Scalable ML

10605-10805

Gradient Descent

Barnabás Póczos

Books & Papers to Read

- **Nesterov**: Introductory lectures on convex optimization
- Many slides are taken from **Ryan Tibshirani**
- Pictures and notes are from **Sebastian Ruder:**
  An overview of gradient descent optimization algorithms

Gradient Descent

Consider unconstrained minimization of $f : \mathbb{R}^n \to \mathbb{R}$, convex and differentiable. We want to solve

$$\min_{x \in \mathbb{R}^n} f(x),$$

i.e., find $x^\star$ such that $f(x^\star) = \min_x f(x)$

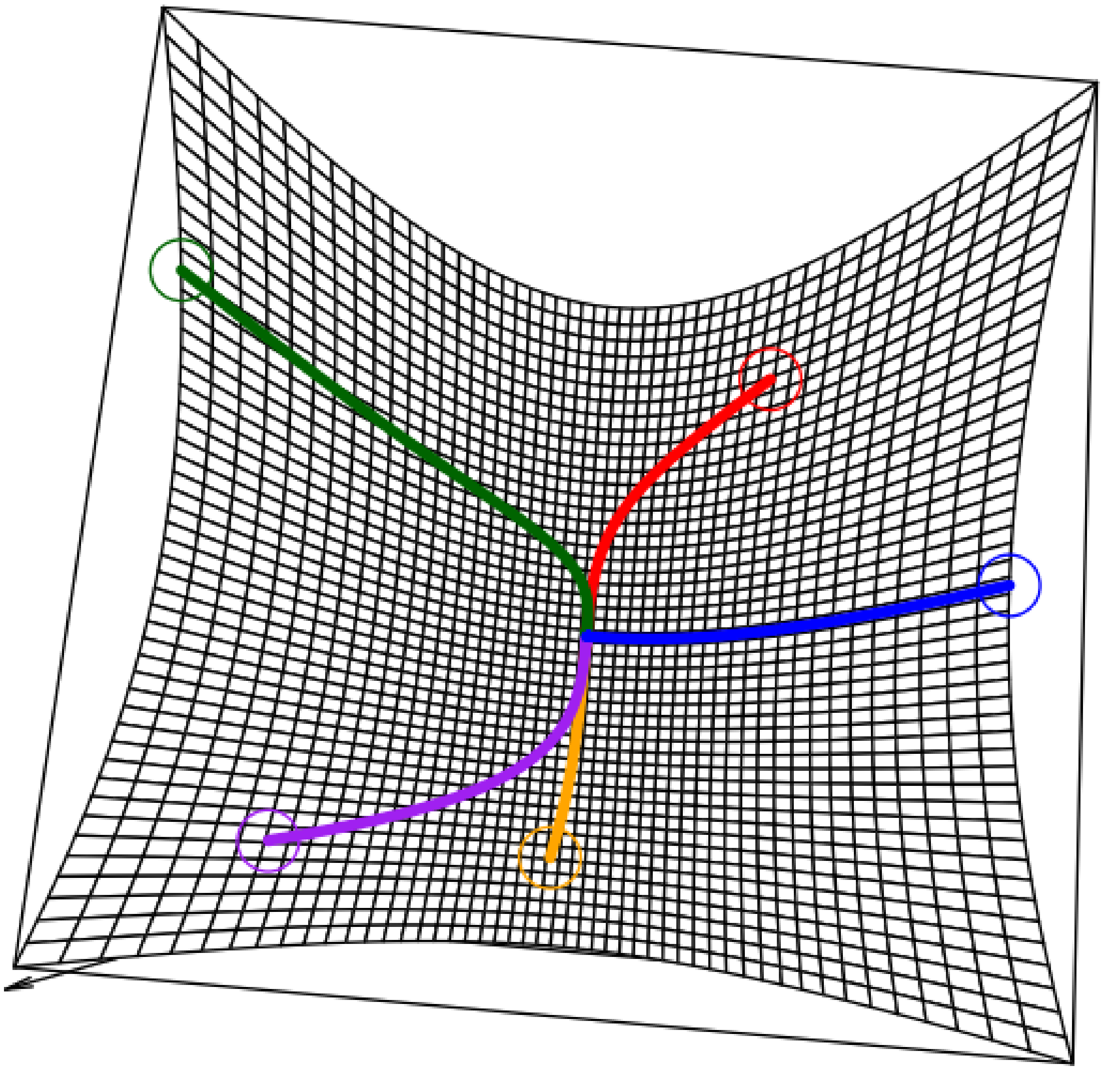
Gradient descent: choose initial $x^{(0)} \in \mathbb{R}^n$, repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \nabla f(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

