Machine Learning

CMPT 726

Mo Chen
SFU School of Computing Science

2022-10-18

Nonlinear Regression and Optimization

Nonlinear Least Squares

Recall: By replacing raw data with features, we can make the prediction depend *non-linearly* on the raw data.
$\hat{y} = \vec{w}^\top \phi(\vec{x})$

However, the prediction $\hat{y}$ still depends linearly on the parameters $\vec{w}$.

Can we make the prediction depend *non-linearly* on the parameters?

E.g.: $\hat{y} = (\vec{w}^\top \vec{x})^2$

Nonlinear Least Squares

Suppose we have the following data generating process:

$y = f(\vec{x}; \vec{w}) + \sigma\epsilon$, where $\epsilon \sim \mathcal{N}(0,1)$

So, $y | \vec{x}, \vec{w}, \sigma \sim \mathcal{N}(f(\vec{x}; \vec{w}), \sigma^2)$

Hence the likelihood function is:

$$\mathcal{L}(\vec{w}, \sigma; \mathcal{D}) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - f(\vec{x}_i; \vec{w}))^2}{2\sigma^2}\right)$$

$$\hat{\vec{w}}_{\text{MLE}} = \arg\max_{\vec{w}} \log \mathcal{L}(\vec{w}, \sigma; \mathcal{D}) = \arg\min_{\vec{w}} \sum_{i=1}^{N} \big(y_i - f(\vec{x}_i; \vec{w})\big)^2$$

How do we find the optimal parameters $\hat{\vec{w}}_{\text{MLE}}$?

Nonlinear Least Squares

To summarize, nonlinear least squares requires solving the following:

$$\widehat{\vec{w}}_{\mathrm{MLE}} = \arg\max_{\vec{w}} \log \mathcal{L}(\vec{w}, \sigma; \mathcal{D}) = \arg\min_{\vec{w}} \sum_{i=1}^{N} \left(y_i - f(\vec{x}_i; \vec{w})\right)^2$$

This can be viewed as solving an instance of the following more general problem:

$$\vec{\theta}^* = \arg\min_{\vec{\theta}} L\left(\vec{\theta}\right)$$

L is known as the "objective function"

$L\left(\vec{\theta}\right)$ is known as the "objective value"

In this case, $\vec{\theta} = \vec{w}$ and $L(\vec{\theta}) = -\log \mathcal{L}(\vec{\theta}, \sigma; \mathcal{D})$.

Such a problem is known as an **optimization problem**, or more specifically, an unconstrained minimization problem.

Nonlinear Least Squares

Goal: Find $\vec{\theta}^* = \arg\min_{\vec{\theta}} L(\vec{\theta})$

The first step to finding a global minimum is to find a critical point. We can try to find the critical points by setting the gradient to zero:

$$\frac{\partial}{\partial \vec{w}} \left(\sum_{i=1}^{N} \left(y_i - f(\vec{x}_i; \vec{w})\right)^2 \right) = \sum_{i=1}^{N} \frac{\partial}{\partial \vec{w}} \left(y_i - f(\vec{x}_i; \vec{w})\right)^2$$

$$= -\sum_{i=1}^{N} 2 \left(y_i - f\left(\vec{x}_i; \vec{w}\right)\right) \frac{\partial f}{\partial \vec{w}} (\vec{x}_i; \vec{w}) = \vec{0}$$

Cannot solve this in closed form in general because the form of $\frac{\partial f}{\partial \vec{w}}$ could be arbitrary.

Iterative Optimization Algorithms

Goal: Find $\vec{\theta}^* = \arg\min_{\vec{\theta}} L(\vec{\theta})$

Because we cannot in general solve for the critical point analytically, we find it numerically using an iterative optimization algorithm.

The simplest of these algorithms is **gradient descent**.

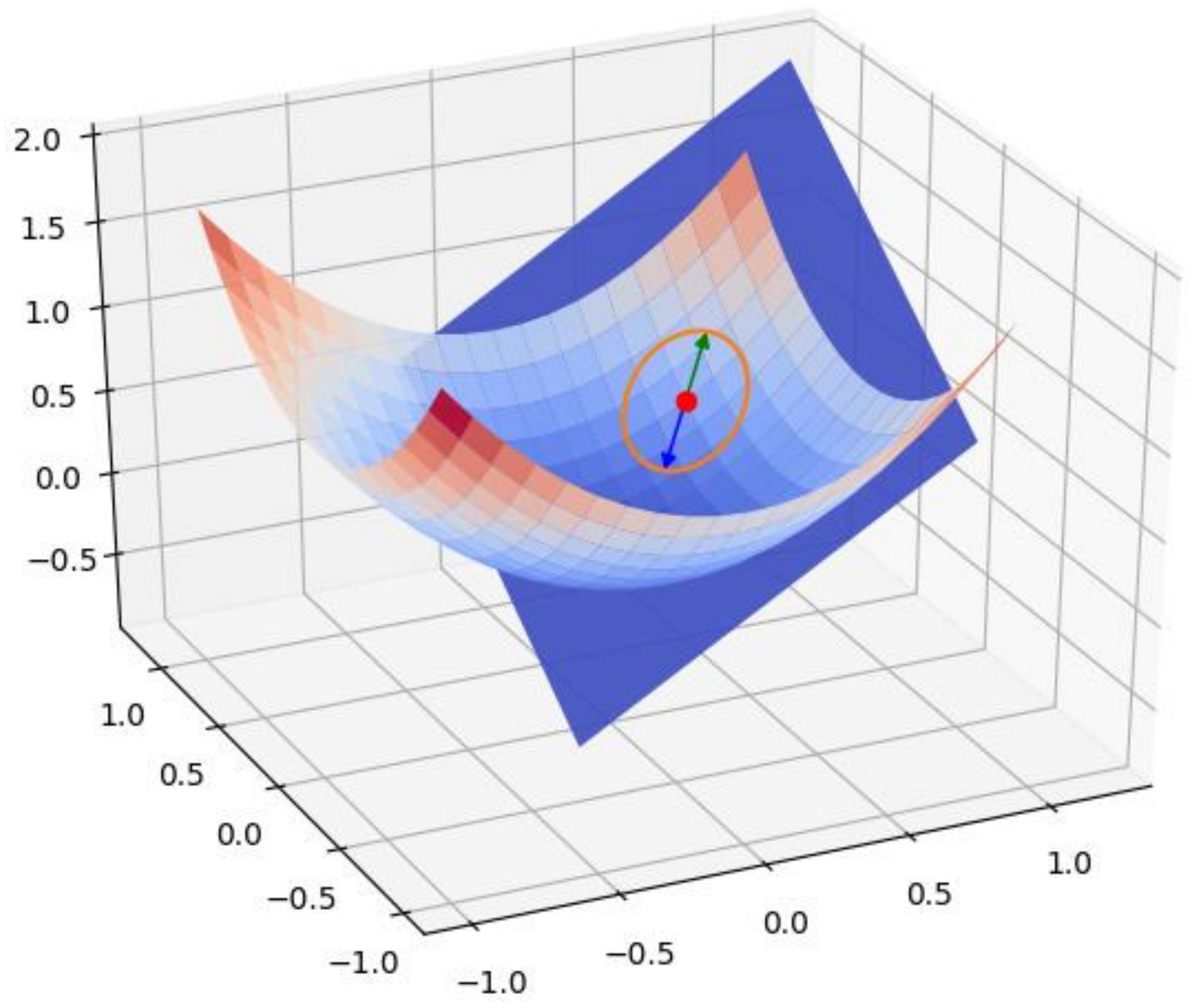
```
$\vec{\theta}^{(0)} \leftarrow \text{random vector}$
$\text{for } t = 1,2,3,\ldots$
    $\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} - \gamma_t \frac{\partial L}{\partial \vec{\theta}}\left(\vec{\theta}^{(t-1)}\right)$
    $\text{if algorithm has converged}$
        $\text{return } \vec{\theta}^{(t)}$
```

γ_t is a hyperparameter and is known as the “step size” or “learning rate”

Iterative Optimization Algorithms

Gradient Descent:

```
$\overrightarrow{\theta}^{(0)} \leftarrow$ random vector
for $t = 1,2,3,\ldots$
    $\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} - \gamma_t \frac{\partial L}{\partial \vec{\theta}}\left(\vec{\theta}^{(t-1)}\right)$
    if algorithm has converged
        return $\overrightarrow{\theta}^{(t)}$
```

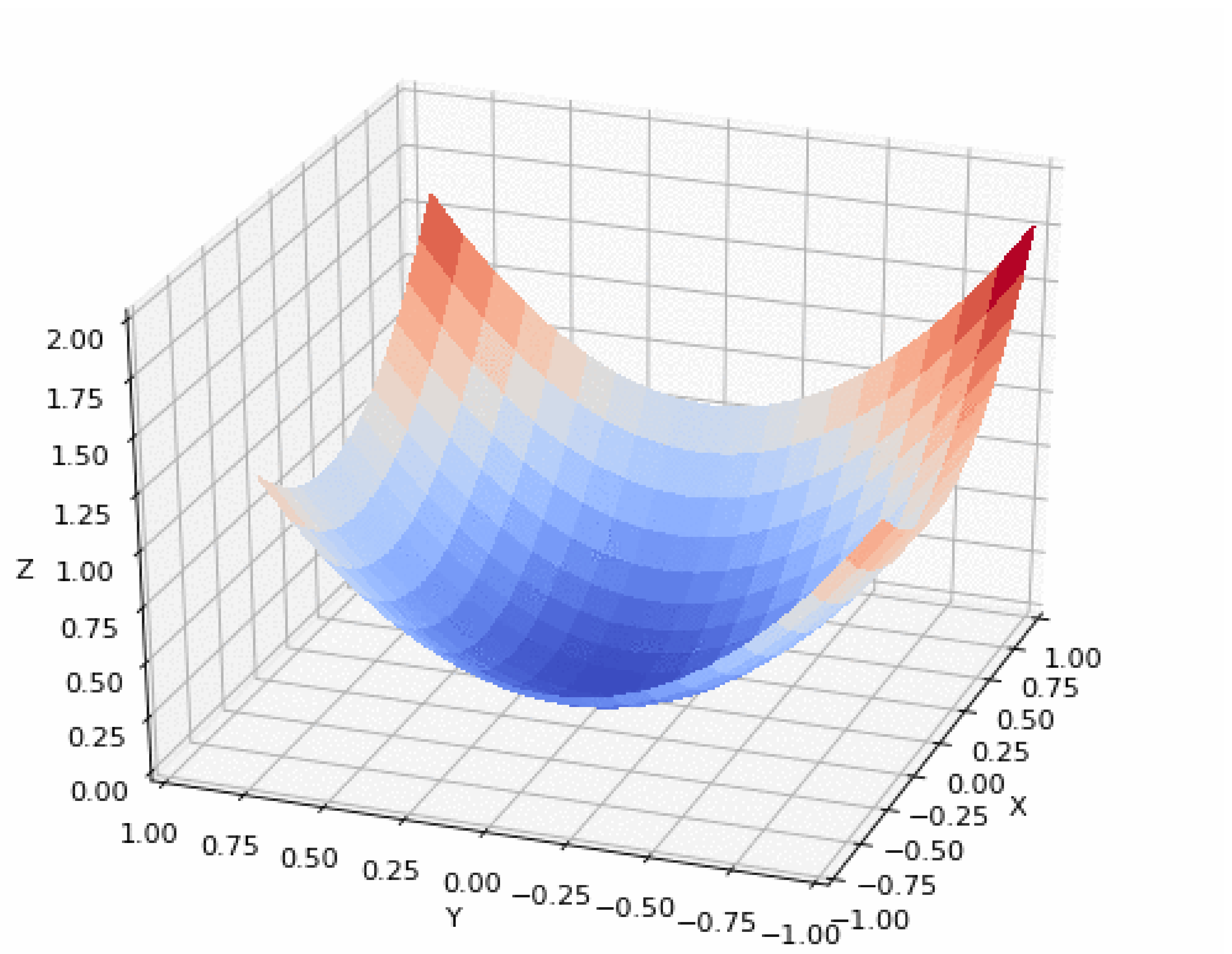


Credit: Pierre Vigier

Iterative Optimization Algorithms

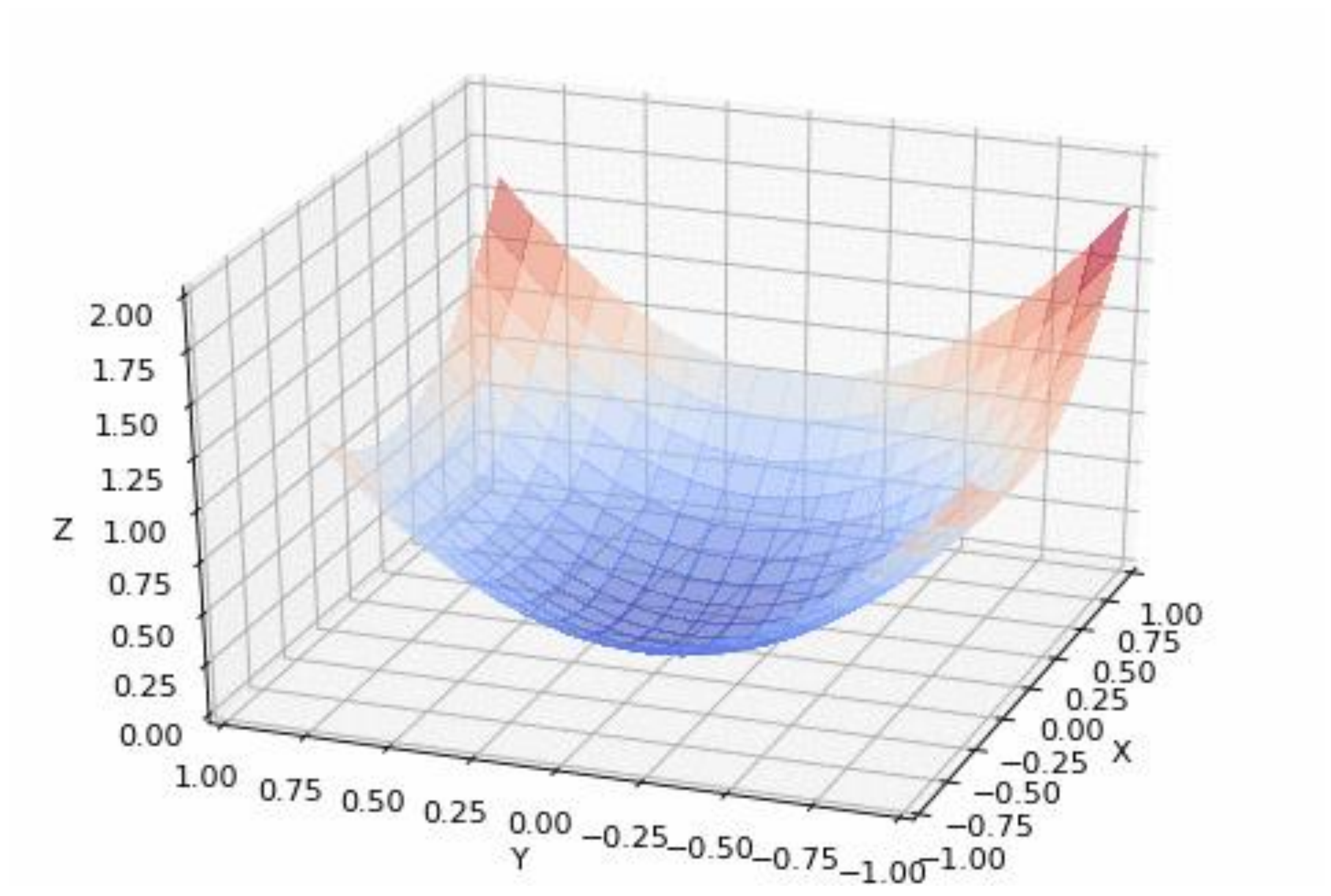
Gradient Descent:

```
$\overrightarrow{\theta}^{(0)} \leftarrow$ random vector
for $t = 1,2,3,\ldots$
    $\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} - \gamma_t \frac{\partial L}{\partial \vec{\theta}}\left(\vec{\theta}^{(t-1)}\right)$
    if algorithm has converged
        return $\overrightarrow{\theta}^{(t)}$
```

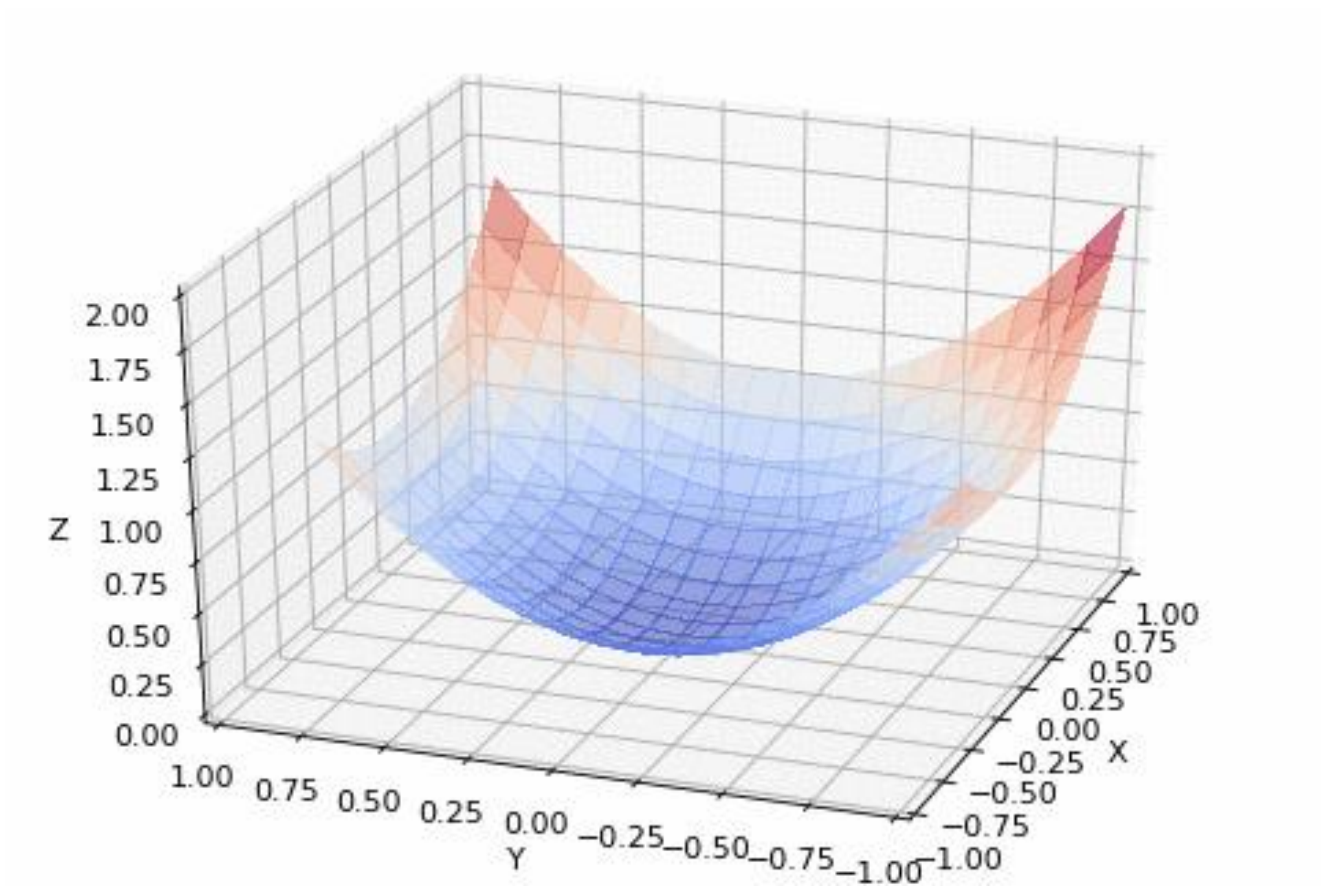


Credit: Pierre Vigier

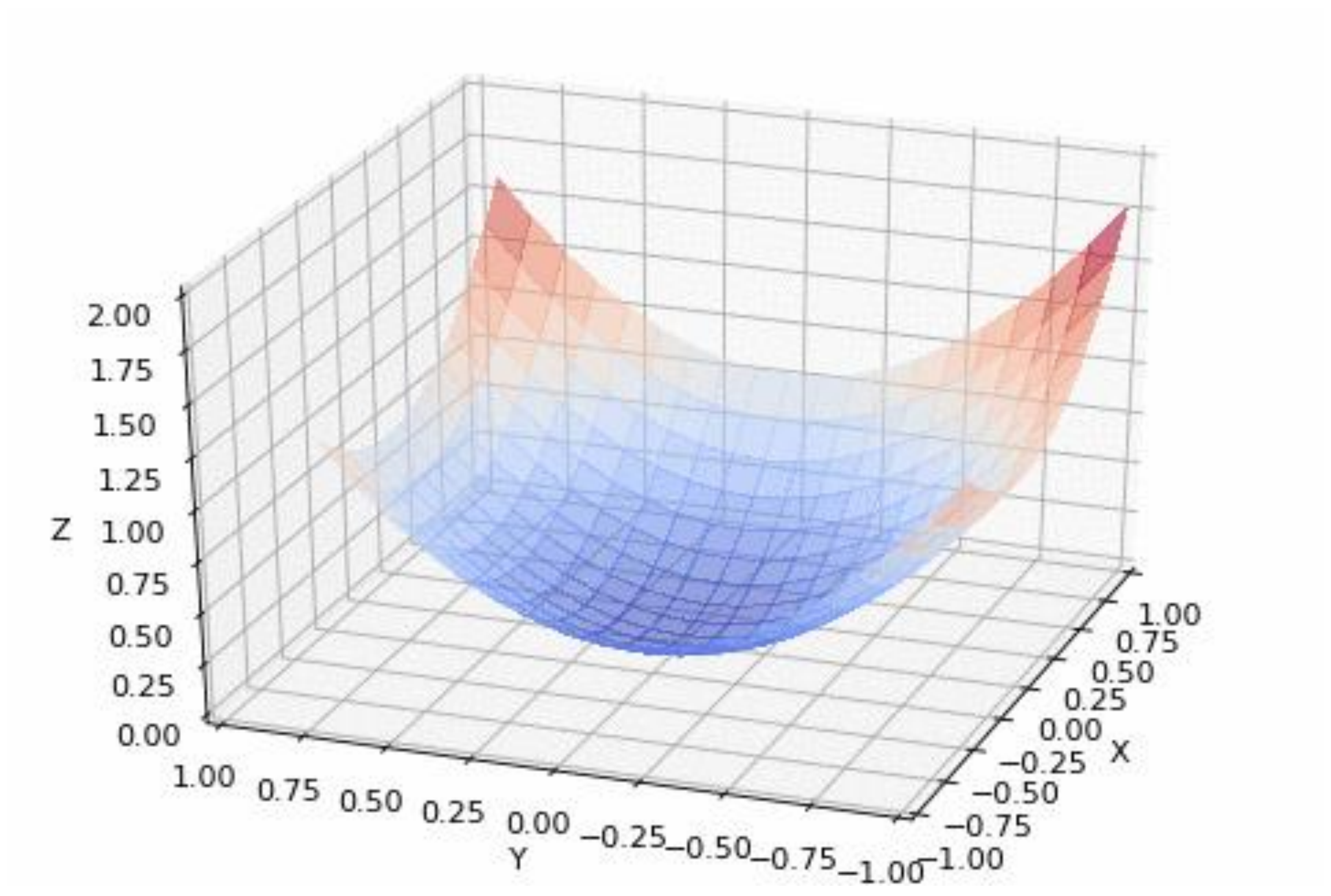
Effect of Step Size



Step size too large



Step size just right



Step size too small

Convergence

What does it mean for the algorithm to converge?

