CSC 461: Machine Learning

Fall 2024

Gradient descent and machine learning

Prof. Marco Alvarez, Computer Science
University of Rhode Island

Machine learning as optimization

- Goal
  - find parameters that minimize loss/error
- Challenge
  - complex loss landscapes
  - large-scale datasets
- Solution
  - gradient-based optimization methods

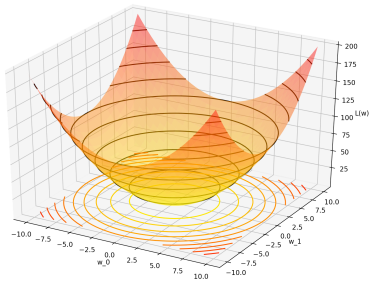
From linear regression …

$$L(\mathbf{w}) = \frac{1}{n}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

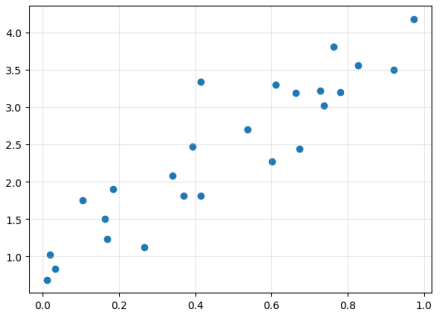
continuous, differentiable, **convex**

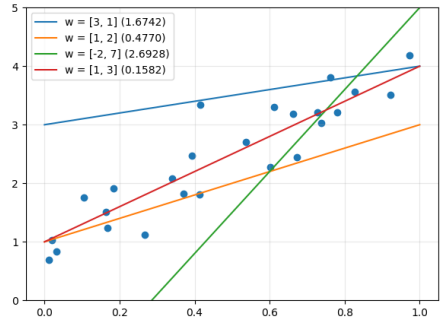
$$\mathbf{w}^* = \arg\min_{\mathbf{w}} L(\mathbf{w}, \mathscr{D})$$

optimal solution

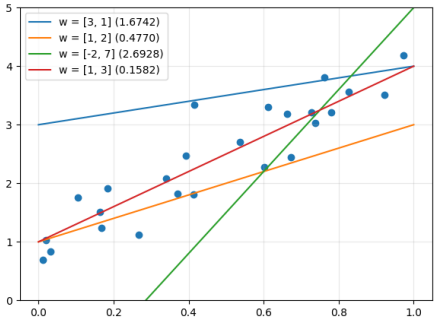


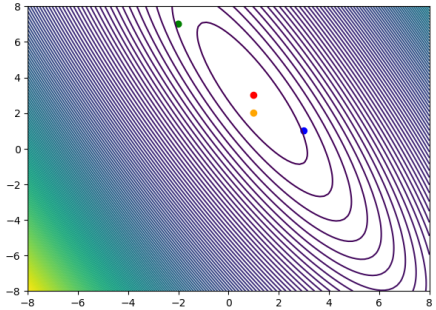
Data space





Data space vs parameter space

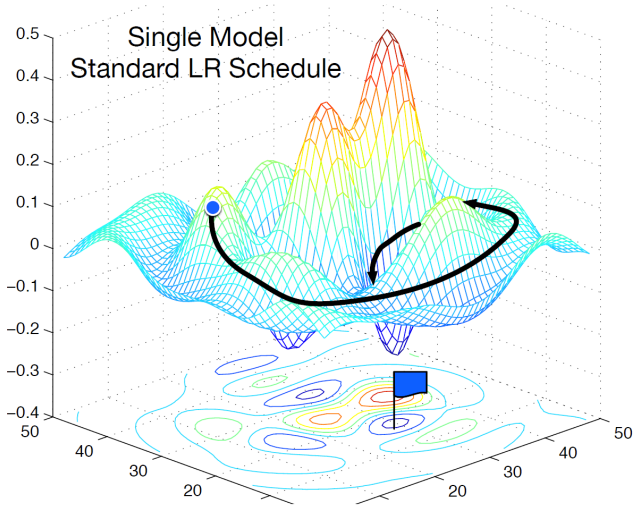




weight space (model parameters)
optimization happens in weight space

Gradient descent in ML

Gradient descent in ML



Gradient descent in ML

- Want to **minimize** a loss function $L(\mathbf{w})$?
  - use gradient descent:
    - start with a guess $\mathbf{w}^{(0)}$
    - use $\mathbf{w}^{(0)}$ to generate $\mathbf{w}^{(1)}$
    - use $\mathbf{w}^{(1)}$ to generate $\mathbf{w}^{(2)}$
    - …

opposite direction of the gradient

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}} L\left(\mathbf{w}^{(t)}\right)$$

