Notes: Deterministic Annealing

1. Review of Simulated (Stochastic) Annealing:

- SA is an optimization technique that can be employed to find global minima or maxima.
- It is derived from statistical mechanics and simulates the process of annealing, which is the gradual cooling of metal or glass where the material’s molecules settle into an optimum lattice structure.
- SA simulates the random evolution of a physical system and reaches equilibrium as the steady-state distribution over states of a corresponding Markov chain.
- SA can be shown to converge in probability to the set of globally optimal solutions.
- Can be extremely computationally expensive for many problems.

\[
C(x) = 5(\sin(x))^3 + x^2 - 2x + 8
\]

- Basic Algorithm – Cost function \( C, \Delta(C) = C(x_{\text{new}}) - C(x_{\text{old}}), \) temperature \( T, \ k = \text{Boltzmann’s constant.} \)

1. Set the independent variables to their expected values - this is used as the initial centering point.
2. Set \( T \) to a relatively high number.
3. Perturb the independent variables.
4. Calculate the new result with the new variables.
5. If the new result is lower than or equal to the best, save the result.
6. Else if the new result is higher than the best, choose a random number \( r \) uniformly from \([0,1]\). If \( r < \exp\left(-\Delta(C)/kT\right) \), then save result.
7. Repeat steps 3-6 \( n \) number of times.
8. If an improvement has been made after the \( n \) number of iterations, set the center point to be the best point.
9. Reduce the temperature.
10. Repeat steps 3-9 for \( t \) number of temperatures.

- Steps 3-6 correspond to the Metropolis algorithm which we saw in the MCMC paper. Recall that in the MCMC paper, Metropolis is defined as having an acceptance probability of:

\[
A(x^{(i)}, x^*) = \min\{1, \frac{p(x^*)}{p(x^{(i)})}\}
\]

- When we define our cost function, \( C \), to be \(-\log(p(x))\), we can see that steps 3-6 are equivalent to Metropolis because we accept with probability 1 when our new result is better than the last (corresponding to a ratio of >1 in the Metropolis formula), otherwise (disregarding constants \( T \) and \( k \)) we accept with probability

\[
\exp\{\log(x^*) - \log(x^{(i)})\} = \frac{x^*}{x^{(i)}}
\]

2. **Deterministic Annealing**:

- Minimizes the Lagrangian (which is analogous to the Helmholtz free energy equation in statistical mechanics):

\[
F = \langle D \rangle - TH,
\]

where \( \langle D \rangle \) is the expectation of the target function to be minimized, \( T \) is a Lagrange multiplier (temperature), and \( H \) is the Shannon entropy of the Markov chain.

- Rather than simulating the exact stochastic evolution of the system, DA determines the effective distribution over the states of the system at each temperature and optimizes the expected value of the cost function (free energy equation \( F \)) directly.

- So the main difference of DA as compared to SA is that it does not evolve a minimum at each temperature through many modifications to the target function’s parameters, but instead finds the minimum at each temperature directly.

- Does not guarantee a global optimum, but can still avoid many local minima.

- Basic Algorithm:

1. Set parameters: initial temperature \( T_i \), final temperature \( T_f \), minimum entropy \( H_{min} \) and annealing schedule function \( \alpha() \).
2. Set \( T = T_i \) and \( \gamma = Y_{init} \)
3. \( F = \min_{\langle \theta_i \rangle, \gamma} \{ \langle D \rangle - TH \} \) (Using gradient descent or similar)
4. Lower temperature: \( T \leftarrow \alpha(T) \)
5. If \( T > T_f \) goto step 3; else goto step 6.
6. Quenching: Increase \( \gamma \) according to \( \gamma \leftarrow q(\gamma), \min_{\langle \theta_i \rangle} \langle D \rangle \)
7. If $H > H_{\text{min}}$, goto step 6

- Note that $q(\gamma)$ is a ‘quenching’ schedule which, for practical reasons, is used to gradually increase $\gamma$ to a very high value when $T$ falls below a certain level. A typical quenching schedule might be $q(\gamma) = 1.2^\gamma$.

3. Pattern Classification:

- Our task:
  - Given features $X$ (continuous) and classes $Y$ (discrete)
  - Estimate $Y$ from $X$
  - Objective: Design classifier $C(X)$ to minimize $P(C(X) \neq Y)$

- Two approaches: Modeling & Direct

  - Modeling:
    - Assume parametric form of classifier. (ex. Gaussian)
    - Learn the pdf for each class.
    - Classify based on the maximum (ie. $\hat{y} = \arg \max_i d_i(x)$, where $d_i(x)$ is the discriminant function for class $i$ generated by model $H_i$).
    - Strong assumption on the nature of the pdf.
    - Implies constraint on the form of the classifier.
    - Sub-optimal

  - Direct:
    - Assume parametric form of classifier.
    - Minimize misclassification.
    - Hard problem because cost function is non-convex and has zero gradient almost everywhere.
    - Use deterministic annealing.

