Optimization for Training II

Second-Order Methods Training algorithm

OPTIMIZATION METHODS

Topics: Types of optimization methods.

- Practical optimization methods breakdown into two categories:
    1. First-order methods
    2. Second-order methods

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{a}) + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{a})(\boldsymbol{\theta} - \boldsymbol{a}) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{a})^\top \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{a})$$

- Today we will focus on first-order methods

OPTIMIZATION METHODS

Topics: Types of optimization methods.

- Second order methods we will consider:
  1. Newton's method
  2. Conjugate gradient method
  3. BFGS (L-BFGS)
  4. Hessian-Free optimization

NEWTON'S METHOD

- Considering the quadratic approximation to the loss function:

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{a}) + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{a})(\boldsymbol{\theta} - \boldsymbol{a}) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{a})^\top \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{a})$$

$$\nabla_{\boldsymbol{\theta}} \hat{J}(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{a}) + \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{a})$$

$$\nabla_{\boldsymbol{\theta}} \hat{J}(\boldsymbol{\theta}) = 0 \Rightarrow \boxed{\boldsymbol{\theta}^* = \boldsymbol{a} - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{a})}$$

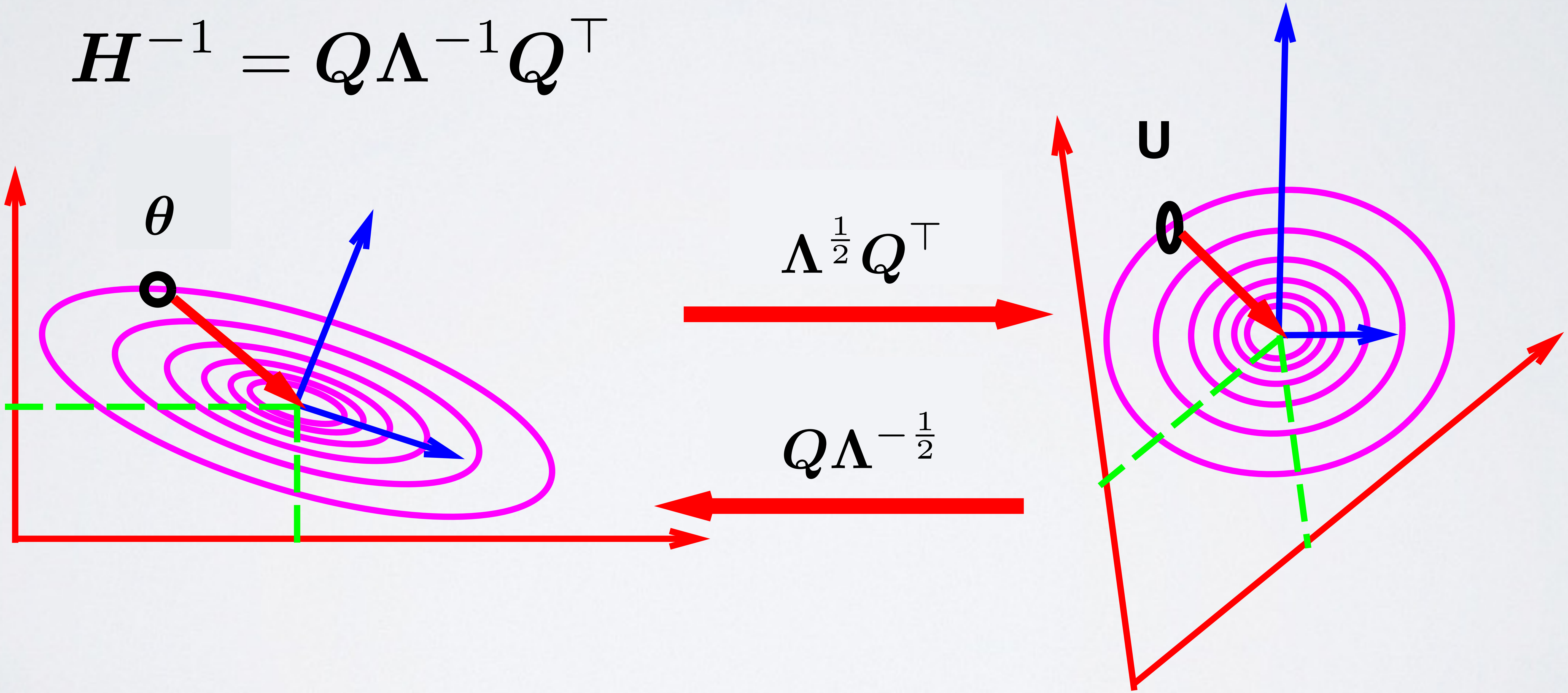
Newton's method update

NEWTON'S METHOD

- How to think about Newton's method

