

# CSC 561: Neural Networks and Deep Learning

## Optimization (part 1)

Marco Alvarez

Department of Computer Science and Statistics  
University of Rhode Island

Spring 2025



## Neural Networks: Optimization Part 1

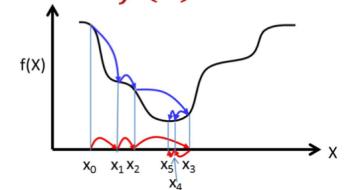
Intro to Deep Learning, Spring 2025

1

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

## Recap: Gradient Descent Algorithm

- In order to minimize any function  $f(x)$  w.r.t.  $x$
- Initialize:
  - $x^0$
  - $k = 0$
- Do
  - $k = k + 1$
  - $x^{k+1} = x^k - \eta \nabla_x f^T$
- while  $|f(x^k) - f(x^{k-1})| > \varepsilon$



3

## Does backprop do the right thing?

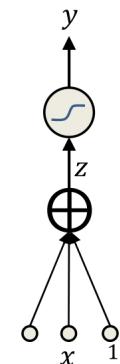
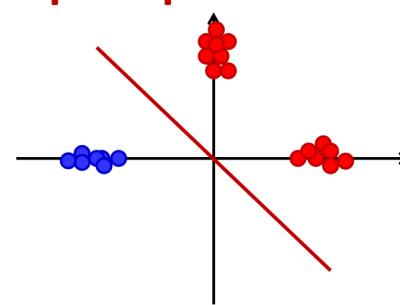
- Is backprop always right?

- Assuming it actually finds the minimum of the divergence function?

(Actual question: Does gradient descent find the right solution, even when it finds the actual minimum)

12

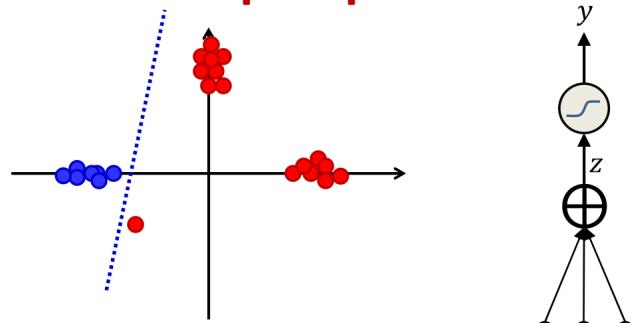
## Backprop fails to separate where perceptron succeeds



- Brady, Raghavan, Slawny, '89
- Several linearly separable training examples
- Simple setup: **both backprop and perceptron algorithms find solutions**

24

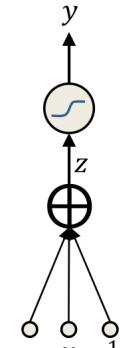
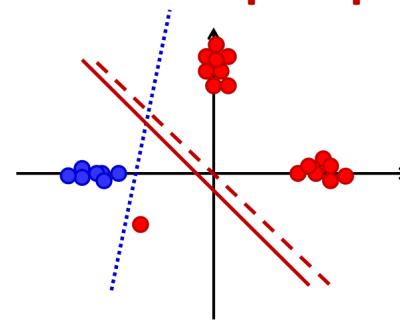
## A more complex problem



- Adding a “spoiler” (or a small number of spoilers)
  - Perceptron finds the linear separator,

25

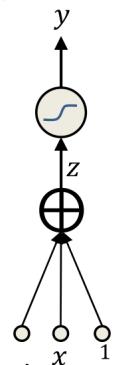
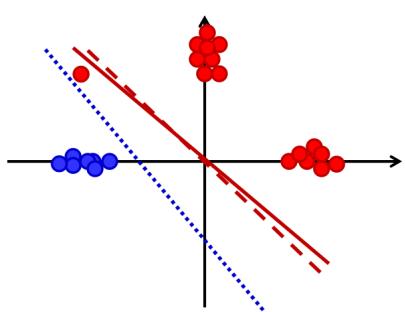
## A more complex problem



- Adding a “spoiler” (or a small number of spoilers)
  - Perceptron finds the linear separator,
  - Backprop does not find a separator
    - A single additional input does not change the loss function significantly
      - Assuming weights are constrained to be bounded

26

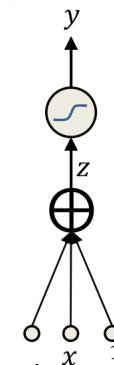
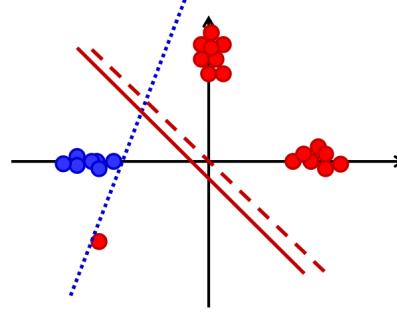
## A more complex problem



- Adding a “spoiler” (or a small number of spoilers)
  - Perceptron finds the linear separator,
  - For bounded  $w$ , backprop does not find a separator
    - A single additional input does not change the loss function significantly