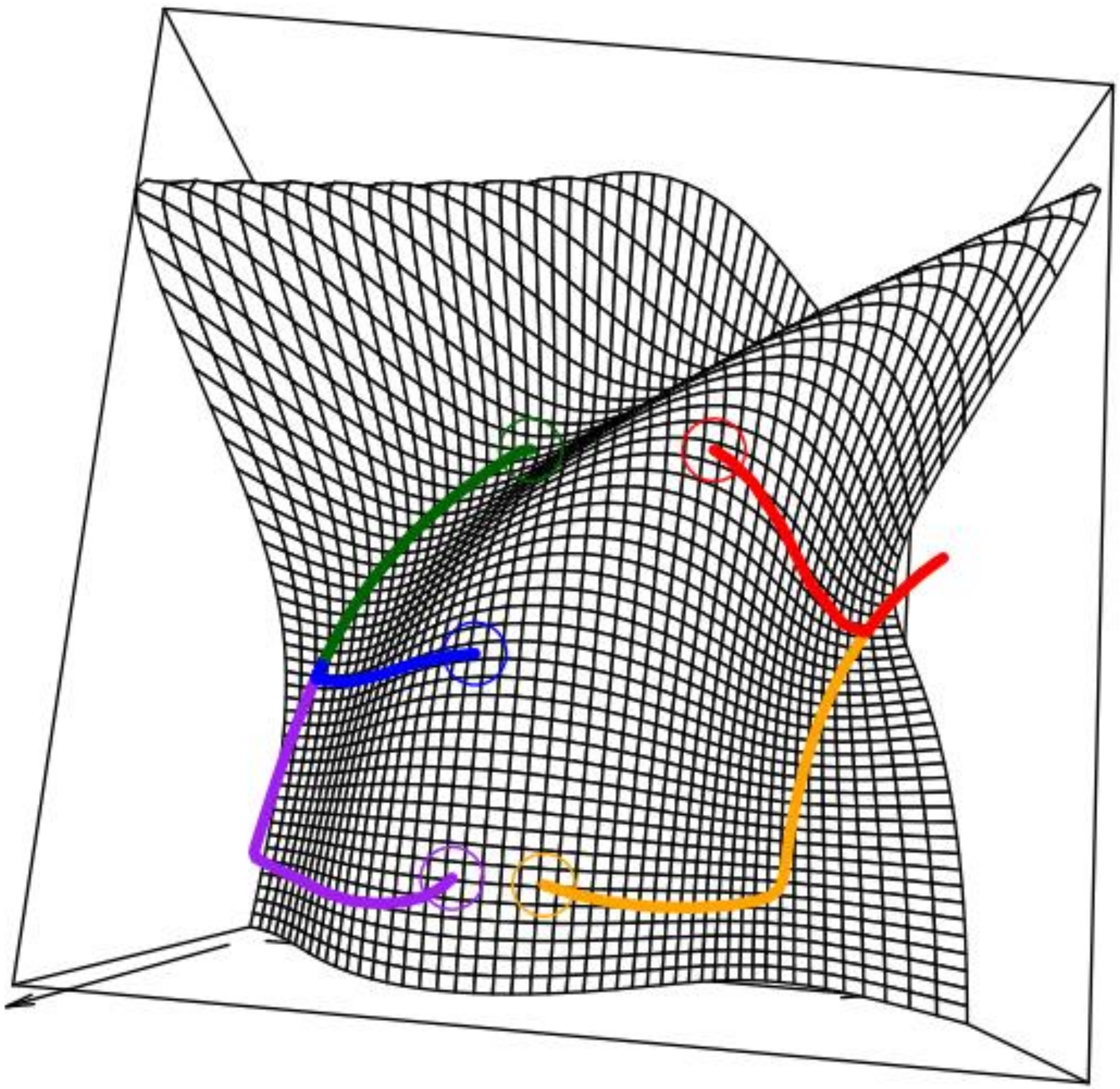
Stop at some point

Here t_k is the step size at iteration k.

