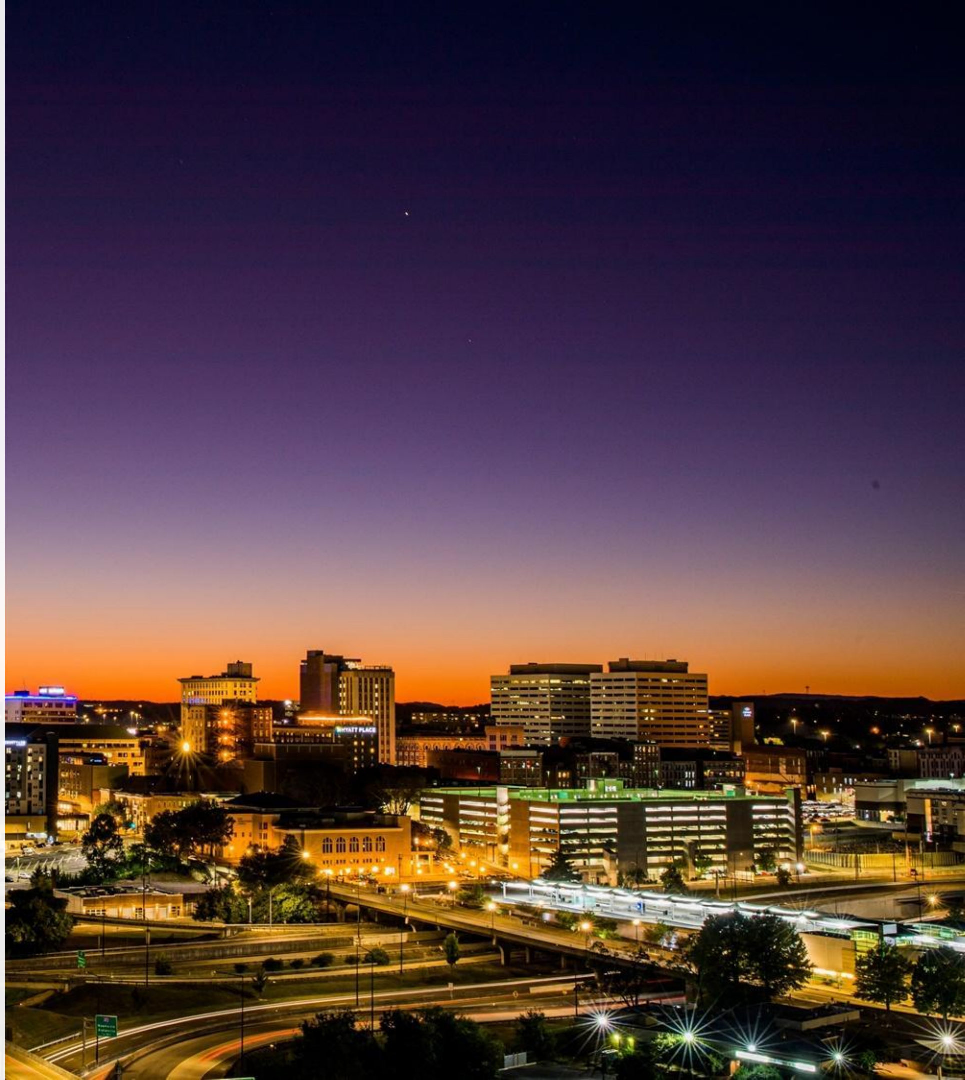
Optimization

Christal, Layal, Leanto

THE UNIVERSITY OF TENNESSEE KNOXVILLE

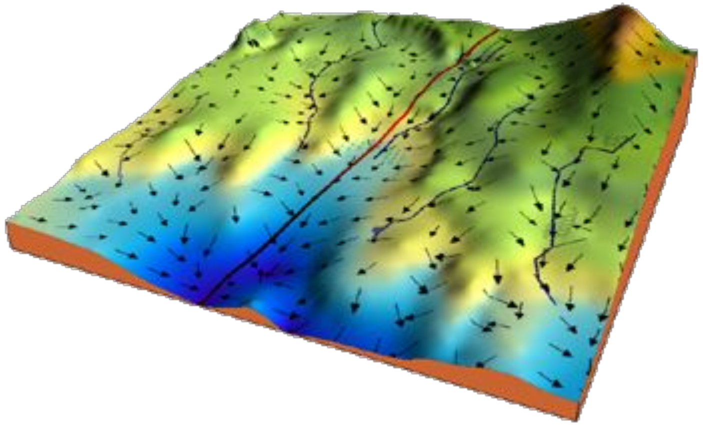


Optimization

- Motivation
- Gradient Descent
- Demo 1
- Gradient Descent with Momentum
- Adaptive Moment Estimation (Adam)
- Demo 2
- Stochastic Gradient Descent
- Demo 3

Motivation

Consider a loss space given in the context of some cost function

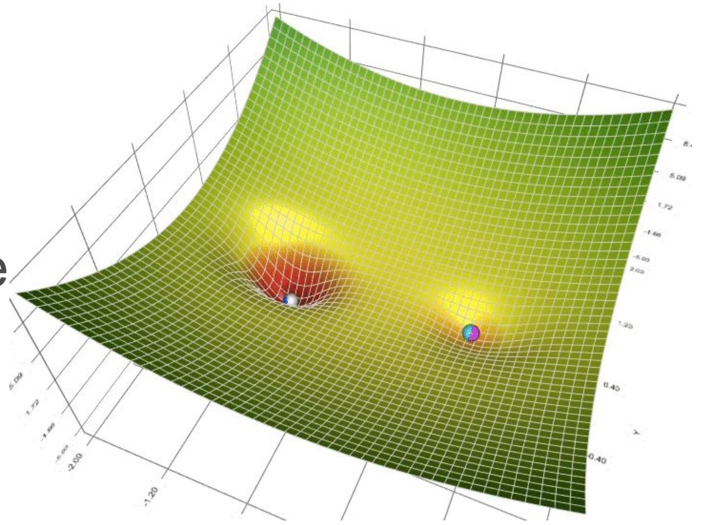


Our goal is to move from the high loss area to the low loss ‘solution’

- We start from some randomly initialized location and take steps in the direction of lower cost iteratively

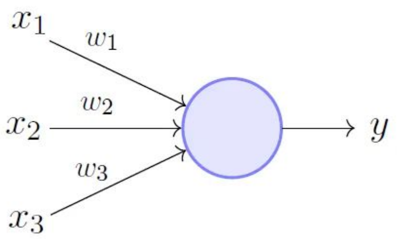
- It aims to minimize (or maximize) a given objective function by iteratively adjusting the parameters of a model or function

