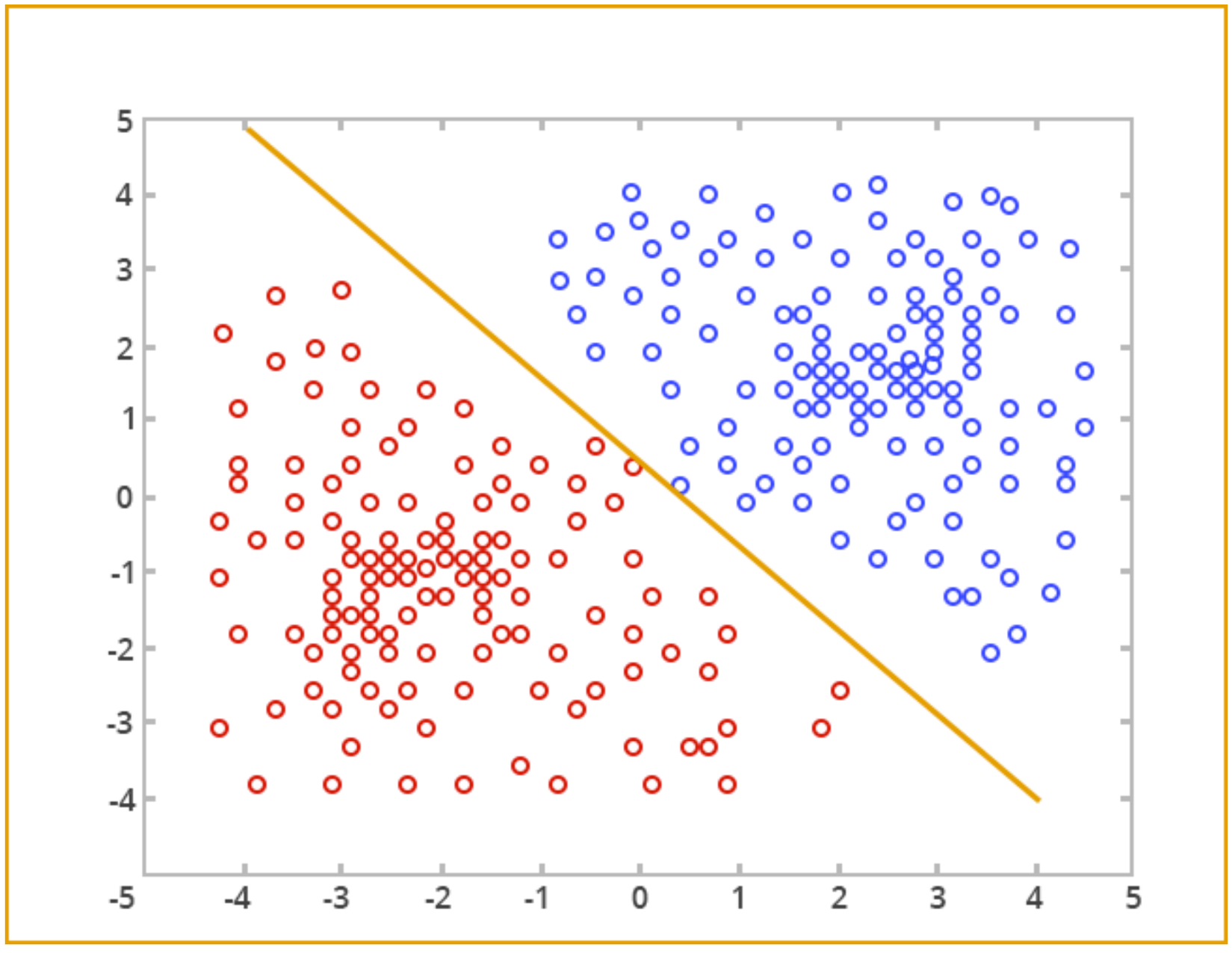


Iterative Reweighted Least Squares

Leandro L. Minku

Outline

- Problems with Gradient Descent
- Iterative Reweighed Least Squares / Newton-Raphson
  - Intuition of the update rule
  - How the update rule was obtained

Gradient Descent: Local Minima and Learning Rate

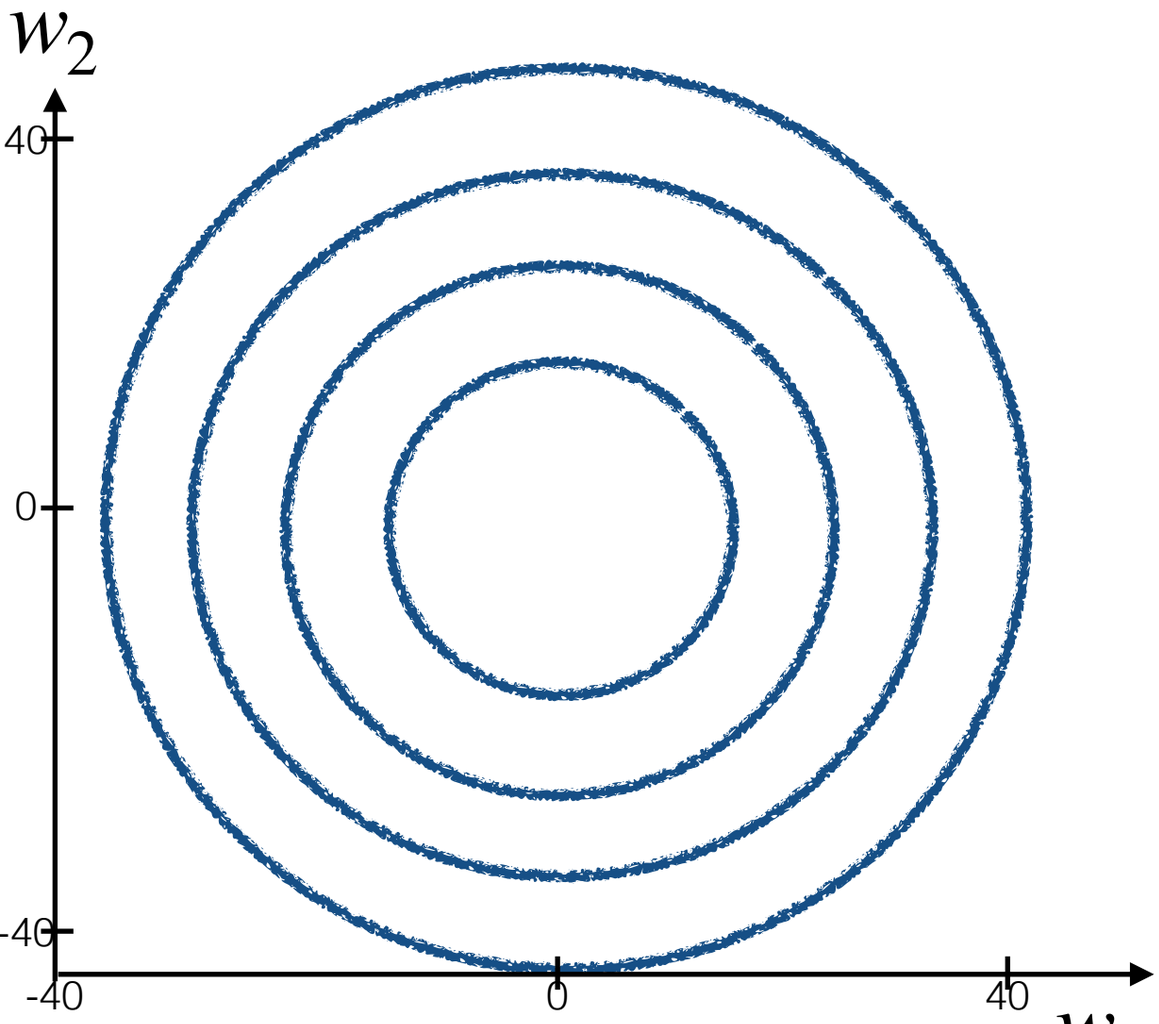
$$\mathbf{w} = \mathbf{w} - \eta \nabla E(\mathbf{w})$$

- Gradient descent can get stuck in local minima.
  - But this is not an issue in the case of Logistic Regression using Cross-Entropy Loss, because the function being optimised is strictly convex.

- We have also discussed that a too large learning rate can make it difficult to find the optimum.
  - The algorithm may jump across the optimum.

- A too small learning rate is also not ideal.
  - It may take a long time to find the optimum.