Starting Point



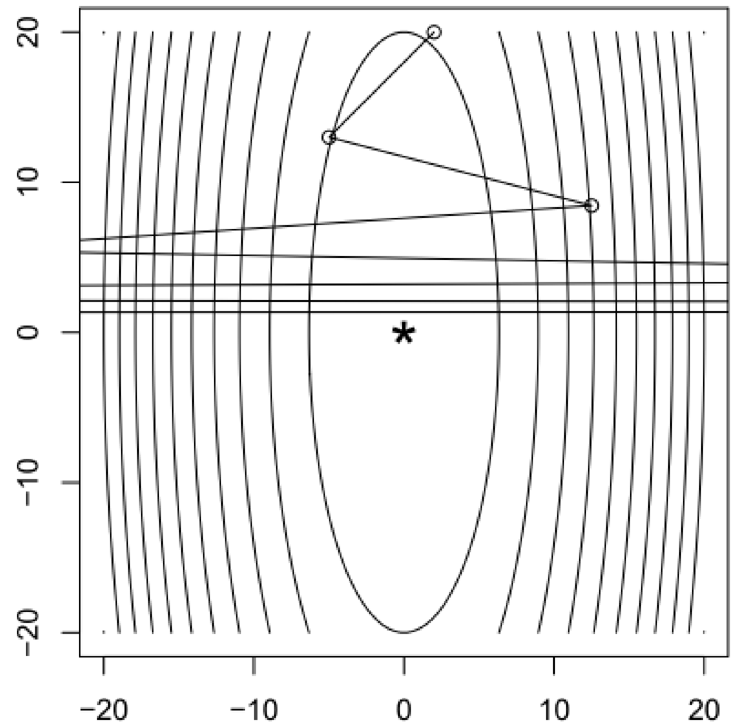
Starting Point



Fixed step size can be too big

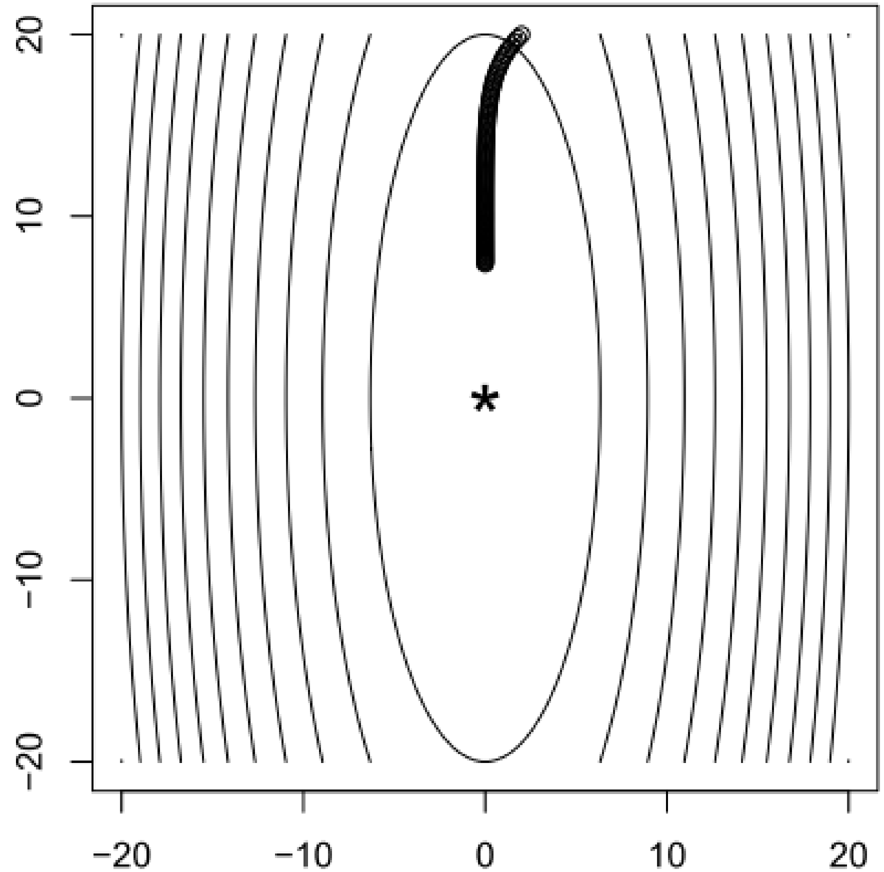
Simply using $t_k = t$ constant for all iterations $k = 1, 2, 3, \ldots,$ can diverge if t is too big.

Consider $f(x) = (10x_1^2 + x_2^2)/2$, gradient descent after 8 steps:



Fixed step size can be too small

Can be slow if t is too small. Same example, gradient descent after 100 steps:



Convergence Rates

A sequence $\{s_i\}$ exhibits **linear** convergence if $\lim_{i\to\infty} s_i = \bar{s}$, and

$$0 < \lim_{i\to\infty} \frac{|s_{i+1} - \bar{s}|}{|s_i - \bar{s}|} = \delta < 1$$

Example: $s_i = cq^i,\ 0 < q < 1$

exponential

$$\frac{|s_{i+1} - \bar{s}|}{|s_i - \bar{s}|} = \frac{cq^{i+1}}{cq^i} = q < 1$$

(log is linear)

Superlinear rate: $\delta = 0$

[faster than linear]

Example: $s_i = \frac{c}{i!}$

$$\frac{|s_{i+1} - \bar{s}|}{|s_i - \bar{s}|} = \frac{ci!}{c(i+1)!} = \frac{1}{i+1} \to 0$$

Sublinear rate: $\delta = 1$

[slower than linear]

Example: $s_i = \frac{c}{i^a}\ ,\ a > 0$

polynomial

$$\frac{|s_{i+1} - \bar{s}|}{|s_i - \bar{s}|} = \frac{ci^a}{c(i+1)^a} = \left(\frac{i}{i+1}\right)^a \to 1$$