- Gradient descent (GD) based optimization algorithms are efficient and easily scalable depending on the size and scale of the datasets



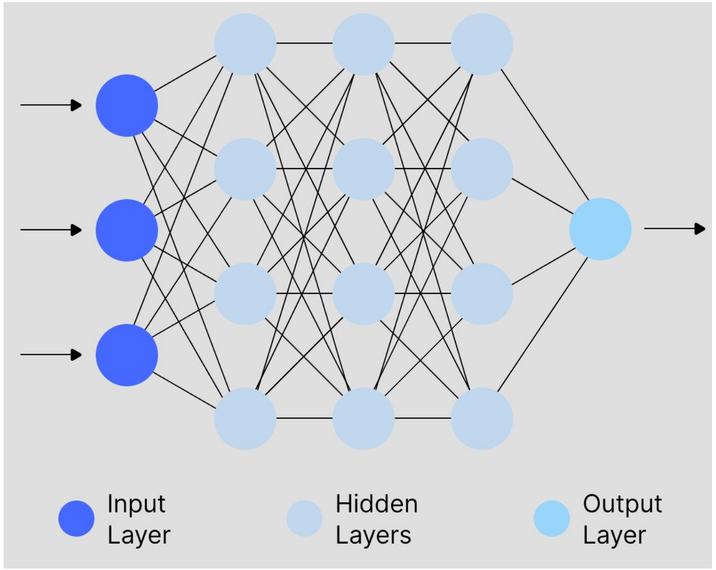
Perceptrons and Neural Networks

Neural networks can be conceptualized as function approximators



$$\hat{Y} = f(X; W)$$

Learn to approximate complex functions by adjusting their parameters W



Applications

- Gradient descent based algorithms are widely used in Machine Learning and Deep Learning
- Used in optimization problems in engineering, finance, logistics, and telecommunications
- Denoising, compression, and feature extraction in signal processing

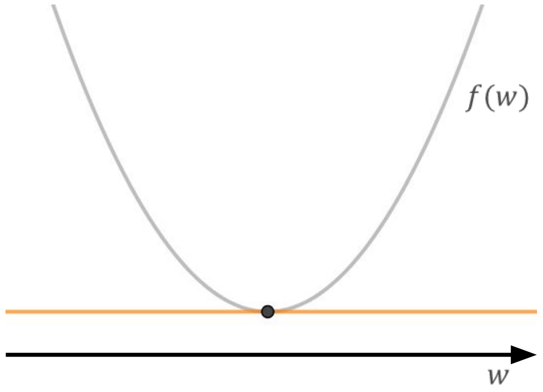
Gradient Descent

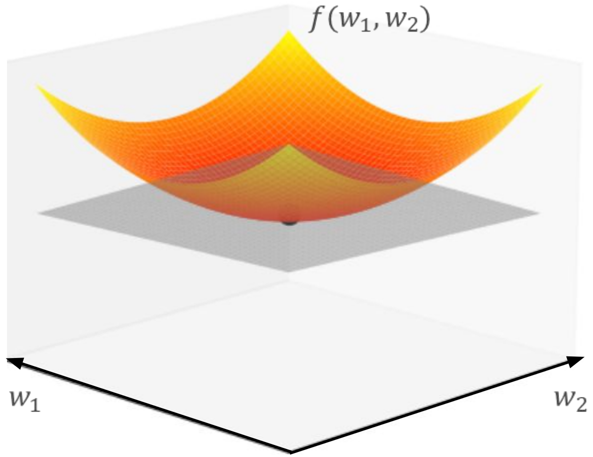
Minimize a Cost or a Loss Function

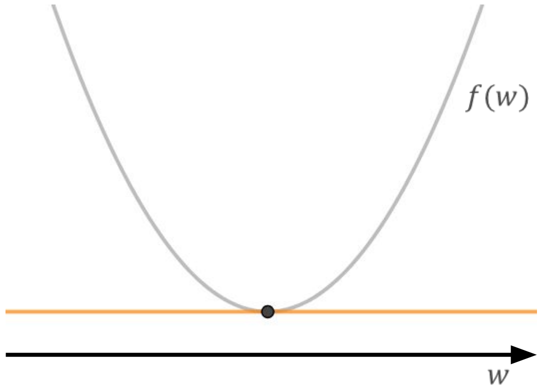
Mean Squared Error (MSE) $$C = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Binary Cross-Entropy Loss: $$-\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

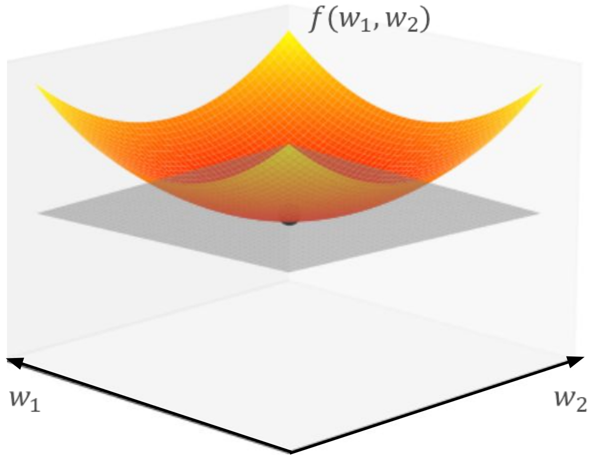
Categorical Cross-Entropy Loss: $$-\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij})$$