$$\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top \qquad \text{(eigendecomposition of } \boldsymbol{H}\text{)}$$

$$\boldsymbol{H}^{-1} = \boldsymbol{Q}\boldsymbol{\Lambda}^{-1}\boldsymbol{Q}^\top$$



NEWTON'S METHOD

- Newton's method as an algorithm:

Algorithm 1 Newton update at time t

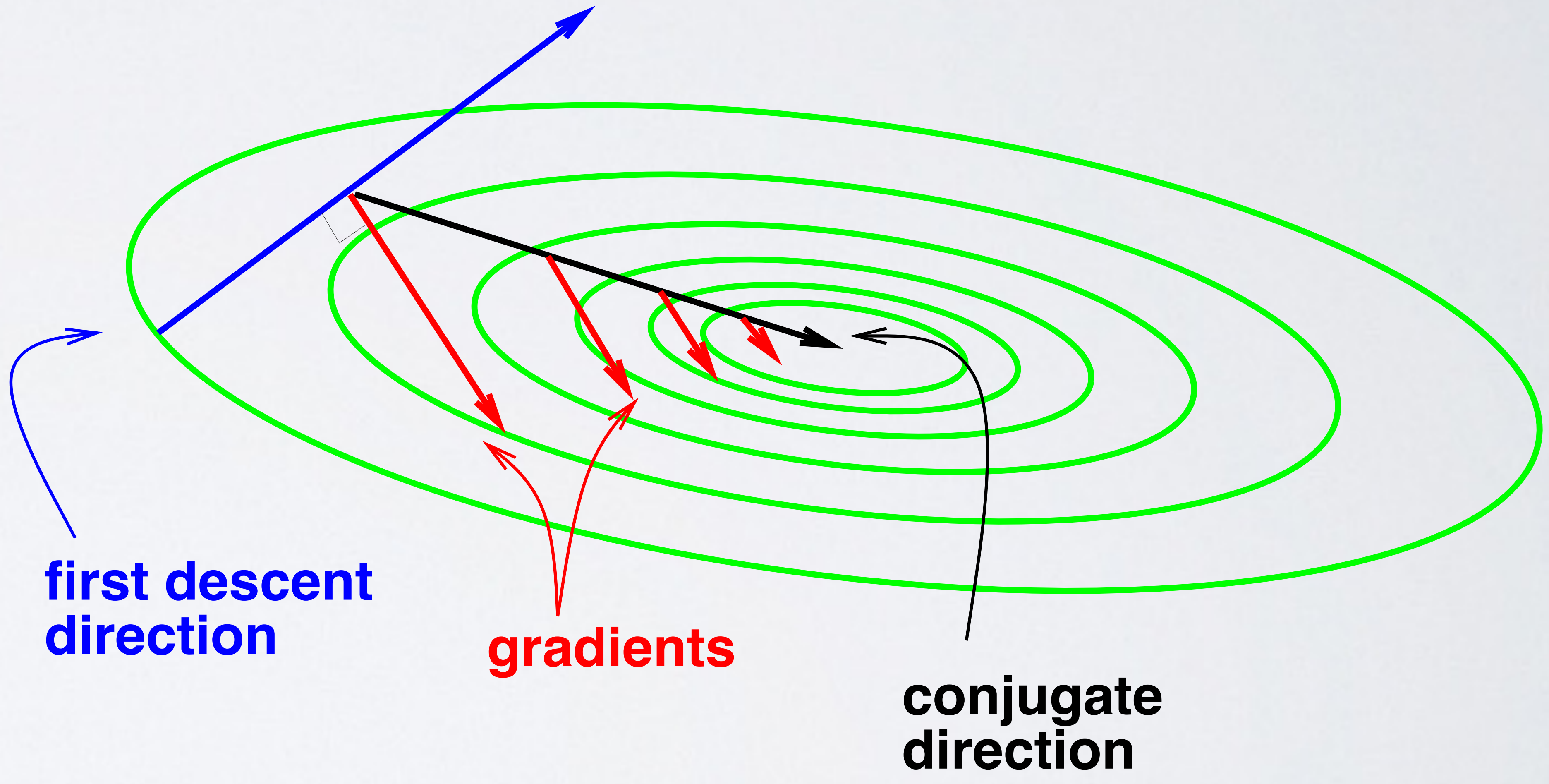
Require: Global learning rate η.
Require: Initial parameter $\boldsymbol{\theta}_0$
  for $t = 1$ to T %(T = total number of updates) **do**
    Compute Hessian inverse: $\boldsymbol{H}_t^{-1}$
    Compute gradien: $\boldsymbol{g}_t$ (via batch backpropagation)
    Compute update: $\Delta\boldsymbol{\theta}_t = \boldsymbol{H}_t^{-1}\boldsymbol{g}_t$
    Apply update: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \Delta\boldsymbol{\theta}_t$
  end for