Quadratic rate: (log-log is linear)

$$\lim_{i\to\infty} \frac{|s_{i+1} - \bar{s}|}{|s_i - \bar{s}|^2} < \infty$$

Example: $s_i = q^{2^i}\ ,\ 0 < q < 1$

Double-exponential

Convergence Analysis

Assume that $f : \mathbb{R}^n \to \mathbb{R}$ is convex and differentiable, and additionally

$$\|\nabla f(x) - \nabla f(y)\|_2 \le L\|x - y\|_2 \quad \text{for any } x, y$$

That is, ∇f is Lipschitz continuous with constant $L > 0$

Theorem:

Gradient descent with fixed step size $t \le 1/L$ satisfies

$$f(x^{(k)}) - f(x^\star) \le \frac{\|x^{(0)} - x^\star\|_2^2}{2tk}$$

That is, gradient descent with small fixed step size has convergence rate $O(1/k)$

To get $f(x^{(k)}) - f(x^\star) \le \epsilon$, we need $O(1/\epsilon)$ iterations.

Strong Convexity

Strong convexity of f means for some $d > 0$,

$$\nabla^2 f(x) \succeq dI \quad \text{for any } x$$

Under Lipschitz assumption as before, and also assuming strong convexity:

Theorem:

Gradient descent with fixed small step size $t \leq 2/(d+L)$ satisfies

$$f(x^{(k)}) - f(x^\star) \leq c^k \frac{L}{2} \|x^{(0)} - x^\star\|_2^2$$

for some $0 < c < 1$.

That is, rate with strong convexity is $O(c^k)$, exponentially fast!

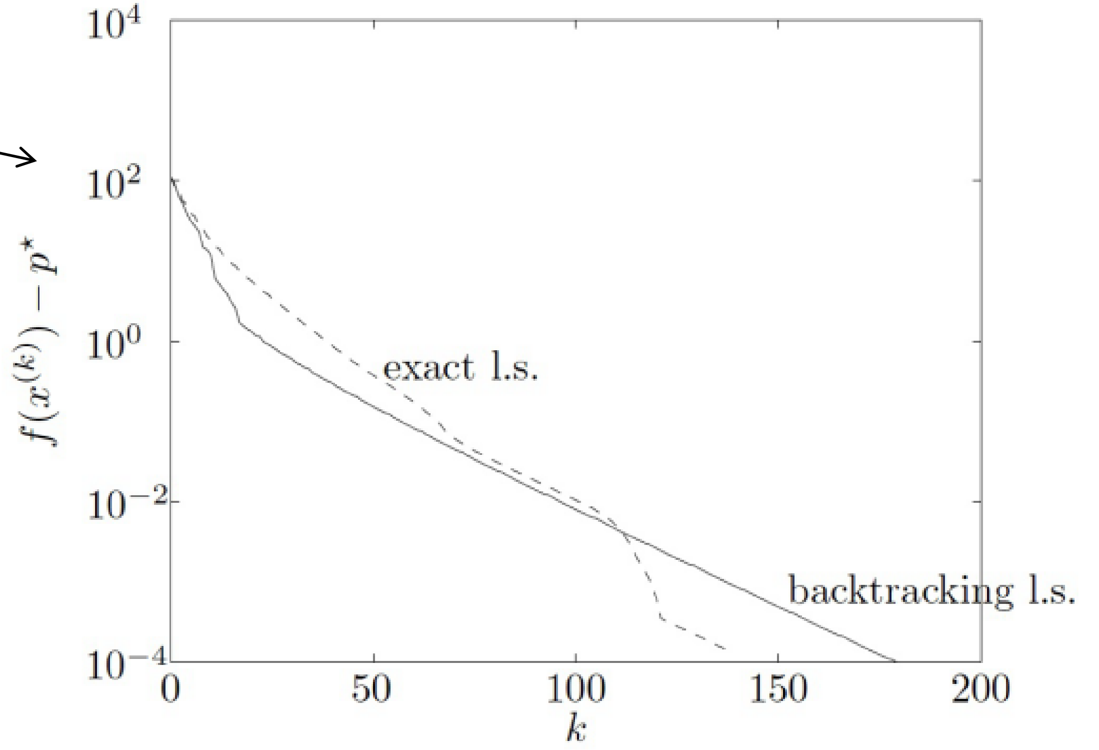
To get $f(x^{(k)}) - f(x^\star) \leq \epsilon$, we need $O(\log(1/\epsilon))$ iterations.

Called linear convergence!