Differential Curvature

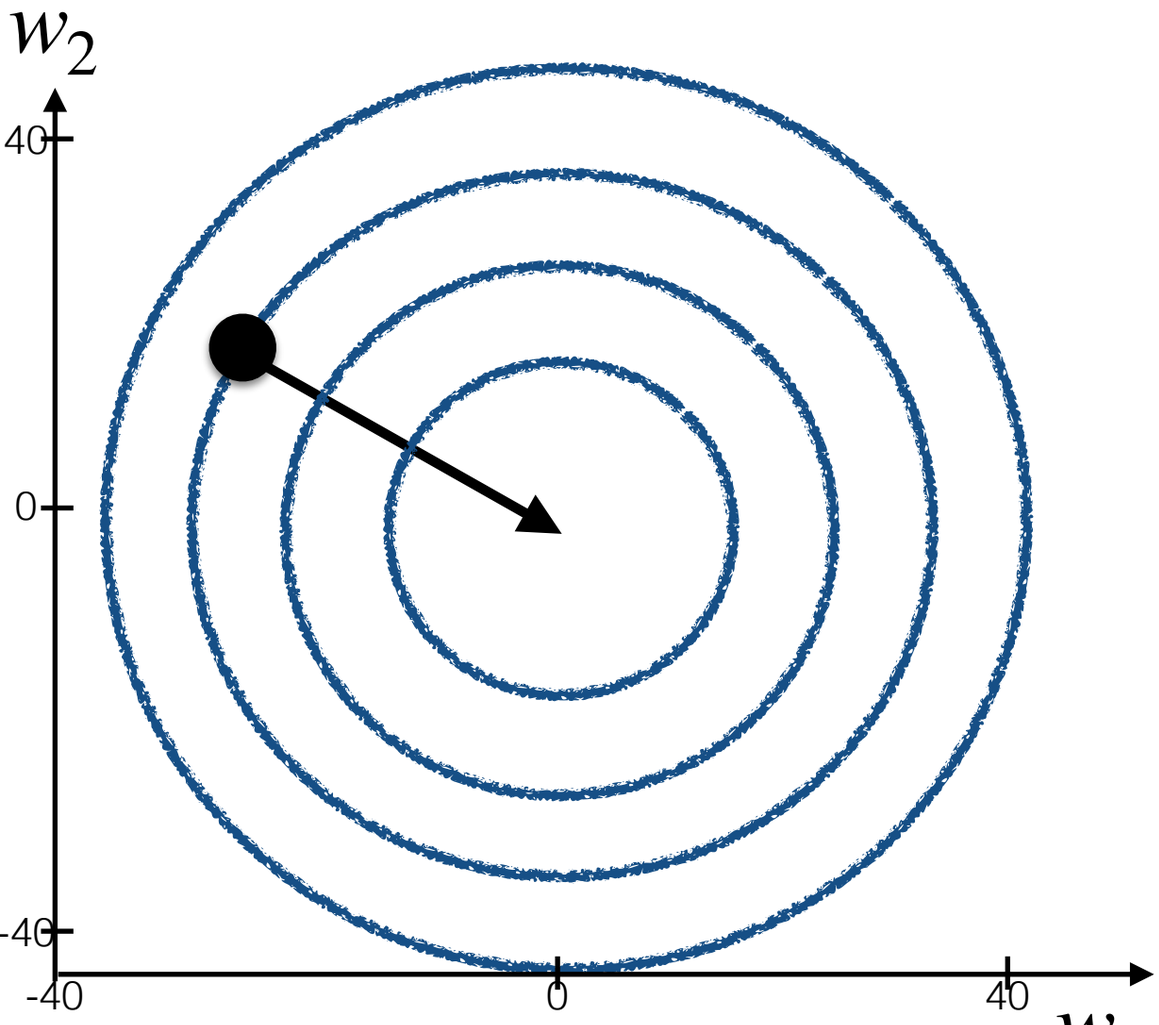


$$E(\mathbf{w}) = w_1^2 + w_2^2$$

Contour plots of the loss function, where each line corresponds to points in the input space where the loss is the same.

Figure adapted from: C. Aggarwal. Linear Algebra and Optimization for Machine Learning. Springer, 2020.

