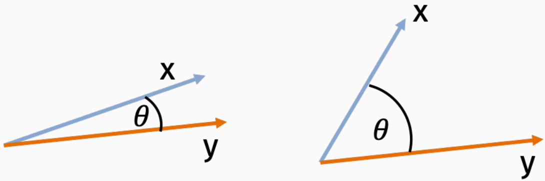
CS-GY 9223 I: Lecture 5
Gradient Decent and Its Many Forms

NYU Tandon School of Engineering, Prof. Christopher Musco

Cosine similarity $\cos(\theta(\mathbf{x}, \mathbf{y})) = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\|_2 \|\mathbf{y}\|_2}$:



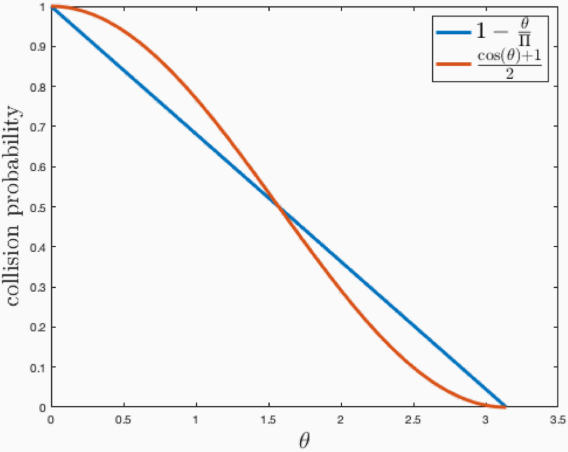
$$-1 \leq \cos(\theta(\mathbf{x}, \mathbf{y})) \leq 1.$$

Locality sensitive hash for cosine similarity:

- Let $\mathbf{g} \in \mathbb{R}^d$ be randomly chosen with each entry $\mathcal{N}(0, 1)$.
- $h : \mathbb{R}^d \to \{-1, 1\}$ is definied $h(\mathbf{x}) = \text{sign}(\langle \mathbf{g}, \mathbf{x} \rangle)$.

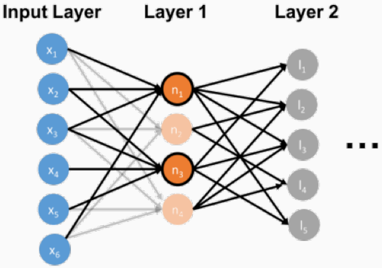
If $\cos(\theta(\mathbf{x}, \mathbf{y})) = v$, what is $\Pr[h(\mathbf{x}) == h(\mathbf{y})]$?

SIMHASH ANALYSIS



$1 - \frac{\theta}{\Pi}$
$\frac{\cos(\theta)+1}{2}$
collision probability
θ
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0.5
1.5
2
2.5
3
3.5

Work of Anshumali Shrivastava at Rice University and coauthors.

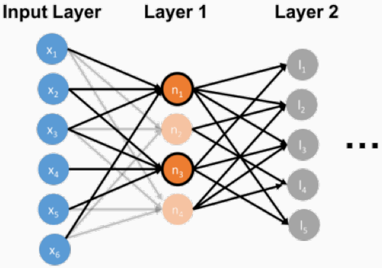


$$n_i = \sigma\left(\sum_{j=1}^{m} w(x_j, n_i)\cdot x_j\right) = \sigma(\langle w_i, x\rangle)$$

- Number of multiplications to evaluate $\mathcal{N}(\mathbf{x})$:
  $|\mathbf{x}| \cdot |\text{layer } 1| + |\text{layer } 1| \cdot |\text{layer } 2| + |\text{layer } 2| \cdot |\text{layer } 3| + \ldots$.
- For an approximate solution, only consider neurons on each each with high activation.