Two different notions of convergence:

- Convergence to a target objective value: as the iteration number increases, the objective value gets closer to the target objective value (e.g.: the minimum objective value)

- Convergence to a target parameter vector: as the iteration number increases, the parameter vector gets closer to the target parameter vector (e.g.: a global minimum, a local minimum or a critical point)

Typically, we don’t know the target parameter vector or objective value.

- To detect convergence (which can be used to determine when to stop the optimization algorithm), we can check if the change in parameters or objective value from the previous iteration is less than a threshold.

Convergence of Gradient Descent

Roughly speaking, if the objective function is **convex** and ***c*-Lipschitz**, gradient descent with a sufficiently small step size is guaranteed to converge to the minimal objective value.

The gap between the minimal objective value and the objective value at iteration t is at most $\Theta\left(\frac{1}{\sqrt{t}}\right)$.

"Convergence Rate"

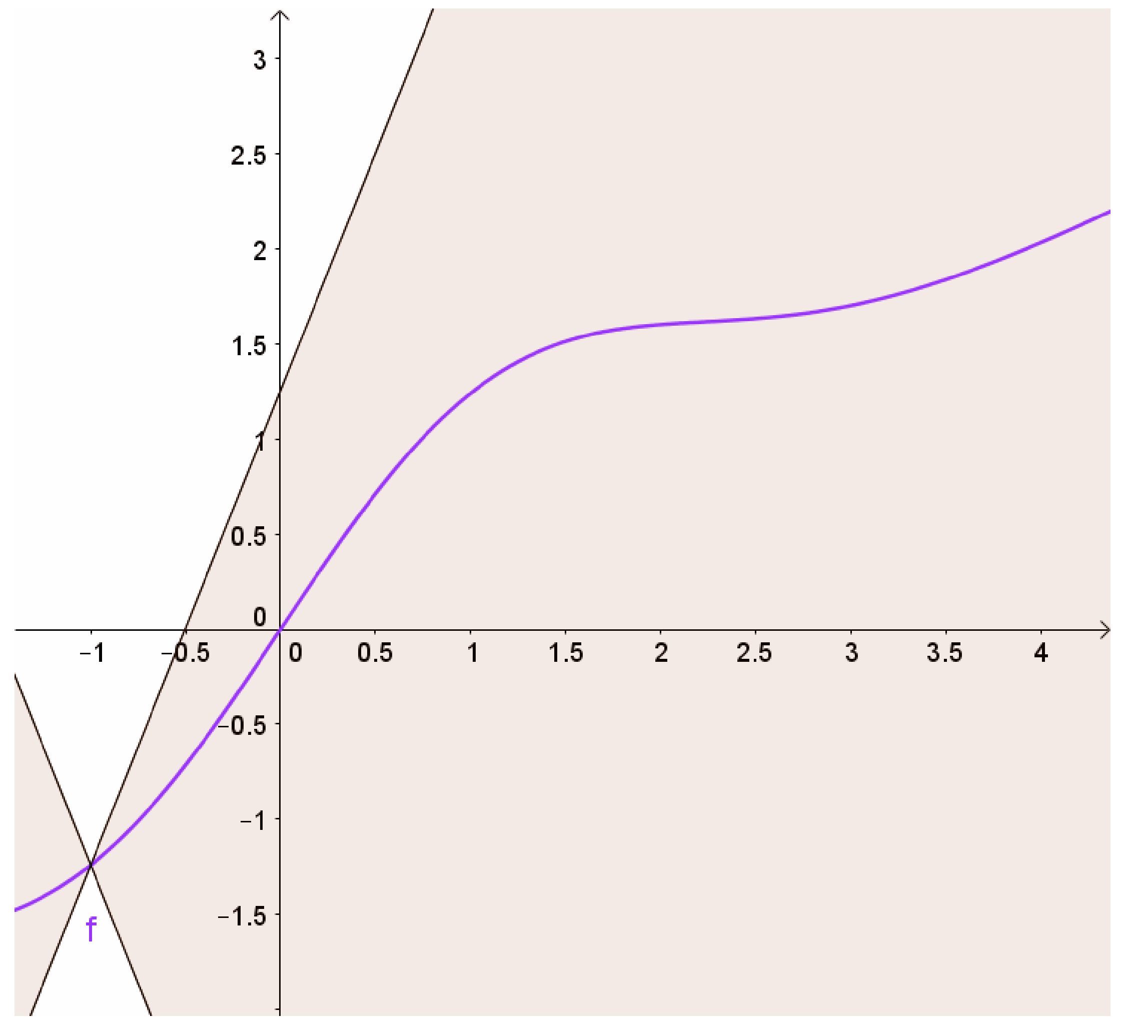
Equivalently, to get to a parameter vector whose objective value that is ϵ larger than the **minimal objective value**, need $\Theta\left(\frac{1}{\epsilon^2}\right)$ iterations.

Recall: A function $L: \mathbb{R}^n \to \mathbb{R}$ is c-Lipschitz if for all $\vec{x}_1, \vec{x}_2$,

$$|L(\vec{x}_1) - L(\vec{x}_2)| \le c\|\vec{x}_1 - \vec{x}_2\|_2$$

An everywhere differentiable function is c-Lipschitz if and only if

$$\left\|\frac{\partial}{\partial \vec{x}} L(\vec{x})\right\|_2 \le c$$



Convergence of Gradient Descent

Roughly speaking, if the objective function is **convex** and **c-Lipschitz**, gradient descent with a sufficiently small step size is guaranteed to converge to the minimal objective value.

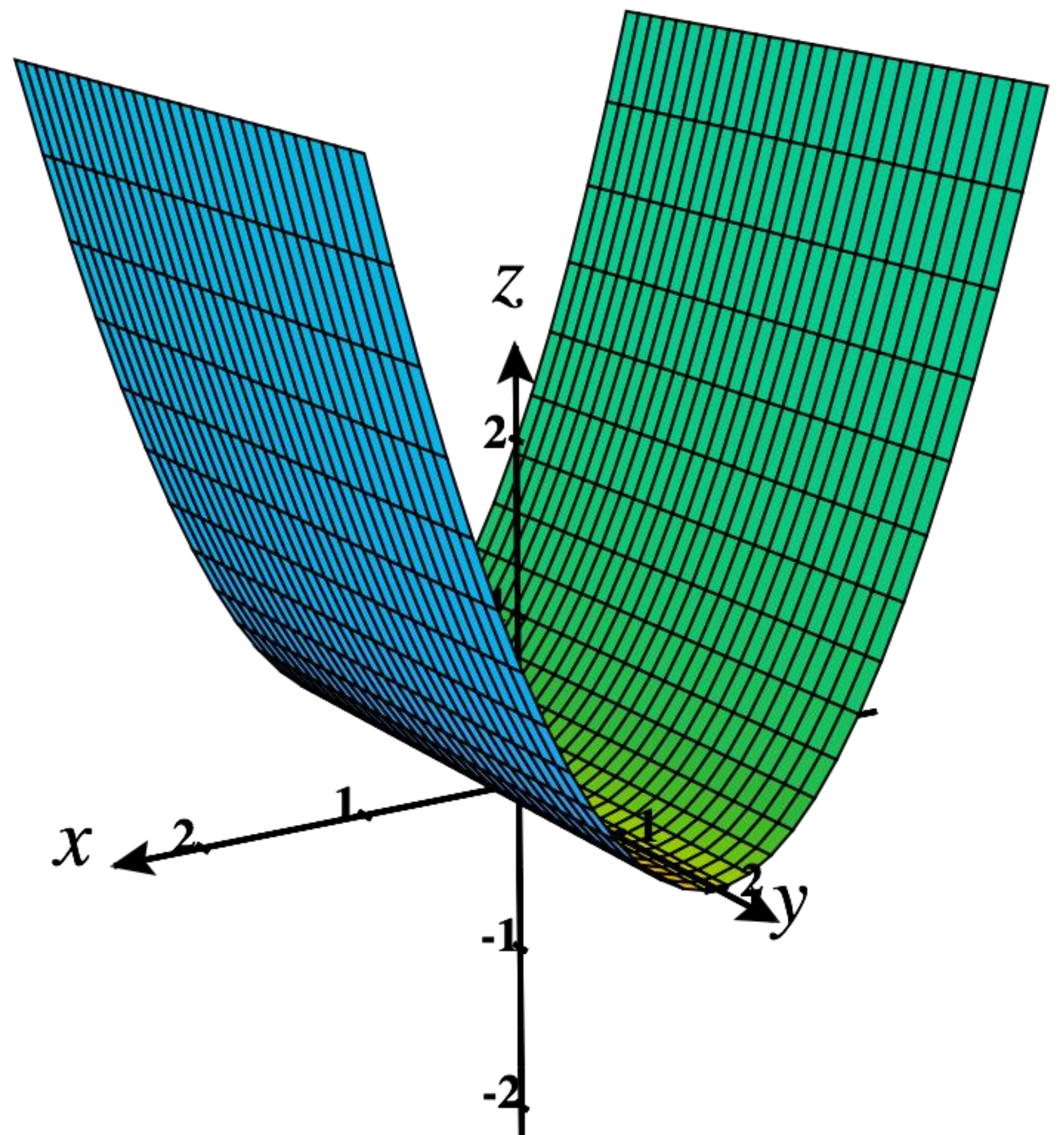
The gap between the minimal objective value and the objective value at iteration t is at most $\Theta\left(\frac{1}{\sqrt{t}}\right)$.

"Convergence Rate"

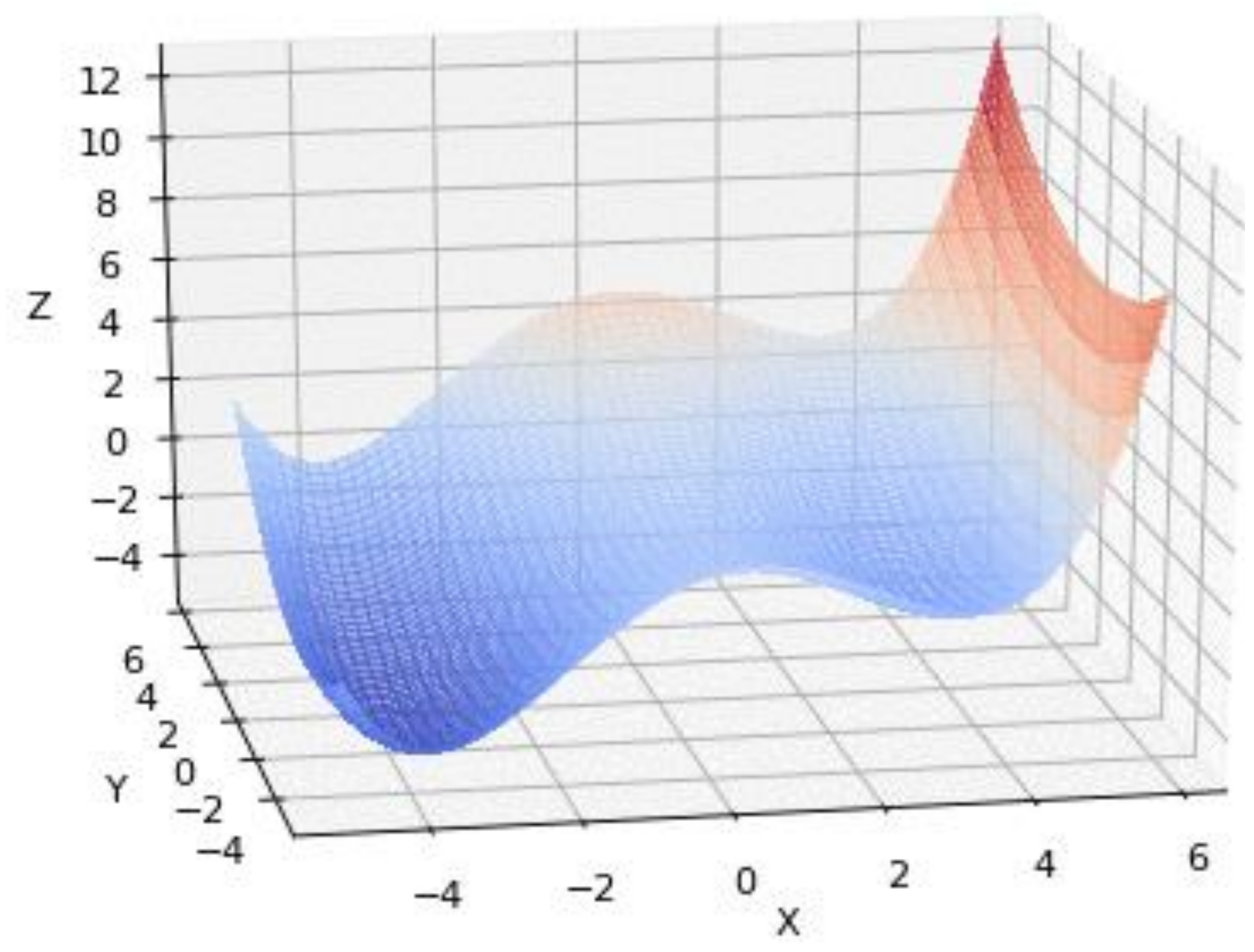
Equivalently, to get to a parameter vector whose objective value that is ϵ larger than the **minimal objective value**, need $\Theta\left(\frac{1}{\epsilon^2}\right)$ iterations.

Recall: A function $L: \mathbb{R}^n \to \mathbb{R}$ is convex if a line segment between any two points on the surface lies on or above the surface.

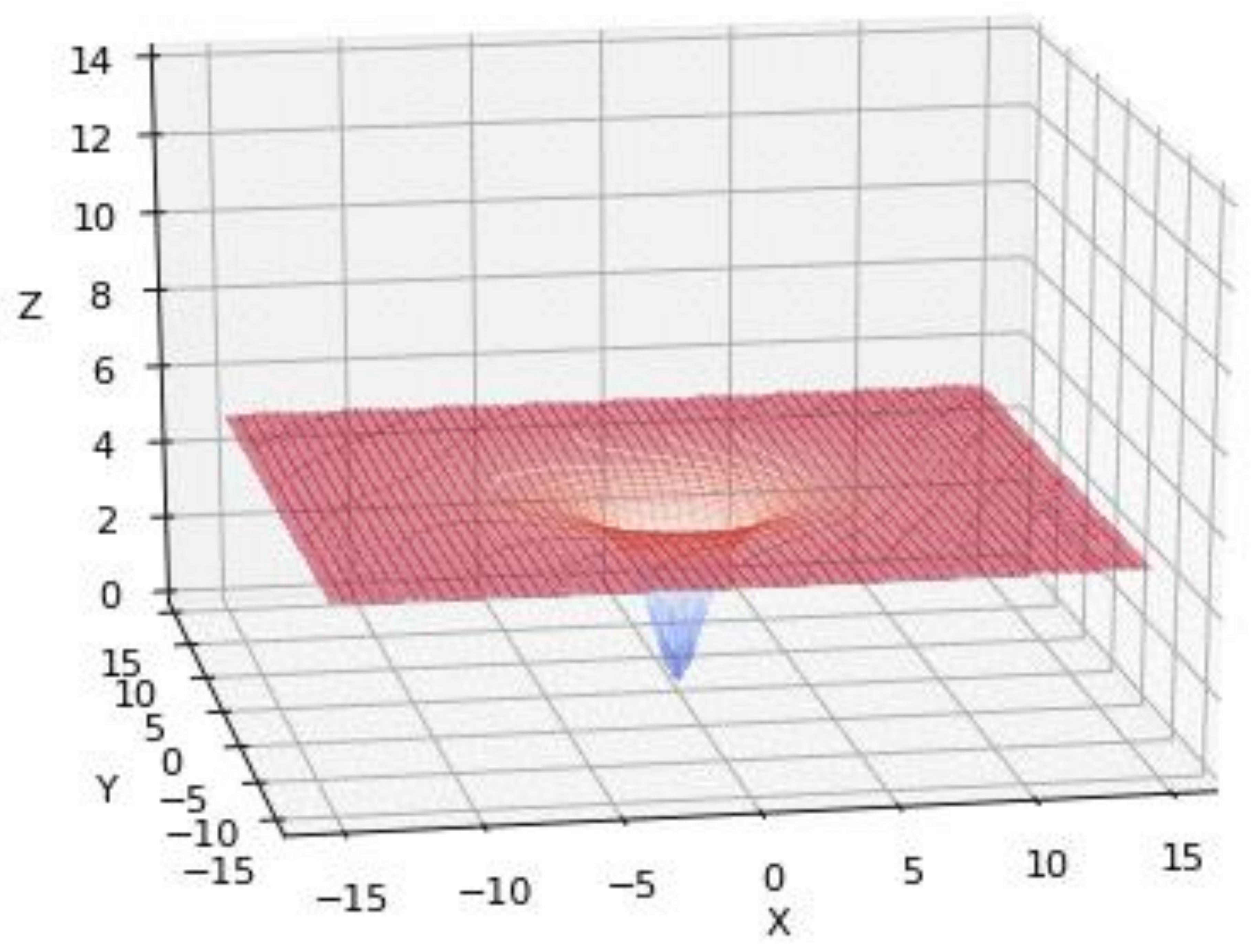
An everywhere twice differentiable function is convex if and only if the Hessian is positive semi-definite.



What Happens on Non-Convex Functions?



