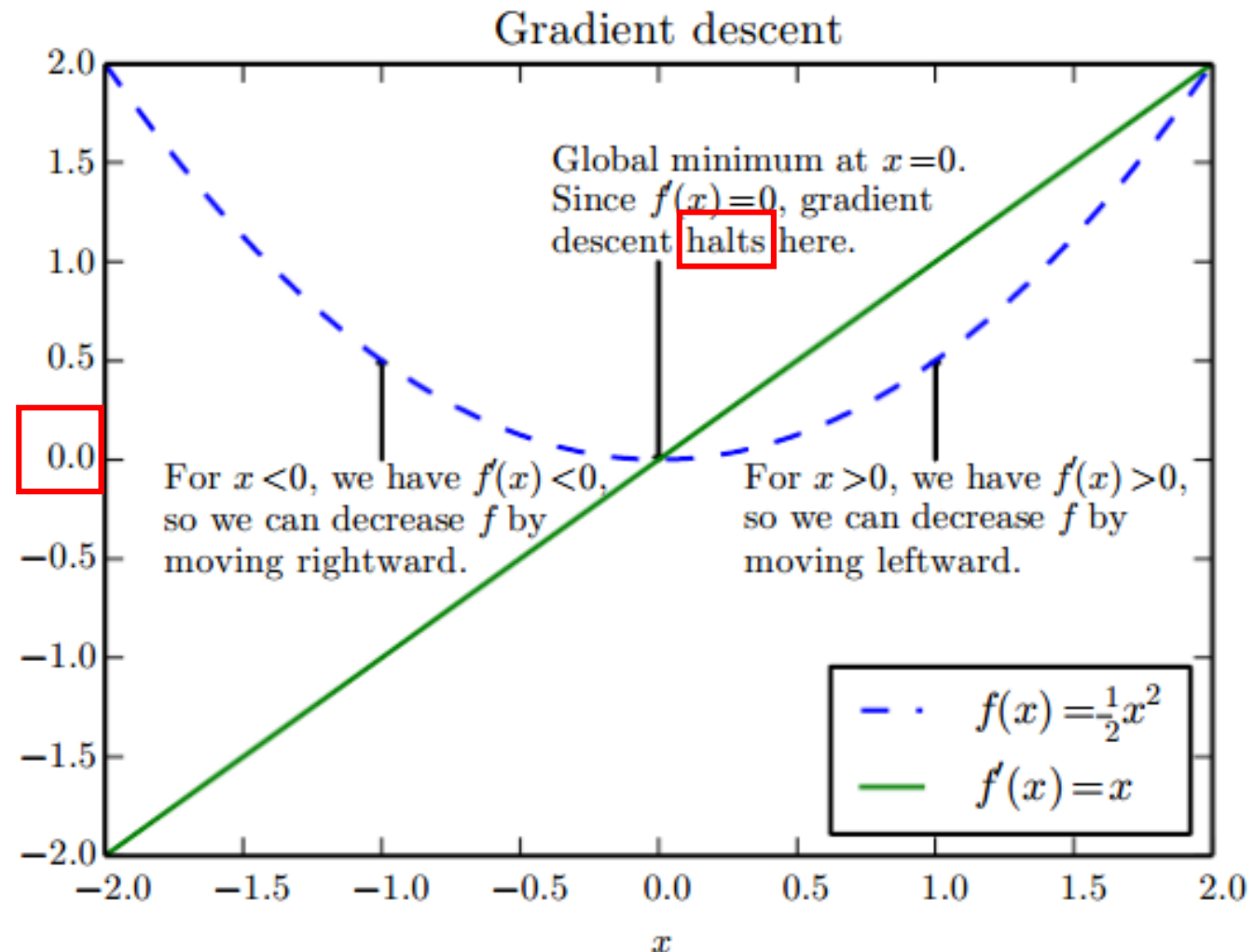
Interpretation1



Blue point is x_t , red point is x_{t+1} .

Interpretation2

- Reduce $f(x)$ by moving x in small steps with opposite sign of the derivative.
- Update rule:
$$x_{t+1} = x_t - \eta_t \nabla f(x_t)$$
- **Critical/stationary** points: Points where $f'(x) = 0$ 驻点



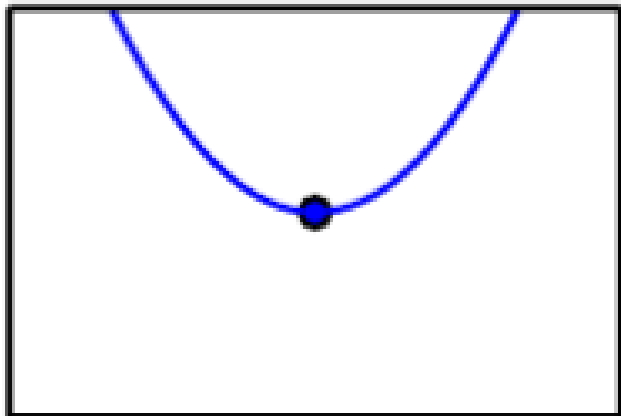
An illustration of gradient descent.

Global VS Local Minimum

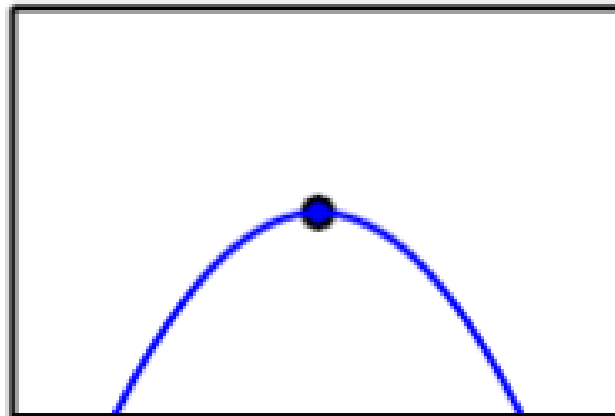
- Global minimum: a point that obtains the absolute lowest value of $f(x)$.
- Local minimum: a point where $f(x)$ is higher than at all neighboring points.
- Saddle points: some critical points are neither maxima or minima. 鞍点

