CSC 561: Neural Networks and Deep Learning

Optimization (part 1)

Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

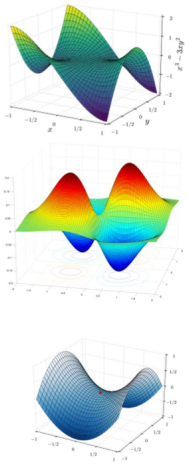
Spring 2024

THINK BIG WE DO℠

Credit: the following slides are taken from one of the "11-785 Introduction to Deep Learning - CMU" lectures

The Loss Surface

- **Popular hypothesis**:
  - In large networks, saddle points are far more common than local minima
    - Frequency of occurrence exponential in network size
  - Most local minima are equivalent
    - And close to global minimum
  - This is not true for small networks
- **Saddle point:** A point where
  - The slope is zero
  - The surface increases in some directions, but decreases in others
    - Some of the Eigenvalues of the Hessian are positive; others are negative
  - Gradient descent algorithms often get "stuck" in saddle points

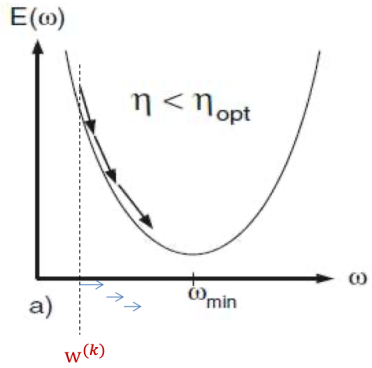


Convergence for quadratic surfaces

$$Minimize\ E = \frac{1}{2}aw^2 + bw + c$$

$$w^{(k+1)} = w^{(k)} - \eta \frac{dE\left(w^{(k)}\right)}{dw}$$

Gradient descent with fixed step size η to estimate *scalar* parameter w

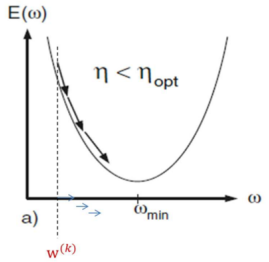


- Gradient descent to find the optimum of a quadratic, starting from $w^{(k)}$
- Assuming fixed step size η
- What is the optimal step size η to get there fastest?

Convergence for quadratic surfaces

$$E = \frac{1}{2}aw^2 + bw + c$$

$$w^{(k+1)} = w^{(k)} - \eta \frac{dE(w^{(k)})}{dw}$$



- Any quadratic objective can be written as

$$E(w) = E(w^{(k)}) + E'(w^{(k)})(w - w^{(k)}) + \frac{1}{2}E''(w^{(k)})(w - w^{(k)})^2$$

  - Taylor expansion

- Minimizing w.r.t w, we get (Newton's method)

$$w_{min} = w^{(k)} - E''(w^{(k)})^{-1}E'(w^{(k)})$$

- Note:

$$\frac{dE(w^{(k)})}{dw} = E'(w^{(k)})$$

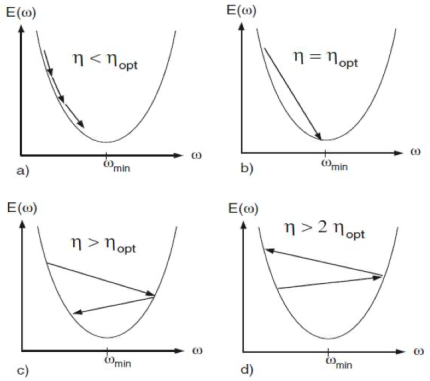
- Comparing to the gradient descent rule, we see that we can arrive at the optimum in a single step using the optimum step size

$$\eta_{opt} = E''(w^{(k)})^{-1} = \boldsymbol{a^{-1}}$$

With non-optimal step size

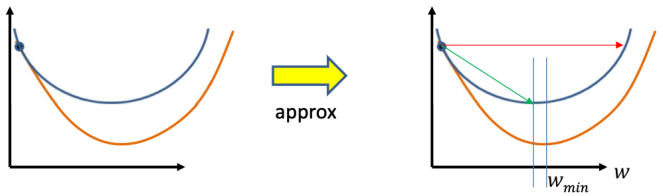
$$w^{(k+1)} = w^{(k)} - \eta \frac{dE(w^{(k)})}{dw}$$

