CSC 561: Neural Networks and Deep Learning

Gradient Descent

Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

Spring 2024

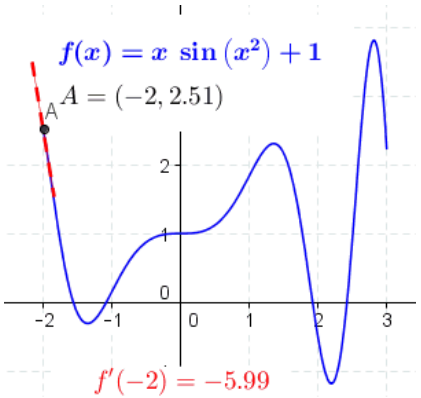
THINK BIG WE DO℠

The gradient

From calculus …

- Given a function $f(x)$, its derivative $f'(x)$ tells us how much a very small increment to the argument will change the value of the function
  - steepness of the function at x
  - rate of change at x
- Positive slope
  - very small increase in x **increases** $f(x)$
- Negative slope
  - very small increase in x **decreases** $f(x)$

Scalar function of scalar argument



For a function $y = f(x)$ the derivative is given by $f'(x) = \alpha$, such that $\Delta y = \alpha \Delta x$

https://en.wikipedia.org/wiki/Derivative

Scalar function of vector argument

- Input is now a column vector $\mathbf{x}$ $\quad y = f(\mathbf{x})$
  - note that $\Delta\mathbf{x}$ is also a vector $\quad \Delta y = \alpha\Delta\mathbf{x}$
- **Derivative** is a row vector $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_d]$
  - partial derivative α_i indicates how y changes when x_i is incremented, also represented as $\frac{\partial y}{\partial x_i}$

$$\Delta y = \frac{\partial y}{\partial x_1}\Delta x_1 + \frac{\partial y}{\partial x_2}\Delta x_2 + \ldots + \frac{\partial y}{\partial x_d}\Delta x_d$$

Examples

- What is the derivative of:

$$f(\mathbf{x}) = x_1^3 + 2x_2 + 5x_3^4?$$

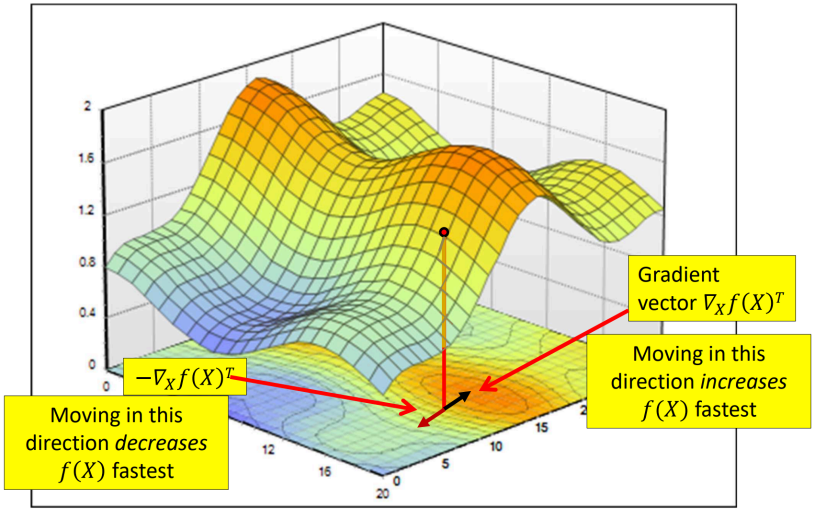
$$f(x, y, z) = x^3 + 2y + 5z^4?$$

$$f(x, y, z) = x^3y^2 + 2xy + 5yz^4?$$

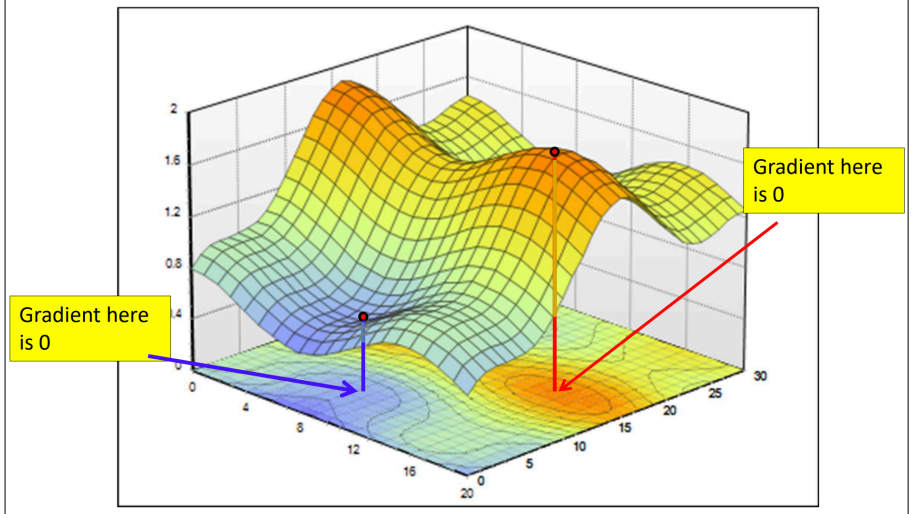
Gradient

- A **gradient** is the transpose of the **derivative**
  - a column vector of partial derivatives

$$\nabla_{\mathbf{x}} f(\mathbf{x})^T = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_d} \end{bmatrix}$$



From 11-785 Introduction to Deep Learning at CMU



The Hessian

- Square matrix containing all second-order partial derivatives

$$\nabla^2_{\mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_d \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_d \partial x_d} \end{bmatrix}$$

Gradient descent

Unconstrained optimization

- Given a multivariate function $f(\mathbf{x})$
  - calculate the derivative $\nabla_{\mathbf{x}} f(\mathbf{x})$ and solve for $\mathbf{x}$ where the derivative is zero
  - compute the Hessian at solution $\mathbf{x}^*$
    - if positive definite (positive eigenvalues) then it is a local minima
    - if negative definite (negative eigenvalues) then it is a local maxima
  - note this approach is not scalable
- Example
  - $f(x, y, z) = x^3 + 3x^2y - yz^3 + z^2$

Alternative solution

- Iterative methods
  - apply an update rule iteratively until finding the solution (or approximating it)

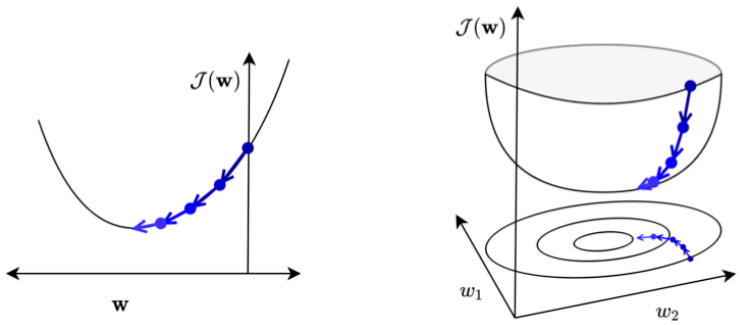
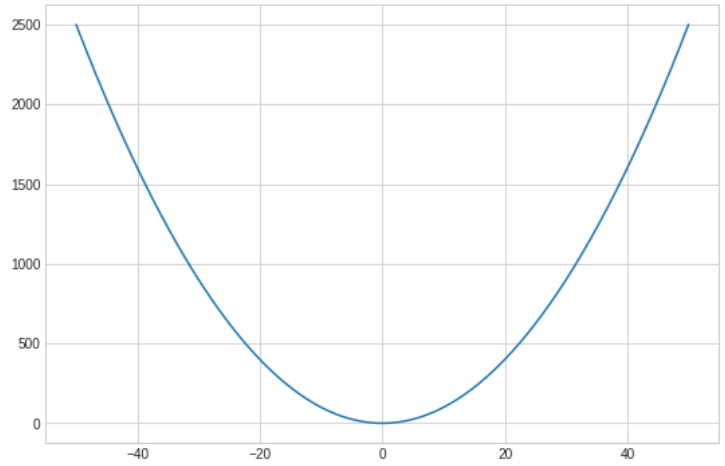


