Machine Learning

Greedy Local Search

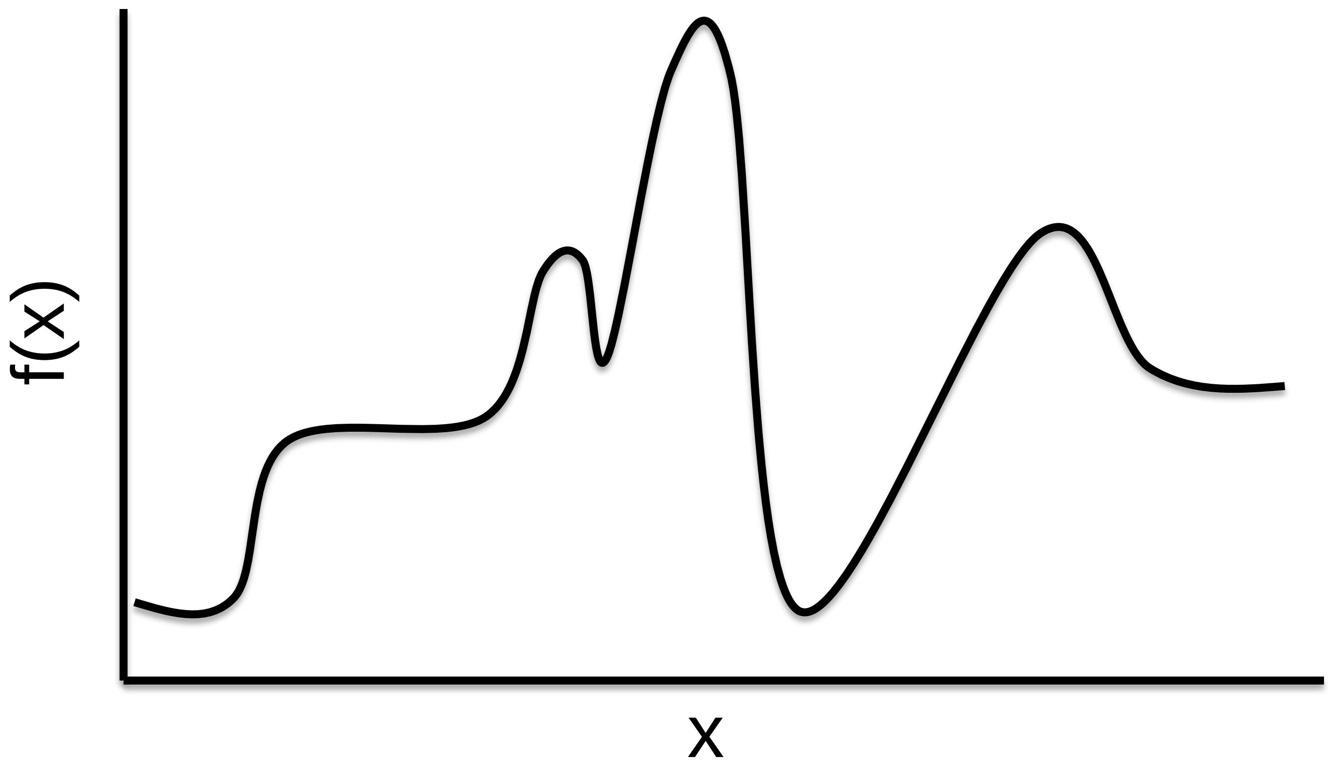
With slides from Bryan Pardo, Stuart Russell

ML in a Nutshell

- Every machine learning algorithm has three components:
  - Representation
    - E.g., Decision trees, instances
  - Evaluation
    - E.g., accuracy on test set
  - **Optimization**
    - How do you **find** the best hypothesis?