Gradient descent with fixed step size η to estimate scalar parameter w



- For $\eta < \eta_{opt}$ the algorithm will converge monotonically
- For $2\eta_{opt} > \eta > \eta_{opt}$ we have oscillating convergence
- For $\eta > 2\eta_{opt}$ we get divergence

For generic differentiable convex objectives



- Any differentiable convex objective $E(w)$ can be approximated as

$$E \approx E(w^{(k)}) + (w - w^{(k)})\frac{dE(w^{(k)})}{dw} + \frac{1}{2}(w - w^{(k)})^2\frac{d^2E(w^{(k)})}{dw^2} + \cdots$$

  - Taylor expansion

- Using the same logic as before, we get (Newton's method)

$$\eta_{opt} = \left(\frac{d^2E(w^{(k)})}{dw^2}\right)^{-1}$$

- We can get divergence if $\eta \geq 2\eta_{opt}$

For functions of *multivariate* inputs

$$E = g(\mathbf{w}), \mathbf{w} \text{ is a vector } \mathbf{w} = [w_1, w_2, \ldots, w_N]$$

- Consider a simple quadratic convex (paraboloid) function

$$E = \frac{1}{2}\mathbf{w}^T\mathbf{A}\mathbf{w} + \mathbf{w}^T\mathbf{b} + c$$

  - Since $E^T = E$ (E is scalar), $\mathbf{A}$ can always be made symmetric
    - For strictly **convex** E, $\mathbf{A}$ is always positive definite, and has positive eigenvalues

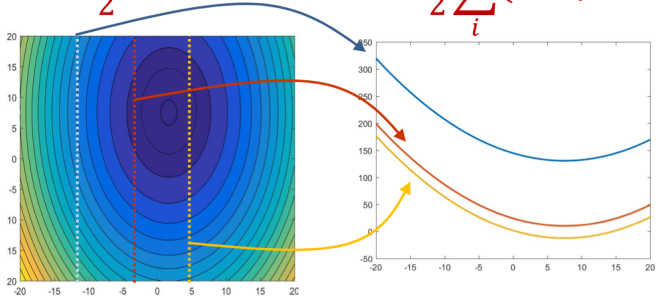
- When $\mathbf{A}$ is diagonal:

$$E = \frac{1}{2}\sum_i (a_{ii}w_i^2 + b_i w_i) + c$$

  - The w_is are *uncoupled*
  - For paraboloid (*convex*) E, the a_{ii} values are all positive
  - Just a sum of N independent quadratic functions

Multivariate Quadratic with Diagonal A

$$E = \frac{1}{2}\mathbf{w}^T\mathbf{A}\mathbf{w} + \mathbf{w}^T\mathbf{b} + c = \frac{1}{2}\sum_i \left(a_{ii}w_i^2 + b_i w_i\right) + c$$

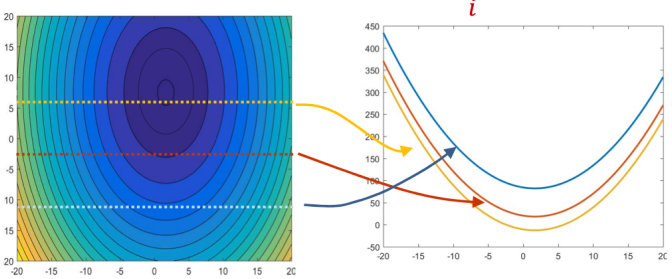


- Equal-value contours will be parallel to the axes
  - All "slices" parallel to an axis are shifted versions of one another

$$E = \frac{1}{2}a_{ii}w_i^2 + b_i w_i + c + C(\neg w_i)$$

Multivariate Quadratic with Diagonal A

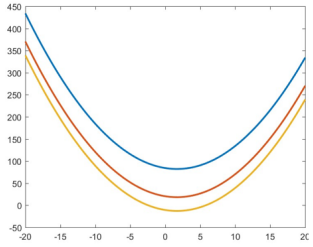
$$E = \frac{1}{2}\mathbf{w}^T\mathbf{A}\mathbf{w} + \mathbf{w}^T\mathbf{b} + c = \frac{1}{2}\sum_i \left(a_{ii}w_i^2 + b_i w_i\right) + c$$



- Equal-value contours will be parallel to the axis
  - All "slices" parallel to an axis are shifted versions of one another

