Gradient Descent

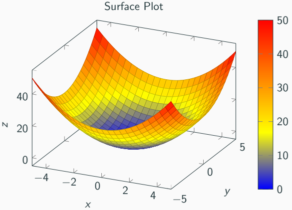
Nipun Batra

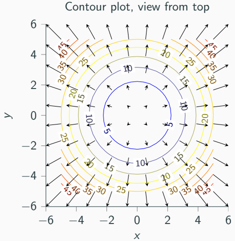
January 28, 2024

IIT Gandhinagar

Contour Plot And Gradients

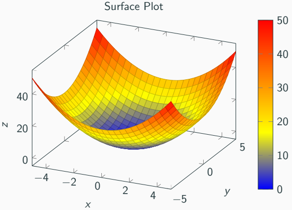
$$z = f(x, y) = x^2 + y^2$$

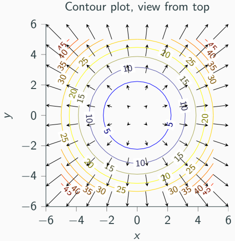




Contour Plot And Gradients

$z = f(x, y) = x^2 + y^2$

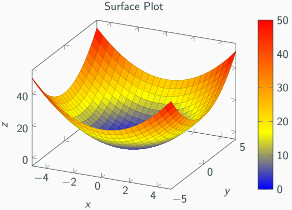


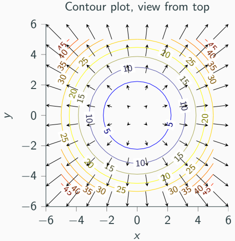


Gradient denotes the direction of steepest ascent or the direction in which there is a maximum increase in f(x,y)

Contour Plot And Gradients

$z = f(x, y) = x^2 + y^2$

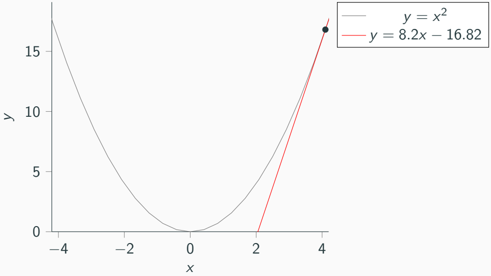




Gradient denotes the direction of steepest ascent or the direction in which there is a maximum increase in f(x,y)

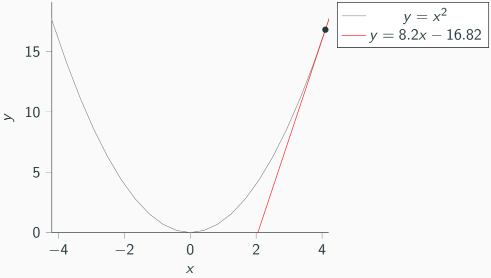
$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

Gradient Descent



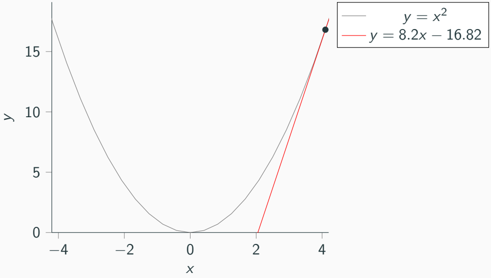
- $y = x^2$

Gradient Descent



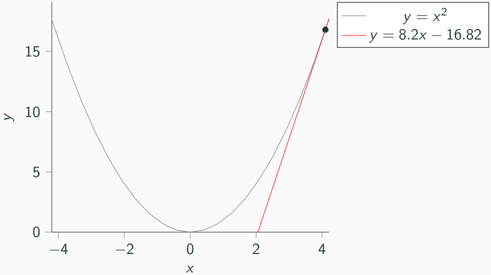
- $y = x^2$
- Key idea: Go to x_1 from x_0 such that $y_1 < y_0$

Gradient Descent



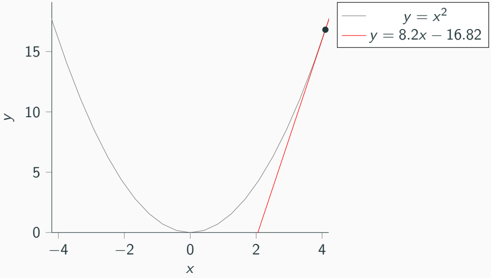
- $y = x^2$
- Key idea: Go to x_1 from x_0 such that $y_1 < y_0$
- $\frac{\partial y}{\partial x} = 2x$

Gradient Descent



- $y = x^2$
- Key idea: Go to x_1 from x_0 such that $y_1 < y_0$
- $\frac{\partial y}{\partial x} = 2x$
- At x = 4.1, we have max. decrease along the direction of -$\frac{\partial y}{\partial x}$

Gradient Descent



- $y = x^2$
- Key idea: Go to x_1 from x_0 such that $y_1 < y_0$
- $\frac{\partial y}{\partial x} = 2x$
- At x = 4.1, we have max. decrease along the direction of -$\frac{\partial y}{\partial x}$
- Equation of tangent at x=4.1: y = 8.2x-16.82

Gradient Descent

General Optimization Technique

Question: Find minimum y

Gradient Descent

General Optimization Technique

Question: Find minimum y

- Start with some x_0

Gradient Descent

General Optimization Technique

Question: Find minimum y

- Start with some x_0
- Till convergence or iterations exhahusted

Gradient Descent

General Optimization Technique

Question: Find minimum y

- Start with some x_0
- Till convergence or iterations exhahusted
  - $x_i = x_{i-1} - \alpha \frac{\partial y}{\partial x} x_{i-1}$

Gradient Descent

General Optimization Technique

Question: Find minimum y

- Start with some x_0
- Till convergence or iterations exhahusted
  - $x_i = x_{i-1} - \alpha \frac{\partial y}{\partial x} x_{i-1}$

Gradient Descent

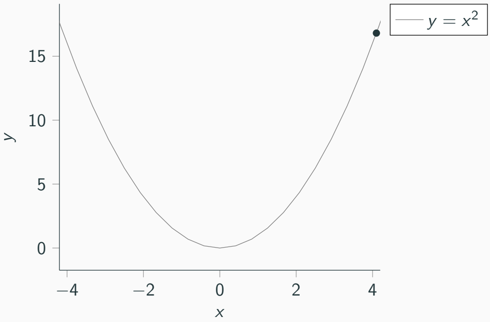
General Optimization Technique

Question: Find minimum y

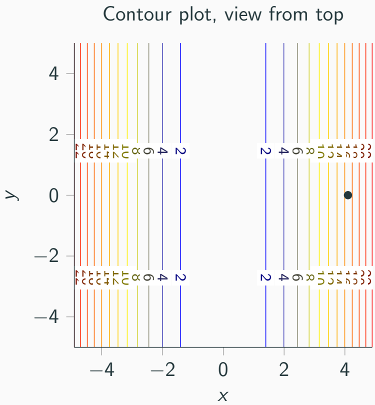
- Start with some x_0
- Till convergence or iterations exhahusted
  - $x_i = x_{i-1} - \alpha \frac{\partial y}{\partial x} x_{i-1}$