Linear Convergence

Called linear convergence, because looks linear on a semi-log plot:

Log scale



(From B & V page 487)

Conditions

A function f having Lipschitz gradient and being strongly convex can be summarized as:

$$dI \preceq \nabla^2 f(x) \preceq LI \quad \text{for all } x \in \mathbb{R}^n,$$

for constants $L > d > 0$

Lower bounds for small k

First-order method: iterative method, updates $x^{(k)}$ in

$$x^{(0)} + \text{span}\{\nabla f(x^{(0)}), \nabla f(x^{(1)}), \dots \nabla f(x^{(k-1)})\}$$

We already know: $O(1/k)$ rate can be achieved with gradient descent over problem class of convex, differentiable functions with Lipschitz continuous gradients.

Can we create a better first order method than Gradient Descent?

Lower bounds for small k

Assume that $f : \mathbb{R}^n \to \mathbb{R}$ is convex and differentiable, and additionally

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2 \quad \text{for any } x, y$$

That is, ∇f is Lipschitz continuous with constant $L > 0$

Theorem (Nesterov): For any $k \leq (n-1)/2$ and any starting point $x^{(0)}$, there is a function f in the problem class such that any first-order method satisfies

$$f(x^{(k)}) - f(x^\star) \geq \frac{3L\|x^{(0)} - x^\star\|_2^2}{32(k+1)^2}$$

Gradient Descent Variants

Batch gradient descent

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

Batch gradient descent:

Vanilla gradient descent, aka batch gradient descent, computes the gradient of the cost function w.r.t. to the parameters for the entire training dataset:

$$\theta_+ = \theta - \eta\nabla_\theta J(\theta)$$

$$= \theta - \eta\nabla_\theta\left[\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - f_\theta\left(x^{(i)}\right)\right)^2\right]$$

Batch gradient descent

$$\theta_{+} = \theta - \eta \nabla_{\theta} \left[\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} - f_{\theta}\left(x^{(i)}\right) \right)^2 \right]$$

- As we **need to calculate the gradients for the whole dataset** to perform just one update, batch gradient descent can be very slow and is intractable for datasets that do not fit in memory.

- Batch gradient descent also does not allow us to update our model online, i.e. with new examples on-the-fly.

- Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update.

Stochastic gradient descent

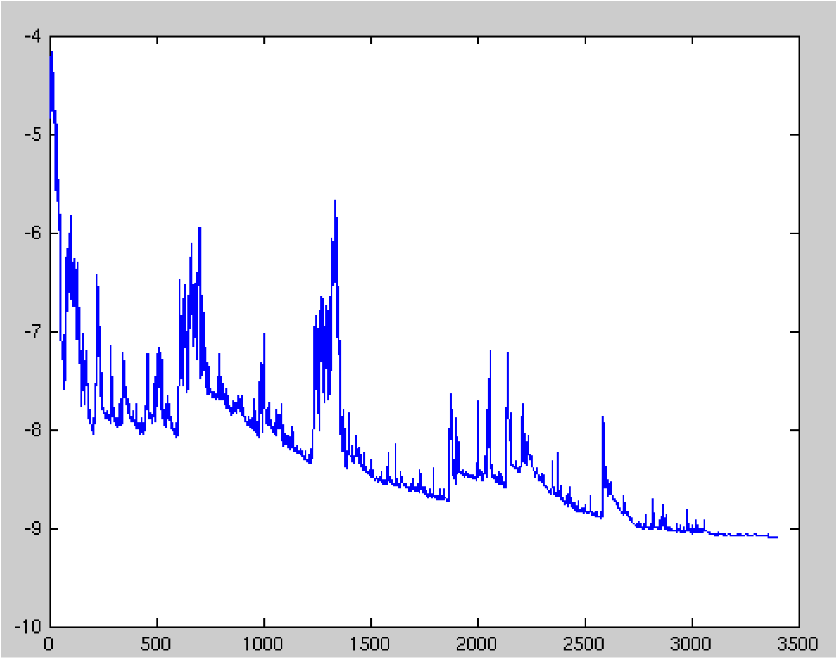
Stochastic gradient descent (SGD) in contrast performs a parameter update for *each* training example $x^{(i)}$ and label $y^{(i)}$:

$$\theta_+ = \theta - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

$$= \theta - \eta \nabla_\theta \left[(y^{(i)} - f_\theta(x^{(i)}))^2 \right]$$

Stochastic gradient descent

- One gradient update for each instance
- Can be used online
- Higher variance than GD
- Can avoid bad local minimum points because of the fluctuation



SGD fluctuation

Mini-batch gradient descent

Mini-batch gradient descent:

Mini-batch gradient descent finally takes the best of both worlds and performs an update for every mini-batch of n training examples:

$$\theta_{+} = \theta - \eta \nabla_{\theta} J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

$$= \theta - \eta \nabla_{\theta} \left[\frac{1}{n} \sum_{j=i}^{i+n} \left(y^{(j)} - f_{\theta}\left(x^{(j)} \right) \right)^2 \right]$$