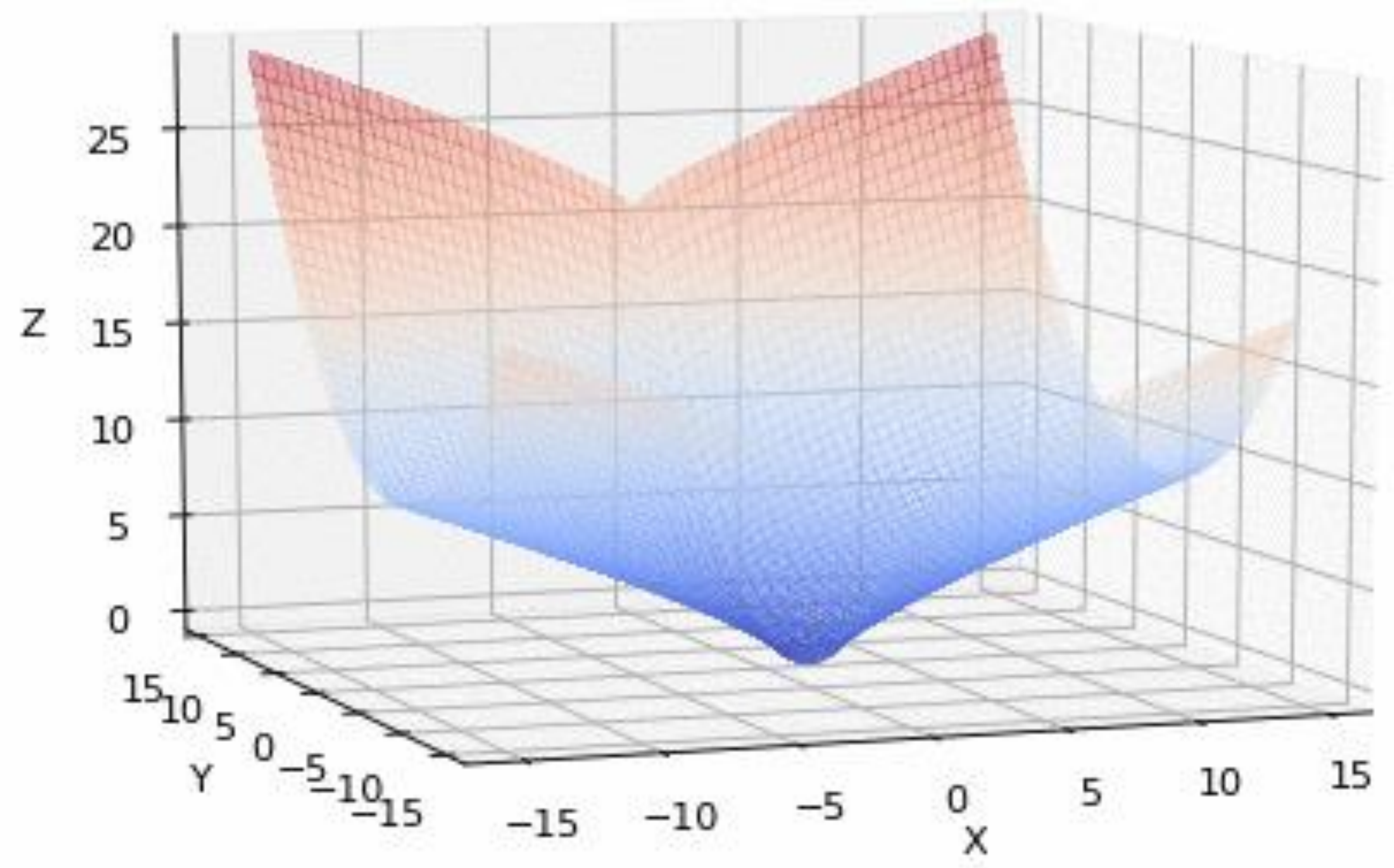
Gets stuck in local minimum

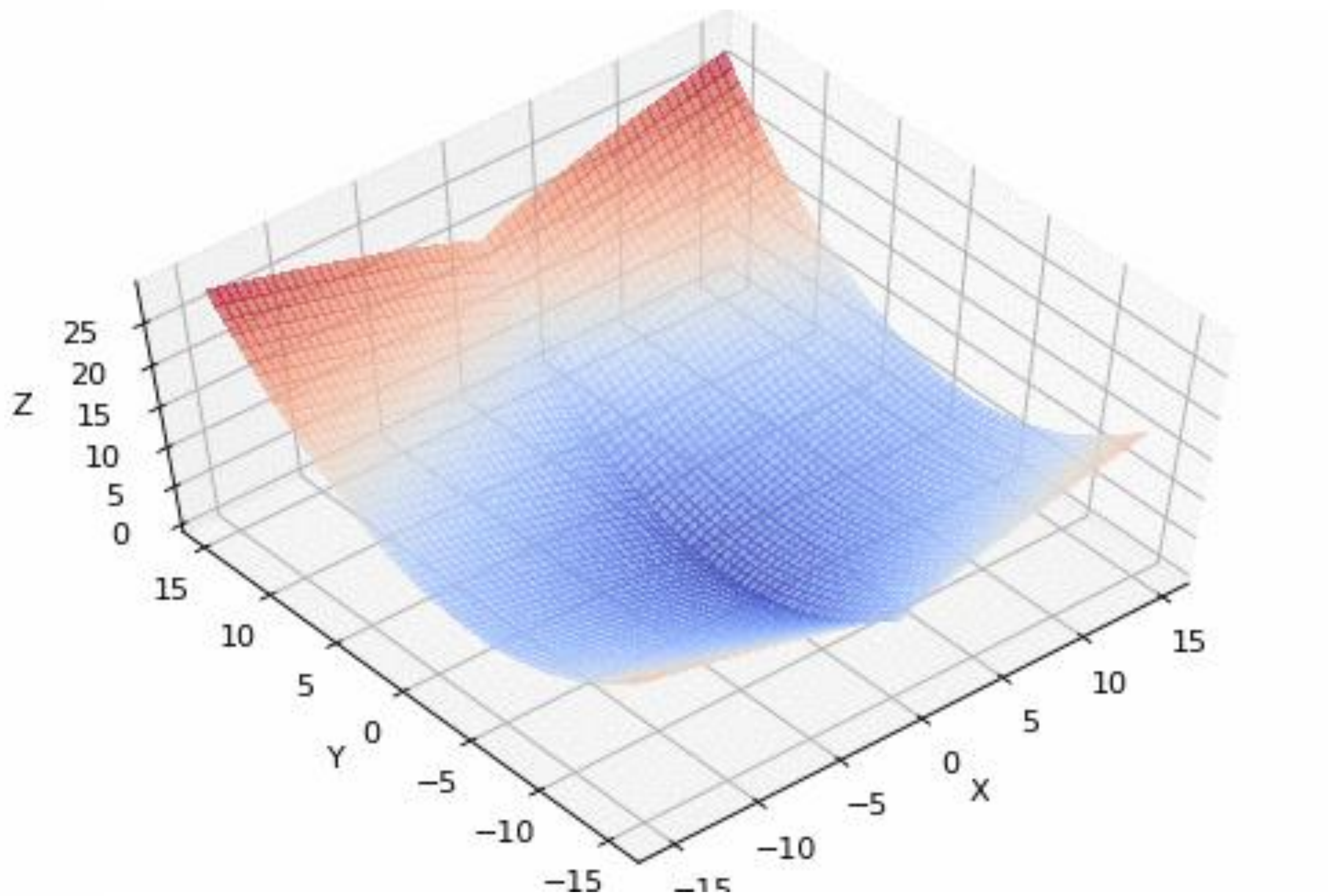


Very slow convergence due to vanishing gradients

What Happens on Non-Lipschitz Functions?

$f(x,y) = |x|^{0.8} + \frac{(y+3)^2}{15}$, so $\frac{\partial f}{\partial x}(x,y) = \text{sign}(x)\frac{0.8}{|x|^{0.2}}$ As $x \to 0$, $\left|\frac{\partial f}{\partial x}(x,y)\right| \to \infty$





Front view

Divergence due to exploding gradients

Top view

Slow Convergence to Optimal Parameter

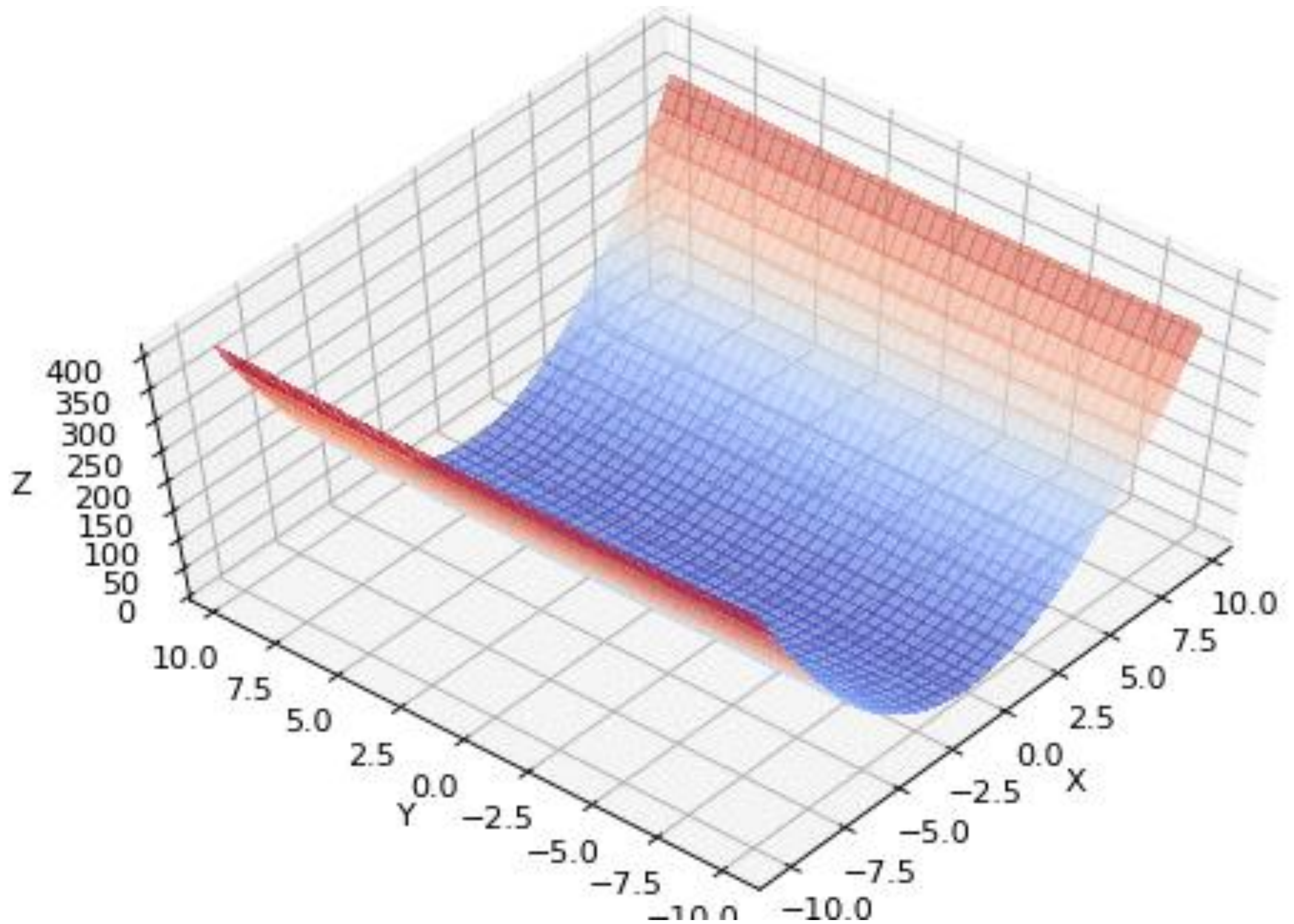
Consider $f(x,y) = 4x^2 + \frac{(y+3)^2}{15}$, which is strictly convex.

$$\frac{\partial^2}{\partial(x,y)\partial(x,y)^\top} f(x,y) = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2}(x,y) & \frac{\partial^2 f}{\partial x \partial y}(x,y) \\ \frac{\partial^2 f}{\partial y \partial x}(x,y) & \frac{\partial^2 f}{\partial y^2}(x,y) \end{pmatrix}$$

$$= \begin{pmatrix} 8 & 0 \\ 0 & \frac{2}{15} \end{pmatrix} \succ 0$$

As shown, gradient descent converges to the optimal parameter vector $\begin{pmatrix} 0 \\ -3 \end{pmatrix}$ very slowly.

(Though it does converge to the minimal objective value fairly quickly.)



Gradient Descent with Momentum

Often known as just “momentum” or “Polyak’s heavy ball method”.

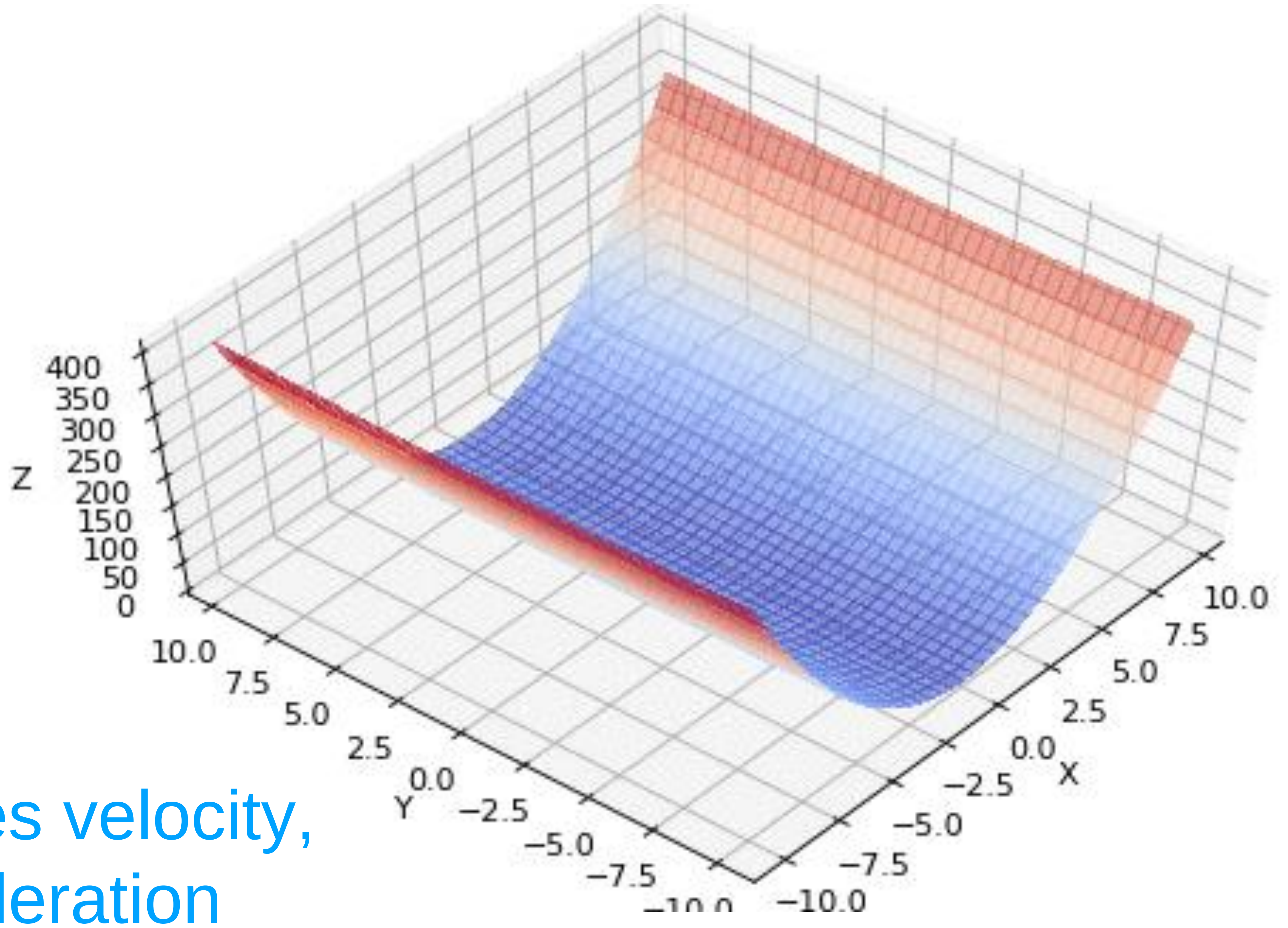
Gradient Descent with Momentum:

```
$\vec{\theta}^{(0)} \leftarrow$ random vector
$\Delta\vec{\theta} = \vec{0}$
for $t = 1,2,3,\ldots$
    $\Delta\vec{\theta} \leftarrow \alpha\Delta\vec{\theta} - \gamma_t \frac{\partial L}{\partial \vec{\theta}}\left(\vec{\theta}^{(t-1)}\right)$
    $\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} + \Delta\vec{\theta}$
    if algorithm has converged
        return $\vec{\theta}^{(t)}$
```

Akin to position of a particle

Akin to velocity of the particle

Gradient changes velocity, causing acceleration



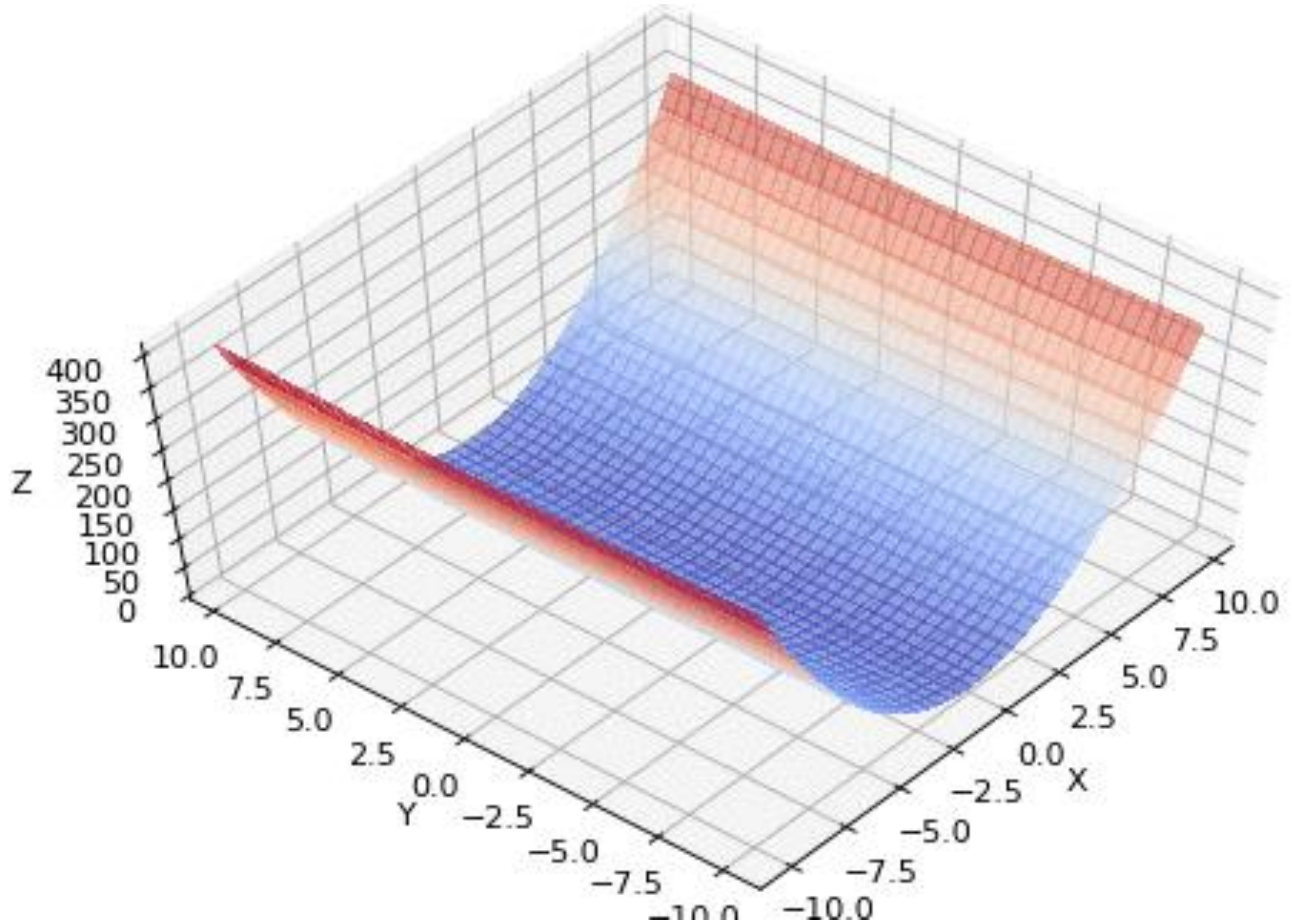
Gradient Descent with Momentum

Often known as just “momentum” or “Polyak’s heavy ball method”.

Gradient Descent with Momentum:

```
$\vec{\theta}^{(0)} \leftarrow$ random vector
$\Delta\vec{\theta} = \vec{0}$
for $t = 1,2,3, \ldots$
    $\Delta\vec{\theta} \leftarrow \alpha\Delta\vec{\theta} - \gamma_t \frac{\partial L}{\partial \vec{\theta}}\left(\vec{\theta}^{(t-1)}\right)$
    $\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} + \Delta\vec{\theta}$
    if algorithm has converged
        return $\vec{\theta}^{(t)}$
```

$\alpha \in [0,1)$ is a hyperparameter and is known as the “momentum parameter”



Momentum vs. Gradient Descent

Often known as just “momentum” or “Polyak’s heavy ball method”.

Gradient Descent with Momentum:

```
$\vec{\theta}^{(0)} \leftarrow$ random vector
$\Delta\vec{\theta} = \vec{0}$
for $t = 1,2,3,\ldots$
    $\Delta\vec{\theta} \leftarrow \alpha\Delta\vec{\theta} - \gamma_t \frac{\partial L}{\partial \vec{\theta}}\left(\vec{\theta}^{(t-1)}\right)$
    $\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} + \Delta\vec{\theta}$
    if algorithm has converged
        return $\vec{\theta}^{(t)}$
```

Gradient changes velocity

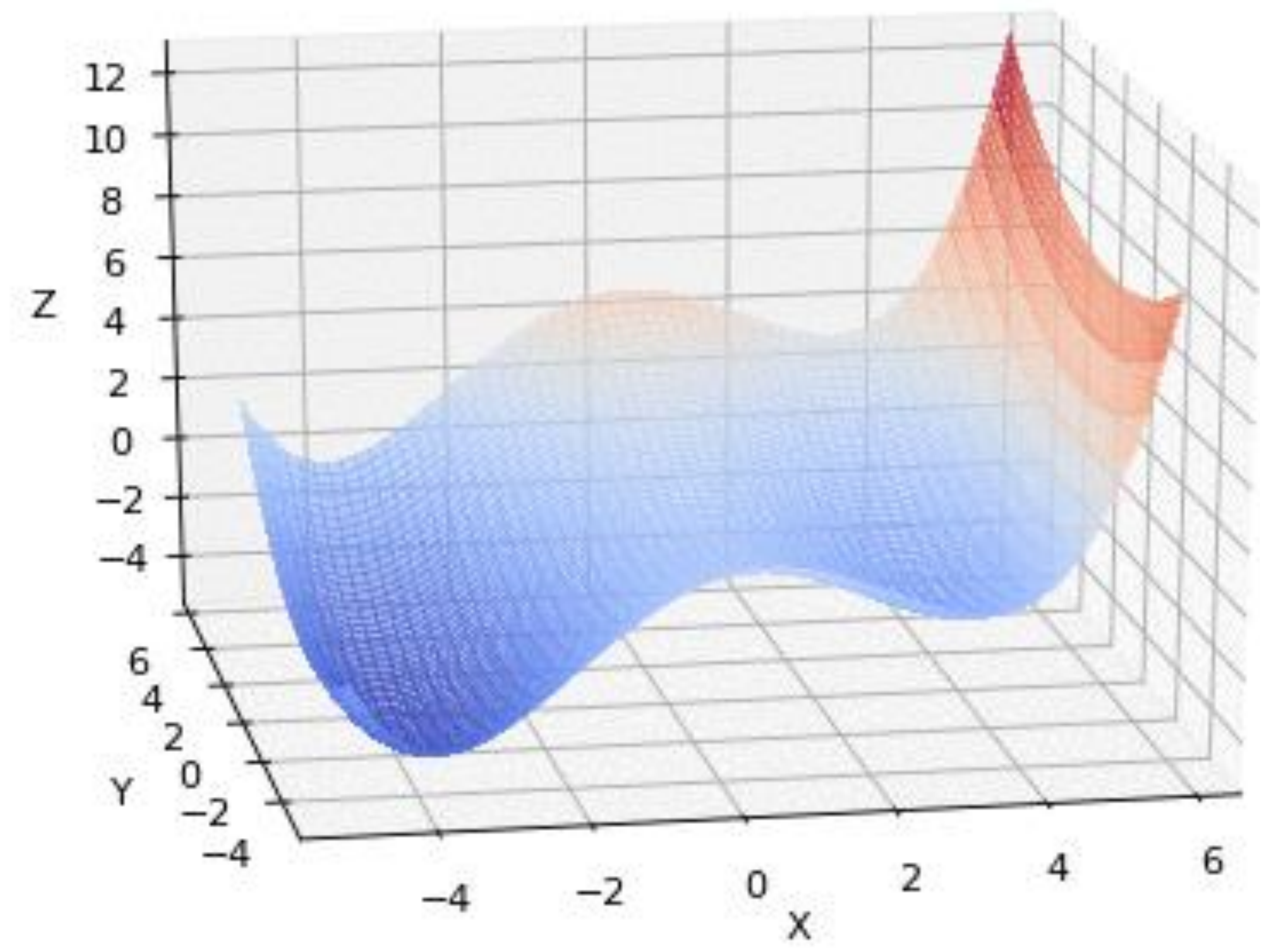
Gradient Descent:

```
$\vec{\theta}^{(0)} \leftarrow$ random vector
for $t = 1,2,3,\ldots$
    $\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} - \gamma_t \frac{\partial L}{\partial \vec{\theta}}\left(\vec{\theta}^{(t-1)}\right)$
    if algorithm has converged
        return $\vec{\theta}^{(t)}$
```