Hill-climbing (greedy local search)

$$\text{find} \quad x_{\max} = \underset{x \in X}{\arg\max}(f(x))$$



Greedy local search needs

- A “successor” function
  - Says what states I can reach from the current one.
  - Often implicitly a distance measure.
- An objective (error) function
  - Tells me how good a state is
- Enough memory to hold
  - The best state found so far
  - The current state
  - The state it’s considering moving to

Hill-climbing search

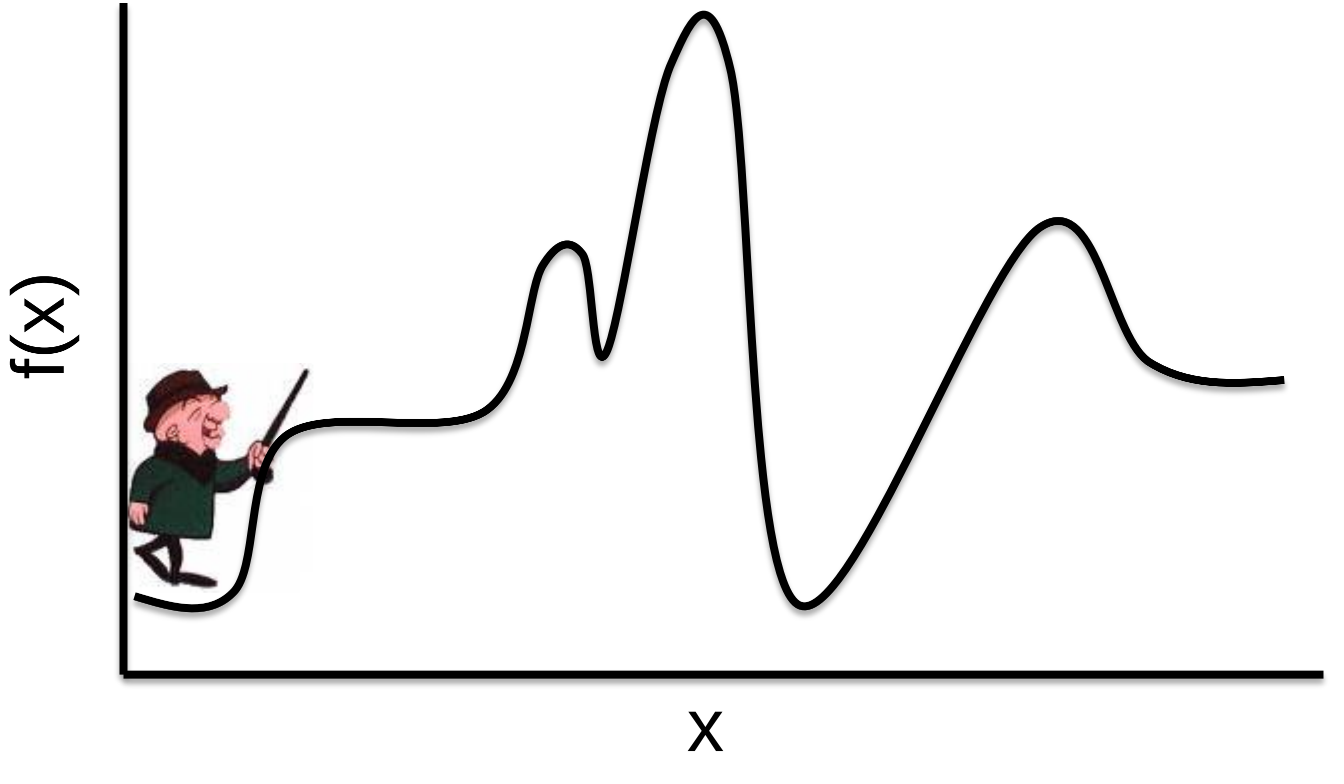
- "Like climbing Everest in thick fog with amnesia"