Types of critical points

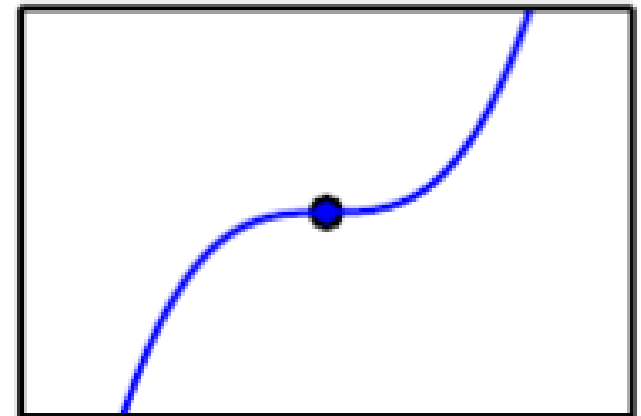
Minimum



Maximum

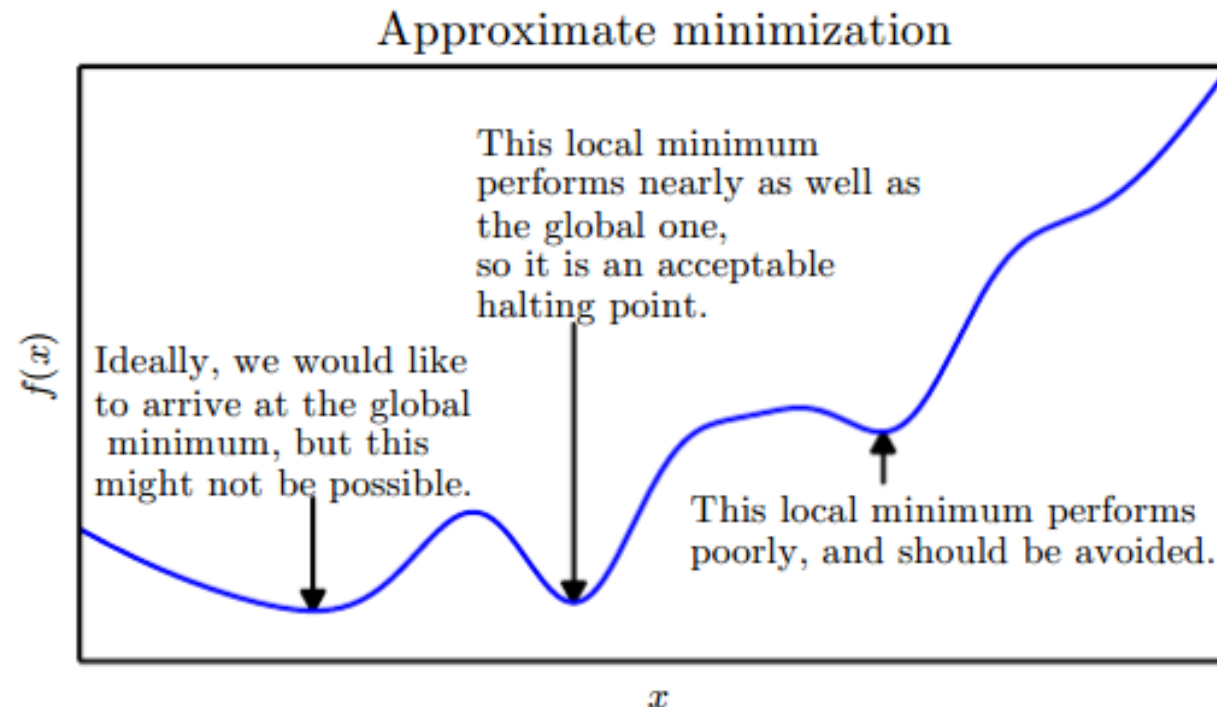


Saddle points



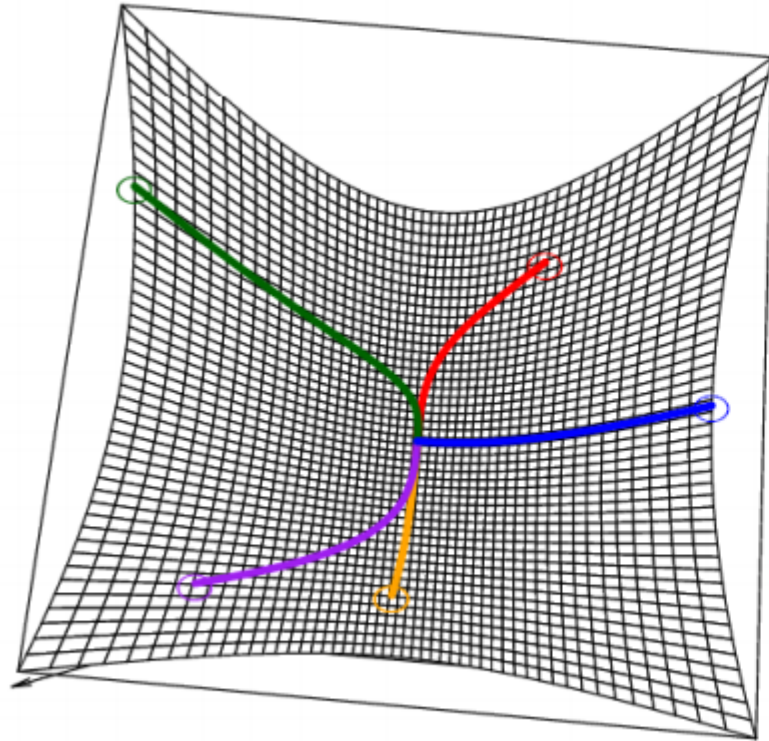
Global VS Local Minimum

- Global minimum: a point that obtains the absolute lowest value of $f(x)$.
- Local minimum: a point where $f(x)$ is higher than at all neighboring points.
- Saddle points: some critical points are neither maxima or minima. 鞍点

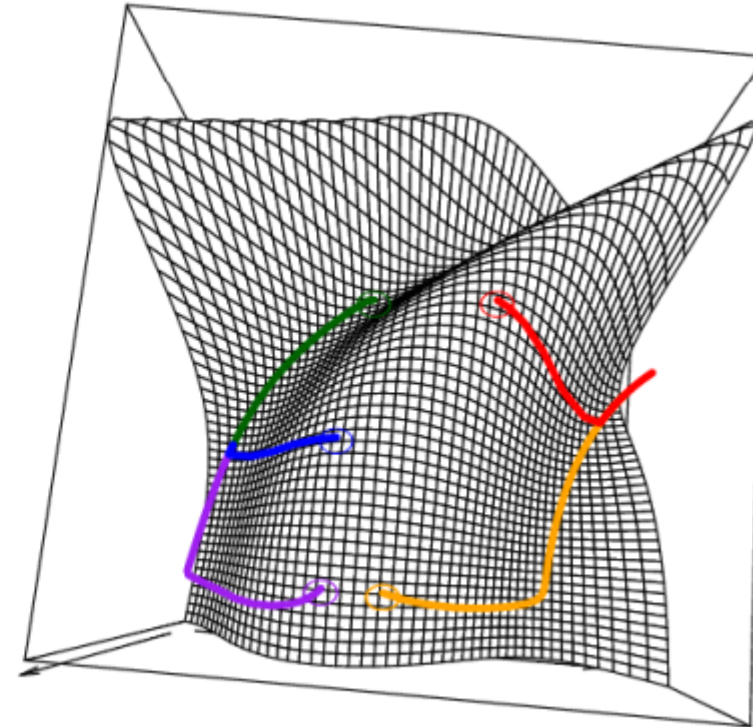


Different Starting Points

- Gradient Descent with different starting points are illustrated in different colors.



(a) Convex function



(b) Non-convex function

- (a): Strictly convex function: Converge to the global optimum.
- (b): Non-convex function: Different paths may end up at different local optima.

Gradient Descent [Cauchy 1847]

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

How to choose step sizes?

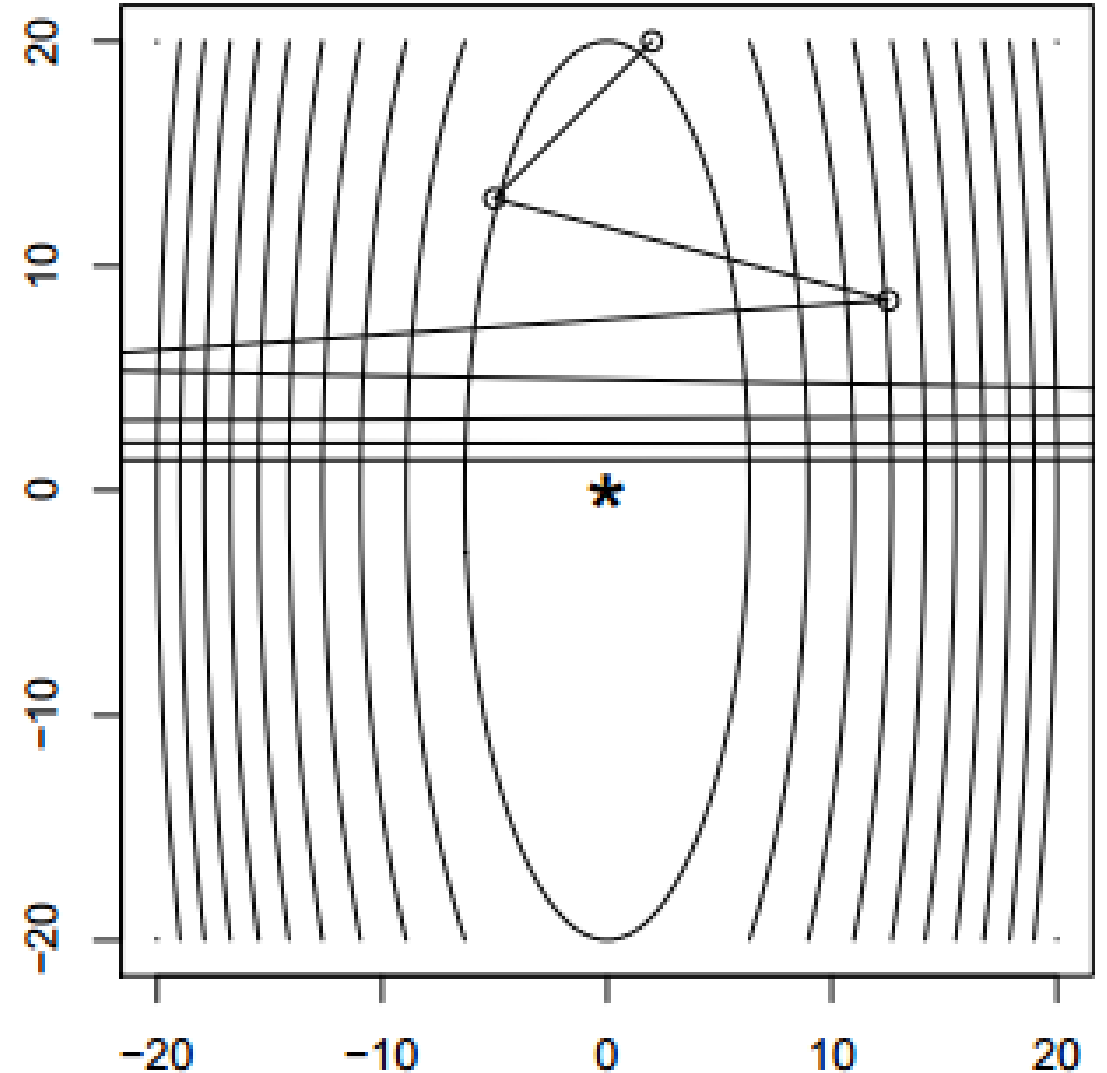
- Gradient Descent requires a step size η controlling the amount of gradient updated to the current point at each iteration.
- It is naïve to set $\eta_t = \eta$ for all iterations.

Fixed Step Sizes

Considering $f(x) = (10x_1^2 + x_2^2)/2$

If η is too big, can lead to divergence.

- The learning function oscillates away from the optimal point.
- As shown, it oscillates after 8 steps.