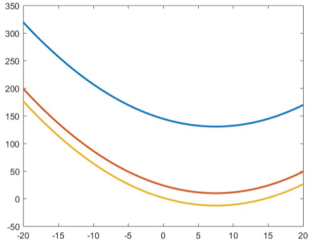
$$E = \frac{1}{2}a_{ii}w_i^2 + b_i w_i + c + C(\neg w_i)$$

"Descents" are uncoupled



$$E = \frac{1}{2}a_{11}w_1^2 + b_1 w_1 + c + C(\neg w_1)$$

$$\eta_{1,opt} = a_{11}^{-1}$$

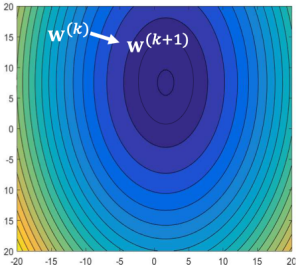


$$E = \frac{1}{2}a_{22}w_2^2 + b_2 w_2 + c + C(\neg w_2)$$

$$\eta_{2,opt} = a_{22}^{-1}$$

- The optimum of each coordinate is not affected by the other coordinates
  - I.e. we could optimize each coordinate independently
- **Note: Optimal learning rate is different for the different coordinates**

Vector update rule

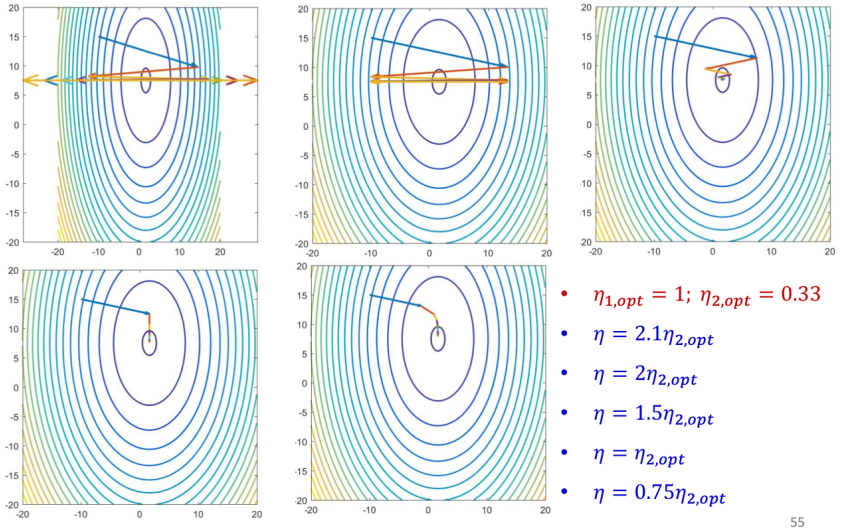


$$\mathbf{w}^{(k+1)} \leftarrow \mathbf{w}^{(k)} - \eta\nabla_{\mathbf{w}}E^{\top}$$

$$w_i^{(k+1)} = w_i^{(k)} - \eta\frac{\partial E\left(w_i^{(k)}\right)}{d\partial \mathrm{w}}$$

- Conventional vector update rules for gradient descent: update entire vector against direction of gradient
  - Note : Gradient is perpendicular to equal value contour
  - The same learning rate is applied to all components

Dependence on learning rate



- $\eta_{1,opt} = 1;\ \eta_{2,opt} = 0.33$
- $\eta = 2.1\eta_{2,opt}$
- $\eta = 2\eta_{2,opt}$
- $\eta = 1.5\eta_{2,opt}$
- $\eta = \eta_{2,opt}$
- $\eta = 0.75\eta_{2,opt}$

Problem with vector update rule

$$\mathbf{w}^{(k+1)} \leftarrow \mathbf{w}^{(k)} - \eta \nabla_{\mathbf{w}} E^T$$

$$w_i^{(k+1)} = w_i^{(k)} - \eta \frac{\partial E\left(w_i^{(k)}\right)}{\partial \mathrm{w}}$$

$$\eta_{i,opt} = \left(\frac{\partial^2 E\left(w_i^{(k)}\right)}{\partial w_i^2}\right)^{-1} = a_{ii}^{-1}$$

- The learning rate must be lower than twice the *smallest* optimal learning rate for any component