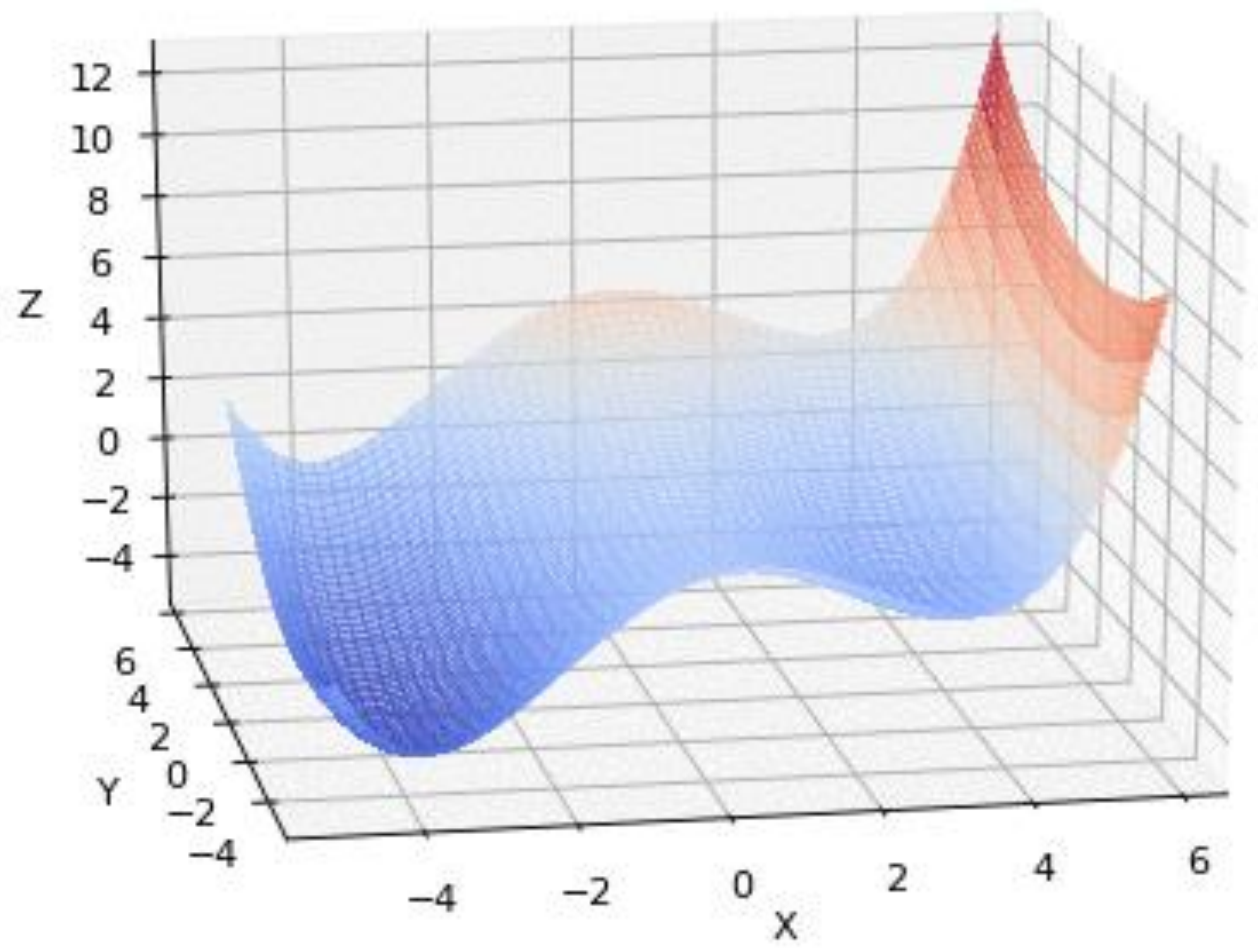
Gradient changes position

Equivalent to setting $\alpha = 0$

What Happens on Non-Convex Functions?

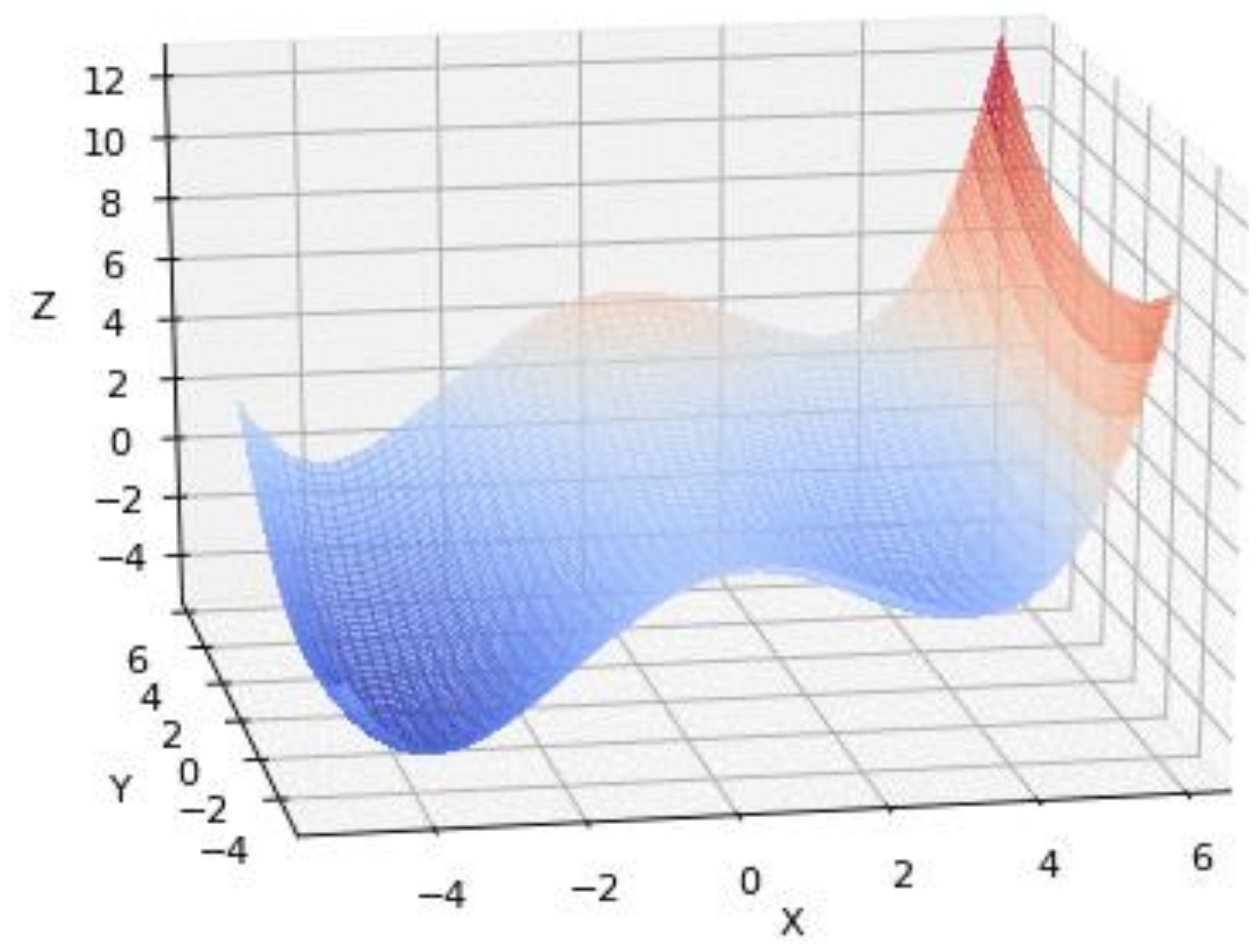


Gradient Descent
Gets stuck in local minimum

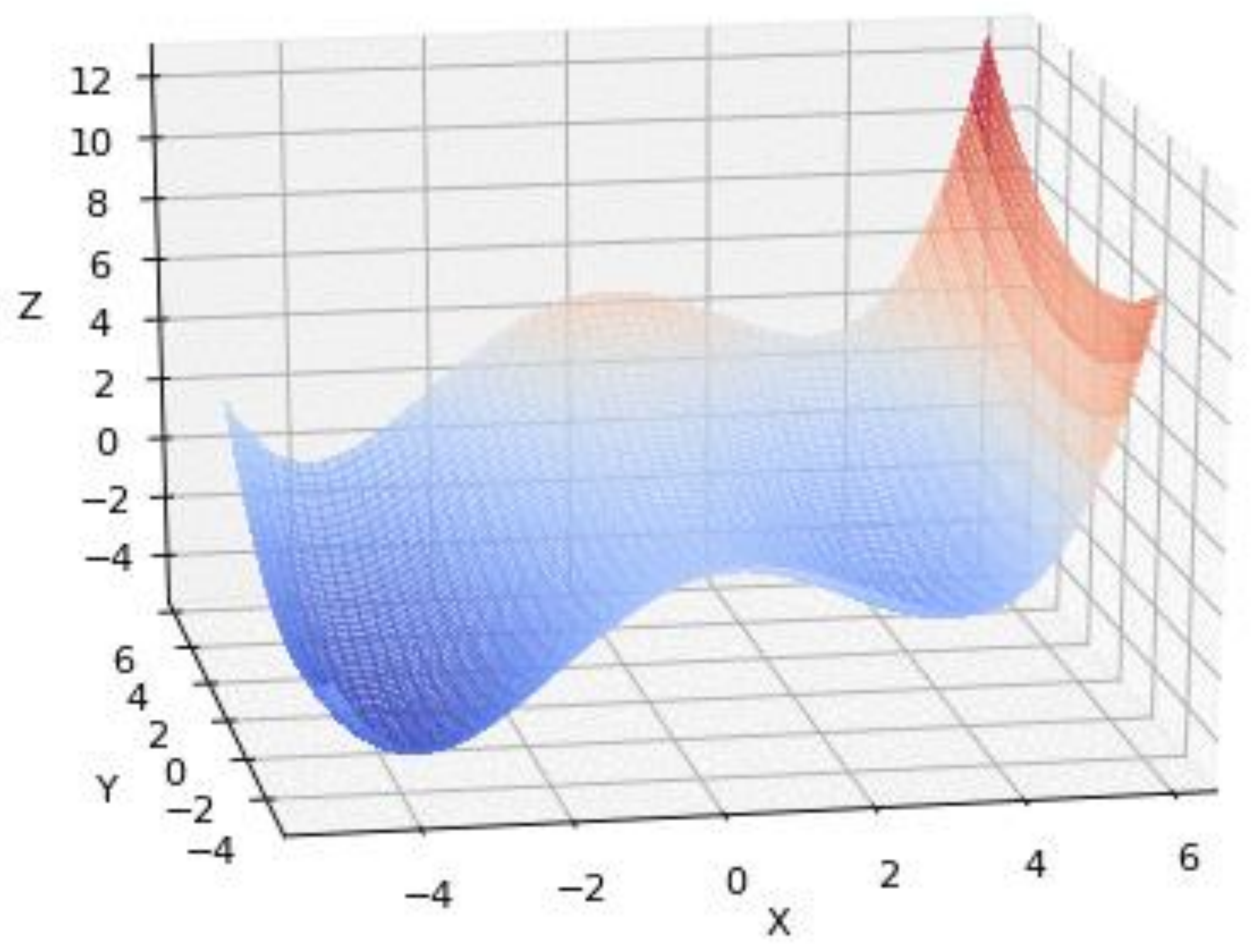


Momentum
Sometimes gets past local minimum

What Happens on Non-Convex Functions?

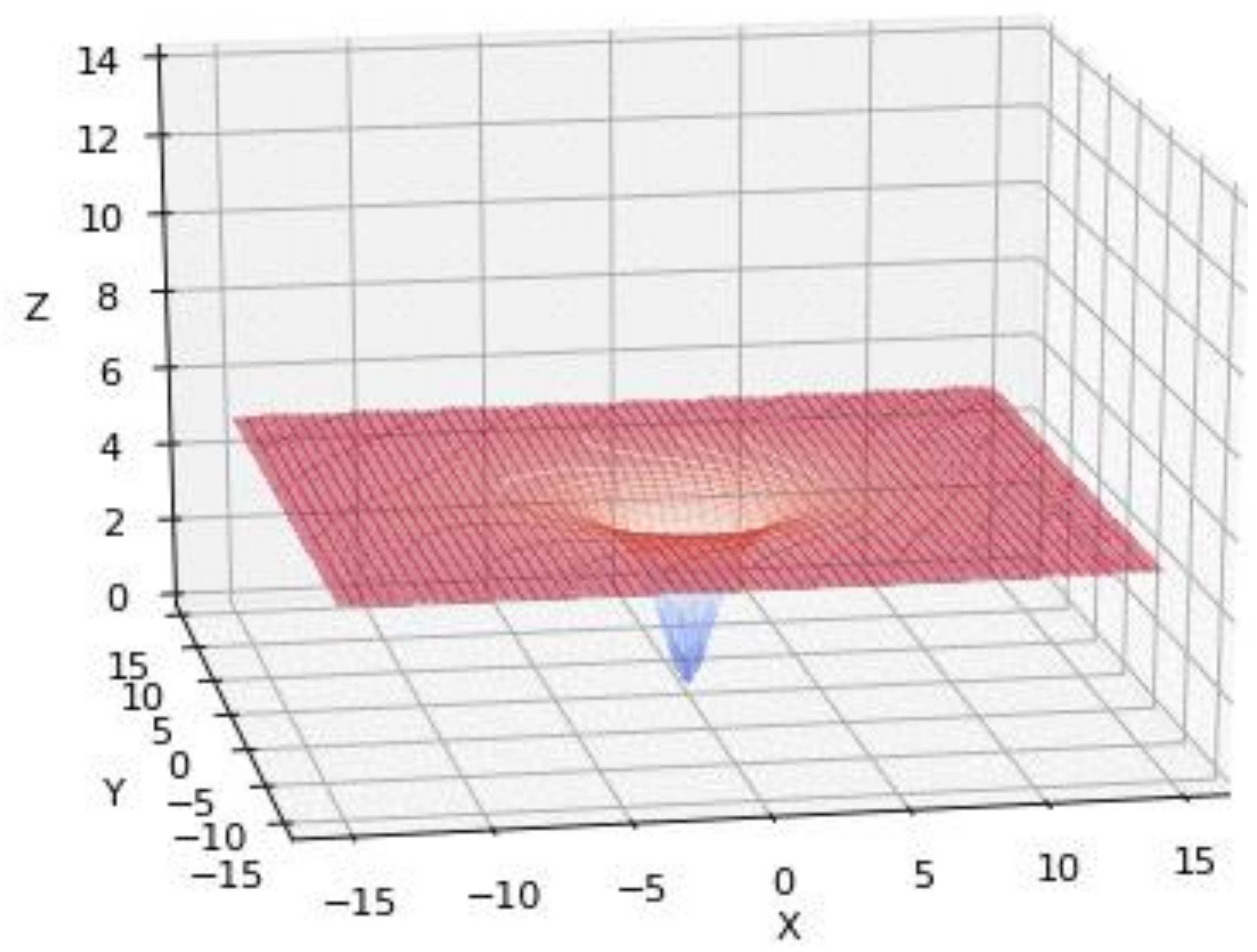


Gradient Descent
Gets stuck in local minimum

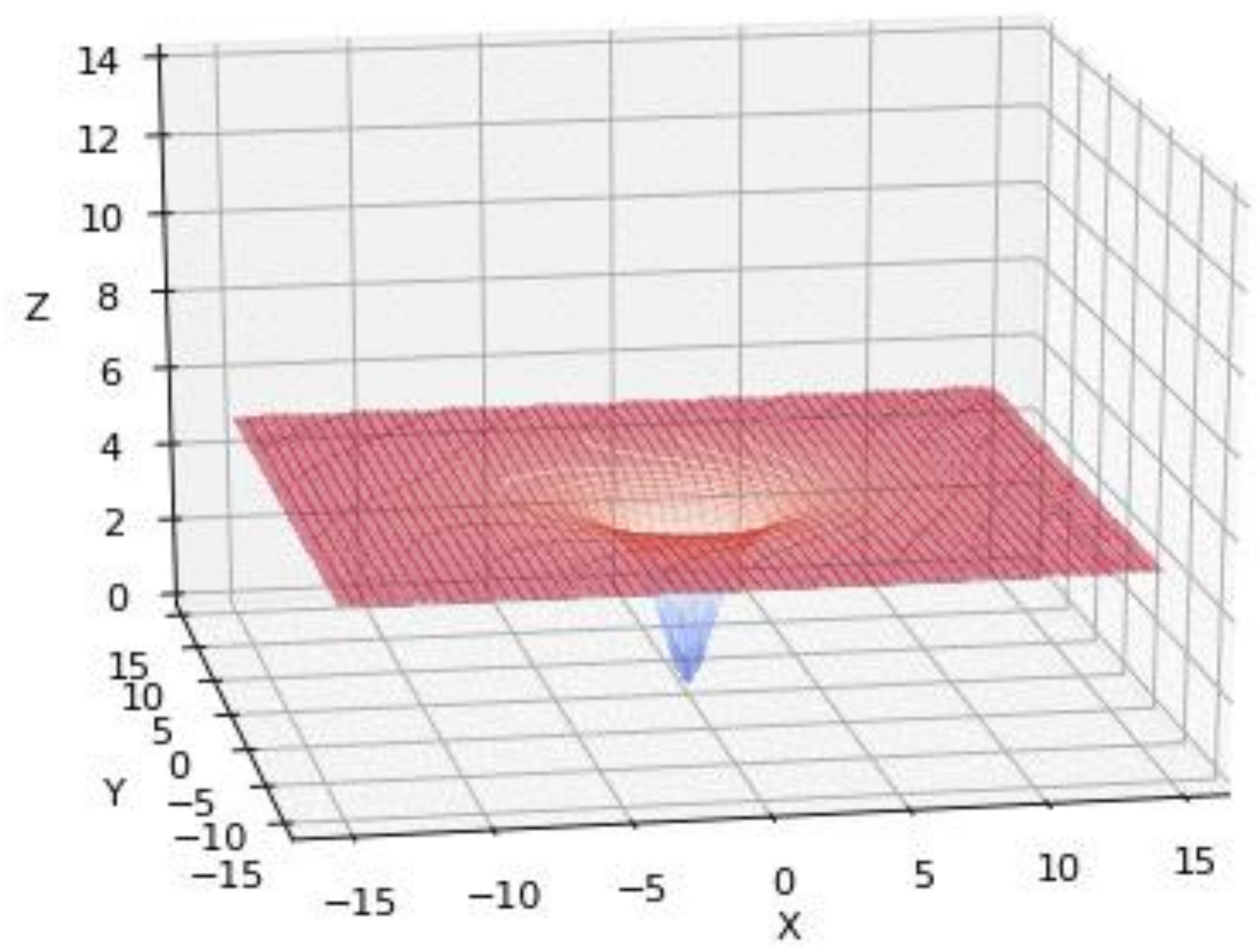


Momentum
With smaller step size, returns to local minimum

What Happens on Non-Convex Functions?

