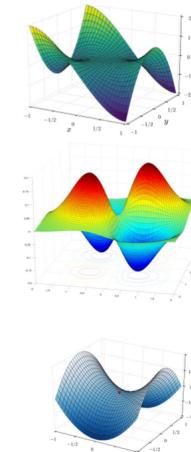


# Neural Networks: Optimization Part 1

Intro to Deep Learning, Spring 2024

1



36

## 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

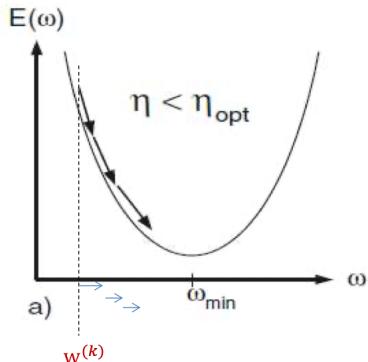
- The slope is zero
- The surface increases in some directions, but decreases in others
  - Some of the Eigenvalues of the Hessian are positive; others are negative
- Gradient descent algorithms often get “stuck” in saddle points

## Convergence for quadratic surfaces

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

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

Gradient descent with fixed step size  $\eta$  to estimate **scalar** parameter  $w$



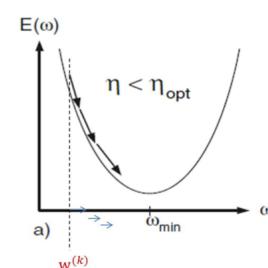
- Gradient descent to find the optimum of a quadratic, starting from  $w^{(k)}$
- Assuming fixed step size  $\eta$
- What is the optimal step size  $\eta$  to get there fastest?

44

## 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

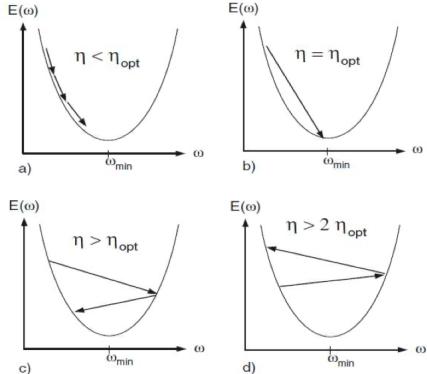
- Minimizing w.r.t  $w$ , we get (Newton's method)
 
$$w_{min} = w^{(k)} - E''(w^{(k)})^{-1} E'(w^{(k)})$$
- Note:
 
$$\frac{dE(w^{(k)})}{dw} = E'(w^{(k)})$$
- Comparing to the gradient descent rule, we see that we can arrive at the optimum in a single step using the optimum step size
 
$$\eta_{opt} = E''(w^{(k)})^{-1} = a^{-1}$$

45

## With non-optimal step size

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

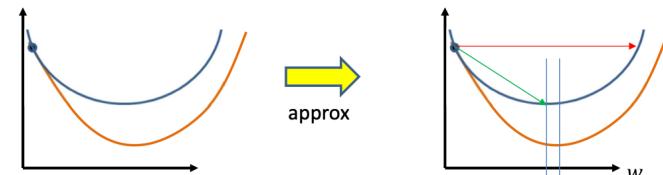
Gradient descent with fixed step size  $\eta$  to estimate scalar parameter  $w$



- 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

46

## For generic differentiable convex objectives



- Any differentiable convex objective  $E(w)$  can be approximated as
 
$$E \approx E(w^{(k)}) + (w - w^{(k)}) \frac{dE(w^{(k)})}{dw} + \frac{1}{2} (w - w^{(k)})^2 \frac{d^2E(w^{(k)})}{dw^2} + \dots$$
  - 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  $\eta \geq 2\eta_{opt}$

47

## For functions of *multivariate* inputs

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

- 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:

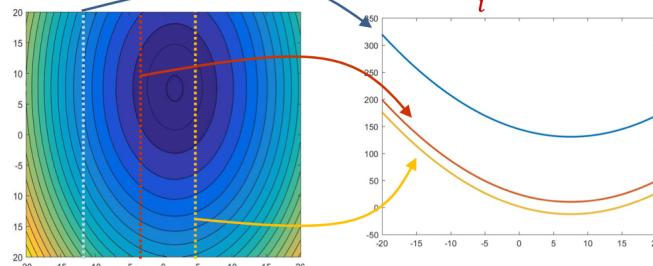
$$E = \frac{1}{2} \sum_i (a_{ii} w_i^2 + b_i w_i) + c$$