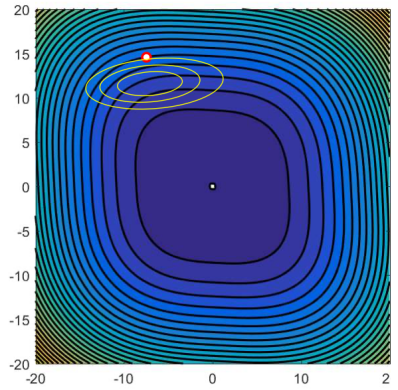
$$\eta < 2 \min_i \eta_{i,opt}$$

  - Otherwise the learning will diverge
- This, however, makes the learning very slow
  - And will oscillate in all directions where $\eta_{i,opt} \leq \eta < 2\eta_{i,opt}$

Convergence

- Convergence behaviors become increasingly unpredictable as dimensions increase
- For the fastest convergence, ideally, the learning rate η must be close to both, the largest $\eta_{i,opt}$ and the smallest $\eta_{i,opt}$
  - To ensure convergence in every direction
  - Generally infeasible
- Convergence is particularly slow if $\frac{\max_i \eta_{i,opt}}{\min_i \eta_{i,opt}}$ is large
  - The "condition" number is small

Minimization by Newton's method $(\eta = 1)$



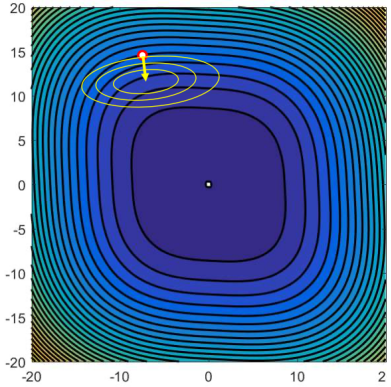
Fit a quadratic at each point and find the minimum of that quadratic

- Iterated localized optimization with quadratic approximations

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta H_E\left(\mathbf{w}^{(k)}\right)^{-1} \nabla_{\mathbf{w}} E\left(\mathbf{w}^{(k)}\right)^T$$

  - $\eta = 1$

Minimization by Newton's method ($\eta = 1$)

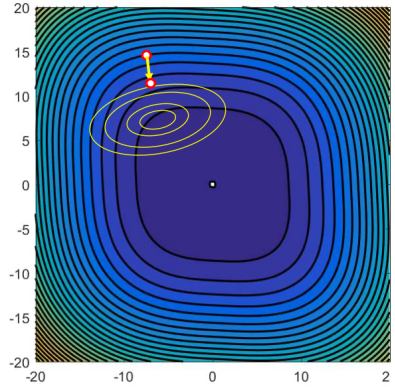


- Iterated localized optimization with quadratic approximations

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta H_E\left(\boldsymbol{w}^{(k)}\right)^{-1} \nabla_{\mathbf{w}} E\left(\mathbf{w}^{(k)}\right)^T$$

– $\eta = 1$

Minimization by Newton's method ($\eta = 1$)

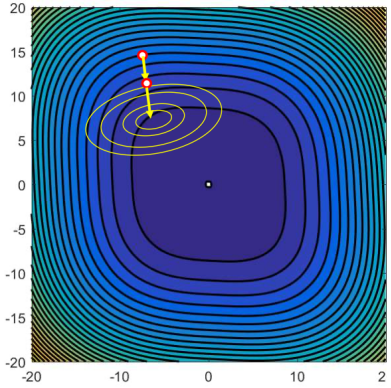


- Iterated localized optimization with quadratic approximations

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta H_E\left(\boldsymbol{w}^{(k)}\right)^{-1} \nabla_{\mathbf{w}} E\left(\mathbf{w}^{(k)}\right)^T$$

– $\eta = 1$

Minimization by Newton's method ($\eta = 1$)

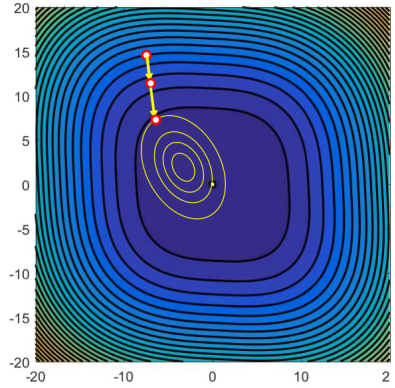


- Iterated localized optimization with quadratic approximations

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta H_E\left(\boldsymbol{w}^{(k)}\right)^{-1} \nabla_{\mathbf{w}} E\left(\mathbf{w}^{(k)}\right)^T$$