$f(w)$
w


$f(w_1, w_2)$
w_1
w_2



$$\frac{d}{dw} f(w) = 0$$



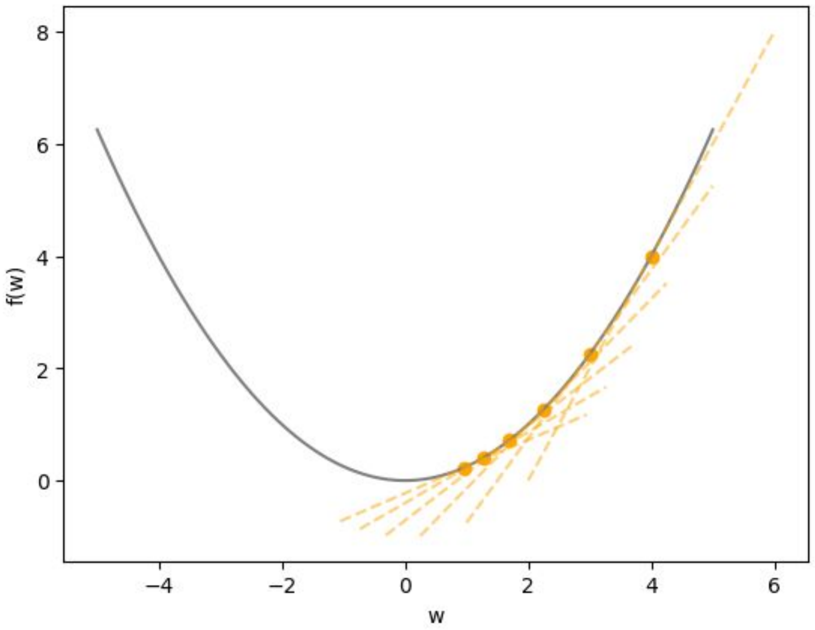
$$\frac{\partial}{\partial w_1} f(\mathbf{w}) = 0$$
$$\frac{\partial}{\partial w_2} f(\mathbf{w}) = 0$$

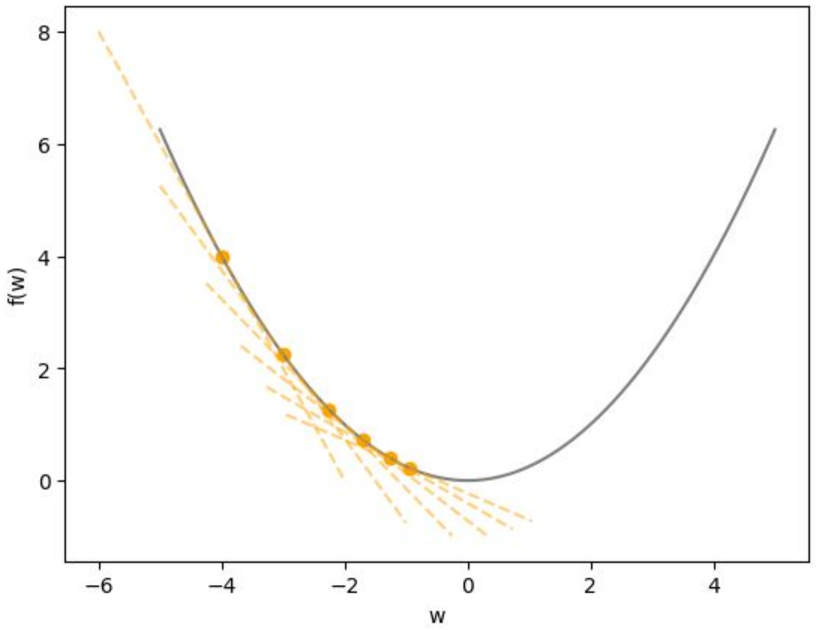
Multiple dimensions: $\boldsymbol{w} = (w_1, , w_2, \ldots, w_N)^T$

Gradient

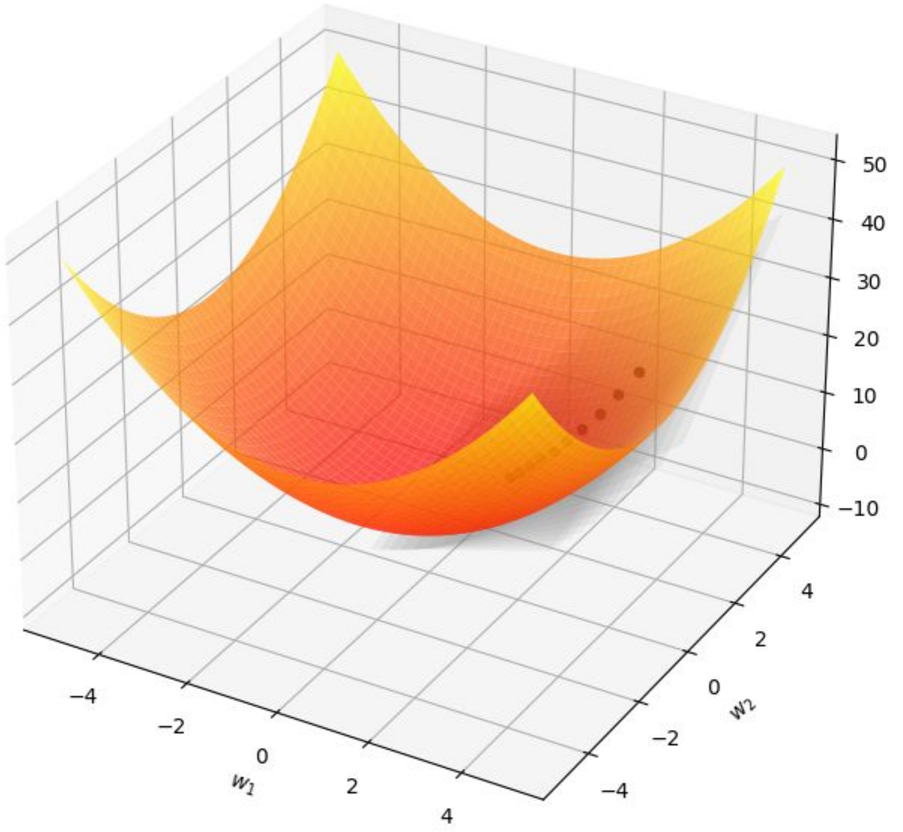
$$\frac{\partial}{\partial w_1} f(\boldsymbol{w}) = 0$$
$$\frac{\partial}{\partial w_2} f(\boldsymbol{w}) = 0$$
$$\vdots$$
$$\frac{\partial}{\partial w_N} f(\boldsymbol{w}) = 0$$

$$\nabla f(\mathbf{w}) = \mathbf{0}$$


f(w)
w
8
6
4
2
0
−4
−2
0
2
4
6


f(w)
w
8
6
4
2
0
−6
−4
−2
0
2
4

f(*w*0) decreases fastest if one moves from w0 in the direction of the negative gradient of *f* at *w*0.