Here, α is the learning rate or step parameter

Iteration 0

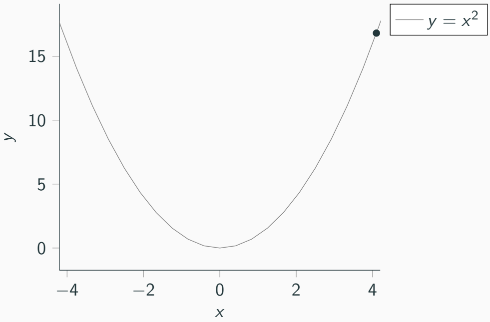


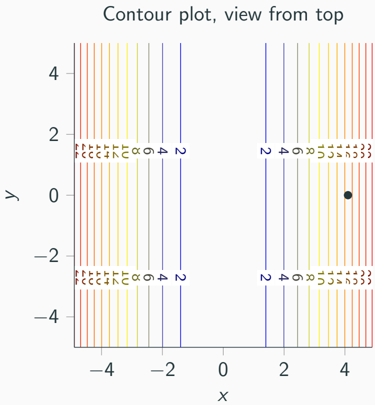
$y = x^2$
15
10
5
0
−4
−2
0
2
4
x
y



Contour plot, view from top
4
2
0
−2
−4
−4
−2
0
2
4
x
y

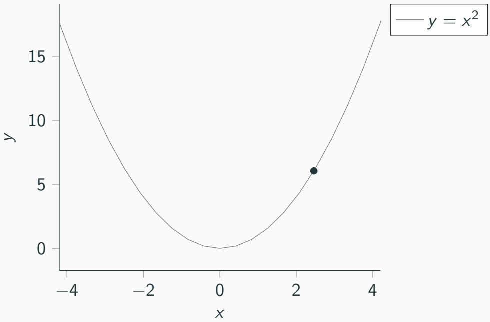
Iteration 0

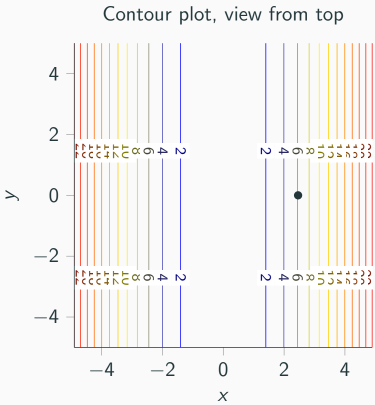




Let us start with initial x value of $x_0 = 4.1$ and learning rate $\alpha = 0.2$

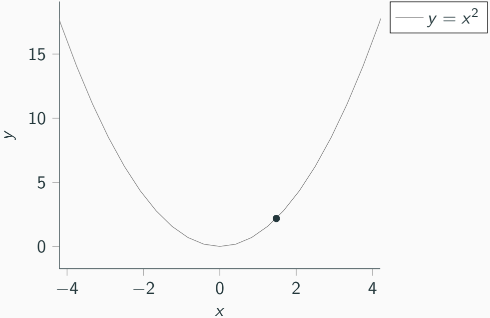
Iteration 1

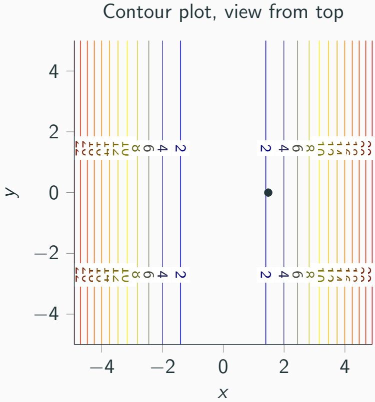




$$x = 4.1 - 0.2 \times 2 \times 4.1 = 2.46$$

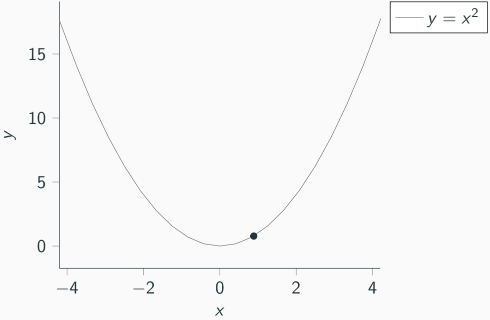
Iteration 2



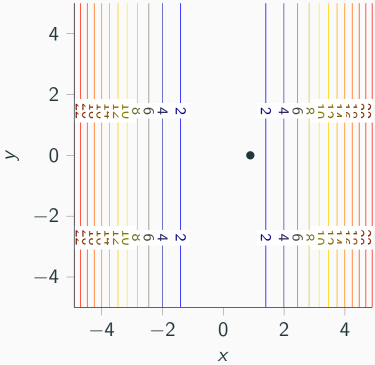


$$x = 2.46 - 0.2 \times 2 \times 2.46 = 1.48$$

Iteration 3

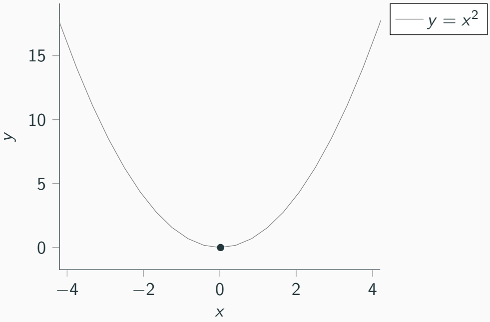


Contour plot, view from top

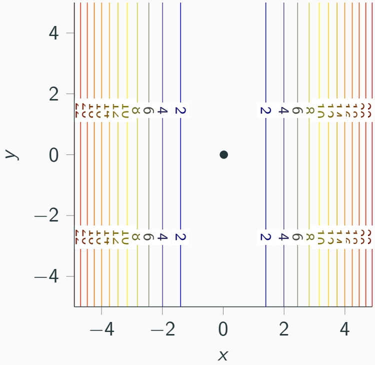


$$x = 1.48 - 0.2 \times 2 \times 1.48 = 0.89$$

Iteration 10



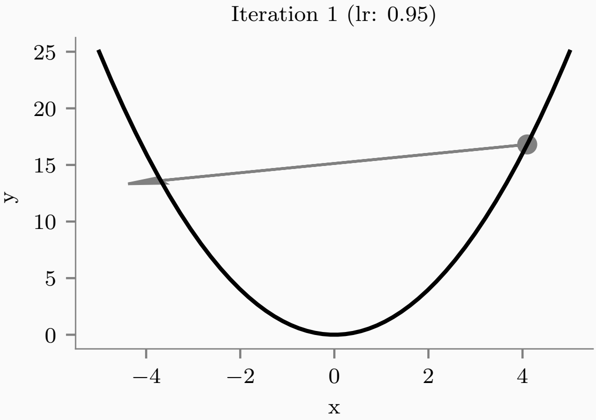
Contour plot, view from top



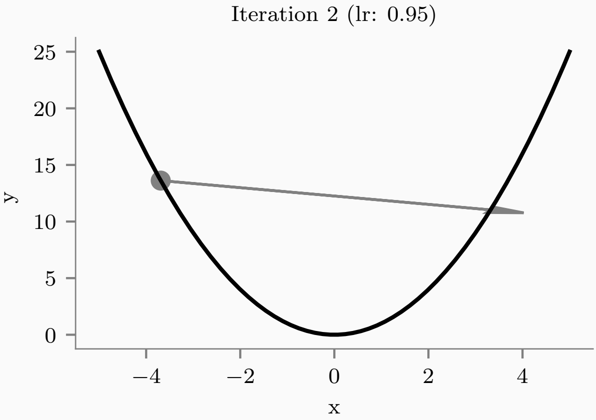
What if α is large?

The model starts overshooting!

