

# 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

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

- 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:

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

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

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

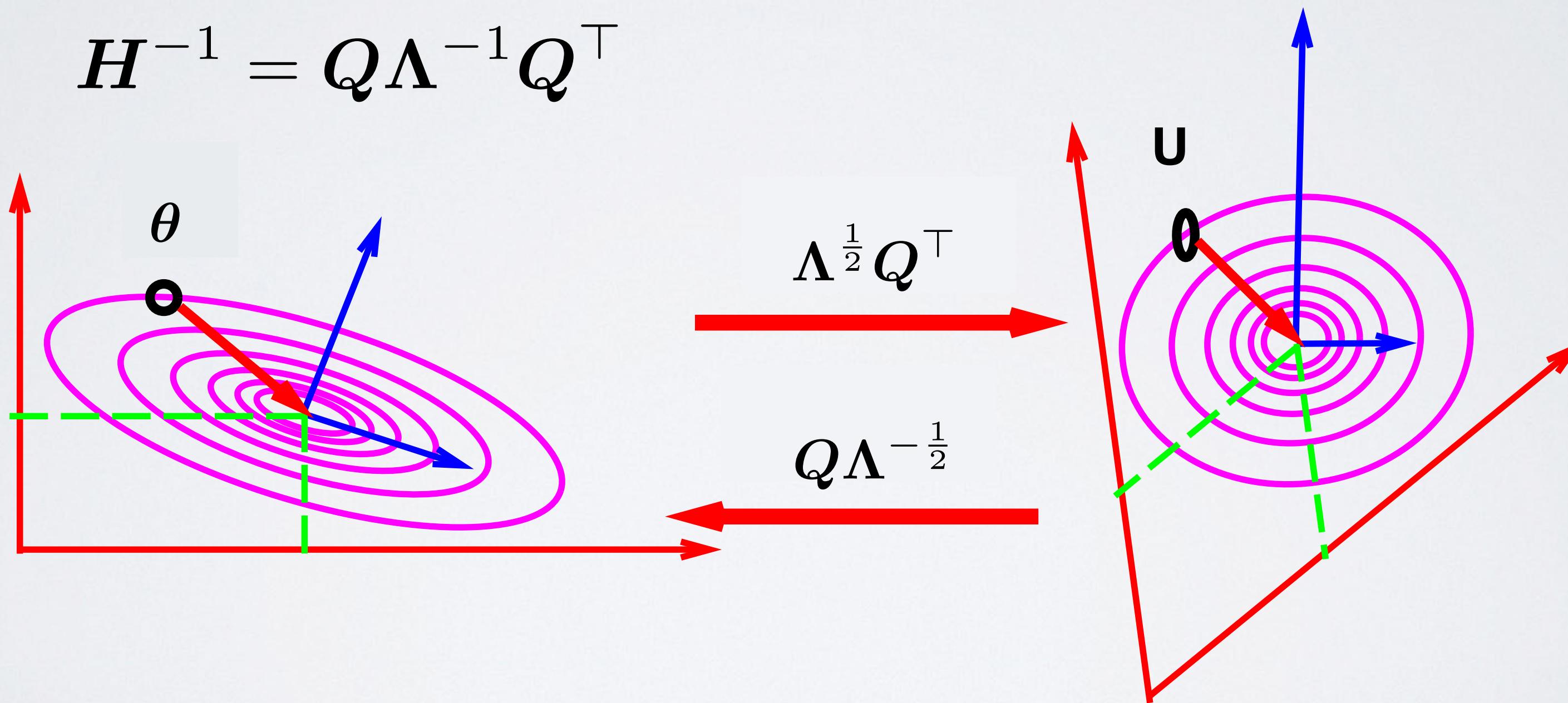
Newton's method update

# NEWTON'S METHOD

- How to think about Newton's method

$$H = Q\Lambda Q^T \quad (\text{eigendecomposition of } H)$$

$$H^{-1} = Q\Lambda^{-1}Q^T$$



# NEWTON'S METHOD

- Newton's method as an algorithm:

---

**Algorithm 1** Newton update at time t

---

**Require:** Global learning rate  $\eta$ .

**Require:** Initial parameter  $\theta_0$

**for**  $t = 1$  to  $T$  %( $T$  = total number of updates) **do**

    Compute Hessian inverse:  $H_t^{-1}$

    Compute gradient:  $g_t$  (via batch backpropagation)