Taylor's Theorem

$$f(\mathbf{w} + \Delta\mathbf{w}) = f(\mathbf{w}) + \nabla_w f \cdot \Delta\mathbf{w} + O(|\Delta\mathbf{w}|^2)$$

$$\Delta f = f(\mathbf{w} + \Delta\mathbf{w}) - f(\mathbf{w})$$

Goal: $\Delta f \leq 0$

Taylor's Theorem

$$f(\mathbf{w} + \Delta\mathbf{w}) = f(\mathbf{w}) + \nabla_w f \cdot \Delta\mathbf{w} + O(|\Delta\mathbf{w}|^2)$$

$$\Delta f = f(\mathbf{w} + \Delta\mathbf{w}) - f(\mathbf{w})$$

Goal: $\Delta f \leq 0$

$$\Delta\mathbf{w} = -\eta \nabla_w f$$

$\eta > 0$

learning rate

Taylor's Theorem

$$f(\mathbf{w} + \Delta\mathbf{w}) = f(\mathbf{w}) + \nabla_w f \cdot \Delta\mathbf{w} + O(|\Delta\mathbf{w}|^2)$$

$$\Delta f = f(\mathbf{w} + \Delta\mathbf{w}) - f(\mathbf{w})$$

Goal: $\Delta f \leq 0$

$$\Delta f \approx \nabla_w f \cdot (-\eta \nabla_w f)$$
$$= -\eta \| \nabla_w f \|^2$$
$$\leq 0$$

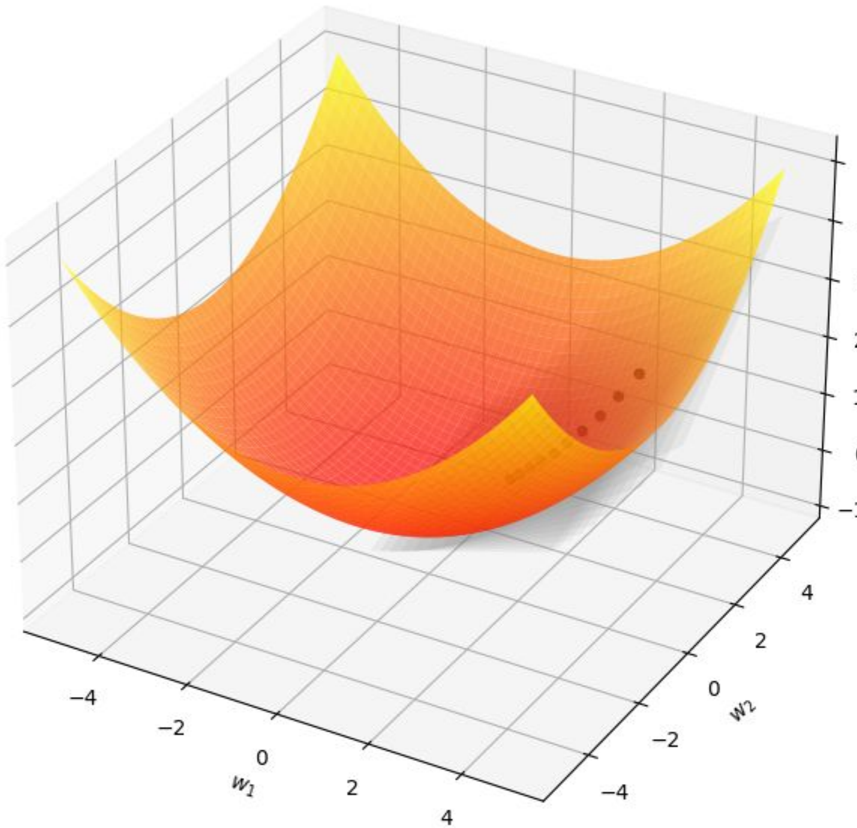
$$\Delta\mathbf{w} = -\eta \nabla_w f$$

$\eta > 0$
learning rate

Iterations

Initialize $\mathbf{w}_0$

$$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \eta \nabla_w f(\mathbf{w}_k)$$



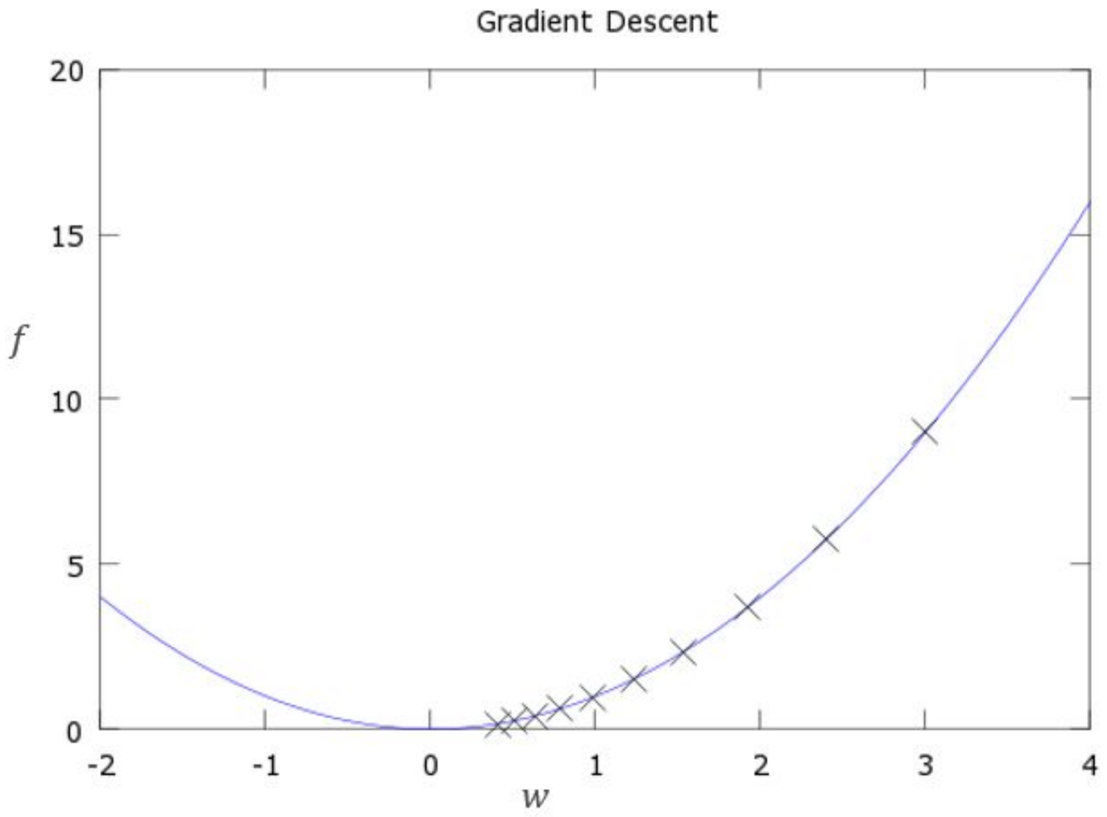
Gradient Descent Minimization - Single Variable Example