Batch gradient descent

Batch gradient descent

$$w^{(t)} \rightarrow w^{(t+1)} \rightarrow w^{(t+2)} \rightarrow \dots$$

`compute gradient`

`requires a full pass over the training data`

1 **EPOCH** refers to 1 pass over the data

Initialize vector **w**

Repeat until convergence (or termination criteria)

$$w_j^{(t+1)} = w_j^{(t)} - \eta \frac{\partial L\left(\mathbf{w}^{(t)}\right)}{\partial w_j^{(t)}}$$

simultaneously update all w_j for $j = 0, \dots, d$

Gradient for linear regression

with respect to a single w_j

$$\frac{\partial L(\mathbf{w})}{\partial w_j} = \frac{\partial}{\partial w_j}\left(\frac{1}{n}\sum_{i=0}^{n}\left(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)}\right)^2\right)$$

$$= \frac{2}{n}\sum_{i=0}^{n}\left(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)}\right)\frac{\partial}{\partial w_j}\left(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)}\right)$$

$$= \frac{2}{n}\sum_{i=0}^{n}\left(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)}\right)\ x_j^{(i)}$$ partial derivative

Gradient for linear regression

with respect to **w**

$$\frac{\partial L(\mathbf{w})}{\partial w_j} = \frac{\partial}{\partial w_j}\left(\frac{1}{n}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2\right)$$

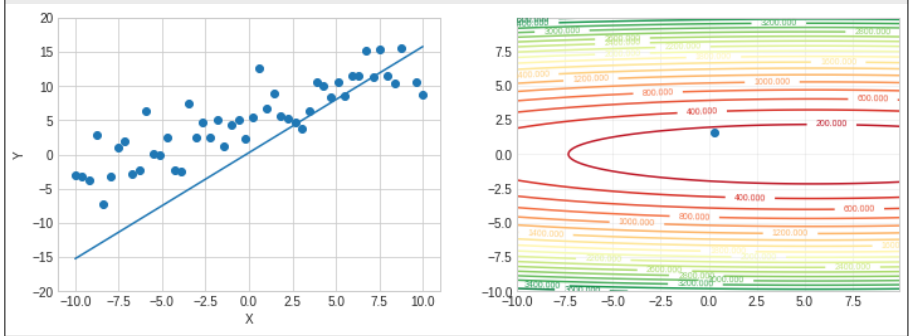
$$= \frac{2}{n}X^T(\mathbf{X}\mathbf{w} - y)$$ gradient vector

Show me the code

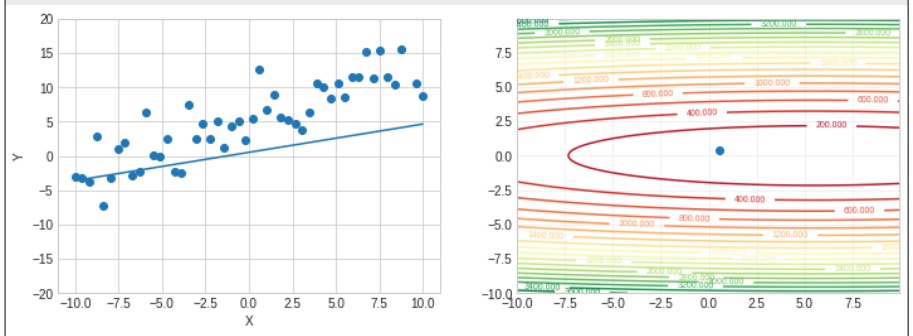
```
# initialize variables and weights
l_rate = 0.025
n_epochs = 10
w = np.zeros([2,1])

# apply gradient descent
for i in range(n_epochs):
  grad = 2 * (augX.T @ (augX @ w - Y) / n_points)
  w = w - l_rate * grad
```