– $\eta = 1$

Minimization by Newton's method

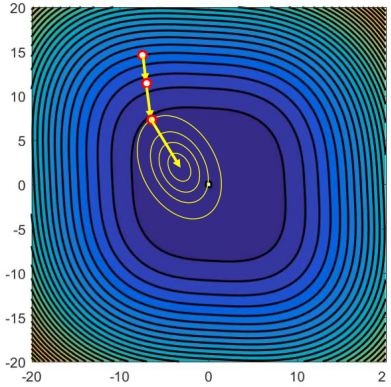


- Iterated localized optimization with quadratic approximations

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta H_E\left(\boldsymbol{w}^{(k)}\right)^{-1} \nabla_{\mathbf{w}} E\left(\mathbf{w}^{(k)}\right)^T$$

– $\eta = 1$

Minimization by Newton's method

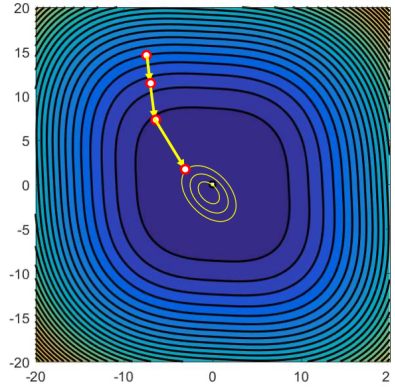


- Iterated localized optimization with quadratic approximations

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta H_E\left(\boldsymbol{w}^{(k)}\right)^{-1} \nabla_{\mathbf{w}} E\left(\mathbf{w}^{(k)}\right)^T$$

– $\eta = 1$

Minimization by Newton's method

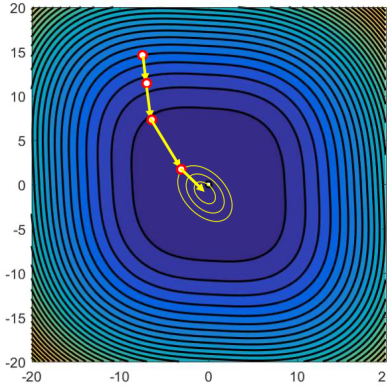


- Iterated localized optimization with quadratic approximations

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta H_E\left(\boldsymbol{w}^{(k)}\right)^{-1} \nabla_{\mathbf{w}} E\left(\mathbf{w}^{(k)}\right)^T$$

– $\eta = 1$

Minimization by Newton's method

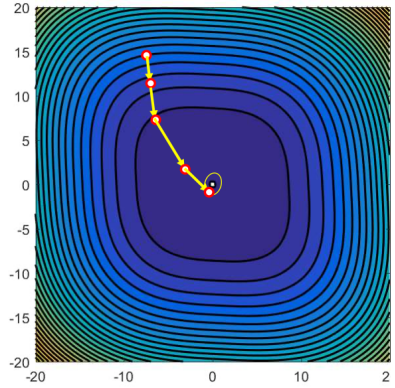


- Iterated localized optimization with quadratic approximations

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta H_E\left(\boldsymbol{w}^{(k)}\right)^{-1} \nabla_{\mathbf{w}} E\left(\mathbf{w}^{(k)}\right)^T$$

– $\eta = 1$

Minimization by Newton's method

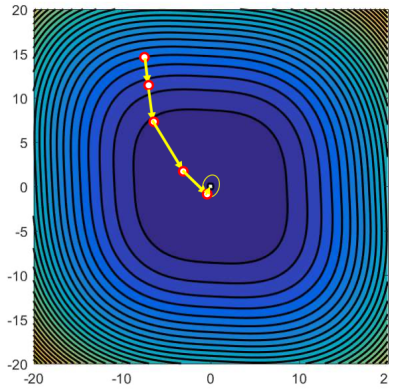


- Iterated localized optimization with quadratic approximations

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta H_E\left(\boldsymbol{w}^{(k)}\right)^{-1} \nabla_{\mathbf{w}} E\left(\mathbf{w}^{(k)}\right)^T$$