SIMHASH TO SPEEDUP NEURAL NETWORKS

Work of Anshumali Shrivastava at Rice University and coauthors.



$$n_i = \sigma\left(\sum_{j=1}^{m} w(x_j, n_i) \cdot x_j\right) = \sigma(\langle w_i, x\rangle)$$

- High activation = large value of $\sigma(\langle \mathbf{w}_i, \mathbf{x}\rangle)$.
- Typically $\sigma(\langle \mathbf{w}_i, \mathbf{x}\rangle)$ increases as $\langle \mathbf{w}_i, \mathbf{x}\rangle$ increases.
- Use LSH/SimHash to quickly find all $\mathbf{w}_i$ for which $\langle \mathbf{w}_i, \mathbf{x}\rangle$ is large and only include these terms in the sum.

Optimization

Have some function $f : \mathbb{R}^d \rightarrow \mathbb{R}$. Want to find $\hat{\mathbf{x}}$ such that:

$$f(\hat{\mathbf{x}}) = \min_{\mathbf{x}} f(\mathbf{x}).$$

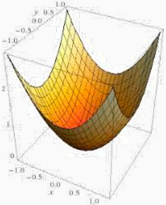
Or at least $\hat{\mathbf{x}}$ is close to a minimum. E.g. $f(\hat{\mathbf{x}}) \leq \min_{\mathbf{x}} f(\mathbf{x}) + \epsilon$

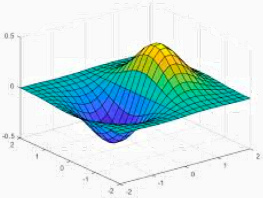
Often we have some additional constraints:

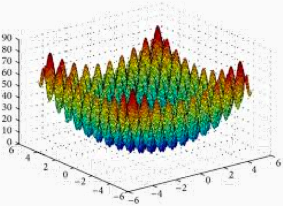
- $\mathbf{x} > 0$
- $\|\mathbf{x}\|_2 \leq R$, $\|\mathbf{x}\|_1 \leq R$
- $\mathbf{a}^T\mathbf{x} > c$.

Dimension $d = 1$:

Dimension $d = 2$:







Machine learning: Want to learn a model that maps input

- numerical data vectors
- images, video
- text documents

to prediction

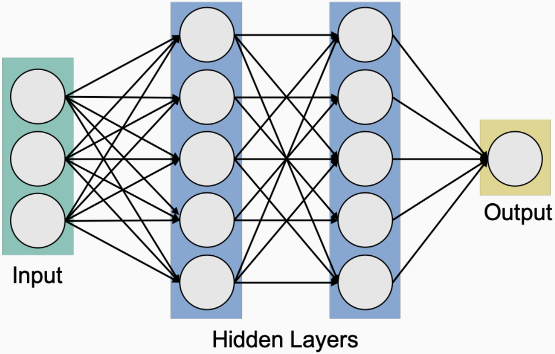
- numerical value (probability stock price increases)
- label (is the image a cat? does the image contain a car?)
- decision (turn car left, rotate robotic arm)

Let $M_{\mathbf{x}}$ be a model with parameters $\mathbf{x} = \{x_1, \ldots, x_d\}$.

Example:

$$M_{\mathbf{x}}(\mathbf{y}) = \mathrm{sign}(\mathbf{y}^T \mathbf{x}_{1:d-1} + \mathbf{x}_d)$$

Example:



Classic approach in <u>supervised learning</u>: Find a model that works well on data that you already have the answer for (labels, values, classes, etc.).

- Model $M_{\mathbf{x}}$ parameterized by a vector of numbers $\mathbf{x}$.
- Dataset $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(n)}$ with outputs $o^{(1)}, \ldots, o^{(n)}$.

Want to find $\hat{\mathbf{x}}$ so that $M_{\hat{\mathbf{x}}}(\mathbf{y}^{(i)}) \approx o^{(i)}$ for $i \in 1, \ldots, n$.

How do we turn this into a function minization problem?

Loss function $L(M_{\mathbf{x}}(\mathbf{y}), o)$: Some measure of distance between prediction $M_{\mathbf{x}}(\mathbf{y})$ and true output o. Increases if they are further apart.

