UL01. Randomized Optimization

Optimization:

- An optimization problem typically include:
  - Input space X
  - Objective (Fitness) function $f: x \rightarrow \mathbb{R}$
  - Goal: Find $x^* \in X$ such that $f(x^*) = max_x f(x)$
- Optimization is meant to help us:
  - Find the best process.
  - Find the best route.
  - Find the root.
  - Find the best parameters of the learning algorithms.

Optimization Approaches:

- Generate and test: Small input space – Complex function.
- Calculus: Function has derivative – solvable derivative = 0
- Newton's method: Function has derivative – Iteratively improve – Single optima.
- What if we have:
  - Big input space.
  - Complex function.
  - No (or hard to find) derivative.
  - Many local optima.
- In this case we use Randomized Optimization.

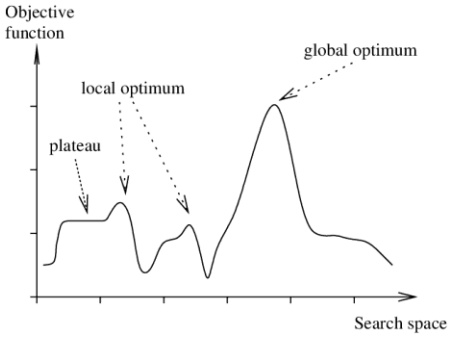
Hill Climbing:

- Guess $x \in X$.
- Repeat:

  Let $n^* = argmax_{n \in N(x)} f(n)$

  If $f(n^*) > f(x): x = n$

  Else: stop (Local minima)



Random Restart Hill Climbing:

Once a local optima is reached, try again starting from a randomly chosen x.

- Advantages:
  1. Multiple tries to find a good starting place.
  2. Not much more expensive (constant factor).

Simulated Annealing:

- Similar to Random Restart Hill Climbing.
- Difference: Don't always improve, sometimes you should explore the space.
- For a finite set of iterations:
  1. Sample new point $x_+ \; in \; N(x)$
  2. Jump to new sample with probability given by an acceptance probability function $P(x, x_+, T)$
  3. Decrease temperature $T > 0$

$$P(x, x_+, T) = \begin{cases} 1, & if \; f(x_+) \geq f(x) \\ e^{\frac{f(x_+) - f(x)}{T}}, & otherwise \end{cases}$$

- If T is big ($T \to \infty$), the exponential part will evaluate to almost 1, and we'll always accept the next point.
- If T is small ($T \to 0$), the exponential will evaluate to 0, and we'll always reject the point.
- The probability of ending at x:

$$P(ending \; at \; x) = \frac{e^{\frac{f(x)}{T}}}{Z_T}$$

- Decreasing T puts all the weight on $f(x)$ and eventually, pushes the probability to its maximum.
- However, we need to decrease T slowly to avoid ending up in a local minima.
- This is called Boltzmann Distribution.

Genetic Algorithms:

- With start with a population of individuals.
- Mutation: We pick a single example and do a local search in its neighbors.
- Crossover: We assume that population holds information. We do parallel random searches across the population.
- Generations: We iterate to improve.
- Algorithm:
  - P_0 → An initial population of size K
  - Repeat until converged:
    1. Compute fitness of all $x \in P_+$
    2. Select "most fit" individuals (Top half → Highest probability, weighted probability → Temperature).
    3. Pair up individuals, replacing "least fit" individuals via crossover/mutation.

Problems with Randomized Optimization Algorithms:

- There's no structure or learning. You start with a point and end up with a point.
- It's not clear what kind of probability distribution we're dealing with.

MIMIC - Finding Optima by Estimating Probability Densities:

- Directly model the distribution.
- Successively refine the model.
- Convey structure.
- Pseudo code:
  - Generate samples from probability distribution $P^{\theta_t}(x)$ → Generate population.
  - Set θ_{t+1} to n[th] percentile. → Retain fittest.
  - Retain only those samples such that $f(x) \geq \theta_{t+1}$ → Retain fittest.
  - Estimate $P^{\theta_{t+1}}(x)$ → Estimate a new distribution.
  - Repeat.
- What we're doing here is:
  1. We have some threshold θ.
  2. We generate a probability distribution that is uniform over all points that have a fitness value $\geq \theta$.
     - This means we generates all the points whose fitness is at least as good as θ.
  3. Take from those the points whose fitness is much higher than θ (Maybe highest 50%).
  4. Keep repeating till you reach θ_{max}.
- This way helps us retain the structure from time step to time step.
- This should work as intended if:
  - We can estimate $P^{\theta_{t+1}}(x)$ given a finite set of data.
  - $P^{\theta_t}(x) \cong P^{\theta_{t+1}}(x)$. That is, when I generate $P^{\theta_t}(x)$, it also gives me samples for the next distribution $P^{\theta_{t+1}}(x)$.