– $\eta = 1$

Minimization by Newton's method

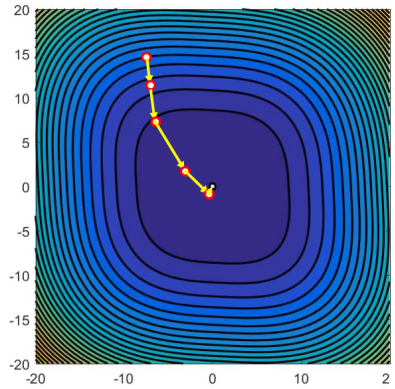


- Iterated localized optimization with quadratic approximations

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta H_E\left(\boldsymbol{w}^{(k)}\right)^{-1} \nabla_{\mathbf{w}} E\left(\mathbf{w}^{(k)}\right)^T$$

– $\eta = 1$

Minimization by Newton's method



- Iterated localized optimization with quadratic approximations

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta H_E\left(\boldsymbol{w}^{(k)}\right)^{-1} \nabla_{\mathbf{w}} E\left(\mathbf{w}^{(k)}\right)^T$$

– $\eta = 1$

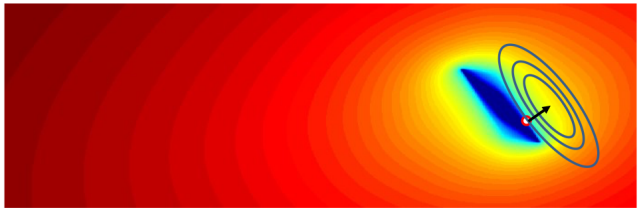
Issues: 1. The Hessian

- Normalized update rule

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta H_E\left(\boldsymbol{w}^{(k)}\right)^{-1} \nabla_{\mathbf{w}} E\left(\mathbf{w}^{(k)}\right)^T$$

- For complex models such as neural networks, with a very large number of parameters, the Hessian $H_E\left(\boldsymbol{w}^{(k)}\right)$ is extremely difficult to compute
  - For a network with only 100,000 parameters, the Hessian will have 10^{10} cross-derivative terms
  - And its even harder to invert, since it will be enormous

Issues: 1. The Hessian

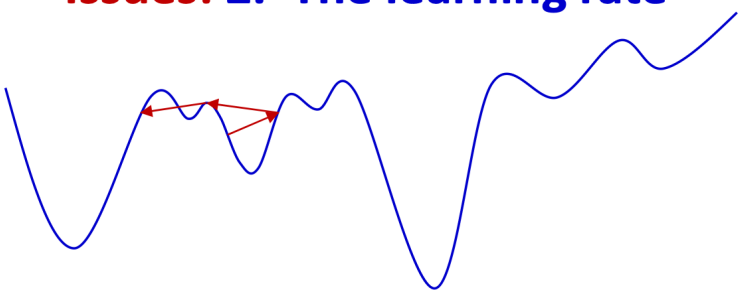


- For non-convex functions, the Hessian may not be positive semi-definite, in which case the algorithm can *diverge*
  - Goes away from, rather than towards the minimum
  - Now requires additional checks to avoid movement in directions corresponding to –ve Eigenvalues of the Hessian

Issues: 1 – contd.

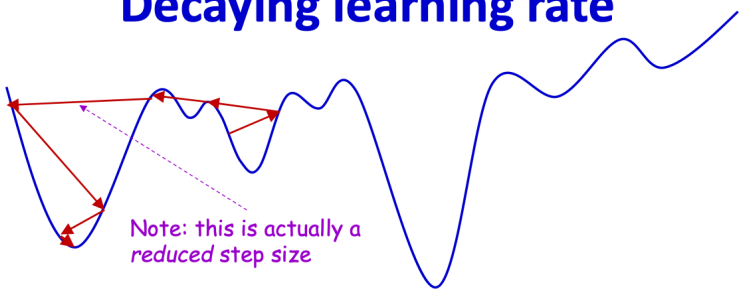
- A great many approaches have been proposed in the literature to *approximate* the Hessian in a number of ways and improve its positive definiteness
  - Boyden-Fletcher-Goldfarb-Shanno (BFGS)
    - And "low-memory" BFGS (L-BFGS)
    - Estimate Hessian from finite differences
  - Levenberg-Marquardt
    - Estimate Hessian from Jacobians
    - Diagonal load it to ensure positive definiteness
  - Other "Quasi-newton" methods
- Hessian estimates may even be *local* to a set of variables
- Not particularly popular anymore for large neural networks..

