CS450: Numerical Analysis Optimization

Howard Hao Yang

Assistant Professor, ZJU-UIUC Institute

13/12/2023

Announcement :

roughly. 3 student → 1 group

A list of papers ~ 40

ZJU-UIUC INSTITUTE

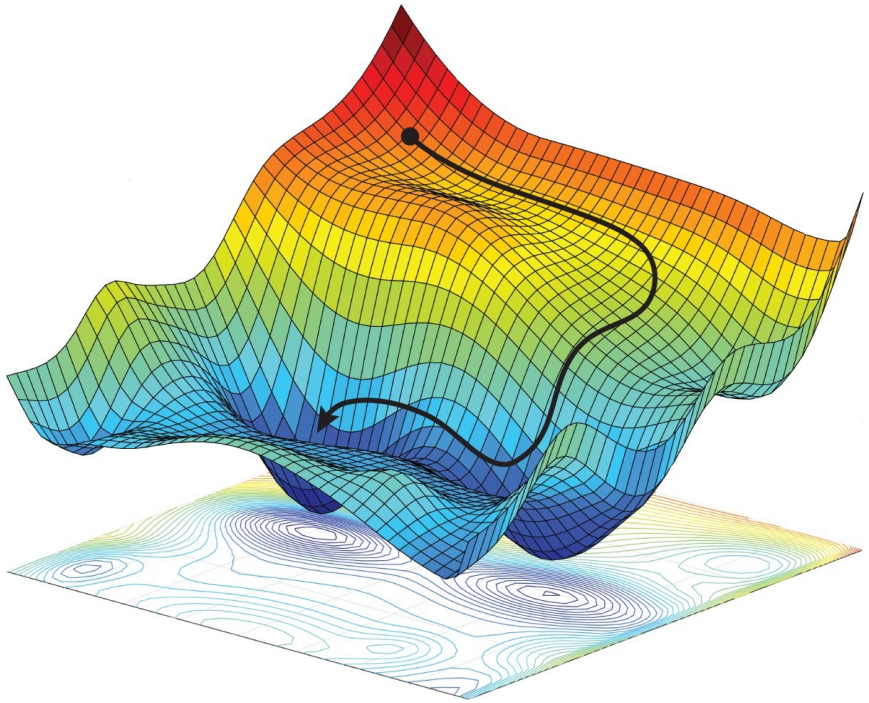
Two more HWs

- Course project: "Review" a technical paper
  - What is the problem the authors aim to solve
  - What is the proposed method
  - What is the advantage
  - What is the setback (in your point of view)
- Potential improvements
- Presentation (Dec. 27 & 29 class):15min talk + 2 min Q&A
- Report

Today's Agenda

- Solving high-dimensional optimization problems
- Gradient descent and its variants
  - Heavy-ball method
  - RMSprop
  - AdaGrad
  - Adam

Optimization



$f \rightarrow$ convex

- The goal is to accomplish the following optimization problem

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \qquad \text{subject to} \quad \boldsymbol{x} \in R^n$$

Iterative Descent Algorithms

- Suppose we have an unconstrained optimization problem,

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \qquad \text{subject to} \quad \boldsymbol{x} \in R^n$$

- Iterative descent algorithms
  - Start with an initial point $\boldsymbol{x}_0$ and construct a sequence $\{\boldsymbol{x}_t\}$ such that

$$f(\boldsymbol{x}_{t+1}) < f(\boldsymbol{x}_t), \qquad t = 0,1, \ldots.$$

  - In each iteration, search in descent direction as follows

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \eta_t \boldsymbol{d}_t$$

where $\boldsymbol{d}_t$ is the descent direction, and η_t is the stepsize

Gradient Descent (GD)

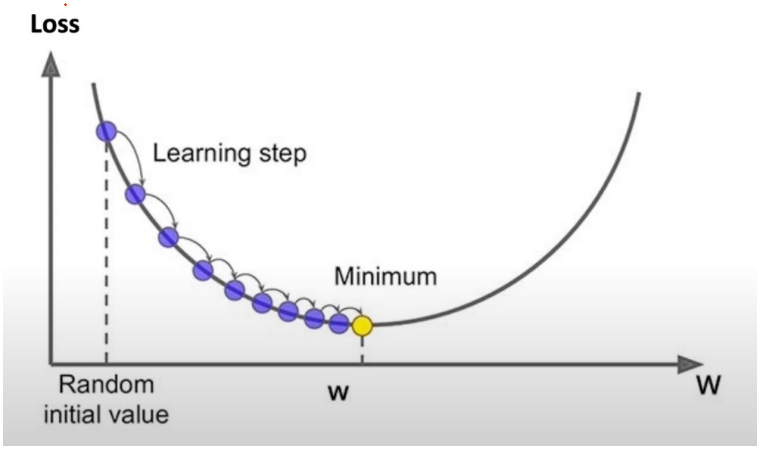
- An important example of iterative descent algorithms: Gradient descent
  - Start with an initial point $\boldsymbol{x}_0$ and update as follows