- Let’s work with a simple loss function $f(w) = w^2$
- We update the parameter based on the update rule

$$w \rightarrow w - \eta \frac{d}{dw} f(\mathrm{w})$$

$$\frac{d}{dw} f(\mathrm{w}) = 2\mathrm{w}$$

If we set $\eta = 0.1, \quad w_0 = 3$. After 10 iterations:



Iterations	w	Update Step
1	3	0.6
2	2.4	0.48
3	1.92	0.384
4	1.536	0.307
5	1.229	0.246
6	0.983	0.197
7	0.786	0.157
8	0.629	0.126
9	0.503	0.101
10	0.403	0.081

Minimize a Cost or a Loss Function

Mean Squared Error (MSE)

$$\mathcal{C} = \frac{1}{2N}\sum_{n=1}^{N}\left(F^{(n)} - y^{(n)}\right)^2 = \frac{1}{2N}||\vec{F} - \vec{y}||^2$$

How to choose η ?

Too small: huge computational cost. Very slow.

Too large: Steps will oscillate. Overshoot the minimum.
The algorithm becomes unstable.

How to choose η ?

Common choices:

- Use a fixed η value for each step. For simplicity, choose 10^{γ}, where γ is a negative integer.
- Use a diminishing steplength like $\eta = \frac{1}{k}$ at step k.

Non Convex Functions

Run it several times with different initializations and/or learning rates.

When does gradient descent stop?

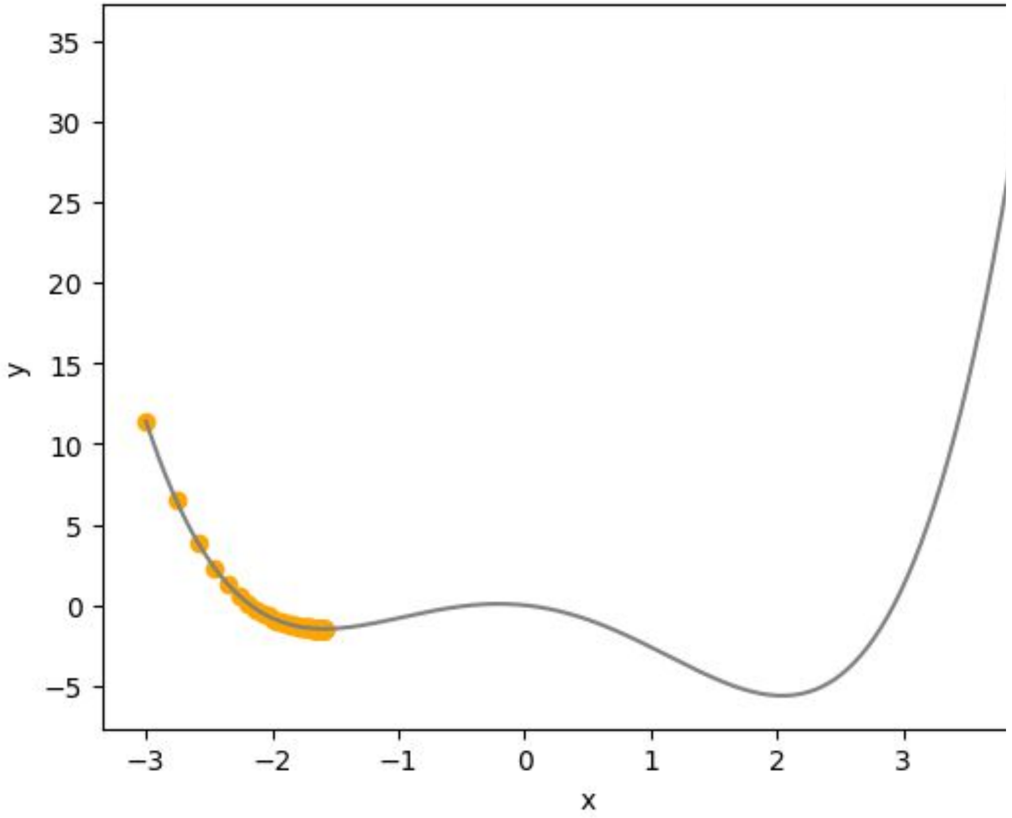
- Halt when steps no longer make sufficient progress or when corresponding evaluations no longer differ substantially (set a threshold).

- Assign a fixed number of maximum iterations.

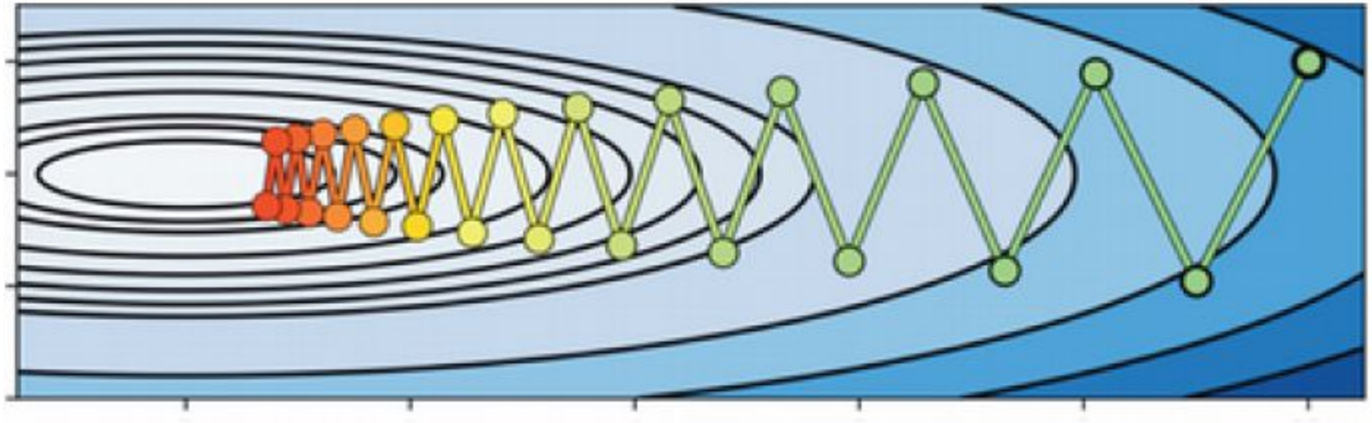
Weakness

The magnitude of the negative gradient can vanish rapidly near stationary points, leading gradient descent to slowly crawl near minima and saddle points. This prevents the algorithm from finding the global minimum.

$$\|\mathbf{w}_{\mathrm{k}} - \mathbf{w}_{\mathrm{k}-1}\|_2 = \eta \|\nabla f(\mathbf{w}_{\mathrm{k}-1})\|_2$$



Weakness



The direction of the negative gradient can rapidly oscillate producing zig-zagging steps that take considerable time to reach a minimum point.