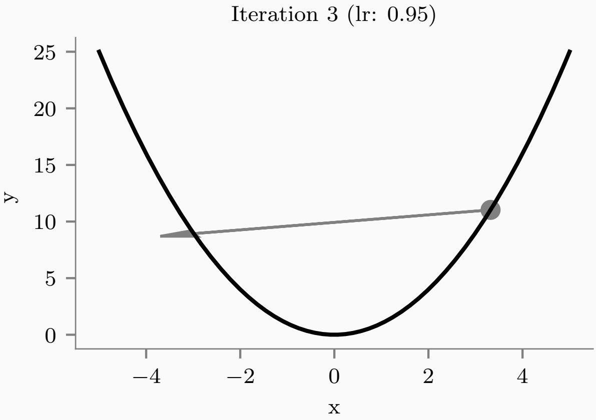
Overshooting



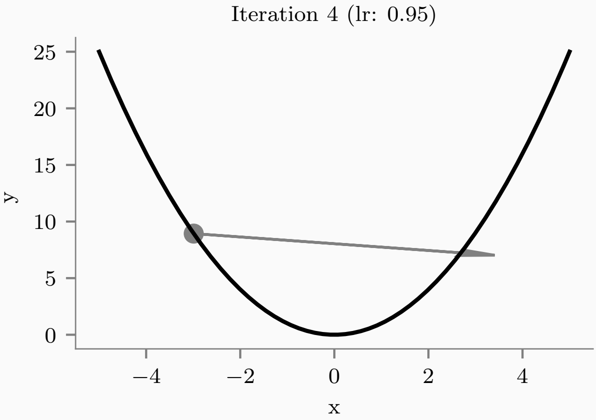
Overshooting



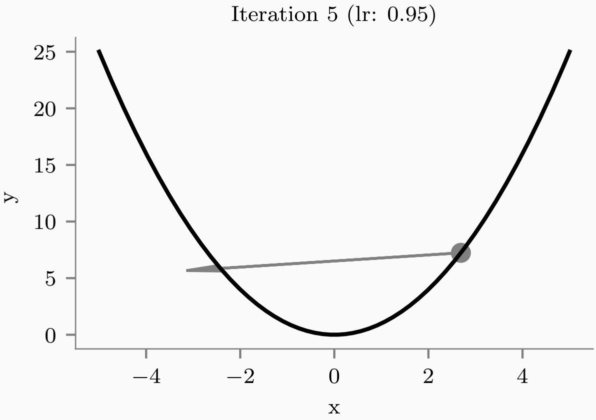
Overshooting



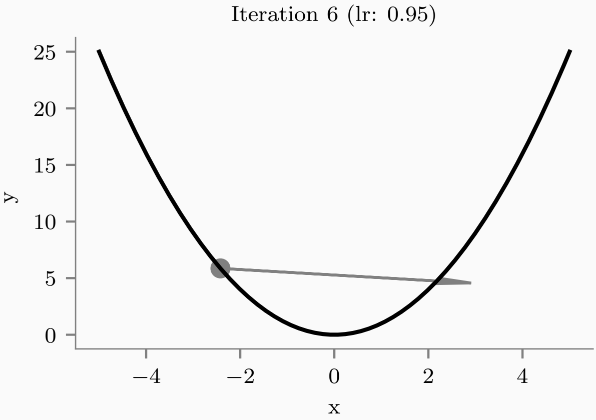
Overshooting



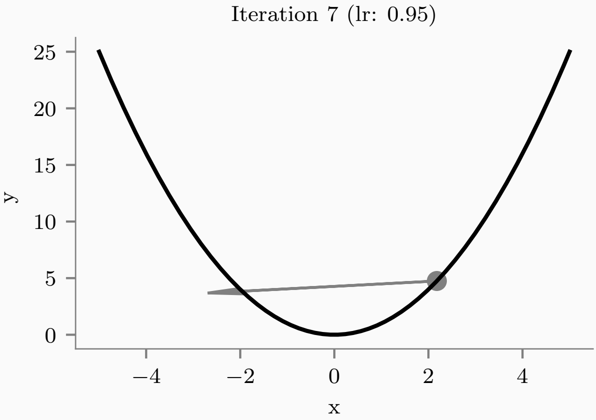
Overshooting



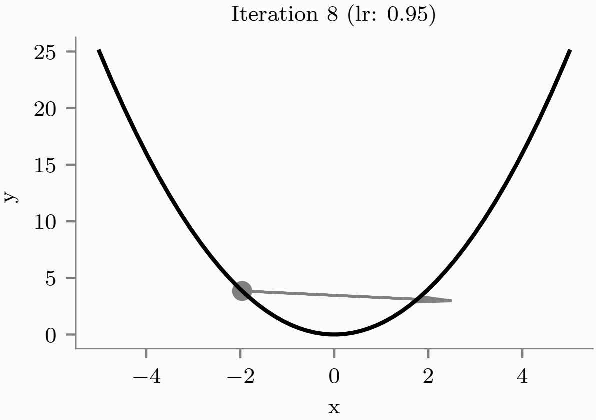
Overshooting



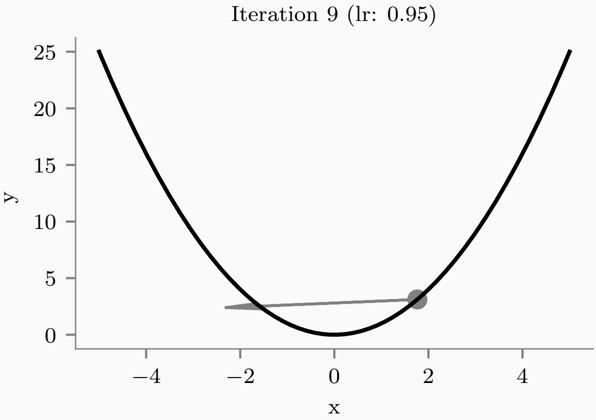
Overshooting



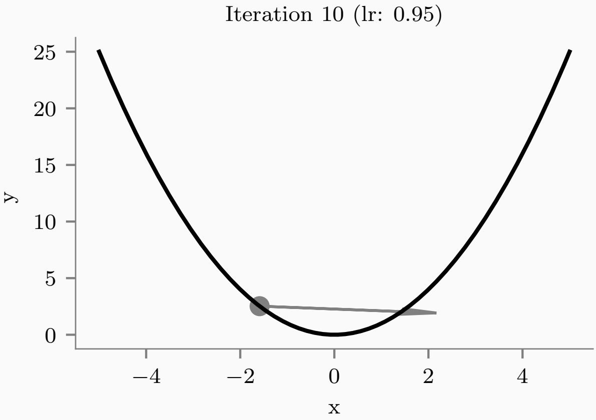
Overshooting



Overshooting



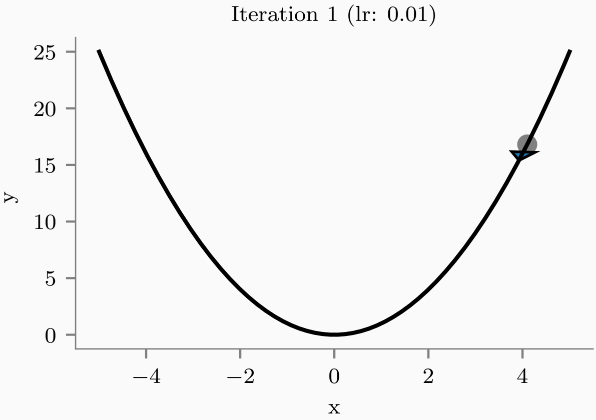
Overshooting



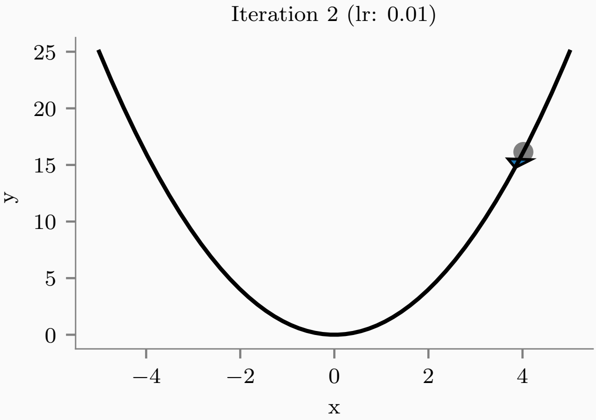
What if α is very small?