Challenges:

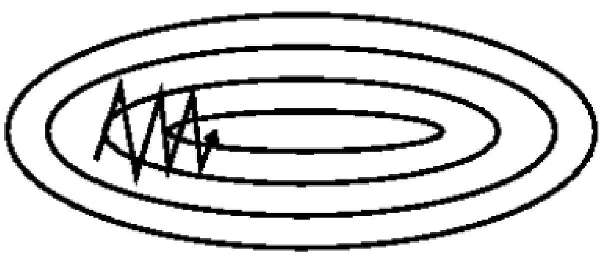
- ❑ Choosing a proper learning rate can be difficult
- ❑ Same learning rate applies to all parameters
- ❑ Can get stuck in saddle points and local minimum points

Momentum method

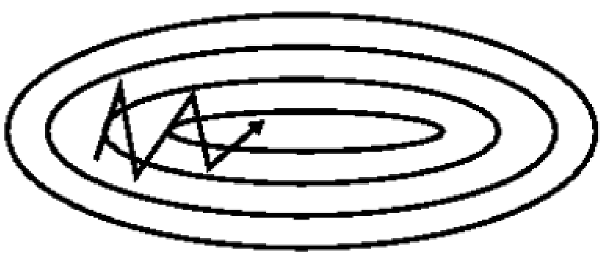
- SGD has trouble navigating areas where the surface curves much more steeply in one dimension than in another.
- In these scenarios, SGD oscillates across the slopes making only slow progress toward the optimum.
- Momentum method dampens by adding a fraction gamma of the update vector of the past time step to the current update vector

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta_t)$$

$$\theta_{t+1} = \theta_t - v_t = \theta_t - \gamma v_{t-1} - \eta \nabla_\theta J(\theta_t)$$



(a) SGD without momentum



(b) SGD with momentum

Source: Genevieve B. Orr

Nesterov's Accelerated Gradient (NAG)

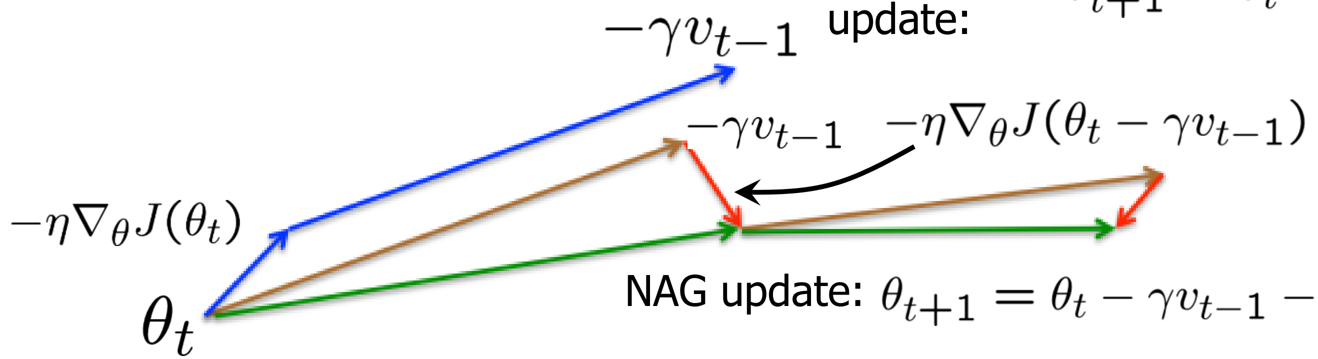
Momentum method:
$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta_t)$$
$$\theta_{t+1} = \theta_t - v_t = \theta_t - \gamma v_{t-1} - \eta \nabla_\theta J(\theta_t)$$

In the momentum method, we move to $\theta_t - \gamma\, v_{t-1}$ and then correct this with the gradient at θ_t, $\eta \nabla_\theta J(\theta_t)$.

NAG method:

In NAG, we us use this new location when calculating the gradient.
$$v_t = \gamma\, v_{t-1} + \eta \nabla_\theta J(\theta_t - \gamma v_{t-1})$$
$$\theta_{t+1} = \theta_t - v_t = \theta_t - \gamma v_{t-1} - \eta \nabla_\theta J(\theta_t - \gamma v_{t-1})$$

Momentum update: $\theta_{t+1} = \theta_t - \gamma v_{t-1} - \eta \nabla_\theta J(\theta_t)$



NAG update: $\theta_{t+1} = \theta_t - \gamma v_{t-1} - \eta \nabla_\theta J(\theta_t - \gamma v_{t-1})$

Source: Hinton's lecture

Adagrad

Adagrad: adapts the learning rates to the parameters: performing larger updates for infrequent, and smaller updates for frequent parameter updates**.**

SGD: Let $g_{t,i} = [\nabla_\theta J(\theta_t)]_i$, the i^{th} cordinate of the gradient

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

Adagrad:

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

$G^t \in \mathbb{R}^{d \times d}$ here is a diagonal matrix where each diagonal element $G^t_{ii} = \sum_{\tau=0}^{t} g^2_{\tau,i}$ is the sum of the squares of the gradients up to time step t.