    Compute update:  $\Delta\theta_t = H_t^{-1}g_t$

    Apply update:  $\theta_{t+1} = \theta_t + \Delta\theta_t$

**end for**

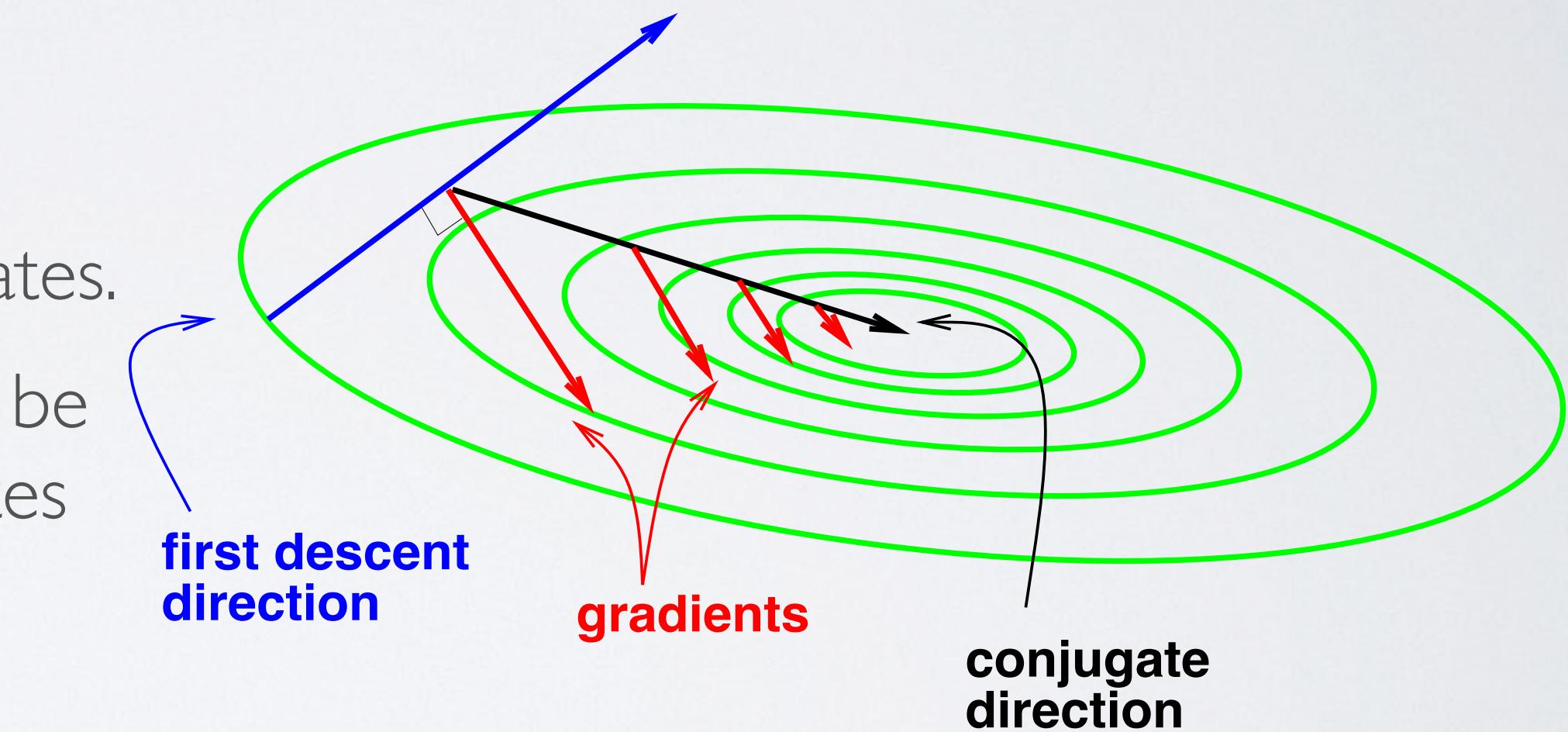
---

# NEWTON'S METHOD

- Problems with Newton's method:
  - ▶ Computation of  $\mathbf{H}^{-1}$  is  $O(n^3)$  (where  $n$  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).