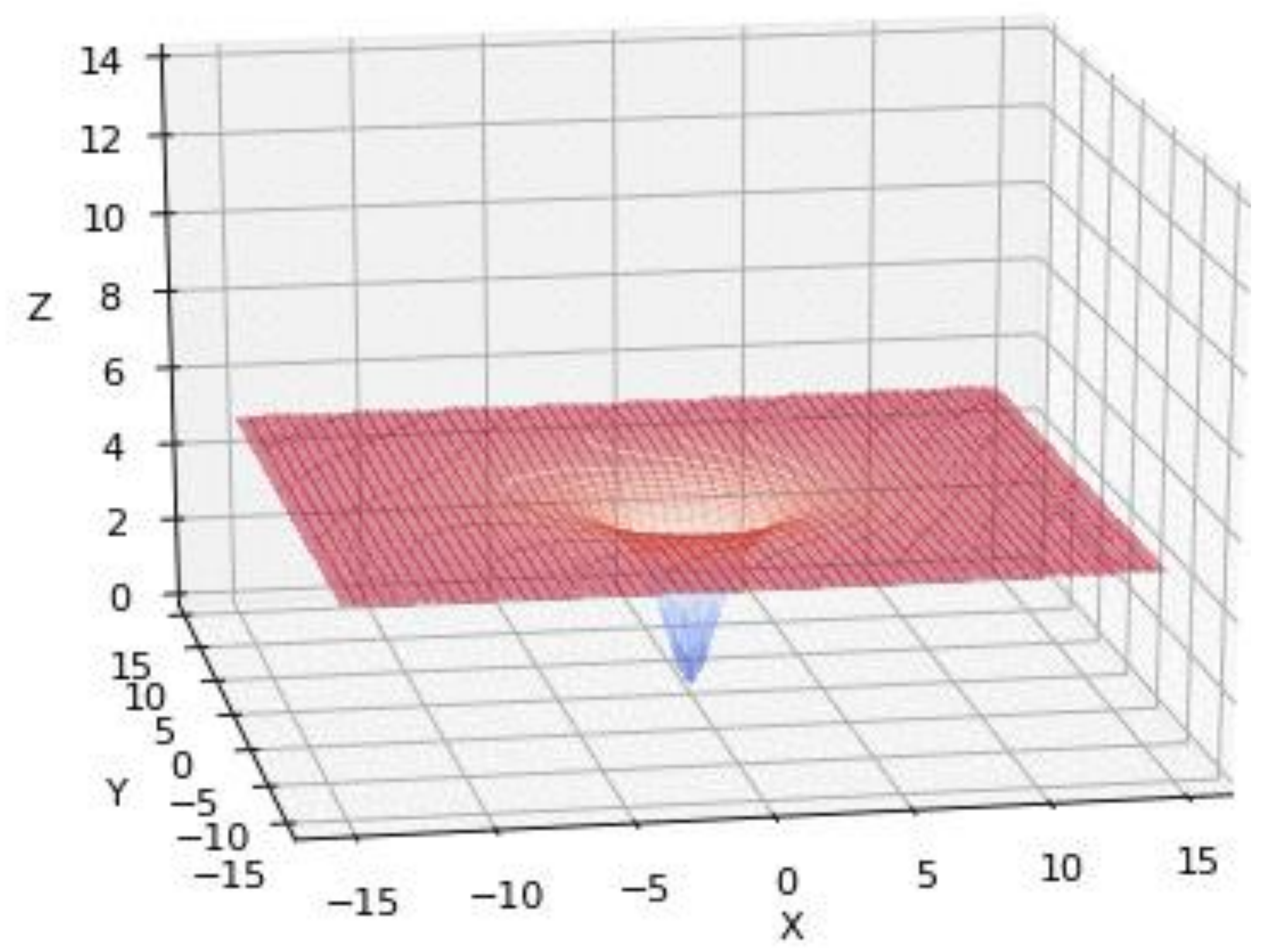


Gradient Descent
Very slow convergence due to vanishing gradients

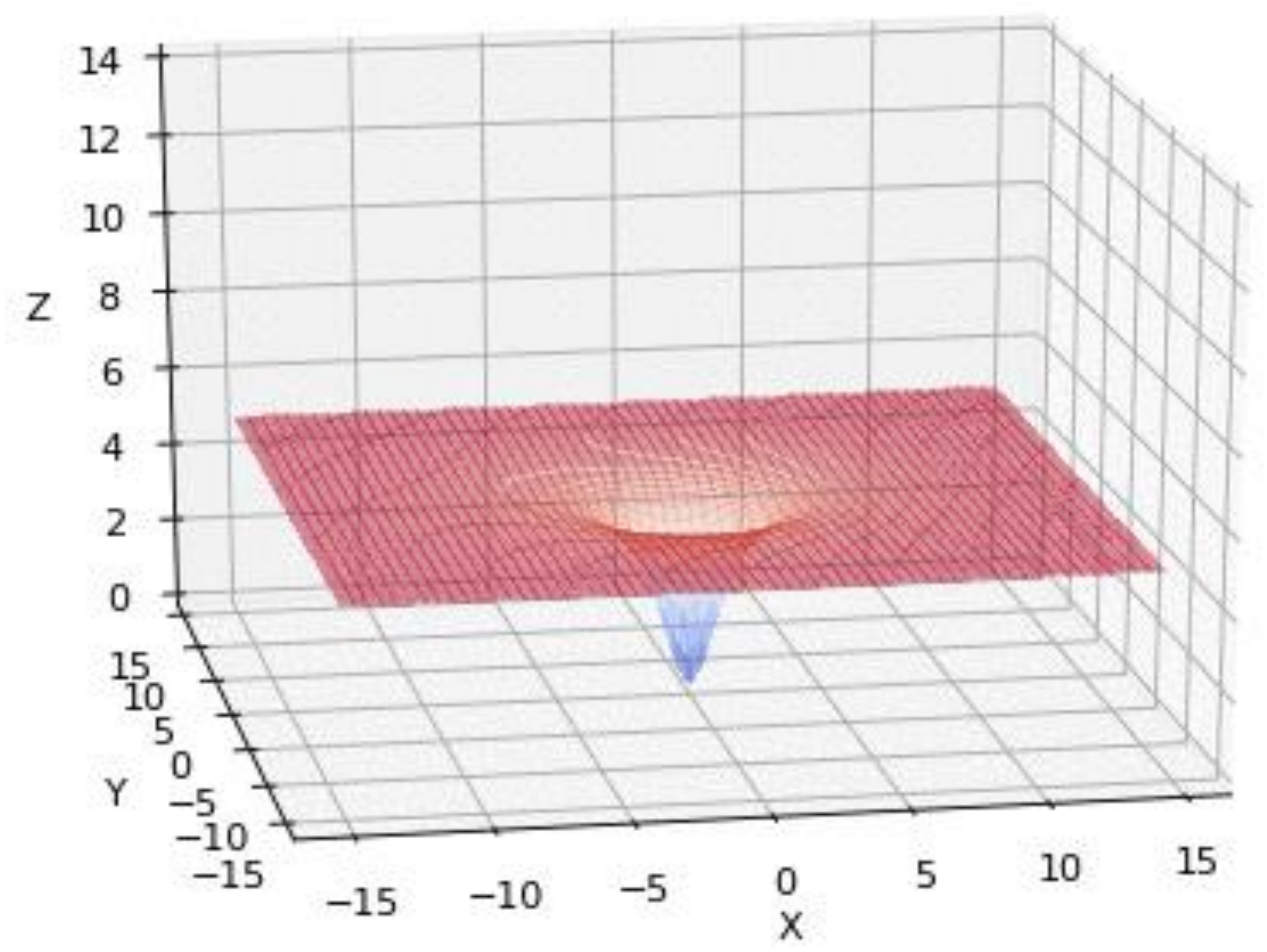


Momentum
Sometimes converges faster despite vanishing gradients

What Happens on Non-Convex Functions?

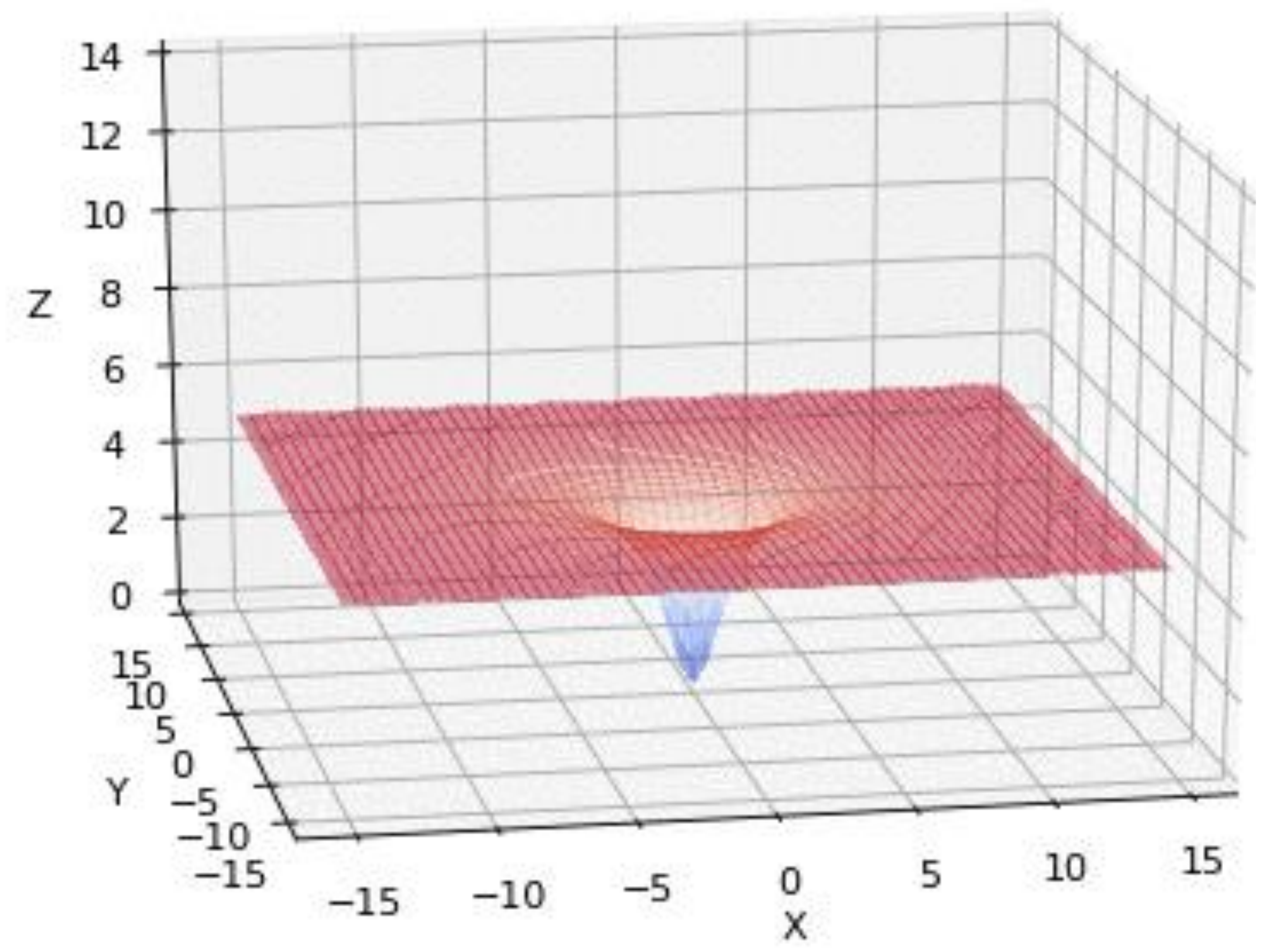


Gradient Descent
Very slow convergence due to vanishing gradients

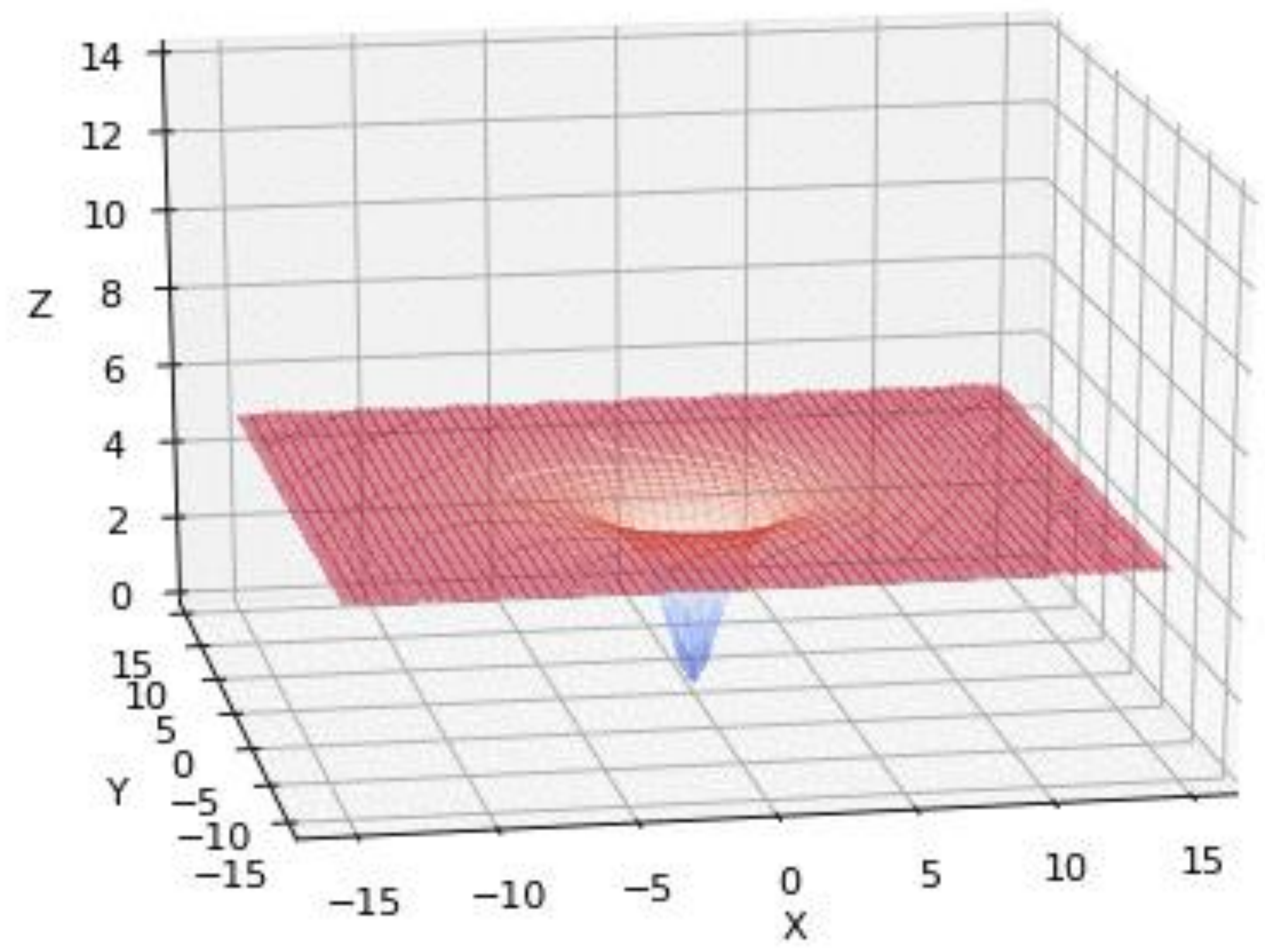


Momentum
With smaller step size, still converges slowly

What Happens on Non-Convex Functions?



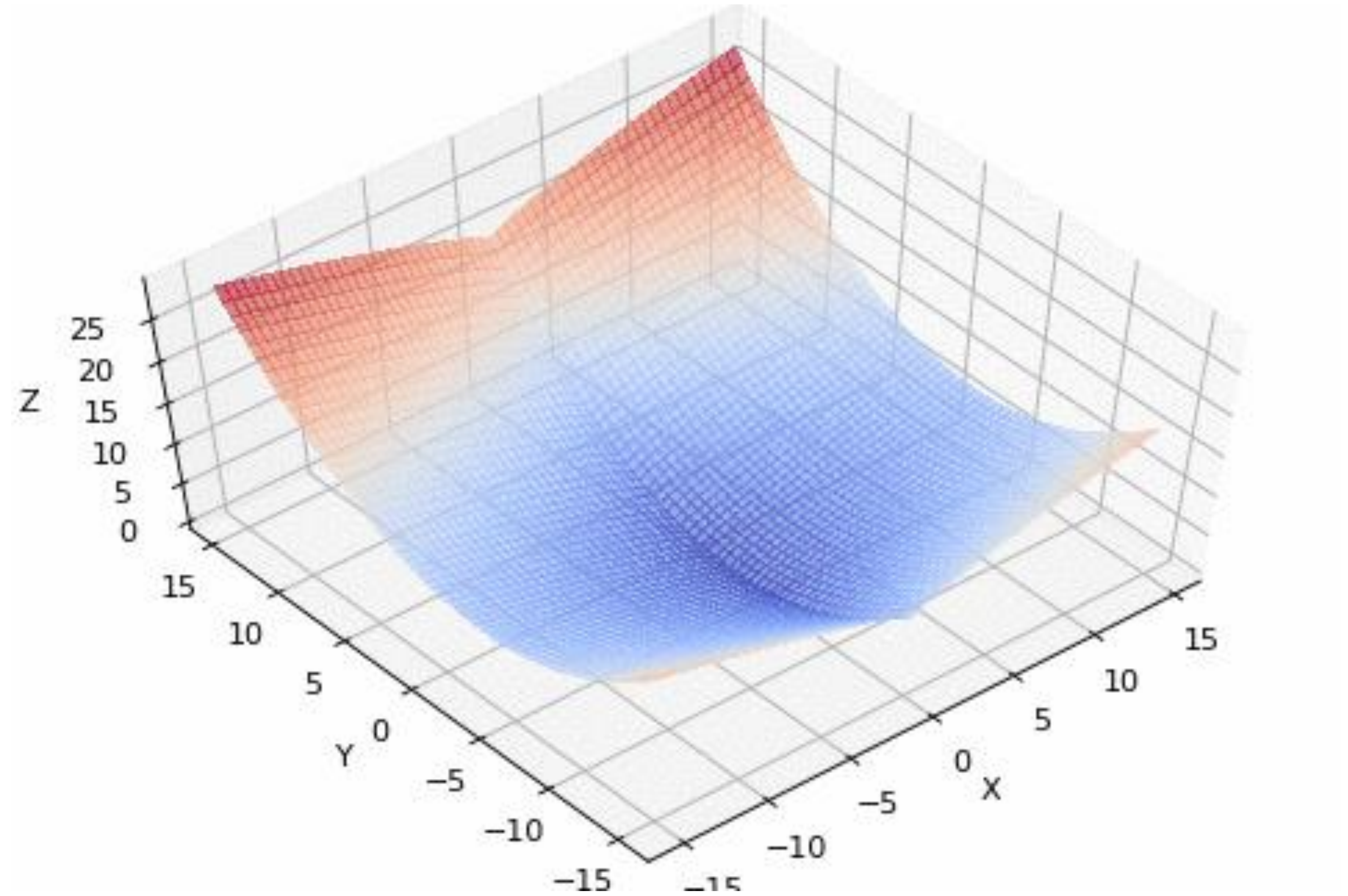
Gradient Descent
Very slow convergence due to vanishing gradients



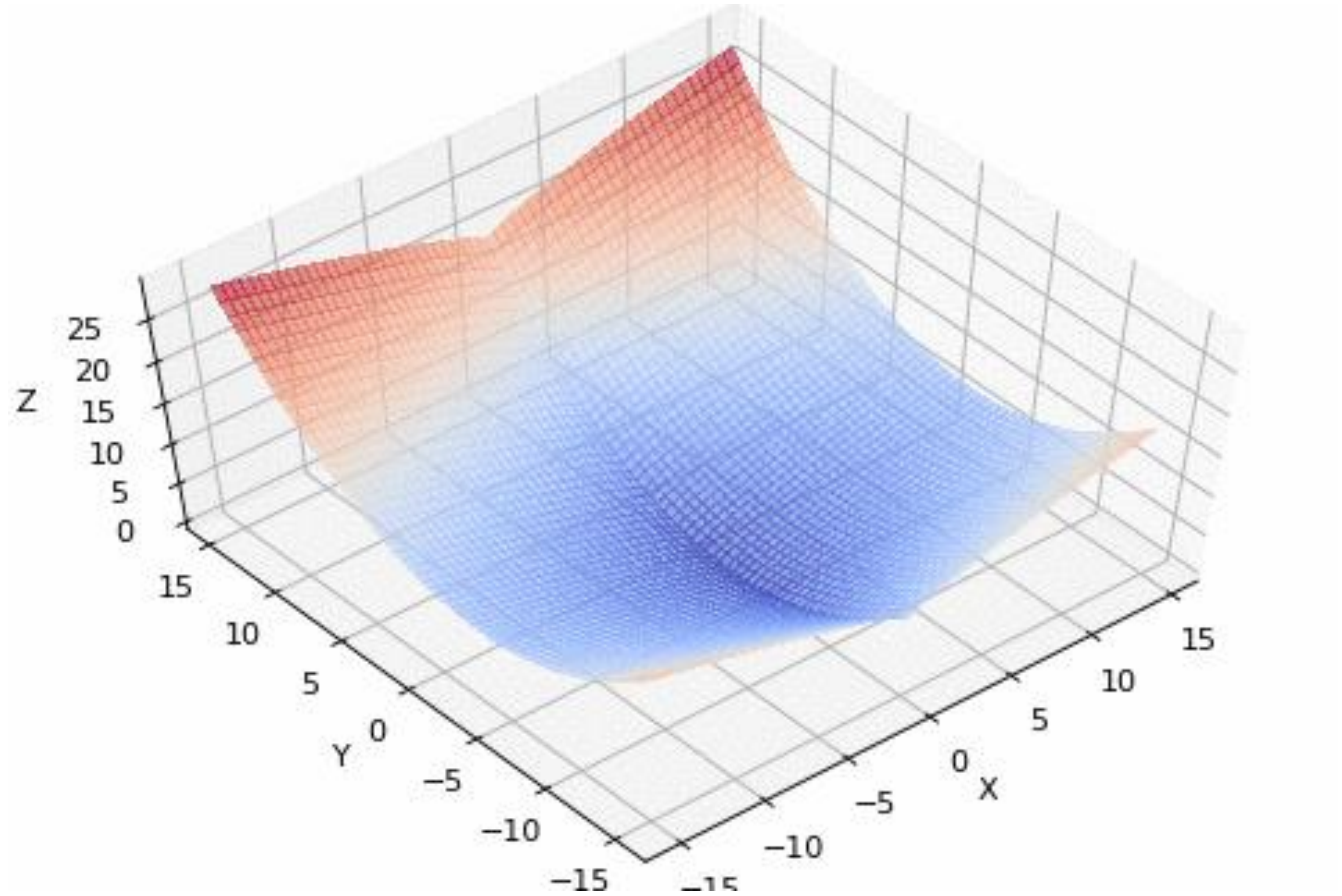
Momentum
With larger step size, can overshoot

What Happens on Non-Lipschitz Functions?

$f(x,y) = |x|^{0.8} + \frac{(y+3)^2}{15}$, so $\frac{\partial f}{\partial x}(x,y) = \text{sign}(x)\frac{0.8}{|x|^{0.2}}$ As $x \to 0$, $\left|\frac{\partial f}{\partial x}(x,y)\right| \to \infty$



Gradient Descent
Diverges



Momentum
Diverges also, albeit less severely

Ways Around Non-Lipschitzness

Apply a strictly increasing transformation to the objective function

- E.g.: The function $x \mapsto \sqrt{x}$ is non-Lipschitz. So instead of solving $\min_{\vec{w}} \|\vec{y} - X\vec{w}\|_2 = \min_{\vec{w}} \sqrt{(\vec{y} - X\vec{w})^\top (\vec{y} - X\vec{w})}$, we apply the transformation $y \mapsto y^2$ which is strictly increasing for $y \geq 0$) to the objective and solve $\min_{\vec{w}} \|\vec{y} - X\vec{w}\|_2^2$.