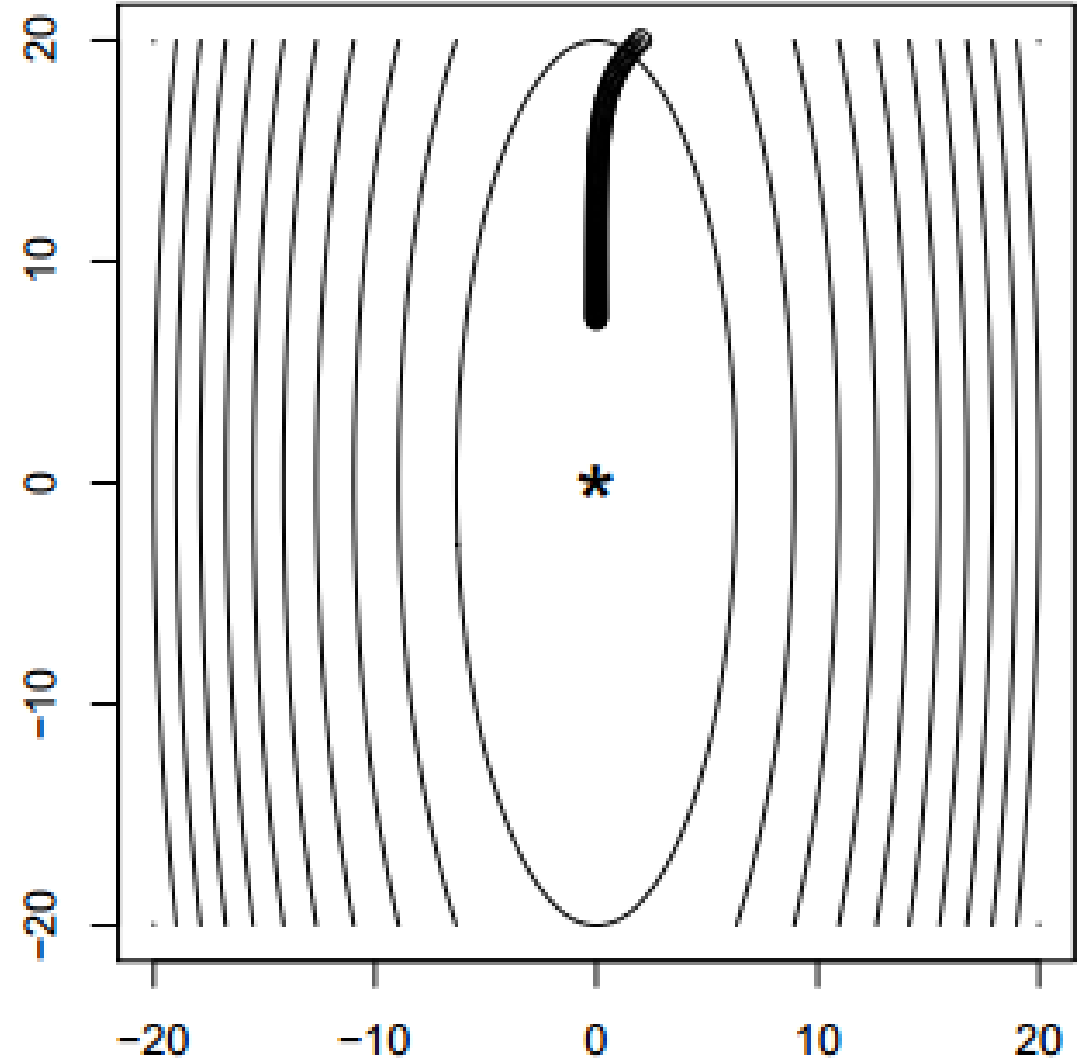


Fixed Step Sizes

Considering $f(x) = (10x_1^2 + x_2^2)/2$

If η is too small, takes longer time for the function to converge.

- As shown, GD after 100 steps.

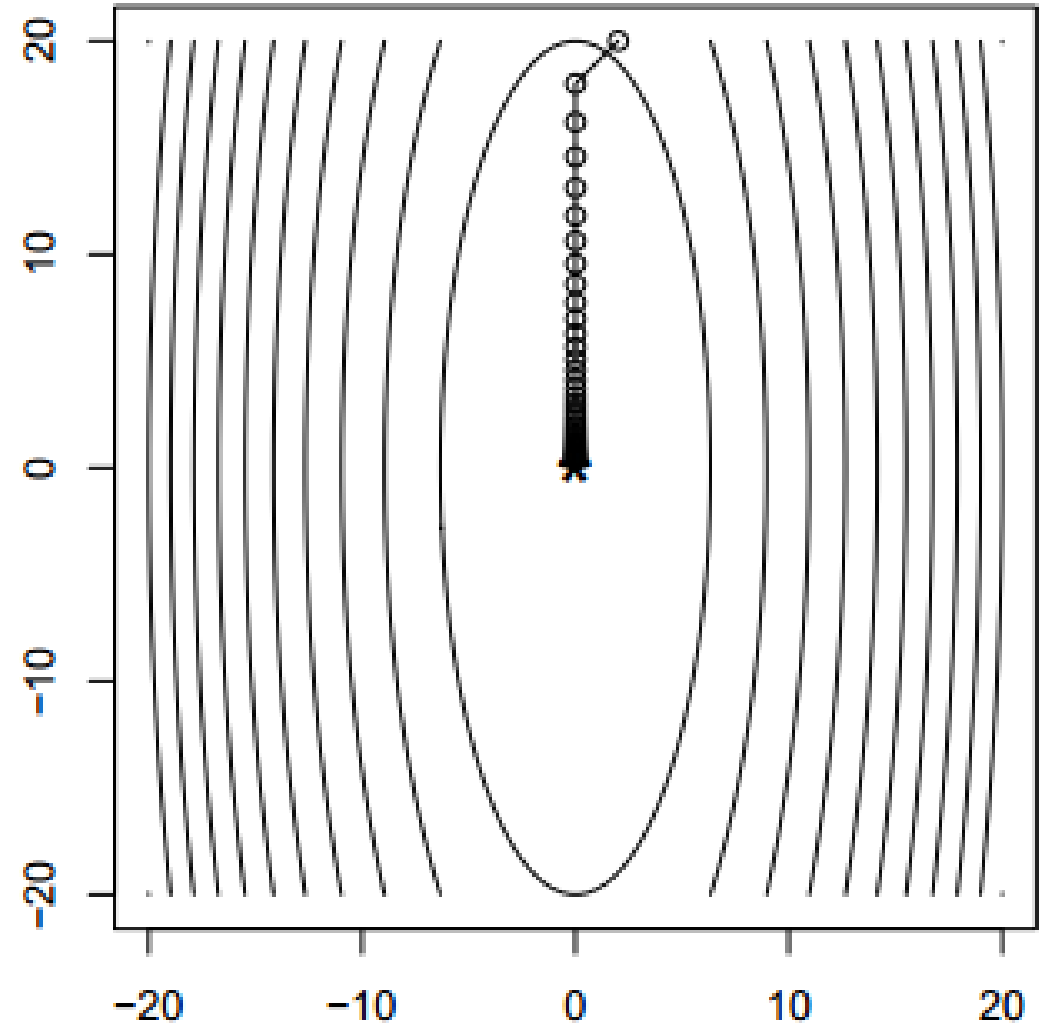


Fixed Step Sizes

Considering $f(x) = (10x_1^2 + x_2^2)/2$

Same example, gradient descent after 40 appropriately sized steps.

This porridge is too hot! - too cold! -juuussst right!



Line Search

- For algorithm 1, we need to find a search direction d_k
 - $f(x_k + \alpha d_k) < f(x_k) \quad \forall \alpha \in [0, \epsilon] \iff \nabla f(x_k)^T d_k < 0$

Algorithm 1: General gradient method with line search

```
1 Specify  $x_0$  as an initial guess for the minimum and  $k = 0$ ;  
2 repeat  
3   | Compute a search direction  $d_k$ ;  
4   | Choose a step size  $\alpha_k > 0$  to minimize  $h(\alpha) = f(x_k + \alpha d_k)$ ;  
5   | Update  $x_{k+1} = x_k + \alpha_k d_k$  and  $k = k + 1$ ;  
6 until Convergence ;
```

Exact Line Search

- Given the search direction d_k , the line search need to solve,
 - $\alpha = \arg \min_{\alpha \geq 0} h(\alpha) = \arg \min_{\alpha \geq 0} f(x_k + \alpha d_k)$
- If $h(\alpha)$ is **differentiable** and **convex**, then the optimal should satisfy,

$$h'(\alpha) = \nabla f(x_k + \alpha d_k)^T d_k = 0$$

- Since $\nabla f(x_k)^T d_k < 0$, i.e., $h'(0) < 0$.
- **If we can find $\hat{\alpha}$ such that $h'(\hat{\alpha}) > 0$** , then there must exist $\alpha^* \in [0, \hat{\alpha})$ such that $h'(\alpha^*) = 0$.
- There are many iterative algorithms to find the estimate of α^* .

Bisection Line Search

In this method, we begin with an interval $[\alpha_l, \alpha_h]$ ($h'(\alpha_l) < 0, h'(\alpha_h) > 0$) and divide the interval in two halves, i.e., $[\alpha_l, (\alpha_l + \alpha_h)/2]$ and $[(\alpha_l + \alpha_h)/2, \alpha_h]$.

A next search interval is chosen according to the sign of $h'((\alpha_l + \alpha_h)/2)$.

