Introduction to Artificial Intelligence

Search by Optimization

Andres Mendez-Vazquez

August 30, 2020

Outline

Outline

Introduction

Optimization Searches

Sometimes referred to as iterative improvement or local search.

Introduction

Optimization Searches

Sometimes referred to as iterative improvement or local search.

We will talk about four simple but effective techniques:

1. Gradient Descent
2. Hillclimbing
3. Random Restart Hillclimbing
4. Simulated Annealing

Introduction

Optimization Searches

Sometimes referred to as iterative improvement or local search.

We will talk about four simple but effective techniques:

1. Gradient Descent
2. Hillclimbing
3. Random Restart Hillclimbing
4. Simulated Annealing

Introduction

Optimization Searches

Sometimes referred to as iterative improvement or local search.

We will talk about four simple but effective techniques:

1. Gradient Descent
2. Hillclimbing
3. Random Restart Hillclimbing
4. Simulated Annealing

Introduction

Optimization Searches

Sometimes referred to as iterative improvement or local search.

We will talk about four simple but effective techniques:

1. Gradient Descent
2. Hillclimbing
3. Random Restart Hillclimbing
4. Simulated Annealing

Why?

Local Search

Algorithm that explores the search space of possible solutions in sequential fashion, moving from a current state to a "nearby" one.

Why?

Local Search

Algorithm that explores the search space of possible solutions in sequential fashion, moving from a current state to a "nearby" one.

Why is this important?

- Set of configurations may be too large to be enumerated explicitly.
- Might there not be a poly-time algorithm for finding the maximum of the problem efficiently.
  - Thus local improvements can be a solution to the problem

Why?

Local Search

Algorithm that explores the search space of possible solutions in sequential fashion, moving from a current state to a "nearby" one.

Why is this important?

- Set of configurations may be too large to be enumerated explicitly.
- Might there not be a poly-time algorithm for finding the maximum of the problem efficiently.
  - Thus local improvements can be a solution to the problem

Why?

Local Search

Algorithm that explores the search space of possible solutions in sequential fashion, moving from a current state to a "nearby" one.

Why is this important?

- Set of configurations may be too large to be enumerated explicitly.
- Might there not be a poly-time algorithm for finding the maximum of the problem efficiently.
  - Thus local improvements can be a solution to the problem

Local Search Facts

What is interesting about Local Search

- It keeps track of single current state
- It move only to neighboring states

Local Search Facts

What is interesting about Local Search

- It keeps track of single current state
- It move only to neighboring states

Advantages

- It uses very little memory
- It can often find reasonable solutions in large or infinite (continuous) state spaces

Local Search Facts

What is interesting about Local Search

- It keeps track of single current state
- It move only to neighboring states

Advantages

- It uses very little memory
- It can often find reasonable solutions in large or infinite (continuous) state spaces

Local Search Facts

What is interesting about Local Search

- It keeps track of single current state
- It move only to neighboring states

Advantages

- It uses very little memory
- It can often find reasonable solutions in large or infinite (continuous) state spaces

Outline

The Hill Climbing Heuristic (Greedy Local Search)

What is Hill Climbing?

Hill climbing is a mathematical optimization technique which belongs to the family of local search.

Process

Iteratively it tries to improve the solution by changing one element of the solution so far.

- It is applicable to find a solution for the TSP.

Then

- If the change produces a better solution, an incremental change is made to the new solution.
  - Until no improvements can be made!!!

The Hill Climbing Heuristic (Greedy Local Search)

What is Hill Climbing?

Hill climbing is a mathematical optimization technique which belongs to the family of local search.

Process

Iteratively it tries to improve the solution by changing one element of the solution so far.

- It is applicable to find a solution for the TSP.

Then

- If the change produces a better solution, an incremental change is made to the new solution.
  - Until no improvements can be made!!!

The Hill Climbing Heuristic (Greedy Local Search)

What is Hill Climbing?

Hill climbing is a mathematical optimization technique which belongs to the family of local search.

Process

Iteratively it tries to improve the solution by changing one element of the solution so far.

- It is applicable to find a solution for the TSP.

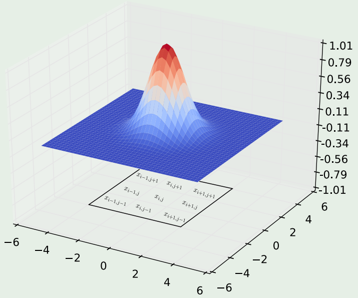
Then

- If the change produces a better solution, an incremental change is made to the new solution.
    - Until no improvements can be made!!!

Example

Maybe you want to minimize something based in a Gaussian Function

$x_{i-1,j+1}$	$x_{i,j+1}$	$x_{i+1,j+1}$
$x_{i-1,j}$	$x_{i,j}$	$x_{i+1,j}$
$x_{i-1,j-1}$	$x_{i,j-1}$	$x_{i+1,j-1}$



Outline

Algorithm

Discrete Space Hill Climbing Algorithm

1. currentNode = startNode
2. while true
3. $\quad L$ = NEIGHBORS(currentNode)
4. $\quad$ nextEval = $-\infty$
5. $\quad$ nextNode = NULL
6. $\quad$ for all $x \in L$
7. $\quad\quad$ if (EVAL(x) > nextEval)
8. $\quad\quad\quad$ nextNode = x
9. $\quad\quad\quad$ nextEval = EVAL(x)
10. $\quad\quad$ if nextEval <= EVAL(currentNode)
11. $\quad\quad\quad$ //Return current node since no better neighbors exist
12. $\quad\quad\quad$ return currentNode
13. $\quad\quad$ currentNode = nextNode

Algorithm

Discrete Space Hill Climbing Algorithm

```
currentNode = startNode
while true
    L = NEIGHBORS(currentNode)
    nextEval = -∞
    nextNode = NULL
    for all x ∈ L
        if (EVAL(x) > nextEval)
            nextNode = x
            nextEval = EVAL(x)
        if nextEval <= EVAL(currentNode)
            //Return current node since no better neighbors exist
            return currentNode
        currentNode = nextNode
```

Algorithm

Discrete Space Hill Climbing Algorithm

```
currentNode = startNode
while true
    L = NEIGHBORS(currentNode)
    nextEval = -∞
    nextNode = NULL
    for all x ∈ L
        if (EVAL(x) > nextEval)
            nextNode = x
            nextEval = EVAL(x)
        if nextEval <= EVAL(currentNode)
            //Return current node since no better neighbors exist
            return currentNode
        currentNode = nextNode
```

Algorithm

Discrete Space Hill Climbing Algorithm

```
currentNode = startNode
while true
    L = NEIGHBORS(currentNode)
    nextEval = −∞
    nextNode = NULL
    for all x ∈ L
        if (EVAL(x) > nextEval)
            nextNode = x
            nextEval = EVAL(x)
        if nextEval <= EVAL(currentNode)
            //Return current node since no better neighbors exist
            return currentNode
        currentNode = nextNode
```

Outline

Example: TSP

Goal

The main idea of TSP is the problem faced by a salesman to visit all towns or cities in an area, without visiting the same town twice.

The Simplest Version

- It assumes that each town is a point in the $\mathbb{R}^2$ plane.
- Thus a node in the problem is a sequence of cities to be visited in order

$$X_i = \langle x_1, x_2, ..., x_n \rangle$$

Where $x_1 == x_n$

The "Eval" function

$$Eval\left(\langle x_1, x_2, ..., x_n \rangle\right) = \sum_{i=1}^{n} \|x_{i+1} - x_i\|$$

Example: TSP

Goal

The main idea of TSP is the problem faced by a salesman to visit all towns or cities in an area, without visiting the same town twice.

The Simplest Version

- It assumes that each town is a point in the $\mathbb{R}^2$ plane.
- Thus a node in the problem is a sequence of cities to be visited in order

$$X_i = \langle \boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n \rangle$$

Where $\boldsymbol{x}_1 == \boldsymbol{x}_n$

The "Eval" function

$$Eval\left(\langle \boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n \rangle\right) = \sum_{i=1}^{n} \|\boldsymbol{x}_{i+1} - \boldsymbol{x}_i\|$$

Example: TSP

Goal

The main idea of TSP is the problem faced by a salesman to visit all towns or cities in an area, without visiting the same town twice.

The Simplest Version

- It assumes that each town is a point in the $\mathbb{R}^2$ plane.
- Thus a node in the problem is a sequence of cities to be visited in order

$$X_i = \langle \boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n \rangle$$

Where $\boldsymbol{x}_1 == \boldsymbol{x}_n$

The "Eval" function

$$Eval\left(\langle \boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n \rangle\right) = \sum_{i=1}^{n} \|\boldsymbol{x}_{i+1} - \boldsymbol{x}_i\|$$

Stuff to think about

Something Notable

The definition of the neighborhoods is not obvious or unique in general.

In our example

A new neighbor will be created by swapping two cities

Stuff to think about

Something Notable

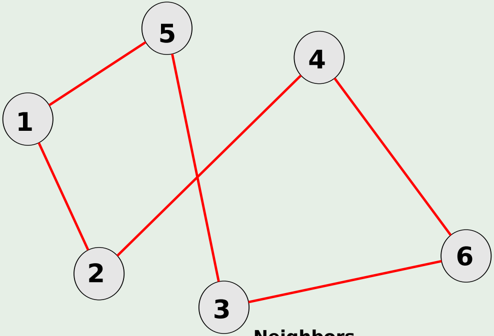
The definition of the neighborhoods is not obvious or unique in general.

In our example

A new neighbor will be created by swapping two cities

Example

Local State

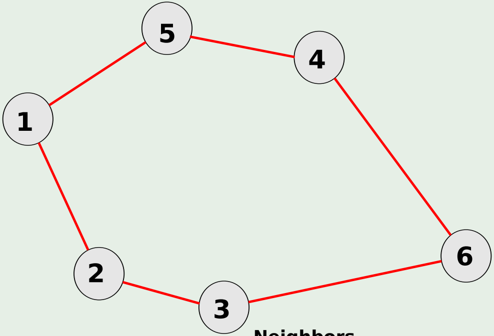


Neighbors

$$\langle 1, 5, \boxed{3}, 6, \boxed{4}, 2, 1 \rangle$$

Example

New Local State



Neighbors

$$\langle 1, 5, \boxed{4}, 6, \boxed{3}, 2, 1 \rangle$$

Actually

This idea of neighborhood

It is used for Genetic Algorithm (GA) for mutating elements in the population!!!

Thus

By looking at elements of the neighborhood

- New permutations by swapping cities

It is possible to obtain local improvements by improving

$$Eval\left(\langle x_1, x_2, \ldots, x_n \rangle\right) = \sum_{i=1}^{n} \|x_{i+1} - x_i\|$$

Actually

This idea of neighborhood

It is used for Genetic Algorithm (GA) for mutating elements in the population!!!

Thus

By looking at elements of the neighborhood

- New permutations by swapping cities

It is possible to obtain local improvements by improving

$$Eval\left(\langle x_1, x_2, ..., x_n \rangle\right) = \sum_{i=1}^{n} \left\| x_{i+1} - x_i \right\|$$

Actually

This idea of neighborhood

It is used for Genetic Algorithm (GA) for mutating elements in the population!!!

Thus

By looking at elements of the neighborhood

- New permutations by swapping cities

It is possible to obtain local improvements by improving

$$Eval\left(\langle \boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n \rangle\right) = \sum_{i=1}^{n} \left\| \boldsymbol{x}_{i+1} - \boldsymbol{x}_i \right\|$$

However!!!

NON-ADMISSIBLE SEARCH

Fact: as problem graphs are so huge, waiting for the algorithm to terminate becomes unacceptable.

However!!!

NON-ADMISSIBLE SEARCH

Fact: as problem graphs are so huge, waiting for the algorithm to terminate becomes unacceptable.