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

48

## Multivariate Quadratic with Diagonal $\mathbf{A}$

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



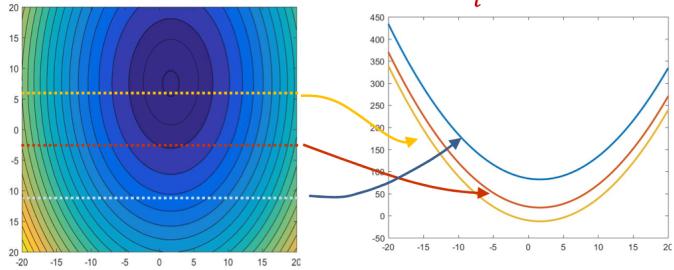
- Equal-value contours will be parallel to the axes
  - All "slices" parallel to an axis are shifted versions of one another

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

50

## Multivariate Quadratic with Diagonal A

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

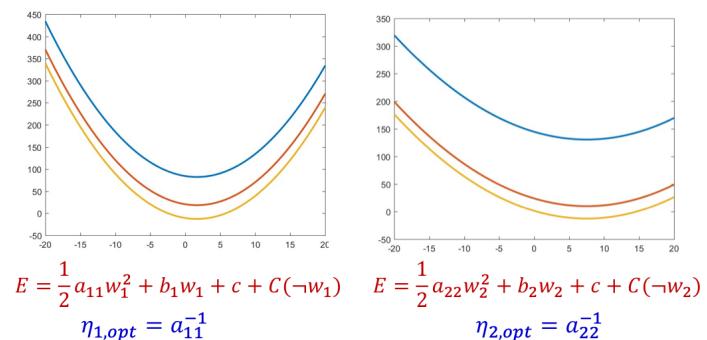


- Equal-value contours will be parallel to the axis
  - All “slices” parallel to an axis are shifted versions of one another

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

51

## “Descents” are uncoupled



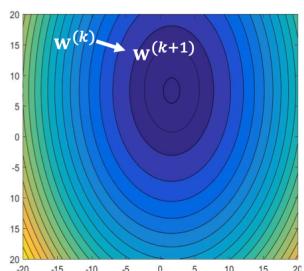
$$E = \frac{1}{2} a_{11} w_1^2 + b_1 w_1 + c + C(-w_1) \quad E = \frac{1}{2} a_{22} w_2^2 + b_2 w_2 + c + C(-w_2)$$

$$\eta_{1,opt} = a_{11}^{-1} \quad \eta_{2,opt} = a_{22}^{-1}$$

- 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**

52

## Vector update rule



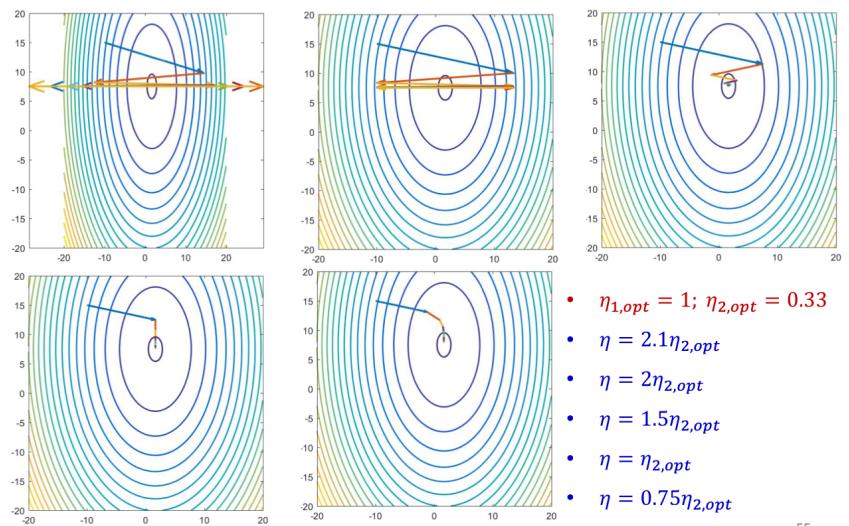
$$\mathbf{w}^{(k+1)} \leftarrow \mathbf{w}^{(k)} - \eta \nabla_{\mathbf{w}} E^T$$