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

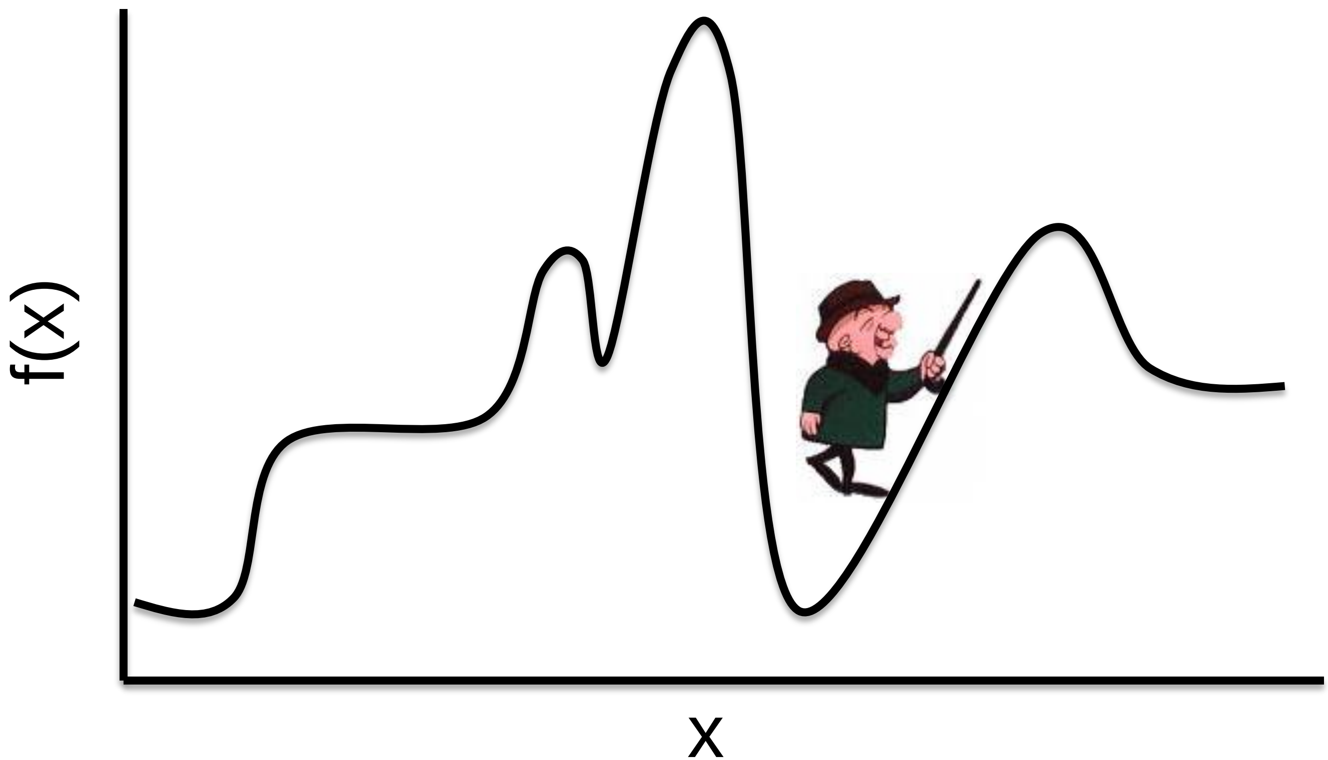
Hill-climbing (greedy local search)

"Like climbing Everest in thick fog with amnesia‟



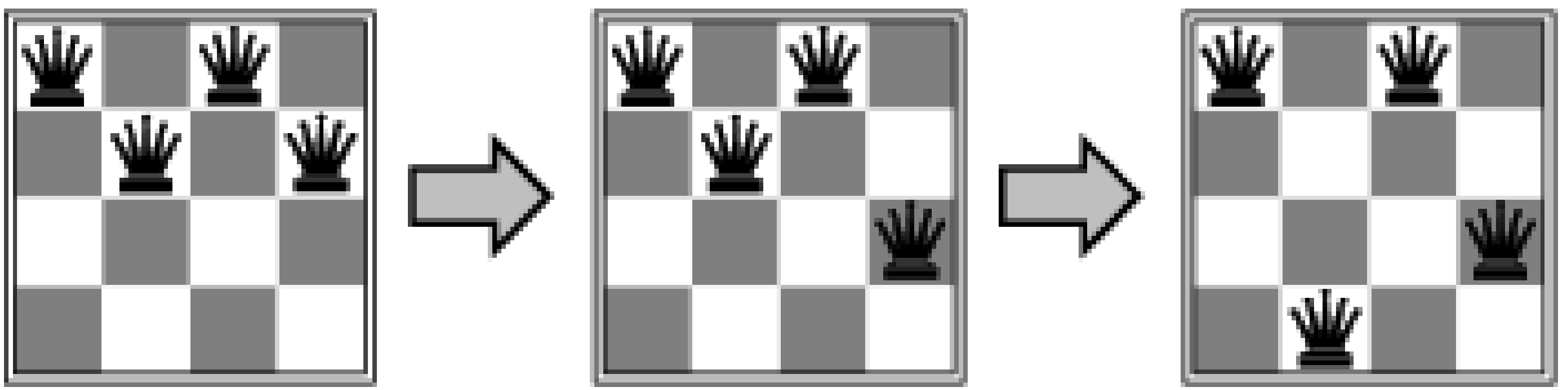
Hill-climbing (greedy local search)