27

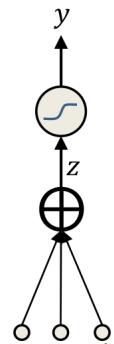
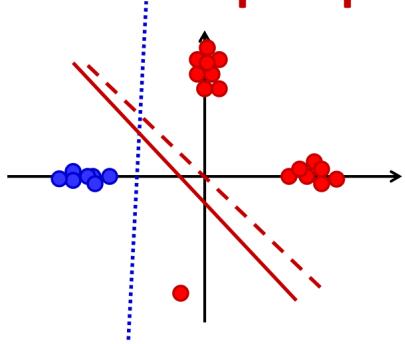
## A more complex problem



- Adding a “spoiler” (or a small number of spoilers)
  - Perceptron finds the linear separator,
  - For bounded  $w$ , backprop does not find a separator
    - A single additional input does not change the loss function significantly

28

## A more complex problem



- Adding a “spoiler” (or a small number of spoilers)
  - Perceptron finds the linear separator,
  - For bounded  $w$ , backprop does not find a separator
    - A single additional input does not change the loss function significantly

29

## So what is happening here?

- The perceptron may change greatly upon adding just a single new training instance
  - But it fits the training data well
  - The perceptron rule has *low bias*
    - Makes no errors if possible
  - But high variance
    - Swings wildly in response to small changes to input
- Backprop is minimally changed by new training instances
  - Prefers consistency over perfection
  - It is a *low-variance* estimator, at the potential cost of bias

30

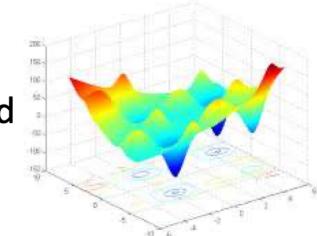
## Backpropagation: Finding the separator

- Backpropagation will often not find a separating solution **even though the solution is within the class of functions learnable by the network**
- This is because the separating solution is not a feasible optimum for the loss function
- One resulting benefit is that a backprop-trained neural network classifier has lower variance than an optimal classifier for the training data

33

## The Loss Surface

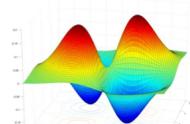
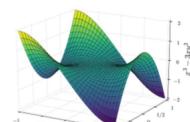
- The example (and statements) earlier assumed the loss objective had a single global optimum that could be found
  - Statement about variance is assuming global optimum
- What about local optima



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



37

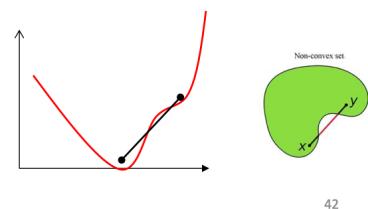
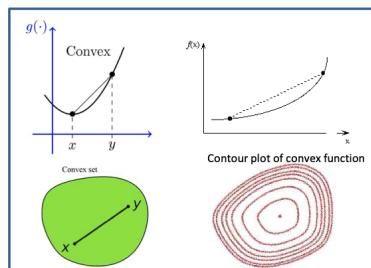
## Convergence

- In the discussion so far we have assumed the training arrives at a local minimum
- Does it always converge?
- How long does it take?
- Hard to analyze for an MLP, but we can look at the problem through the lens of convex optimization

40

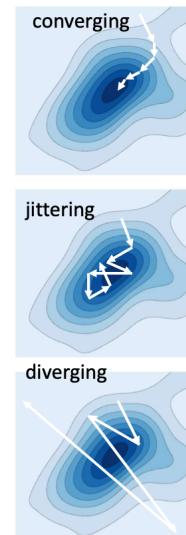
## Convex Loss Functions

- A surface is “convex” if it is continuously curving upward
  - We can connect any two points on or above the surface without intersecting it
  - Many mathematical definitions that are equivalent
- Caveat: Neural network loss surface is generally not convex
  - Streetlight effect



## Convergence of gradient descent

- An iterative algorithm is said to *converge* to a solution if the value updates arrive at a fixed point
  - Where the gradient is 0 and further updates do not change the estimate
- The algorithm may not actually converge
  - It may jitter around the local minimum
  - It may even diverge
- Conditions for convergence?

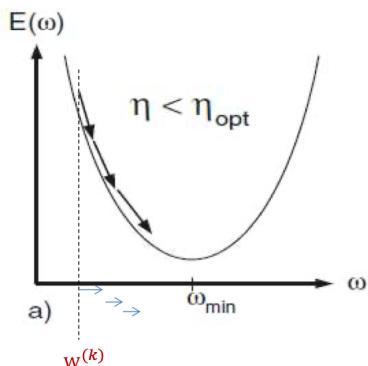


## Convergence for quadratic surfaces

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

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

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



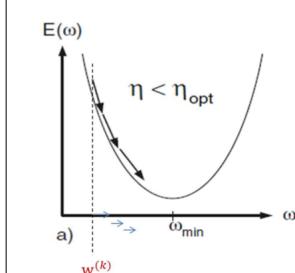
- Gradient descent to find the optimum of a quadratic, starting from  $w^{(k)}$
- Assuming fixed step size  $\eta$
- What is the optimal step size  $\eta$  to get there fastest?