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

- 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

53

## Dependence on learning rate



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

55

## Problem with vector update rule

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

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

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

- 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  $\eta_{i,opt} \leq \eta < 2\eta_{i,opt}$

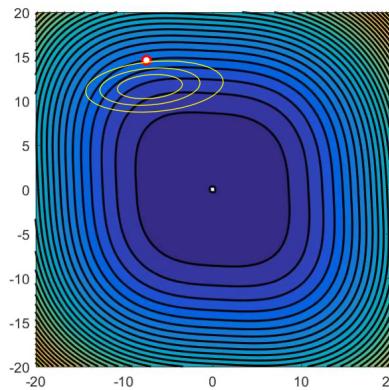
56

## Convergence

- Convergence behaviors become increasingly unpredictable as dimensions increase
- For the fastest convergence, ideally, the learning rate  $\eta$  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

70

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



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

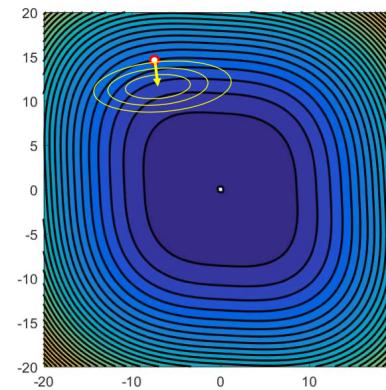
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

86

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



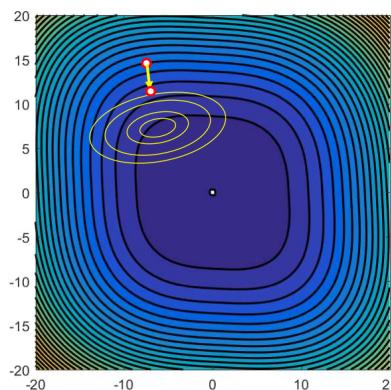
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

87

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



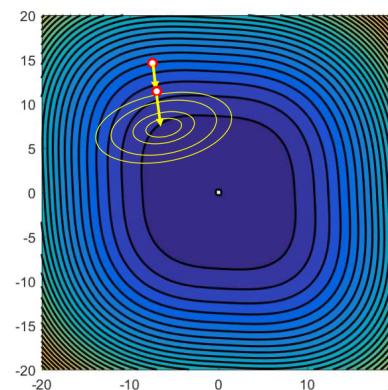
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

88

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



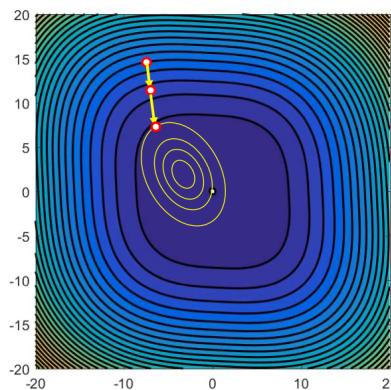
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

89

## Minimization by Newton's method



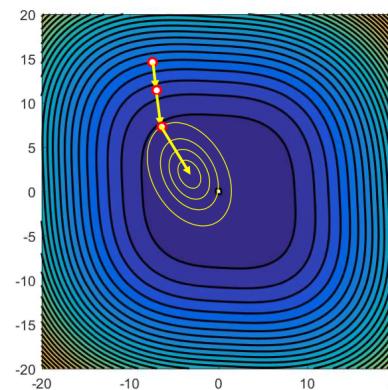
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

90

## Minimization by Newton's method



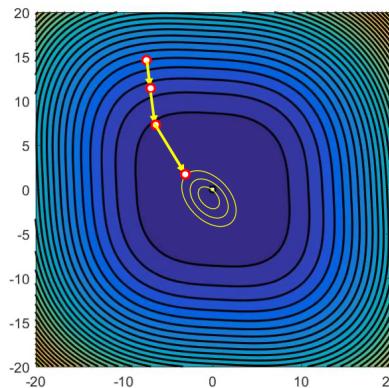
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

91

## Minimization by Newton's method



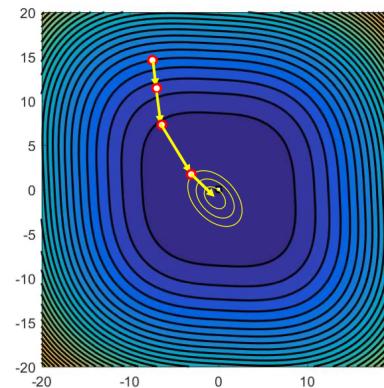
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