Issues: 2. The learning rate



- For complex models such as neural networks the loss function is often not convex
  - Having $\eta > 2\eta_{opt}$ can actually help escape local optima
- However *always* having $\eta > 2\eta_{opt}$ will ensure that you never ever actually find a solution

Decaying learning rate

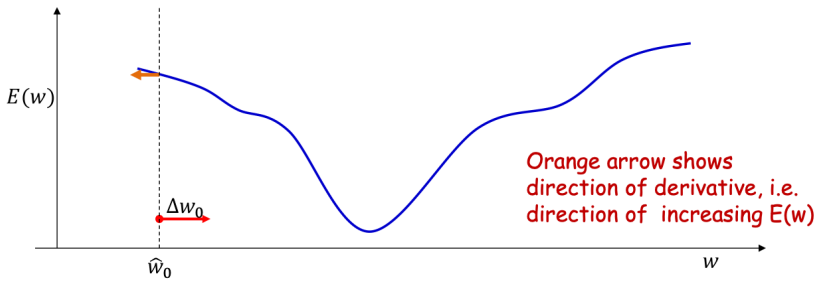


- Start with a large learning rate
  - Greater than 2 (assuming Hessian normalization)
  - Gradually reduce it with iterations

RProp

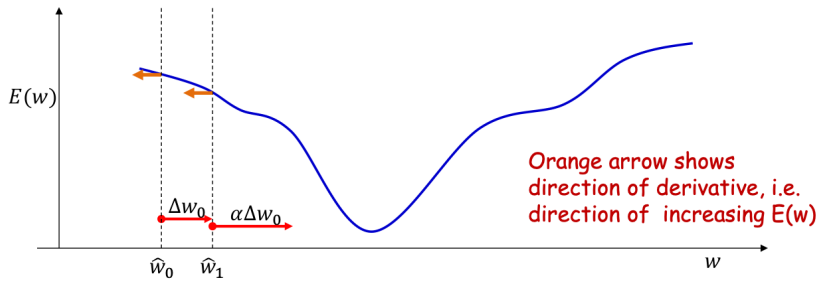
- *Resilient* propagation
- Simple algorithm, to be followed *independently* for each component
  - I.e. steps in different directions are not coupled
- At each time
  - If the derivative at the current location recommends continuing in the same direction as before (i.e. has not changed sign from earlier):
    - *increase* the step, and continue in the same direction
  - If the derivative has changed sign (i.e. we've overshot a minimum)
    - *reduce* the step and reverse direction

Rprop



- Select an initial value $\hat{w}$ and compute the derivative
  - Take an initial step Δw against the derivative
    - In the direction that reduces the function
      - $\Delta w = sign\left(\frac{dE(\hat{w})}{dw}\right)\Delta w$
      - $\hat{w} = \hat{w} - \Delta w$

Rprop

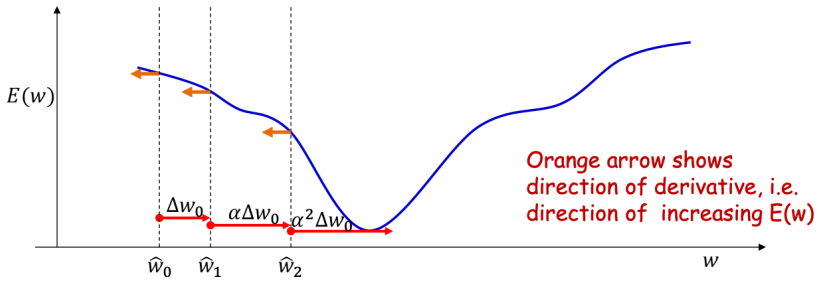


- Compute the derivative in the new location
  - If the derivative has not changed sign from the previous location, increase the step size and take a longer step