# NONLINEAR CONJUGATE GRADIENT

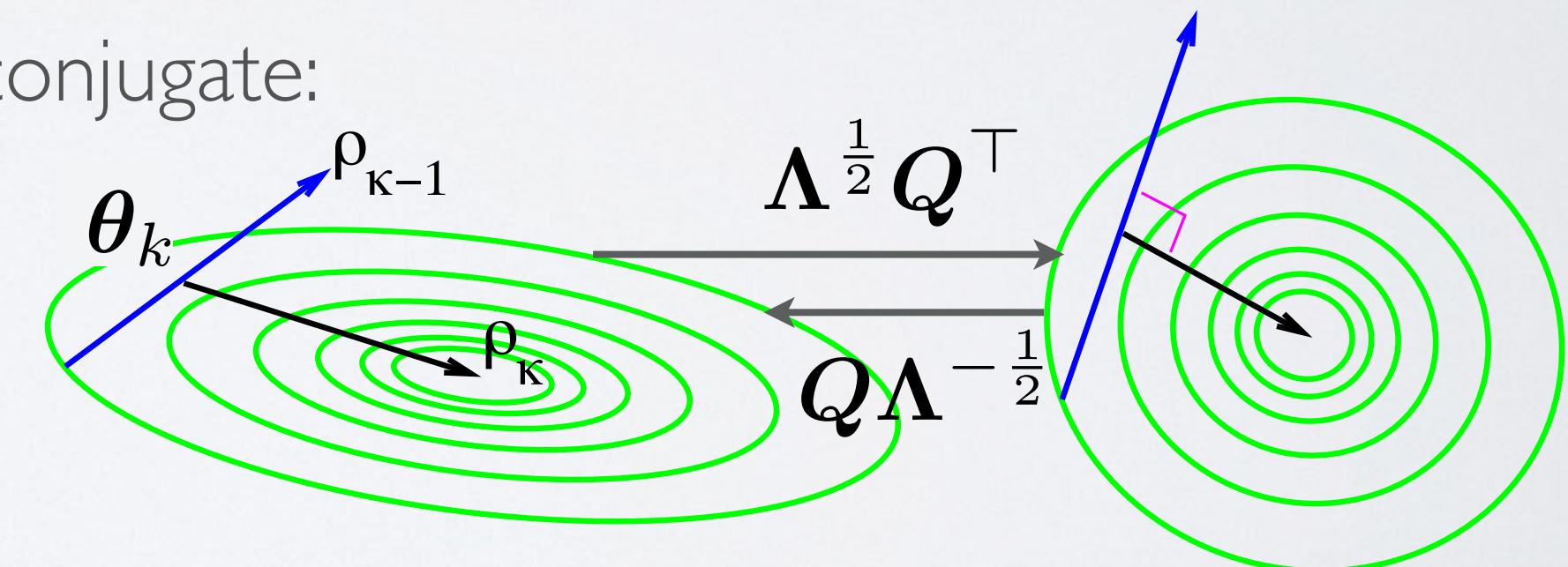
- The direction of the update is given by  $\rho_t = -\nabla_{\theta} J(\theta_t) + \beta_t \rho_{t-1}$

- where  $\beta_t = \frac{\nabla_{\theta} J(\theta_t)^T \nabla_{\theta} J(\theta_t)}{\nabla_{\theta} J(\theta_{t-1})^T \nabla_{\theta} J(\theta_{t-1})}$  (Fletcher - Reeves)

or  $\beta_t = \frac{(\nabla_{\theta} J(\theta_t) - \nabla_{\theta} J(\theta_{t-1}))^T \nabla_{\theta} J(\theta_t)}{\nabla_{\theta} J(\theta_{t-1})^T \nabla_{\theta} J(\theta_{t-1})}$  (Polak - Ribi re)