92

## Minimization by Newton's method



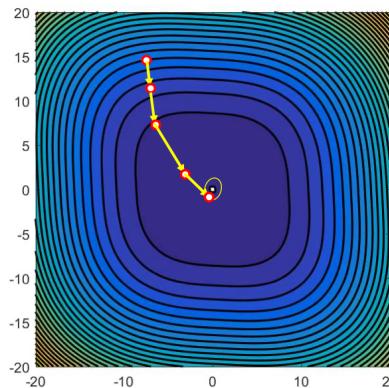
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

93

## Minimization by Newton's method



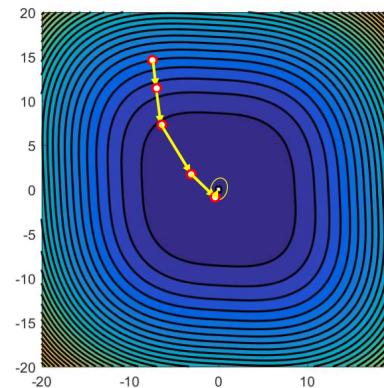
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

94

## Minimization by Newton's method



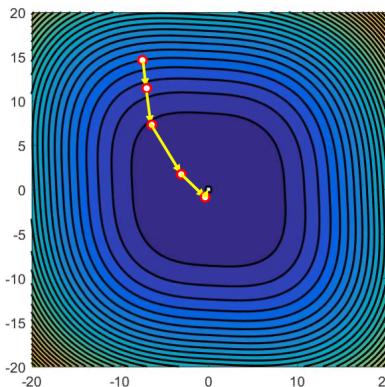
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

95

## Minimization by Newton's method



- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

96

## 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

99

## Issues: 1. The Hessian

- Normalized update rule

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

- For complex models such as neural networks, with a very large number of parameters, the Hessian  $H_E(\mathbf{w}^{(k)})$  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

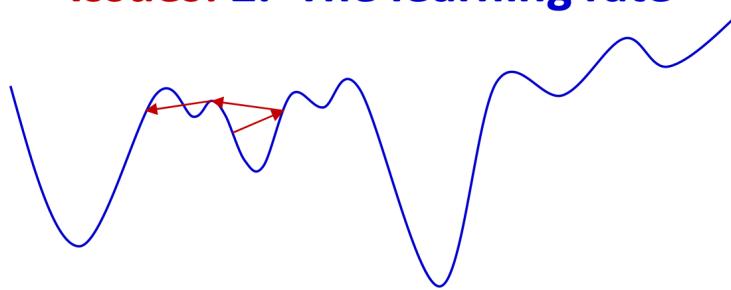
97

## 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
  - Boyd-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..