45

## Convergence for quadratic surfaces

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

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



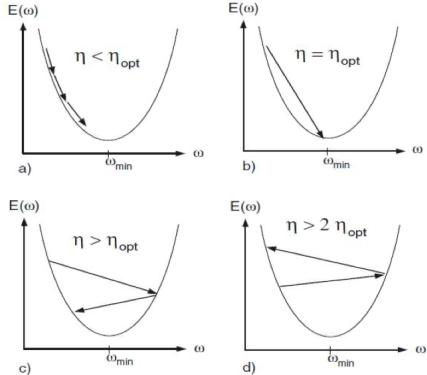
- Any quadratic objective can be written as
 
$$E(w) = E(w^{(k)}) + E'(w^{(k)})(w - w^{(k)}) + \frac{1}{2} E''(w^{(k)})(w - w^{(k)})^2$$
  - Taylor expansion
- Minimizing w.r.t  $w$ , we get (Newton's method)
 
$$w_{min} = w^{(k)} - E''(w^{(k)})^{-1} E'(w^{(k)})$$
- Note:
 
$$\frac{dE(w^{(k)})}{dw} = E'(w^{(k)})$$
- Comparing to the gradient descent rule, we see that we can arrive at the optimum in a single step using the optimum step size
 
$$\eta_{opt} = E''(w^{(k)})^{-1} = a^{-1}$$

46

## With non-optimal step size

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

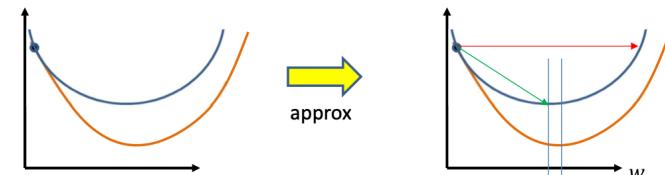
Gradient descent with fixed step size  $\eta$  to estimate scalar parameter  $w$



- 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

47

## For generic differentiable convex objectives

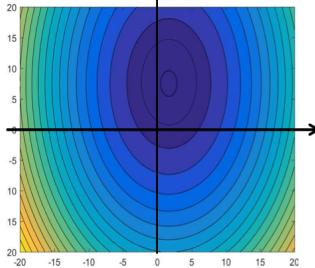


- Any differentiable convex objective  $E(w)$  can be approximated as
 
$$E \approx E(w^{(k)}) + (w - w^{(k)}) \frac{dE(w^{(k)})}{dw} + \frac{1}{2} (w - w^{(k)})^2 \frac{d^2E(w^{(k)})}{dw^2} + \dots$$
 – Taylor expansion
- Using the same logic as before, we get (Newton's method)
 
$$\eta_{opt} = \left( \frac{d^2E(w^{(k)})}{dw^2} \right)^{-1}$$
- We can get divergence if  $\eta \geq 2\eta_{opt}$

48

## Multivariate Quadratic with Diagonal A

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

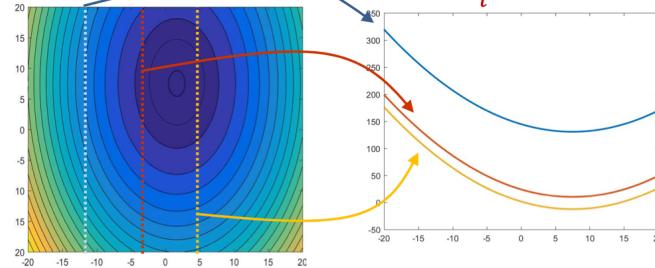


- Equal-value contours will ellipses with principal axes parallel to the spatial axes

50

## Multivariate Quadratic with Diagonal A

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



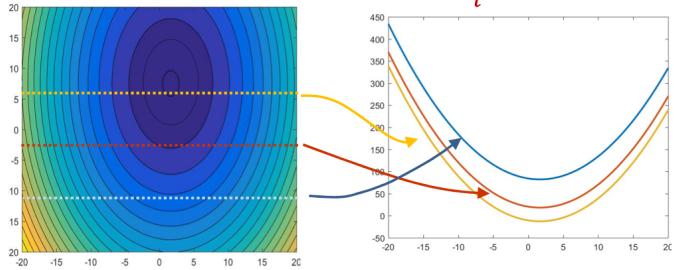
- Equal-value contours will be parallel to the axes
  - All “slices” parallel to an axis are shifted versions of one another

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

51

## Multivariate Quadratic with Diagonal A

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

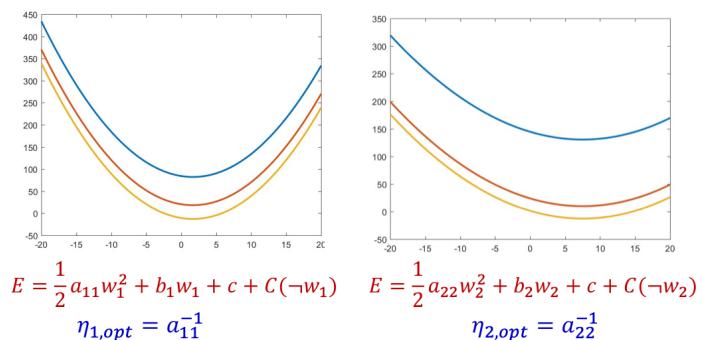


- Equal-value contours will be parallel to the axis
  - All “slices” parallel to an axis are shifted versions of one another