Figure from https://www.cs.toronto.edu/~rgrosse/courses/csc311_f21/lectures/lec03.pdf

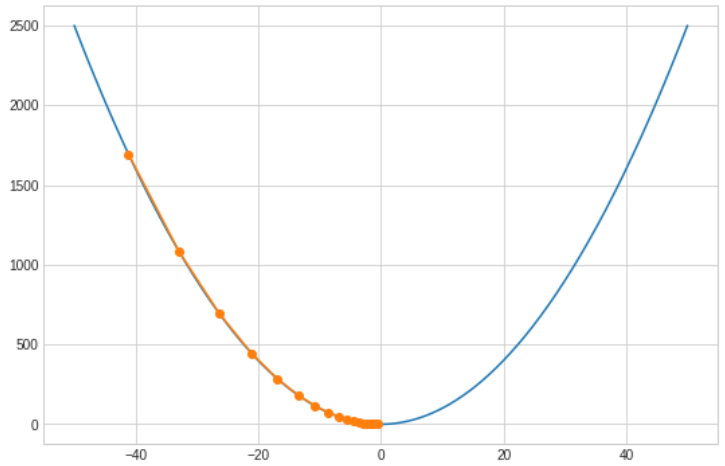
Gradient descent

- Optimization technique used to find the value of x where $f(x)$ is **minimum**
  - guess a starting point
  - walk iteratively (taking steps) in the **opposite direction** of the function's gradient
- Alternatively, to find the **maximum**, walk in the **direction** of the gradient (gradient ascent)
- Step size (a.k.a. **learning rate**) is critical
  - hyperparameter

Example: $f(x) = x^2$



Example: $f(x) = x^2$

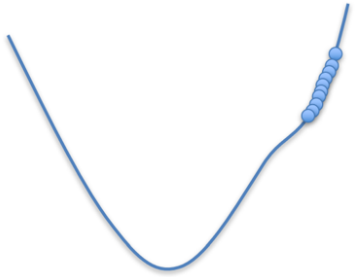


Show me the code

```python
# define a function and its derivative
f = lambda x: x ** 2
df = lambda x: 2 * x

# apply gradient descent
n_steps, l_rate = 10, 2

solution = np.random.randint(-50, 50)

for i in range(n_steps):
    solution = solution - (l_rate * df(solution))
    print(f'{solution:.4f}')
```

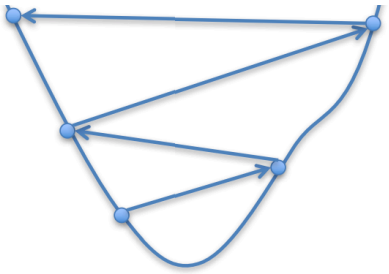

The learning rate

slow convergence



too small

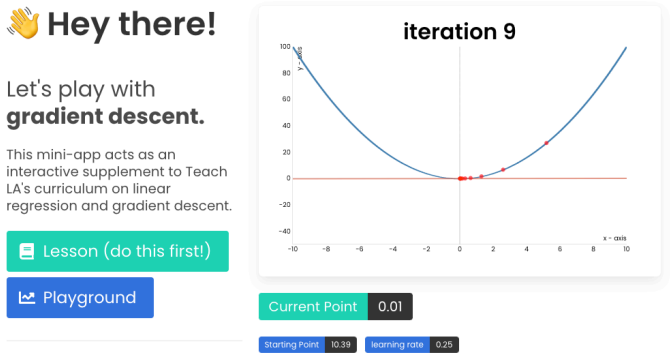
overshoots, may not converge



too large

https://www.cc.gatech.edu/~bboots3/CS4641-Fall2018/Lecture3/03_LinearRegression.pdf

Playground



https://uclaacm.github.io/gradient-descent-visualiser/

Gradient descent and machine learning

Learning (minimizing the loss)

- Want to minimize the **empirical risk**

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} J(\mathbf{w})$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} l\left(h_w, x_i, y_i\right)$$