100

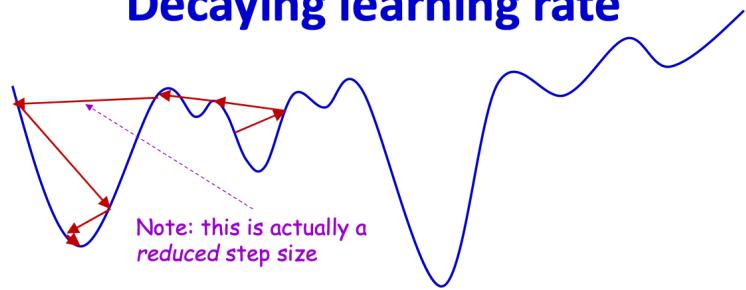
## Issues: 2. The learning rate



- 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

102

## Decaying learning rate



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

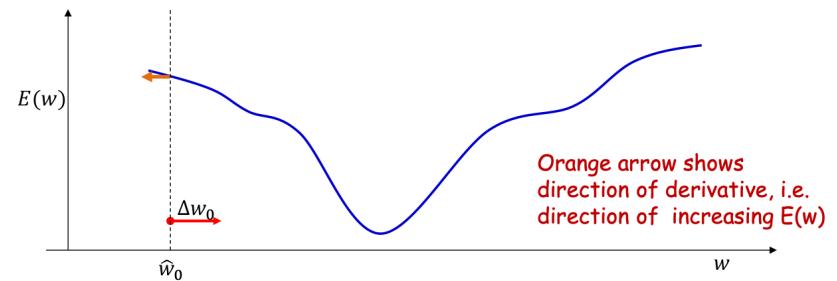
103

## 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

112

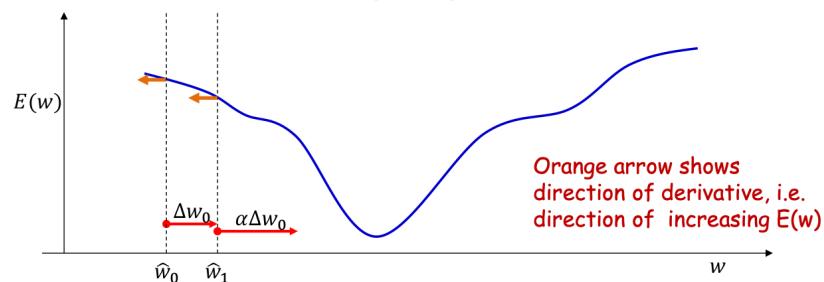
## Rprop



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

113

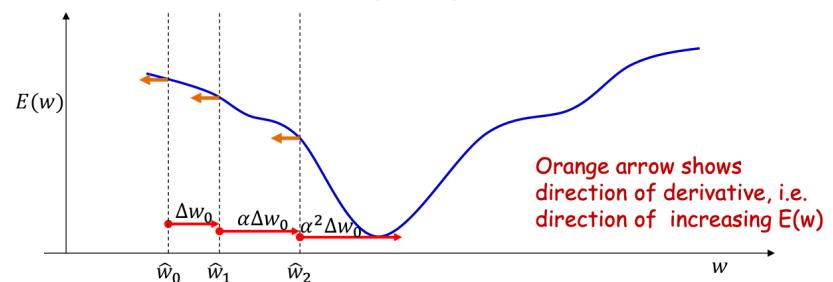
## Rprop



- 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
    - $\alpha > 1$
    - $\Delta w = \alpha \Delta w$
    - $\hat{w} = \hat{w} - \Delta w$

114

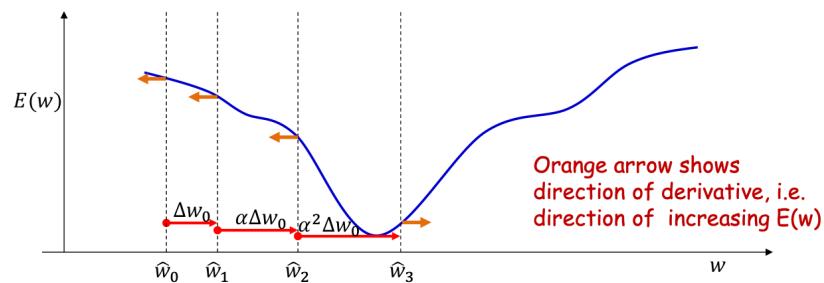
## Rprop



- 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
    - $\alpha > 1$
    - $\Delta w = \alpha \Delta w$
    - $\hat{w} = \hat{w} - \Delta w$