Adagrad's main weakness is its accumulation of the squared gradients in the denominator: Since every added term is positive, the accumulated sum keeps growing during training. This in turn causes the learning rate to shrink and eventually become infinitesimally small.

Adadelta

Adadelta: a solution to Adagrard's too aggressively decreasing learning rate. Running average instead of full average.

Let $g_{t,i} \doteq [\nabla_\theta J(\theta_t)]_i$, the i^{th} cordinate of the gradient

$$E[g^2]_{t,i} \doteq \gamma E[g^2]_{t-1,i} + (1-\gamma) g_{t,i}^2$$

We now simply replace the diagonal matrix G_t with the decaying average over past squared gradients $E[g^2]_t$:

Variant 1 (RMSprop): [= Root Mean Square Propagation]

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[g^2]_{t,i} + \epsilon}} g_{t,i} = \theta_{t,i} - \frac{\eta}{RMS[g]_{t,i}} g_{t,i}$$

Variant 2:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{RMS[\Delta\theta]_{t-1,i}}{RMS[g]_{t,i}} g_{t,i}$$

Adam = Adaptive moment estimation

In addition to storing an exponentially decaying average of past squared gradients v_t like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Bias correction:

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

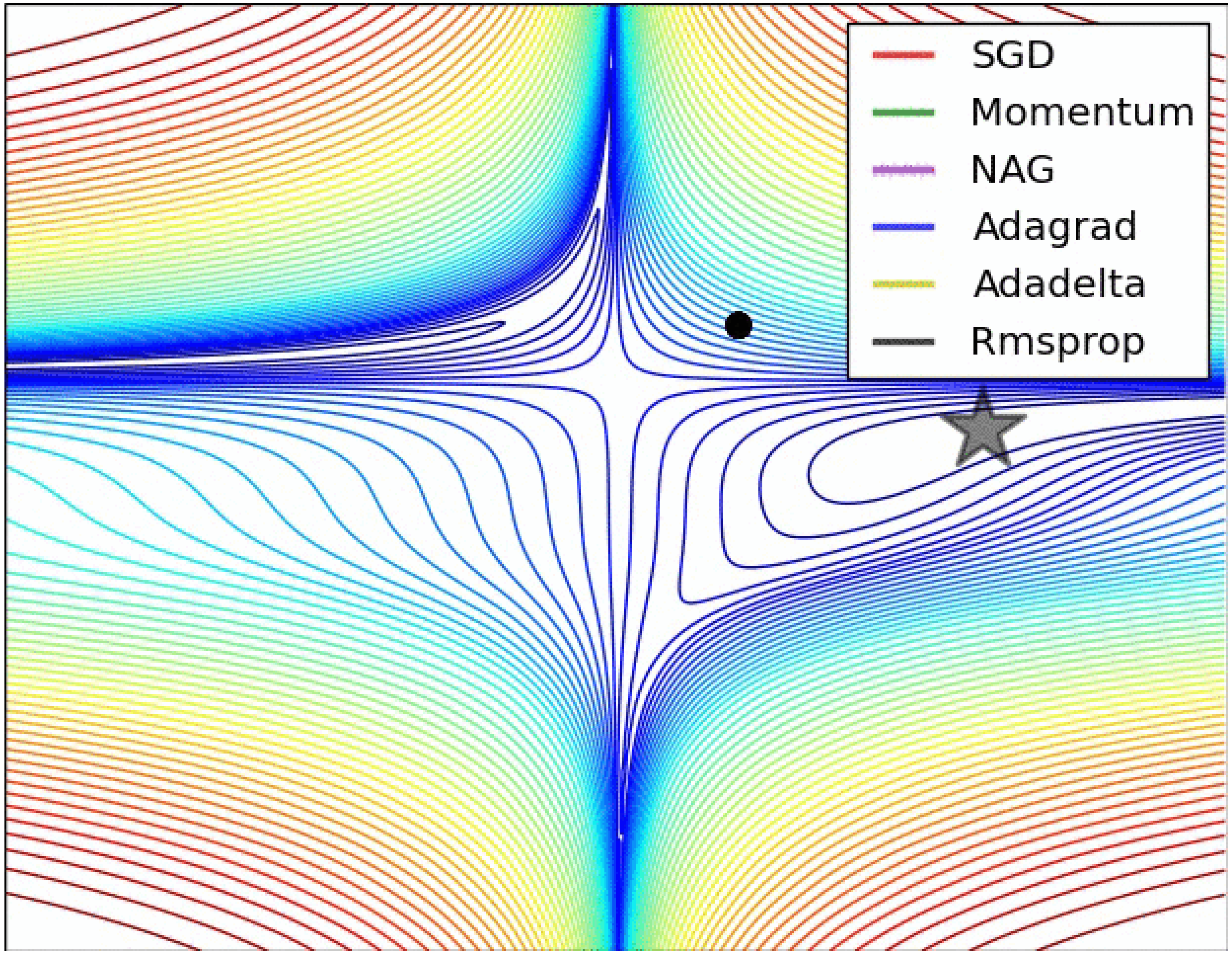
$0 < \beta_1, \beta_2 < 1$ parameters.