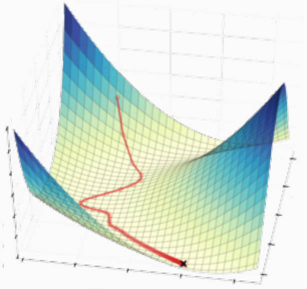
- Squared (ℓ_2) loss: $|M_{\mathbf{x}}(\mathbf{y}) - o|^2$
- Absolute deviation (ℓ_1) loss: $|M_{\mathbf{x}}(\mathbf{y}) - o|$
- Hinge loss: 1 - $o \cdot M_{\mathbf{x}}(\mathbf{y})$
- Cross-entropy loss (log loss).
- Etc.

Empirical risk minimization:

$$f(\mathsf{x}) = \sum_{i=1}^{n} L\left(M_{\mathsf{x}}(\mathsf{y}^{(i)}), o^{(i)}\right)$$

Solve the optimization problem $\min_{\mathsf{x}} f(\mathsf{x})$.

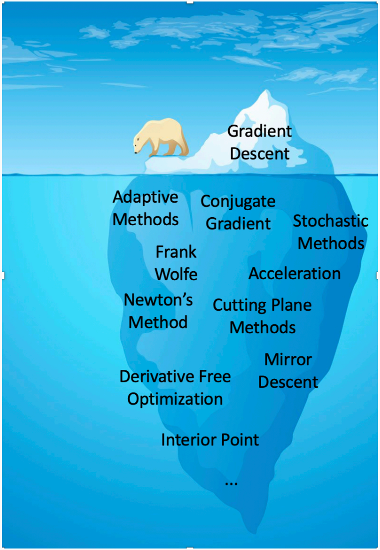
Gradient descent: A greedy algorithm for minimizing functions of multiple variables that often works amazingly well.



Abraham Maslow:

"I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail."

So much to learn!



For $i = 1, \ldots, d$, let x_i be the i^{th} entry of $\mathbf{x}$. Let $\mathbf{e}^{(i)}$ be the i^{th} standard basis vector.

Partial derivative:

$$\frac{\partial f}{\partial x_i}(\mathbf{x}) = \lim_{t \to 0} \frac{f(\mathbf{x} + t\mathbf{e}^{(i)}) - f(\mathbf{x})}{t}$$

Directional derivative:

$$D_{\mathbf{v}} f(\mathbf{x}) = \lim_{t \to 0} \frac{f(\mathbf{x} + t\mathbf{v}) - f(\mathbf{x})}{t}$$

Gradient:

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

Directional derivative:

$$D_{\mathbf{v}} f(\mathbf{x}) = \lim_{t \to 0} \frac{f(\mathbf{x} + t\mathbf{v}) - f(\mathbf{x})}{t} = \nabla f(\mathbf{x})^T \mathbf{v}.$$

Given a function f to minimize, assume we can:

- **Function oracle**: Evaluate $f(\mathbf{x})$ for any $\mathbf{x}$.
- **Gradient oracle**: Evaluate $\nabla f(\mathbf{x})$ for any $\mathbf{x}$.

Linear least-squares regression:

- Given $\mathbf{y}^{(1)}, \ldots \mathbf{y}^{(n)} \in \mathbb{R}^d$, $b^{(1)}, \ldots b^{(n)} \in \mathbb{R}$.
- Want to minimize $f(\mathbf{x}) = \sum_{i=1}^{n} \left(\mathbf{x}^T \mathbf{y}^{(i)} - b^{(i)}\right)^2$.

Matrix view:

Greedy approach: Given a starting point $\mathbf{x}$, make a small adjustment that decreases $f(\mathbf{x})$. In particular, $\mathbf{x} \leftarrow \mathbf{x} + \eta\mathbf{v}$ and $f(\mathbf{x}) \leftarrow f(\mathbf{x} + \eta\mathbf{v})$.