Algorithm 2: Bisection Line Search algorithm

```
1 Initialize  $\alpha_l = 0, \alpha_h = \hat{\alpha}$  and  $k = 0$  //  $h'(\alpha_h) > 0$ ;  
2 while  $k < MAX$  do // prevent infinite loop  
3   Set  $\alpha_m = (\alpha_l + \alpha_h)/2$ ;  
4   if  $h'(\alpha_m) \approx 0$  or  $\alpha_h - \alpha_l < \epsilon$  then // solution found!  
5     return  $\alpha_m$ ;  
6   else if  $h'(\alpha_m) > 0$  then  
7     update  $\alpha_h = \alpha_m$ ;  
8   else  
9     update  $\alpha_l = \alpha_m$ ;  
10   $k = k + 1$ ;
```

Bisection Line Search

- Bisection is accurate but may be expensive in practice.
- Usually not possible to do this minimization exactly.
- Need cheap method guaranteeing sufficient accuracy.
- **Inexact line search** method, much more efficient.

Backtracking Line Search—Armijo Rule

- A line search method to determine the maximum amount (step size) to move along a given search direction d_k .
 - Starts with a large estimate of the step size α for movement;
 - Iteratively shrinking the step size (i.e., "backtracking") until a sufficient decrease of the $f(x_k) - f(x_k + \alpha d_k)$.

$$f(x_k + \alpha d_k) \leq f(x_k) + c_1 \alpha f'(x_k)^T d_k$$

$f: \mathbb{R}^n \rightarrow \mathbb{R}$, $c_1 \in (0,1)$, α : step size, d_k is the search direction which satisfies $\nabla f(x_k)^T d_k < 0$.

Backtracking Line Search—Armijo Rule

$$h'(\alpha) = \nabla f(x_k + \alpha d_k)^T d_k$$

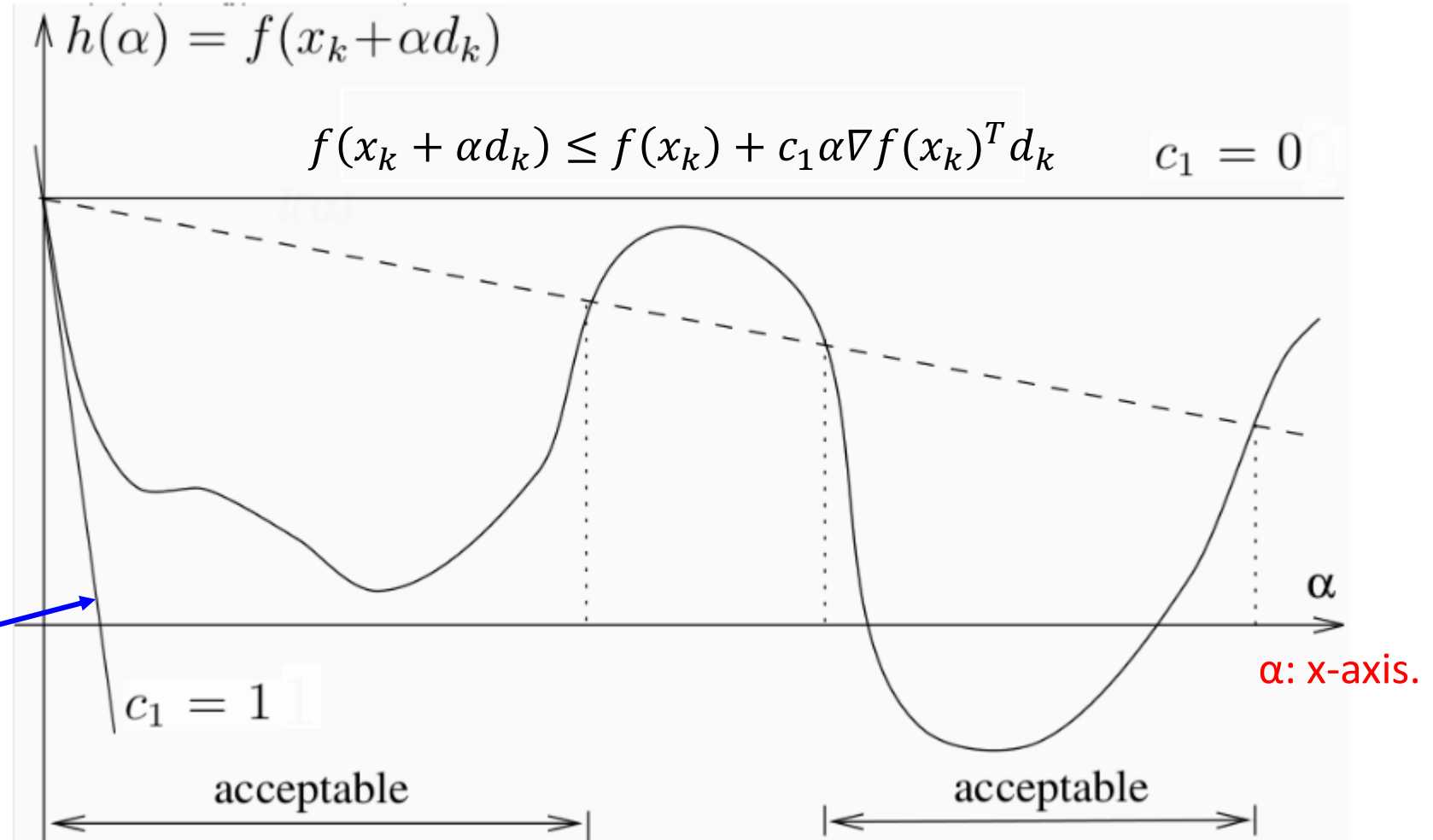
Slope at $\alpha = 0$

$$h'(0) = \nabla f(x_k)^T d_k$$

$$h(0) = f(x_k)$$

The function of the tangent,

$$l(\alpha) = f(x_k) + \alpha \nabla f(x_k)^T d_k$$



Armijo condition for backtracking line search

Backtracking Line Search—Armijo Rule

- As long as α is sufficient small, α must satisfy the Armijo rule.
- That is why we need to start with a large step and then shrink it.

Algorithm 3: Backtracking Line Search with Armijo condition

Input: $c_1 \in (0, 1), \tau \in (0, 1)$, search direction d and step size α_0

1 Initialize $k = 0$;

2 **while** $f(x + \alpha_k d) > f(x) + c_1 \alpha_k f'(x)^T d$ **do**

3 update $\alpha_{k+1} = \tau \alpha_k$;

4 $k = k + 1$;

5 **return** α_k ;

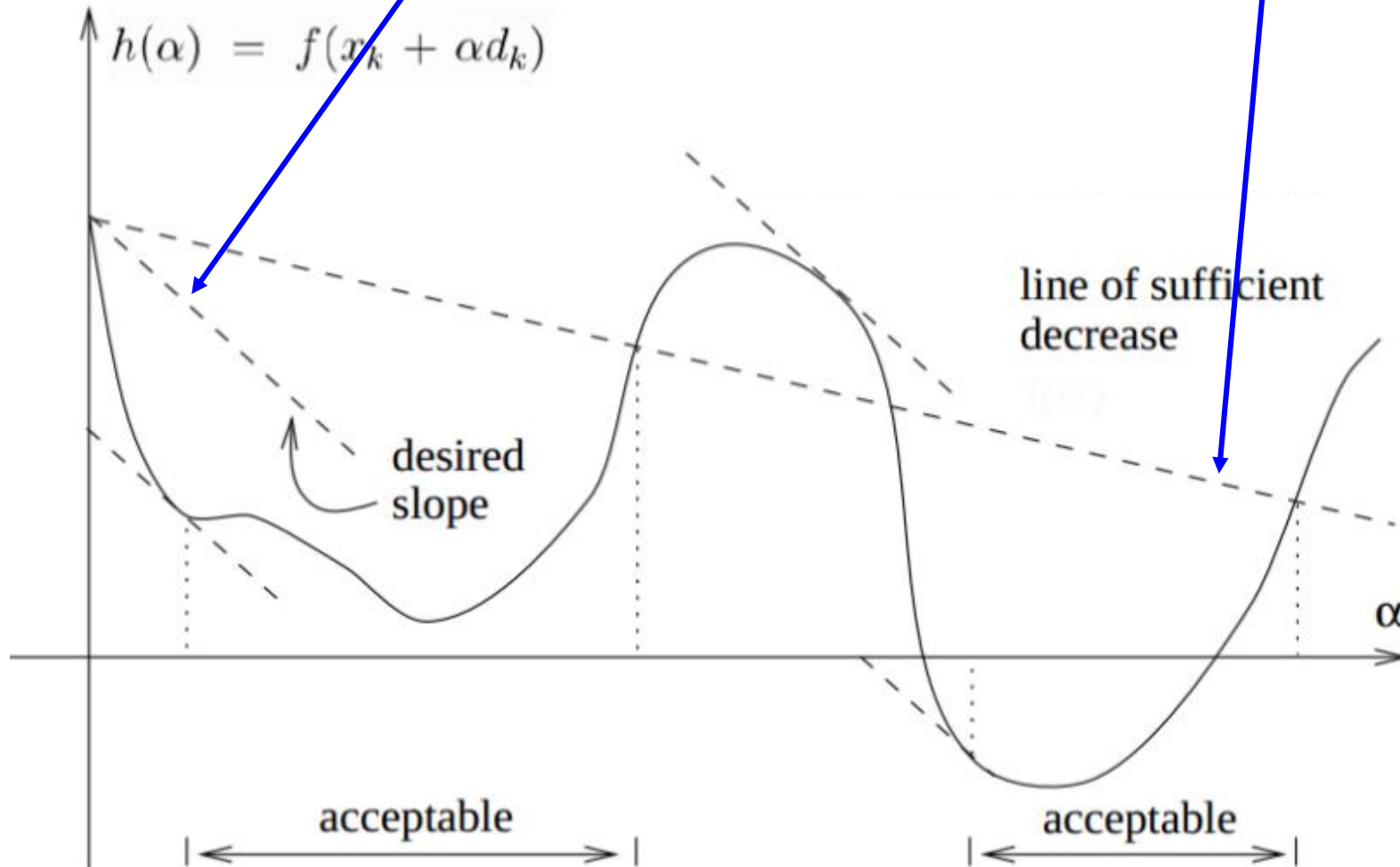
Backtracking Line Search—Wolfe Rule

Armijo Rule:

$$f(x_k + \alpha d_k) \leq f(x_k) + c_1 \alpha \nabla f(x_k)^T d_k$$

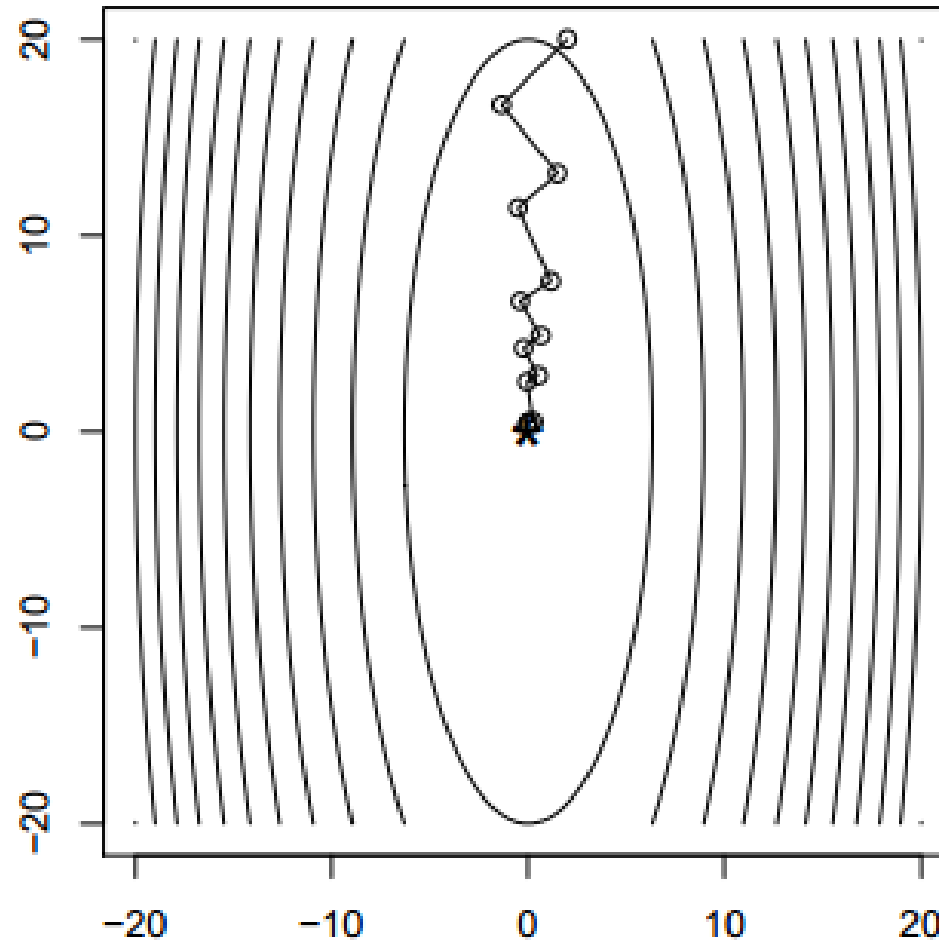
Curvature Condition:

$$f'(x_k + \alpha d_k)^T d_k \geq c_2 f'(x_k)^T d_k$$



Backtracking Line Search

- Backtracking picks up roughly the right step size (13 steps):
- Here $\beta = 0.8$. (Recommend $\beta \in (0.1, 0.8)$)



Deterministic Optimization

- First-order methods: methods that use only the gradient.
- Second-order methods: methods that also use the Hessian matrix.

$$\mathbf{H}(f)_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\mathbf{x})$$

\mathbf{x} : multiple input dimensions.

Newton's Methods

- Motivation: to minimize the local **second-order Taylor** approximation of f .

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \approx \min_{\boldsymbol{x}} f(\boldsymbol{x}_t) + \nabla f(\boldsymbol{x}_t)^T (\boldsymbol{x} - \boldsymbol{x}_t) + \frac{1}{2} (\boldsymbol{x} - \boldsymbol{x}_t)^T \nabla^2 f(\boldsymbol{x}_t) (\boldsymbol{x} - \boldsymbol{x}_t)$$

Newton's Methods

- Motivation: to minimize the local **second-order Taylor** approximation of f .

$$\min_{\mathbf{x}} f(\mathbf{x}) \approx \min_{\mathbf{x}} f(\mathbf{x}_t) + \nabla f(\mathbf{x}_t)^T (\mathbf{x} - \mathbf{x}_t) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_t)^T \nabla^2 f(\mathbf{x}_t) (\mathbf{x} - \mathbf{x}_t)$$

- Take the derivative of \mathbf{x} on both side, we have,

$$\frac{df(\mathbf{x})}{d\mathbf{x}} = \nabla f(\mathbf{x}_t) + \nabla^2 f(\mathbf{x}_t) (\mathbf{x} - \mathbf{x}_t) = \mathbf{0}$$

Newton's Methods

- Motivation: to minimize the local **second-order Taylor** approximation of f .

$$\min_{\mathbf{x}} f(\mathbf{x}) \approx \min_{\mathbf{x}} f(\mathbf{x}_t) + \nabla f(\mathbf{x}_t)^T (\mathbf{x} - \mathbf{x}_t) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_t)^T \nabla^2 f(\mathbf{x}_t) (\mathbf{x} - \mathbf{x}_t)$$

- Take the derivative of \mathbf{x} on both side, we have,

$$\frac{df(\mathbf{x})}{d\mathbf{x}} = \nabla f(\mathbf{x}_t) + \nabla^2 f(\mathbf{x}_t) (\mathbf{x} - \mathbf{x}_t) = \mathbf{0}$$

- Update rule: suppose $\nabla^2 f(\mathbf{x}_t)$ is positive definite,

$$\mathbf{x} = \mathbf{x}_t - [\nabla^2 f(\mathbf{x}_t)]^{-1} \nabla f(\mathbf{x}_t)$$

Newton's Methods

- Motivation: to minimize the local **second-order Taylor** approximation of f .

$$\min_x f(x) \approx \min_x f(x_t) + f'(x_t)(x - x_t) + \frac{1}{2}f''(x_t)(x - x_t)^2$$

- Take the derivative of x on both side, we have,

$$f'(x) = f'(x_t) + f''(x_t)(x - x_t) = 0$$

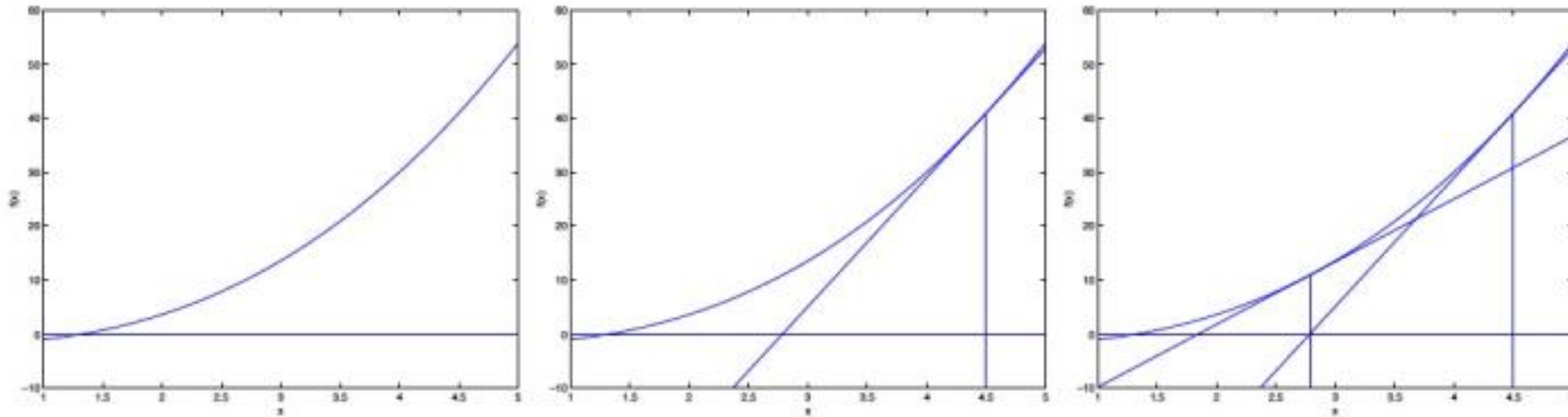
- Update rule: suppose $f''(x_t) \neq 0$,

$$x = x_t - \frac{f'(x_t)}{f''(x_t)}$$

Newton's Methods

- In numerical analysis, Newton's Methods is to find successively better approximations to the roots of a real-valued function, (i.e, $z: f(z) = 0$).

$$z = z_t - \frac{f(z_t)}{f'(z_t)}$$



- In optimization, we want to find the stationary point $f'(x_t) = 0$, i.e.,

$$x = x_t - \frac{f'(x_t)}{f''(x_t)}$$

Newton's Methods

- **Advantage:**

- More **accurate** local approximation of the objective,
- The convergence is much **faster**.