Then

- Heuristic search algorithms were developed that do not insist on the optimal solution.
- Some strategies even sacrifice completeness and may fail to find a solution of a solvable problem instance.
  - **But the have strategies to decrease the likelihood of such errors.**

- Enforced Hill Climbing
- Weighted A*
- Overconsistent A*
- etc

However!!!

NON-ADMISSIBLE SEARCH

Fact: as problem graphs are so huge, waiting for the algorithm to terminate becomes unacceptable.

Then

- Heuristic search algorithms were developed that do not insist on the optimal solution.
- Some strategies even sacrifice completeness and may fail to find a solution of a solvable problem instance.
  - **But the have strategies to decrease the likelihood of such errors.**

Thus

Several algorithms have been adapted to these type of constraints:

- **Enforced Hill Climbing**
- Weighted A*
- Overconsistent A*
- etc

However!!!

NON-ADMISSIBLE SEARCH

Fact: as problem graphs are so huge, waiting for the algorithm to terminate becomes unacceptable.

Then

- Heuristic search algorithms were developed that do not insist on the optimal solution.
- Some strategies even sacrifice completeness and may fail to find a solution of a solvable problem instance.
  - **But the have strategies to decrease the likelihood of such errors.**

Thus

Several algorithms have been adapted to these type of constraints:

- **Enforced Hill Climbing**
- Weighted A*
- Overconsistent A*
- etc

However!!!

NON-ADMISSIBLE SEARCH

Fact: as problem graphs are so huge, waiting for the algorithm to terminate becomes unacceptable.

Then

- Heuristic search algorithms were developed that do not insist on the optimal solution.
- Some strategies even sacrifice completeness and may fail to find a solution of a solvable problem instance.
  - **But the have strategies to decrease the likelihood of such errors.**

Thus

Several algorithms have been adapted to these type of constraints:

- Enforced Hill Climbing
- Weighted A*
- Overconsistent A*
- etc

However!!!

NON-ADMISSIBLE SEARCH

Fact: as problem graphs are so huge, waiting for the algorithm to terminate becomes unacceptable.

Then

- Heuristic search algorithms were developed that do not insist on the optimal solution.
- Some strategies even sacrifice completeness and may fail to find a solution of a solvable problem instance.
  - **But the have strategies to decrease the likelihood of such errors.**

Thus

Several algorithms have been adapted to these type of constraints:

- **Enforced Hill Climbing**
- Weighted A*
- Overconsistent A*
- etc

However!!!

NON-ADMISSIBLE SEARCH

Fact: as problem graphs are so huge, waiting for the algorithm to terminate becomes unacceptable.

Then

- Heuristic search algorithms were developed that do not insist on the optimal solution.
- Some strategies even sacrifice completeness and may fail to find a solution of a solvable problem instance.
  - **But the have strategies to decrease the likelihood of such errors.**

Thus

Several algorithms have been adapted to these type of constraints:

- **Enforced Hill Climbing**
- Weighted A*
- Overconsistent A*
- etc

However!!!

NON-ADMISSIBLE SEARCH

Fact: as problem graphs are so huge, waiting for the algorithm to terminate becomes unacceptable.

Then

- Heuristic search algorithms were developed that do not insist on the optimal solution.
- Some strategies even sacrifice completeness and may fail to find a solution of a solvable problem instance.
  - **But the have strategies to decrease the likelihood of such errors.**

Thus

Several algorithms have been adapted to these type of constraints:

- **Enforced Hill Climbing**
- Weighted A*
- Overconsistent A*
- etc

However!!!

NON-ADMISSIBLE SEARCH

Fact: as problem graphs are so huge, waiting for the algorithm to terminate becomes unacceptable.

Then

- Heuristic search algorithms were developed that do not insist on the optimal solution.
- Some strategies even sacrifice completeness and may fail to find a solution of a solvable problem instance.
  - **But the have strategies to decrease the likelihood of such errors.**

Thus

Several algorithms have been adapted to these type of constraints:

- **Enforced Hill Climbing**
- Weighted A*
- Overconsistent A*
- etc

Outline

Enforced Hill Climbing

Problem

- Hill Climbing does not necessarily find optimal solutions
- Hill Climbing can be trapped in state space problem graph with dead ends

Enforced Hill Climbing

Problem

- Hill Climbing does not necessarily find optimal solutions
- Hill Climbing can be trapped in state space problem graph with dead ends

We want something more stable

A more stable version is enforced hill-climbing.

- It picks a successor node, only if it has a strictly better evaluation than the current node.

- After all the better node might not be in the immediate neighborhood!!!

Enforced Hill Climbing

Problem

- Hill Climbing does not necessarily find optimal solutions
- Hill Climbing can be trapped in state space problem graph with dead ends

We want something more stable

A more stable version is enforced hill-climbing.

- It picks a successor node, only if it has a strictly better evaluation than the current node.

How?

- Using Breadth-First Search!!!
- After all the better node might not be in the immediate neighborhood!!!

Enforced Hill Climbing

Problem

- Hill Climbing does not necessarily find optimal solutions
- Hill Climbing can be trapped in state space problem graph with dead ends

We want something more stable

A more stable version is enforced hill-climbing.

- It picks a successor node, only if it has a strictly better evaluation than the current node.

How?

- Using Breadth-First Search!!!
- After all the better node might not be in the immediate neighborhood!!!

Enforced Hill Climbing

Problem

- Hill Climbing does not necessarily find optimal solutions
- Hill Climbing can be trapped in state space problem graph with dead ends

We want something more stable

A more stable version is enforced hill-climbing.

- It picks a successor node, only if it has a strictly better evaluation than the current node.

How?

- Using Breadth-First Search!!!
- After all the better node might not be in the immediate neighborhood!!!

Enforced Hill Climbing

Problem

- Hill Climbing does not necessarily find optimal solutions
- Hill Climbing can be trapped in state space problem graph with dead ends

We want something more stable

A more stable version is enforced hill-climbing.

- It picks a successor node, only if it has a strictly better evaluation than the current node.

How?

- Using Breadth-First Search!!!
- After all the better node might not be in the immediate neighborhood!!!

Algorithm

Enforced-Hill-Climbing$(s, h, Expand)$

Input: Implicitly given graph with start node s, successor generating function $Expand$ and a heuristic h

Output: Path to node $t \in T$

```
u = s
ht = h(s)
while (ht ≠ 0)
        (u', ht') = EHC-BFS(u, h, Expand)
        if (ht' = ∞) return ∅
        u = u'
        ht = ht'
return Path(u)
```

Algorithm

EHC-BFS$(u, h, Expand)$

Input: Node u with evaluation $h(u)$

Output: Node v with evaluation $h(v) < h(u)$ or failure

```
Enqueue (Q, u)
while (Q ≠ 0)
        v = Dequeue (Q)
        if (h(v) < h(u)) return (v, h (v))
        Succ (v) = Expand (v)
        for each w ∈ Succ (v)
                Enqueue (w)
return (·, ∞)
```

Outline

The Problem with Dead-Ends

Definition

Given a planning task, a state S is called a dead end **if and only** if it is reachable and no sequence of actions achieves the goal from it.

We will look more about this in Classic Planning

"The FF Planning System: Fast Plan Generation Through Heuristic Search"

The Problem with Dead-Ends

Definition

Given a planning task, a state S is called a dead end **if and only** if it is reachable and no sequence of actions achieves the goal from it.

We will look more about this in Classic Planning

"The FF Planning System: Fast Plan Generation Through Heuristic Search"

Then

Theorem (Completeness Enforced Hill-Climbing)

If the state space graph contains no dead-ends then EHC-BFS will find a solution.

Then

Theorem (Completeness Enforced Hill-Climbing)

If the state space graph contains no dead-ends then EHC-BFS will find a solution.

Proof

1. There is only one case that the algorithm does not find a solution.
   1. For some intermediate node v, no better evaluated node v can be found.
2. Since BFS is a complete search method $\Longrightarrow$ BFS will find a node on a solution path with better evaluation.

Then

Theorem (Completeness Enforced Hill-Climbing)

If the state space graph contains no dead-ends then EHC-BFS will find a solution.

Proof

1. There is only one case that the algorithm does not find a solution.
   1. For some intermediate node v, no better evaluated node v can be found.
2. Since BFS is a complete search method $\Longrightarrow$ BFS will find a node on a solution path with better evaluation.

Finally

In fact, because there are not dead-ends, the evaluation h will decrease along a solution path until finding a $t \in T$ such that $h(t) = 0$.

Then

Theorem (Completeness Enforced Hill-Climbing)

If the state space graph contains no dead-ends then EHC-BFS will find a solution.

Proof

1. There is only one case that the algorithm does not find a solution.
   1. For some intermediate node v, no better evaluated node v can be found.
2. Since BFS is a complete search method $\Longrightarrow$ BFS will find a node on a solution path with better evaluation.

Finally

In fact, because there are not dead-ends, the evaluation h will decrease along a solution path until finding a $t \in T$ such that $h(t) = 0$.

Then

Theorem (Completeness Enforced Hill-Climbing)

If the state space graph contains no dead-ends then EHC-BFS will find a solution.

Proof

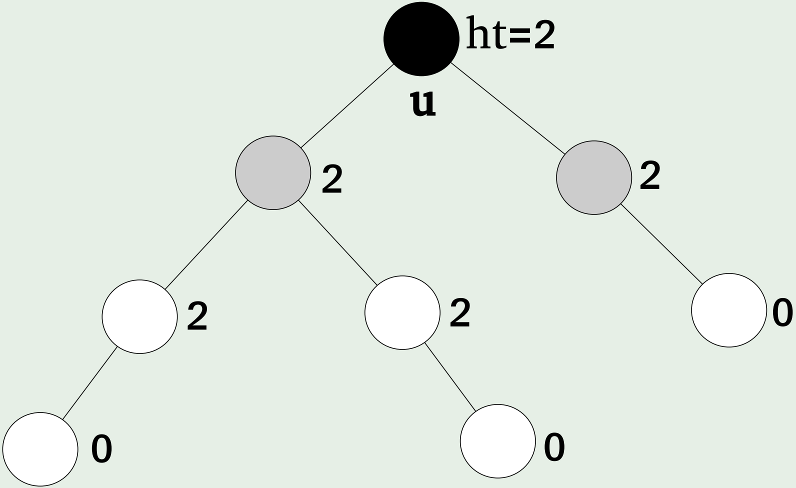
1. There is only one case that the algorithm does not find a solution.
    1. For some intermediate node v, no better evaluated node v can be found.
2. Since BFS is a complete search method $\Longrightarrow$ BFS will find a node on a solution path with better evaluation.

Finally

In fact, because there are not dead-ends, the evaluation h will decrease along a solution path until finding a $t \in T$ such that $h(t) = 0$.