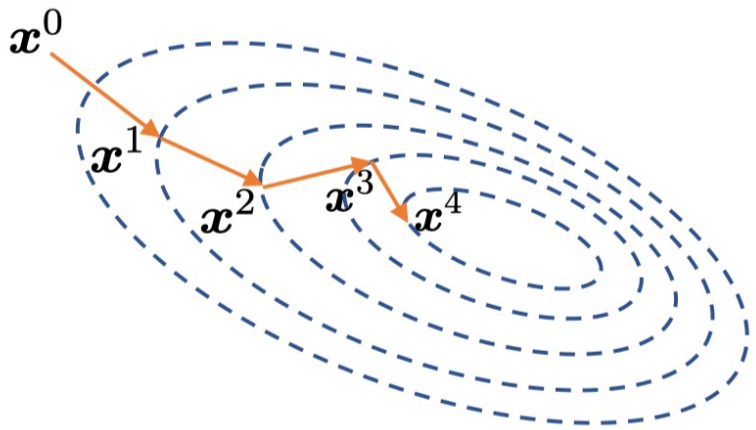
$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta_t \nabla f(\boldsymbol{x}_t)$$

$\eta_t = \eta$

  - Descent direction: $\boldsymbol{d}_t = -\nabla f(\boldsymbol{x}_t)$, a.k.a., steepest descent -- why?

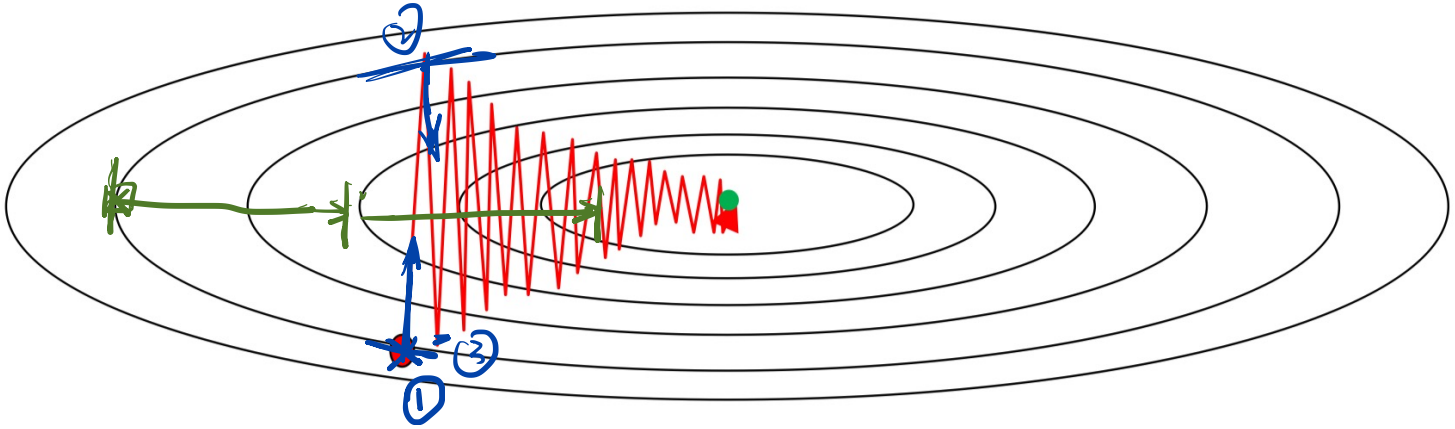
$$\text{argmin}_{\boldsymbol{d}:\|\boldsymbol{d}\|_2 \le 1} \nabla f(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{d} = -\|\nabla f(\boldsymbol{x})\|$$





Is Gradient the Best Direction?