NEWTON'S METHOD

- Problems with Newton's method:
  - Computation of $\boldsymbol{H}^{-1}$ is $O(n^3)$ (where $\boldsymbol{n}$ is the number of parameters)
  - Memory requirement is $O(n^2)$
- These considerations make Newton's method impractical for the vast majority of applications of deep learning
  - Our models are typically far too large.

CONJUGATE GRADIENT

- Can we recover some of the advantages of Newton's method without the extreme computational and memory requirements?
- Surprisingly, yes! One way is the conjugate gradient method.
- **Basic Idea:** Problem with gradient descent is the direction it picks, undoing progress of previous updates.
  - **Solution:** adjust the direction to be **conjugate** to the previous updates (doesn't undo progress).



CONJUGATE GRADIENT

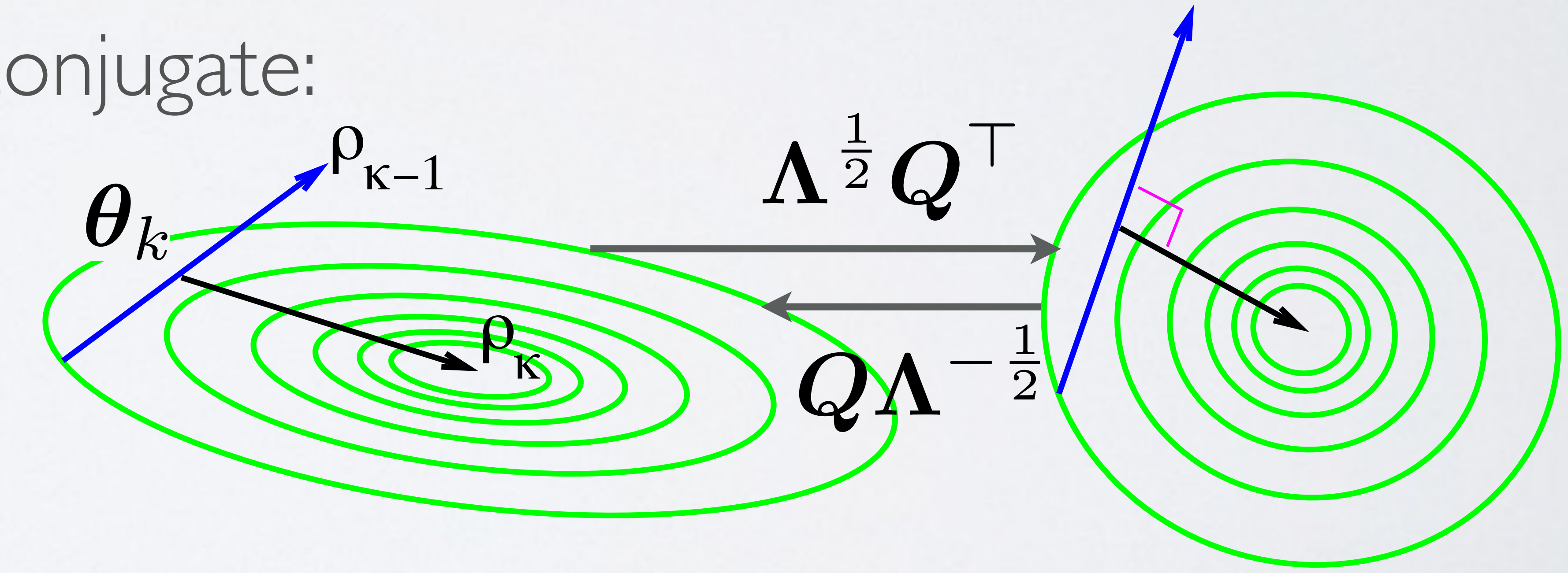
- The direction of the update is given by $\boldsymbol{\rho}_t = -\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t) + \beta_t \boldsymbol{\rho}_{t-1}$

- where $\beta_t = \dfrac{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)}{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})}$ (Fletcher - Reeves)

  or $\beta_t = \dfrac{(\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t) - \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1}))^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)}{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})}$ (Polak - Ribière)