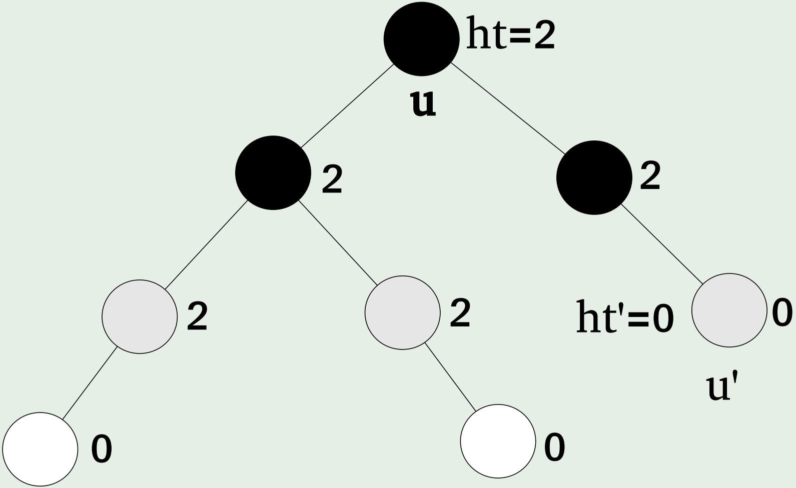
Example

Example of enforced hill-climbing (two iterations)



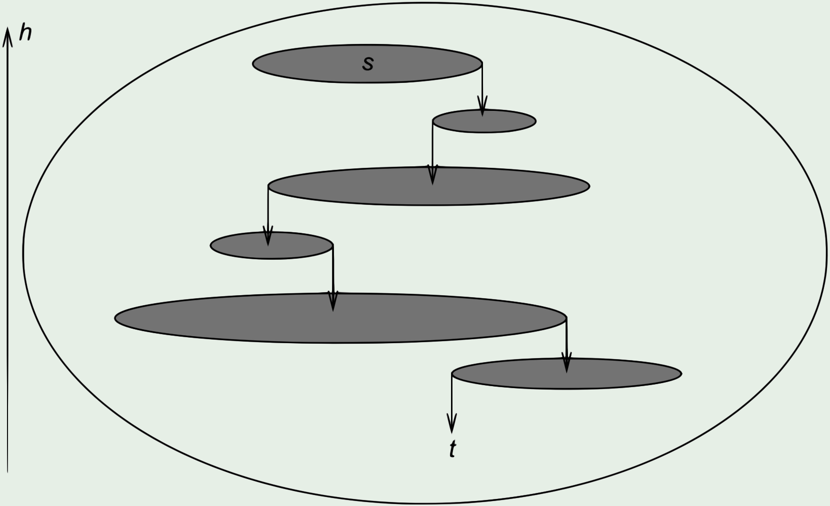
Example

Example of enforced hill-climbing (two iterations)



It is more

Search plateaus generated with enforced hill-climbing



Even with this...

Unavoidable

Hill climbing is subject to getting stuck in a variety of local conditions.

Even with this...

Unavoidable

Hill climbing is subject to getting stuck in a variety of local conditions.

Two Solutions

- Random restart hill climbing
- Simulated annealing

Even with this...

Unavoidable

Hill climbing is subject to getting stuck in a variety of local conditions.

Two Solutions

- Random restart hill climbing
- Simulated annealing

Random Restart Hill Climbing

Pretty obvious what this is...

- Generate a random start state
- Run hill climbing and store answer
- Iterate, keeping the current best answer as you go
  - Stopping... when?

Random Restart Hill Climbing

Pretty obvious what this is...

- Generate a random start state
- Run hill climbing and store answer
- Iterate, keeping the current best answer as you go
  - Stopping... when?

Random Restart Hill Climbing

Pretty obvious what this is...

- Generate a random start state
- Run hill climbing and store answer
- Iterate, keeping the current best answer as you go
    - Stopping... when?

Random Restart Hill Climbing

Pretty obvious what this is...

- Generate a random start state
- Run hill climbing and store answer
- Iterate, keeping the current best answer as you go
  - Stopping... when?

Outline

We can do better by using Simulated Annealing

Definition of terms

- Annealing: Solid material is heated past its melting point and then cooled back into a solid state.
- Structural properties of the cooled solid depends on the rate of cooling.

We can do better by using Simulated Annealing

Definition of terms

- Annealing: Solid material is heated past its melting point and then cooled back into a solid state.
- Structural properties of the cooled solid depends on the rate of cooling.

Authors

- Metropolis et al. (1953) simulated the change in energy of the system as it cools, until it converges to a steady "frozen" state.
- Kirkpatrick et al. (1983) suggested using SA for optimization, applied it to VLSI design and TSP

We can do better by using Simulated Annealing

Definition of terms

- Annealing: Solid material is heated past its melting point and then cooled back into a solid state.
- Structural properties of the cooled solid depends on the rate of cooling.

Authors

- Metropolis et al. (1953) simulated the change in energy of the system as it cools, until it converges to a steady "frozen" state.
- Kirkpatrick et al. (1983) suggested using SA for optimization, applied it to VLSI design and TSP

We can do better by using Simulated Annealing

Definition of terms

- Annealing: Solid material is heated past its melting point and then cooled back into a solid state.
- Structural properties of the cooled solid depends on the rate of cooling.

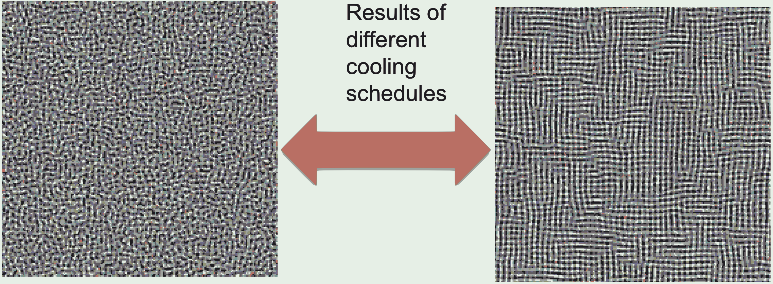
Authors

- Metropolis et al. (1953) simulated the change in energy of the system as it cools, until it converges to a steady "frozen" state.
- Kirkpatrick et al. (1983) suggested using SA for optimization, applied it to VLSI design and TSP

Analogy

Slowly cool down a heated solid, so that all particles arrange in the ground energy state

At each temperature wait until the solid reaches its thermal equilibrium



Now

Concepts

- $f : \Omega \to \mathbb{R}$ be an objective function
- $\omega *$ the global minimum
  - $f(\omega *) \leq f(\omega)$ for all $\omega \in \Omega$

Now

Concepts

- $f : \Omega \rightarrow \mathbb{R}$ be an objective function
- $\omega*$ the global minimum
  - $f(\omega*) \leq f(\omega)$ for all $\omega \in \Omega$

In addition

There is a neighborhood function $N(\omega)$ for $\omega \in \Omega$

- Then a new ω' is generated randomly or by using a predefined rule

Now

Concepts

- $f : \Omega \rightarrow \mathbb{R}$ be an objective function
- $\omega*$ the global minimum
  - $f(\omega*) \leq f(\omega)$ for all $\omega \in \Omega$

In addition

There is a neighborhood function $N(\omega)$ for $\omega \in \Omega$

Thus

- Simulated Annealing starts with an initial solution $\omega \in \Omega$
- Then a new ω' is generated randomly or by using a predefined rule.

Now

Concepts

- $f : \Omega \to \mathbb{R}$ be an objective function
- $\omega*$ the global minimum
  - $f(\omega*) \leq f(\omega)$ for all $\omega \in \Omega$

In addition

There is a neighborhood function $N(\omega)$ for $\omega \in \Omega$

Thus

- Simulated Annealing starts with an initial solution $\omega \in \Omega$
- Then a new ω' is generated randomly or by using a predefined rule.

Now

Concepts

- $f : \Omega \to \mathbb{R}$ be an objective function
- $\omega*$ the global minimum
  - $f(\omega*) \leq f(\omega)$ for all $\omega \in \Omega$

In addition

There is a neighborhood function $N(\omega)$ for $\omega \in \Omega$

Thus

- Simulated Annealing starts with an initial solution $\omega \in \Omega$
- Then a new ω' is generated randomly or by using a predefined rule.

Now

Concepts

- $f : \Omega \to \mathbb{R}$ be an objective function
- $\omega*$ the global minimum
  - $f(\omega*) \leq f(\omega)$ for all $\omega \in \Omega$

In addition

There is a neighborhood function $N(\omega)$ for $\omega \in \Omega$

Thus

- Simulated Annealing starts with an initial solution $\omega \in \Omega$
- Then a new ω' is generated randomly or by using a predefined rule.

Outline

The Metropolis Acceptance Criterion

Something Notable

The criterion models how a thermodynamic system moves from the current solution $\omega \in \Omega$ to a candidate solution $\omega' \in N(\omega)$.

Acceptance Probability

$$P(\text{Accept } \omega') = \begin{cases} \exp\left\{-\frac{f(\omega')-f(\omega)}{t_k}\right\} & \text{if } f(\omega') - f(\omega) > 0 \\ 1 & \text{if } f(\omega') - f(\omega) \leq 0 \end{cases}$$

Where t_k is a temperature parameter at iteration k

$$t_k > 0 \text{ for all } k \text{ and } \lim_{k \to +\infty} t_k = 0$$

The Metropolis Acceptance Criterion

Something Notable

The criterion models how a thermodynamic system moves from the current solution $\omega \in \Omega$ to a candidate solution $\omega' \in N(\omega)$.

Acceptance Probability

$$P(\text{Accept } \omega') = \begin{cases} \exp\left\{-\frac{f(\omega')-f(\omega)}{t_k}\right\} & \text{if } f(\omega') - f(\omega) > 0 \\ 1 & \text{if } f(\omega') - f(\omega) \leq 0 \end{cases}$$

Where t_k is a temperature parameter at iteration k

$$t_k > 0 \text{ for all } k \text{ and } \lim_{k \to +\infty} t_k = 0$$

The Metropolis Acceptance Criterion

Something Notable

The criterion models how a thermodynamic system moves from the current solution $\omega \in \Omega$ to a candidate solution $\omega' \in N(\omega)$.

Acceptance Probability

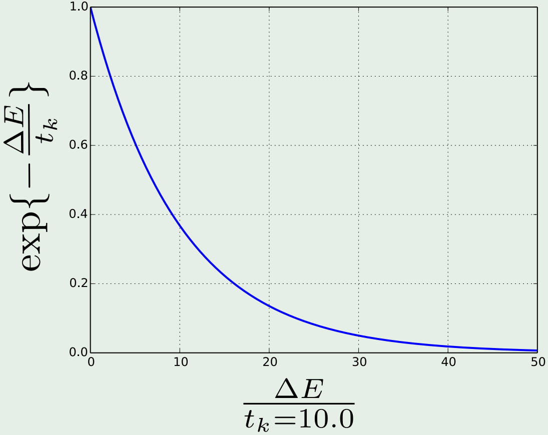
$$P(\text{Accept } \omega') = \begin{cases} \exp\left\{-\frac{f(\omega')-f(\omega)}{t_k}\right\} & \text{if } f(\omega') - f(\omega) > 0 \\ 1 & \text{if } f(\omega') - f(\omega) \leq 0 \end{cases}$$

Where t_k is a temperature parameter at iteration k

$$t_k > 0 \text{ for all } k \text{ and } \lim_{k \to +\infty} t_k = 0$$

We call $\Delta E = f(\omega') - f(\omega)$

Something Quite Interesting



Thus

If $\frac{\Delta E}{t_k} \to \infty$

$$\exp\left\{-\frac{\Delta E}{t_k}\right\} \to 0$$

If $\frac{\Delta E}{t_k} \to 0$

$$\exp\left\{-\frac{\Delta E}{t_k}\right\} \to 1$$

Thus

If $\frac{\Delta E}{t_k} \to \infty$

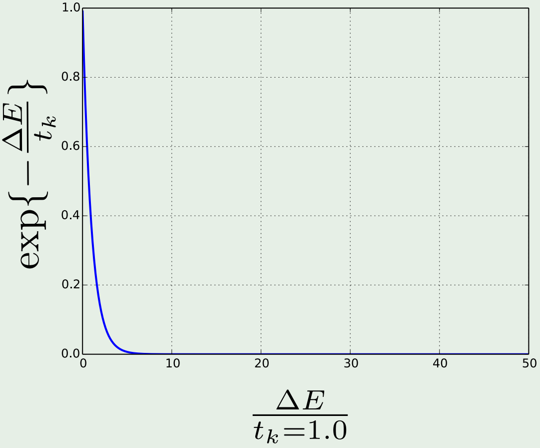
$$\exp\left\{-\frac{\Delta E}{t_k}\right\} \to 0$$

If $\frac{\Delta E}{t_k} \to 0$

$$\exp\left\{-\frac{\Delta E}{t_k}\right\} \to 1$$

We call $\Delta E = f(\omega') - f(\omega)$

Something Quite Interesting



Meaning

The larger is the t_k

The more probable we accept larger jumps from $f(\omega)$.

The smaller is t_k

We tend to accept only small jumps from $f(\omega)$.

Meaning

The larger is the t_k

The more probable we accept larger jumps from $f(\omega)$.