- Issues with GD:
  - GD focuses on improving cost per iteration, which might be too "short-sighted"
  - GD might sometimes zigzag/experience abrupt changes



Is Gradient the Best Direction? (Cont'd)

ZJU-UIUC INSTITUTE
Zhejiang University/University of Illinois at Urbana-Champaign Institute

- Issues with GD:
  - GD focuses on improving cost per iteration, which might be too "short-sighted"
  - GD might sometimes zigzag/experience abrupt changes
- Possible solutions:
  - Exploit information from history/past iterations
  - Add buffer (like momentum) to yield smoother trajectory

Heavy-Ball Method

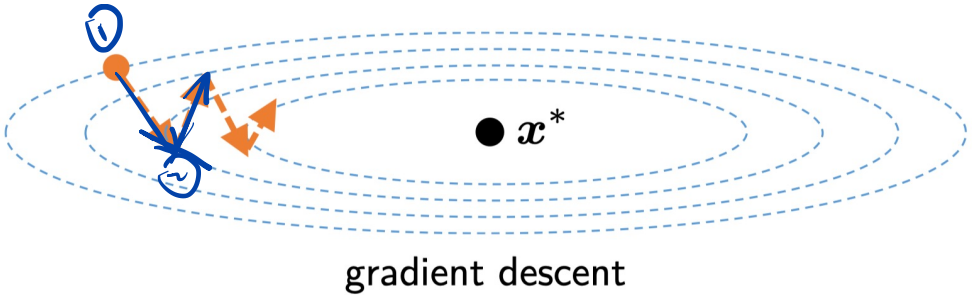
- Given an unconstrained optimization problem,

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{x} \in R^n$$

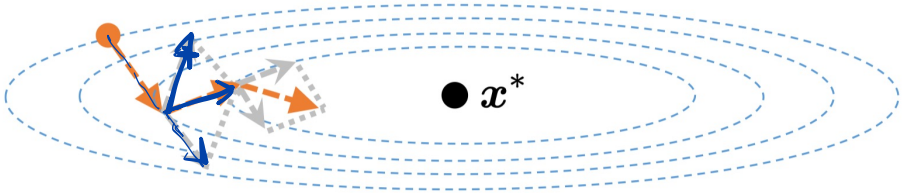
- Heavy-ball method runs as
  - Start with an initial point $\boldsymbol{x}_0$ and update as follows

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta_t \nabla f(\boldsymbol{x}_t) + \theta_t(\boldsymbol{x}_t - \boldsymbol{x}_{t-1})$$

  - Add inertia to the "ball" (i.e., include momentum term) to mitigate zigzaging

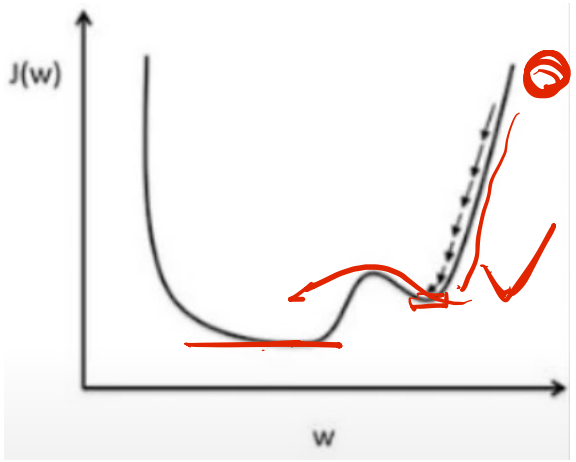


gradient descent



heavy-ball method

Heavy-Ball Method (Cont'd)



- Heavy-ball method
  - Add inertia to the "ball" (i.e., include momentum term)

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta_t \nabla f(\boldsymbol{x}_t) + \theta_t(\boldsymbol{x}_t - \boldsymbol{x}_{t-1})$$

  - Achieve faster convergence
  - Better generalization capability (for machine learning applications)