Gradient clipping

- No theoretical justification for this, but sometimes works well in practice
- $\frac{\partial L}{\partial \theta_i}\left(\vec{\theta}^{(t)}\right) \leftarrow \min\left(\left|\frac{\partial L}{\partial \theta_i}\left(\vec{\theta}^{(t)}\right)\right|, 10^3\right) \operatorname{sign}\left(\frac{\partial L}{\partial \theta_i}\left(\vec{\theta}^{(t)}\right)\right) \forall i \in \{1, \ldots, n\}$

Nesterov’s Accelerated Gradient (NAG)

Gradient Descent with Momentum:

```
$\vec{\theta}^{(0)} \leftarrow$ random vector
$\Delta\vec{\theta} = \vec{0}$
for $t = 1,2,3,\ldots$
    $\Delta\vec{\theta} \leftarrow \alpha\Delta\vec{\theta} - \gamma_t \frac{\partial L}{\partial\vec{\theta}}(\vec{\theta}^{(t-1)})$
    $\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} + \Delta\vec{\theta}$
    if algorithm has converged
        return $\vec{\theta}^{(t)}$
```

Nesterov’s Accelerated Gradient:

```
$\vec{\theta}^{(0)} \leftarrow$ random vector
$\Delta\vec{\theta} = \vec{0}$
for $t = 1,2,3,\ldots$
    $\Delta\vec{\theta} \leftarrow \alpha\Delta\vec{\theta} - \gamma_t \frac{\partial L}{\partial\vec{\theta}}(\vec{\theta}^{(t-1)} + \alpha\Delta\vec{\theta})$
    $\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} + \Delta\vec{\theta}$
    if algorithm has converged
        return $\vec{\theta}^{(t)}$
```

Lookahead: account for the inertia when choosing where to compute gradient

Adaptive Gradient Methods

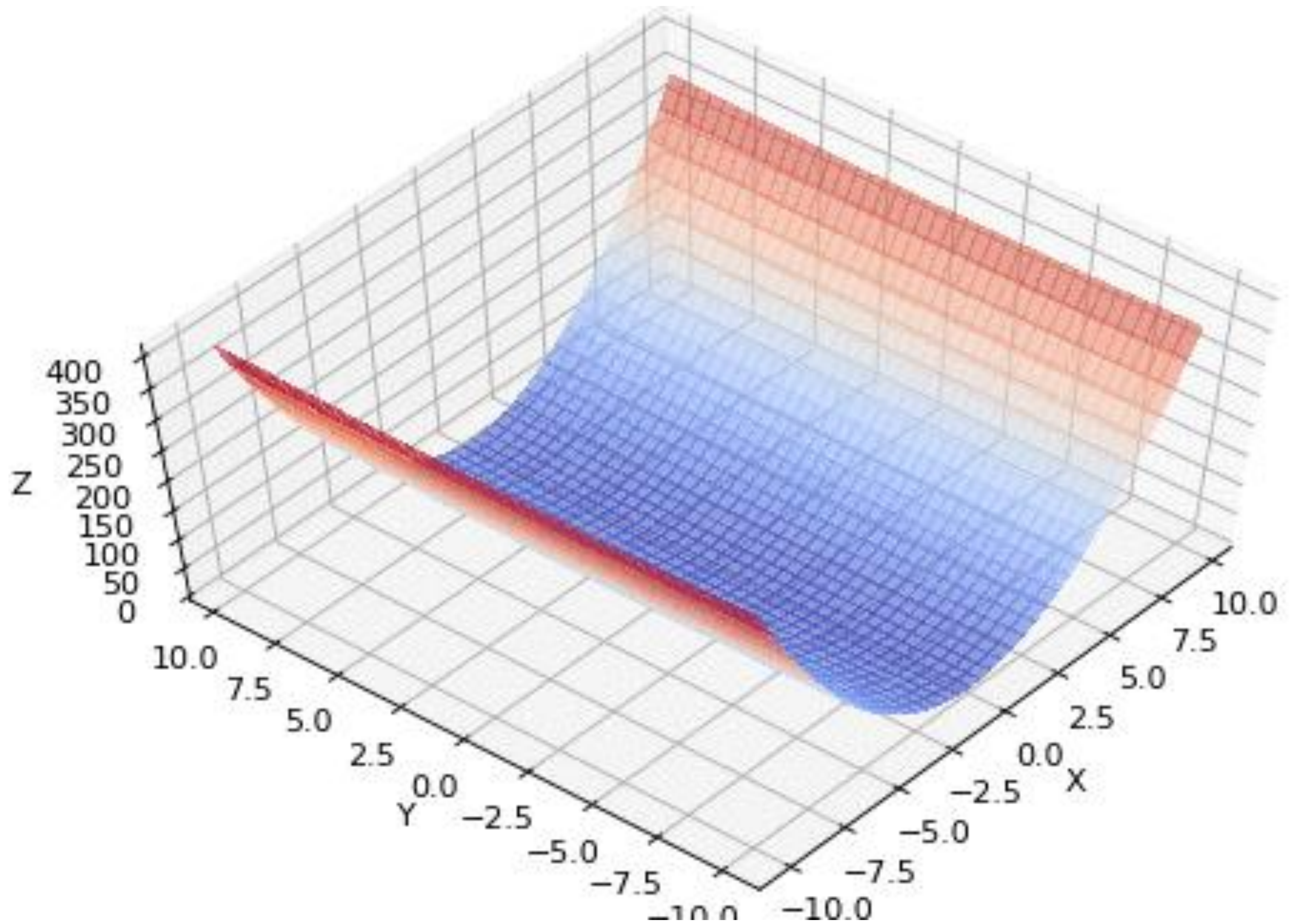
Recall the function $f(x,y) = 4x^2 + \frac{(y+3)^2}{15}$.

$$\frac{\partial^2}{\partial(x,y)\partial(x,y)^\top} f(x,y) = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2}(x,y) & \frac{\partial^2 f}{\partial x \partial y}(x,y) \\ \frac{\partial^2 f}{\partial y \partial x}(x,y) & \frac{\partial^2 f}{\partial y^2}(x,y) \end{pmatrix}$$

$$= \begin{pmatrix} 8 & 0 \\ 0 & \frac{2}{15} \end{pmatrix} \succ 0$$

The eigenvalues of the Hessian (which in this case are the diagonal entries) have very different magnitudes.

Gradient descent converges to the optimal parameters slowly.



Adaptive Gradient Methods

Recall the function $f(x,y) = 4x^2 + \frac{(y+3)^2}{15}$.

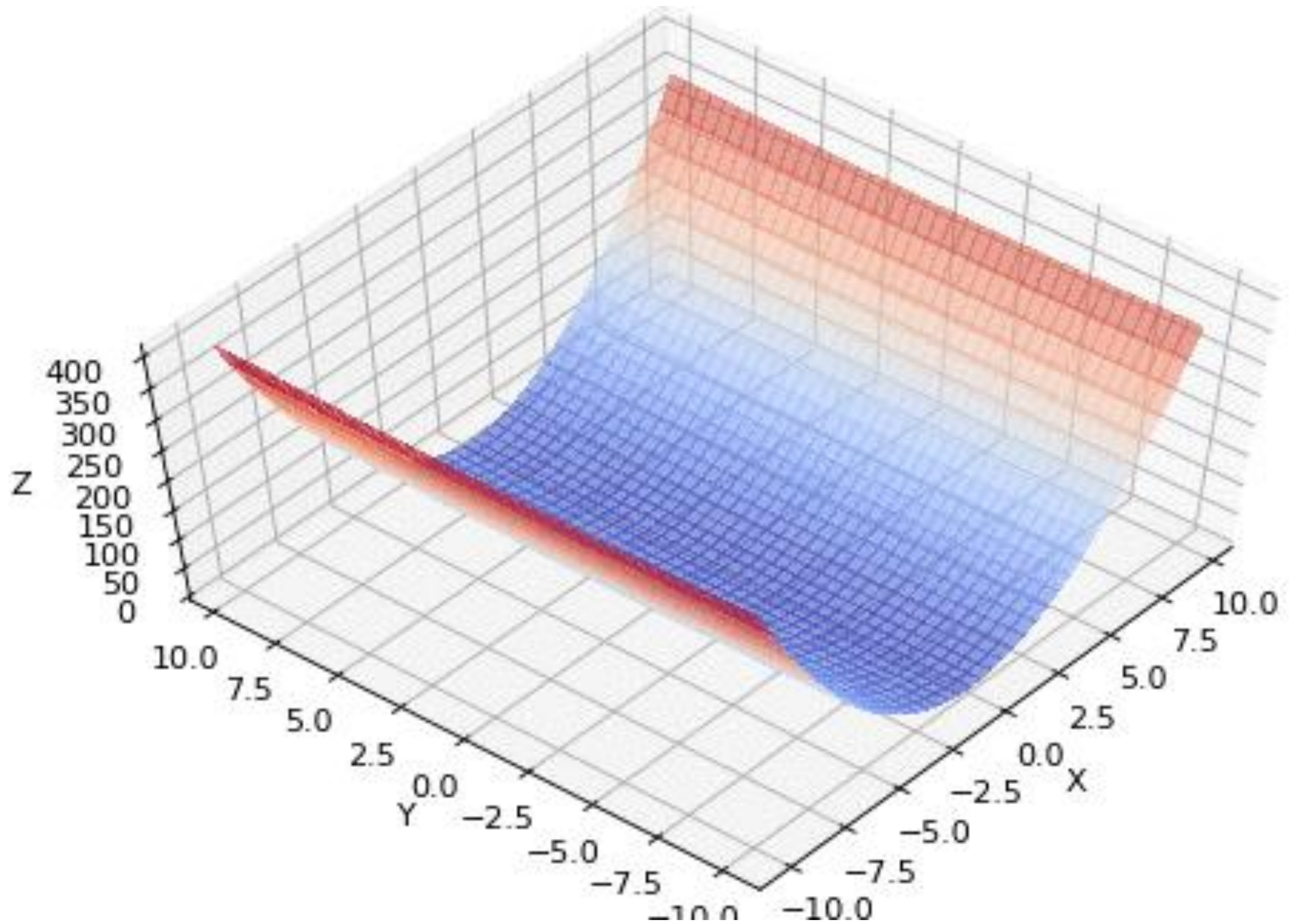
$$\frac{\partial^2}{\partial(x,y)\partial(x,y)^\top} f(x,y) = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2}(x,y) & \frac{\partial^2 f}{\partial x \partial y}(x,y) \\ \frac{\partial^2 f}{\partial y \partial x}(x,y) & \frac{\partial^2 f}{\partial y^2}(x,y) \end{pmatrix}$$

$$= \begin{pmatrix} 8 & 0 \\ 0 & \frac{2}{15} \end{pmatrix} \succ 0$$

Idea: Try to make the coefficient of each term 1.

Let $x' = 2x, y' = \frac{y}{\sqrt{15}}$, so $x = \frac{x'}{2}, y = \sqrt{15}y'$.

"Reparameterization"



Adaptive Gradient Methods

Let $x' = 2x$, $y' = \frac{y}{\sqrt{15}}$, so $x = \frac{x'}{2}$, $y = \sqrt{15}y'$.

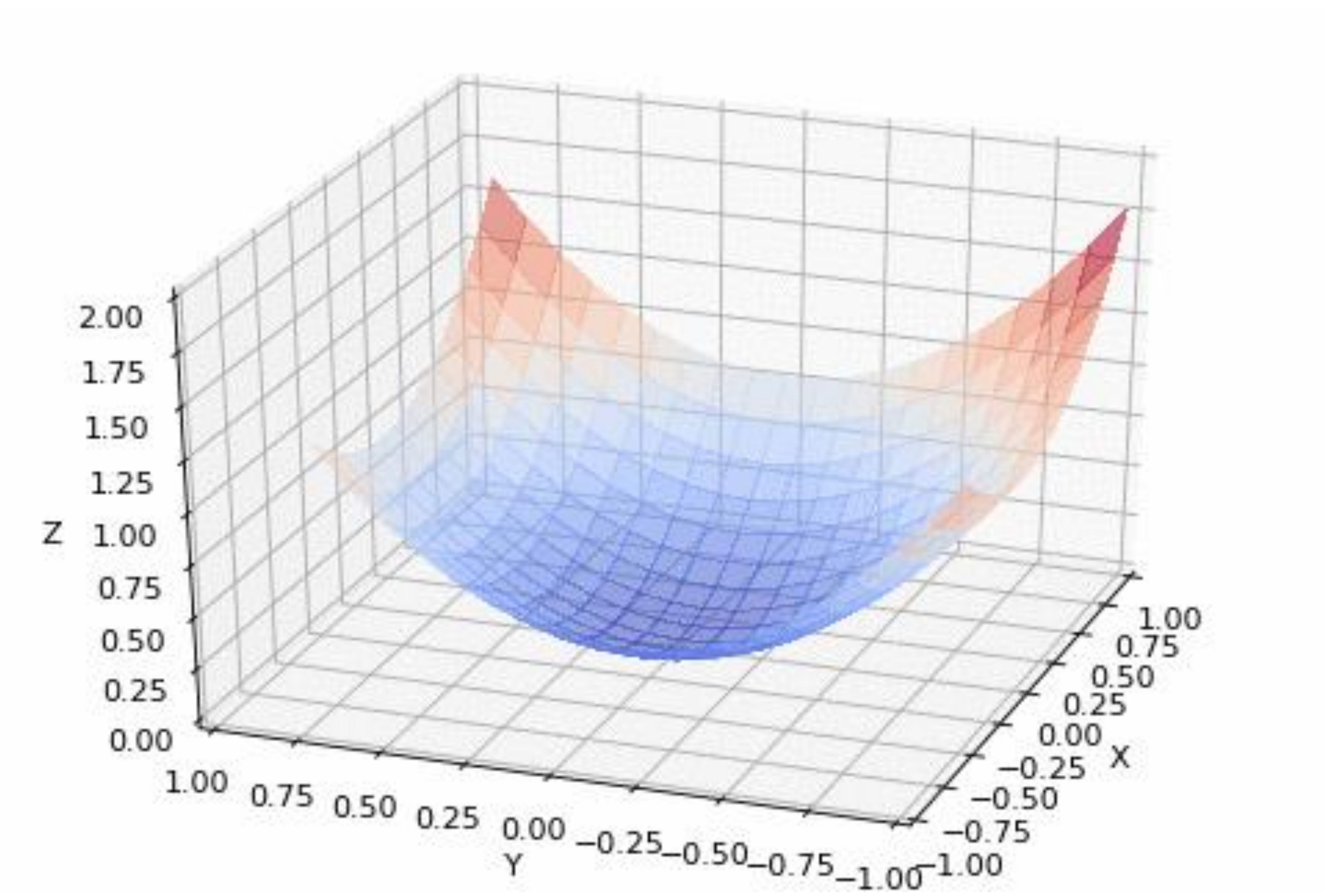
$$f(x, y) = 4x^2 + \frac{(y+3)^2}{15}$$

$$= 4\left(\frac{x'}{2}\right)^2 + \frac{(\sqrt{15}y'+3)^2}{15}$$

$$= 4\left(\frac{x'}{2}\right)^2 + \frac{\left(\sqrt{15}\left(y'+\frac{3}{\sqrt{15}}\right)\right)^2}{15}$$

$$= 4\left(\frac{x'^2}{4}\right) + \frac{15\left(y'+\frac{3}{\sqrt{15}}\right)^2}{15}$$

$$= x'^2 + \left(y' + \frac{3}{\sqrt{15}}\right)^2$$



Gradient descent converges quickly after reparameterization!

Adaptive Gradient Methods

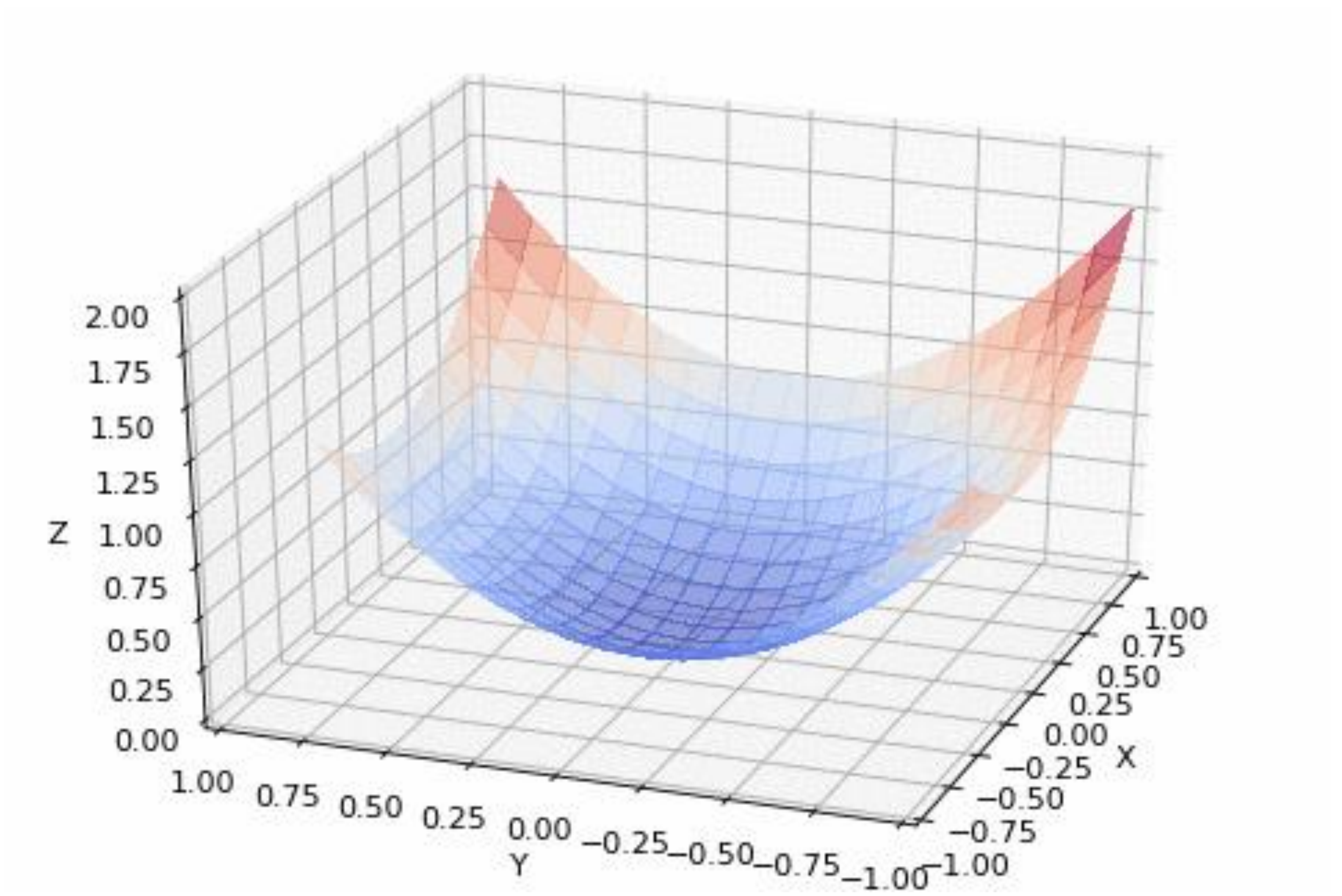
In general, suppose we reparametrize $\vec{\theta}' = A^{-1}\vec{\theta}$, so $\vec{\theta} = A\vec{\theta}'$. We can then write $L(\vec{\theta})$ as $L\left(A\vec{\theta}'\right)$

$$\frac{\partial L}{\partial \vec{\theta}'} = \frac{\partial\left(A\vec{\theta}'\right)}{\partial \vec{\theta}'} \frac{\partial L}{\partial\left(A\vec{\theta}'\right)}$$

$$= \frac{\partial\left(A\vec{\theta}'\right)}{\partial \vec{\theta}'} \frac{\partial L}{\partial \vec{\theta}}$$

$$= A^{\mathsf{T}} \frac{\partial L}{\partial \vec{\theta}}$$

$$\frac{\partial L}{\partial \vec{\theta}} = (A^{\mathsf{T}})^{-1} \frac{\partial L}{\partial \vec{\theta}'}$$



Challenge: We don't know which A to choose to make gradient descent converge quickly.

Adaptive gradient methods pick A adaptively based on past gradients, effectively reparameterizing on-the-fly.