The smaller is t_k

We tend to accept only small jumps from $f(\omega)$.

Thus if the temperature is slowly reduced

Something Notable

The system will reach an equilibrium at certain iteration k

This equilibrium follows the Boltzmann distribution

It is the probability of the system being in state $\omega \in \Omega$ with energy $f(\omega)$ at temperature T such that

$$P\left\{\text{System in state } \omega \text{ at temp } T\right\} = \frac{\exp\left\{-\frac{f(\omega)}{t_k}\right\}}{\sum_{\omega'' \in \Omega} \exp\left\{-\frac{f(\omega'')}{t_k}\right\}}$$

Thus if the temperature is slowly reduced

Something Notable

The system will reach an equilibrium at certain iteration k

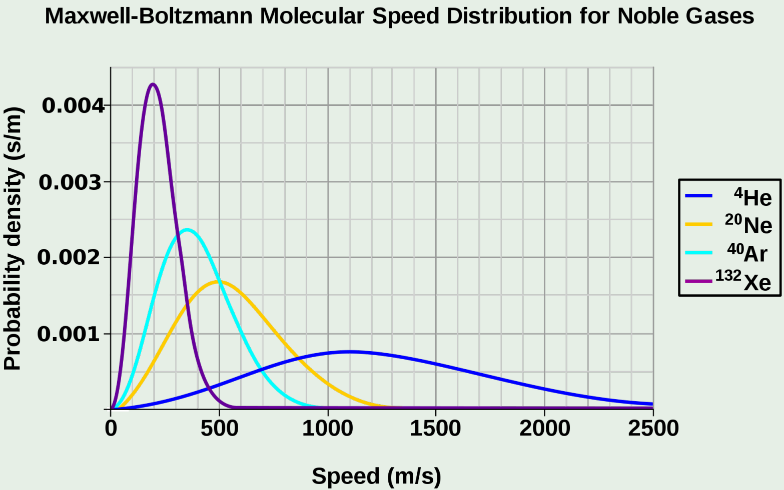
This equilibrium follows the Boltzmann distribution

It is the probability of the system being in state $\omega \in \Omega$ with energy $f(\omega)$ at temperature T such that

$$P\left\{\text{System in state } \omega \text{ at temp } T\right\} = \frac{\exp\left\{-\frac{f(\omega)}{t_k}\right\}}{\sum_{\omega'' \in \Omega} \exp\left\{-\frac{f(\omega'')}{t_k}\right\}}$$

Example of Boltzmann distribution

For Gases



Outline

Parameters

We have

1. M_k number of steps at each temperature t_k
2. ϵ_t reduction in temperature
3. ϵ threshold for stopping criteria

Parameters

We have

1. M_k number of steps at each temperature t_k
2. ϵ_t reduction in temperature
3. ϵ threshold for stopping criteria

Parameters

We have

1. M_k number of steps at each temperature t_k
2. ϵ_t reduction in temperature
3. ϵ threshold for stopping criteria

Parameters

We have

1. M_k number of steps at each temperature t_k
2. ϵ_t reduction in temperature
3. ϵ threshold for stopping criteria

Algorithm

Simulated_Annealing$(\omega, M_k, \epsilon_t, \epsilon, t_k, f)$

$\Delta E = \infty$

while $|\Delta E| > \epsilon$

  for $i = 0, 1, 2, ..., M_k$

    Randomly select ω' **in** $N(\omega)$

    $\Delta E = f(\omega') - f(\omega)$

    if $\Delta E \leq 0$

      $\omega = \omega'$

    if $\Delta E > 0$

      $\omega = \omega'$ **with probability** $Pr\{Accepted\} = \exp\left\{\frac{-\Delta E}{t_k}\right\}$

  $t_k = t_k - \epsilon_t$ **# We can also use** $t_k = \epsilon_t \cdot t_k$

Meaning of probability $Pr\left\{Accepted\right\} = \exp\left\{\frac{-\Delta E}{t_k}\right\}$

Basically

You draw a random value α from the distribution $U\left(0,1\right)$

Then if

$\exp\left\{\frac{-\Delta E}{t_k}\right\} > \alpha$ you make $\omega = \omega'$

Else

You reject state ω'

Meaning of probability $Pr\left\{Accepted\right\} = \exp\left\{\frac{-\Delta E}{t_k}\right\}$

Basically

You draw a random value α from the distribution $U\left(0,1\right)$

Then if

$\exp\left\{\frac{-\Delta E}{t_k}\right\} > \alpha$ you make $\omega = \omega'$

Else

You reject state ω'

Meaning of probability $Pr\left\{Accepted\right\} = \exp\left\{\frac{-\Delta E}{t_k}\right\}$

Basically

You draw a random value α from the distribution $U(0,1)$

Then if

$\exp\left\{\frac{-\Delta E}{t_k}\right\} > \alpha$ you make $\omega = \omega'$

Else

You reject state ω'

Outline

What happen if you have the following

What if you have a cost function with the following characteristics

- It is parametrically defined.
- It is smooth.

What happen if you have the following

What if you have a cost function with the following characteristics

- It is parametrically defined.
- It is smooth.

We can use the following technique

Gradient Descent

What happen if you have the following

What if you have a cost function with the following characteristics

- It is parametrically defined.
- It is smooth.

We can use the following technique

Gradient Descent

Example

Consider the following hypothetical problem

1. x =sales price of Intel's newest chip (in \$1000's of dollars)
2. $f(x)$ =profit per chip when it costs \$1000.00 dollars

Example

Consider the following hypothetical problem

1. x =sales price of Intel's newest chip (in \$1000's of dollars)
2. $f(x)$ =profit per chip when it costs \$1000.00 dollars

Assume that Intel's marketing research team has found that the profit per chip (as a function of x) is

$$f(x) = x^2 - x^3$$

Example

Consider the following hypothetical problem

1. x =sales price of Intel's newest chip (in \$1000's of dollars)
2. $f(x)$ =profit per chip when it costs \$1000.00 dollars

Assume that Intel's marketing research team has found that the profit per chip (as a function of x) is

$$f(x) = x^2 - x^3$$

Assume

we must have x non-negative and no greater than one in percentage.

Example

Consider the following hypothetical problem

1. x =sales price of Intel's newest chip (in \$1000's of dollars)
2. $f(x)$ =profit per chip when it costs \$1000.00 dollars

Assume that Intel's marketing research team has found that the profit per chip (as a function of x) is

$$f(x) = x^2 - x^3$$

Assume

we must have x non-negative and no greater than one in percentage.

Thus

Maximization

Objective function is profit $f(x)$ that needs to be maximized.

Thus

Solution to the optimization problem will be the optimum chip sales price.

Thus

Maximization

Objective function is profit $f(x)$ that needs to be maximized.

Thus

Solution to the optimization problem will be the optimum chip sales price.

Outline

Important Notes about Optimization Problems

What we want

We are interested in knowing those points $\boldsymbol{x} \in D \subseteq \mathbb{R}^n$ such that $f(\boldsymbol{x}_0) \leq f(\boldsymbol{x})$ of $f(\boldsymbol{x}_0) \geq f(\boldsymbol{x})$

Or

A minimum or a maximum point x_0.

The process of finding x_0

It is a search process using certain properties of the function.

Important Notes about Optimization Problems

What we want

We are interested in knowing those points $\boldsymbol{x} \in D \subseteq \mathbb{R}^n$ such that $f(\boldsymbol{x}_0) \leq f(\boldsymbol{x})$ of $f(\boldsymbol{x}_0) \geq f(\boldsymbol{x})$

Or

A minimum or a maximum point $\boldsymbol{x}_0$.

The process of finding $\boldsymbol{x}_0$

It is a search process using certain properties of the function.

Important Notes about Optimization Problems

What we want

We are interested in knowing those points $\boldsymbol{x} \in D \subseteq \mathbb{R}^n$ such that $f(\boldsymbol{x}_0) \leq f(\boldsymbol{x})$ of $f(\boldsymbol{x}_0) \geq f(\boldsymbol{x})$

Or

A minimum or a maximum point $\boldsymbol{x}_0$.

The process of finding $\boldsymbol{x}_0$

It is a search process using certain properties of the function.

Thus

Local vs Global Minimum/Maximum

- Local minimum/maximum is the minimum/maximum in a neighborhood $L \subset D$.
- Global minimum/maximum is the lowest value of f for all $x \in D$
  - it is usually much harder to find.

Thus

Local vs Global Minimum/Maximum

- Local minimum/maximum is the minimum/maximum in a neighborhood $L \subset D$.
- Global minimum/maximum is the lowest value of f for all $x \in D$
  - it is usually much harder to find.

Examples of minimums

Thus

Local vs Global Minimum/Maximum

- Local minimum/maximum is the minimum/maximum in a neighborhood $L \subset D$.
- Global minimum/maximum is the lowest value of f for all $x \in D$
  - **it is usually much harder to find.**

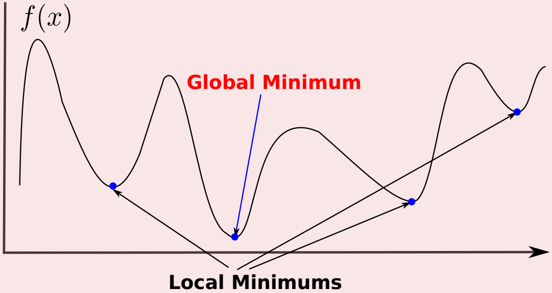
Examples of minimums

Thus

Local vs Global Minimum/Maximum

- Local minimum/maximum is the minimum/maximum in a neighborhood $L \subset D$.
- Global minimum/maximum is the lowest value of f for all $x \in D$
  - **it is usually much harder to find.**

Examples of minimums



Furthermore

Something Notable

Optimization is a very difficult problem in general.

- Especially when x is high dimensional, unless f is simple (e.g. linear) and known analytically.

Furthermore

Something Notable

Optimization is a very difficult problem in general.

- Especially when x is high dimensional, unless f is simple (e.g. linear) and known analytically.

We have this classification

1. Analytical methods - They work fine when f can be handled in an analytical way.
2. Numerical methods - Here, we use inherent properties of the function like the rate of change of the function.

Furthermore

Something Notable

Optimization is a very difficult problem in general.

- Especially when x is high dimensional, unless f is simple (e.g. linear) and known analytically.

We have this classification

1. Analytical methods - They work fine when f can be handled in an analytical way.
2. Numerical methods - Here, we use inherent properties of the function like the rate of change of the function.

In our case

We will look at the Gradient Descent Method!!!

Furthermore

Something Notable

Optimization is a very difficult problem in general.

- Especially when x is high dimensional, unless f is simple (e.g. linear) and known analytically.

We have this classification

1. Analytical methods - They work fine when f can be handled in an analytical way.
2. Numerical methods - Here, we use inherent properties of the function like the rate of change of the function.

In our case

We will look at the Gradient Descent Method!!!

Furthermore

Something Notable

Optimization is a very difficult problem in general.

- Especially when x is high dimensional, unless f is simple (e.g. linear) and known analytically.

We have this classification

1. Analytical methods - They work fine when f can be handled in an analytical way.
2. Numerical methods - Here, we use inherent properties of the function like the rate of change of the function.

In our case

We will look at the Gradient Descent Method!!!