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

52

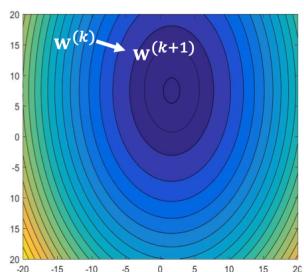
## “Descents” are uncoupled



- 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**

53

## Vector update rule



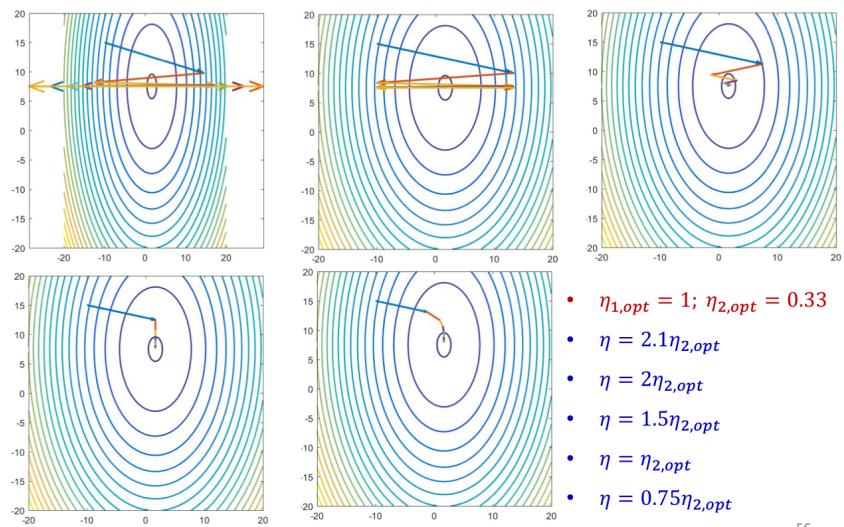
$$\mathbf{w}^{(k+1)} \leftarrow \mathbf{w}^{(k)} - \eta \nabla_{\mathbf{w}} E^T$$

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

- 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

54

## Dependence on learning rate



56

## Problem with vector update rule

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

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

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

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

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

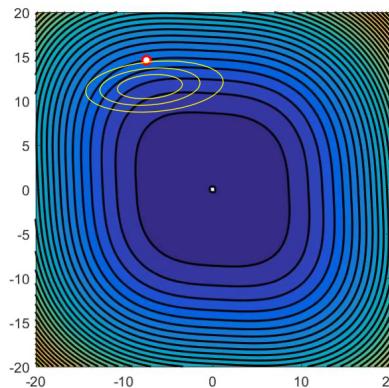
– Otherwise the learning will diverge

- This, however, makes the learning very slow

– And will oscillate in all directions where  $\eta_{i,opt} \leq \eta < 2\eta_{i,opt}$

57

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



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

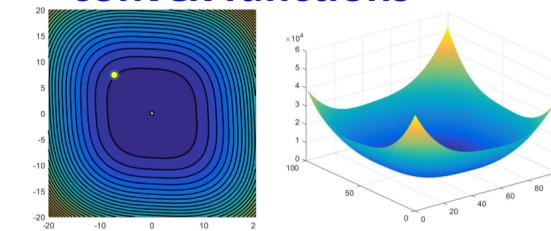
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

87

## Generic differentiable multivariate convex functions



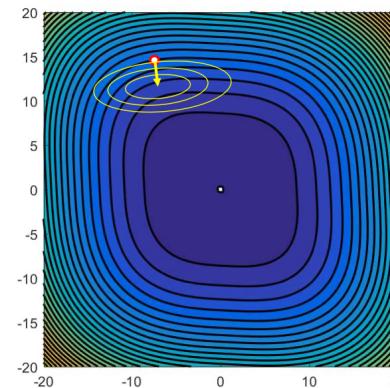
- For generic convex multivariate functions (not necessarily quadratic), we can employ quadratic Taylor series expansions and much of the analysis still applies
- Taylor expansion

$$E(\mathbf{w}) \approx E(\mathbf{w}^{(k)}) + \nabla_{\mathbf{w}} E(\mathbf{w}^{(k)}) (\mathbf{w} - \mathbf{w}^{(k)}) + \frac{1}{2} (\mathbf{w} - \mathbf{w}^{(k)})^T H_E(\mathbf{w}^{(k)}) (\mathbf{w} - \mathbf{w}^{(k)})$$

- The optimal step size is inversely proportional to the Eigen values of the Hessian
  - The second derivative along the orthogonal coordinates
  - For the smoothest convergence, these must all be equal

59

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



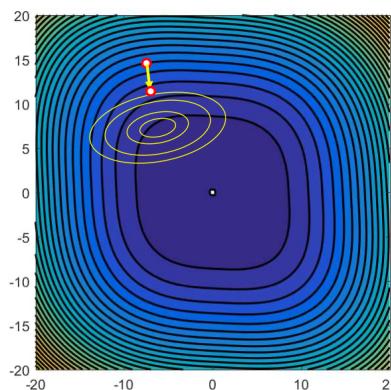
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

88

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



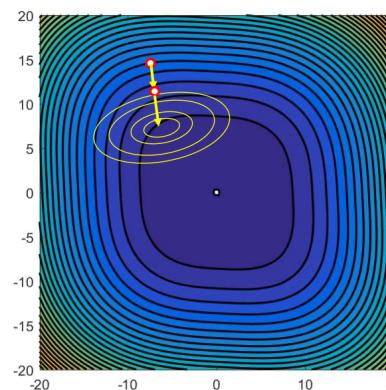
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