Demo

https://github.com/cbaillar/Optimization

Gradient Descent with Momentum

Why use momentum?

- Escape local minima and saddle points
- Aids in faster convergence by reducing oscillations
- Smooths out weight updates for stability
- Reduces model complexity and prevents overfitting
- Can be used in combination with other optimization algorithms for improved performance

Adding momentum

We add information about the past gradient to add ‘momentum’ to our update step, so the new update step becomes

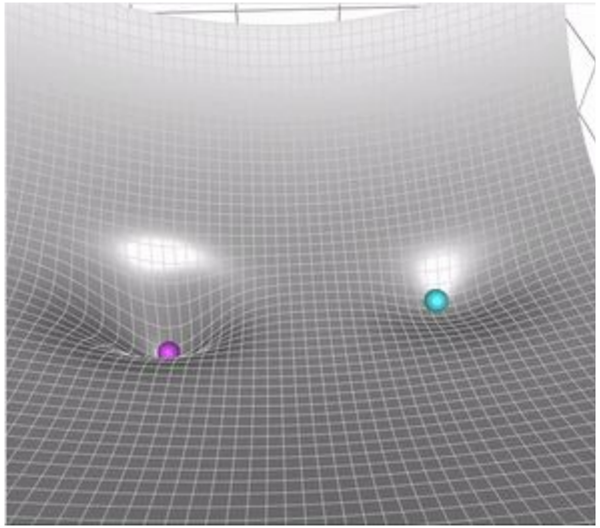
$$\mathbf{v}_k \leftarrow \beta \mathbf{v}_{k-1} + (1-\beta)\nabla_w f(\mathbf{w}_k)$$

β is the momentum coefficient $\in [0, 1]$

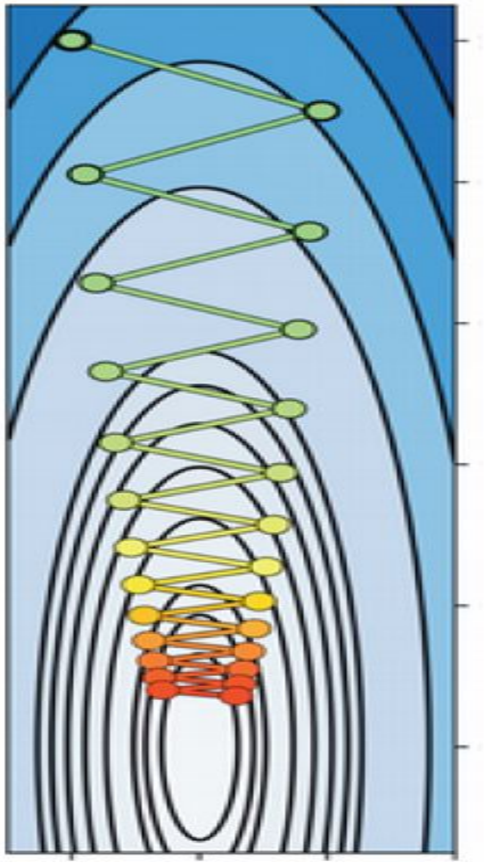
and the the new weights are:

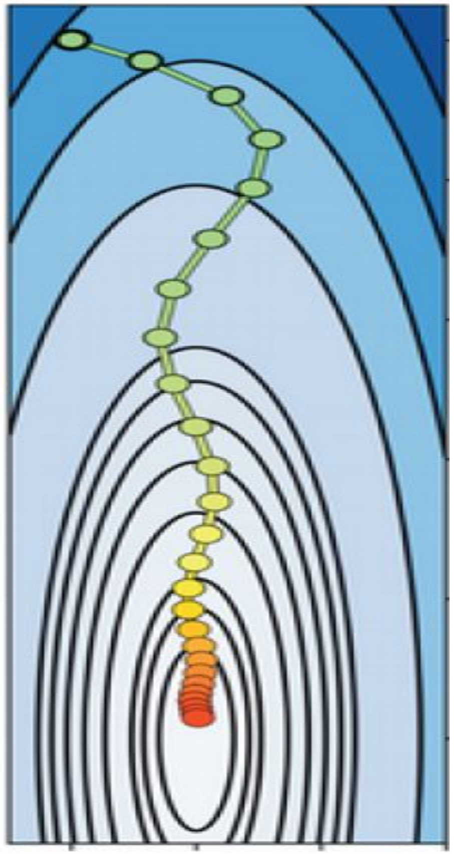
$$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \eta \mathbf{v}_k$$