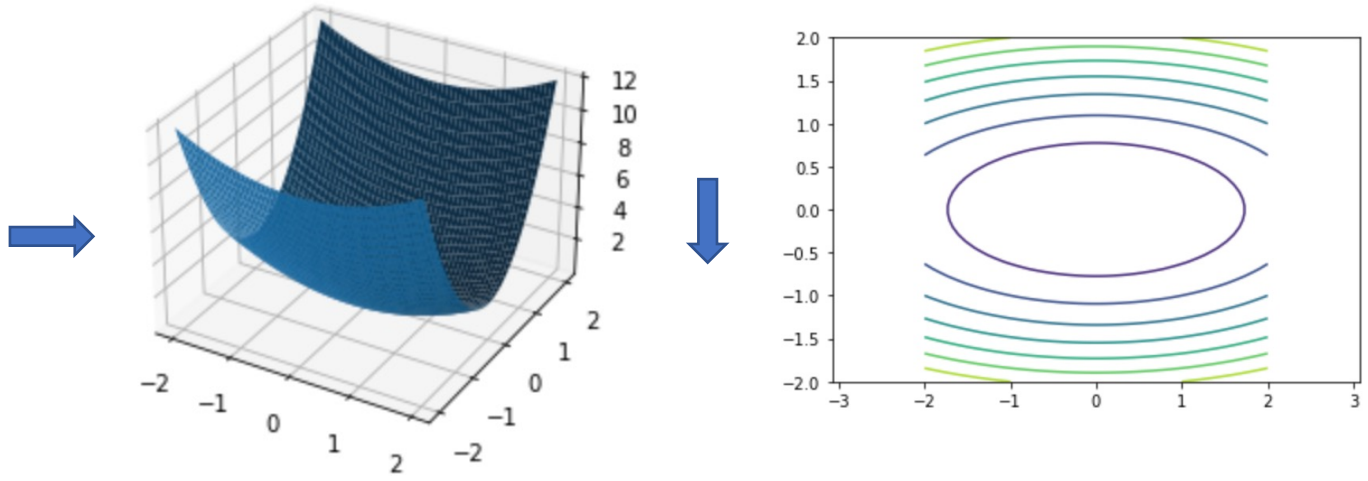
Illustrative Example

- Given an unconstrained optimization problem,

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^*)^T \boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x}^*)$$

```python
def f(x):
    return 0.5*x[0]**2 + 2.5*x[1]**2

def df(x):
    return np.array([x[0], 5*x[1]])
```

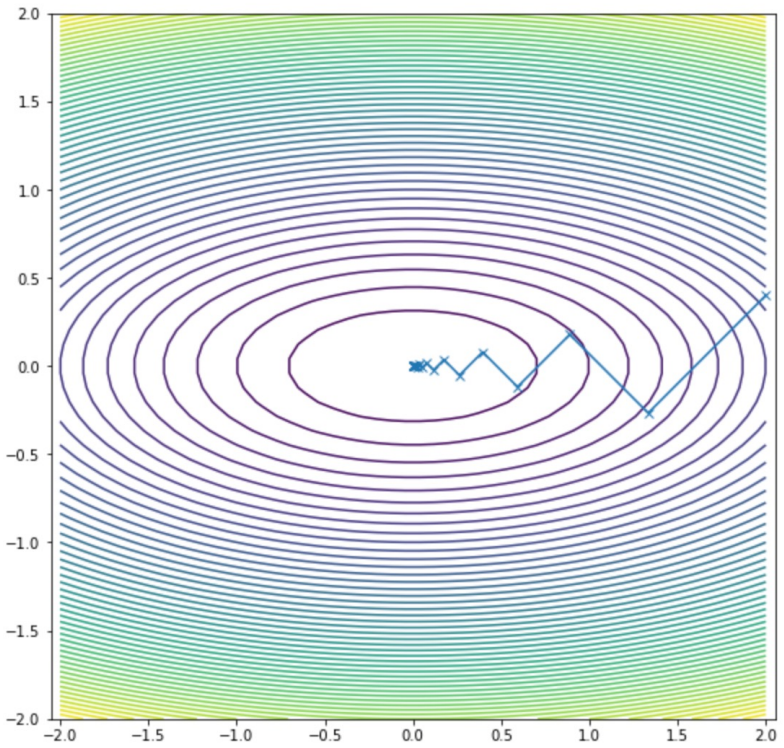
```python
fig = pt.figure()
ax = fig.gca(projection="3d")

xmesh, ymesh = np.mgrid[-2:2:50j,-2:2:50j]
fmesh = f(np.array([xmesh, ymesh]))
ax.plot_surface(xmesh, ymesh, fmesh)
```



Illustrative Example (Cont'd)

- Steepest descent

```
#clear
x = guesses[-1]
s = -df(x)

def f1d(alpha):
    return f(x + alpha*s)

alpha_opt = sopt.golden(f1d)
next_guess = x + alpha_opt * s
guesses.append(next_guess)

print(next_guess)
```




```
for i, guess in enumerate(guesses):
    print(i, la.norm(guess, 2))
```