$\alpha > 1$

- $\Delta w = \alpha\Delta w$
- $\hat{w} = \hat{w} - \Delta w$

Rprop

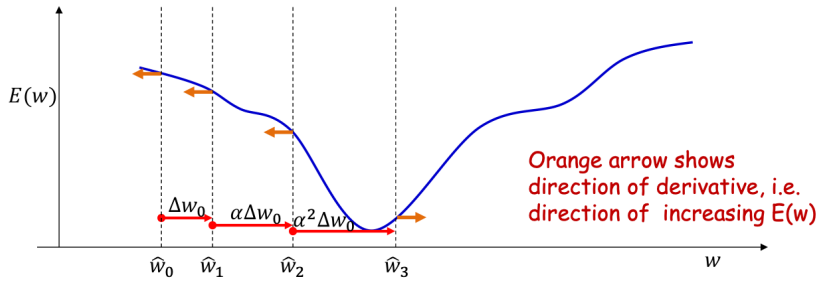


- Compute the derivative in the new location
  - If the derivative has not changed sign from the previous location, increase the step size and take a step

$\alpha > 1$

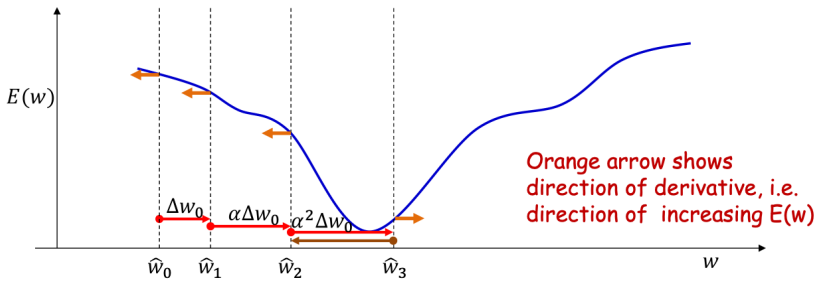
- $\Delta w = \alpha\Delta w$
- $\hat{w} = \hat{w} - \Delta w$

Rprop



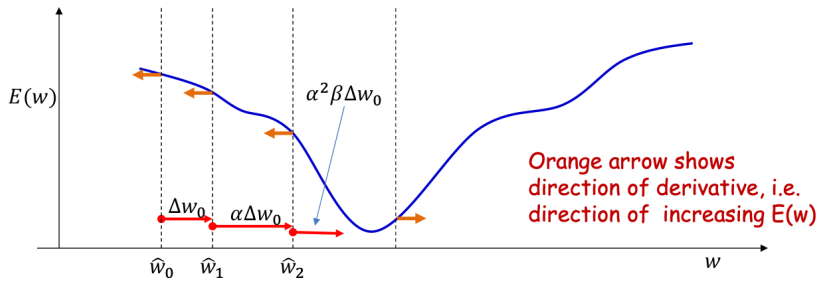
- Compute the derivative in the new location
  - If the derivative has changed sign

Rprop



- Compute the derivative in the new location
  - If the derivative has changed sign
  - Return to the previous location
    - $\hat{w} = \hat{w} + \Delta w$

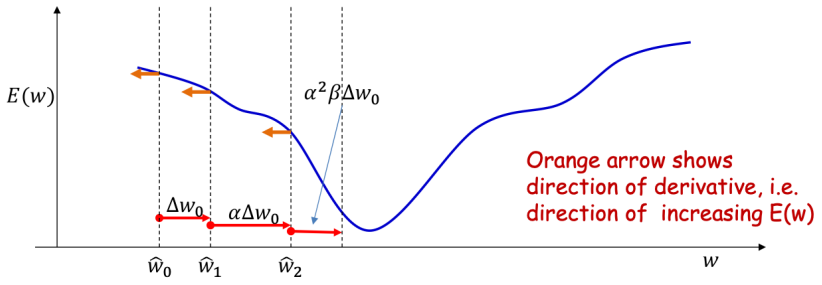
Rprop



- Compute the derivative in the new location
  - If the derivative has changed sign
  - Return to the previous location
    - $\hat{w} = \hat{w} + \Delta w$
  - Shrink the step
    - $\Delta w = \beta\Delta w$

$\beta < 1$

Rprop



- Compute the derivative in the new location
  - If the derivative has changed sign
  - Return to the previous location
    - $\hat{w} = \hat{w} + \Delta w$
  - Shrink the step
    - $\Delta w = \beta\Delta w$
  - Take the smaller step forward
    - $\hat{w} = \hat{w} - \Delta w$

$\beta < 1$

RProp

- A remarkably simple first-order algorithm, that is frequently much more efficient than gradient descent.
  - And can even be competitive against some of the more advanced second-order methods
- Only makes minimal assumptions about the loss function
  - No convexity assumption