Differential Curvature

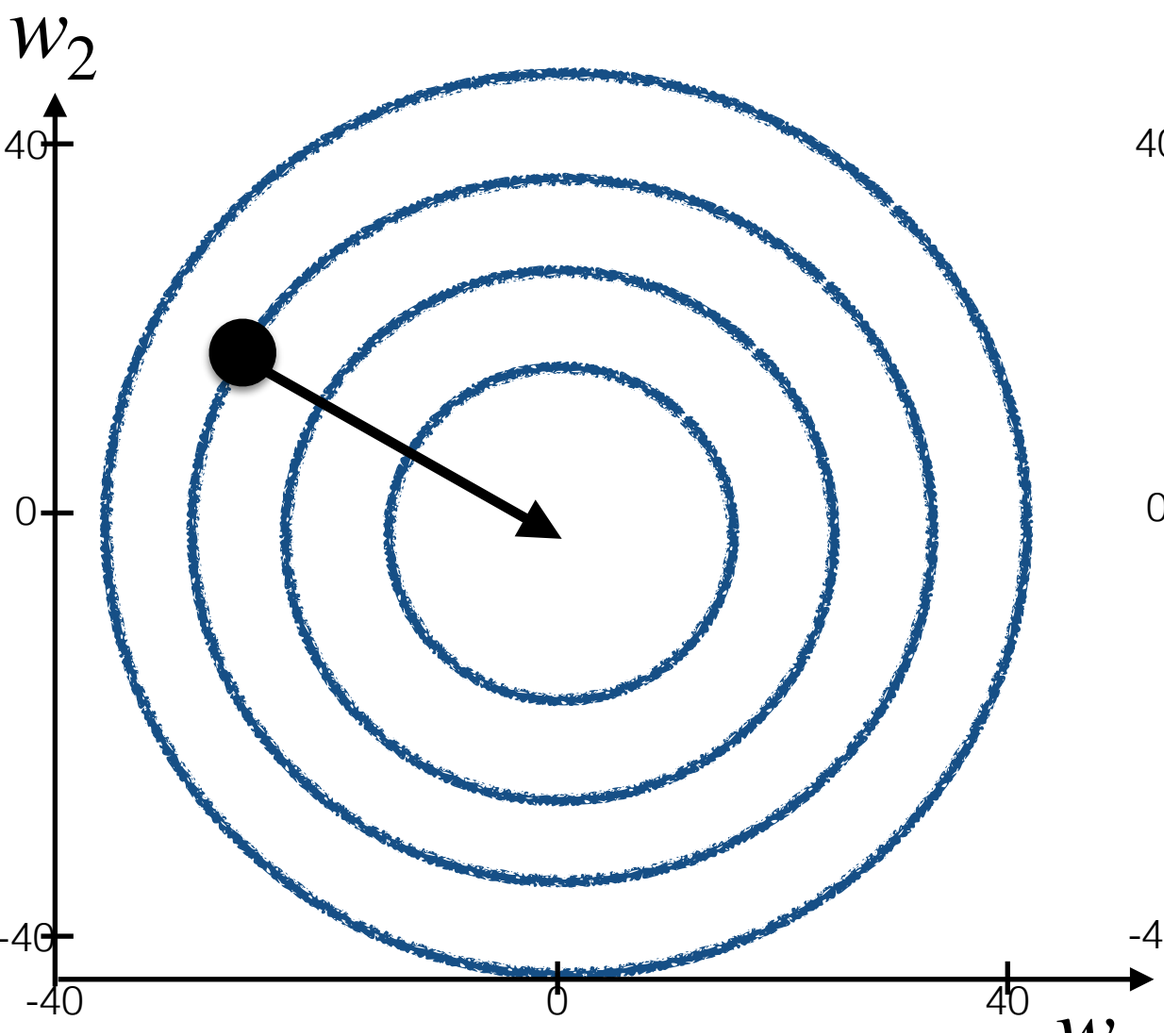


$$E(\mathbf{w}) = w_1^2 + w_2^2$$

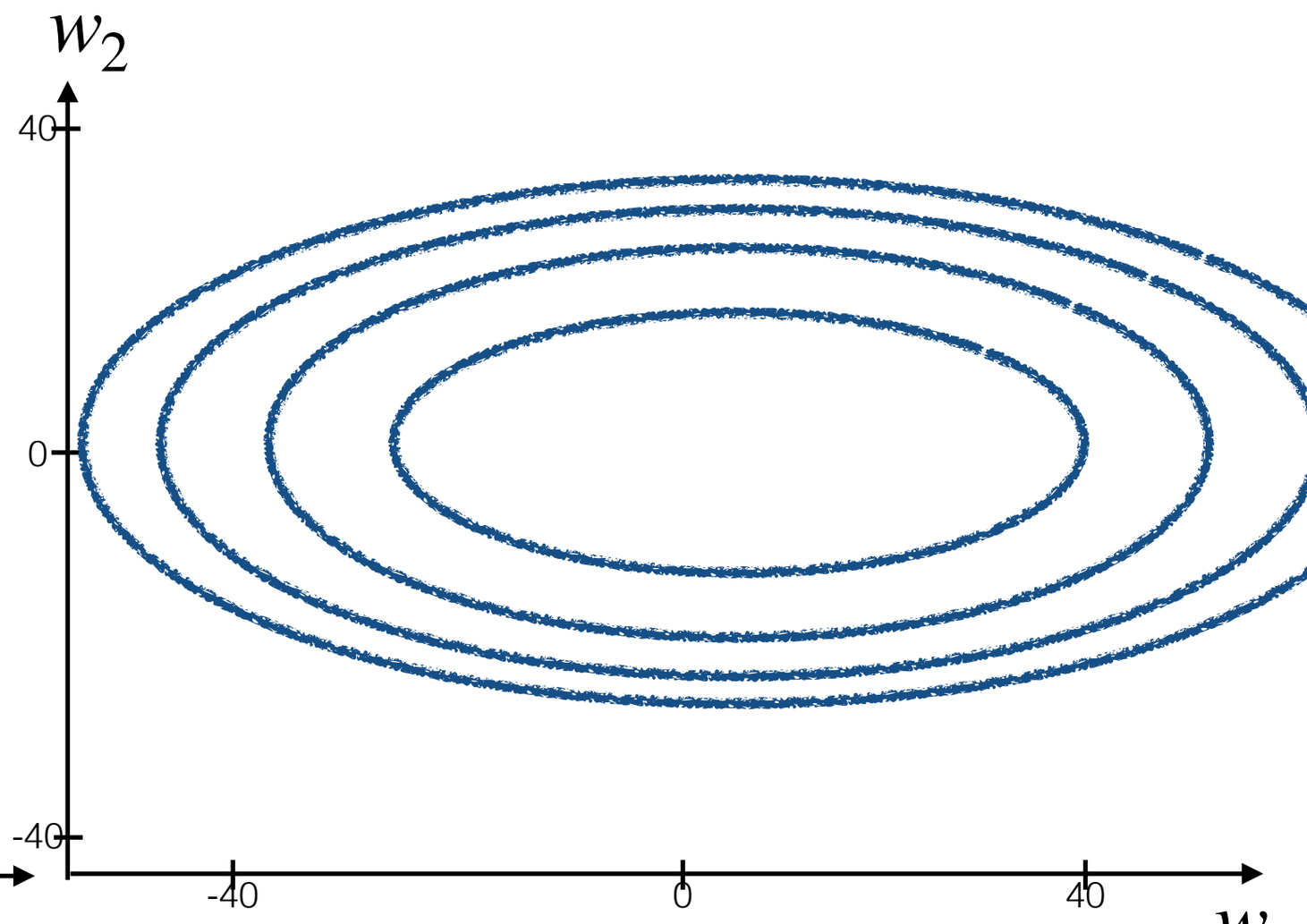
Steepest descent forms a 90 degree angle with the line.

Figure adapted from: C. Aggarwal. Linear Algebra and Optimization for Machine Learning. Springer, 2020.

Differential Curvature



$$E(\mathbf{w}) = w_1^2 + w_2^2$$



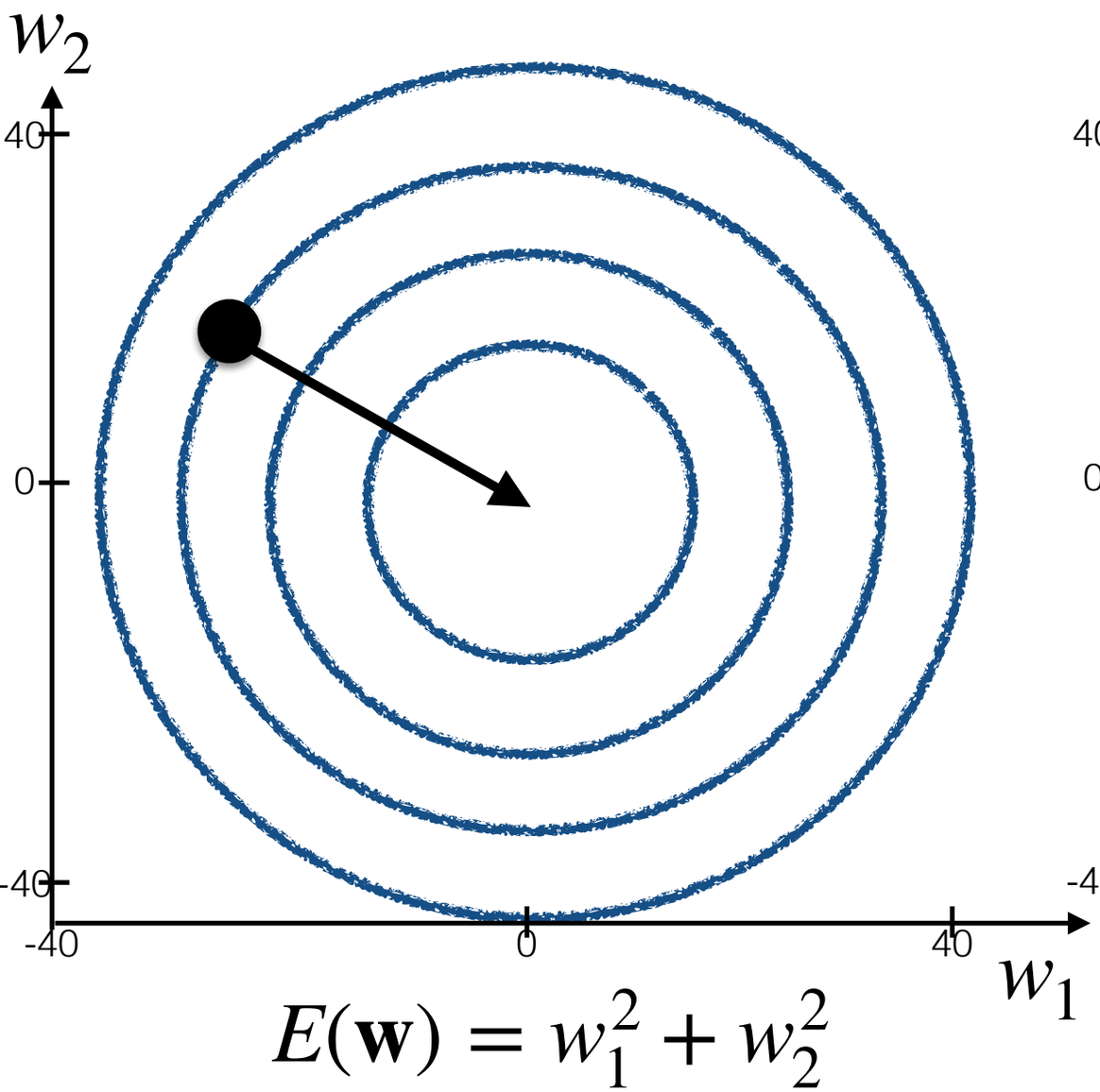
$$E(\mathbf{w}) = w_1^2 + 4w_2^2$$

In the elliptical bowl function, the gradients along the w_1 and w_2 axis have different magnitudes.