89

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



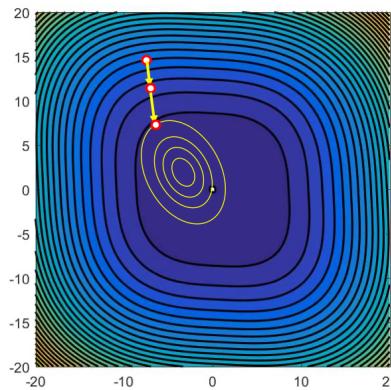
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

90

## Minimization by Newton's method



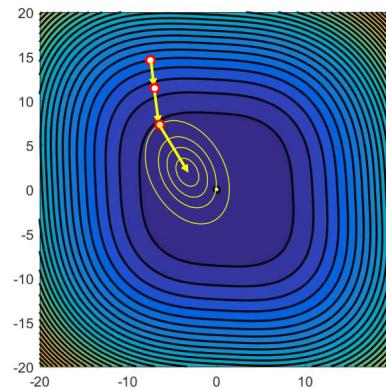
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

91

## Minimization by Newton's method



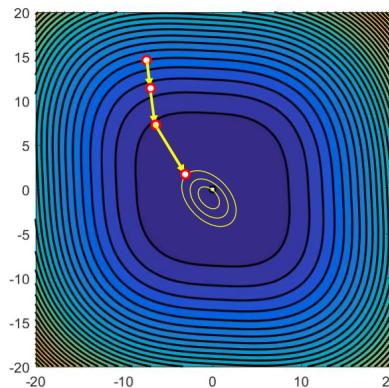
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

92

## Minimization by Newton's method



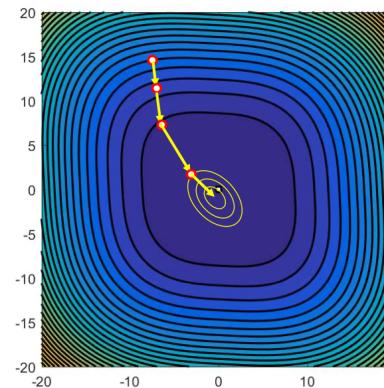
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

93

## Minimization by Newton's method



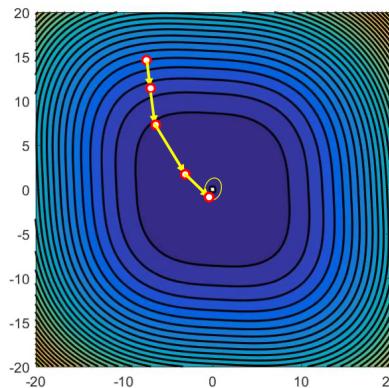
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

94

## Minimization by Newton's method



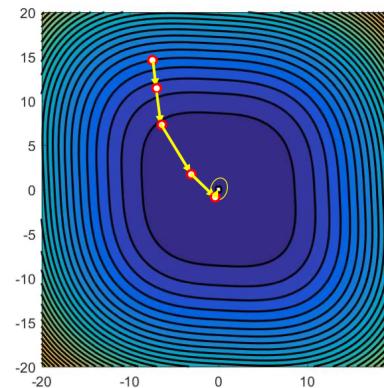
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

95

## Minimization by Newton's method



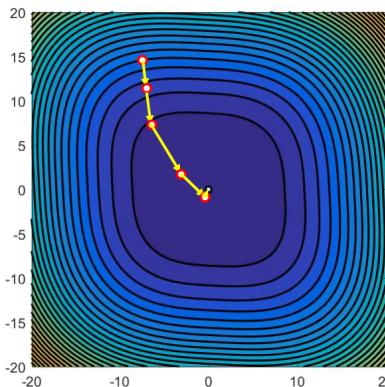
- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

96

## Minimization by Newton's method



- Iterated localized optimization with quadratic approximations

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

–  $\eta = 1$

97

## 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

100

## 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

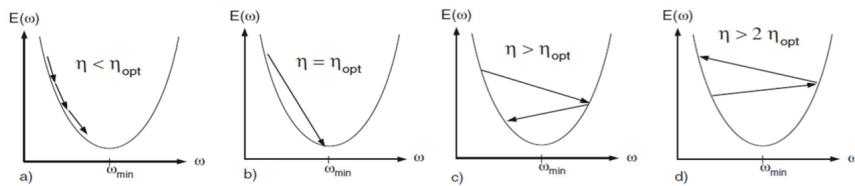
98

## 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..

101

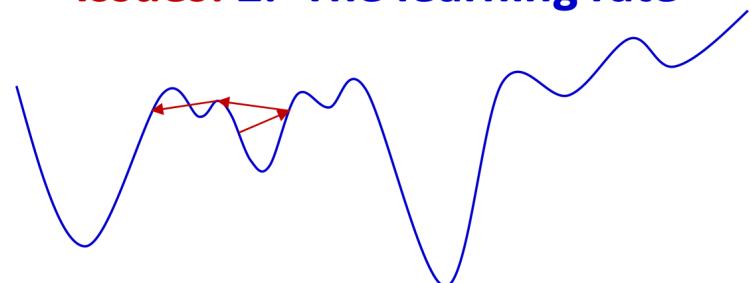
## Issues: 2. The learning rate



- Much of the analysis we just saw was based on trying to ensure that the step size was not so large as to cause divergence within a convex region
  - $\eta < 2\eta_{opt}$