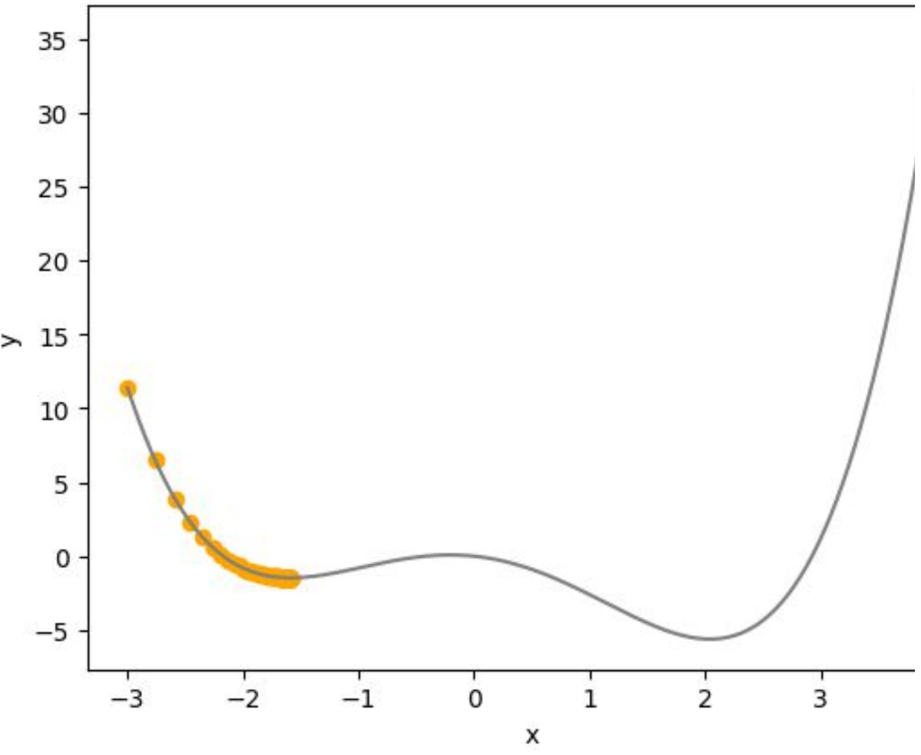
Imagine rolling a ball down a hill



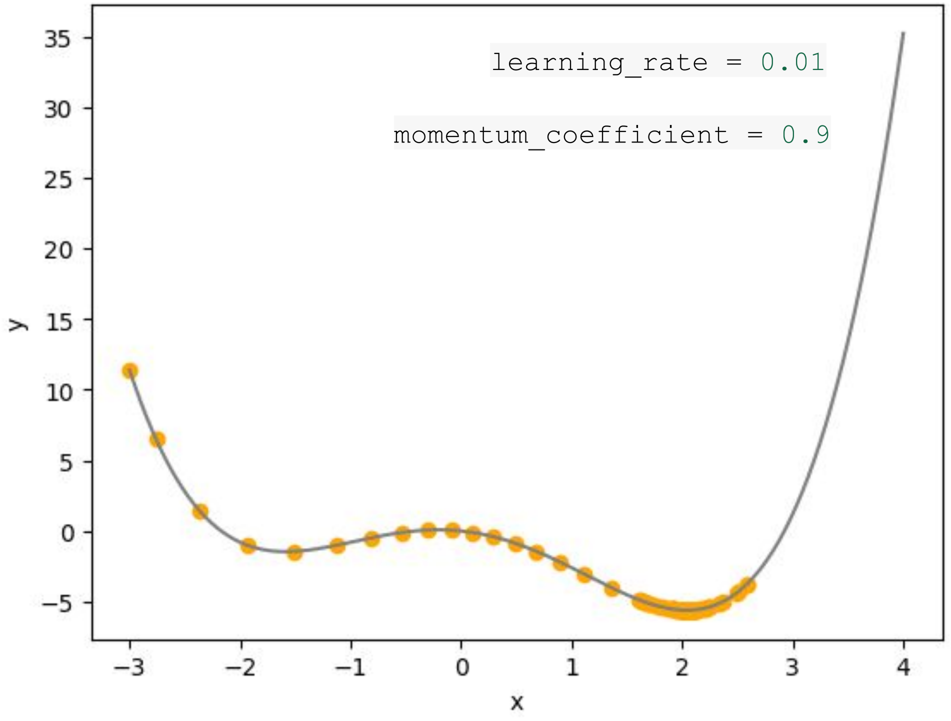
- The velocity accumulates the gradients over time with exponentially decaying weights controlled by the momentum coefficient β
- This accumulation introduces a smoothing effect on the optimization path and allows the algorithm to maintain momentum in promising directions


Without momentum


With momentum



Without momentum



```
learning_rate = 0.01
momentum_coefficient = 0.9
```

With momentum

Adaptive Moment Estimation (Adam)

Idea: Adjust the learning rates to achieve faster convergence.

Calculate exponential averages for both the descent direction and its magnitude.

Result: Fast convergence. Efficiency. Bias correction.

$$\mathbf{g}_k = \nabla_w f(\mathbf{w}_k)$$

The first-moment vector at step *k.*

$$\mathbf{m}_k = \beta_1 \mathbf{m}_{k-1} + (1 - \beta_1)\mathbf{g}_k$$

$\beta 1$ is the exponential decay rate for the first moment estimates (0.9).

The second-moment vector at step *k.*

$$\mathbf{v}_k = \beta_2 \mathbf{v}_{k-1} + (1 - \beta_2)\mathbf{g}_k^2$$

$\beta 2$ is the exponential decay rate for the second-moment estimates (0.999).

First Moment (Mean): Tracks the average of the gradients over time.

Second Moment (Variance): Tracks the average of the squared gradients over time.

$$\mathbf{g}_k = \nabla_w f(\mathbf{w}_k)$$

The first-moment vector at step *k.* $$\mathbf{m}_k = \beta_1 \mathbf{m}_{k-1} + (1 - \beta_1)\mathbf{g}_k$$

The second-moment vector at step *k.* $$\mathbf{v}_k = \beta_2 \mathbf{v}_{k-1} + (1 - \beta_2)\mathbf{g}_k^2$$

Correct the bias in the moments.

$$\hat{\mathbf{m}}_k = \frac{\mathbf{m}_k}{1 - (\beta_1)^k} \qquad \hat{\mathbf{v}}_k = \frac{\mathbf{v}_k}{1 - (\beta_2)^k}$$

Since m and v are initialized to 0, they are biased toward 0

leading to more stable and efficient optimization

$$\mathbf{g}_k = \nabla_w f(\mathbf{w}_k)$$

The first-moment vector at step *k.*

$$\mathbf{m}_k = \beta_1 \mathbf{m}_{k-1} + (1-\beta_1)\mathbf{g}_k$$

The second-moment vector at step *k.*

$$\mathbf{v}_k = \beta_2 \mathbf{v}_{k-1} + (1-\beta_2)\mathbf{g}_k^2$$

Correct the bias in the moments.

$$\hat{\mathbf{m}}_k = \frac{\mathbf{m}_k}{1-(\beta_1)^k} \qquad \hat{\mathbf{v}}_k = \frac{\mathbf{v}_k}{1-(\beta_2)^k}$$

Update of the model parameters.

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \frac{\hat{\mathbf{m}}_k}{\sqrt{\hat{\mathbf{v}}_k} + \epsilon}$$

$$\eta = 10^{-3}$$
$$\epsilon = 10^{-8}$$

ϵ is a small scalar added to prevent division by zero and maintain numerical stability.

For simple functions, a larger starting learning rate such as 0.1 is preferred.