Then the rate of convergence is small. It takes more time for a model to reach the minimum cost!

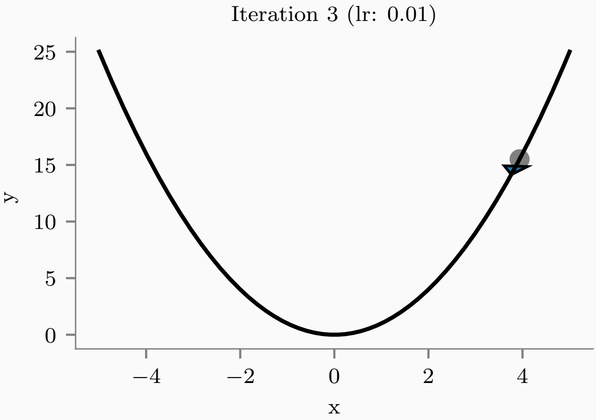
Slow Convergence



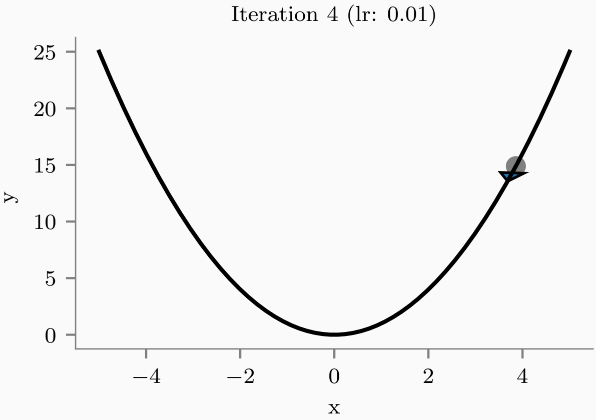
Slow Convergence



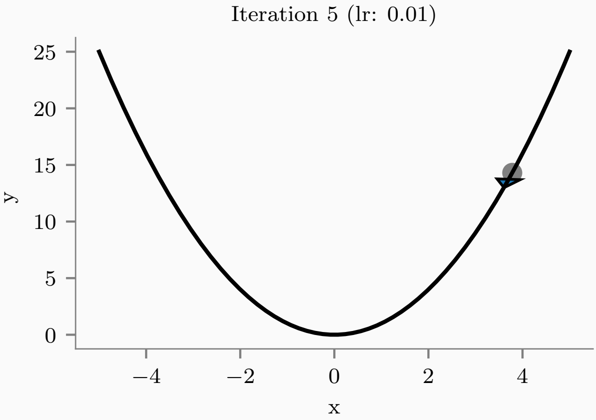
Slow Convergence



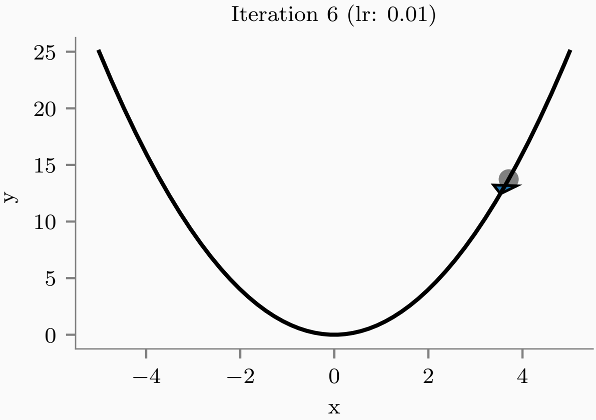
Slow Convergence



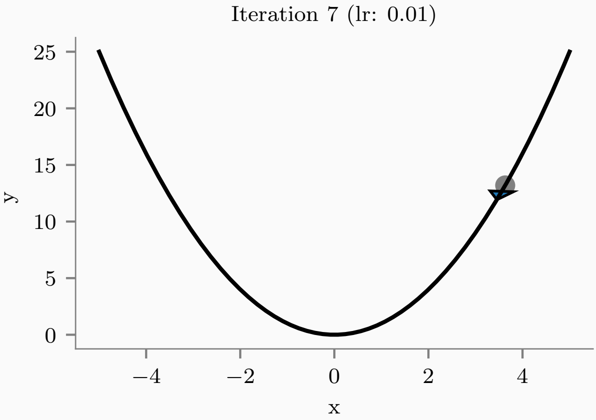
Slow Convergence



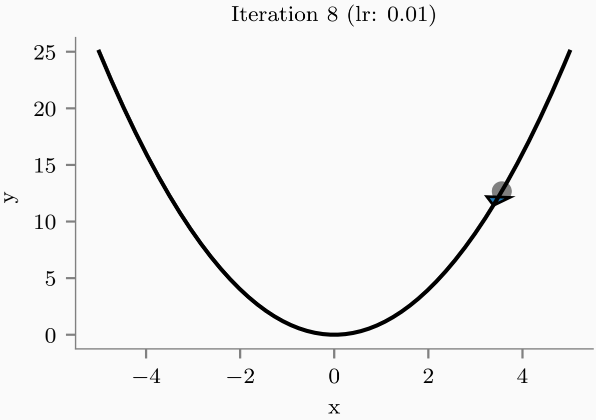
Slow Convergence



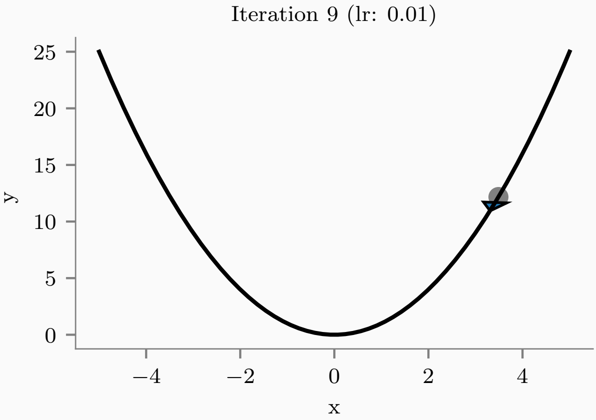
Slow Convergence



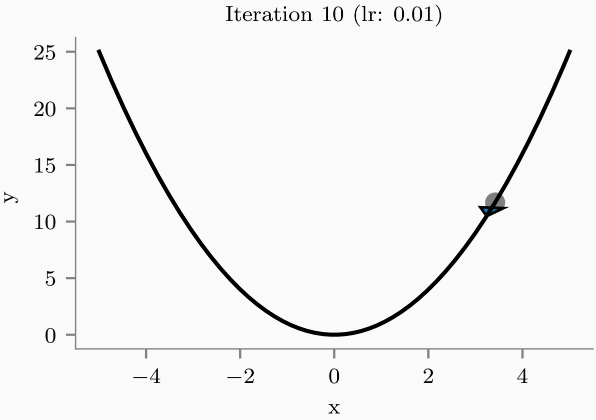
Slow Convergence



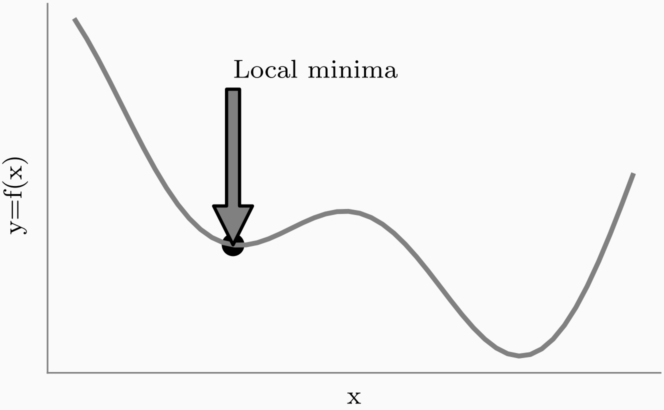
Slow Convergence



Slow Convergence



Local Minima



Gradient Descent for Linear Regression

- **Loss function** is usually a function defined on a data point, prediction and label, and measures the penalty.