It is easy to get stuck in local maxima



Example: *n*-queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
-



Greedy local search needs

- A “successor” (distance?) function

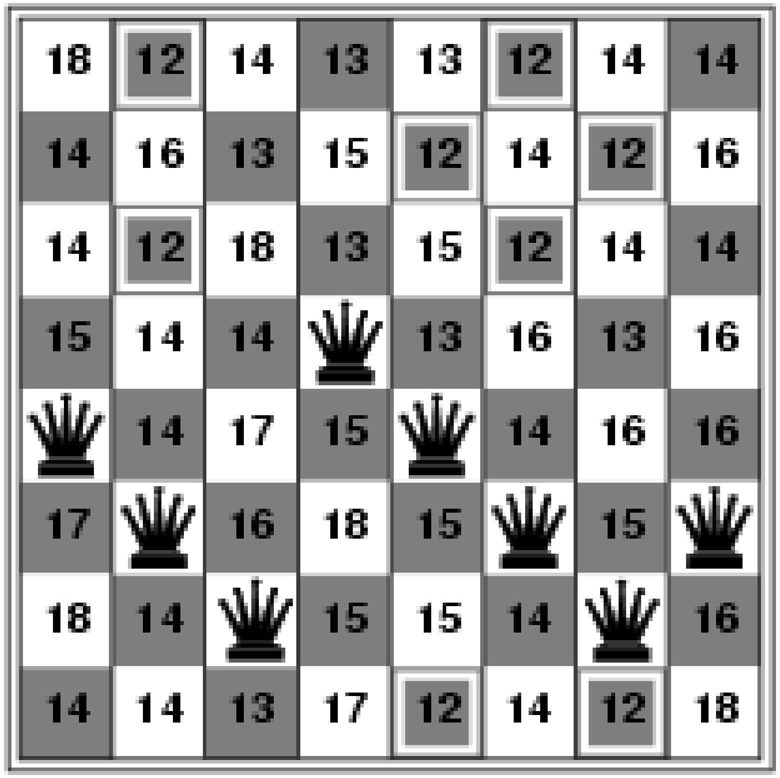
  Any board position that is reachable by moving one queen in her column.

- An optimality (error?) measure

  How many queen pairs can attack each other?