115

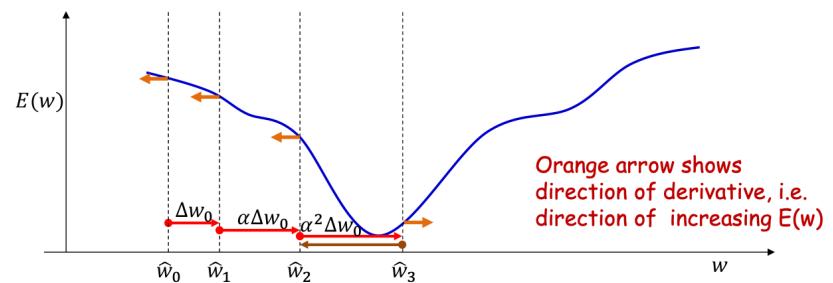
## Rprop



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

116

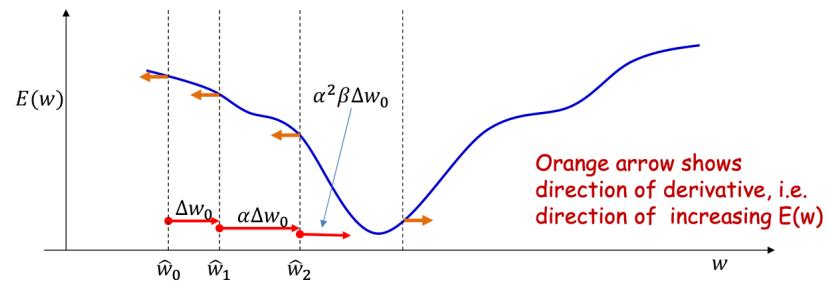
## Rprop



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

117

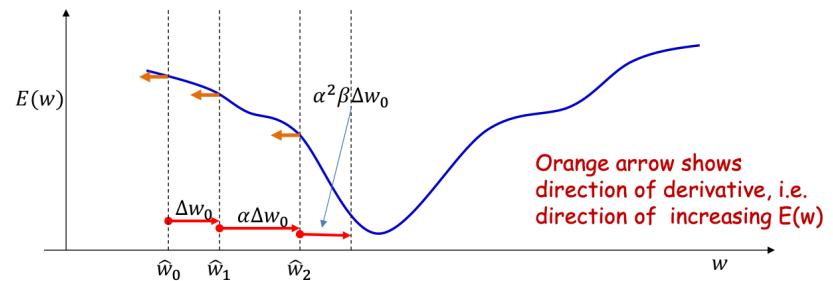
## Rprop



- 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
      - $\Delta w = \beta \Delta w$
- $\beta < 1$**

118

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

119

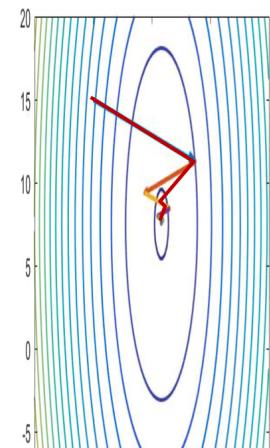
## RProp

- 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

122

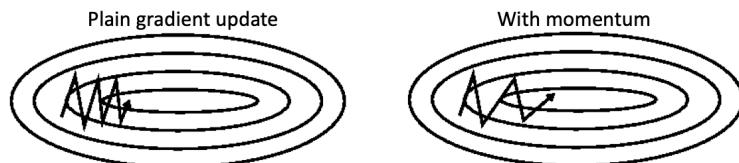
## The momentum methods

- Maintain a running average of all past steps
  - In directions in which the convergence is smooth, the average will have a large value
  - In directions in which the estimate swings, the positive and negative swings will cancel out in the average
- Update with the running average, rather than the current gradient



134

## Momentum Update



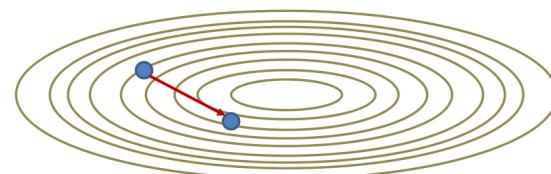
- The momentum method maintains a running average of all gradients until the *current step*

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W \text{Loss}(W^{(k-1)})^T$$
$$W^{(k)} = W^{(k-1)} + \Delta W^{(k)}$$

- Typical  $\beta$  value is 0.9
- The running average steps
  - Get longer in directions where gradient retains the same sign
  - Become shorter in directions where the sign keeps flipping

135

## Momentum Update



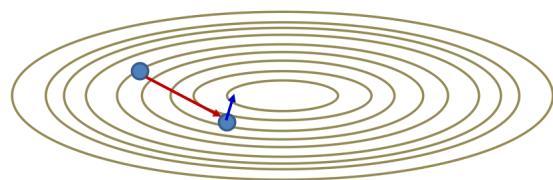
- The momentum method

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W \text{Loss}(W^{(k-1)})^T$$

- At any iteration, to compute the current step:

138

## Momentum Update



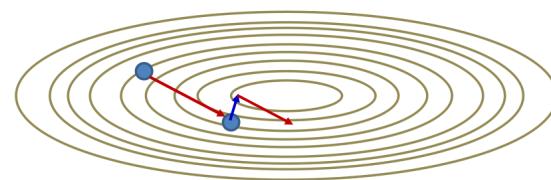
- The momentum method

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W \text{Loss}(W^{(k-1)})^T$$

- At any iteration, to compute the current step:
  - First computes the gradient step at the current location

139

## Momentum Update



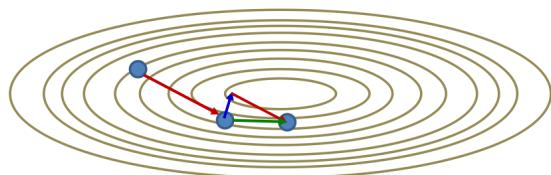
- The momentum method

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W \text{Loss}(W^{(k-1)})^T$$