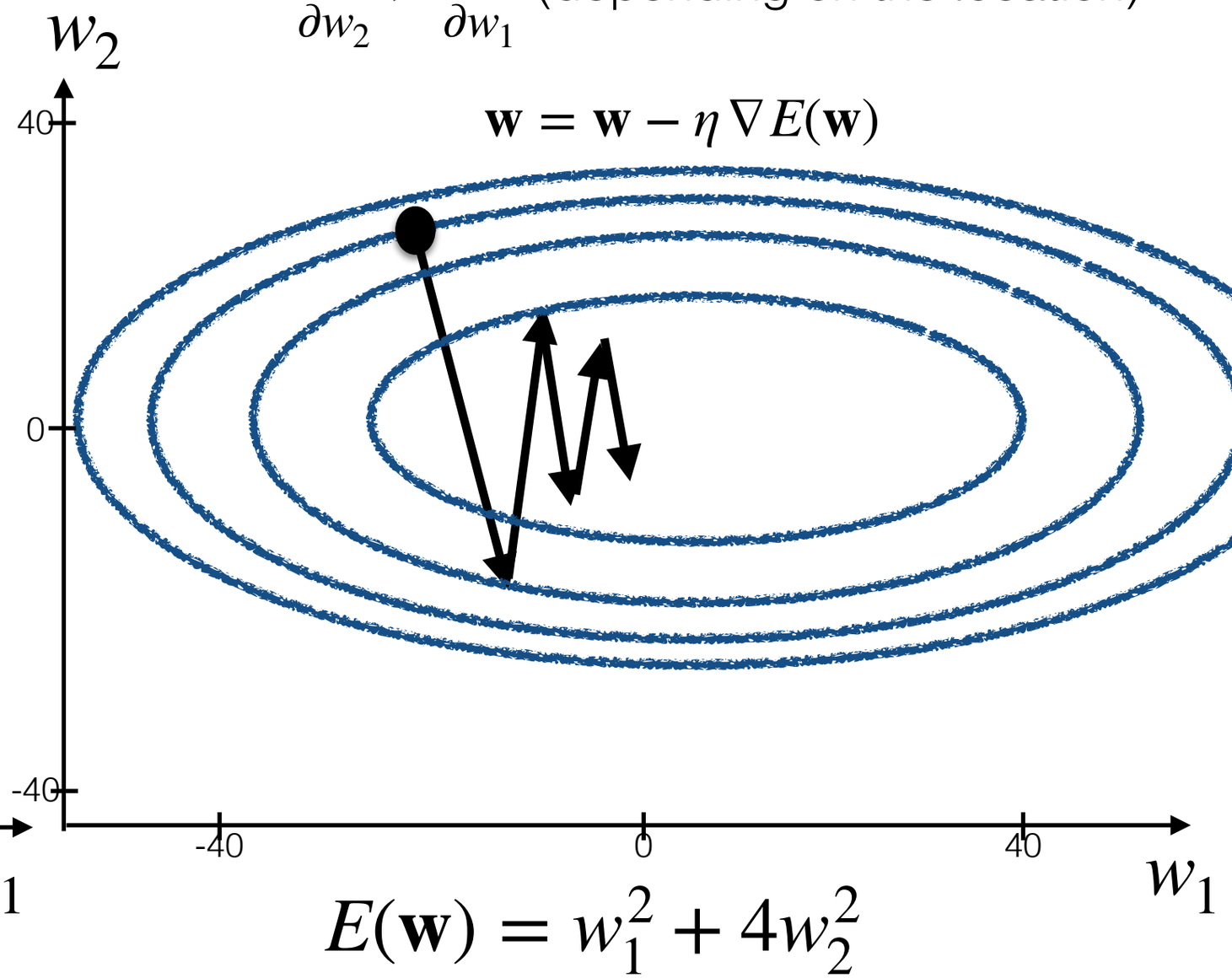
Figure adapted from: C. Aggarwal. Linear Algebra and Optimization for Machine Learning. Springer, 2020.

Differential Curvature

$\frac{\partial E}{\partial w_2} > \frac{\partial E}{\partial w_1}$ (depending on the location)



$E(\mathbf{w}) = w_1^2 + w_2^2$



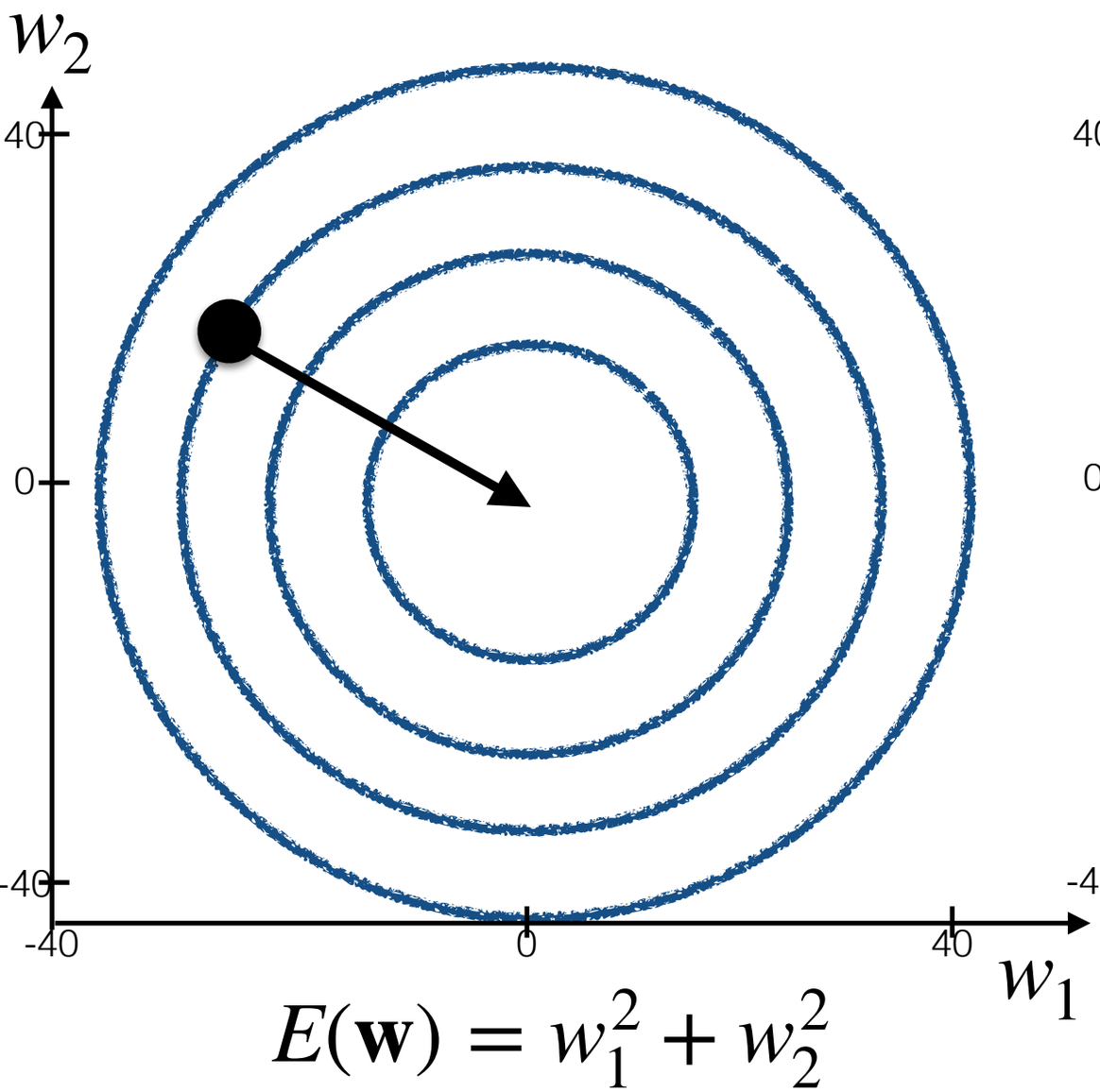
$\mathbf{w} = \mathbf{w} - \eta \nabla E(\mathbf{w})$