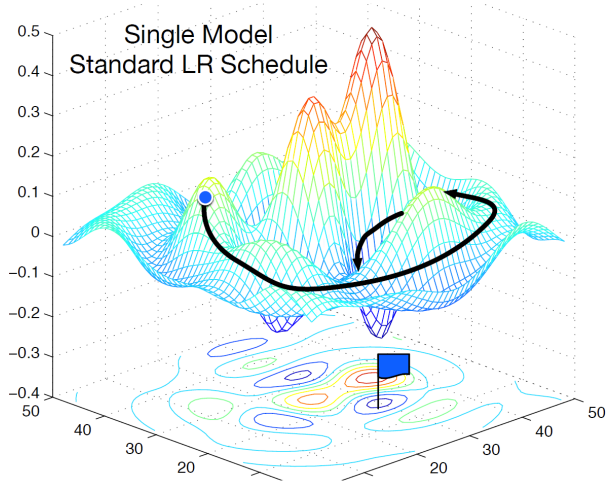
Learning (minimizing the loss)

- Can use gradient descent
  - iteratively update the desired **parameters** (**weights**) until convergence or a maximum number of iterations

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} J(\mathbf{w})$$

opposite direction of the gradient

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



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

Updating the parameters

- **Full** gradient descent
  - a.k.a. batch gradient descent
  - update parameters with the gradient from the entire training data
  - 1 iteration = 1 pass over the training data
- **Mini-batch** gradient descent
  - update parameters with the gradient from subsets of the training data (mini-batches)
  - 1 iteration = 1 pass over a mini-batch of the training data
- **Stochastic** gradient descent
  - update parameters with the gradient from a single datapoint
  - 1 iteration = 1 datapoint

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

full gradient descent

requires a full pass over the training data

can be slow on large datasets

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

mini-batch gradient descent

use a subset (mini-batch) at each iteration

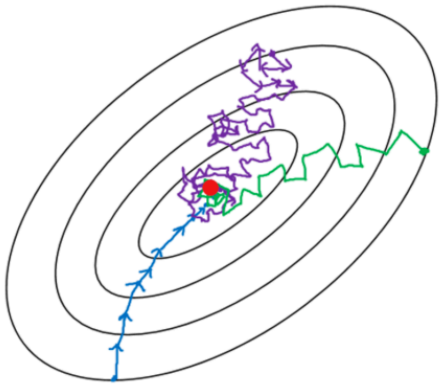
$\mathbf{w}^{(t)} \rightarrow \mathbf{w}^{(t+1)} \rightarrow \mathbf{w}^{(t+2)} \rightarrow \ldots$

stochastic gradient descent

use a single datapoint at each iteration

scalable but with high variance (loss increasing at some iterations)

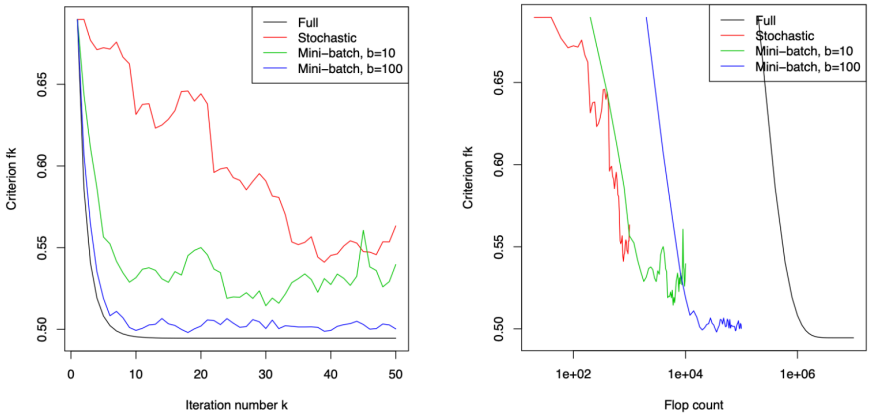
Convergence



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

Efficiency

Consider logistic regression with 10000 training samples and $\mathbf{x_i} \in \mathbb{R}^{20}$



Remarks on mini-batch and stochastic GD

- Both are **unbiased** estimators of the full gradient
- Both can march toward a solution before even seeing all the data
- **Constant** computational cost of each weight update
  - regardless the length of the training data
- SGD with a fixed learning rate cannot guarantee convergence
  - it does not minimize the gradient of the full data directly
  - **must shrink** learning rate during training