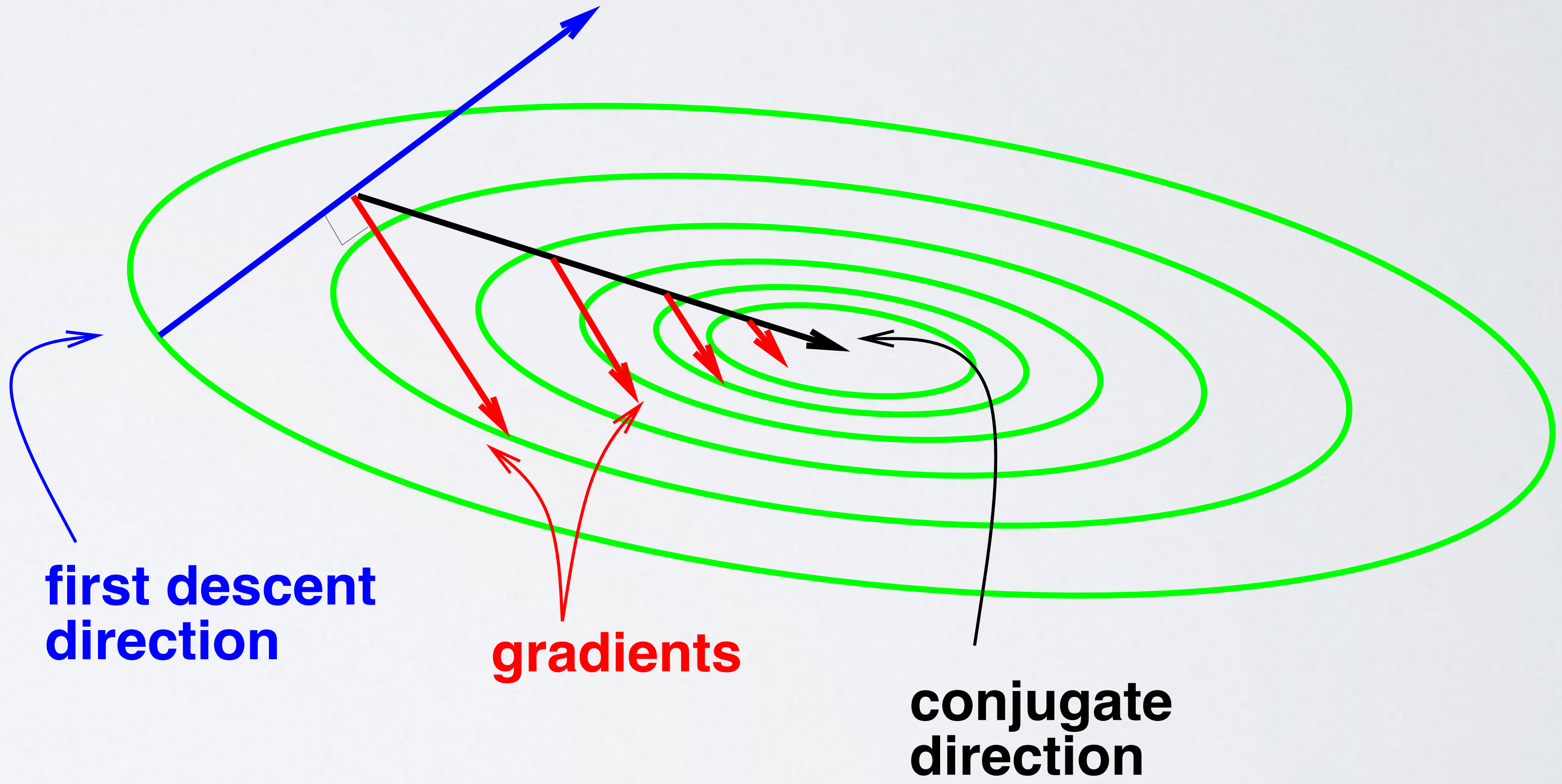
- β_t is designed to make $\boldsymbol{\rho}_t$ and $\boldsymbol{\rho}_{t-1}$ conjugate:

$$\boldsymbol{\rho}_t^\top H \boldsymbol{\rho}_{t-1} = 0$$



CONJUGATE GRADIENT

- Properties of Conjugate Gradient methods:
  - For a quadratic bowl, it's an $O(n)$ computation
  - No explicit use of the Hessian.
  - Uses line search.
  - Designed for batch learning.



CONJUGATE GRADIENT

- Conjugate gradient algorithm:

Algorithm 1 Conjugate gradient method

Require: Initial parameters $\boldsymbol{\theta}_0$

  Initialize $\boldsymbol{\rho}_0 = \mathbf{0}$

  while stopping criterion not met **do**

    Compute gradient: $\boldsymbol{g}_t = \nabla J(\boldsymbol{\theta}_t)$ (via batch backpropagation)

    Compute $\beta_t = \frac{(\boldsymbol{g}_t - \boldsymbol{g}_{t-1})^\top \boldsymbol{g}_t}{\boldsymbol{g}_{t-1}^\top \boldsymbol{g}_{t-1}}$ (Polak - Ribière)

    Compute search direction: $\boldsymbol{\rho}_t = -\boldsymbol{g}_t + \beta_t \boldsymbol{\rho}_{t-1}$