$E(\mathbf{w}) = w_1^2 + 4w_2^2$

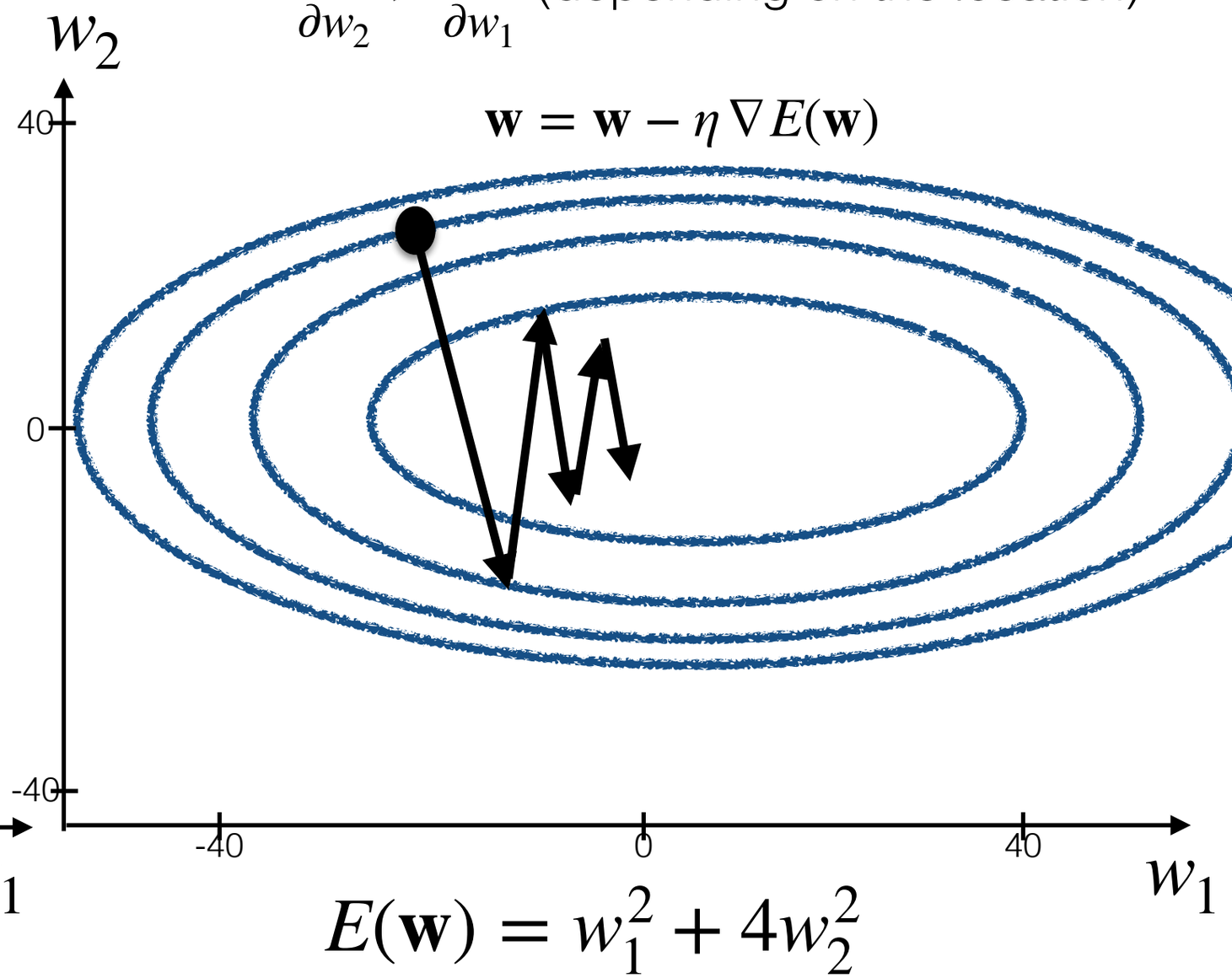
The path of the steepest descent in most loss functions is only an instantaneous direction of best movement, and is not the best direction in the longer term!

Differential Curvature

$\frac{\partial E}{\partial w_2} > \frac{\partial E}{\partial w_1}$ (depending on the location)



$E(\mathbf{w}) = w_1^2 + w_2^2$



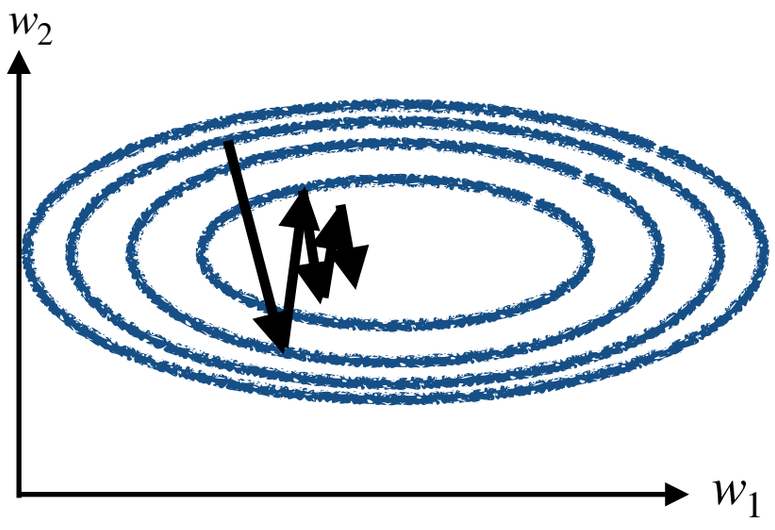
$E(\mathbf{w}) = w_1^2 + 4w_2^2$

Changes in w_1 are smaller. Changes in the w_2 are larger, but get undone in the next iteration. So, overall, the optimisation process is slow.

Figure adapted from: C. Aggarwal. Linear Algebra and Optimization for Machine Learning. Springer, 2020.

Standardisation

- Different partial derivatives with respect to different weights can be a result of different input variables having different scales and variances, affecting the loss function to different extents.

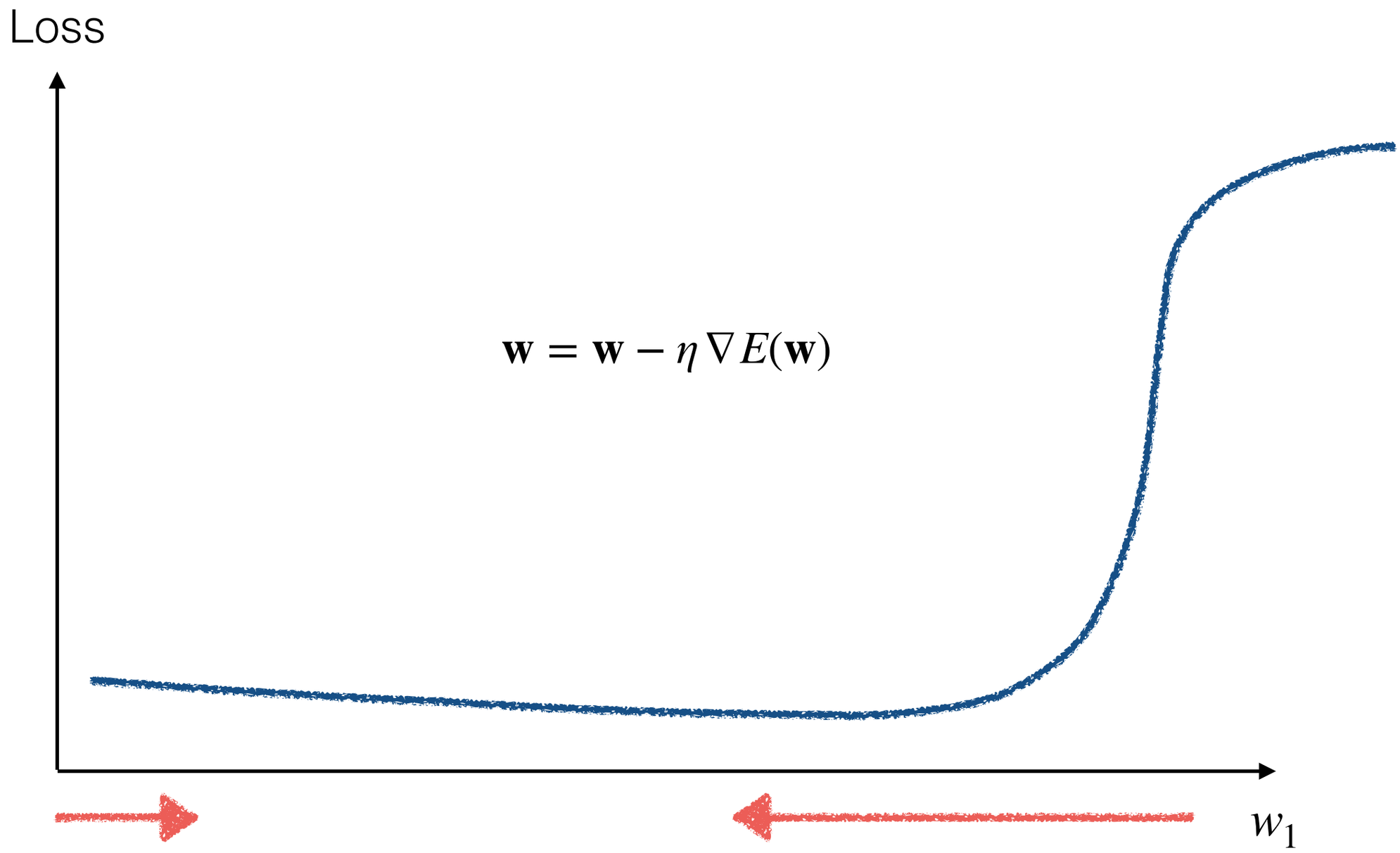


The larger effect of w_2 may be a result of the training examples having larger values for the feature x_2 than for x_1.

$$E(\mathbf{w}) = w_1^2 + 4w_2^2$$

- Standardising input variables (e.g., by deducting the mean from each input variable and then dividing by the standard deviation) can help with this.
- However, this may not be enough to solve the issue of overshooting along certain axis.