- At any iteration, to compute the current step:
  - First computes the gradient step at the current location
  - Then adds in the scaled *previous step*
    - Which is actually a running average

140

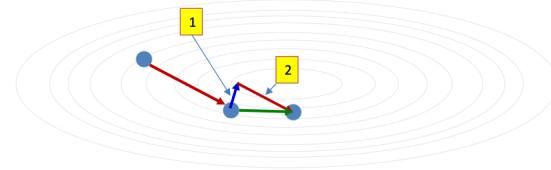
## Momentum Update



- The momentum method
$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Loss(W^{(k-1)})^T$$
- At any iteration, to compute the current step:
  - First computes the gradient step at the current location
  - Then adds in the scaled *previous* step
    - Which is actually a running average
  - To get the final step

141

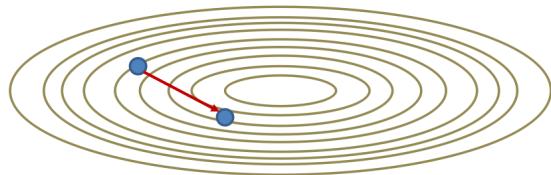
## Momentum update



- Momentum update steps are actually computed in two stages
  - First: We take a step against the gradient at the current location
  - Second: Then we add a scaled version of the previous step
- The procedure can be made more optimal by reversing the order of operations..

142

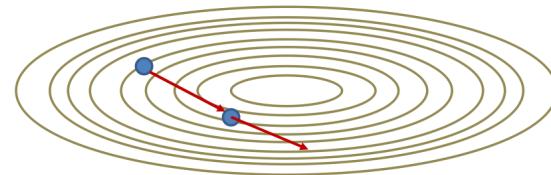
## Nestorov's Accelerated Gradient



- Change the order of operations
- At any iteration, to compute the current step:

143

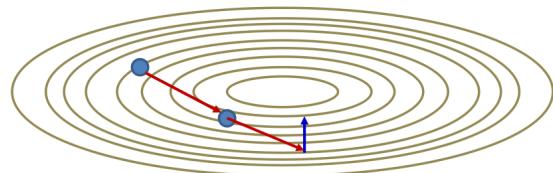
## Nestorov's Accelerated Gradient



- Change the order of operations
- At any iteration, to compute the current step:
  - First extend the previous step

144

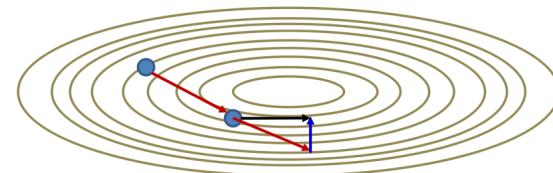
## Nestorov's Accelerated Gradient



- Change the order of operations
- At any iteration, to compute the current step:
  - First extend the previous step
  - Then compute the gradient step at the resultant position

145

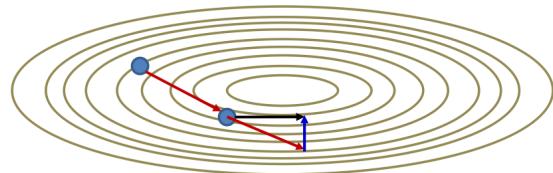
## Nestorov's Accelerated Gradient



- Change the order of operations
- At any iteration, to compute the current step:
  - First extend the previous step
  - Then compute the gradient step at the resultant position
  - Add the two to obtain the final step

146

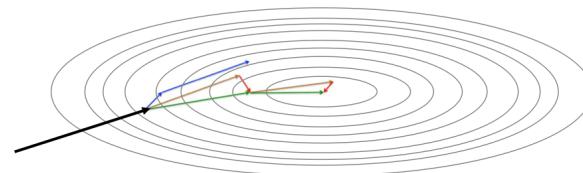
## Nestorov's Accelerated Gradient



- Nestorov's method
- $$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W \text{Loss}(W^{(k-1)} + \beta \Delta W^{(k-1)})^T$$
- $$W^{(k)} = W^{(k-1)} + \Delta W^{(k)}$$

147

## Nestorov's Accelerated Gradient



- Comparison with momentum (example from Hinton)
- Converges much faster

148