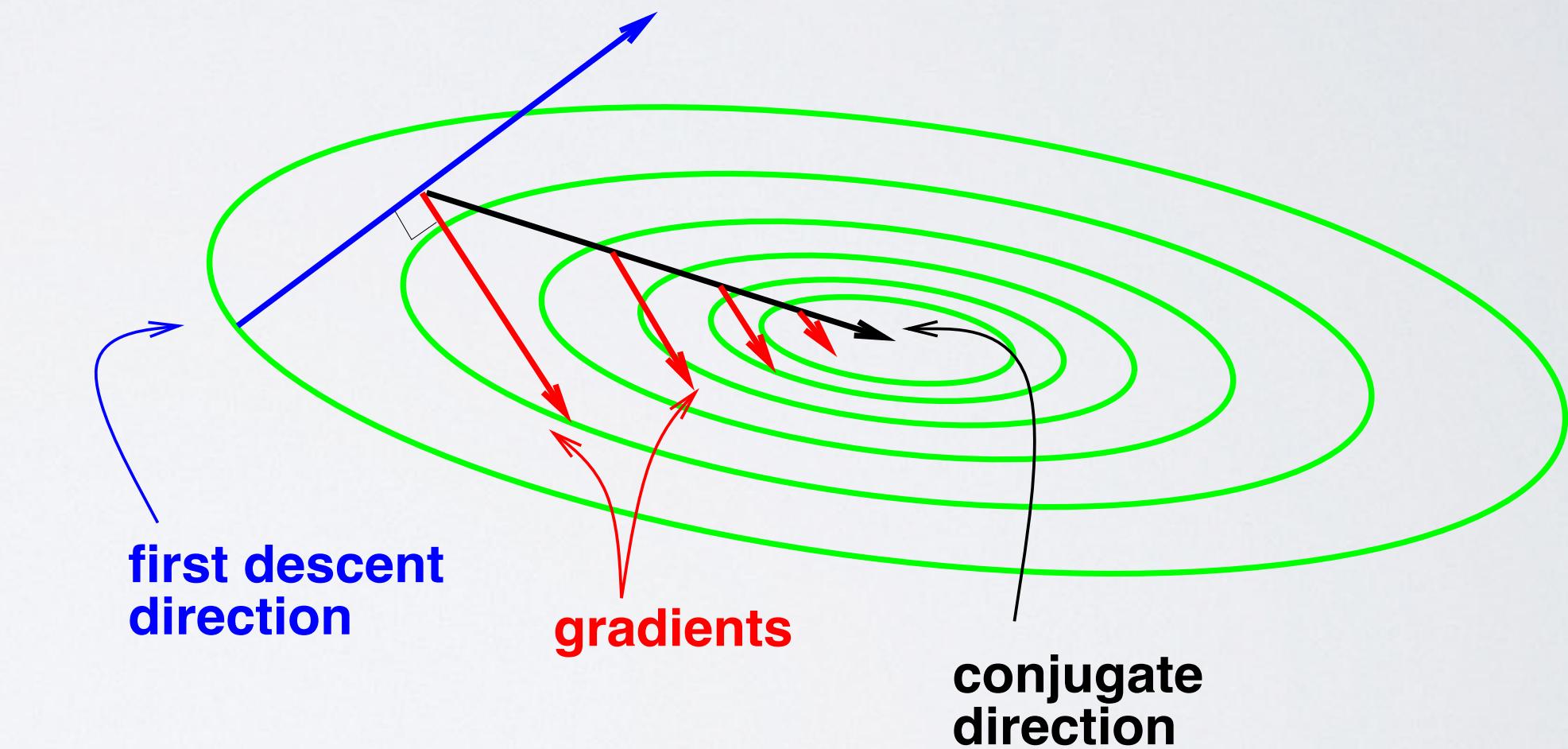
- $\beta_t$  is designed to make  $\rho_t$  and  $\rho_{t-1}$  conjugate:

$$\rho_t^T H \rho_{t-1} = 0$$



# NONLINEAR 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.



# NONLINEAR CONJUGATE GRADIENT

- Conjugate gradient algorithm:

---

## Algorithm 1 Conjugate gradient method

---

**Require:** Initial parameters  $\theta_0$

Initialize  $\rho_0 = 0$

**while** stopping criterion not met **do**

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

    Compute  $\beta_t = \frac{(\mathbf{g}_t - \mathbf{g}_{t-1})^\top \mathbf{g}_t}{\mathbf{g}_{t-1}^\top \mathbf{g}_{t-1}}$  (Polak - Ribi  re)