Difficult Topologies



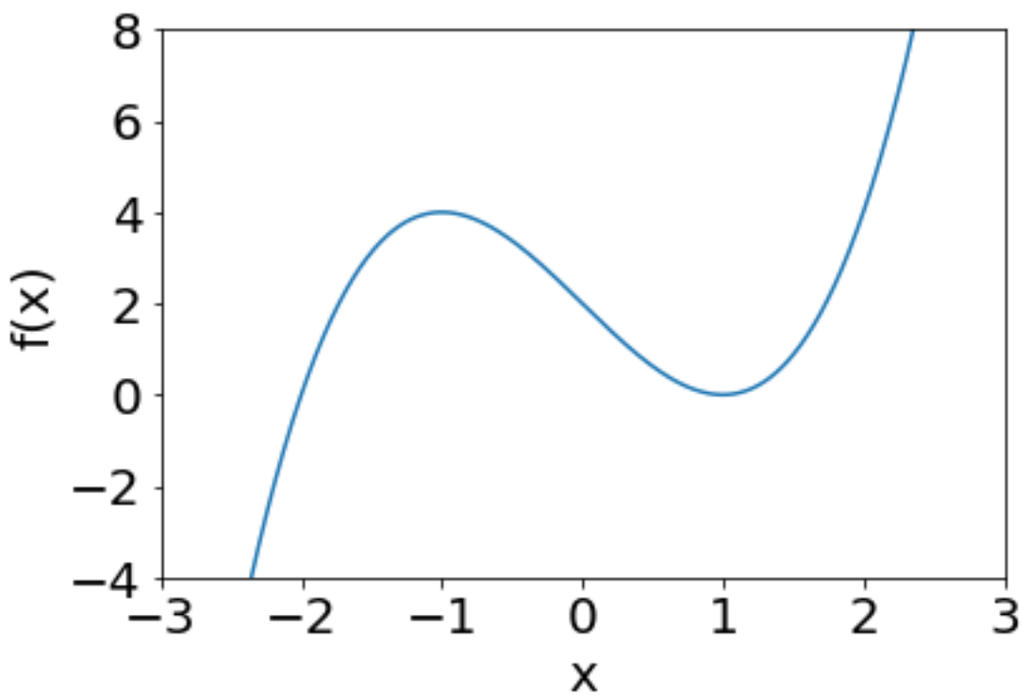
Smaller gradient, slow due to small steps

Big gradient, likely to overshoot

If we could make use of information about the curvature itself (i.e., about how the slope itself is changing), this may help to improve on this issue.

Second-Order Derivatives

$$\frac{d}{dx}\left(\frac{df}{dx}\right) = \frac{d^2}{dx^2}f(x) = \frac{d^2f}{dx^2} = f''(x) = f^{(2)}$$



Second derivative tells us what the curvature is (convex +, concave -).

Second derivative tells us whether the slope (gradient) is changing more or less rapidly, (larger magnitude f'', more rapid changes, higher curvature)

We can make use of second-order derivatives to tell if the gradient is changing too much, and make smaller weight updates accordingly!

Newton-Raphson Method:
Univariate Case for Illustration Purposes

Newton-Raphson: Univariate Weight Update Rule

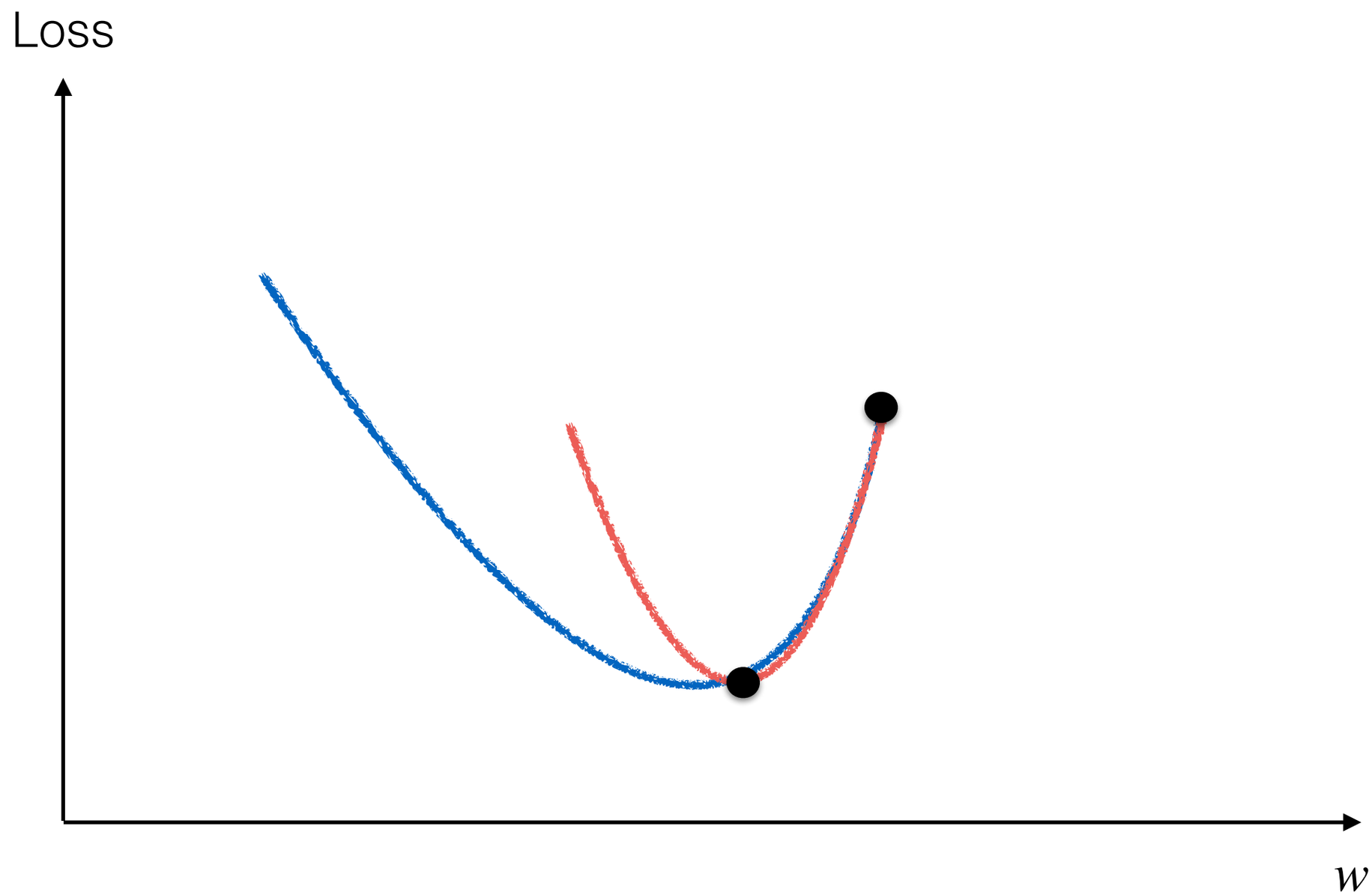
We still move in the opposite direction of
the gradient (if we are in a convex region)

$$w = w - \frac{E'(w)}{E''(w)}$$

But we will reduce the size of the
update if the curvature is high

PS: note the absence of a learning rate here.