Update rule:

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

SGD optimization on loss surface contour.

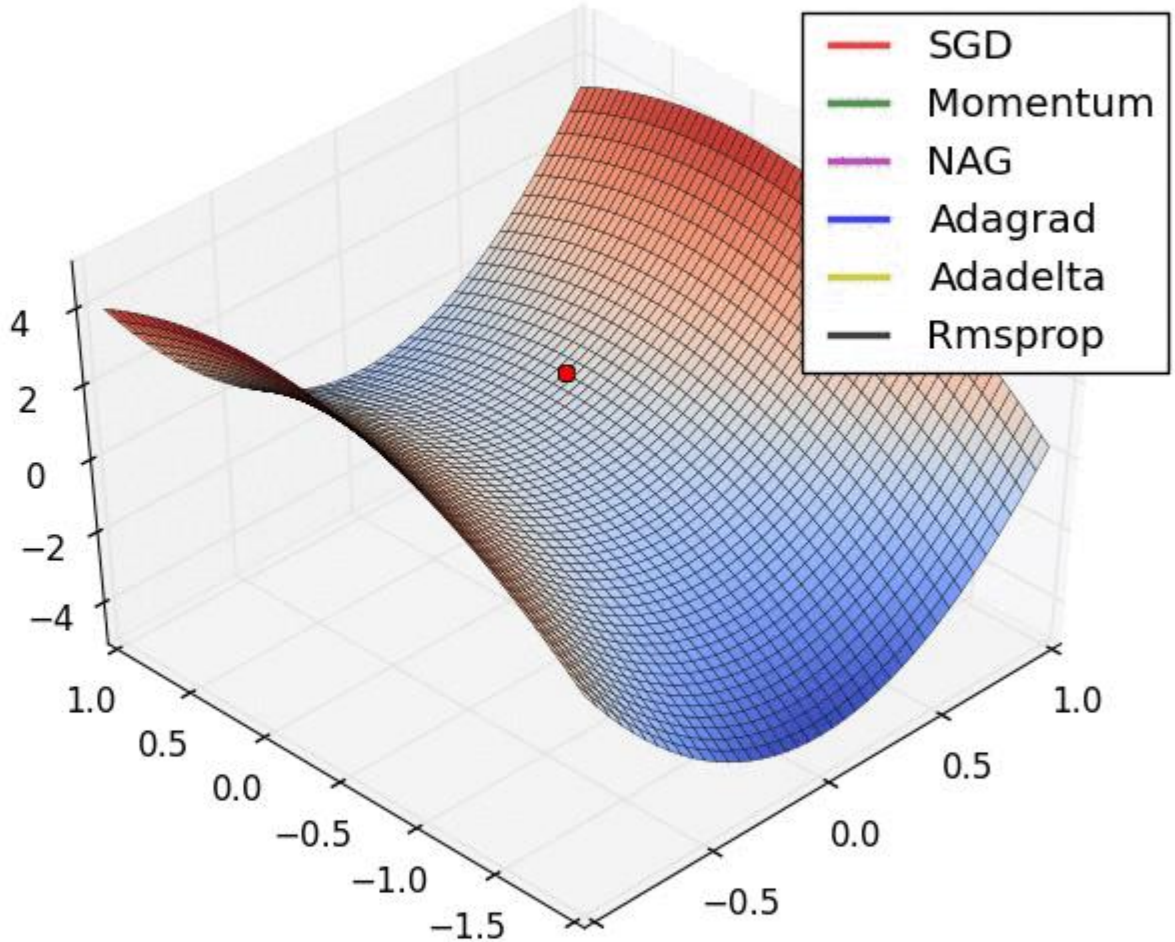
Image credit: http://sebastianruder.com/optimizing-gradient-descent/



As we can see, the adaptive learning-rate methods, i.e. Adagrad, Adadelta, RMSprop, and Adam are most suitable and provide the best convergence for these scenarios

SGD optimization on saddle point

Image credit: http://sebastianruder.com/optimizing-gradient-descent/



As we can see, the adaptive learning-rate methods, i.e. Adagrad, Adadelta, RMSprop, and Adam are most suitable and provide the best convergence for these scenarios.

Thanks for your attention!