What property might I want in $\mathbf{v}$?

$$D_{\mathbf{v}}f(\mathbf{x}) = \lim_{t\to 0} \frac{f(\mathbf{x} + t\mathbf{v}) - f(\mathbf{x})}{t} = \nabla f(\mathbf{x})^T\mathbf{v}.$$

Prototype algorithm:

- Choose arbitrary starting point $\mathbf{x}^{(1)}$.
- For $i = 1, \ldots, T$:
    - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla f(\mathbf{x}^{(i)})$
- Return $\mathbf{x}^{(t)}$.

η is a step-size parameter. Needs to be chosen ahead of time or adapted on the go.

Example in one dimension:

Claim (Gradient descent = Steepest descent)

$\frac{-\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|_2} = \arg\min_{\mathbf{v}, \|\mathbf{v}\|_2 \leq 1} \nabla f(\mathbf{x})^T \mathbf{v}$

Note: We could have chosen to restrict $\mathbf{v}$ using a different norm. What if we had restricted $\|\mathbf{v}\|_1 \leq 1$? $\|\mathbf{v}\|_\infty \leq 1$? These choices lead to variants of generalized steepest descent..

In general, gradient descent can be proven to converge (and we understand how quickly it converges) for convex functions.

Definition (Convex)

A function f is convex iff for any $\mathbf{x}, \mathbf{y}, \lambda \in [0, 1]$:

$$(1 - \lambda) \cdot f(\mathbf{x}) + \lambda \cdot f(\mathbf{y}) \geq f\left((1 - \lambda) \cdot \mathbf{x} + \lambda \cdot \mathbf{y}\right)$$

Definition (Convex)

A function f is convex iff for any $\mathbf{x}, \mathbf{y}$:

$$f(\mathbf{x}+\mathbf{z}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{z}$$

$$f(\mathbf{x}) - f(\mathbf{y}) \leq \nabla f(\mathbf{x})^T (\mathbf{x}-\mathbf{y})$$

Definition (Convex)

A function f is convex iff for any $\mathbf{x}, \mathbf{y}$:

$$f(\mathbf{x}) - f(\mathbf{y}) \leq \nabla f(\mathbf{x})^T(\mathbf{x} - \mathbf{y})$$

Assume:

- f is convex.
- Lipschitz function: for all $\mathbf{x}$, $\|\nabla f(\mathbf{x})\|_2 \leq G$.
- Starting radius: $\|\mathbf{x}^* - \mathbf{x}^{(1)}\|_2 \leq R$.

Gradient descent:

- Choose number of steps T.
- $\eta = \frac{R}{G\sqrt{T}}$
- For $i = 1, \ldots, T$:
  - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla f(\mathbf{x}^{(i)})$
- Return $\hat{\mathbf{x}} = \arg\min_{\mathbf{x}^{(i)}} f(\mathbf{x}^{(i)})$.
- Alternatively, return $\hat{\mathbf{x}} = \frac{1}{T}\sum_{i=1}^{T} \mathbf{x}^{(i)}$.

Claim (GD Convergence Bound)

If $T \geq \frac{R^2G^2}{\epsilon^2}$, *then* $f(\hat{\mathbf{x}}) \leq f(\mathbf{x}^*) + \epsilon$.

Claim (GD Convergence Bound)

If $T \geq \frac{R^2G^2}{\epsilon^2}$, *then* $f(\hat{\mathbf{x}}) \leq f(\mathbf{x}^*) + \epsilon$.

Claim (GD Convergence Bound)

If $T \geq \frac{R^2 G^2}{\epsilon^2}$, *then* $f(\hat{\mathbf{x}}) \leq f(\mathbf{x}^*) + \epsilon$.

Instead of a single function f to minimize, assume we have an unknown and changing set of objective functions:

$$f_1, \ldots, f_T.$$

- At each time step, choose $\mathbf{x}^{(i)}$.
- f_i is revealed and we pay cost $f_i(\mathbf{x}^{(i)})$
- **Goal**: Minimize $\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)})$.