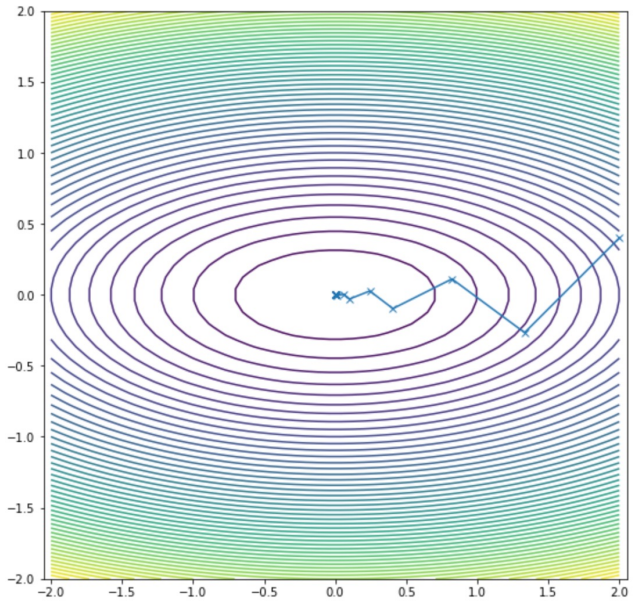
```
0 2.039607805437114
1 1.3597385362818037
2 0.9064923598953564
3 0.6043282354184075
4 0.402885493365651
5 0.268590327422494
6 0.1790602196535349
7 0.11937347850578087
8 0.07958231961357493
9 0.05305487971599619
10 0.03536991972494859
11 0.023579946499640153
12 0.0157199643742649
13 0.01047997619145283
14 0.006986650869603477
15 0.004657767182597052
16 0.00310517817085647
17 0.0020701187313361103
18 0.001380079194276325 7
19 0.0009200527639282007
20 0.0006133685351837453
21 0.0004089123383293087
22 0.00027260823928739256
23 0.00018173881650443178
24 0.000121159217718 1258
25 8.077280684766319e-05
26 5.3848541322305716e-05
27 3.589902516077526e-05
28 2.393268515749757e-05
29 1.595512232752962e-05
30 1.0636748894671809e-05
31 7.0911654939472895e-06
32 4.727443954112351e-06
33 3.1516290931456465e-06
34 2.101086219381249e-06
35 1.4007240373068642e-06
36 9.33816096573883e-07
37 6.22544015961566e-07
```

Illustrative Example (Cont'd)

ZJU-UIUC INSTITUTE
Zhejiang University/University of Illinois at Urbana-Champaign Institute

- Heavy-ball method

```
#clear
x = guesses[-1]
s = -df(x)

def f1d(alpha):
    return f(x + alpha*s)

alpha_opt = sopt.golden(f1d)
next_guess = x + alpha_opt * s

if len(guesses) >= 2:
    next_guess = next_guess + beta * (guesses[-1] - guesses[-2])

guesses.append(next_guess)

print(next_guess)
```




```
for i, guess in enumerate(guesses):
    print(i, la.norm(guess, 2))
```

```
0 2.039607805437114
1 1.3597385362818037
2 0.8296957665641046
3 0.41785029923493805
4 0.24532582175768264
5 0.10508154968346078
6 0.059791922016322774
7 0.015328144502394192
8 0.006032673968378717
9 0.0014933302414913351
10 0.0005424642572479524
11 0.0004793689076159881 5
12 0.0003323789236199735
13 2.406707379103359e-05
14 4.320675845534447e-05
15 1.3649136343804876e-05
16 5.333125657925825e-06
17 2.656196917967379e-06
18 1.2987876444411142e-06
19 6.899983272977852e-07
20 3.833795500074894e-07
21 1.3300046913355573e-07
```

Working Example

- Given an unconstrained optimization problem,

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^*)^T \boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x}^*)$$

for some $n \times n$ positive definite matrix $\boldsymbol{A}$.

- What is the gradient of this? $\nabla f(\boldsymbol{x}) = \boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x}^*)$
- How does Gradient Descent proceed?
  - Start with an initial point $\boldsymbol{x}_0$ and update the parameter as follows

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta_t \boldsymbol{A}(\boldsymbol{x}_t - \boldsymbol{x}^*)$$

Working Example (Cont'd)

- Calculate the convergence rate: According to the GD update rule, we have

$$\boldsymbol{e}_{t+1} = \boldsymbol{x}_{t+1} - \boldsymbol{x}^* = \boldsymbol{x}_t - \boldsymbol{x}^* - \eta_t \boldsymbol{A}(\boldsymbol{x}_t - \boldsymbol{x}^*) = (\boldsymbol{I} - \eta_t \boldsymbol{A})\boldsymbol{e}_t$$

- As such,

$$\|\boldsymbol{e}_{t+1}\| \leq \|\boldsymbol{I} - \eta_t \boldsymbol{A}\| \cdot \|\boldsymbol{e}_t\|$$

- How do we bound $\|\boldsymbol{I} - \eta_t \boldsymbol{A}\|$?