Quadratic Approximation of the Loss



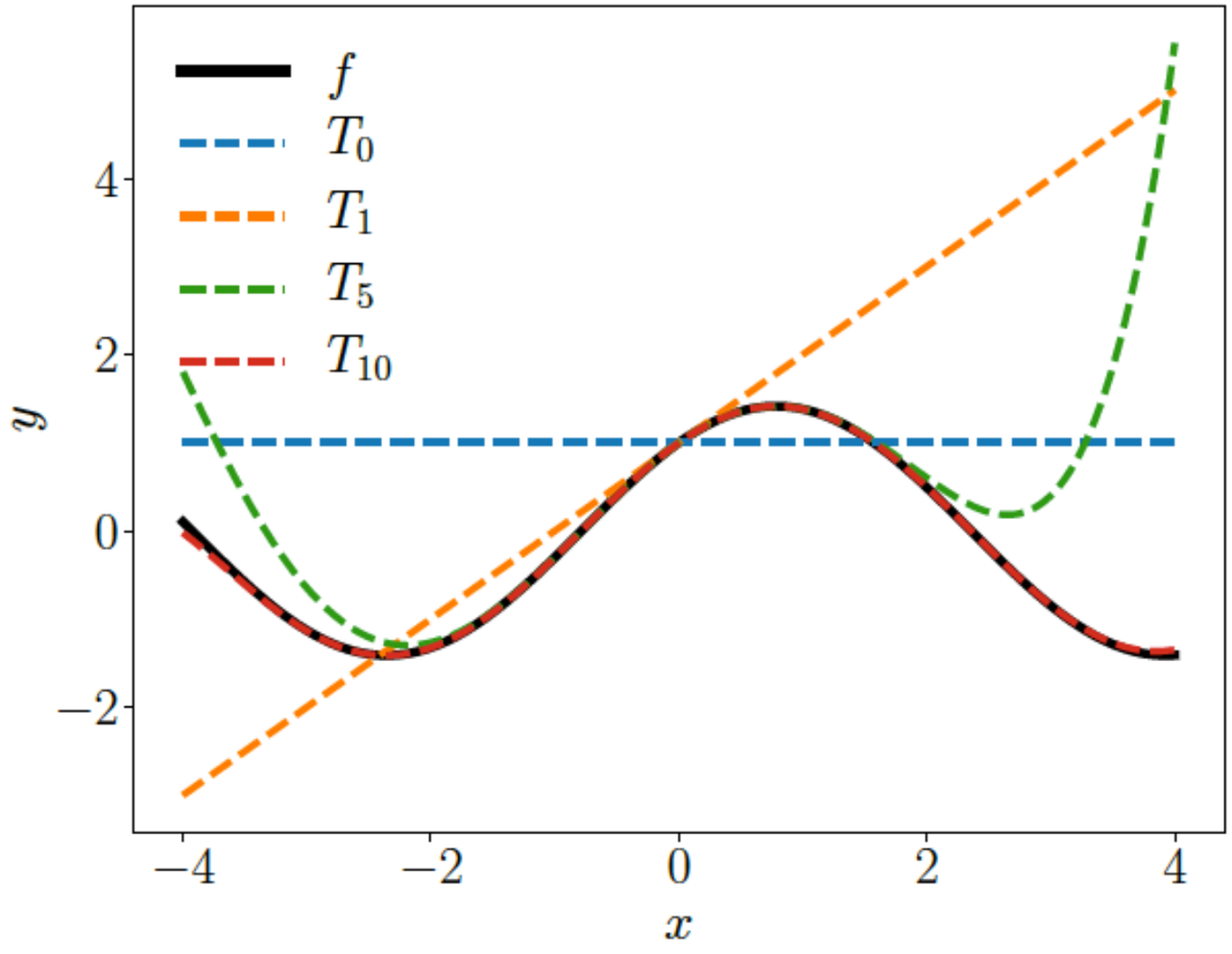
Using Taylor Polynomial for a Local Approximation of $E(w)$

- The Taylor polynomial of degree n can be used to approximate a function $E(w)$ at w_0:

$$T_n(w) = \sum_{k=0}^{n} \frac{E^{(k)}(w_0)}{k!}(w - w_0)^k$$

where $E^{(k)}(w_0)$ is the k-th order derivative of E at w_0.

Illustration of Taylor Polynomials



Approximation at $x = 0$.

PS: The approximation is better near $x = 0$.

$$f(x) = \sin(x) + \cos(x)$$

Figure from: M. P. Deisenroth, A. A. Faisal and C. S. Ong. Mathematics for Machine Learning. Cambridge University Press, 2020.

Using Taylor Polynomial for a Local Approximation of $E(w)$

- Taylor polynomial of degree 2 to approximate our loss function at w_0:

$$T_n(w) = \sum_{k=0}^{2} \frac{E^{(k)}(w_0)}{k!}(w - w_0)^k$$

where $E^{(k)}(w_0)$ is the k-th order derivative of E at w_0.

$$T_n(w) = \frac{E^{(0)}(w_0)}{0!}(w - w_0)^0 + \frac{E^{(1)}(w_0)}{1!}(w - w_0)^1 + \frac{E^{(2)}(w_0)}{2!}(w - w_0)^2$$

$$= E(w_0)(w - w_0)^0 + E'(w_0)(w - w_0)^1 + \frac{E''(w_0)}{2}(w - w_0)^2$$

$$= E(w_0) + (w - w_0)E'(w_0) + \frac{(w - w_0)^2}{2}E''(w_0)$$

Using Taylor Polynomial for a Local Approximation of $E(w)$

- Taylor polynomial of degree 2 to approximate a function at w_0:

$$T_n(w) = E(w_0) + (w - w_0)E'(w_0) + \frac{(w - w_0)^2}{2}E''(w_0)$$

- Let's consider that w_0 is the current value of the coefficient and w is the new value after the adjustment, i.e., we are approximating $E(w)$ around the current value of the coefficient.

- We want to find the new value w that leads to the minimum (or maximum) of this quadratic approximation, in an attempt to move closer to the minimum (or maximum) of the original function.

Weight Update Rule

- To find the step change that takes us to the minimum (or maximum) of this quadratic function, we need to solve the following for w:

$$\frac{d}{dw}\left(E(w_0) + (w - w_0)E'(w_0) + \frac{(w - w_0)^2}{2}E''(w_0)\right) = 0$$

$$E'(w_0) + (w - w_0)E''(w_0) = 0 \qquad (w - w_0) = -\frac{E'(w_0)}{E''(w_0)}$$

$$(w - w_0)E''(w_0) = -E'(w_0)$$

- This will lead to the following weight update rule:

$$w = w_0 - \frac{E'(w_0)}{E''(w_0)} \quad \rightarrow \quad w = w - \frac{E'(w)}{E''(w)}$$

Newton-Raphson Method: Multivariate Case

Second Order Partial Derivatives and The Hessian

$$\frac{\partial}{\partial x_i}\left(\frac{\partial f}{\partial x_i}\right) = \frac{\partial^2 f}{\partial x_i^2}, \quad \frac{\partial}{\partial x_i}\left(\frac{\partial f}{\partial x_j}\right) = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

$$H(f(\mathbf{x})) = H_f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_0^2}, & \frac{\partial^2 f}{\partial x_0 \partial x_1}, & \cdots, & \frac{\partial^2 f}{\partial x_0 \partial x_d} \\ \frac{\partial^2 f}{\partial x_1 \partial x_0}, & \frac{\partial^2 f}{\partial x_1^2}, & \cdots, & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_0}, & \frac{\partial^2 f}{\partial x_d \partial x_1}, & \cdots, & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix}$$

Second-order partial derivative is the partial derivative of the partial derivative of $f(\mathbf{x})$, i.e., tells the rate of change of the partial derivative.

Weight Update Rule

- Univariate update rule:

$$w = w - \frac{E'(w)}{E''(w)}$$

- Multivariate update rule:

$$\mathbf{w} = \mathbf{w} - H_E^{-1}(\mathbf{w})\,\nabla E(\mathbf{w})$$

where $H_E^{-1}(\mathbf{w})$ is the inverse of the Hessian at the old $\mathbf{w}$ and $\nabla E(\mathbf{w})$ is the gradient at the old $\mathbf{w}$.