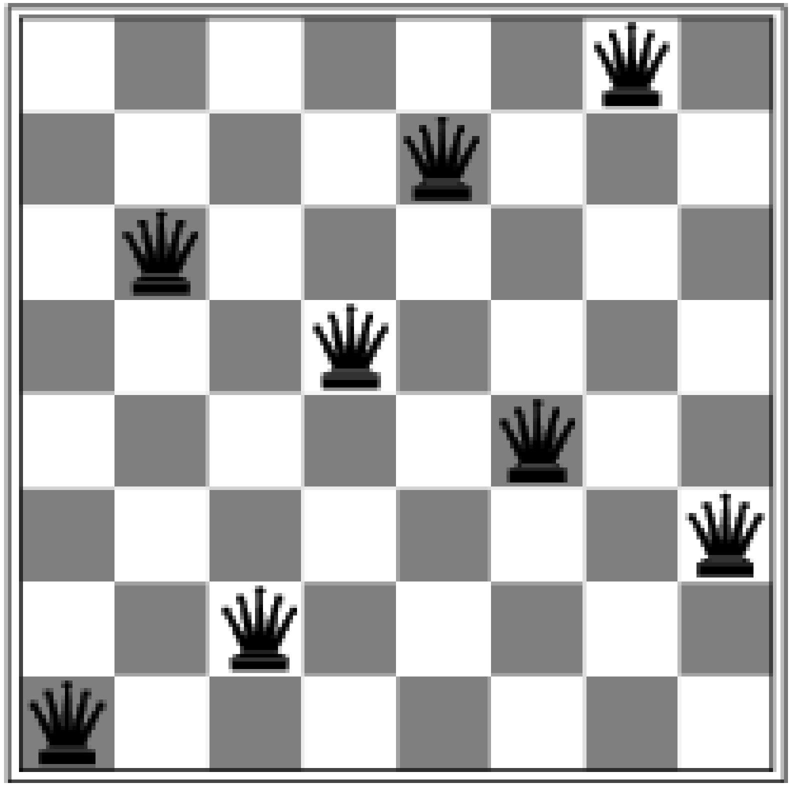
Hill-climbing search: 8-queens problem

h =17



- *h* = number of pairs of queens that are attacking each other, either directly or indirectly

Hill-climbing search: 8-queens problem



- A local minimum with *h = 1*

Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: *current*, a node
    next, a node
    T, a "temperature" controlling prob. of downward steps

  $current \leftarrow$ MAKE-NODE(INITIAL-STATE[*problem*])
  for $t \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow schedule[t]$
    if $T = 0$ **then return** *current*
    $next \leftarrow$ a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
    if $\Delta E > 0$ **then** $current \leftarrow next$
    else $current \leftarrow next$ only with probability $e^{\Delta E/T}$

Properties of simulated annealing

- One can prove: If *T* decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1

- Widely used in VLSI layout, airline scheduling, etc

Local beam search

- Keep track of *k* states rather than just one
- Start with *k* randomly generated states
- At each iteration, all the successors of all *k* states are generated
- If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat.

Let’s look at a demo

Results on 8-queens

	Random	Sim Anneal	Greedy
	600+	173	4
	15	119	4
	154	114	5
Average	**256+**	**135**	**4**

- Note: on other problems, your mileage may vary

Continuous Optimization

- Many AI problems require optimizing a function $f(\mathbf{x})$, which takes continuous values for input vector $\mathbf{x}$