Working Example (Cont'd)

- Calculate the convergence rate: According to the GD update rule, we have

$$\boldsymbol{e}_{t+1} = \boldsymbol{x}_{t+1} - \boldsymbol{x}^* = \boldsymbol{x}_t - \boldsymbol{x}^* - \eta_t \boldsymbol{A}(\boldsymbol{x}_t - \boldsymbol{x}^*) = (\boldsymbol{I} - \eta_t \boldsymbol{A})\boldsymbol{e}_t$$

- As such,

$$\|\boldsymbol{e}_{t+1}\| \leq \|\boldsymbol{I} - \eta_t \boldsymbol{A}\| \cdot \|\boldsymbol{e}_t\|$$

η_t

- How do we bound $\|\boldsymbol{I} - \eta_t \boldsymbol{A}\|$?

$$\|\boldsymbol{I} - \eta_t \boldsymbol{A}\| = \max\{|1 - \eta_t \lambda_1|, |1 - \eta_t \lambda_n|\} = \frac{\lambda_1 - \lambda_n}{\lambda_1 + \lambda_n}$$

- The error then goes down by

$$\|\boldsymbol{e}_t\| \leq \left(\frac{\lambda_1 - \lambda_n}{\lambda_1 + \lambda_n}\right)^t \|\boldsymbol{e}_0\|$$

Working Example (Cont'd)

ZJU-UIUC INSTITUTE
Zhejiang University/University of Illinois at Urbana-Champaign Institute

- Given an unconstrained optimization problem,

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^*)^T \boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x}^*)$$

for some $n \times n$ positive definite matrix $\boldsymbol{A}$.

- What is the gradient of this? $\nabla f(\boldsymbol{x}) = \boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x}^*)$
- How does Heavy-Ball method proceed?
  - Start with an initial point $\boldsymbol{x}_0$ and update the parameter as follows

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta_t \boldsymbol{A}(\boldsymbol{x}_t - \boldsymbol{x}^*) + \theta_t(\boldsymbol{x}_t - \boldsymbol{x}_{t-1})$$

Working Example (Cont'd)

- Calculate the convergence rate: According to the update rule, we have

$$\boldsymbol{e}_{t+1} = \boldsymbol{x}_{t+1} - \boldsymbol{x}^* = \boldsymbol{e}_t - \eta_t \boldsymbol{A}\boldsymbol{e}_t + \theta_t(\boldsymbol{e}_t - \boldsymbol{e}_{t-1})$$

Heavy ball

- Pack the above into the following form

$$\begin{bmatrix} \boldsymbol{e}_{t+1} \\ \boldsymbol{e}_t \end{bmatrix} = \begin{bmatrix} (1+\theta_t)\boldsymbol{I} & -\theta_t \boldsymbol{I} \\ \boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{e}_t \\ \boldsymbol{e}_{t-1} \end{bmatrix} - \begin{bmatrix} \eta_t \boldsymbol{A}\boldsymbol{e}_t \\ \boldsymbol{0} \end{bmatrix}$$

$$= \begin{bmatrix} (1+\theta_t)\boldsymbol{I} - \eta_t \boldsymbol{A} & -\theta_t \boldsymbol{I} \\ \boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{e}_t \\ \boldsymbol{e}_{t-1} \end{bmatrix}$$

- Therefore

$$\left\| \begin{bmatrix} \boldsymbol{e}_{t+1} \\ \boldsymbol{e}_t \end{bmatrix} \right\| \leq \left\| \begin{bmatrix} (1+\theta_t)\boldsymbol{I} - \eta_t \boldsymbol{A} & -\theta_t \boldsymbol{I} \\ \boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \right\| \cdot \left\| \begin{bmatrix} \boldsymbol{e}_t \\ \boldsymbol{e}_{t-1} \end{bmatrix} \right\|$$

Working Example (Cont'd)

- How to bound $\left\| \begin{bmatrix} (1+\theta_t)\boldsymbol{I} - \eta_t \boldsymbol{A} & -\theta_t \boldsymbol{I} \\ \boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \right\|$? The following holds:

$$\left\| \begin{bmatrix} (1+\theta_t)\boldsymbol{I} - \eta_t \boldsymbol{A} & -\theta_t \boldsymbol{I} \\ \boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \right\| \leq \max_{1 \leq i \leq n} \left\| \begin{bmatrix} 1+\theta_t - \eta_t \lambda_i & -\theta_t \\ 1 & 0 \end{bmatrix} \right\|$$

$\lambda_1 = 9$
$\lambda_n = 1$

- Homework problem: show that

$$\max_{1 \leq i \leq n} \left\| \begin{bmatrix} 1+\theta_t - \eta_t \lambda_i & -\theta_t \\ 1 & 0 \end{bmatrix} \right\| \leq \frac{\sqrt{\lambda_1/\lambda_n} - 1}{\sqrt{\lambda_1/\lambda_n} + 1}$$