    Compute search direction:  $\rho_t = -\mathbf{g}_t + \beta_t \rho_{t-1}$

    Perform line search to find:  $\eta^* = \operatorname{argmin}_\eta J(\theta_t + \eta \rho_t)$

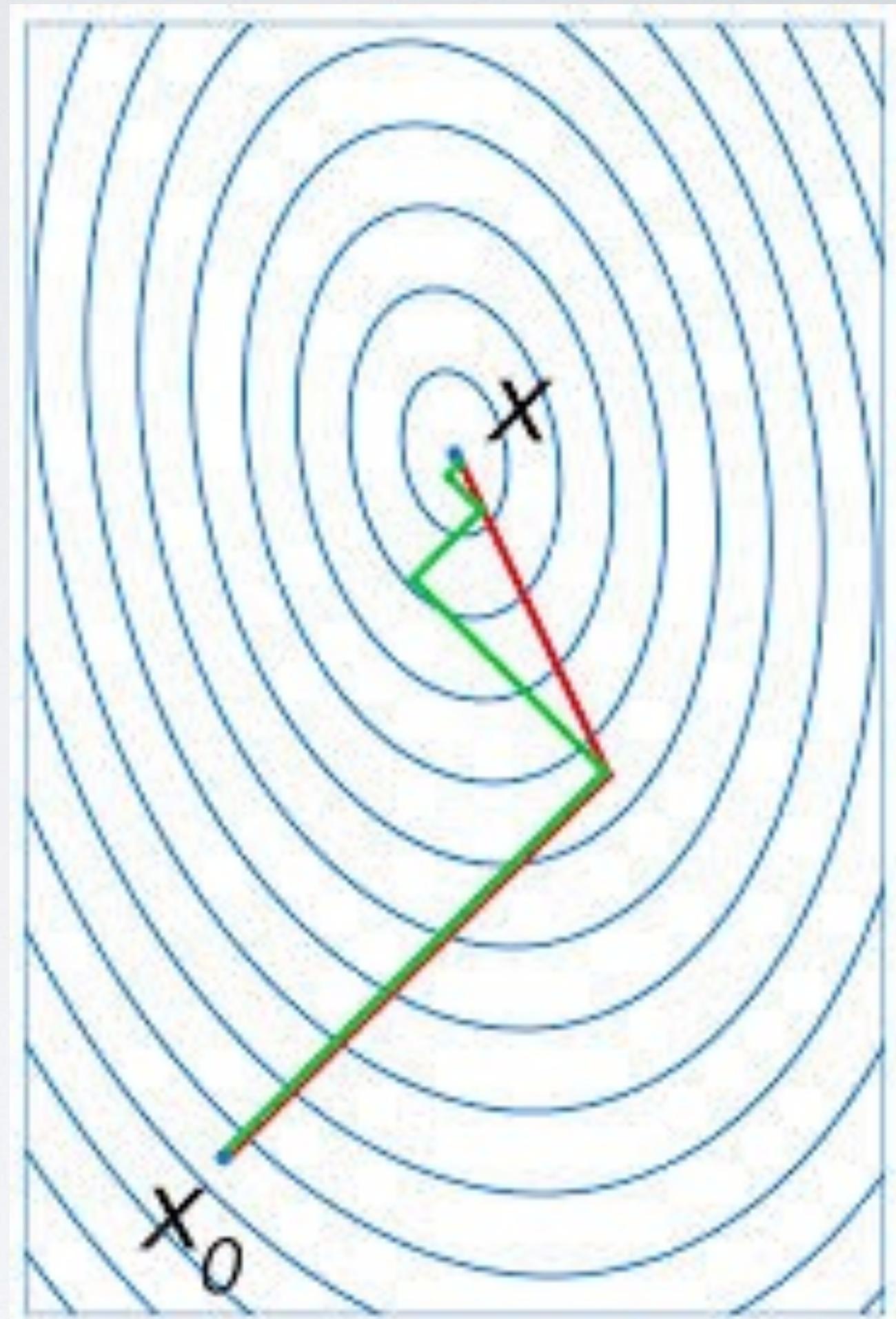
    Apply update:  $\theta_{t+1} = \theta_t + \eta^* \rho_t$