Gradient Descent for Linear Regression

- **Loss function** is usually a function defined on a data point, prediction and label, and measures the penalty.
- square loss $l\left(f\left(x_i|\theta\right), y_i\right) = \left(f\left(x_i|\theta\right) - y_i\right)^2$, used in linear regression

Gradient Descent for Linear Regression

- **Loss function** is usually a function defined on a data point, prediction and label, and measures the penalty.
- square loss $l\left(f\left(x_i|\theta\right), y_i\right) = \left(f\left(x_i|\theta\right) - y_i\right)^2$, used in linear regression
- **Cost function** is usually more general. It might be a sum of loss functions over your training set plus some model complexity penalty (regularization). For example:

Gradient Descent for Linear Regression

- **Loss function** is usually a function defined on a data point, prediction and label, and measures the penalty.
- square loss $l\left(f\left(x_i|\theta\right), y_i\right) = \left(f\left(x_i|\theta\right) - y_i\right)^2$, used in linear regression
- **Cost function** is usually more general. It might be a sum of loss functions over your training set plus some model complexity penalty (regularization). For example:
- Mean Squared Error $MSE(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(f\left(x_i|\theta\right) - y_i\right)^2$

Gradient Descent for Linear Regression

- **Loss function** is usually a function defined on a data point, prediction and label, and measures the penalty.
- square loss $l\left(f\left(x_i|\theta\right), y_i\right) = \left(f\left(x_i|\theta\right) - y_i\right)^2$, used in linear regression
- **Cost function** is usually more general. It might be a sum of loss functions over your training set plus some model complexity penalty (regularization). For example:
- Mean Squared Error $MSE(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(f\left(x_i|\theta\right) - y_i\right)^2$
- **Objective function** is the most general term for any function that you optimize during training.

Gradient Descent for Linear Regression

- **Loss function** is usually a function defined on a data point, prediction and label, and measures the penalty.
- square loss $l\left(f\left(x_i|\theta\right), y_i\right) = \left(f\left(x_i|\theta\right) - y_i\right)^2$, used in linear regression
- **Cost function** is usually more general. It might be a sum of loss functions over your training set plus some model complexity penalty (regularization). For example:
- Mean Squared Error $MSE(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(f\left(x_i|\theta\right) - y_i\right)^2$
- **Objective function** is the most general term for any function that you optimize during training.

Gradient Descent for Linear Regression

- **Loss function** is usually a function defined on a data point, prediction and label, and measures the penalty.
- square loss $l\left(f\left(x_i|\theta\right), y_i\right) = \left(f\left(x_i|\theta\right) - y_i\right)^2$, used in linear regression
- **Cost function** is usually more general. It might be a sum of loss functions over your training set plus some model complexity penalty (regularization). For example:
- Mean Squared Error $MSE(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(f\left(x_i|\theta\right) - y_i\right)^2$
- **Objective function** is the most general term for any function that you optimize during training.

We have thus far seen:

Gradient Descent for Linear Regression

- **Loss function** is usually a function defined on a data point, prediction and label, and measures the penalty.
- square loss $l\left(f\left(x_i|\theta\right), y_i\right) = \left(f\left(x_i|\theta\right) - y_i\right)^2$, used in linear regression
- **Cost function** is usually more general. It might be a sum of loss functions over your training set plus some model complexity penalty (regularization). For example:
- Mean Squared Error $MSE(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(f\left(x_i|\theta\right) - y_i\right)^2$
- **Objective function** is the most general term for any function that you optimize during training.

We have thus far seen:

$$\sum \epsilon_i^2 = \sum (y_i - (\theta_0 + \theta_1 x_i))^2$$

Gradient Descent Algorithm

Start with random values of θ_0 and θ_1
Till convergence

- $\theta_0 = \theta_0 - \frac{\partial}{\partial \theta_0}(\sum \epsilon_i^2)$
- $\theta_1 = \theta_1 - \frac{\partial}{\partial \theta_1}(\sum \epsilon_i^2)$

The updates have to be done simultaneously!

Gradient Descent Algorithm

- $\frac{\partial}{\partial\theta_0}(\sum \epsilon_i^2) = 2\sum(y_i - (\theta_0 + \theta_1 x_i))(-1)$
  $\frac{\partial}{\partial\theta_1}(\sum \epsilon_i^2) = 2\sum(y_i - (\theta_0 + \theta_1 x_i))(-x_i)$

Gradient Descent : Example

Learn $y = \theta_0 + \theta_1 x$ on following dataset, using gradient descent where initially $(\theta_0, \theta_1) = (4, 0)$ and step-size, $\alpha = 0.1$, for 2 iterations.

x	**y**
1	1
2	2
3	3

Gradient Descent : Example

Our predictor, $\hat{y} = \theta_0 + \theta_1 x$

Error for i^{th} datapoint, $\epsilon_i = y_i - \hat{y}_i$

$\epsilon_1 = 1 - \theta_0 - \theta_1$

$\epsilon_2 = 2 - \theta_0 - 2\theta_1$

$\epsilon_3 = 3 - \theta_0 - 3\theta_1$

$$\text{MSE} = \frac{\epsilon_1^2 + \epsilon_2^2 + \epsilon_3^2}{3} = \frac{14 + 3\theta_0^2 + 14\theta_1^2 - 12\theta_0 - 28\theta_1 + 12\theta_0\theta_1}{3}$$

Difference between SSE and MSE

$$\sum \epsilon_i^2 \textit{ increases as the number of examples increase}$$

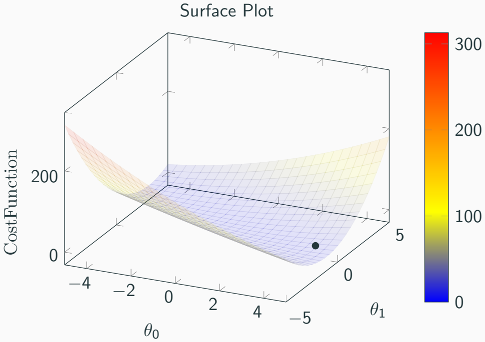
So, we use MSE

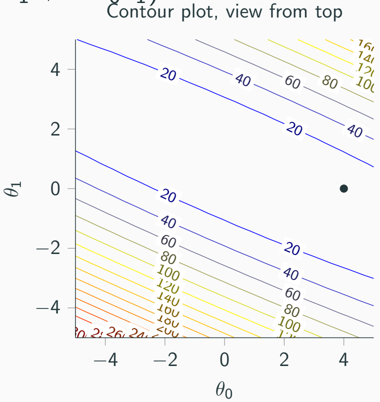
$$MSE = \frac{1}{n} \sum \epsilon_i^2$$

Here n denotes the number of samples

Iteration 0

$$\text{MSE} = \frac{1}{3}(14 + 3\theta_0^2 + 14\theta_1^2 - 12\theta_0 - 28\theta_1 + 12\theta_0\theta_1)$$