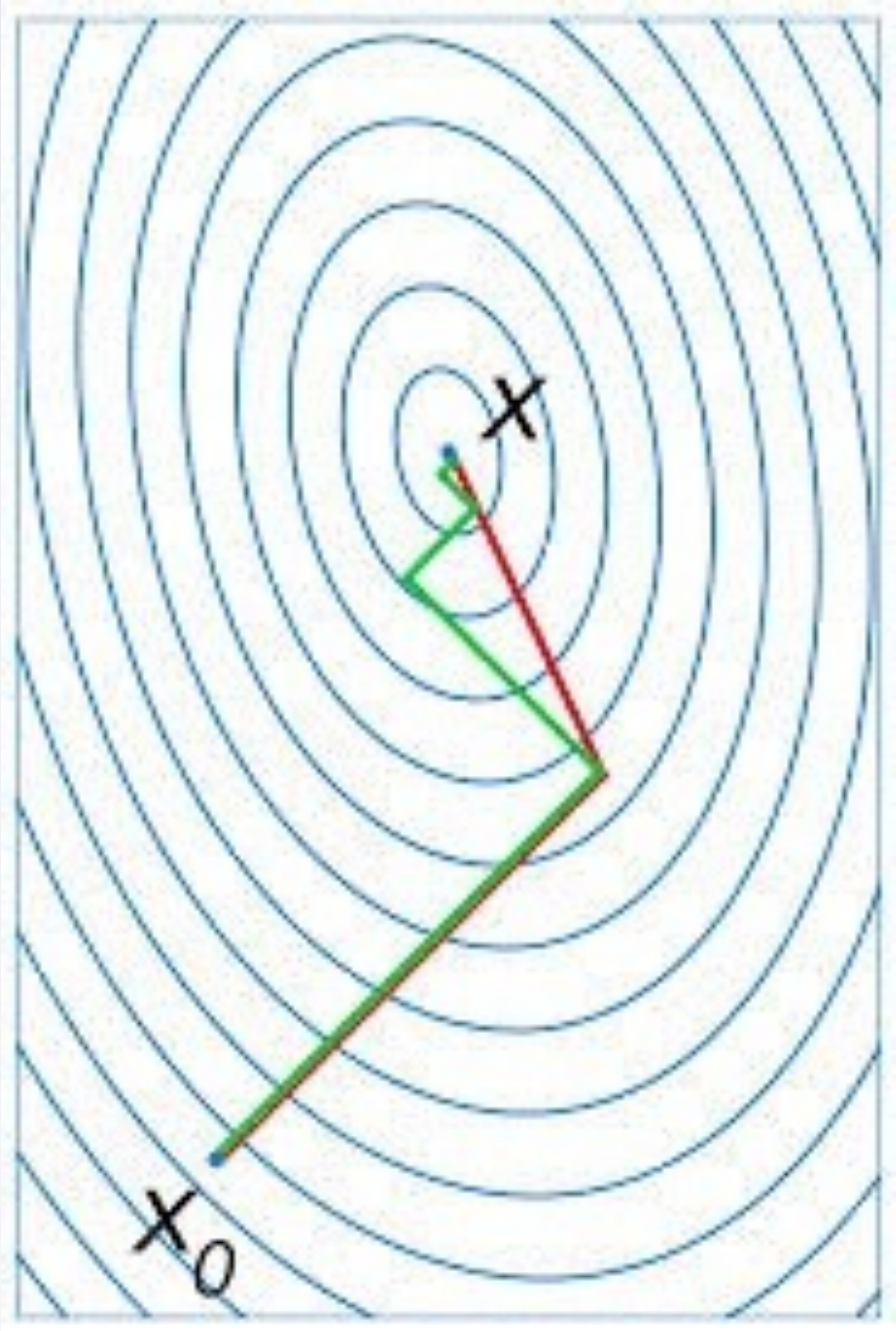
    Perform line search to find: $\eta^* = \operatorname{argmin}_\eta J(\boldsymbol{\theta}_t + \eta \boldsymbol{\rho}_t)$

    Apply update: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta^* \boldsymbol{\rho}_t$

  end while

CONJUGATE GRADIENT

- Comparing CG to Gradient Descent (Figure from wikipedia).
- Green: Gradient Descent
- Red: Conjugate Gradient
- For N-dimensional quadratic loss, CG finds the solution in N steps.