- **Disadvantage:**

- Need to compute the **second derivatives**
- Need to compute the **inverse** of Hessian (time/storage consuming)

Quasi Newton's Methods

- **Main Idea:** To approximate the inverse with a matrix B_t that is iteratively refined by low rank updates to become a better approximation of $[\nabla^2 f(x_t)]^{-1}$.

Quasi Newton's Methods

- BFGS (Broyden–Fletcher–Goldfarb–Shanno):

Broyden, Fletcher, Goldfarb, Shanno



Optimization Methods

- **Deterministic Optimization**
 - The data for the given problem are known accurately.
- **Stochastic Optimization**
 - Refers to a collection of methods for minimizing or maximizing an objective function when randomness is present.

Optimization Methods

- **Deterministic Optimization**
 - The data for the given problem are known accurately.
- **Stochastic Optimization**
 - Refers to a collection of methods for minimizing or maximizing an objective function when randomness is present.

Stochastic Gradient Descent

We now minimize the *empirical risk*,

m is the number of training examples.

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x}, y) \sim \hat{P}_{data}} L(f(\mathbf{x}, \boldsymbol{\theta}), y) = \frac{1}{m} \sum_{i=1}^m L(f(\mathbf{x}^{(i)}, \boldsymbol{\theta}), y^{(i)})$$

- Optimization algorithms that use the entire training set simultaneously are called *deterministic* or *batch* gradient methods.
- This terminology “*batch*” can be somewhat *confusing*.
 - We use the term “*batch size*” to describe the size of a minibatch in the stochastic gradient descent.
 - We use the term “*batch gradient descent*” to imply the use of the full training set.

Stochastic Gradient Descent

- Optimization algorithms that use only a **single** example at a time are sometimes called *stochastic* or sometimes *online* methods.

Stochastic Gradient Descent

- Optimization algorithms that use only a **single** example at a time are sometimes called **stochastic** or sometimes **online** methods.

- E.g., consider the cost function of linear regression as

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

- Then we have ,
$$\begin{aligned} \frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_{\theta}(x) - y)^2 \\ &= 2 \cdot \frac{1}{2} (h_{\theta}(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_{\theta}(x) - y) \\ &= (h_{\theta}(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^n \theta_i x_i - y \right) \\ &= (h_{\theta}(x) - y) x_j \end{aligned}$$

Batch vs Stochastic Gradient Descent

- **Batch** gradient descent has to scan through the entire training set before taking a single step—a costly operation if m is large.

Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^m (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)} \quad (\text{for every } j).$$

}

Batch GD

- **SGD** can start making progress right away, and continues to make progress with each example it looks at.
- Often, SGD gets θ “close” to the minimum much **faster** than **batch** gradient descent.

Loop {

for i=1 to m, {

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)} \quad (\text{for every } j).$$

}

}

SGD

Batch vs Stochastic Gradient Descent

- SGD may **never “converge”** to the minimum, and the parameters θ will keep oscillating around the minimum of $J(\theta)$;
- **But in practice** most of the values near the minimum will be reasonably **good approximations** to the true minimum.
- Therefore, when the training set is large, SGD is often preferred over batch gradient descent.

Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^m (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)} \quad (\text{for every } j).$$

}

Batch GD

Loop {

for i=1 to m, {

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)} \quad (\text{for every } j).$$

}

}

SGD

Stochastic Gradient Descent

- Most algorithms (called *minibatch* or *minibatch stochastic* methods) fall somewhere in between, using more than one but less than all of the training examples.
- **Note:** Confusing again..... It is now common to simply call them *stochastic* methods.

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k .

Require: Initial parameter θ

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

 Apply update: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

end while

Accelerated SGD for Deep Learning

- **Polyak's Classical Momentum** [Polyak 1964]
- **Nesterov's Momentum** [Nesterov 1983]

Polyak's Classical Momentum [Polyak 1964]

- The classical momentum (CM) accumulates an **exponentially decaying moving average of past gradients** and continues to move in their direction.
- Letting η be the learning rate.

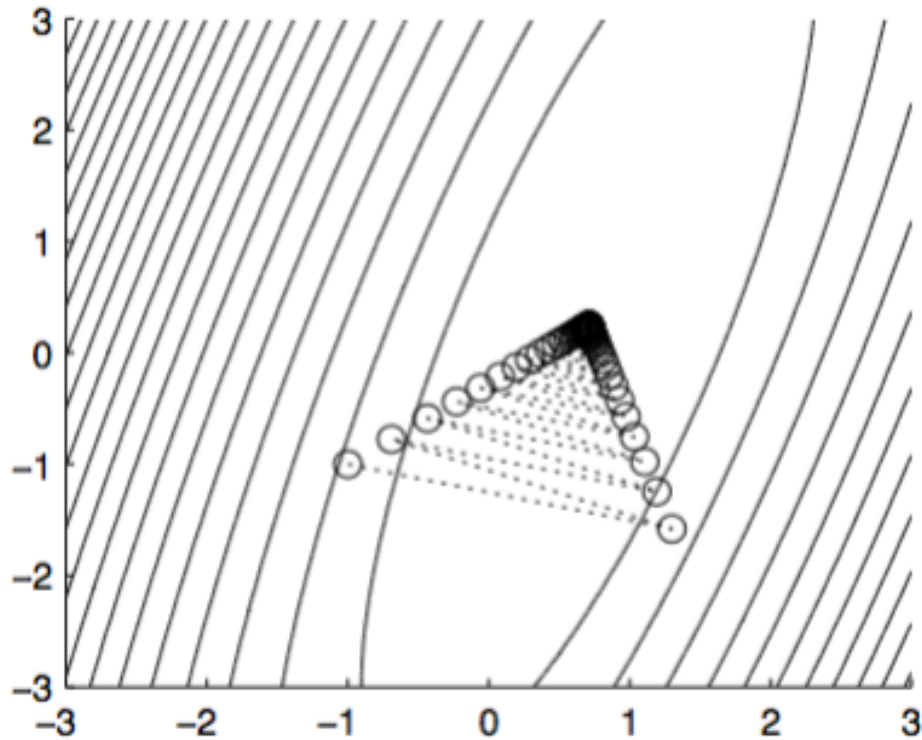
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_{t+1}$$

$$\mathbf{v}_{t+1} = \mu \mathbf{v}_t - \eta \nabla_{\mathbf{w}_t} f$$

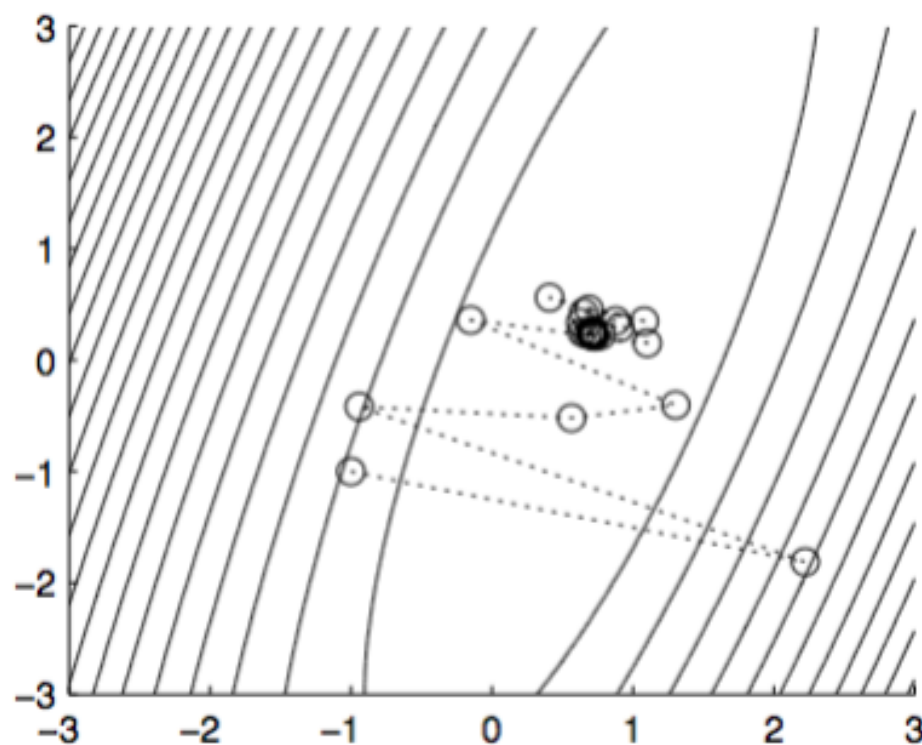
- Velocity vector \mathbf{v}_t : a **memory** that accumulates the directions of reduction that were chosen in the previous t steps.
- The influence of \mathbf{v} is controlled by the **momentum coefficient** $\mu \in [0,1]$. μ is usually slightly less than 1. When $\mu = 0$: it is just the Gradient Descent.