Gradient Descent : Example

$$\frac{\partial MSE}{\partial \theta_0} = \frac{2\sum_i (y_i - \theta_0 - \theta_1 x_i)(-1)}{N} = \frac{2\sum_i \epsilon_i (-1)}{N}$$

$$\frac{\partial MSE}{\partial \theta_1} = \frac{2\sum_i (y_i - \theta_0 - \theta_1 x_i)(-x_i)}{N} = \frac{2\sum_i \epsilon_i (-x_i)}{N}$$

Gradient Descent : Example

Iteration 1

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Gradient Descent : Example

Iteration 1

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 4 - 0.2 \frac{((1-(4+0))(-1)+(2-(4+0))(-1)+(3-(4+0))(-1))}{3}$$

$$\theta_0 = 3.6$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Gradient Descent : Example

Iteration 1

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 4 - 0.2 \frac{((1-(4+0))(-1)+(2-(4+0))(-1)+(3-(4+0))(-1))}{3}$$

$$\theta_0 = 3.6$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

$$\theta_1 = 0 - 0.2 \frac{((1-(4+0))(-1)+(2-(4+0))(-2)+(3-(4+0))(-3))}{3}$$

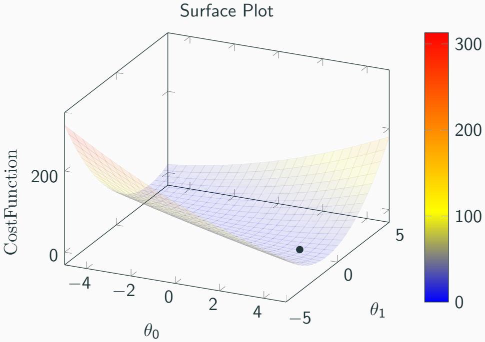
$$\theta_1 = -0.67$$

Iteration 1

$$\text{MSE} = \frac{1}{3}(14 + 3\theta_0^2 + 14\theta_1^2 - 12\theta_0 - 28\theta_1 + 12\theta_0\theta_1)$$

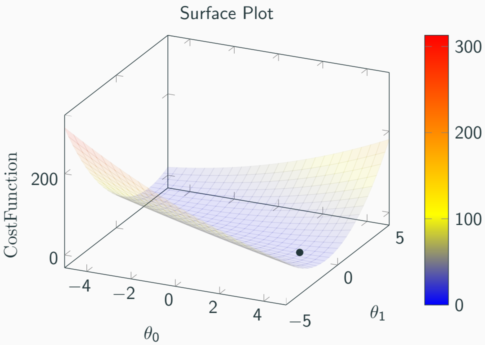
Iteration 1

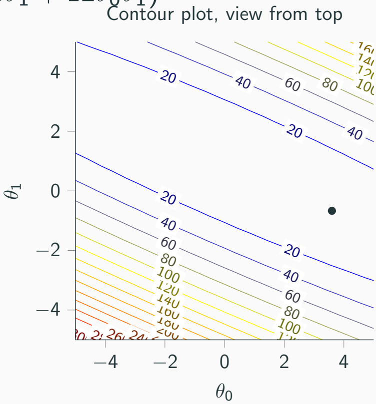
$$\text{MSE} = \frac{1}{3}(14 + 3\theta_0^2 + 14\theta_1^2 - 12\theta_0 - 28\theta_1 + 12\theta_0\theta_1)$$



Iteration 1

$$\text{MSE} = \frac{1}{3}(14 + 3\theta_0^2 + 14\theta_1^2 - 12\theta_0 - 28\theta_1 + 12\theta_0\theta_1)$$





Gradient Descent : Example

Iteration 2

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Gradient Descent : Example

Iteration 2

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 =$$
$$3.6 - 0.2 \frac{((1-(3.6-0.67))(-1)+(2-(3.6-0.67\times2))(-1)+(3-(3.6-0.67\times3))(-1))}{3}$$

$$\theta_0 = 3.54$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Gradient Descent : Example

Iteration 2

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 3.6 - 0.2 \frac{((1-(3.6-0.67))(-1)+(2-(3.6-0.67\times 2))(-1)+(3-(3.6-0.67\times 3))(-1))}{3}$$

$$\theta_0 = 3.54$$

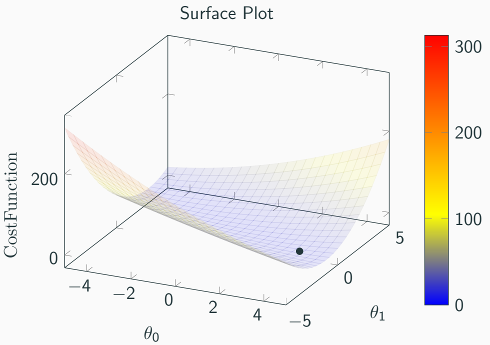
$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

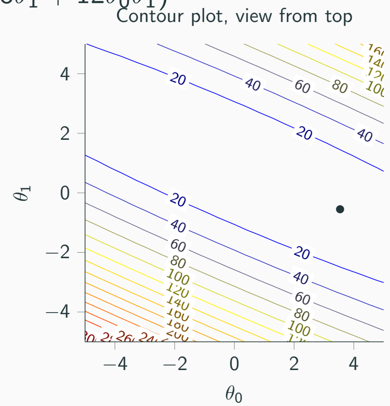
$$\theta_0 = 3.6 - 0.2 \frac{((1-(3.6-0.67))(-1)+(2-(3.6-0.67\times 2))(-2)+(3-(3.6-0.67\times 3))(-3))}{3}$$

$$\theta_0 = -0.55$$

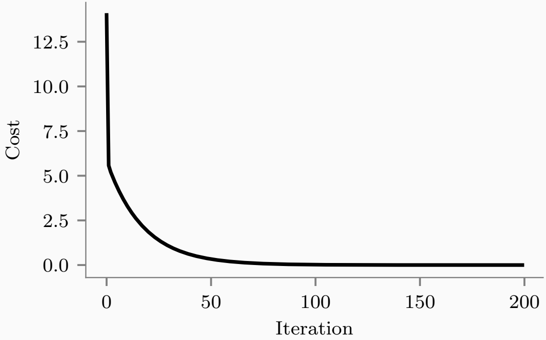
Iteration 2

$$\text{MSE} = \frac{1}{3}(14 + 3\theta_0^2 + 14\theta_1^2 - 12\theta_0 - 28\theta_1 + 12\theta_0\theta_1)$$