4. Using DA in the Classification Problem:

- Temporarily allow random classification instead of choosing the winning class by taking the max of the discriminant functions as in: $\hat{y} = \arg \max_i d_i(x)$
- Choose class $j$ for sample $x$ with probability, $P_x(j) \propto e^{d_j(x)}$
- $P(y_j \mid x) = \frac{e^{d_j(x)}}{\sum_j e^{d_j(x)}}$
• $\gamma$ controls the ‘fuzziness’ of the distribution.
• As $\gamma \to \infty$, the highest discriminant function wins with probability 1.
• For finite positive $\gamma$, classes with larger discriminant function values are assigned higher probabilities of winning.
• For $\gamma = 0$, the distribution is uniform.
• Probability of error: $\langle P_e \rangle = 1 - \frac{1}{N} \sum_i P_{x_i}(y_i)$
• We also need to define the entropy of our distribution so that we can constrain the level of randomness in the classifier:

$$H = -\frac{1}{N} \sum_i \sum_j P_{x_i}(j) \log P_{x_i}(j)$$

• Ultimate objective: minimize $P_e$ when $H = 0$.
• Lagrangian equivalent: $F = \min_{(H, \gamma)} \langle P_e \rangle$ ($< P_e >$ = $TH$)
• As $T \to 0$, optimization reduces to the unconstrained minimization of $\langle P_e \rangle$,
  which forces $\gamma \to \infty$ and leads to the optimal non-random maximum discriminant classifier.
• When $T = 0$, classifier becomes a hard (non-random) classifier.
• Much faster than SA.

5. Using DA with HMMs for Speech Recognition:

• Normally, we use the Maximum Likelihood technique to design HMM classifiers for speech recognition. While ML is fast, it does not directly address the goal of classifier design which is minimization of error.
• Minimum Classification Error techniques attempt to solve the direct problem instead. However this is challenging because the error surface is extremely difficult to optimize due to its piecewise-linear nature which prevents the use of gradient descent techniques.
• DA allows us to directly minimize the cost surface while simultaneously avoiding many local minima.
• Given a training set $\{(x_1, c_1), (x_2, c_2), \ldots, (x_N, c_N)\}$, where $x_i$ (a feature vector of length $l_i$) is an utterance of the word $c_i$ which belongs to a finite sized dictionary $C = \{1, \ldots, J\}$.
• Recognition system consists of a set of HMMs $\{H_j; j = 1, 2, \ldots, J\}$, one for each word in the dictionary.
• Each model $H_j$ has $S_j$ states and is fully supported by a parameter set $\theta_j$, which consists of the usual HMM components (priors $\pi_j$, transition probabilities $A_j$, and observation probabilities $B_j$).
• We determine our classifier $C$, which maps training pattern $x_i$ to class $C(x_i)$, in the following way:
  - Given the pattern $x_i$, a path score is computed for each HMM $H_j$ and for each sequence of states $s = (s(0), s(1), \ldots, s(l_j))$ in $H_j$’s trellis:
\[ l(x_i, s, H_j) = \frac{1}{l_i} \{ \log \pi_j [s(1)] + \sum_{t=1}^{l_i-1} \log A_j [s(t), s(t+1)] + \sum_{t=1}^{l_i} \log B_j [s(t), x_i(t)] \} \]

- The path with the highest score is determined and we map \( x_i \) to the class of the HMM that the winning path belongs to.
- \( d_j(x_i) = \max_{s \in S_{l_i}(H_j)} l(x_i, s, H_j) \) where \( S_{l_i}(H_j) \) is the set of states of length \( l_i \) in model \( H_j \).
- \( C(x_i) = \arg \max_j d_j(x_i) \)

- The empirical error rate is:
  \[ P_e = \min_{\theta_j} \{ 1 - \frac{1}{N} \sum_{i=1}^{N} \delta(C(x_i), c_i) \} \]
  where \( \delta() \) is the Kronecker delta function.

- The problem is the piecewise constant nature of \( P_e \), making direct descent optimization impossible.

- ML avoids this problem by replacing the true cost function with a sub-optimal design objective.

- To use DA, we must first define a randomized version of our best path classification rule which instead of assigning pattern \( x_i \) to a unique winning state sequence, associates each pattern \( x_i \) with every state sequence \( s \), in the trellis of every HMM \( H_j \) with probability \( P[s, j \mid x_i] \).

- This is the probability of selecting state sequence \( s \) and consequently, HMM \( H_j \) and is optimally modeled by the Gibbs distribution:
  \[ P[s, j \mid x_i] = \frac{e^{\gamma(x_i, s, H_j)}}{\sum_{j'} \sum_{s' \in S_{l_i}(H_{j'})} e^{\gamma(x_i, s', H_{j'})}} \]

- The \( \gamma \) parameter determines the level of entropy of the distribution in the same way as described earlier.

- We can now actually minimize the classifier error using the expected misclassification rate of the random classifier:
  \[ \langle P_e \rangle = 1 - \frac{1}{N} \sum_{i=1}^{N} P[c_i \mid x_i] \]

- \( P[c_i \mid x_i] \) is the probability that the correct class is selected as the winner and can be found by summing over paths:
  \[ P[c_i \mid x_i] = \sum_{s \in S_{l_i}(H_{j_i})} P[s, c_i \mid x_i] \]

- So our design problem for the random classifier is to find the optimal values of the model parameters \( \{ \theta_j \} \) and \( \gamma \), which determine \( P[s, c_i \mid x_i] \) so as to minimize the probability of error.

- In order to keep from getting stuck in shallow local minima, we need to gradually reduce the randomness of the classifier through a constraint on the entropy, \( H \):
\[ H = -\frac{1}{N} \sum_{i} \sum_{j} \sum_{s \in S_i(H_i)} P[s, j | x_i] \log P[s, j | x_i] \]

- Our constrained optimization problem is equivalent to the unconstrained Lagrangian:
  \[ F = \min_{(\theta_j)_j} \{ < P_e > - \Theta H \} \]
- We now use DA as described earlier and obtain a sequence of solutions of decreasing entropy and \( <P_e> \) leading to a best path classifier in the limit.