AdaGrad

Gradient Descent:

```
$\vec{\theta}^{(0)} \leftarrow$ random vector
for $t = 1,2,3,\ldots$
    $\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} - \gamma_t \frac{\partial L}{\partial \vec{\theta}}(\vec{\theta}^{(t-1)})$
    if algorithm has converged
        return $\vec{\theta}^{(t)}$
```

AdaGrad (short for “adaptive gradient”):

```
$\vec{\theta}^{(0)} \leftarrow$ random vector
for $t = 1,2,3,\ldots$
    $D \leftarrow \operatorname{diag}\left(\left\{\sqrt{\sum_{t'=1}^{t-1}\left(\frac{\partial L}{\partial \theta_i}(\vec{\theta}^{(t')})\right)^2}\right\}_{i=1}^{n}\right)$
    $\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} - \gamma_t D^{-1} \frac{\partial L}{\partial \vec{\theta}}\left(\vec{\theta}^{(t-1)}\right)$
    if algorithm has converged
        return $\vec{\theta}^{(t)}$
```

“Preconditioner”

Adam

Gradient Descent with Momentum:

```
$\vec{\theta}^{(0)} \leftarrow$ random vector
$\Delta\vec{\theta} = \vec{0}$
for $t = 1,2,3,\ldots$
    $\Delta\vec{\theta} \leftarrow \alpha\Delta\vec{\theta} - \gamma_t \frac{\partial L}{\partial \vec{\theta}}(\vec{\theta}^{(t-1)})$
    $\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} + \Delta\vec{\theta}$
    if algorithm has converged
        return $\vec{\theta}^{(t)}$
```

Adam (short for "adaptive moment estimation"):

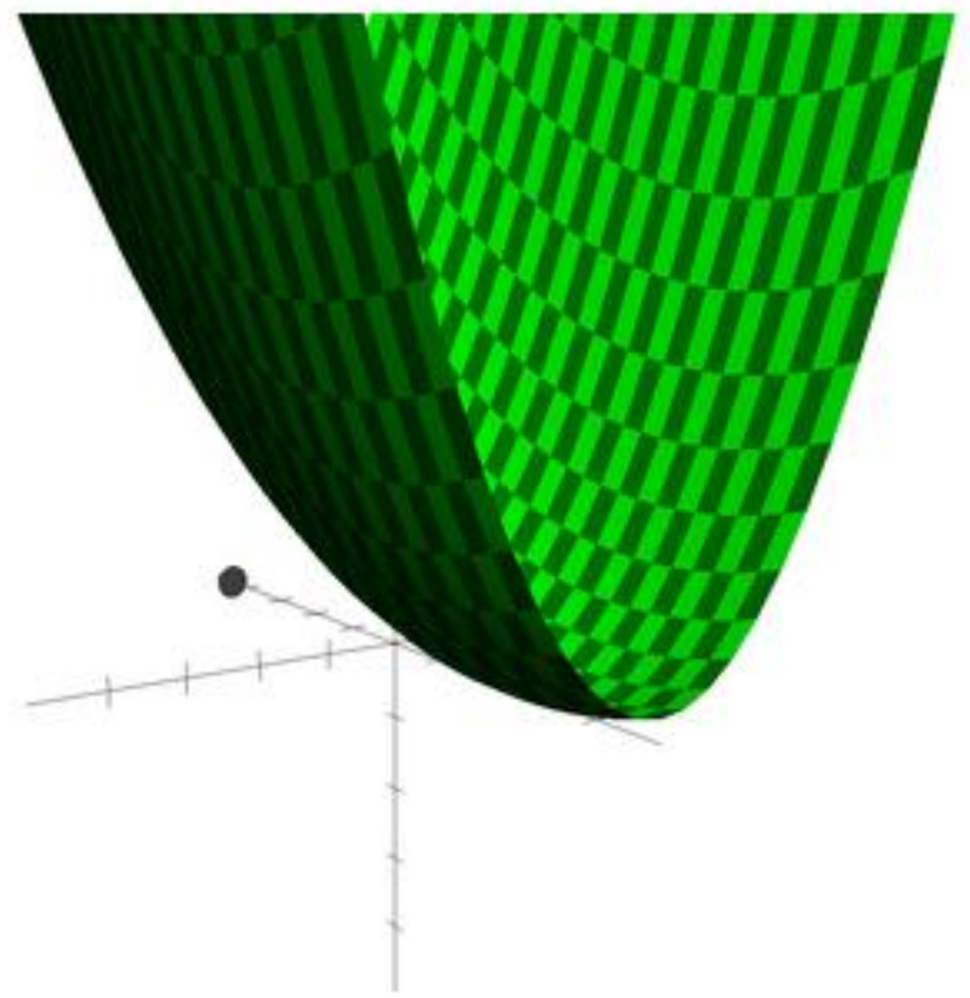
```
$\vec{\theta}^{(0)} \leftarrow$ random vector
for $t = 1,2,3,\ldots$
    $\vec{m} \leftarrow \alpha\vec{m} + (1-\alpha)\frac{\partial L}{\partial \vec{\theta}}(\vec{\theta}^{(t-1)})$
    $D \leftarrow \beta D + (1-\beta)\operatorname{diag}\left(\left\{\left(\frac{\partial L}{\partial \theta_i}\left(\vec{\theta}^{(t-1)}\right)\right)^2\right\}_{i=1}^{n}\right)$
    $\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} - \gamma_t\left(\left(\frac{1}{1-\beta^t}D\right)^{\frac{1}{2}} + \epsilon I\right)^{-1}\left(\frac{1}{1-\alpha^t}\vec{m}\right)$

    if algorithm has converged
        return $\vec{\theta}^{(t)}$
```

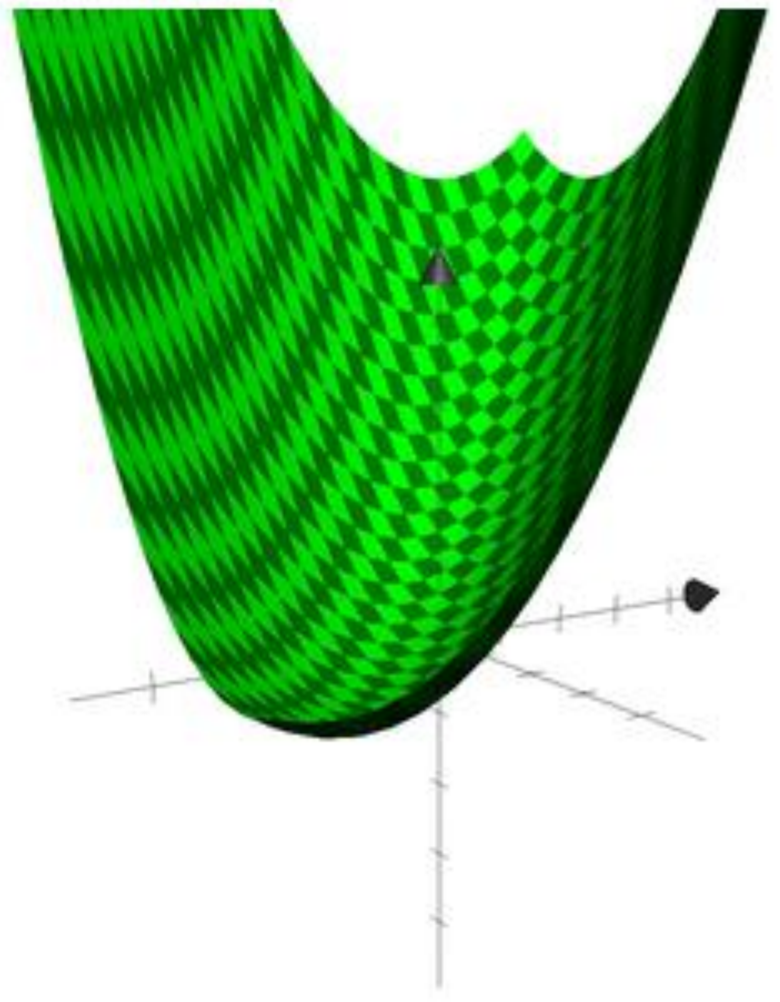
Non-Diagonal Preconditions

Previous methods use a diagonal preconditioner, which corresponds to choosing a diagonal reparameterization matrix A.

If A is diagonal, $\vec{\theta} = A\vec{\theta}' = \begin{pmatrix} a_{1,1}\theta_1 \\ \vdots \\ a_{n,n}\theta_n \end{pmatrix}$. So, the reparameterization can only scale along each axis.

In some cases, this is less than ideal.





Non-Diagonal Preconditions

Example: rotate the function $f(x,y) = 4x^2 + \frac{(y+3)^2}{15}$ by 45 degrees.

We apply the transformation

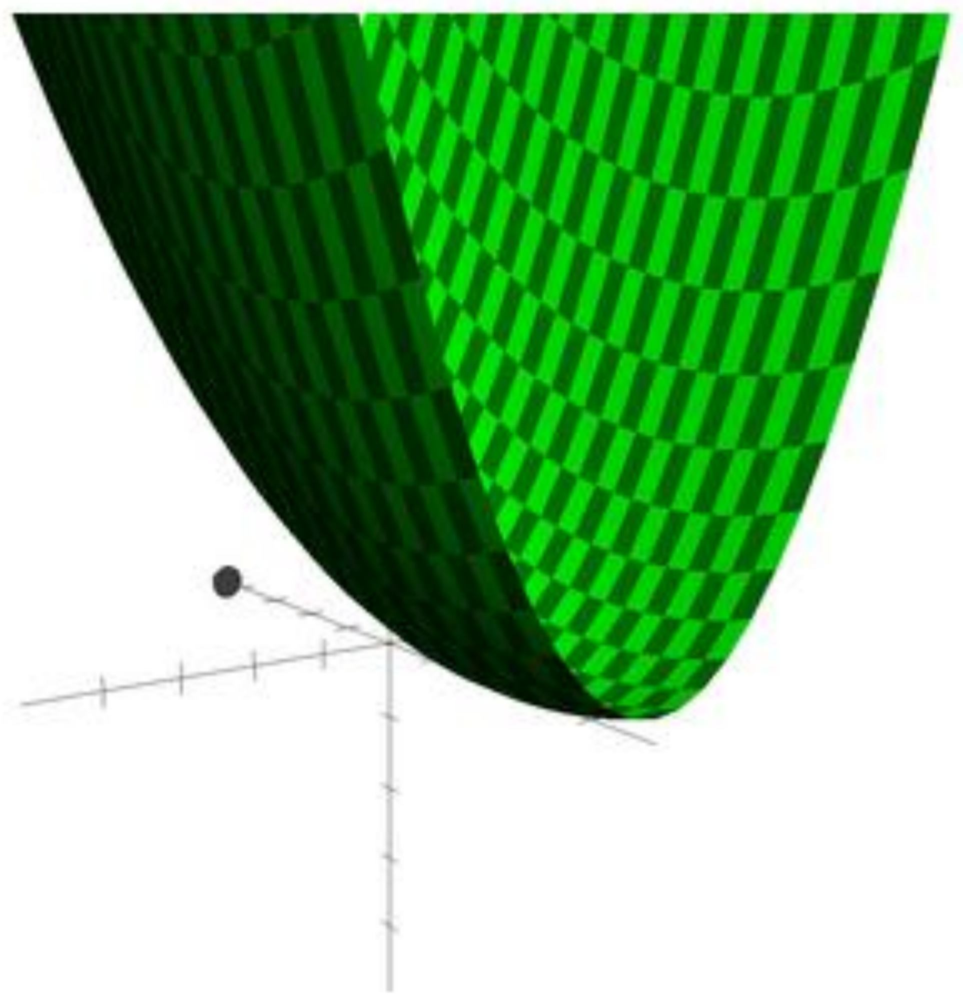
$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x/\sqrt{2} - y/\sqrt{2} \\ x/\sqrt{2} + y/\sqrt{2} \end{pmatrix}$$

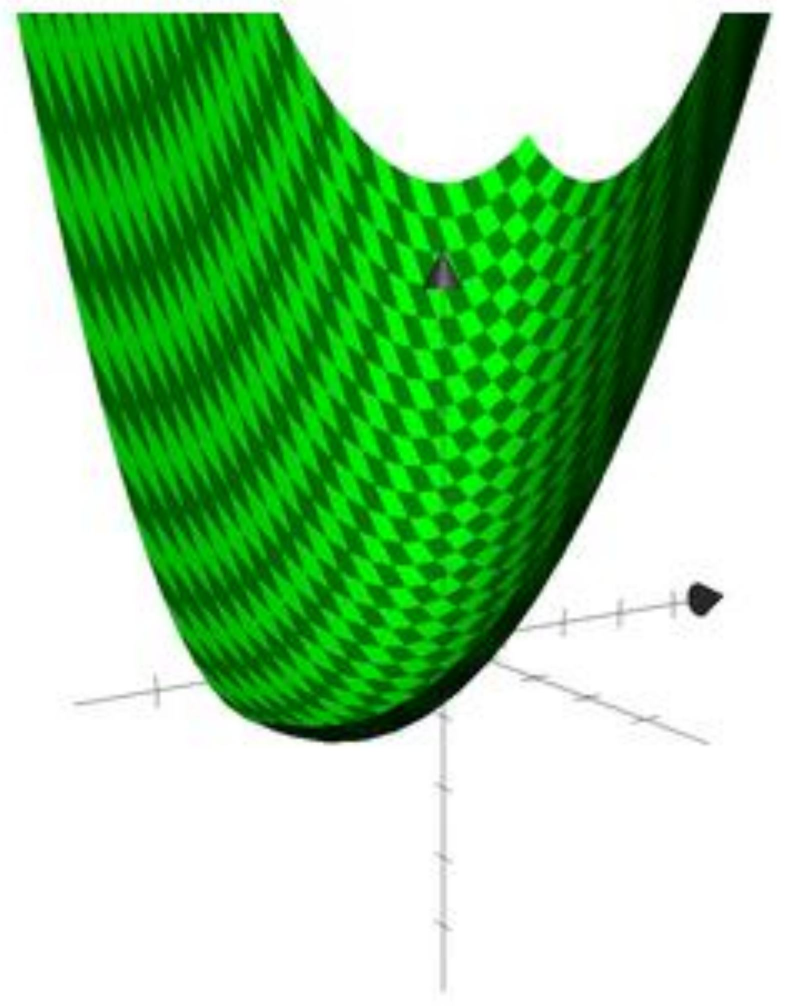
$$f(x,y) = 4\left(x/\sqrt{2} - y/\sqrt{2}\right)^2 + \frac{\left(x/\sqrt{2} + y/\sqrt{2} + 3\right)^2}{15}$$

$$= 2(x-y)^2 + \frac{\left(x + y + 3\sqrt{2}\right)^2}{30}$$

Both x and y are in both terms, cannot make the coefficient in each term 1 by scaling.

Instead, we can reparameterize using a non-diagonal matrix A.





Newton’s Method

Idea: Use inverse Hessian as preconditioner.

Known as a **second-order optimization method** because it uses the Hessian (from the second-order term in the Taylor expansion) in addition to the gradient (from the first-order term in the Taylor expansion).

Methods that only use the gradient are known as **first-order methods**.

Gradient Descent:

```
$\vec{\theta}^{(0)} \leftarrow$ random vector
for $t = 1,2,3,\ldots$
    $\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} - \gamma_t \frac{\partial L}{\partial \vec{\theta}}(\vec{\theta}^{(t-1)})$
    if algorithm has converged
        return $\vec{\theta}^{(t)}$
```

Newton’s Method:

```
$\vec{\theta}^{(0)} \leftarrow$ random vector
for $t = 1,2,3,\ldots$
    $H \leftarrow \frac{\partial^2 L}{\partial \vec{\theta} \partial \vec{\theta}^{\top}}(\vec{\theta}^{(t-1)})$
    $\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} - \gamma_t H^{-1} \frac{\partial L}{\partial \vec{\theta}}(\vec{\theta}^{(t-1)})$
    if algorithm has converged
        return $\vec{\theta}^{(t)}$
```

Newton’s Method

Benefits:

Invariant to invertible linear reparameterizations.

When initialized sufficiently close to a local minimum, Newton’s method will converge very quickly at a quadratic rate (that is, as the number of iterations t increases, the gap between the local minimum and the objective value decreases at a rate of $O\left(e^{-e^{t}}\right)$.

Newton’s Method

Drawbacks:

May not try to minimize the objective function - may actually maximize it!

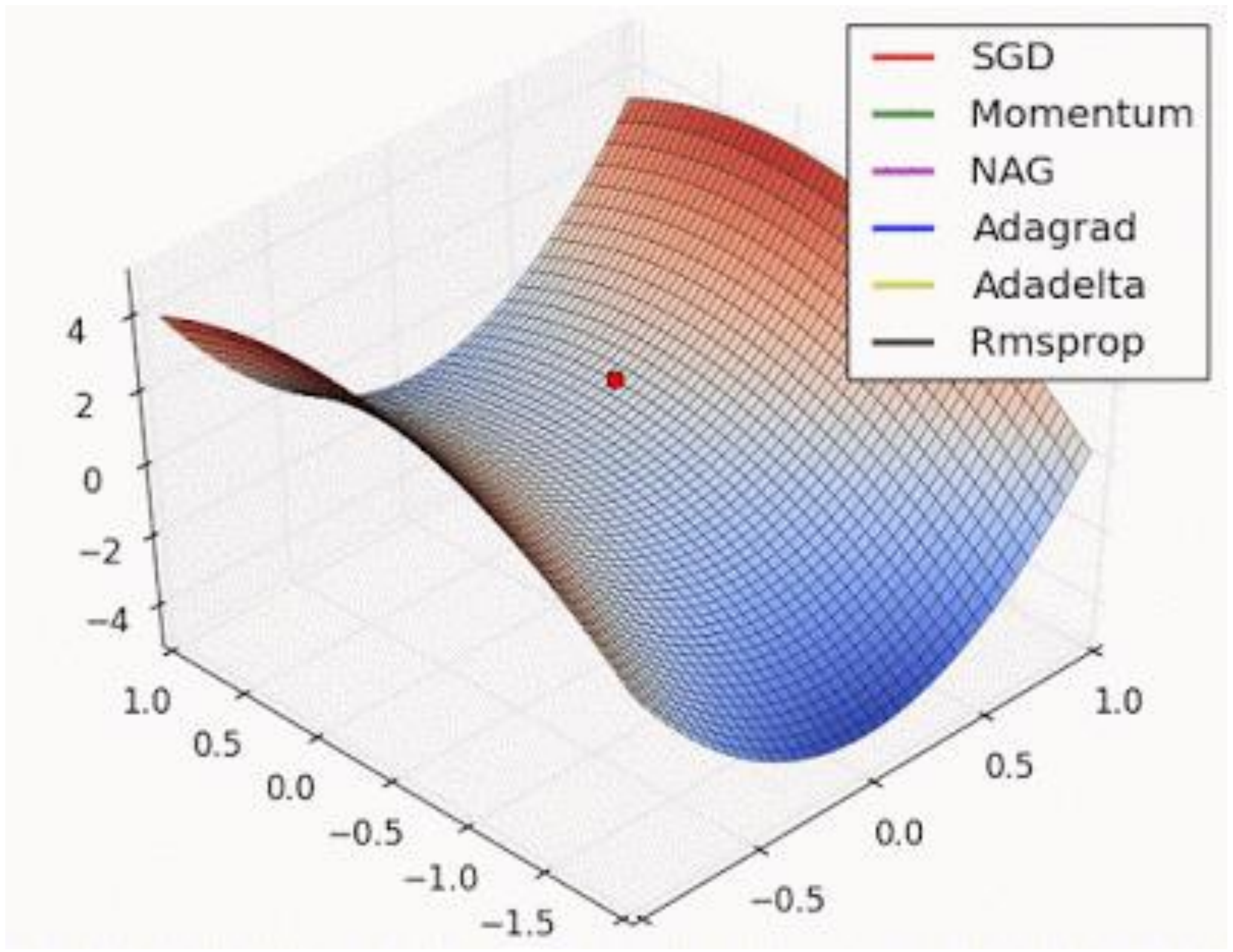
- Recall the update formula: $\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} - \gamma_t H^{-1} \frac{\partial L}{\partial \vec{\theta}} \left(\vec{\theta}^{(t-1)}\right)$
- If the Hessian is negative definite, will move in the opposite direction from the direction of steepest descent.

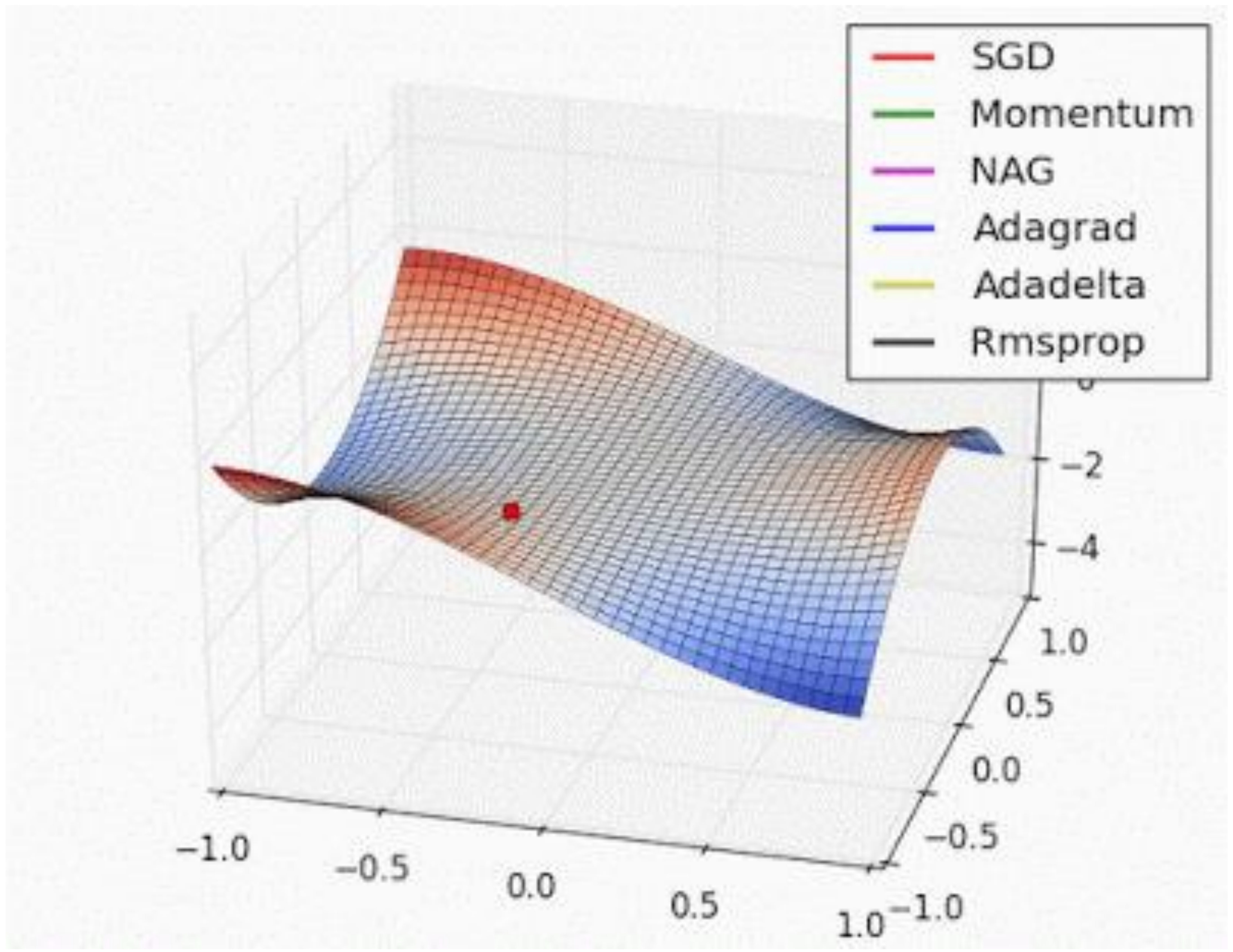
When not initialized close to a local minimum, may diverge!