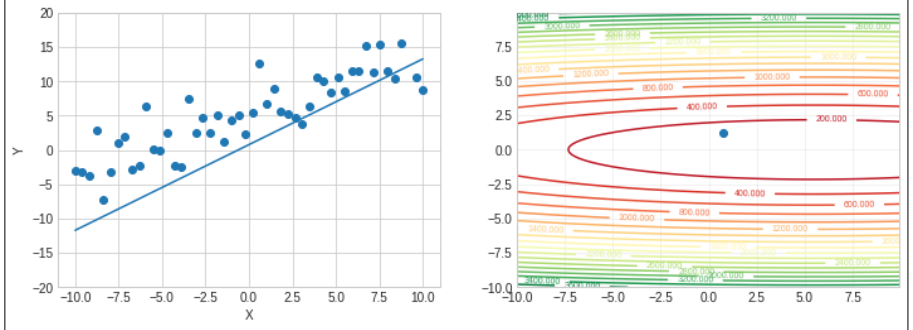
After 1 iteration



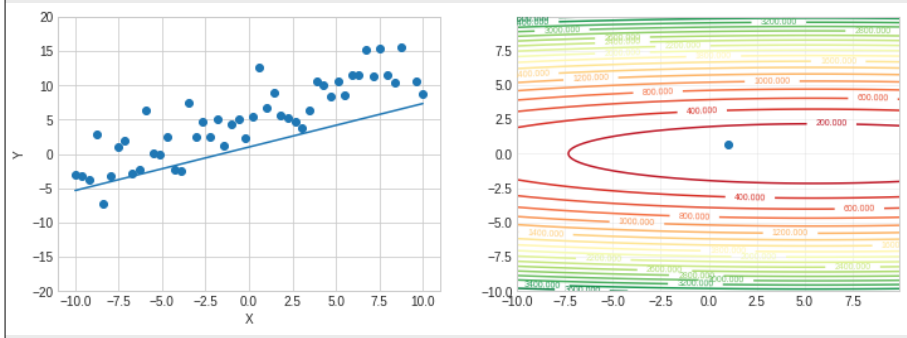
After 2 iterations



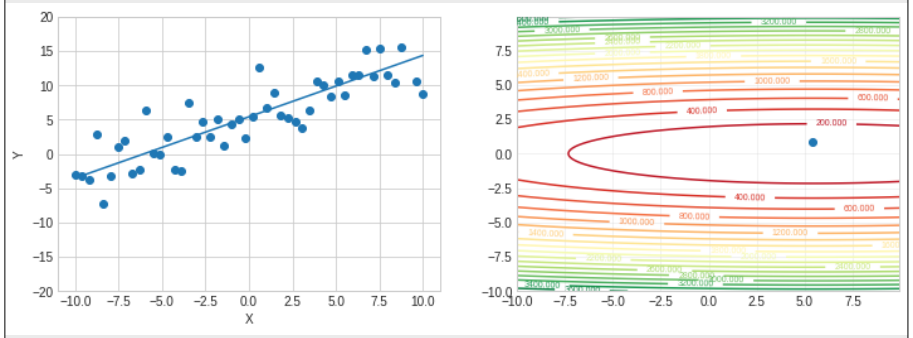
After 3 iterations



After 4 iterations



After 100 iterations



Performance

- Each iteration of batch gradient descent uses the entire training set
  - can be **slow** for big datasets

$$w_j^{(t+1)} = w_j^{(t)} - \eta \frac{2}{n} \sum_{i=1}^{n} \left(\mathbf{w}^{(t)T} \mathbf{x}^{(i)} - y^{(i)} \right) x_j^{(i)}$$