Analytical Method: Differentiating

Assume f is known analytically and twice differentiable

The critical points of f, i.e. the points of potential maximum or minimum, can be found using the equation:

$$\frac{df}{dx} = 0 \tag{1}$$

Analytical Method: Differentiating

Assume f is known analytically and twice differentiable

The critical points of f, i.e. the points of potential maximum or minimum, can be found using the equation:

$$\frac{df}{dx} = 0 \tag{1}$$

For example

$$\frac{df\left(x\right)}{dx} = \frac{d\left[x^2 - x^3\right]}{dx} = 2x - 3x^2 = 0$$

Analytical Method: Differentiating

Assume f is known analytically and twice differentiable

The critical points of f, i.e. the points of potential maximum or minimum, can be found using the equation:

$$\frac{df}{dx} = 0 \tag{1}$$

For example

$$\frac{df(x)}{dx} = \frac{d\left[x^2 - x^3\right]}{dx} = 2x - 3x^2 = 0$$

Finding the roots $x_1, x_2, \ldots, x_k$

$$x = \frac{2}{3}$$

Analytical Method: Differentiating

Assume f is known analytically and twice differentiable

The critical points of f, i.e. the points of potential maximum or minimum, can be found using the equation:

$$\frac{df}{dx} = 0 \tag{1}$$

For example

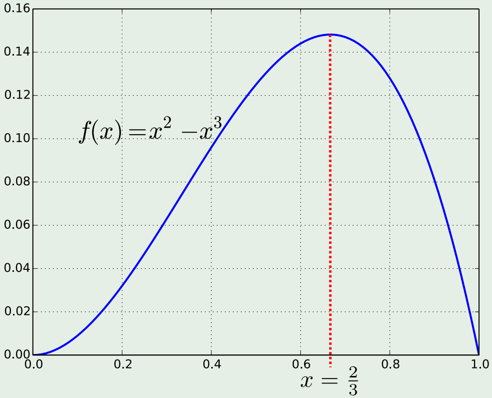
$$\frac{df(x)}{dx} = \frac{d\left[x^2 - x^3\right]}{dx} = 2x - 3x^2 = 0$$

Finding the roots $x_1, x_2, ..., x_k$

$$x = \frac{2}{3}$$

Example

We have the following



Do we have a Maximum or a Minimum

Second Derivative Test

The sign of the second derivative tells if each of those points is a maximum or a minimum:

1. If $\frac{d^2f(x_i)}{dx^2} > 0$ for $x = x_i$ then x_i is a minimum.
2. If $\frac{d^2f(x_i)}{dx^2} < 0$ for $x = x_i$ then x_i is a maximum.

Do we have a Maximum or a Minimum

Second Derivative Test

The sign of the second derivative tells if each of those points is a maximum or a minimum:

1. If $\frac{d^2 f(x_i)}{dx^2} > 0$ for $x = x_i$then x_i is a minimum.
2. If $\frac{d^2 f(x_i)}{dx^2} < 0$ for $x = x_i$then x_i is a maximum.

Do we have a Maximum or a Minimum

Second Derivative Test

The sign of the second derivative tells if each of those points is a maximum or a minimum:

1. If $\frac{d^2 f(x_i)}{dx^2} > 0$ for $x = x_i$then x_i is a minimum.
2. If $\frac{d^2 f(x_i)}{dx^2} < 0$ for $x = x_i$then x_i is a maximum.

Do we have a Maximum or a Minimum

Second Derivative Test

The sign of the second derivative tells if each of those points is a maximum or a minimum:

1. If $\frac{d^2 f(x_i)}{dx^2} > 0$ for $x = x_i$then x_i is a minimum.
2. If $\frac{d^2 f(x_i)}{dx^2} < 0$ for $x = x_i$then x_i is a maximum.

Example

In our case

$$\frac{d^2 f(x)}{dx^2} = 2 - 6x$$

Then

$$\frac{d^2 f\left(\frac{2}{3}\right)}{dx^2} = 2 - 6 \times \frac{2}{3} = 2 - 4 = -2$$

Maximum Profit for the $1000.00 dollar Chip

$$\$667.00$$

Example

In our case

$$\frac{d^2 f(x)}{dx^2} = 2 - 6x$$

Then

$$\frac{d^2 f\left(\frac{2}{3}\right)}{dx^2} = 2 - 6 \times \frac{2}{3} = 2 - 4 = -2$$

Maximum Profit for the $1000.00 dollar Chip

$$\$667.00$$

Example

In our case

$$\frac{d^2 f(x)}{dx^2} = 2 - 6x$$

Then

$$\frac{d^2 f\left(\frac{2}{3}\right)}{dx^2} = 2 - 6 \times \frac{2}{3} = 2 - 4 = -2$$

Maximum Profit for the $1000.00 dollar Chip

$$\$667.00$$

What if $\frac{d^2 f(x_i)}{dx^2} = 0$?

Question

If the second derivative is 0 in a critical point x_i, then x_i may or may not be a minimum or a maximum of f. **WHY?**

We have for $x^3 - 3x^2 + x - 2$

With derivative

$$\frac{d^2 f(x)}{dx^2} = 6x - 6$$

What if $\frac{d^2 f(x_i)}{dx^2} = 0$?

Question

If the second derivative is 0 in a critical point x_i, then x_i may or may not be a minimum or a maximum of f. **WHY?**

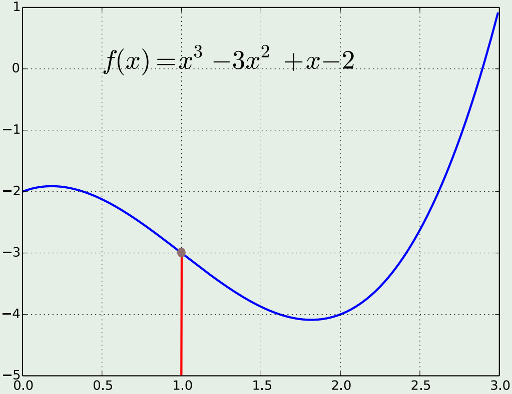
We have for $x^3 - 3x^2 + x - 2$

With derivative

$$\frac{d^2 f(x)}{dx^2} = 6x - 6$$

Actually a point where $\frac{d^2 f(x_i)}{dx^2} = 0$

We have a change in the "curvature $\approx \frac{d^2 f(x)}{dx^2}$"



Properties of Differentiating

Generalization

To move to higher dimensional functions, we will require to take partial derivatives!!!

Solving

A system of equations!!!

Remark

For a bounded D the only possible points of maximum/minimum are critical or boundary ones, so, in principle, we can find the global extremum.

Properties of Differentiating

Generalization

To move to higher dimensional functions, we will require to take partial derivatives!!!

Solving

A system of equations!!!

Remark

For a bounded D the only possible points of maximum/minimum are critical or boundary ones, so, in principle, we can find the global extremum.

Properties of Differentiating

Generalization

To move to higher dimensional functions, we will require to take partial derivatives!!!

Solving

A system of equations!!!

Remark

For a bounded D the only possible points of maximum/minimum are critical or boundary ones, so, in principle, we can find the global extremum.

Problems

A lot of them

- Potential problems include transcendent equations, not solvable analytically.
- High cost of finding derivatives, especially in high dimensions (e.g. for neural networks)

Thus

Partial Solution of the problems comes from a numerical technique called the gradient descent

Problems

A lot of them

- Potential problems include transcendent equations, not solvable analytically.
- High cost of finding derivatives, especially in high dimensions (e.g. for neural networks)

Thus

Partial Solution of the problems comes from a numerical technique called the gradient descent

Outline

Numerical Method: Gradient Descent

Imagine the following

- f is a smooth objective function.
- Now you have a x_0 state and you need to find the next one.

Numerical Method: Gradient Descent

Imagine the following

- f is a smooth objective function.
- Now you have a x_0 state and you need to find the next one.

Something Notable

We want to find x in the neighborhood D of x_0 such that

$$f(x) < f(x_0)$$

Numerical Method: Gradient Descent

Imagine the following

- f is a smooth objective function.
- Now you have a $\boldsymbol{x}_0$ state and you need to find the next one.

Something Notable

We want to find $\boldsymbol{x}$ in the neighborhood D of $\boldsymbol{x}_0$ such that

$$f(\boldsymbol{x}) < f(\boldsymbol{x}_0)$$

Taylor's Expansion

Using the first order Taylor's expansion around point $\boldsymbol{x} \in \mathbb{R}^n$ for $f : \mathbb{R}^n \to \mathbb{R}$

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_0) + \nabla f(\boldsymbol{x}_0)^T \cdot (\boldsymbol{x} - \boldsymbol{x}_0) + O\left(\|\boldsymbol{x} - \boldsymbol{x}_0\|^2\right)$$

Note:

- Actually the Taylor's expansions are polynomial approximation to the function!!!
- $\nabla f(\boldsymbol{x}) = \left[\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \frac{\partial f(\boldsymbol{x})}{\partial x_2}, \ldots, \frac{\partial f(\boldsymbol{x})}{\partial x_n}\right]^T$ with $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$

Taylor's Expansion

Using the first order Taylor's expansion around point $\boldsymbol{x} \in \mathbb{R}^n$ for $f : \mathbb{R}^n \rightarrow \mathbb{R}$

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_0) + \nabla f(\boldsymbol{x}_0)^T \cdot (\boldsymbol{x} - \boldsymbol{x}_0) + O\left(\|\boldsymbol{x} - \boldsymbol{x}_0\|^2\right)$$

Note:

- Actually the Taylor's expansions are polynomial approximation to the function!!!
- $\nabla f(\boldsymbol{x}) = \left[\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \frac{\partial f(\boldsymbol{x})}{\partial x_2}, \ldots, \frac{\partial f(\boldsymbol{x})}{\partial x_n}\right]^T$ with $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$

If we can find a neighborhood D small enough

We can discard the terms of the second and higher orders because the linear approximation is enough!!!

Taylor's Expansion

Using the first order Taylor's expansion around point $\boldsymbol{x} \in \mathbb{R}^n$ for $f : \mathbb{R}^n \to \mathbb{R}$

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_0) + \nabla f(\boldsymbol{x}_0)^T \cdot (\boldsymbol{x} - \boldsymbol{x}_0) + O\left(\|\boldsymbol{x} - \boldsymbol{x}_0\|^2\right)$$

Note:

- Actually the Taylor's expansions are polynomial approximation to the function!!!
- $\nabla f(\boldsymbol{x}) = \left[\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \frac{\partial f(\boldsymbol{x})}{\partial x_2}, ..., \frac{\partial f(\boldsymbol{x})}{\partial x_n}\right]^T$ with $\boldsymbol{x} = (x_1, x_2, ..., x_n)^T$

If we can find a neighborhood D small enough

We can discard the terms of the second and higher orders because the linear approximation is enough!!!

Taylor's Expansion

Using the first order Taylor's expansion around point $\boldsymbol{x} \in \mathbb{R}^n$ for $f : \mathbb{R}^n \to \mathbb{R}$

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_0) + \nabla f(\boldsymbol{x}_0)^T \cdot (\boldsymbol{x} - \boldsymbol{x}_0) + O\left(\|\boldsymbol{x} - \boldsymbol{x}_0\|^2\right)$$

Note:

- Actually the Taylor's expansions are polynomial approximation to the function!!!
- $\nabla f(\boldsymbol{x}) = \left[\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \frac{\partial f(\boldsymbol{x})}{\partial x_2}, ..., \frac{\partial f(\boldsymbol{x})}{\partial x_n}\right]^T$ with $\boldsymbol{x} = (x_1, x_2, ..., x_n)^T$

If we can find a neighborhood D small enough

We can discard the terms of the second and higher orders because the linear approximation is enough!!!

How do we do this?

Simple

$$\boldsymbol{x} = \boldsymbol{x}_0 + h\boldsymbol{u}$$

where x_0 and u are vectors and h is a constant.

How do we do this?

Simple

$$\boldsymbol{x} = \boldsymbol{x}_0 + h\boldsymbol{u}$$

where $\boldsymbol{x}_0$ and $\boldsymbol{u}$ are vectors and h is a constant.

Then we get

$$f(\boldsymbol{x}_0 + h\boldsymbol{u}) - f(\boldsymbol{x}_0) = h\nabla f(\boldsymbol{x}_0)^T \cdot \boldsymbol{u} + h^2 O(1)$$

How do we do this?

Simple

$$\boldsymbol{x} = \boldsymbol{x}_0 + h\boldsymbol{u}$$

where $\boldsymbol{x}_0$ and $\boldsymbol{u}$ are vectors and h is a constant.