102

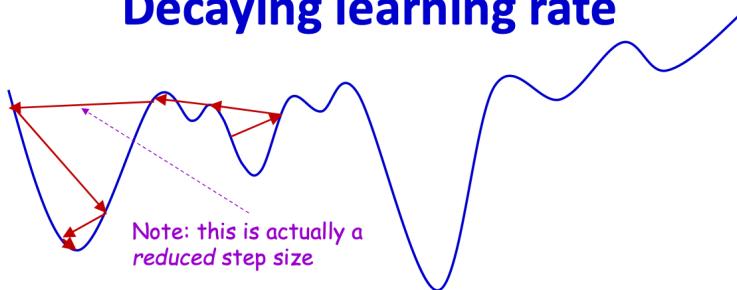
## Issues: 2. The learning rate



- 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

103

## Decaying learning rate



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

104

## Decaying learning rate

- Typical decay schedules
  - Linear decay:  $\eta_k = \frac{\eta_0}{k+1}$
  - Quadratic decay:  $\eta_k = \frac{\eta_0}{(k+1)^2}$
  - Exponential decay:  $\eta_k = \eta_0 e^{-\beta k}$ , where  $\beta > 0$
- A common approach (for nnets):
  - Train with a fixed learning rate  $\eta$  until loss (or performance on a held-out data set) stagnates
  - $\eta \leftarrow \alpha\eta$ , where  $\alpha < 1$  (typically 0.1)
  - Return to step 1 and continue training from where we left off

105

## Story so far : Convergence

- Gradient descent can miss obvious answers
  - And this may be a *good* thing
- Convergence issues abound
  - The loss surface has many saddle points
    - Although, perhaps, not so many bad local minima
    - Gradient descent can stagnate on saddle points
  - Vanilla gradient descent may not converge, or may converge tooooooo slowly
    - The optimal learning rate for one component may be too high or too low for others

106

## Derivative-inspired algorithms

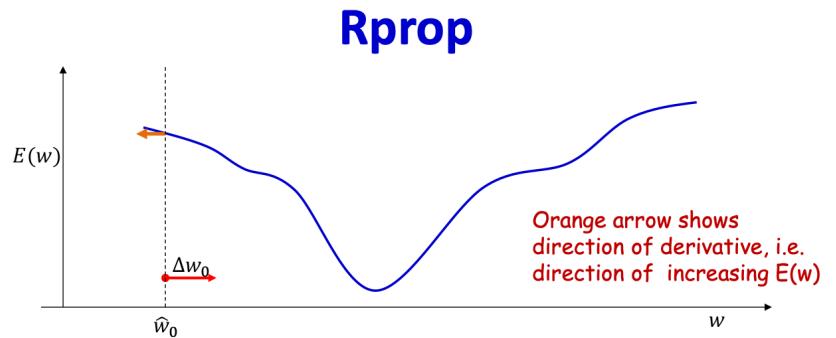
- Algorithms that use derivative information for trends, but do not follow them absolutely
- Rprop
- Quick prop

112

## 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

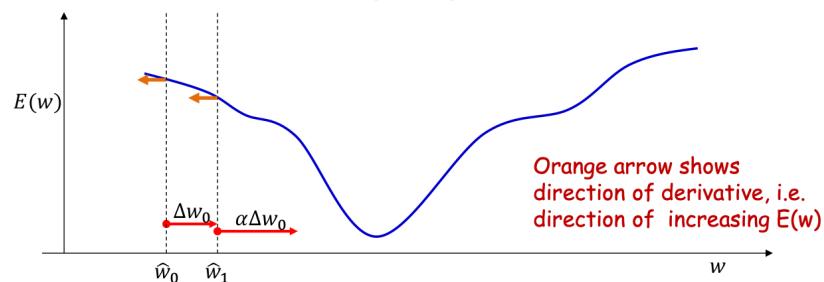
113



- 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
      - $\Delta w = \text{sign} \left( \frac{dE(\hat{w})}{dw} \right) \Delta w$
      - $\hat{w} = \hat{w} - \Delta w$