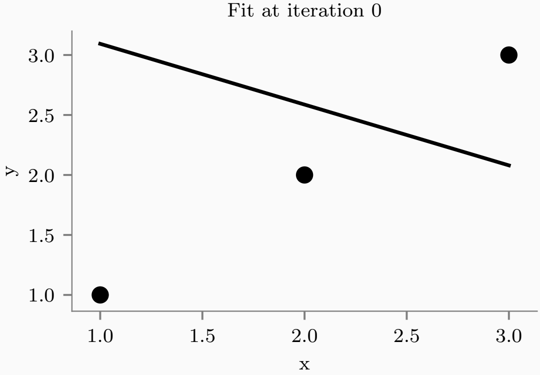




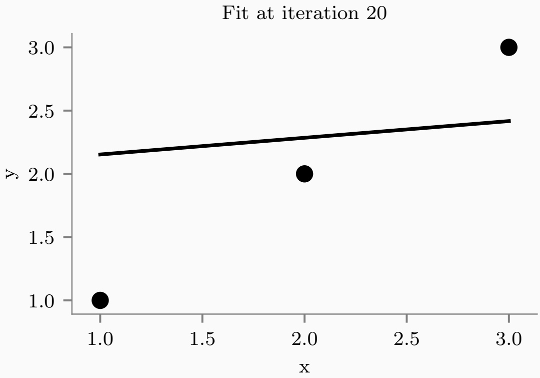
Cost v/s Iterations ($\alpha = 0.1$)



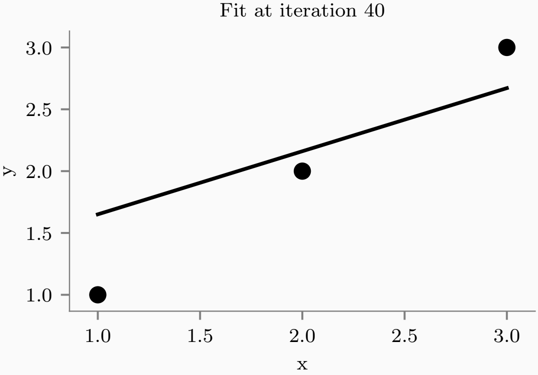
Fit at iteration 0



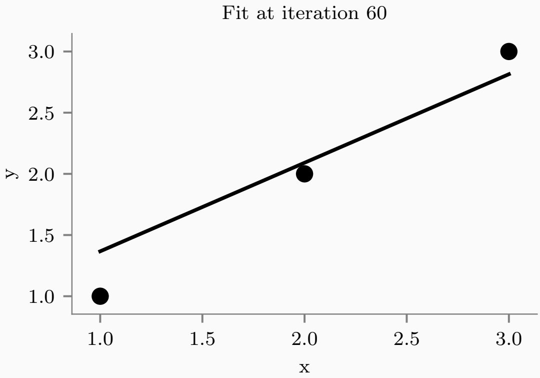
Fit at iteration 20



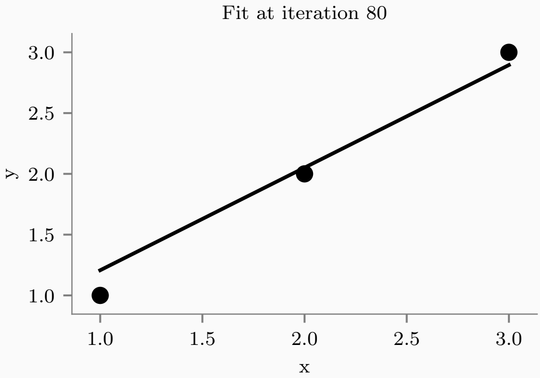
Fit at iteration 40



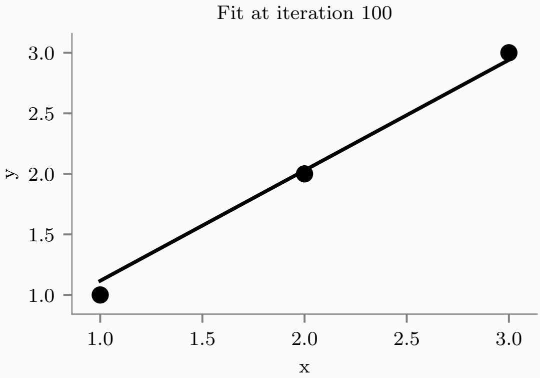
Fit at iteration 60



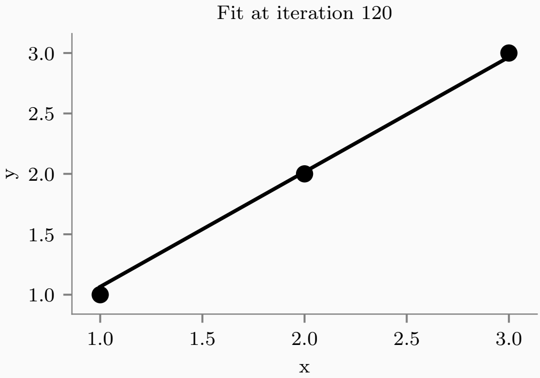
Fit at iteration 80



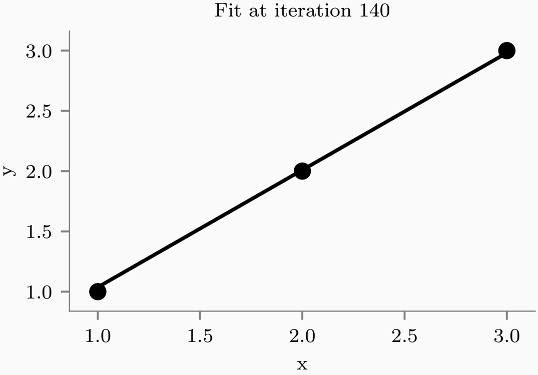
Fit at iteration 100



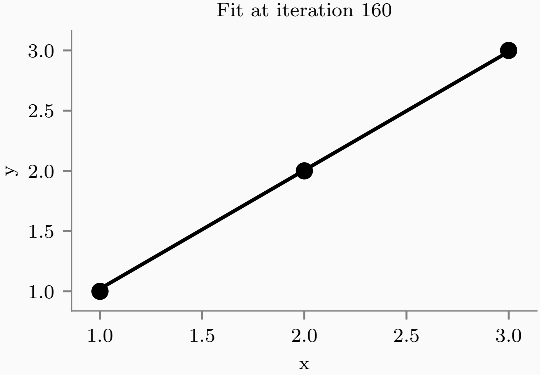
Fit at iteration 120



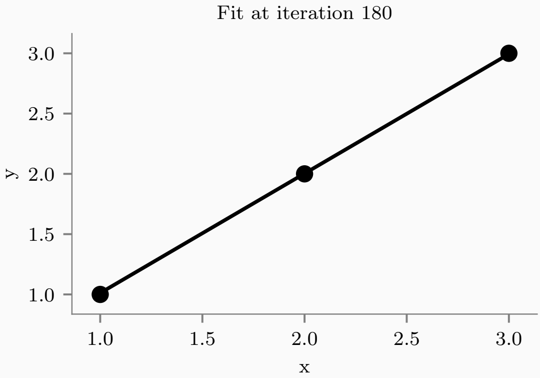
Fit at iteration 140



Fit at iteration 160



Fit at iteration 180



Iteration v/s Epcohs for gradient descent

- Iteration: Each time you update the parameters of the model

Iteration v/s Epcohs for gradient descent

- Iteration: Each time you update the parameters of the model
- Epoch: Each time you have seen all the set of examples

Gradient Descent (GD)

- Dataset: $D = \{(X, y)\}$ of size N
- Initialize θ
- For epoch e in $[1, E]$
  - Predict $\hat{y} = pred(X, \theta)$
  - Compute loss: $J(\theta) = loss(y, \hat{y})$
  - Compute gradient: $\nabla J(\theta) = grad(J)(\theta)$
  - Update: $\theta = \theta - \alpha \nabla J(\theta)$

Stochastic Gradient Descent (SGD)