Then we get

$$f(\boldsymbol{x}_0 + h\boldsymbol{u}) - f(\boldsymbol{x}_0) = h\nabla f(\boldsymbol{x}_0)^T \cdot \boldsymbol{u} + h^2 O(1)$$

We make h^2 term insignificant by shrinking h

Thus, if we want to decrease $f(\boldsymbol{x}_0 + h\boldsymbol{u}) - f(\boldsymbol{x}_0) < 0$ the fastest, enforcing $f(\boldsymbol{x}_0 + h\boldsymbol{u}) < f(\boldsymbol{x}_0)$:

$$f(\boldsymbol{x}_0 + h\boldsymbol{u}) - f(\boldsymbol{x}_0) \approx h\nabla f(\boldsymbol{x}_0)^T \cdot \boldsymbol{u}$$

How do we do this?

Simple

$$\boldsymbol{x} = \boldsymbol{x}_0 + h\boldsymbol{u}$$

where $\boldsymbol{x}_0$ and $\boldsymbol{u}$ are vectors and h is a constant.

Then we get

$$f(\boldsymbol{x}_0 + h\boldsymbol{u}) - f(\boldsymbol{x}_0) = h\nabla f(\boldsymbol{x}_0)^T \cdot \boldsymbol{u} + h^2 O(1)$$

We make h^2 term insignificant by shrinking h

Thus, if we want to decrease $f(\boldsymbol{x}_0 + h\boldsymbol{u}) - f(\boldsymbol{x}_0) < 0$ the fastest, enforcing $f(\boldsymbol{x}_0 + h\boldsymbol{u}) < f(\boldsymbol{x}_0)$:

$$f(\boldsymbol{x}_0 + h\boldsymbol{u}) - f(\boldsymbol{x}_0) \approx h\nabla f(\boldsymbol{x}_0)^T \cdot \boldsymbol{u}$$

How do we do this?

Simple

$$\boldsymbol{x} = \boldsymbol{x}_0 + h\boldsymbol{u}$$

where $\boldsymbol{x}_0$ and $\boldsymbol{u}$ are vectors and h is a constant.

Then we get

$$f\left(\boldsymbol{x}_0 + h\boldsymbol{u}\right) - f\left(\boldsymbol{x}_0\right) = h\nabla f\left(\boldsymbol{x}_0\right)^T \cdot \boldsymbol{u} + h^2 O\left(1\right)$$

We make h^2 term insignificant by shrinking h

Thus, if we want to decrease $f\left(\boldsymbol{x}_0 + h\boldsymbol{u}\right) - f\left(\boldsymbol{x}_0\right) < 0$ the fastest, enforcing $f\left(\boldsymbol{x}_0 + h\boldsymbol{u}\right) < f\left(\boldsymbol{x}_0\right)$:

$$f\left(\boldsymbol{x}_0 + h\boldsymbol{u}\right) - f\left(\boldsymbol{x}_0\right) \approx h\nabla f\left(\boldsymbol{x}_0\right)^T \cdot \boldsymbol{u}$$

Then

We minimize

$$\nabla f\left(\boldsymbol{x}_0\right)^T \cdot \boldsymbol{u}$$

Thus, the unit vector that minimize

In order to obtain the largest difference

$$\boldsymbol{u} = -\frac{\nabla f\left(\boldsymbol{x}_0\right)}{\left\|\nabla f\left(\boldsymbol{x}_0\right)\right\|}$$

Then

$$\nabla f\left(\boldsymbol{x}_0\right)^T \times -\frac{\nabla f\left(\boldsymbol{x}_0\right)}{\left\|\nabla f\left(\boldsymbol{x}_0\right)\right\|} = -\left\|\nabla f\left(\boldsymbol{x}_0\right)\right\| < 0$$

Then

We minimize

$$\nabla f\left(\boldsymbol{x}_0\right)^T \cdot \boldsymbol{u}$$

Thus, the unit vector that minimize

In order to obtain the largest difference

$$\boldsymbol{u} = -\frac{\nabla f\left(\boldsymbol{x}_0\right)}{\left\|\nabla f\left(\boldsymbol{x}_0\right)\right\|}$$

Then

$$\nabla f\left(\boldsymbol{x}_0\right)^T \times -\frac{\nabla f\left(\boldsymbol{x}_0\right)}{\left\|\nabla f\left(\boldsymbol{x}_0\right)\right\|} = -\left\|\nabla f\left(\boldsymbol{x}_0\right)\right\| < 0$$

Then

We minimize

$$\nabla f(\boldsymbol{x}_0)^T \cdot \boldsymbol{u}$$

Thus, the unit vector that minimize

In order to obtain the largest difference

$$\boldsymbol{u} = -\frac{\nabla f(\boldsymbol{x}_0)}{\|\nabla f(\boldsymbol{x}_0)\|}$$

Then

$$\nabla f(\boldsymbol{x}_0)^T \times -\frac{\nabla f(\boldsymbol{x}_0)}{\|\nabla f(\boldsymbol{x}_0)\|} = -\|\nabla f(\boldsymbol{x}_0)\| < 0$$

Therefore

We have that

$$
\begin{aligned}
\boldsymbol{x} &= \boldsymbol{x}_0 + h\boldsymbol{u} \\
&= \boldsymbol{x}_0 - h\frac{\nabla f\left(\boldsymbol{x}_0\right)}{\left\|\nabla f\left(\boldsymbol{x}_0\right)\right\|} \\
&= \boldsymbol{x}_0 - h'\nabla f\left(\boldsymbol{x}_0\right)
\end{aligned}
$$

With $h' = \frac{h}{\|\nabla f(\boldsymbol{x}_0)\|}$

Therefore

We have that

$$
\begin{aligned}
\boldsymbol{x} &= \boldsymbol{x}_0 + h\boldsymbol{u}\\
&= \boldsymbol{x}_0 - h\frac{\nabla f\left(\boldsymbol{x}_0\right)}{\left\|\nabla f\left(\boldsymbol{x}_0\right)\right\|}\\
&= \boldsymbol{x}_0 - h'\nabla f\left(\boldsymbol{x}_0\right)
\end{aligned}
$$

With $h' = \frac{h}{\|\nabla f(\boldsymbol{x}_0)\|}$

Therefore

We have that

$$\begin{aligned} \boldsymbol{x} &= \boldsymbol{x}_0 + h\boldsymbol{u} \\ &= \boldsymbol{x}_0 - h\frac{\nabla f\left(\boldsymbol{x}_0\right)}{\left\|\nabla f\left(\boldsymbol{x}_0\right)\right\|} \\ &= \boldsymbol{x}_0 - h'\nabla f\left(\boldsymbol{x}_0\right) \end{aligned}$$

With $h' = \frac{h}{\|\nabla f(\boldsymbol{x}_0)\|}$

Outline

Gradient Descent

In the method of Gradient descent, we have a cost function $J(\boldsymbol{w})$ where

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \eta \nabla J(\boldsymbol{w}(\boldsymbol{n}))$$

How, we prove that $J(\boldsymbol{w}(n+1)) < J(\boldsymbol{w}(n))$?

We use the first-order Taylor series expansion around $\boldsymbol{w}(n)$

$$J(\boldsymbol{w}(n+1)) \approx J(\boldsymbol{w}(n)) + \nabla J^T(\boldsymbol{w}(n)) \Delta \boldsymbol{w}(n) \quad (2)$$

Remark: This is quite true when the step size is quite small!!! In addition, $\Delta \boldsymbol{w}(n) = \boldsymbol{w}(n+1) - \boldsymbol{w}(n)$

Gradient Descent

In the method of Gradient descent, we have a cost function $J(\boldsymbol{w})$ where

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \eta \nabla J(\boldsymbol{w}(\boldsymbol{n}))$$

How, we prove that $J(\boldsymbol{w}(n+1)) < J(\boldsymbol{w}(n))$?

We use the first-order Taylor series expansion around $\boldsymbol{w}(n)$

$$J(\boldsymbol{w}(n+1)) \approx J(\boldsymbol{w}(n)) + \nabla J^T(\boldsymbol{w}(\boldsymbol{n})) \Delta\boldsymbol{w}(n) \quad (2)$$

Remark: This is quite true when the step size is quite small!!! In addition, $\Delta\boldsymbol{w}(n) = \boldsymbol{w}(n+1) - \boldsymbol{w}(n)$

Why? Look at the case in $\mathbb{R}$

The equation of the tangent line to the curve $y = J\left(w\left(n\right)\right)$

$$L\left(w\left(n\right)\right) = J'\left(w\left(n\right)\right)\left[w\left(n+1\right) - w\left(n\right)\right] + J\left(w\left(n\right)\right) \tag{3}$$

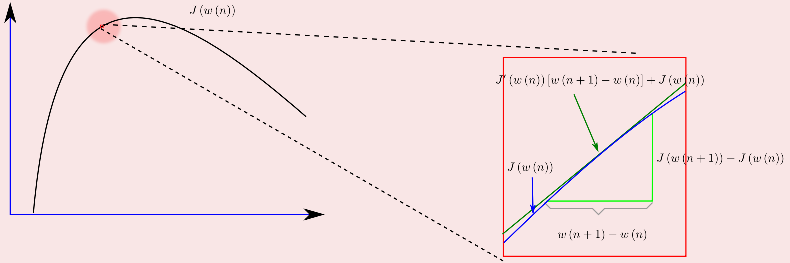
Example

Why? Look at the case in $\mathbb{R}$

The equation of the tangent line to the curve $y = J(w(n))$

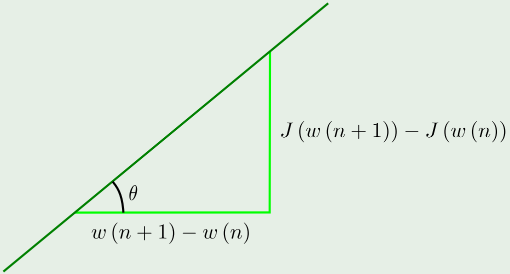
$$L(w(n)) = J'(w(n))[w(n+1) - w(n)] + J(w(n)) \tag{3}$$

Example



Thus, we have that in $\mathbb{R}$

Remember Something quite Classic



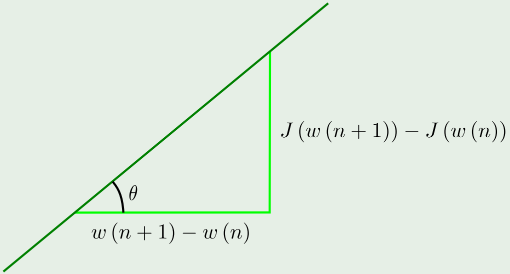
$$\tan\theta = \frac{J(w(n+1)) - J(w(n))}{w(n+1) - w(n)}$$

$$\tan\theta\,(w(n+1) - w(n)) = J(w(n+1)) - J(w(n))$$

$$J'(w(n))\,(w(n+1) - w(n)) = J(w(n+1)) - J(w(n))$$

Thus, we have that in $\mathbb{R}$

Remember Something quite Classic



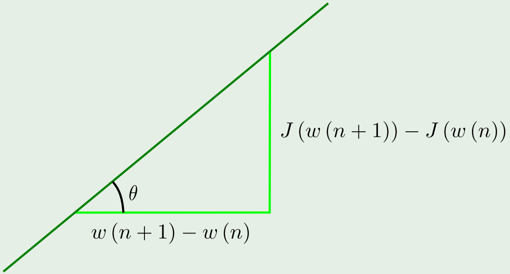
$$\tan\theta = \frac{J(w(n+1)) - J(w(n))}{w(n+1) - w(n)}$$

$$\tan\theta\,(w(n+1) - w(n)) = J(w(n+1)) - J(w(n))$$

$$J'(w(n))\,(w(n+1) - w(n)) = J(w(n+1)) - J(w(n))$$

Thus, we have that in $\mathbb{R}$

Remember Something quite Classic



$$\tan\theta = \frac{J(w(n+1)) - J(w(n))}{w(n+1) - w(n)}$$
$$\tan\theta\,(w(n+1) - w(n)) = J(w(n+1)) - J(w(n))$$
$$J'(w(n))\,(w(n+1) - w(n)) = J(w(n+1)) - J(w(n))$$