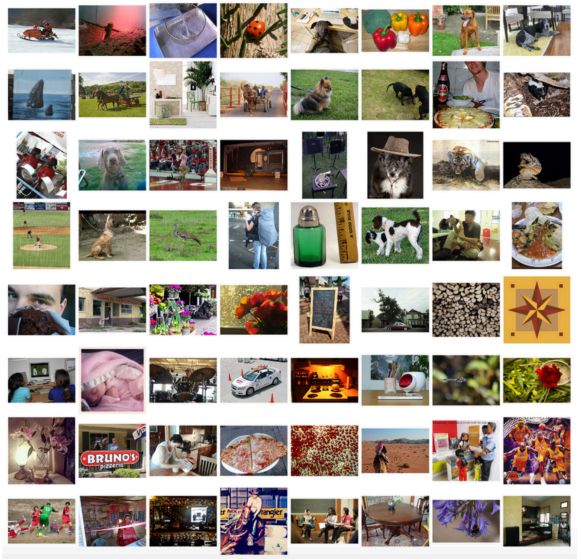
update for a single weight

sum over all instances in the training set

IM..GENET

1000 categories
1,281,167 training images
50,000 validation images
100,000 test images

The most highly-used subset of ImageNet



Stochastic and mini-batch gradient descent

Stochastic gradient descent

- Why not calculating an **approximate gradient**?
  - use a **single datapoint** at each update (stochastic sampling)
- Scalable method
  - huge impact on real-world applications
- Behavior
  - at some iterations loss may increase, but the algorithm can still make progress

Stochastic gradient descent

$$w^{(t)} \rightarrow w^{(t+1)} \rightarrow w^{(t+2)} \rightarrow \dots$$

compute gradient

use a single datapoint

Initialize vector **w**

Repeat until convergence (or termination criteria)

randomly shuffle the dataset

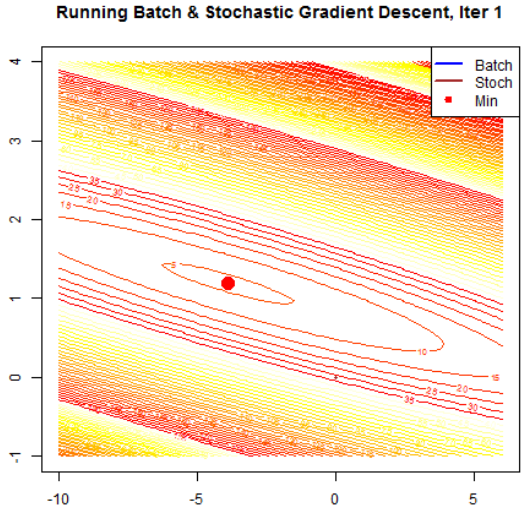
for each datapoint **x**

$$w_j^{(t+1)} = w_j^{(t)} - \eta \frac{\partial l\left(\mathbf{w}^{(t)}\right)}{\partial w_j^{(t)}}$$

individual loss on a single datapoint **x**

simultaneously update all w_j for $j = 0,\dots,d$

Batch vs Stochastic



https://sandipanweb.wordpress.com/2016/07/09/convergence-of-the-gradient-descent-algorithms-stochastic-and-batch-for-the-linear-logistic-regression-and-perceptron/

Mini-batch gradient descent

$$w^{(t)} \rightarrow w^{(t+1)} \rightarrow w^{(t+2)} \rightarrow \dots$$

compute gradient

use a small subset of the data

Initialize vector **w**

Repeat until convergence (or termination criteria)

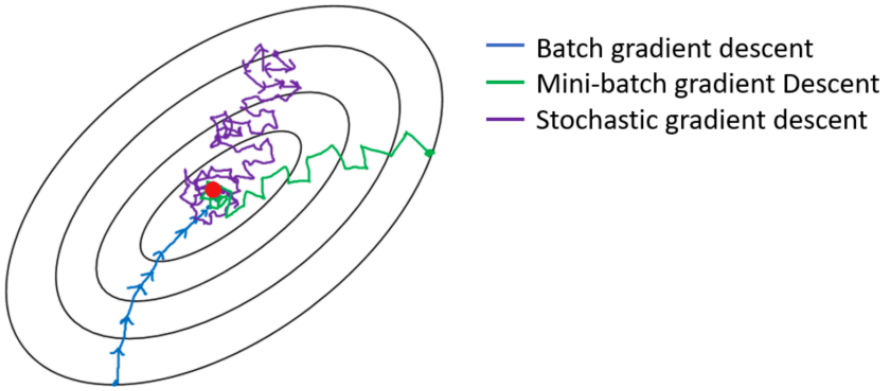
randomly split the dataset into batches of size m

for each batch

$$w_j^{(t+1)} = w_j^{(t)} - \eta \frac{\partial L\left(\mathbf{w}^{(t)}\right)}{\partial w_j^{(t)}}$$