- Dataset: $D = \{(X, y)\}$ of size N
- Initialize θ
- For epoch e in $[1, E]$
  - Shuffle D
  - For i in $[1, N]$
    - Predict $\hat{y}_i = \mathit{pred}(X_i, \theta)$
    - Compute loss: $J(\theta) = \mathit{loss}(y_i, \hat{y}_i)$
    - Compute gradient: $\nabla J(\theta) = \mathit{grad}(J)(\theta)$
    - Update: $\theta = \theta - \alpha \nabla J(\theta)$

Mini-Batch Gradient Descent (MBGD)

- Dataset: $D = \{(X, y)\}$ of size N
- Initialize θ
- For epoch e in $[1, E]$
  - Shuffle D
  - $Batches = make\_batches(D, B)$
  - For b in $Batches$
    - $X\_b, y\_b = b$
    - Predict $\hat{y\_b} = pred(X\_b, \theta)$
    - Compute loss: $J(\theta) = loss(y\_b, \hat{y\_b})$
    - Compute gradient: $\nabla J(\theta) = grad(J)(\theta)$
    - Update: $\theta = \theta - \alpha \nabla J(\theta)$

Gradient Descent vs SGD

Vanilla Gradient Descent

- in Vanilla (Batch) gradient descent: We update params after going through all the data

Gradient Descent vs SGD

Vanilla Gradient Descent

- in Vanilla (Batch) gradient descent: We update params after going through all the data
- Smooth curve for Iteration vs Cost

Gradient Descent vs SGD

Vanilla Gradient Descent

- in Vanilla (Batch) gradient descent: We update params after going through all the data
- Smooth curve for Iteration vs Cost
- For a single update, it needs to compute the gradient over all the samples. Hence takes more time

Gradient Descent vs SGD

Vanilla Gradient Descent

- in Vanilla (Batch) gradient descent: We update params after going through all the data
- Smooth curve for Iteration vs Cost
- For a single update, it needs to compute the gradient over all the samples. Hence takes more time

Gradient Descent vs SGD

Vanilla Gradient Descent

- in Vanilla (Batch) gradient descent: We update params after going through all the data
- Smooth curve for Iteration vs Cost
- For a single update, it needs to compute the gradient over all the samples. Hence takes more time

Stochastic Gradient Descent

- In SGD, we update parameters after seeing each each point

Gradient Descent vs SGD

Vanilla Gradient Descent

- in Vanilla (Batch) gradient descent: We update params after going through all the data
- Smooth curve for Iteration vs Cost
- For a single update, it needs to compute the gradient over all the samples. Hence takes more time

Stochastic Gradient Descent

- In SGD, we update parameters after seeing each each point
- Noisier curve for iteration vs cost

Gradient Descent vs SGD

Vanilla Gradient Descent

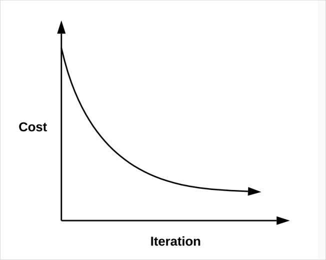
- in Vanilla (Batch) gradient descent: We update params after going through all the data
- Smooth curve for Iteration vs Cost
- For a single update, it needs to compute the gradient over all the samples. Hence takes more time

Stochastic Gradient Descent

- In SGD, we update parameters after seeing each each point
- Noisier curve for iteration vs cost
- For a single update, it computes the gradient over one example. Hence lesser time

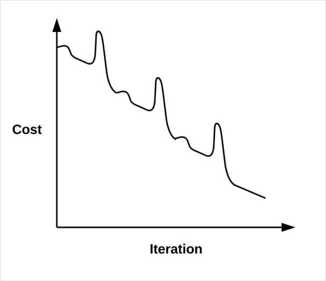
Gradient Descent

- Slower in speed
  (Needs to see many examples before update)
- Smooth convergence
- $iterations = epochs$



Stochastic Gradient Descent

- Faster in speed
- Noisy convergence
- $iterations = epochs \times examples$



Stochastic Gradient Descent : Example

Learn $y = \theta_0 + \theta_1 x$ on following dataset, using SGD where initially $(\theta_0, \theta_1) = (4, 0)$ and step-size, $\alpha = 0.1$, for 1 epoch (3 iterations).

x	**y**
2	2
3	3
1	1

Stochastic Gradient Descent : Example

Our predictor, $\hat{y} = \theta_0 + \theta_1 x$

Error for i^{th} datapoint, $e_i = y_i - \hat{y}_i$

$e_1 = 1 - \theta_0 - \theta_1$

$e_2 = 2 - \theta_0 - 2\theta_1$

$e_3 = 3 - \theta_0 - 3\theta_1$

While using SGD, we compute the MSE using only 1 datapoint per iteration.

So MSE is e_1^2 for iteration 1 and e_2^2 for iteration 2.

Stochastic Gradient Descent : Example

For Iteration i

$$\frac{\partial MSE}{\partial \theta_0} = 2\left(y_i - \theta_0 - \theta_1 x_i\right)(-1) = 2e_i\,(-1)$$

$$\frac{\partial MSE}{\partial \theta_1} = 2\left(y_i - \theta_0 - \theta_1 x_i\right)(-x_i) = 2e_i\,(-x_i)$$

Stochastic Gradient Descent : Example

Iteration 1

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Iteration 1

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 4 - 0.1 \times 2 \times (2 - (4 + 0))\,(-1)$$

$$\theta_0 = 3.6$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Stochastic Gradient Descent : Example

Iteration 1

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 4 - 0.1 \times 2 \times (2 - (4 + 0))\,(-1)$$

$$\theta_0 = 3.6$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

$$\theta_1 = 0 - 0.1 \times 2 \times (2 - (4 + 0))\,(-2)$$

$$\theta_1 = -0.8$$

Stochastic Gradient Descent : Example

Iteration 2

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Stochastic Gradient Descent : Example

Iteration 2

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 3.6 - 0.1 \times 2 \times (3 - (3.6 - 0.8 \times 3))\,(-1)$$

$$\theta_0 = 3.96$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Stochastic Gradient Descent : Example

Iteration 2

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 3.6 - 0.1 \times 2 \times (3 - (3.6 - 0.8 \times 3))\,(-1)$$

$$\theta_0 = 3.96$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

$$\theta_0 = -0.8 - 0.1 \times 2 \times (3 - (3.6 - 0.8 \times 3))\,(-3)$$

$$\theta_1 = 0.28$$

Stochastic Gradient Descent : Example

Iteration 3

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Stochastic Gradient Descent : Example

Iteration 3

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 3.96 - 0.1 \times 2 \times (1 - (3.96 + 0.28 \times 1))(-1)$$

$$\theta_0 = 3.312$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Stochastic Gradient Descent : Example

Iteration 3

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_0 = 3.96 - 0.1 \times 2 \times (1 - (3.96 + 0.28 \times 1))\,(-1)$$

$$\theta_0 = 3.312$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

$$\theta_0 = 0.28 - 0.1 \times 2 \times (1 - (3.96 + 0.28 \times 1))\,(-1)$$

$$\theta_1 = -0.368$$

Mini-Batch Gradient Descent

In mini-batch gradient descent, we compute the gradient over a mini-batch of samples, thereby getting the best of both worlds.

When to use Gradient Descent

Gradient Descent

- Good for online setting (more data over time, no need to create new matrices!)
- Good for large data

Normal systems

- Good for simple data
- No need to worry about learning rates, etc
- Non trivial to solve

Projected Gradient Descent

For θ_i, if we want to impose the condition that $\theta_i >= 0$

$$\theta_i = max(\theta_i - \alpha \frac{\partial \epsilon(\theta_0, \theta_1, ...)}{\partial \theta_i}, 0)$$