Thus, we have that

Using the First Taylor expansion

$$J\left(w\left(n\right)\right) \approx J\left(w\left(n\right)\right) + J'\left(w\left(n\right)\right)\left[w\left(n+1\right) - w\left(n\right)\right] \quad (4)$$

Now, for Many Variables

An hyperplane in $\mathbb{R}^n$ is a set of the form

$$H = \left\{ \boldsymbol{x} | \boldsymbol{a}^T \boldsymbol{x} = b \right\} \tag{5}$$

Given $\boldsymbol{x} \in H$ and $\boldsymbol{x}_0 \in H$

$$b = \boldsymbol{a}^T \boldsymbol{x} = \boldsymbol{a}^T \boldsymbol{x}_0$$

Thus, we have that

$$H = \left\{ \boldsymbol{x} | \boldsymbol{a}^T (\boldsymbol{x} - \boldsymbol{x}_0) = 0 \right\}$$

Now, for Many Variables

An hyperplane in $\mathbb{R}^n$ is a set of the form

$$H = \left\{ \boldsymbol{x} | \boldsymbol{a}^T \boldsymbol{x} = b \right\} \tag{5}$$

Given $\boldsymbol{x} \in H$ and $\boldsymbol{x}_0 \in H$

$$b = \boldsymbol{a}^T \boldsymbol{x} = \boldsymbol{a}^T \boldsymbol{x}_0$$

Thus, we have that

$$H = \left\{ \boldsymbol{x} | \boldsymbol{a}^T (\boldsymbol{x} - \boldsymbol{x}_0) = 0 \right\}$$

Now, for Many Variables

An hyperplane in $\mathbb{R}^n$ is a set of the form

$$H = \left\{ \boldsymbol{x} | \boldsymbol{a}^T \boldsymbol{x} = b \right\} \tag{5}$$

Given $\boldsymbol{x} \in H$ and $\boldsymbol{x}_0 \in H$

$$b = \boldsymbol{a}^T \boldsymbol{x} = \boldsymbol{a}^T \boldsymbol{x}_0$$

Thus, we have that

$$H = \left\{ \boldsymbol{x} | \boldsymbol{a}^T \left(\boldsymbol{x} - \boldsymbol{x}_0 \right) = 0 \right\}$$

Thus, we have the following definition

Definition (Differentiability)

Assume that J is defined in a disk D containing $\boldsymbol{w}(n)$. We say that J is differentiable at $\boldsymbol{w}(n)$ if:

1. $\frac{\partial J(\boldsymbol{w}(n))}{\partial w_i}$ exist for all $i = 1, ..., n$.
2. J is locally linear at $\boldsymbol{w}(n)$.

Thus, we have the following definition

Definition (Differentiability)

Assume that J is defined in a disk D containing $\boldsymbol{w}(n)$. We say that J is differentiable at $\boldsymbol{w}(n)$ if:

1. $\frac{\partial J(\boldsymbol{w}(n))}{\partial w_i}$ exist for all $i = 1, ..., n$.
2. J is locally linear at $\boldsymbol{w}(n)$.

Thus, we have the following definition

Definition (Differentiability)

Assume that J is defined in a disk D containing $\boldsymbol{w}(n)$. We say that J is differentiable at $\boldsymbol{w}(n)$ if:

1. $\frac{\partial J(\boldsymbol{w}(n))}{\partial w_i}$ exist for all $i = 1, ..., n$.
2. J is locally linear at $\boldsymbol{w}(n)$.

Thus, given $J\left(\boldsymbol{w}\left(n\right)\right)$

We know that we have the following operator

$$\nabla = \left(\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, ..., \frac{\partial}{\partial w_m}\right) \tag{6}$$

Thus, given $J\left(\boldsymbol{w}\left(n\right)\right)$

We know that we have the following operator

$$\nabla=\left(\frac{\partial}{\partial w_1},\frac{\partial}{\partial w_2},...,\frac{\partial}{\partial w_m}\right) \tag{6}$$

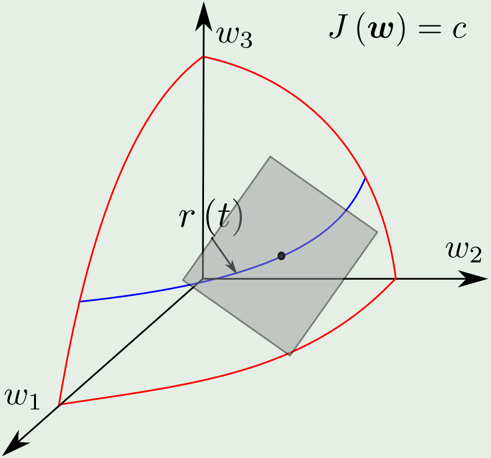
Thus, we have

$$\begin{aligned}\nabla J\left(\boldsymbol{w}\left(n\right)\right)&=\left(\frac{\partial J\left(\boldsymbol{w}\left(n\right)\right)}{\partial w_1},\frac{\partial J\left(\boldsymbol{w}\left(n\right)\right)}{\partial w_2},...,\frac{\partial J\left(\boldsymbol{w}\left(n\right)\right)}{\partial w_m}\right)\\&=\sum_{i=1}^{m}\hat{w}_i\frac{\partial J\left(\boldsymbol{w}\left(n\right)\right)}{\partial w_i}\end{aligned}$$

Where: $\hat{w}_i^T=(1,0,...,0)\in\mathbb{R}$

Now

Given a curve function $r(t)$ that lies on the level set $J(\boldsymbol{w}(n)) = c$ (When is in $\mathbb{R}^3$)



Level Set

Definition

$$\{(w_1, w_2, ..., w_m) \in \mathbb{R}^m | J\,(w_1, w_2, ..., w_m) = c\} \quad (7)$$

Remark: In a normal Calculus course we will use x and f instead of $\boldsymbol{w}$ and J.

Where

Any curve has the following parametrization

$$r : [a, b] \to \mathbb{R}^m$$
$$r(t) = (w_1(t), ..., w_m(t))$$

With $r(n+1) = (w_1(n+1), ..., w_m(n+1))$

We can write the parametrized version of it

$$z(t) = J(w_1(t), w_2(t), ..., w_m(t)) = c \tag{8}$$

Differentiating with respect to t and using the chain rule for multiple variables

$$\frac{dz(t)}{dt} = \sum_{i=1}^{m} \frac{\partial J(\boldsymbol{w}(t))}{\partial w_i} \cdot \frac{dw_i(t)}{dt} = 0 \tag{9}$$

Where

Any curve has the following parametrization

$$r : [a, b] \to \mathbb{R}^m$$
$$r(t) = (w_1(t), ..., w_m(t))$$

With $r(n+1) = (w_1(n+1), ..., w_m(n+1))$

We can write the parametrized version of it

$$z(t) = J(w_1(t), w_2(t), ..., w_m(t)) = c \tag{8}$$

Differentiating with respect to t and using the chain rule for multiple variables

$$\frac{dz(t)}{dt} = \sum_{i=1}^{m} \frac{\partial J(\boldsymbol{w}(t))}{\partial w_i} \cdot \frac{dw_i(t)}{dt} = 0 \tag{9}$$

Where

Any curve has the following parametrization

$$r:[a,b]\to\mathbb{R}^m$$
$$r(t)=(w_1(t),...,w_m(t))$$

With $r(n+1)=(w_1(n+1),...,w_m(n+1))$

We can write the parametrized version of it

$$z(t)=J(w_1(t),w_2(t),...,w_m(t))=c \tag{8}$$

Differentiating with respect to t and using the chain rule for multiple variables

$$\frac{dz(t)}{dt}=\sum_{i=1}^{m}\frac{\partial J(\boldsymbol{w}(t))}{\partial w_i}\cdot\frac{dw_i(t)}{dt}=0 \tag{9}$$

Note

First

Given $y = f(\boldsymbol{u}) = (f_1(\boldsymbol{u}), ..., f_l(\boldsymbol{u}))$ and $\boldsymbol{u} = g(\boldsymbol{x}) = (g_1(\boldsymbol{x}), ..., g_m(\boldsymbol{x}))$.

We have then that

$$\frac{\partial(f_1, f_2, ..., f_l)}{\partial(x_1, x_2, ..., x_k)} = \frac{\partial(f_1, f_2, ..., f_l)}{\partial(g_1, g_2, ..., g_m)} \cdot \frac{\partial(g_1, g_2, ..., g_m)}{\partial(x_1, x_2, ..., x_k)} \tag{10}$$

Thus

$$\begin{aligned}\frac{\partial(f_1, f_2, ..., f_l)}{\partial x_i} &= \frac{\partial(f_1, f_2, ..., f_l)}{\partial(g_1, g_2, ..., g_m)} \cdot \frac{\partial(g_1, g_2, ..., g_m)}{\partial x_i} \\ &= \sum_{k=1}^{m} \frac{\partial(f_1, f_2, ..., f_l)}{\partial g_k} \frac{\partial g_k}{\partial x_i}\end{aligned}$$

Note

First

Given $y = f(\boldsymbol{u}) = (f_1(\boldsymbol{u}), ..., f_l(\boldsymbol{u}))$ and $\boldsymbol{u} = g(\boldsymbol{x}) = (g_1(\boldsymbol{x}), ..., g_m(\boldsymbol{x}))$.

We have then that

$$\frac{\partial(f_1, f_2, ..., f_l)}{\partial(x_1, x_2, ..., x_k)} = \frac{\partial(f_1, f_2, ..., f_l)}{\partial(g_1, g_2, ..., g_m)} \cdot \frac{\partial(g_1, g_2, ..., g_m)}{\partial(x_1, x_2, ..., x_k)} \tag{10}$$

Thus

$$\begin{aligned}\frac{\partial(f_1, f_2, ..., f_l)}{\partial x_i} &= \frac{\partial(f_1, f_2, ..., f_l)}{\partial(g_1, g_2, ..., g_m)} \cdot \frac{\partial(g_1, g_2, ..., g_m)}{\partial x_i}\\ &= \sum_{k=1}^{m} \frac{\partial(f_1, f_2, ..., f_l)}{\partial g_k}\frac{\partial g_k}{\partial x_i}\end{aligned}$$

Note

First

Given $y = f(\boldsymbol{u}) = (f_1(\boldsymbol{u}), ..., f_l(\boldsymbol{u}))$ and $\boldsymbol{u} = g(\boldsymbol{x}) = (g_1(\boldsymbol{x}), ..., g_m(\boldsymbol{x}))$.

We have then that

$$\frac{\partial(f_1, f_2, ..., f_l)}{\partial(x_1, x_2, ..., x_k)} = \frac{\partial(f_1, f_2, ..., f_l)}{\partial(g_1, g_2, ..., g_m)} \cdot \frac{\partial(g_1, g_2, ..., g_m)}{\partial(x_1, x_2, ..., x_k)} \tag{10}$$

Thus

$$\begin{aligned} \frac{\partial(f_1, f_2, ..., f_l)}{\partial x_i} &= \frac{\partial(f_1, f_2, ..., f_l)}{\partial(g_1, g_2, ..., g_m)} \cdot \frac{\partial(g_1, g_2, ..., g_m)}{\partial x_i} \\ &= \sum_{k=1}^{m} \frac{\partial(f_1, f_2, ..., f_l)}{\partial g_k} \frac{\partial g_k}{\partial x_i} \end{aligned}$$

Thus

Evaluating at $t = n$

$$\sum_{i=1}^{m} \frac{\partial J\left(\boldsymbol{w}\left(n\right)\right)}{\partial w_i} \cdot \frac{dw_i(n)}{dt} = 0$$

We have that

$$\nabla J\left(\boldsymbol{w}\left(n\right)\right) \cdot \boldsymbol{r}'\left(n\right) = 0 \qquad (11)$$

This proves that for every level set the gradient is perpendicular to the tangent to any curve that lies on the level set

In particular to the point $\boldsymbol{w}\left(n\right)$.