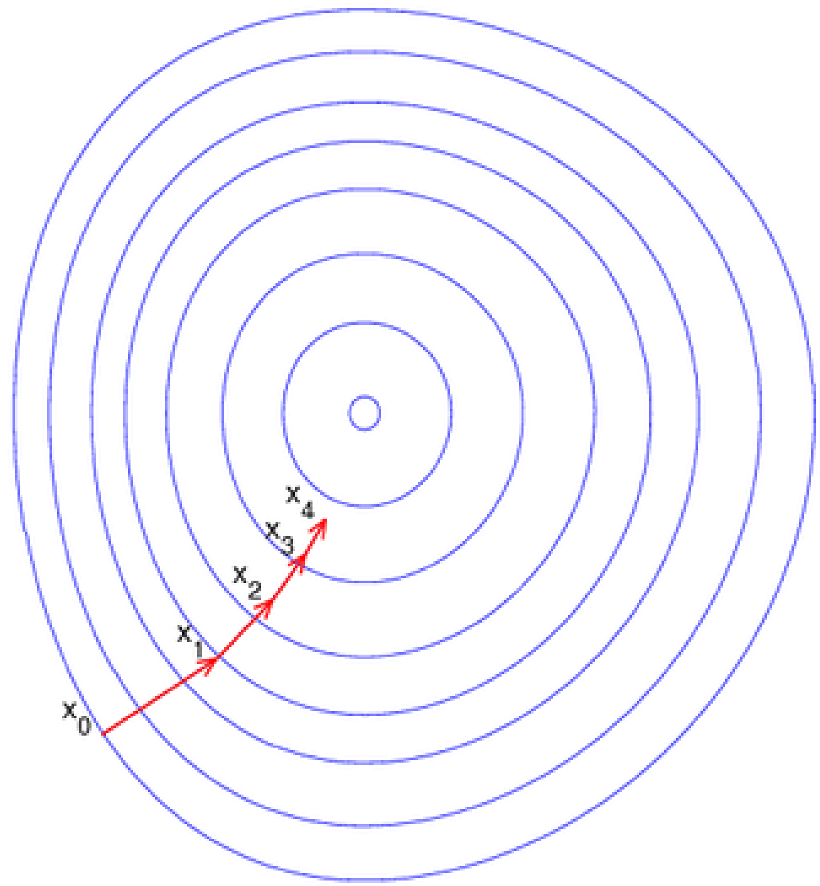
- Huge research area

- Examples:
  - **Machine Learning**
  - Signal/Image Processing
  - Computational biology
  - Finance
  - Weather forecasting
  - Etc., etc.

Gradient Ascent

- Idea: move in direction of steepest ascent (gradient)

- $\mathbf{x}_k = \mathbf{x}_{k\text{-}1} + \eta \nabla f(\mathbf{x}_{k\text{-}1})$



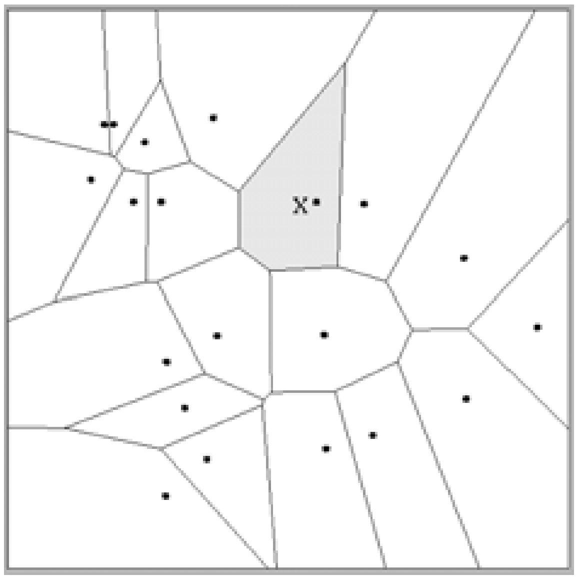
Types of Optimization

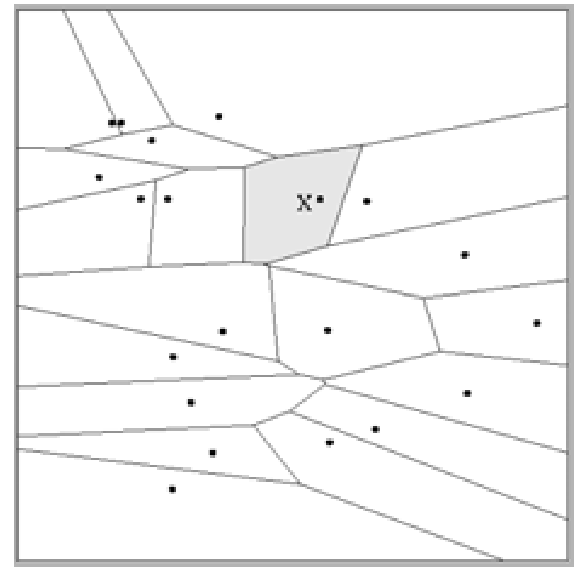
- Linear vs. non-linear
- Analytic vs. Empirical Gradient
- Convex vs. non-convex
- Constrained vs. unconstrained

Continuous Optimization in Practice

- *Lots* of previous work on this
- Use packages

Final example: weights in NN





$$d(x,y) = (x_1 - y_1)^2 + (x_2 - y_2)^2 \quad d(x,y) = (x_1 - y_1)^2 + (3x_2 - 3y_2)^2$$