The Effect of the Newton-Raphson Weight Update Rule on a Quadratic Function

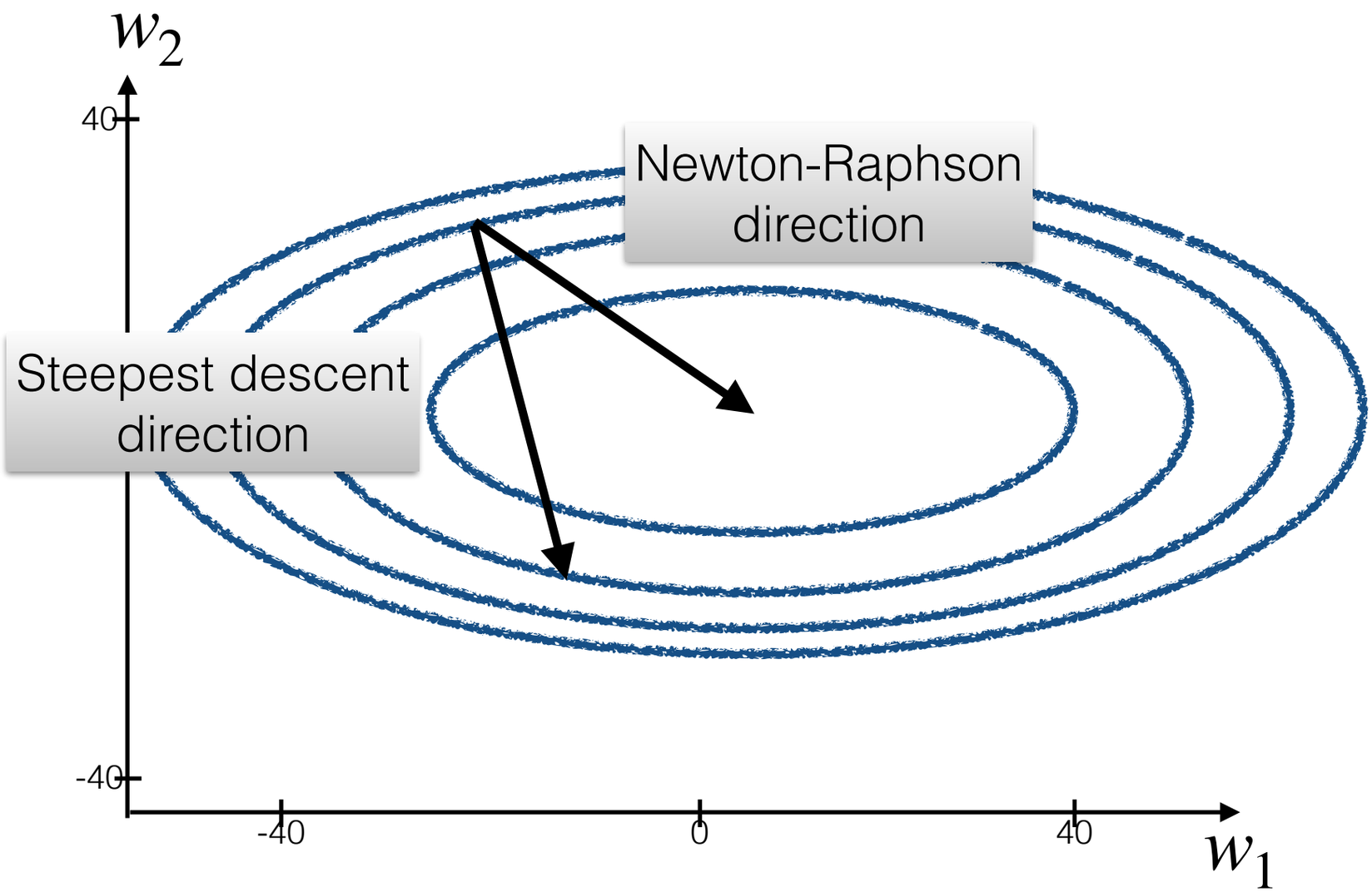
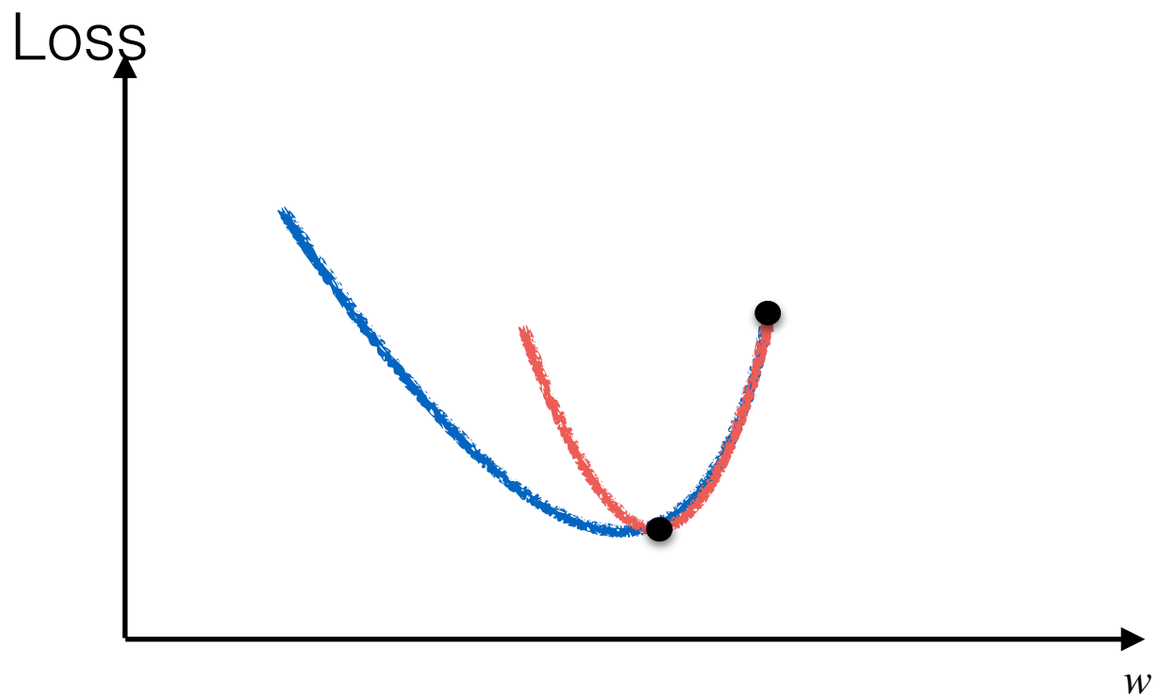


Figure adapted from: C. Aggarwal. Linear Algebra and Optimization for Machine Learning. Springer, 2020.

Weight Update Rule for Non-Quadratic Loss Functions

$$\mathbf{w} = \mathbf{w} - H_E^{-1}(\mathbf{w}) \nabla E(\mathbf{w})$$



- This update will take us to the optimal of the quadratic approximation in a single step.

- However, as the quadratic approximation is not the true loss function, we will need to apply this rule iteratively.

For Logistic Regression — Iterative Reweighted Least Squares

$$\mathbf{w} = \mathbf{w} - H_E^{-1}(\mathbf{w})\,\nabla E(\mathbf{w})$$

$$H_E(\mathbf{w}) = \sum_{i=1}^{N} p(1 \mid \mathbf{x}^{(i)}, \mathbf{w})(1 - p(1 \mid \mathbf{x}^{(i)}, \mathbf{w}))\mathbf{x}^{(i)}\mathbf{x}^{(i)^T}$$

$$\nabla_E(\mathbf{w}) = \sum_{i=1}^{N} (p(1 \mid \mathbf{x}^{(i)}, \mathbf{w}) - y^{(i)})\mathbf{x}^{(i)}$$

Note the dependence of the gradient and Hessian on $\mathbf{w}$.

When we update $\mathbf{w}$, the values of the gradient and Hessian will change, requiring us to iteratively update $\mathbf{w}$ again.

For Logistic Regression — Iterative Reweighted Least Squares

$$\mathbf{w} = \mathbf{w} - H_E^{-1}(\mathbf{w})\,\nabla E(\mathbf{w})$$

$$H_E(\mathbf{w}) = \sum_{i=1}^{N} p(1 \mid \mathbf{x}^{(i)}, \mathbf{w})(1 - p(1 \mid \mathbf{x}^{(i)}, \mathbf{w}))\mathbf{x}^{(i)}\mathbf{x}^{(i)^T}$$

$$\nabla_E(\mathbf{w}) = \sum_{i=1}^{N} (p(1 \mid \mathbf{x}^{(i)}, \mathbf{w}) - y^{(i)})\mathbf{x}^{(i)}$$

The Newton-Raphson update rule comes from a quadratic approximation of the loss function, which is not quadratic. However, its deviations from quadratic are not too large.

To avoid problems with poor quadratic approximations, it is also possible to adopt a learning rate for Newton Raphson.

Summary

- Gradient descent may require a large number of iterations depending on the shape of the loss function.
- Iterative Reweighted Least Squares / Newton-Raphson can be used in an attempt to reach the minimum with less steps.
- It does so by creating a quadratic approximation of the loss function and finding the minimum of this quadratic approximation.
- This results in making use of information about the curvature of the loss function to:
  - avoid large steps in directions where the gradient is changing too much or
  - increase the size of the steps in directions where the gradient is not changing much.

Tutorial Poll

Please fill in by Tuesday 1pm!

Further Reading

- Essential:
  - Aggarwal's book on "Linear Algebra and Optimization for Machine Learning", Section 5.2.2 (Differential Curvature), 5.2.3 (Examples of Difficult Topologies) and 5.4.1 (The Basic Form of the Newton Method).
- Recommended:
  - Bishop's book on "Machine Learning and Pattern Recognition", Section 4.3.3 (Iterative Reweighted Least Squares).
- Optional:
  - Aggarwal's book on "Linear Algebra and Optimization for Machine Learning", Section 4.4.3 (Line Search), Section 5.3.1 (Momentum-Based Learning).