Thus

Evaluating at $t = n$

$$\sum_{i=1}^{m} \frac{\partial J\left(\boldsymbol{w}\left(n\right)\right)}{\partial w_i} \cdot \frac{dw_i(n)}{dt} = 0$$

We have that

$$\nabla J\left(\boldsymbol{w}\left(n\right)\right) \cdot r'\left(n\right) = 0 \tag{11}$$

This proves that for every level set the gradient is perpendicular to the tangent to any curve that lies on the level set

In particular to the point $w\left(n\right)$.

Thus

Evaluating at $t = n$

$$\sum_{i=1}^{m} \frac{\partial J\left(\boldsymbol{w}\left(n\right)\right)}{\partial w_i} \cdot \frac{dw_i(n)}{dt} = 0$$

We have that

$$\nabla J\left(\boldsymbol{w}\left(n\right)\right) \cdot r'\left(n\right) = 0 \qquad (11)$$

This proves that for every level set the gradient is perpendicular to the tangent to any curve that lies on the level set

In particular to the point $\boldsymbol{w}\left(n\right)$.

Now the tangent plane to the surface can be described generally

Thus

$$L\left(\boldsymbol{w}\left(n+1\right)\right)=J\left(\boldsymbol{w}\left(n\right)\right)+\nabla J^{T}\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\left[\boldsymbol{w}\left(n+1\right)-\boldsymbol{w}\left(n\right)\right] \tag{12}$$

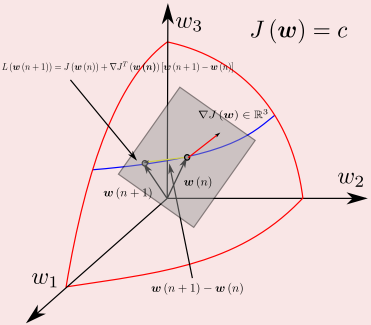
This looks like

Now the tangent plane to the surface can be described generally

Thus

$$L\left(\boldsymbol{w}\left(n+1\right)\right) = J\left(\boldsymbol{w}\left(n\right)\right) + \nabla J^{T}\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\left[\boldsymbol{w}\left(n+1\right) - \boldsymbol{w}\left(n\right)\right] \quad (12)$$

This looks like



Proving the fact about the Gradient Descent

We want the following

$$J\left(\boldsymbol{w}\left(n+1\right)\right) < J\left(\boldsymbol{w}\left(n\right)\right)$$

Using the first-order Taylor approximation

$$J\left(\boldsymbol{w}\left(n+1\right)\right) - J\left(\boldsymbol{w}\left(n\right)\right) \approx \nabla J^{T}\left(\boldsymbol{w}\left(n\right)\right)\Delta\boldsymbol{w}\left(n\right)$$

So, we ask the following

$$\Delta\boldsymbol{w}\left(n\right) \approx -\eta\nabla J\left(\boldsymbol{w}\left(n\right)\right) \text{ with } \eta > 0$$

Proving the fact about the Gradient Descent

We want the following

$$J\left(\boldsymbol{w}\left(n+1\right)\right) < J\left(\boldsymbol{w}\left(n\right)\right)$$

Using the first-order Taylor approximation

$$J\left(\boldsymbol{w}\left(n+1\right)\right) - J\left(\boldsymbol{w}\left(n\right)\right) \approx \nabla J^{T}\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\Delta\boldsymbol{w}\left(n\right)$$

So, we ask the following

$$\Delta\boldsymbol{w}\left(n\right) \approx -\eta\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right) \text{ with } \eta > 0$$

Proving the fact about the Gradient Descent

We want the following

$$J\left(\boldsymbol{w}\left(n+1\right)\right) < J\left(\boldsymbol{w}\left(n\right)\right)$$

Using the first-order Taylor approximation

$$J\left(\boldsymbol{w}\left(n+1\right)\right) - J\left(\boldsymbol{w}\left(n\right)\right) \approx \nabla J^{T}\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\Delta\boldsymbol{w}\left(n\right)$$

So, we ask the following

$$\Delta\boldsymbol{w}\left(n\right) \approx -\eta\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right) \text{ with } \eta > 0$$

Then

We have that

$$J\left(\boldsymbol{w}\left(n+1\right)\right)-J\left(\boldsymbol{w}\left(n\right)\right)\approx-\eta\nabla J^{T}\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)=-\eta\left\Vert \nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\right\Vert ^{2}$$

Thus

$$J\left(\boldsymbol{w}\left(n+1\right)\right)-J\left(\boldsymbol{w}\left(n\right)\right)<0$$

Or

$$J\left(\boldsymbol{w}\left(n+1\right)\right)<J\left(\boldsymbol{w}\left(n\right)\right)$$

Then

We have that

$$J\left(\boldsymbol{w}\left(n+1\right)\right)-J\left(\boldsymbol{w}\left(n\right)\right)\approx-\eta\nabla J^{T}\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)=-\eta\left\Vert \nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\right\Vert ^{2}$$

Thus

$$J\left(\boldsymbol{w}\left(n+1\right)\right)-J\left(\boldsymbol{w}\left(n\right)\right)<0$$

Or

$$J\left(\boldsymbol{w}\left(n+1\right)\right)<J\left(\boldsymbol{w}\left(n\right)\right)$$

Then

We have that

$$J\left(\boldsymbol{w}\left(n+1\right)\right)-J\left(\boldsymbol{w}\left(n\right)\right)\approx-\eta\nabla J^{T}\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)=-\eta\left\Vert \nabla J\left(\boldsymbol{w}\left(\boldsymbol{n}\right)\right)\right\Vert ^{2}$$

Thus

$$J\left(\boldsymbol{w}\left(n+1\right)\right)-J\left(\boldsymbol{w}\left(n\right)\right)<0$$

Or

$$J\left(\boldsymbol{w}\left(n+1\right)\right)<J\left(\boldsymbol{w}\left(n\right)\right)$$

Outline

Algorithm of Gradient Descent

Initialization

1. Guess an init point $\boldsymbol{x}_0$
2. Use a N_{max} iteration count
3. A gradient norm tolerance ϵ_g to know if we have arrived to a critical point.
4. A step tolerance ϵ_x to know if we have done significant progress
5. α_t is known as the step size.
   1. It is chosen to maintain a balance between convergence speed and avoiding divergence.

Algorithm of Gradient Descent

Initialization

1. Guess an init point $\boldsymbol{x}_0$
2. Use a N_{max} iteration count
3. A gradient norm tolerance ϵ_g to know if we have arrived to a critical point.
4. A step tolerance ϵ_x to know if we have done significant progress
5. α_t is known as the step size.
   1. It is chosen to maintain a balance between convergence speed and avoiding divergence.

Algorithm of Gradient Descent

Initialization

1. Guess an init point $\boldsymbol{x}_0$
2. Use a N_{max} iteration count
3. A gradient norm tolerance ϵ_g to know if we have arrived to a critical point.
4. A step tolerance ϵ_x to know if we have done significant progress
5. α_t is known as the step size.
   1. It is chosen to maintain a balance between convergence speed and avoiding divergence.

Algorithm of Gradient Descent

Initialization

1. Guess an init point $\boldsymbol{x}_0$
2. Use a N_{max} iteration count
3. A gradient norm tolerance ϵ_g to know if we have arrived to a critical point.
4. A step tolerance ϵ_x to know if we have done significant progress
5. α_t is known as the step size.
    1. It is chosen to maintain a balance between convergence speed and avoiding divergence.

Algorithm of Gradient Descent

Initialization

1. Guess an init point $\boldsymbol{x}_0$
2. Use a N_{max} iteration count
3. A gradient norm tolerance ϵ_g to know if we have arrived to a critical point.
4. A step tolerance ϵ_x to know if we have done significant progress
5. α_t is known as the step size.
   1. It is chosen to maintain a balance between convergence speed and avoiding divergence.

Algorithm of Gradient Descent

Initialization

1. Guess an init point $\boldsymbol{x}_0$
2. Use a N_{max} iteration count
3. A gradient norm tolerance ϵ_g to know if we have arrived to a critical point.
4. A step tolerance ϵ_x to know if we have done significant progress
5. α_t is known as the step size.
    1. It is chosen to maintain a balance between convergence speed and avoiding divergence.

Finally

Gradient_Descent($x_0, N_{max}, \epsilon_g, \epsilon_t, \alpha_t$)

1. **for** $t = 0, 1, 2, ..., N_{max}$
2. $\quad x_{t+1} = x_t - \alpha_t \nabla f(x_t)$

Finally

Gradient_Descent$(\boldsymbol{x}_0, N_{max}, \epsilon_g, \epsilon_t, \alpha_t)$

1. **for** $t = 0, 1, 2, ..., N_{max}$
2. $\quad \boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \alpha_t \nabla f(\boldsymbol{x}_t)$
3. $\quad$ **if** $\|\nabla f(\boldsymbol{x}_{t+1})\| < \epsilon_g$
4. $\quad\quad$ **return "Converged on critical point"**
5. $\quad$ **if** $\|\boldsymbol{x}_t - \boldsymbol{x}_{t+1}\| < \epsilon_t$
6. $\quad\quad$ **return "Converged on an x value"**
7. $\quad$ **if** $f(\boldsymbol{x}_{t+1}) > f(\boldsymbol{x}_t)$
8. $\quad\quad$ **return "Diverging"**
9. **return "Maximum number of iterations reached"**

Finally

Gradient_Descent$(\boldsymbol{x}_0, N_{max}, \epsilon_g, \epsilon_t, \alpha_t)$

```
for t = 0, 1, 2, ..., N_max
    x_{t+1} = x_t − α_t ∇f(x_t)
    if ‖∇f(x_{t+1})‖ < ε_g
        return "Converged on critical point"
    if ‖x_t − x_{t+1}‖ < ε_t
        return "Converged on an x value"
    if f(x_{t+1}) > f(x_t)
        return "Diverging"
return "Maximum number of iterations reached"
```

Finally

Gradient_Descent$(\boldsymbol{x}_0, N_{max}, \epsilon_g, \epsilon_t, \alpha_t)$

for $t = 0, 1, 2, ..., N_{max}$

  $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \alpha_t \nabla f(\boldsymbol{x}_t)$

  if $\|\nabla f(\boldsymbol{x}_{t+1})\| < \epsilon_g$

    return "Converged on critical point"

  if $\|\boldsymbol{x}_t - \boldsymbol{x}_{t+1}\| < \epsilon_t$

    return "Converged on an x **value"**

  if $f(\boldsymbol{x}_{t+1}) > f(\boldsymbol{x}_t)$

    return "Diverging"

return "Maximum number of iterations reached"

Finally

Gradient_Descent$(\boldsymbol{x}_0, N_{max}, \epsilon_g, \epsilon_t, \alpha_t)$

for $t = 0, 1, 2, \ldots, N_{max}$

  $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \alpha_t \nabla f(\boldsymbol{x}_t)$

  if $\|\nabla f(\boldsymbol{x}_{t+1})\| < \epsilon_g$

    return "Converged on critical point"

  if $\|\boldsymbol{x}_t - \boldsymbol{x}_{t+1}\| < \epsilon_t$

    return "Converged on an x **value"**

  if $f(\boldsymbol{x}_{t+1}) > f(\boldsymbol{x}_t)$

    return "Diverging"

return "Maximum number of iterations reached"

IMPORTANT

I forgot to mention something

$\nabla f(x)$ give us the direction of the fastest change at x.

IMPORTANT

I forgot to mention something

$\nabla f(x)$ give us the direction of the fastest change at x.

Observations

- Gradient descent can only work if at least we can differentiate the cost function
- Gradient descent gets bottled up in local minima or maxima

IMPORTANT

I forgot to mention something

$\nabla f(x)$ give us the direction of the fastest change at x.

Observations

- Gradient descent can only work if at least we can differentiate the cost function
- Gradient descent gets bottled up in local minima or maxima