Polyak's Classical Momentum [Polyak 1964]

- **Motivation:** Key problem of Gradient Descent is “zig-zagging”.



(a) Zig-zagging problem of GD.



(a) Trajectory of CM on same problem.

Polyak's Classical Momentum [Polyak 1964]

- The SGD algorithm with momentum is given as follows.

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_{t+1} \quad \mathbf{v}_{t+1} = \mu \mathbf{v}_t - \eta \nabla_{\mathbf{w}_t} f$$

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate η , momentum parameter μ .

Require: Initial parameter $\boldsymbol{\theta}$, initial velocity \mathbf{v} .

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient estimate: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)})$

 Compute velocity update: $\mathbf{v} \leftarrow \mu \mathbf{v} - \eta \mathbf{g}$

 Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}$

end while

Polyak's Classical Momentum [Polyak 1964]

- If the error surface is a **titled** plane, the ball reaches a terminal velocity.
 - If the momentum is close to 1, this is much **faster** than simple gradient descent.

$$\mathbf{v}_{\infty} \rightarrow \frac{1}{1-\mu} (-\eta \nabla_{\mathbf{w}} f)$$

$$\mu < 1, \nabla_{\mathbf{w}_t} f \approx \nabla_{\mathbf{w}_{t-1}} f$$

Polyak's Classical Momentum [Polyak 1964]

- If the error surface is a **titled** plane, the ball reaches a terminal velocity.
 - If the momentum is close to 1, this is much **faster** than simple gradient descent.

$$\mathbf{v}_\infty \rightarrow \frac{1}{1-\mu} (-\eta \nabla_{\mathbf{w}} f) \quad \mu < 1, \nabla_{\mathbf{w}_t} f \approx \nabla_{\mathbf{w}_{t-1}} f$$

$$\mathbf{v}_{t+1} = \mu \mathbf{v}_t - \eta \nabla_{\mathbf{w}_t} f = \mu(\mu \mathbf{v}_{t-1} - \eta \nabla_{\mathbf{w}_t} f) - \eta \nabla_{\mathbf{w}_t} f = \mu^{t+1} \mathbf{v}_0 - (\mu^t + \dots + \mu^1 + 1) \eta \nabla_{\mathbf{w}_t} f$$

Let $S = (\mu^t + \dots + \mu^1 + 1)$, we have,

$$\mu S + 1 = \mu(\mu^t + \dots + \mu^1 + 1) + 1 = \mu^{t+1} + S \Rightarrow S = \frac{1 - \mu^{t+1}}{1 - \mu}$$

$$\mathbf{v}_{t+1} = \mu^{t+1} \mathbf{v}_0 - \left(\frac{1 - \mu^{t+1}}{1 - \mu} \right) \eta \nabla_{\mathbf{w}_t} f \Rightarrow \mathbf{v}_\infty \rightarrow \frac{1}{1 - \mu} (-\eta \nabla_{\mathbf{w}} f)$$

- If $\mu = 0.99$, then we can speed up 100 times.

Nesterov's Accelerated Gradient [Nesterov 1983]

- The update equations of NAG are:

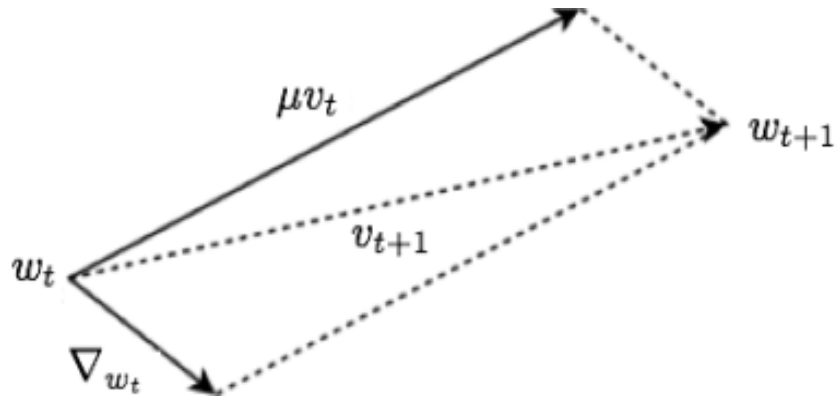
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_{t+1}$$

$$\mathbf{v}_{t+1} = \mu \mathbf{v}_t - \eta \nabla_{\mathbf{w}_t + \mu \mathbf{v}_t} f$$

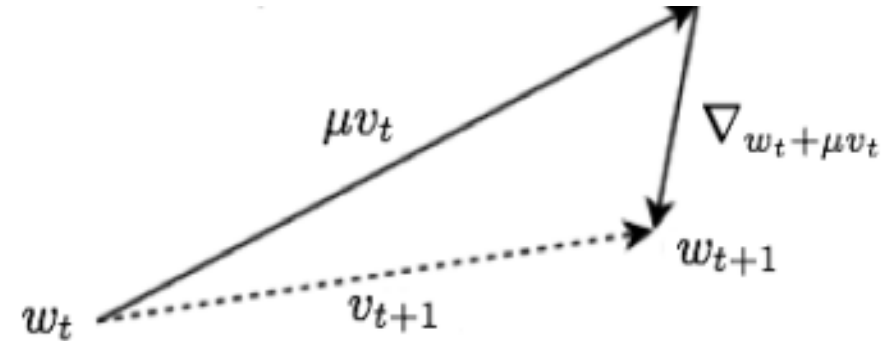
CM

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_{t+1}$$

$$\mathbf{v}_{t+1} = \mu \mathbf{v}_t - \eta \nabla_{\mathbf{w}_t} f$$



CM

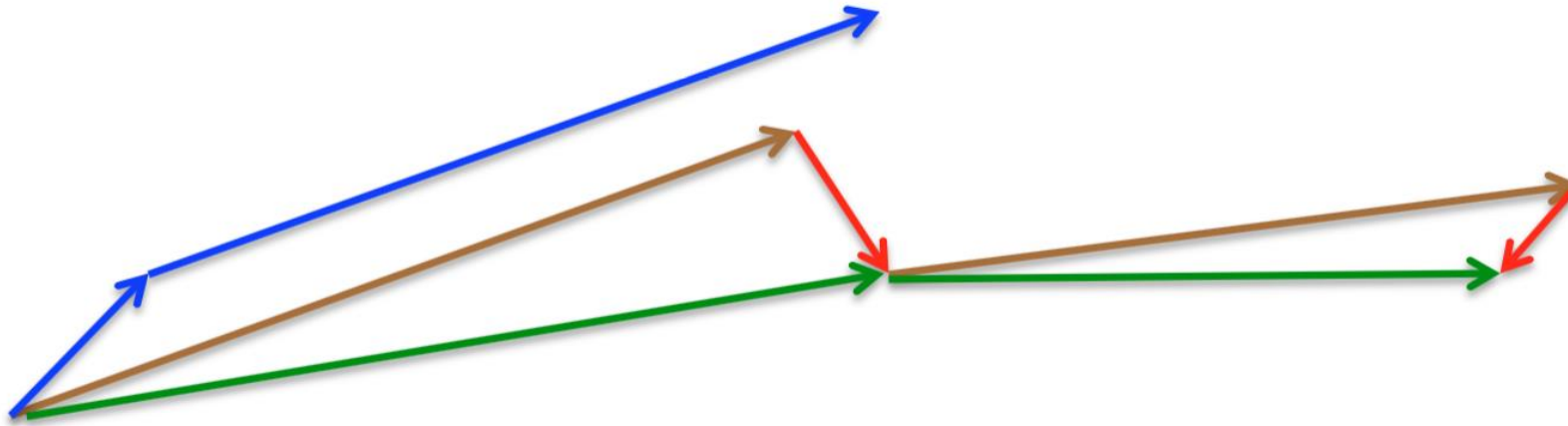


NAG

Illustration of the comparison between CM and NAG.

Nesterov's Accelerated Gradient [Nesterov 1983]

- **First** make a big jump in the direction of the previous accumulated gradient.
- **Then** measure the gradient where you end up and make a **correction**.



$$\lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n = e$$

$$\lim_{\epsilon \rightarrow 0} (1 - \epsilon)^{\frac{1}{\epsilon}} = \frac{1}{e}$$

brown vector = jump, red vector = correction, green vector = accumulated gradient

blue vectors = standard momentum

Nesterov's Accelerated Gradient [Nesterov 1983]

- The update equations of NAG are:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_{t+1}$$

$$\mathbf{v}_{t+1} = \mu \mathbf{v}_t - \eta \nabla_{\boxed{\mathbf{w}_t + \mu \mathbf{v}_t}} f$$

CM

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_{t+1}$$

$$\mathbf{v}_{t+1} = \mu \mathbf{v}_t - \eta \nabla_{\boxed{\mathbf{w}_t}} f$$

- CM → inspecting the gradient at the **current iterate** of \mathbf{w}_t ;
 - Faithfully trusts the current iterate;
- NAG → inspecting **the gradient at $\mathbf{w}_t + \mu \mathbf{v}_t$** .
 - Puts less faith into the current iterate and **looks ahead** in the direction suggested by the velocity vector.
- The small difference allows NAG to adapt faster and in a more stable way.