NONLINEAR CONJUGATE GRADIENT

- In general, since the loss is not quadratic, we will not necessarily reach the minimum with N steps (in N-dimensional parameter space).
- This isn't usually a problem since often N is huge and we don't want to take that many steps anyway.
- Does CG still work? Actually yes, reasonably well.
- But it is still useful to reset the conjugate directions once in a while to shift CG to be closer to gradient descent — this seems to work in practice.

BFGS - A QUASI-NEWTON METHOD

- **Basic idea:** Let's try to approximate the inverse Hessian through incremental low-rank updates.
- If the Hessian changes slowly (relative to our progress in parameter space), then our approximation will be good and we should be able to have close to behaviour of Newton's method.
- Most popular (successful) of these approaches is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method.

BFGS - A QUASI-NEWTON METHOD

- The BFGS algorithm:

Algorithm 1 BFGS method

Require: Initial parameters $\boldsymbol{\theta}_0$

Initialize inverse Hessian $\boldsymbol{M}_0 = \boldsymbol{I}$

while stopping criterion not met **do**

Compute gradient: $\boldsymbol{g}_t = \nabla J(\boldsymbol{\theta}_t)$ (via batch backpropagation)

Compute $\boldsymbol{\phi} = \boldsymbol{g}_t - \boldsymbol{g}_{t-1}$, $\boldsymbol{\Delta} = \boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}$

Approx $\boldsymbol{H}^{-1}$: $\boldsymbol{M}_t = \boldsymbol{M}_{t-1} + \left(1 + \frac{\boldsymbol{\phi}^\top \boldsymbol{M}_{t-1}\boldsymbol{\phi}}{\boldsymbol{\Delta}^\top \boldsymbol{\phi}}\right) \frac{\boldsymbol{\phi}^\top \boldsymbol{\phi}}{\boldsymbol{\Delta}^\top \boldsymbol{\phi}} - \left(\frac{\boldsymbol{\Delta}\boldsymbol{\phi}^\top \boldsymbol{M}_{t-1} + \boldsymbol{M}_{t-1}\boldsymbol{\phi}\boldsymbol{\Delta}^\top}{\boldsymbol{\Delta}^\top \boldsymbol{\phi}}\right)$

Compute search direction: $\boldsymbol{\rho}_t = \boldsymbol{M}_t \boldsymbol{g}_t$

Perform line search to find: $\eta^* = \operatorname{argmin}_\eta J(\boldsymbol{\theta}_t + \eta \boldsymbol{\rho}_t)$

Apply update: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta^* \boldsymbol{\rho}_t$

end while

BFGS - A QUASI-NEWTON METHOD

- Disadvantages of the BFGS method:
  - Computation still $O(n^2)$
  - Memory requirements also $O(n^2)$

- Question: Can we do better?
  - Answer: Yes! (but there may be a price)

L-BFGS - A LIMITED MEMORY VERSION

- **Basic Idea:** don't compute or store inverse Hessian directly. Use the history of gradients and updates as an implicit low-rank approximation.
  - Can represent an approximation to the Hessian with the latest set of updates and gradients:

$$\boldsymbol{\phi}_\tau = \boldsymbol{g}_\tau - \boldsymbol{g}_{\tau-1},\ \boldsymbol{\delta}_\tau = \boldsymbol{\theta}_\tau - \boldsymbol{\theta}_{\tau-1} \text{ for } \tau \in \{t, t-1, t-2, \ldots, t-k\}$$

  - This maintains a rank $\boldsymbol{k}$ approximation to the Hessian.
  - The algorithm is not overly complicated, but offers relatively little insight over BFGS and is excluded on these grounds.

HESSIAN FREE OPTIMIZATION

- **Basic Idea:**
  1. Approximate the loss function with a Gauss-Newton quadratic bowl.
  2. Use a few steps of CG to approximately minimize this approximate loss.
  3. Jump (some part of the distance) to the minimum of this approximate loss.
- Iterate steps 1-3 until converged.

SADDLE-POINT FEATURE