- If the Hessian has small eigenvalues, the inverse Hessian will have large eigenvalues, causing the update to blow up.

The Hessian is very expensive (or even intractable) to compute because the number of entries is quadratic in the dimensions of the parameter vector.

Comparison of Optimization Algorithms



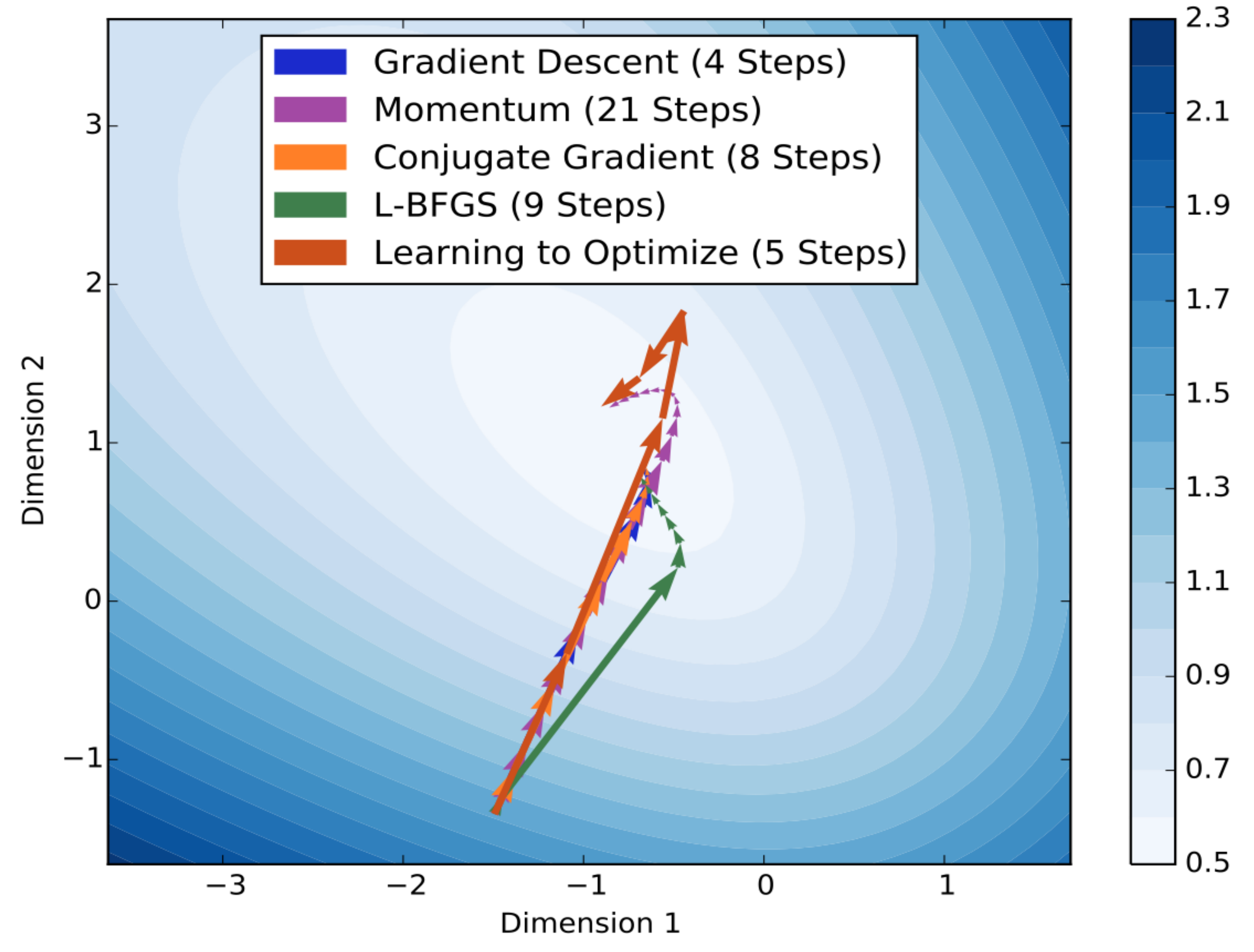


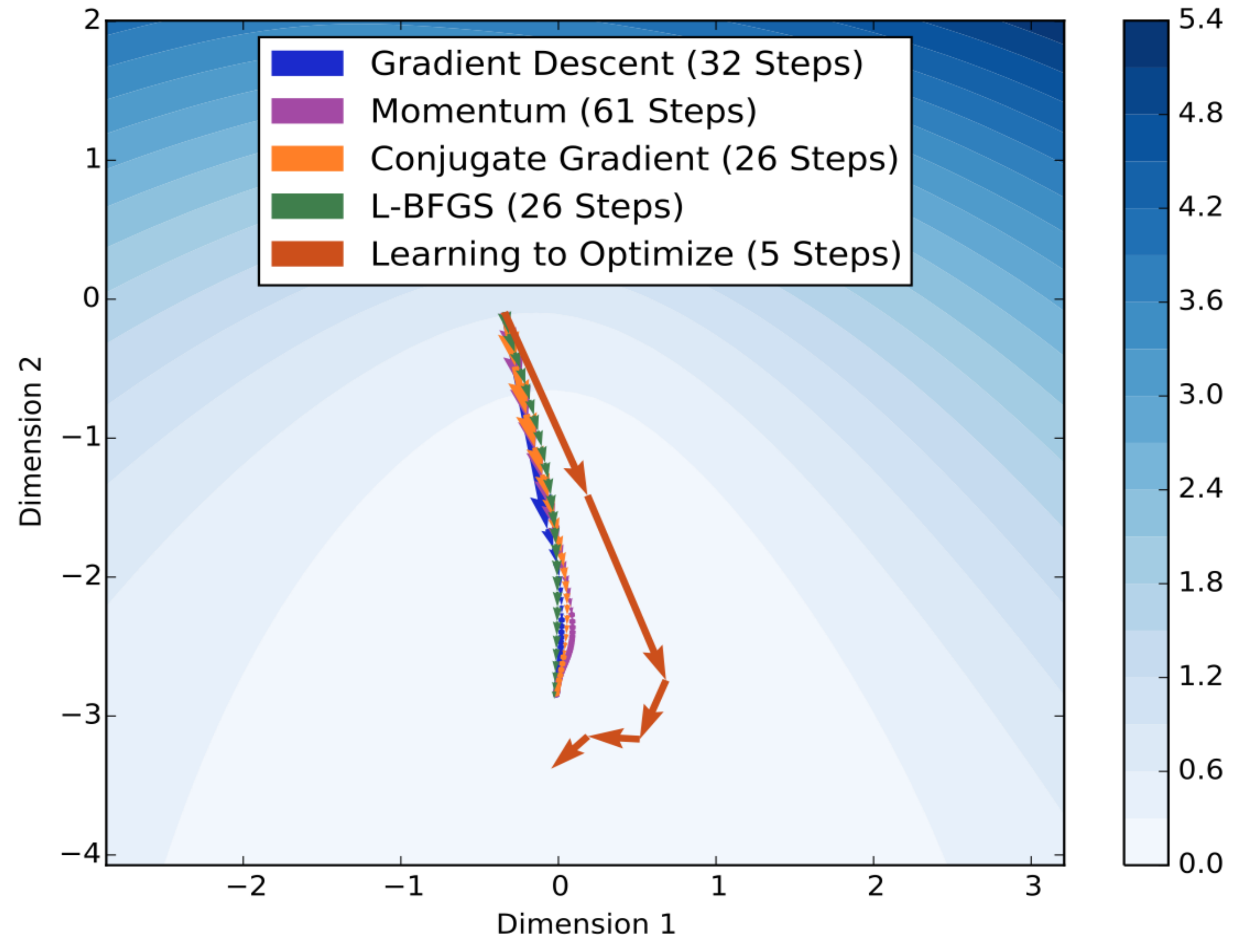
There is no single best optimization algorithm -
which one works best depends on the objective function.

Learning to Optimize

Possible to discover new optimization algorithms automatically using machine learning - one of Prof. Ke Li's research topics.

Idea: Replace the update formula with a model, and then find the parameters of the model that would make the optimization algorithm converge the fastest. Effectively *learning* the update formula from data on optimizing functions.





"Learning to Optimize", Ke Li, Jitendra Malik, arXiv:1606.01885, 2016 and ICLR, 2017

Stochastic Optimization

Also known as "stochastic approximation" when used in broader contexts, such as root finding.

Recall the optimization problem we would like to solve:

$$\vec{\theta}^* = \arg\min_{\vec{\theta}} L(\vec{\theta}), \text{where } L(\vec{\theta}) = -\log \mathcal{L}(\vec{\theta}, \sigma; \mathcal{D}) = \sum_{i=1}^{N} \left(y_i - f(\vec{x}_i; \vec{\theta})\right)^2 \coloneqq \sum_{i=1}^{N} L_i(\vec{\theta})$$

In each iteration of these optimization algorithms, we need to compute the gradient $\frac{\partial L}{\partial \vec{\theta}}\left(\vec{\theta}^{(t-1)}\right)$:

$$\frac{\partial L}{\partial \vec{\theta}}\left(\vec{\theta}^{(t-1)}\right) = \sum_{i=1}^{N} \frac{\partial L_i}{\partial \vec{\theta}}\left(\vec{\theta}^{(t-1)}\right)$$

This involves summing N terms. If the dataset is large, computing this is expensive.

Stochastic Optimization

Key Idea: Instead of computing all the terms, we can compute a random subset of terms and add them together. This can be viewed as a noisy version of the true gradient, which is in expectation the same as the true gradient.

Consider the extreme case of just sampling one term:
Let i' be a discrete uniform RV from 1 to N inclusive, i.e.: $p(i' = k) = \frac{1}{N}$ for all $k \in \{1, \dots, N\}$.

$$E_{i'}\left[N \cdot \frac{\partial L_{i'}}{\partial \vec{\theta}}(\vec{\theta}^{(t-1)})\right] = N \cdot E\left[\frac{\partial L_{i'}}{\partial \vec{\theta}}\left(\vec{\theta}^{(t-1)}\right)\right] = N \sum_{k=1}^{N} p(i' = k) \frac{\partial L_k}{\partial \vec{\theta}}\left(\vec{\theta}^{(t-1)}\right)$$

$$= N \sum_{k=1}^{N} \frac{1}{N} \frac{\partial L_k}{\partial \vec{\theta}}\left(\vec{\theta}^{(t-1)}\right) = \sum_{k=1}^{N} \frac{\partial L_k}{\partial \vec{\theta}}\left(\vec{\theta}^{(t-1)}\right) = \frac{\partial L}{\partial \vec{\theta}}(\vec{\theta}^{(t-1)})$$

When the expectation of a random variable is the same as a fixed variable, the random variable is said to be an **unbiased estimate** of the fixed variable. In this case, the subsampled gradient (known as a **gradient estimate**) is an unbiased estimate of the true gradient.

Stochastic Gradient Descent (SGD)

It turns out that we can replace the true gradient in gradient descent with the gradient estimate. The resulting algorithm is known as stochastic gradient descent (SGD).

Stochastic Gradient Descent (SGD):

```
$\vec{\theta}^{(0)} \leftarrow$ random vector
for $t = 1,2,3,\ldots$
    Sample $i'$ from discrete uniform distribution from 1 to $N$ inclusive
    $\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} - \gamma_t \frac{\partial L_{i'}}{\partial \vec{\theta}}\left(\vec{\theta}^{(t-1)}\right)$
    if algorithm has converged
        return $\vec{\theta}^{(t)}$
```

SGD vs. Gradient Descent

SGD comes with two benefits compared to vanilla gradient descent:

- Computing the gradient estimate is much cheaper
- Less prone to getting stuck at a local minimum or saddle point, so tends to converge faster on non-convex optimization problems

It does have a drawback:

- In the worst case, the number of iterations needed to achieve a given objective value is increased by a factor of N compared to vanilla gradient descent.

Terminology

(Full Batch) Gradient Descent: Computing the gradient on the entire dataset every iteration.

Mini-batch Gradient Descent: Computing the gradient on a subset of data points (known as a mini-batch) every iteration. (Sometimes also referred to as "stochastic gradient descent”)

Stochastic Gradient Descent: Computing the gradient on a single data point every iteration.

The smallest number of iterations needed to perform one pass over the dataset is known as an **epoch**. It is:
- 1 for full batch gradient descent
- N/M for mini-batch gradient descent where M is the size of the mini-batches
- N for stochastic gradient descent

In practice, mini-batch gradient descent tends to work the best.

Other Stochastic Algorithms

Stochastic optimization can be combined with algorithms other than gradient descent to yield stochastic versions of those algorithms.

For example, stochastic gradient descent with momentum, stochastic NAG, stochastic AdaGrad, stochastic Adam.

However, sometimes all these methods are referred to as stochastic gradient methods.

In modern machine learning, because the datasets are typically large, stochastic versions are often assumed to be used by default, and the word “stochastic” is often not explicitly mentioned.

While replacing true gradients with gradient estimates is typically fine for first-order methods, it often does not work well with second-order methods, e.g.: Newton’s method, and quasi-Newton methods like L-BFGS.

Practical Tips for Tuning Hyperparameters

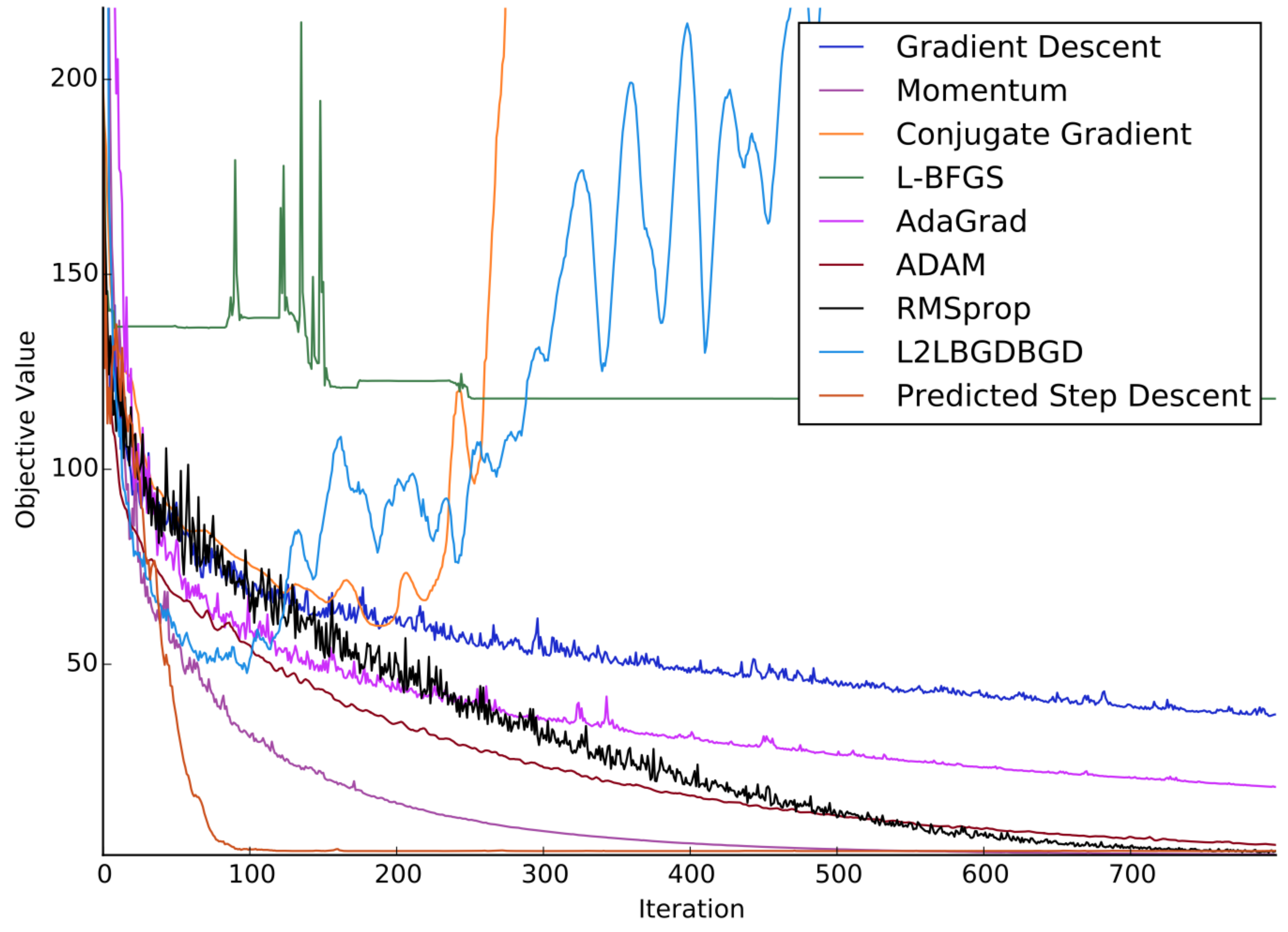
Iterative optimization algorithms have many hyperparameters!

- Step size / learning rate (in all methods): γ_t (possibly varying in each iteration)
  - Initial step size / learning rate
  - Learning rate schedule: constant, step / piecewise constant, inverse time, etc.
- Momentum parameter (in momentum and Adam): α
- Momentum parameter for the preconditioner (in Adam): β

Practical Tips for Tuning Hyperparameters

(Based on personal experience - your mileage may vary.)

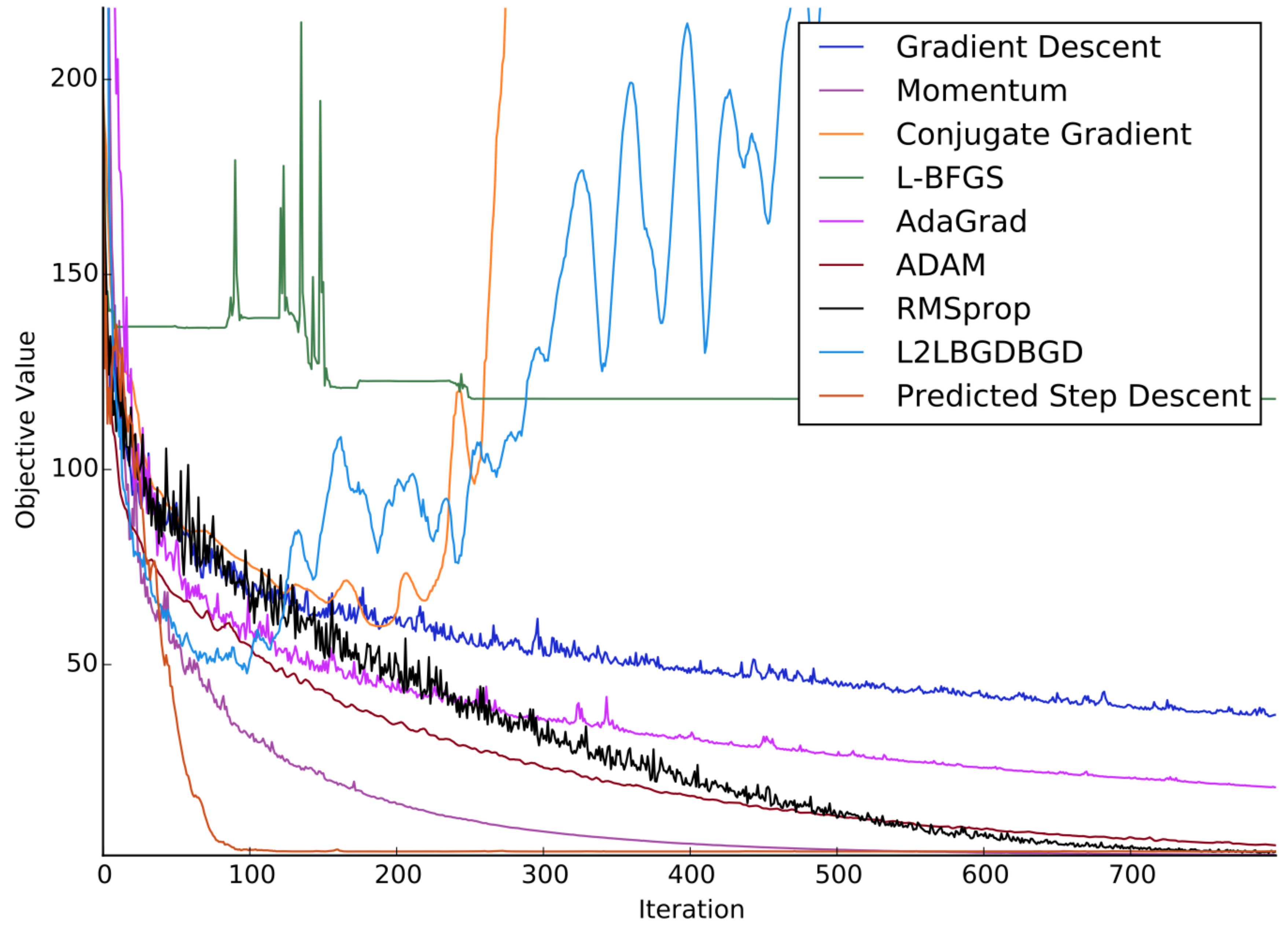
- Step size / learning rate (in all methods): γ_t
  - Start with a constant learning rate schedule, and try different values uniformly sampled on a logarithmic scale, e.g.: 0.1, 0.01, 0.001, etc.
  - If objective value keeps going up or oscillates without going down, try a smaller learning rate
  - If objective value goes down very slowly, try a larger learning rate



Practical Tips for Tuning Hyperparameters

- If objective value goes down steadily but plateaus at a later iteration, try a step / piecewise constant learning rate schedule that decreases the step size after the point in time at which the objective value plateaus

- If the parameter vector is low-dimensional and you are using a stochastic method, try an inverse time learning rate schedule ($\gamma_t = \gamma_0 / t$). Backed by theory - look up “Robbins-Monro sequence”.



Practical Tips for Tuning Hyperparameters

- Momentum parameter (in momentum and Adam): α
    - Typically 0.9 works well. Consider increasing to 0.99 or 0.999 if mini-batch size (number of data points in a mini-batch) is small.
- Momentum parameter for the preconditioner (in Adam): β
    - Typically 0.999 works well.