114

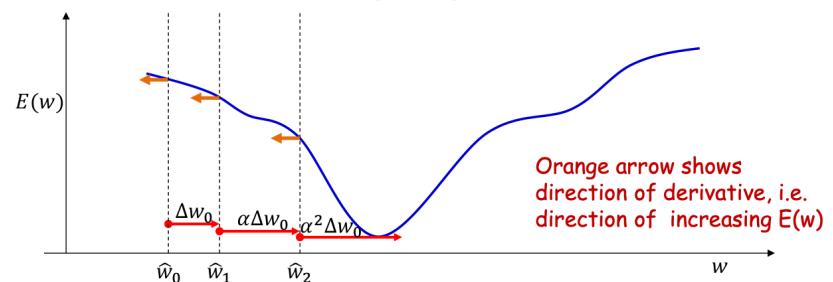
## Rprop



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

115

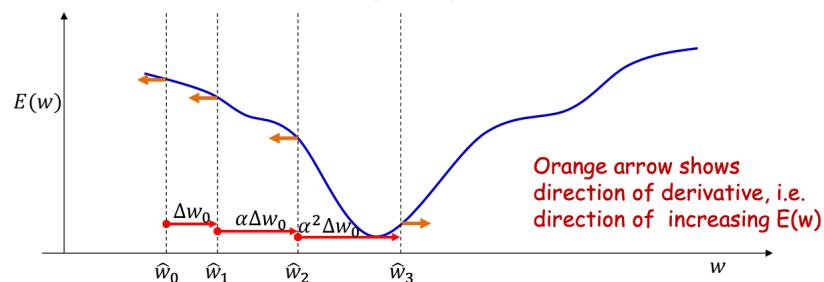
## Rprop



- Compute the derivative in the new location
  - If the derivative has not changed sign from the previous location, increase the step size and take a step
    - $\alpha > 1$
    - $\Delta w = \alpha \Delta w$
    - $\hat{w} = \hat{w} - \Delta w$

116

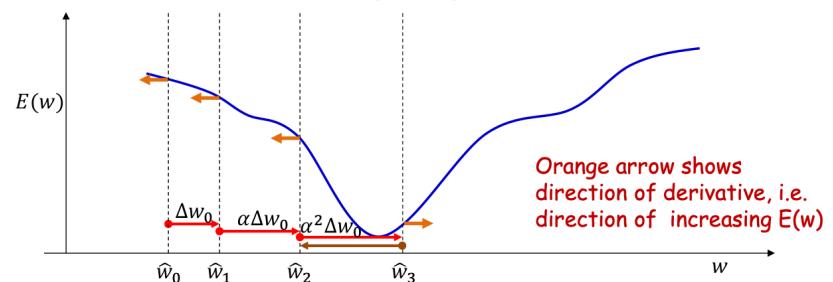
## Rprop



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

117

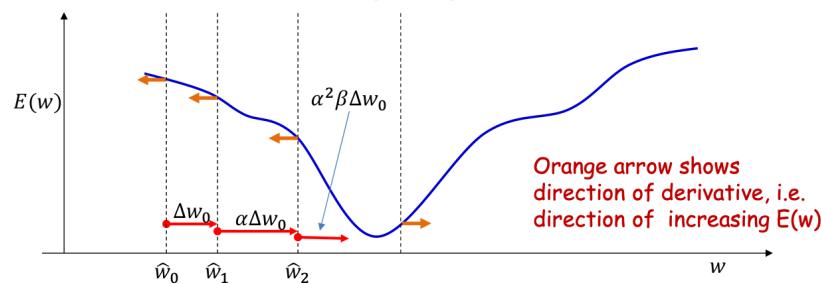
## Rprop



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

118

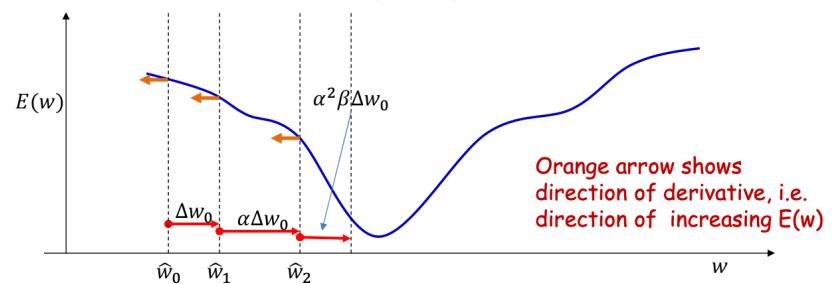
## Rprop



- 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
    - $\Delta w = \beta \Delta w$

119

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

120

## Rprop (simplified)

- Set  $\alpha = 1.2, \beta = 0.5$
- For each layer  $l$ , for each  $i, j$ :
  - Initialize  $w_{l,i,j}, \Delta w_{l,i,j} > 0$ ,
  - $prevD(l, i, j) = \frac{d\text{Loss}(w_{l,i,j})}{dw_{l,i,j}}$
  - $\Delta w_{l,i,j} = \text{sign}(prevD(l, i, j)) \Delta w_{l,i,j}$
  - While not converged:
    - $w_{l,i,j} = w_{l,i,j} - \Delta w_{l,i,j}$
    - $D(l, i, j) = \frac{d\text{Loss}(w_{l,i,j})}{dw_{l,i,j}}$
    - If  $\text{sign}(prevD(l, i, j)) == \text{sign}(D(l, i, j))$ :
      - $\Delta w_{l,i,j} = \alpha \Delta w_{l,i,j}$
      - $prevD(l, i, j) = D(l, i, j)$
    - else:
      - $w_{l,i,j} = w_{l,i,j} + \Delta w_{l,i,j}$
      - $\Delta w_{l,i,j} = \beta \Delta w_{l,i,j}$