- Therefore, heavy-ball converges at the rate

$$\|\boldsymbol{e}_t\| \leq \left(\frac{\sqrt{\lambda_1/\lambda_n} - 1}{\sqrt{\lambda_1/\lambda_n} + 1} \right)^t \|\boldsymbol{e}_0\|$$

$\Leftrightarrow$

$\|e_t\| \leq \left(\frac{\lambda_1 - \lambda_n}{\lambda_1 + \lambda_n} \right)^t \|e_0\|$

$\left(\frac{3-1}{3+1} \right)^t = \left(\frac{2}{4} \right)^t$

$\left(\frac{8}{10} \right)^t$

Nesterov's Method

- Given an unconstrained optimization problem,

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \qquad \text{subject to} \quad \boldsymbol{x} \in R^n$$

- Nesterov's method runs as
  - Start with an initial point $\boldsymbol{x}_0$ and update the parameter as follows

(a) $$\boldsymbol{x}_{t+1} = \boldsymbol{y}_t - \eta_t \nabla f(\boldsymbol{y}_t)$$

(b) $$\boldsymbol{y}_{t+1} = \boldsymbol{x}_{t+1} + \frac{t}{t+3}(\boldsymbol{x}_{t+1} - \boldsymbol{x}_t)$$

  - Alternates between gradient updates and proper extrapolation
  - Every iteration takes nearly the same cost as GD
  - Not a decent method (we might not have $f(\boldsymbol{x}_{t+1}) < f(\boldsymbol{x}_t)$)

Convergence Rate

- Assume f is convex and L −smooth
- **Theorem**: Gradient descent with a constant step size $\eta < 1/L$ satisfies

$$f(\boldsymbol{x}_t) - f^* \leq \frac{\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|_2}{2\eta t}$$

- **Theorem**: Nesterov's method with a constant step size $\eta < 1/L$ satisfies

$$f(\boldsymbol{x}_t) - f^* \leq \frac{2\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|_2}{\eta(t+1)^2}$$

  - Iteration complexity: $O(\frac{1}{\sqrt{\epsilon}})$ – much faster than GD

RMSprop

- Gradients can vary widely even though parameters have the same scale.
- RMSprop scales the (stochastic) gradients by the inverse of a moving average, termed Root-Mean-Square gradient. Defined as

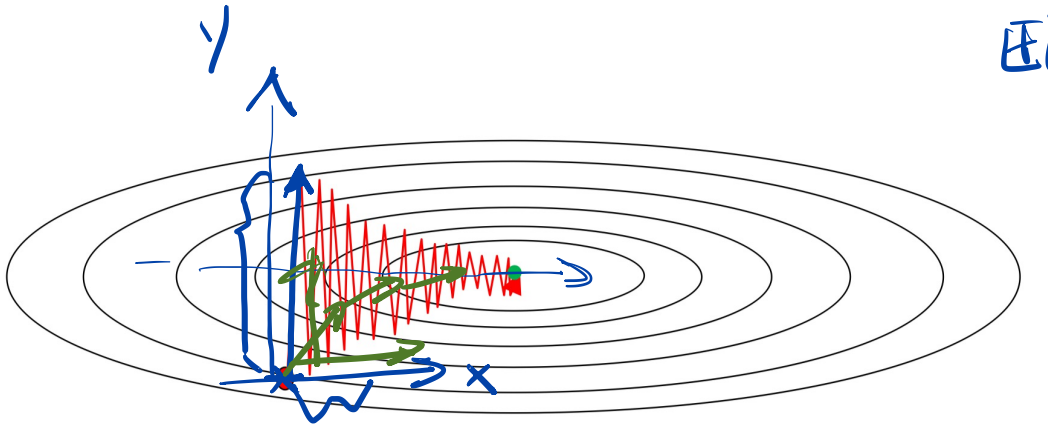
$$\boldsymbol{v}_{t+1} = \beta \boldsymbol{v}_t + (1-\beta)\boldsymbol{g}_t^2$$

  - $\boldsymbol{g}_t$ is the minibatch stochastic gradient evaluated at step t
  - $0 \leq \beta \leq 1$ is a moving average decay factor
  - $\boldsymbol{g}_t^2$ is an element-wise square of $\boldsymbol{g}_t$

- RMSprop

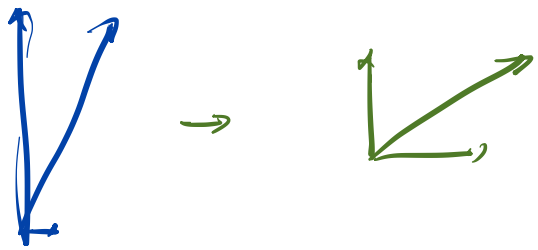
$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \frac{\boldsymbol{g}_t}{\sqrt{\boldsymbol{v}_t}}$$