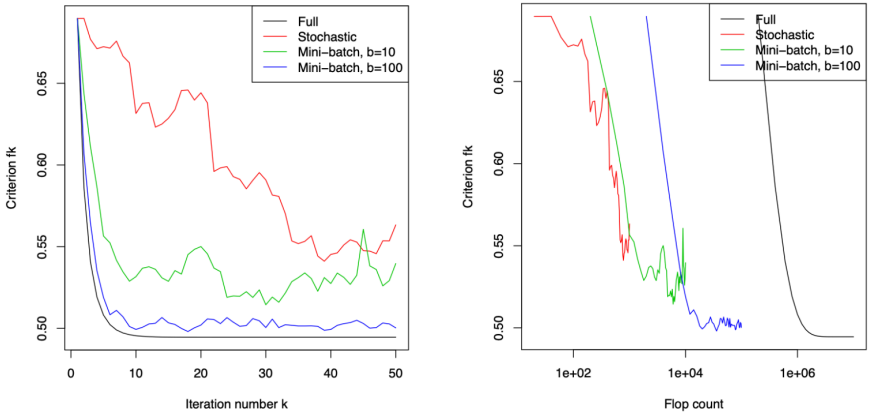
average loss over a mini-batch

simultaneously update all w_j for $j = 0, \dots, d$



https://imaddabbura.github.io/post/gradient-descent-algorithm/

Convergence comparison



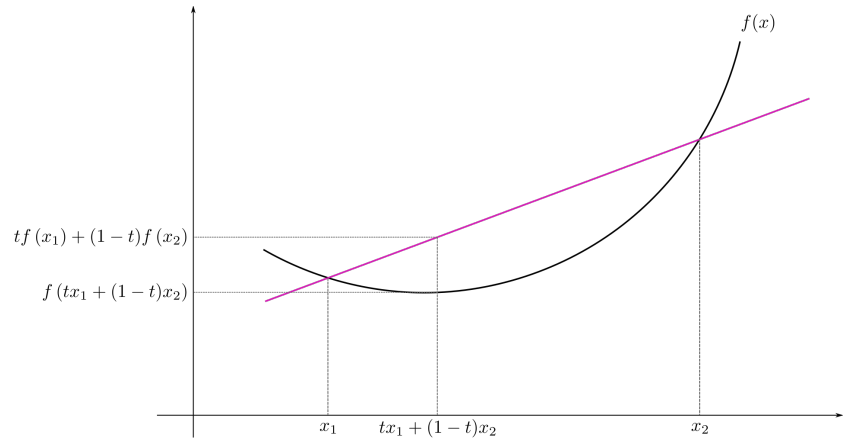
Example from Ryan Tibshirani

Convergence Properties

- Batch GD
  - guaranteed convergence for convex functions
  - slower but more stable
- SGD
  - faster iterations
  - noisier path
  - requires learning rate scheduling
- Mini-batch GD
  - better gradient estimation than SGD
  - more efficient than batch GD
  - popular in deep learning

Convex function

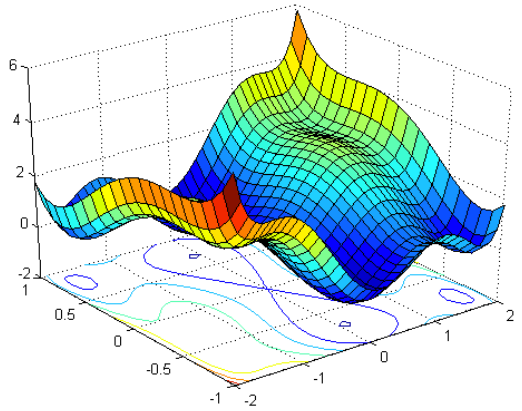
A real-valued function is called convex if the line segment between any two points on the graph of the function lies above the graph between the two points.



From wikipedia (https://en.wikipedia.org/wiki/Convex_function)

Gradient descent and convexity

- If the function is **non-convex**, GD may not find the global minimum



Conclusion

- Gradient Descent
  - foundation of modern ML
- Multiple variants for different scenarios
- Modern techniques handle various challenges
- Critical for training deep neural networks