Demo

https://github.com/cbaillar/Optimization

Stochastic Gradient Descent

Accelerating Training with Randomized Updates

- Addresses the computational inefficiency of traditional Gradient Descent methods when dealing with large datasets in machine learning projects
- Is a modification of gradient descent
- Gradient calculated using a random small part of the dataset instead of the whole
- Reduces computation time

Kinds of Stochastic Gradient Descent

Online SGD:

- Calculate gradient and update weights for each data sample
- Can help find the global minimum, especially if the objective function is convex
- Computationally expensive

Batch SGD:

- Compromise between GD and Online SGD
- Gradients are calculate and weights updated based on a subset of the data rather than individual samples or the whole dataset
- Faster than Online SGD while still introducing stochasticity

Batch Gradient Descent:

Whole data $\longrightarrow$ costly operation if the set is large

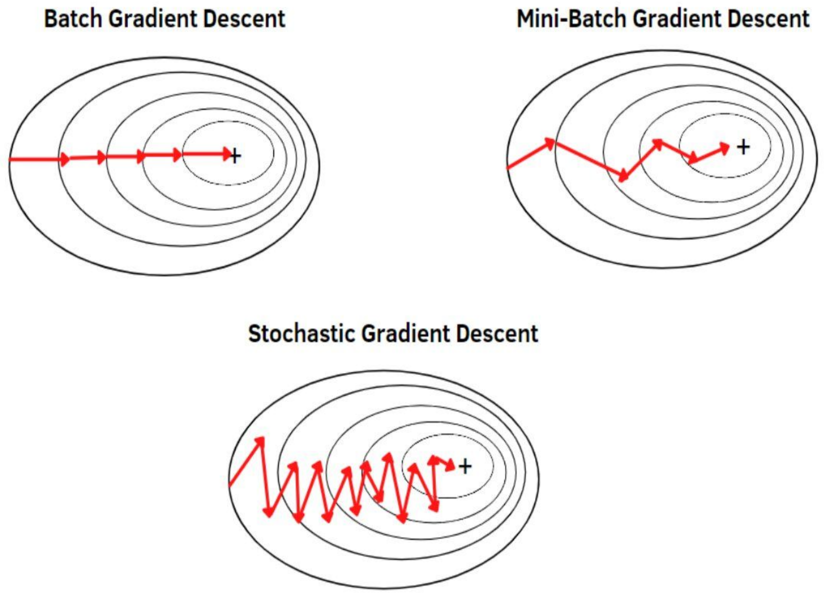
Stochastic Gradient Descent:

Minibatch or subset of data $\longrightarrow$ faster convergence

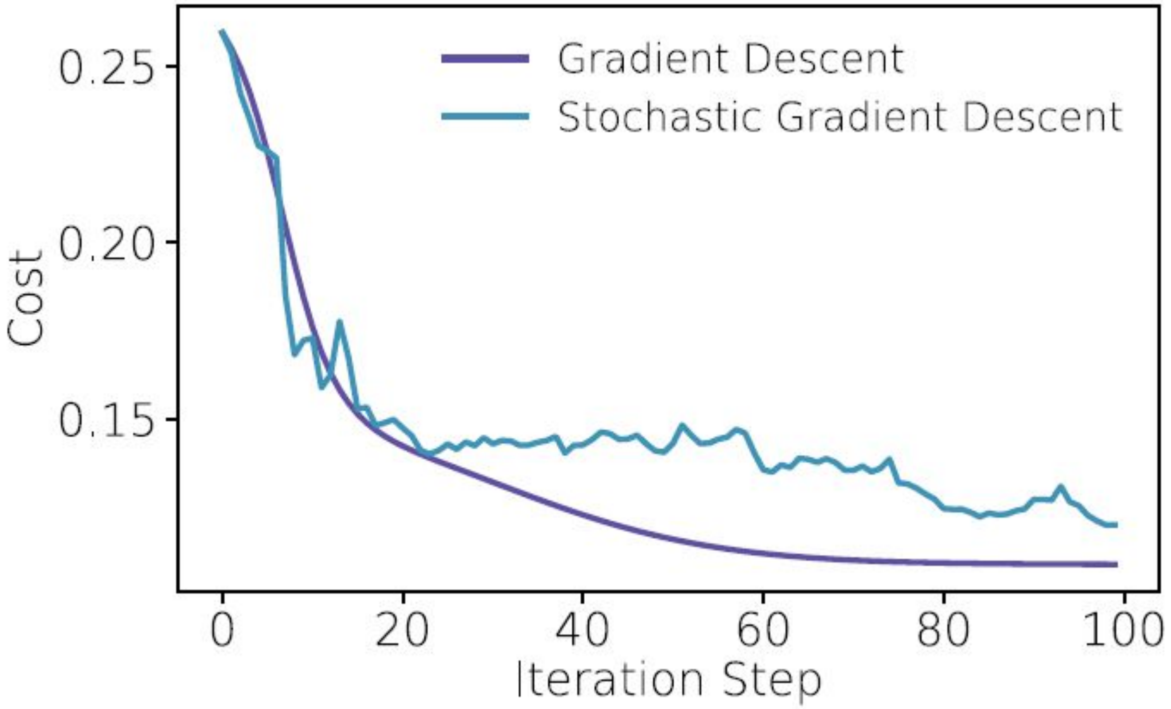
- Mini-Batch GD is a compromise between Batch GD and Online SGD

- Suitable for large datasets that cannot fit into memory

- Stochasticity helps prevent overfitting by introducing noise



It may never “converge” to the minimum, and the parameters will keep oscillating around the minimum; but in practice most of the values near the minimum will be reasonably good approximations to the true minimum



Demo

https://github.com/cbaillar/Optimization

Watt, J., Borhani, R., & Katsaggelos, A. K. (2020). *Machine learning refined: Foundations, algorithms, and applications* (2nd ed.). Cambridge University Press.
Deisenroth, M.P., Faisal, A., & Ong, C.S. (2020). *Mathematics for Machine Learning*. Cambridge University Press

https://towardsdatascience.com/gradient-descent-with-momentum-59420f626c8f
https://optimization.cbe.cornell.edu/
https://towardsdatascience.com/the-math-behind-adam-optimizer-c41407efe59b
https://web.archive.org/web/20180618211933/http://cs229.stanford.edu/notes/cs229-notes1.pdf

Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html
https://mccormickml.com/2014/03/04/gradient-descent-derivation/
https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c
https://statusneo.com/efficientdl-mini-batch-gradient-descent-explained/