Interpretation

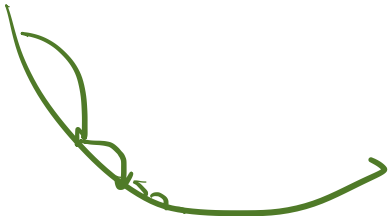


- Consider a 2-D optimization with zigzag trajectory:
  - What does the gradient look like?
  - How would RMSprop improve the optimization/training process?

AdaGrad



- AdaGrad is similar to RMSprop, but uses the cumulative sum of squared gradients, i.e.,



$$\boldsymbol{v}_t = \sum_{j=1}^{t} \boldsymbol{g}_j^2$$

- AdaGrad

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \frac{\boldsymbol{g}_t}{\sqrt{\boldsymbol{v}_t}}$$

  - The scale of $\boldsymbol{v}_t$ tends to grow linearly overtime, so AdaGrad decreases its effective learning rate over time as $\sqrt{t}$

Momentum+RMSprop≈ADAM

ZJU-UIUC INSTITUTE

- Compute moving averages of the gradient and squared gradient
- Treat them as moments

$$\boldsymbol{m}_{t+1} = \beta_1 \boldsymbol{m}_t + (1-\beta_1)\boldsymbol{g}_t$$

$$\boldsymbol{v}_{t+1} = \beta_2 \boldsymbol{v}_t + (1-\beta_2)\boldsymbol{g}_t^2$$

- Adam (Adaptive Momentum Estimator): normalize the momentum update as

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \frac{\boldsymbol{m}_t}{\sqrt{\boldsymbol{v}_t}}$$

  - The parameters are typically chosen as $\beta_1 \approx 0.9$ and $\beta_2 \approx 0.999$

ADAM Bias Correction

- Compute moving averages of the gradient and squared gradient

$$\boldsymbol{m}_{t+1} = \beta_1 \boldsymbol{m}_t + (1-\beta_1)\boldsymbol{g}_t$$

$$\boldsymbol{v}_{t+1} = \beta_2 \boldsymbol{v}_t + (1-\beta_2)\boldsymbol{g}_t^2$$

- Compute moving averages of the gradient and squared gradient

$$\boldsymbol{m}_t^{corr} = \frac{\boldsymbol{m}_t}{1-\beta_1^t}$$

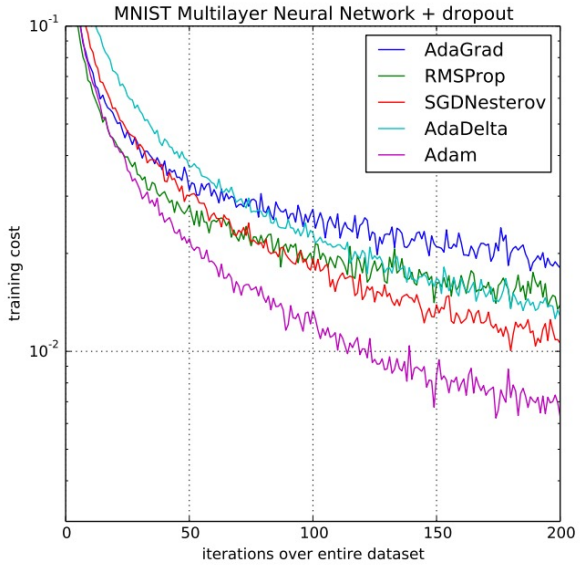
$$\boldsymbol{v}_t^{corr} = \frac{\boldsymbol{v}_t}{1-\beta_2^t}$$

- Rectified Adam update

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \frac{\boldsymbol{m}_t^{corr}}{\sqrt{\boldsymbol{v}_t^{corr}} + \epsilon}$$

ADAM

TITLE	CITED BY	YEAR
Adam: A method for stochastic optimization DP Kingma, J Ba arXiv preprint arXiv:1412.6980	166336	2014



- According to the authors…
  - Computationally efficient
  - Little memory requirements
  - Well suited for problems that are large in terms of data and/or parameters
  - Appropriate for problems with noisy and/or sparse gradients
  - Hyperparameters usually requires little or no tuning

Learning Objectives

- Optimization is everywhere, and is particularly relevant to machine learning
- Convexity is a borderline between “easy” and “hard” optimization approaches
- Iterative descent approach for reaching optimality
- Gradient descent and its variants