**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.



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

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

# BFGS - A QUASI-NEWTON METHOD

- The BFGS algorithm:

## Algorithm 1 BFGS method

**Require:** Initial parameters  $\theta_0$

Initialize inverse Hessian  $M_0 = I$

**while** stopping criterion not met **do**

    Compute gradient:  $g_t = \nabla J(\theta_t)$  (via batch backpropagation)

    Compute  $\phi = g_t - g_{t-1}$ ,  $\Delta = \theta_t - \theta_{t-1}$

    Approx  $H^{-1}$ :  $M_t = M_{t-1} + \left(1 + \frac{\phi^\top M_{t-1} \phi}{\Delta^\top \phi}\right) \frac{\phi^\top \phi}{\Delta^\top \phi} - \left(\frac{\Delta \phi^\top M_{t-1} + M_{t-1} \phi \Delta^\top}{\Delta^\top \phi}\right)$

    Compute search direction:  $\rho_t = M_t g_t$

    Perform line search to find:  $\eta^* = \operatorname{argmin}_\eta J(\theta_t + \eta \rho_t)$

    Apply update:  $\theta_{t+1} = \theta_t + \eta^* \rho_t$

**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

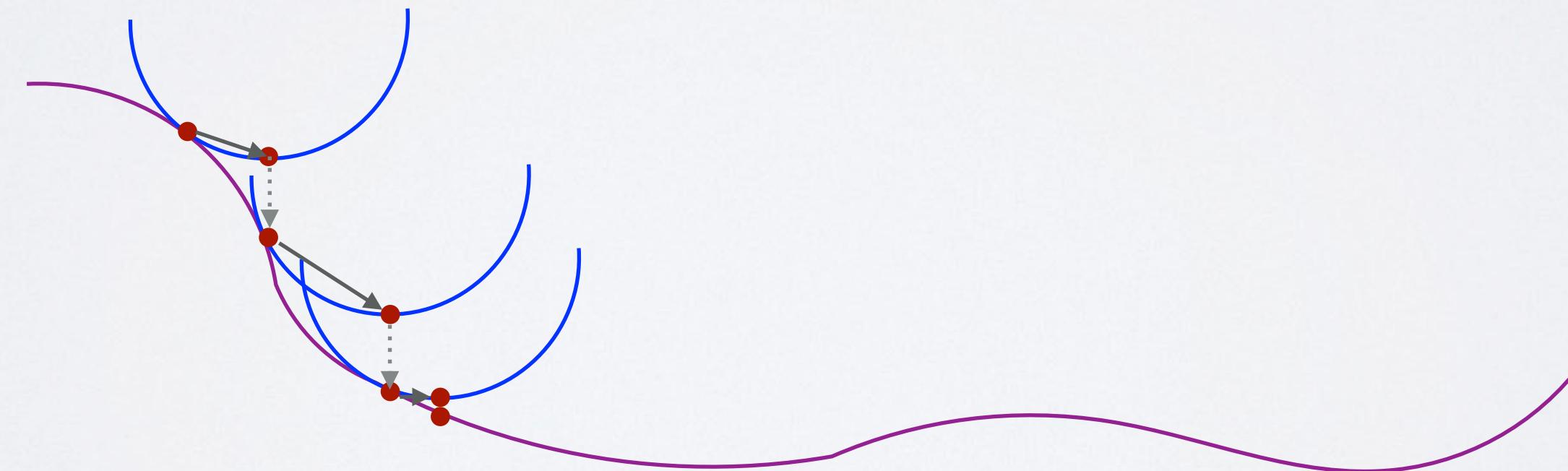
- **Basic Idea:** don't compute or store inverse Hessian directly. Use the history of gradients and updates as an implicit low-rank approximation.
  - ▶ Can represent an approximation to the Hessian with the latest set of updates and gradients:

$$\phi_\tau = \mathbf{g}_\tau - \mathbf{g}_{\tau-1}, \delta_\tau = \boldsymbol{\theta}_\tau - \boldsymbol{\theta}_{\tau-1} \text{ for } \tau \in \{t, t-1, t-2, \dots, t-k\}$$

- ▶ This maintains a rank  $k$  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