122

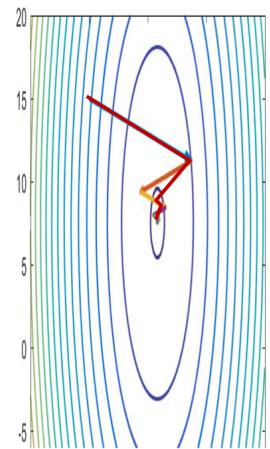
## RProp

- 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

123

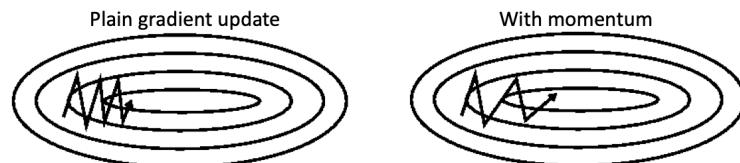
## The momentum methods

- 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



135

## Momentum Update



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

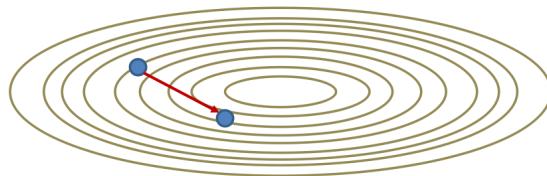
$$\begin{aligned}\text{gradientstep} &= -\eta \nabla_W \text{Loss}(W^{(k-1)})^T \\ \Delta W^{(k)} &= \beta \Delta W^{(k-1)} + \text{gradientstep} \\ W^{(k)} &= W^{(k-1)} + \Delta W^{(k)}\end{aligned}$$

- 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

136

## Momentum Update



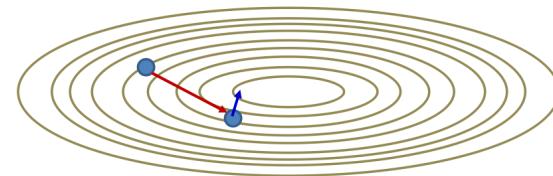
- The momentum method

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

- At any iteration, to compute the current step:

139

## Momentum Update



- The momentum method

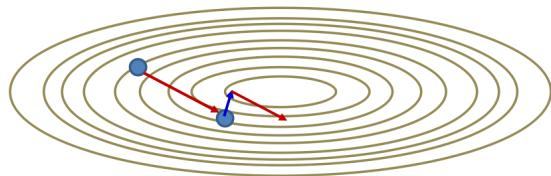
$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W \text{Loss}(W^{(k-1)})^T$$

- At any iteration, to compute the current step:

- First computes the gradient step at the current location

140

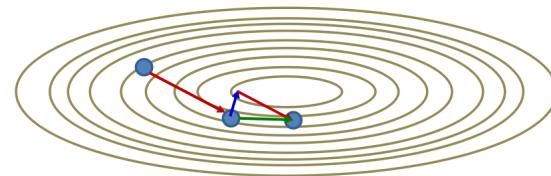
## Momentum Update



- 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

141

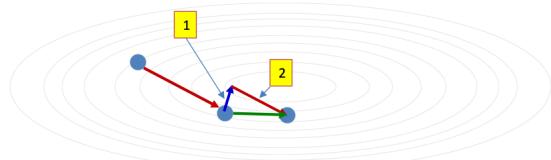
## Momentum Update



- 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

142

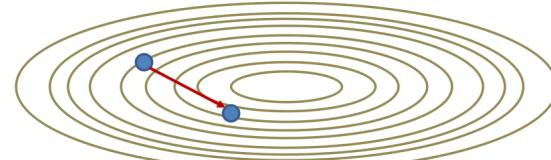
## Momentum update



- 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..

143

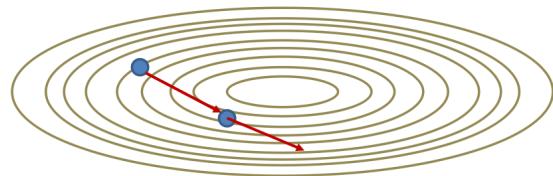
## Nestorov's Accelerated Gradient



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

144

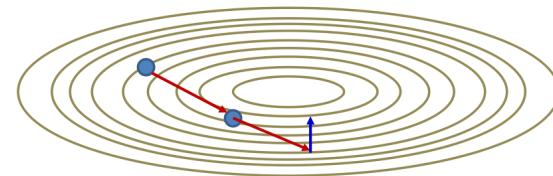
## Nestorov's Accelerated Gradient



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

145

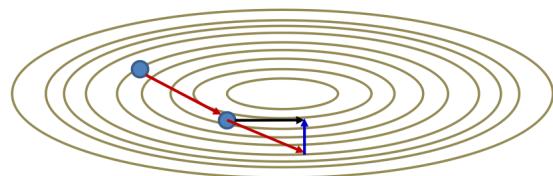
## Nestorov's Accelerated Gradient



- 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

146

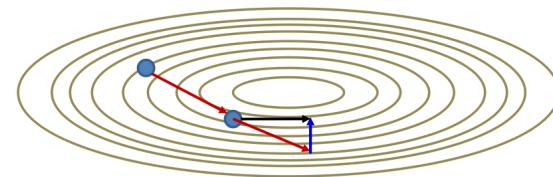
## Nestorov's Accelerated Gradient



- 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

147

## Nestorov's Accelerated Gradient



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

148

## Story so far

- Gradient descent can miss obvious answers
  - And this may be a *good* thing
- Vanilla gradient descent may be too slow or unstable due to the differences between the dimensions
- Second order methods can normalize the variation across dimensions, but are complex
- Adaptive or decaying learning rates can improve convergence
- Methods that decouple the dimensions can improve convergence
- Momentum methods which emphasize directions of steady improvement are demonstrably superior to other methods