Objective: Choose $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(T)}$ so that:

$$\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)}) \leq \left[\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x})\right] + \Delta.$$

We want to compete with the best <u>fixed</u> solution in hindsight.

Assume:

- Lipschitz function: for all $\mathbf{x}$, i, $\|\nabla f_i(\mathbf{x})\|_2 \leq G$.
- Starting radius: $\|\mathbf{x}^* - \mathbf{x}^{(1)}\|_2 \leq R$.

Online Gradient descent:

- Choose number of steps T.
- $\eta = \frac{D}{G\sqrt{T}}$
- For $i = 1, \ldots, T$:
  - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla f_i(\mathbf{x}^{(i)})$
- Play $\mathbf{x}^{(i+1)}$.

Claim (OGD Regret Bound)

After T steps, $\Delta = \left[\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)})\right] - \left[\sum_{i=1}^{T} f_i(\mathbf{x}^*)\right] \leq RG\sqrt{T}$

Claim (OGD Regret Bound)

After T steps, $\Delta = \left[\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)})\right] - \left[\sum_{i=1}^{T} f_i(\mathbf{x}^{*})\right] \leq RG\sqrt{T}$

Recall the machine learning setup. In empirical risk minimization, we can typically write:

$$f(\mathbf{x}) = \sum_{j=1}^{n} f_j(\mathbf{x})$$

where f_i is the loss function for a particular data point.

Linear regression:

$$f(\mathbf{x}) = \sum_{j=1}^{n} (\mathbf{x}^T \mathbf{y}^{(j)} - b^{(j)})^2$$

Pick random $j \in 1, \ldots, n$:

$$\mathbb{E}\left[\nabla f_j(\mathbf{x})\right] = \nabla f(\mathbf{x}).$$

But $\nabla f_j(\mathbf{x})$ can be computed in a $1/n$ fraction of the time!

Main idea: Use random approximate gradient in place of actual gradient.

Trade slower convergence for cheaper iterations.

Assume:

- Lipschitz functions: for all $\mathbf{x}$, j, $\|\nabla f_j(\mathbf{x})\|_2 \leq \frac{G'}{n}$.
- Starting radius: $\|\mathbf{x}^* - \mathbf{x}^{(1)}\|_2 \leq R$.

Stochastic Gradient descent:

- Choose number of steps T.
- $\eta = \frac{D}{G'\sqrt{T}}$
- For $i = 1, \ldots, T$:
  - Pick random $j_i \in 1, \ldots, n$.
  - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla f_{j_i}(\mathbf{x}^{(i)})$
- Return $\hat{\mathbf{x}} = \frac{1}{T}\sum_{i=1}^{T} \mathbf{x}^{(i)}$

Claim (SGD Convergence)

After $T = \frac{R^2 G'^2}{\epsilon^2}$ *iteration:*

$$\mathbb{E}\left[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)\right] \leq \epsilon.$$

Claim (SGD Convergence)

After $T = \frac{R^2 G'^2}{\epsilon^2}$ *iteration:*

$$\mathbb{E}\left[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)\right] \leq \epsilon.$$

Claim (SGD Convergence)

After $T = \frac{R^2 G'^2}{\epsilon^2}$ *iteration:*

$$\mathbb{E}\left[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)\right] \leq \epsilon.$$

Number of iterations for error ϵ:

- **Gradient Descent**: $T = \frac{R^2 G^2}{\epsilon^2}$.
- **Stochastic Gradient Descent**: $T = \frac{R^2 G'^2}{\epsilon^2}$.

Always have $G \leq G'$:

$$\|\nabla f(x)\|_2 \leq \|\nabla f_1(x)\|_2 + \ldots + \|\nabla f_n(x)\|_2 \leq n \cdot \frac{G'}{n} = G'.$$

Fair comparison:

$$\frac{R^2 G'^2}{\epsilon^2} = n \cdot \frac{R^2 G^2}{\epsilon^2}$$