Machine Learning

Greedy Local Search
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)

\[
\text{find } x_{\text{max}} = \arg \max_{x \in X} (f(x))
\]
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

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

$h = 17$
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

```python
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 ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
    ∆E ← VALUE[next] - VALUE[current]
    if ∆E > 0 then current ← next
    else current ← next only with probability e^∆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

<table>
<thead>
<tr>
<th></th>
<th>Random</th>
<th>Sim Anneal</th>
<th>Greedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>600+</td>
<td>173</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>119</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>154</td>
<td>114</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>256+</strong></td>
<td><strong>135</strong></td>
<td><strong>4</strong></td>
</tr>
</tbody>
</table>

- Note: on other problems, your mileage may vary
Continuous Optimization

• Many AI problems require optimizing a function $f(x)$, which takes continuous values for input vector $x$

• 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)

\[ \mathbf{x}_k = \mathbf{x}_{k-1} + \eta \nabla f(\mathbf{x}_{k-1}) \]
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
• Stochastic vs. Batch
  – Try stochastic first

• Analytic Gradients are hard to debug
  – Use packages with gradients built in (e.g. Tensorflow)
  – Do gradient checking
Final example: weights in NN

\[ d(x, y) = (x_1 - y_1)^2 + (x_2 - y_2)^2 \]

\[ d(x, y) = (x_1 - y_1)^2 + (3x_2 - 3y_2)^2 \]
Reading

• **Gradient Descent**
  – See e.g. the python example midway down the page

• Previous:
  – Nearest neighbor ([Elements of Statistical Learning](https://www.springer.com/gp/book/9780387848586) 2.3, 13.3)