MIMIC – Estimating Distributions:

- The chain rule version of a probability distribution would look like that:

$$P(x) = P(x_1|x_2 \dots x_n)P(x_2|x_3 \dots x_n) \dots P(x_n)$$

  - Every x is a vector of features $[x_1, x_2, x_3 \dots x_n]$
  - $P(x)$ would be the probability of all of the feature of example x being a joint distribution of all the features $[x_1, x_2, x_3 \dots x_n]$.
- We cannot (very hard) estimate this because it's an exponential size probability table.
- However, we can estimate it by making an assumption about conditional independence. That assumption would be that we only care about "Dependency Trees":
  - A Dependency Tree is a special case of Bayesian Networks where the network itself is a tree. That means that every variable has exactly one parent.
- Having made this assumption, we can write the distribution as:

$$\hat{P}_\pi = \prod P(x_i|\pi(x_i))$$

  - Where $\pi(x_i)$ is the parent of x_i.
  - Here each variable has exactly one paper means that no feature is ever conditioned on more than one other feature. This makes the conditional probability stays very small.
- The simplest structure we can use is assuming that each variable has no parent $\prod P(x_i)$, but this means we assume that the features are completely independent of each other, which is not realistic all the time.

- Assuming a Dependency Tree (each variable has only one parent) is simple yet helps us capture the underlying relationships.
- The Dependency Tree also helps us capture the same locality information that we get from Cross Over in Genetic Algorithms.

Information Theory:

- In a machine learning setting, we need to know how each feature of the input examples relates to the output, and which feature has the highest effect. In other worlds, we need to figure out which of these features gives the most information about the output.
- These input/output vectors can be considered as a probability density function.
- Information Theory is a mathematical frame work used to compare these density functions to determine:
  - Mutual information: The similarity between different vectors.
  - Entropy: Does the feature in hand carry any information about the output?
- To calculate the Entropy of variable x:

$$Entropy = -\sum P(x) \log P(x)$$

- Joint Entropy: Randomness contained in two variables together.

$$H(x, y) = -\sum P(x, y) \log P(y, x)$$

- Conditional Entropy: Randomness of one variable given another variable.

$$H(y|x) = -\sum P(x, y)\, log\, P(y|x)$$

- If x and y are independent $(x \| y)$:

$$H(y|x) = H(y)$$
$$H(x, y) = H(x) + H(y)$$

- Conditional Entropy is not an indicative measure of dependence:
  - $H(y|x)$ will be small if y is highly dependent on x, or if $H(y)$ is very small to begin with.
- This is why we measure the Mutual Information:

$$I(x, y) = H(y) - H(x|y)$$

- Kullback–Leibler Divergence: Measures the difference between any two distributions.

$$D_{kl}(p \| q) = \sum p(x) \log \frac{p(x)}{q(x)}$$

  - It's a distance measure. But It's a non-negative quantity, and equals zero only if $p(x) = p(x)$, since the log term will equal zero.

Finding Dependency Trees:

- Given an underlying probability distribution P, we want to find $\hat{P}_\pi$ that is the closest we can get to P.
- To get a sense of the similarity between these two distributions, we will measure the Kullback–Leibler divergence (from Information Theory) between the two of them:

$$D_{kl}(P\|\hat{P}_\pi) = \sum P[\log P - \log \hat{P}_\pi]$$

- To find the closest approximate distribution $\hat{P}_\pi$, we need to minimize $D_{kl}(P\|\hat{P}_\pi)$.
  - Substituting in $D_{kl}(P\|\hat{P}_\pi)$ using the information theory equations:

$$D_{kl}(P\|\hat{P}_\pi) = -H(P) + \sum H(x_i|\pi(x_i))$$

  - Given that we need to find π, the first term $-H(P)$ doesn't matter. We end up with a cost function J_π that we need to minimize:

$$J_\pi = \sum H(x_i|\pi(x_i))$$

  - So, the best Dependency Tree would be the one that minimizes the sum of all Entropy of each feature given its parent.
  - To make easier to minimize J_π, we'll add another term to the equation that is not dependent on π and hence will not affect our calculations:

$$\grave{J_\pi} = -\sum H(x_i) + \sum H(x_i|\pi(x_i))$$

    Substituting in the Mutual Information function:

$$\grave{J_\pi} = -\sum I(x_i, \pi(x_i))$$

    This means that minimizing the cost function is the same thing as maximizing the Mutual Information.