Nesterov's Accelerated Gradient [Nesterov 1983]

- The complete Nesterov momentum algorithm is presented as follows.

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_{t+1} \quad \mathbf{v}_{t+1} = \mu \mathbf{v}_t - \eta \nabla_{\mathbf{w}_t + \mu \mathbf{v}_t} f$$

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

Require: Learning rate η , momentum parameter μ .

Require: Initial parameter $\boldsymbol{\theta}$, initial velocity \mathbf{v} .

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding labels $\mathbf{y}^{(i)}$.

 Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \mu \mathbf{v}$

 Compute gradient (at interim point): $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\mathbf{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \mathbf{y}^{(i)})$

 Compute velocity update: $\mathbf{v} \leftarrow \mu \mathbf{v} - \eta \mathbf{g}$

 Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}$

end while

Adagrad

Motivation: Many features are irrelevant, while rare features are often very informative.

Adagrad (Duchi et al, COLT 2010) uses a different learning rate for **every parameter** θ_i at **every time step** t .

$$g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i})$$

The SGD update

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

$G_t \in \mathbb{R}^{d \times d}$ is a 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 .

Weakness: Since every added term is positive, the accumulated sum **keeps growing** which in turn causes the learning rate to **shrink** and eventually become infinitesimally small.

Adadelta & RMSprop

Adadelta is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta **restricts the window of** accumulated past gradients to some **fixed size w** .

Via a decaying mechanism, $E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma)g_t^2$

RMSprop is an unpublished, adaptive learning rate method proposed by Geoff Hinton in his Coursera Class. RMSprop and Adadelta have both been developed **independently** around the same time to resolve Adagrad's radically diminishing learning rates.

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}}g_t$$

Adam

- **Adaptive Moment Estimation (Adam)** is another method that computes adaptive learning rates for each parameter.
- Adam was presented by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto in their **2015** ICLR paper (poster) titled “Adam: A Method for Stochastic Optimization”.
 1. Adam stores an **exponentially decaying** average of past **squared gradients v_t (variance)** like **Adadelta** and **RMSprop**.
 2. Adam also keeps an **exponentially decaying** average of **past gradients m_t (mean)**, similar to **momentum**.

Adam

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t .

Require: α : Stepsize

Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates

Require: $f(\theta)$: Stochastic objective function with parameters θ

Require: θ_0 : Initial parameter vector

$m_0 \leftarrow 0$ (Initialize 1st moment vector)

$v_0 \leftarrow 0$ (Initialize 2nd moment vector)

$t \leftarrow 0$ (Initialize timestep)

while θ_t not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)

✓ $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)

end while

return θ_t (Resulting parameters)

As m_t, v_t are initialized as 0, they are biased towards 0. →

Summary

- **Deterministic Optimization**
 - The data for the given problem are known accurately.
 - **First** order method: e.g., Gradient Descent.
 - Exact Line Search
 - Inexact Line Search
 - Backtracking Line Search—Armijo Rule/Wolfe Rule
 - **Second** order method: e.g., **Newton's** method, Quasi Newton's Methods (BFGS).
- **Stochastic Optimization**
 - Using several samples of the training examples (minibatch).
 - SGD
 - SGD+Acceleration
 - Polyak's Classical Momentum
 - **Nesterov's Momentum**
 - Agrad/Adadelata/RMSprop/**Adam**

Back Up

The following slides are optional just in case that you are interested. 😊

Quasi Newton's Methods

- BFGS (Broyden–Fletcher–Goldfarb–Shanno):

Broyden, Fletcher, Goldfarb, Shanno



BFGS (Broyden–Fletcher–Goldfarb–Shanno):

Main Idea: To approximate the inverse of Hessian with a matrix B_t that is iteratively refined by low rank updates to become a better approximation of $[\nabla^2 f(x_t)]^{-1}$.

- the former inverse hessian estimate H_n^{-1}
- the input differences (s_n)
- the gradient differences (y_n).

QuasiNewton($f, H_0^{-1}, x_0, QuasiUpdate$)

For $n=0,1,\dots$ (until converged):

//Compute search direction and step-size

$$d = H_0^{-1} g_n$$

$$\alpha = \min_{\alpha \geq 0} f(x_n - \alpha d)$$

$$x_{n+1} \leftarrow x_n - \alpha d$$

//store the input and gradient deltas

$$g_{n+1} = \nabla f(x_{n+1})$$

$$s_{n+1} \leftarrow x_{n+1} - x_n$$

$$y_{n+1} \leftarrow g_{n+1} - g_n$$

//update inverse hessian

$$H_{n+1}^{-1} \leftarrow QuasiUpdate(H_n^{-1}, s_{n+1}, y_{n+1})$$

BFGS (Broyden–Fletcher–Goldfarb–Shanno):

$$\mathbf{H}_{n+1}^{-1} \leftarrow \text{QuasiUpdate}(\mathbf{H}_n^{-1}, s_{n+1}, y_{n+1})$$

- What form should *QuasiUpdate* take?
- What if we have *QuasiUpdate* always return the identity matrix?
 - Gradient descent (The search direction is always ∇f_n).
 - This will converge to x^* for convex f .
 - This choice isn't attempting to capture second-order information about f .

Let's first think about the approximation for f near x_n :

$$p_n(\Delta x) = f(x_n) + \Delta x^T \mathbf{g}_n + \frac{1}{2} \Delta x^T \mathbf{H}_n \Delta x$$

BFGS (Broyden–Fletcher–Goldfarb–Shanno):

Let's first think about the approximation for f near x_n :

$$p_n(\Delta x) = f(x_n) + \Delta x^T \mathbf{g}_n + \frac{1}{2} \Delta x^T \mathbf{H}_n \Delta x$$

□ Secant Condition

- A good property for p_n is that its gradient agrees with f at x_n and x_{n-1} , i.e.,

$$\nabla p_n(x_n) = \mathbf{g}_n, \quad \nabla p_n(x_{n-1}) = \mathbf{g}_{n-1}$$

- Using both of the equations above:

$$\nabla p_n(x_n) - \nabla p_n(x_{n-1}) = \mathbf{g}_n - \mathbf{g}_{n-1}$$

- Using the gradient of $p_n(\cdot)$ and cancelling terms we get

$$(\mathbf{g}_n + \mathbf{H}_n x_n) - (\mathbf{g}_n + \mathbf{H}_n x_{n-1}) = (\mathbf{g}_n - \mathbf{g}_{n-1}) \quad \Rightarrow \quad \mathbf{H}_n (x_n - x_{n-1}) = (\mathbf{g}_n - \mathbf{g}_{n-1})$$

BFGS (Broyden–Fletcher–Goldfarb–Shanno):

Let's first think about the approximation for f near x_n :

$$p_n(\Delta x) = f(x_n) + \Delta x^T \mathbf{g}_n + \frac{1}{2} \Delta x^T \mathbf{H}_n \Delta x$$

□ Secant Condition

$$\mathbf{H}_n \begin{matrix} \mathbf{y}_n \\ [x_n - x_{n-1}] \end{matrix} = \begin{matrix} \mathbf{s}_n \\ \mathbf{g}_n - \mathbf{g}_{n-1} \end{matrix}$$

- This ensure: \mathbf{H}_{n+1} behaves like the Hessian at least for the difference $(x_n - x_{n-1})$.
- Assuming \mathbf{H}_n is invertible (i.e., psd), we have,

$$\mathbf{H}_n^{-1} \mathbf{y}_n = \mathbf{s}_n$$

BFGS (Broyden–Fletcher–Goldfarb–Shanno):

Let's first think about the approximation for f near x_n :

$$p_n(\Delta x) = f(x_n) + \Delta x^T \mathbf{g}_n + \frac{1}{2} \Delta x^T \mathbf{H}_n \Delta x$$

□ Symmetric Condition

- A Hessian represents the matrix of 2nd order partial derivatives;
- The Hessian is symmetric since the order of differentiation doesn't matter.

BFGS (Broyden–Fletcher–Goldfarb–Shanno):

- Given the two conditions, we take the most conservative change relative to \mathbf{H}_{n-1} ,

$$\min_{\mathbf{H}^{-1}} \|\mathbf{H}^{-1} - \mathbf{H}_{n-1}^{-1}\|^2$$

$$\text{s.t. } \mathbf{H}^{-1} \mathbf{y}_n = \mathbf{s}_n$$

$$\mathbf{H}^{-1} \text{ is symmetric}$$

$\|\cdot\|$ is the Frobenius norm.

- The solution to this optimization problem is given by ,

$$\mathbf{H}_{n+1}^{-1} = (\mathbf{I} - \rho_n \mathbf{y}_n \mathbf{s}_n^T) \mathbf{H}_n^{-1} (\mathbf{I} - \rho_n \mathbf{s}_n \mathbf{y}_n^T) + \rho_n \mathbf{s}_n \mathbf{s}_n^T$$

$$\rho_n = (\mathbf{y}_n^T \mathbf{s}_n)^{-1}$$

- \mathbf{H}_{n+1}^{-1} is positive definite when \mathbf{H}_n^{-1} is.
 - We can choose any \mathbf{H}_0^{-1} we want, including the \mathbf{I} matrix, this is easy to ensure.