Hyun Min Kang

April 5th, 2011

Recap - Dynamic Polymorphisms

class shape {
  // shape is an abstract class
  public:
    virtual double area() = 0;  // shape object will never be created
}
class rectangle : public shape {
  public:
    double x;
    double y;
    virtual double area() { return x*y; }
};
class circle : public shape {
  public:
    double r;
    circle(double _r) : r(_r) {}
    virtual double area() { return M_PI*r*r; }
};

Recap : Function objects using dynamic polymorphisms

class optFunc {
  public:
    virtual double operator() (std::vector<double>& x) = 0;
};
class arbitraryOptFunc : public optFunc {
  public:
    virtual double operator() (std::vector<double>& x) {
      return 100*(x[1]-x[0]*x[0])*(x[1]-x[0]*x[0])+(1-x[0])*(1-x[0]);
    }
};
class mixLLKFunc : public optFunc {
  ... // many auxiliary functions
  public:
    std::vector<double> data;
    virtual double operator() (std::vector<double>& x) {
      ...}
};

E-M algorithm : A Basic Strategy

- Complete data \((x, z)\) - what we would like to have
  - Observed data \(x\) - individual observations
  - Missing data \(z\) - hidden / missing variables
- The algorithm
  - Use estimated parameters to infer \(z\)
  - Update estimated parameters using \(x\)
  - Repeat until convergence
Recap: The E-M algorithm

Expectation step (E-step)

- Given the current estimates of parameters $\theta^{(t)}$, calculate the conditional distribution of latent variable $z$.
- Then the expected log-likelihood of data given the conditional distribution of $z$ can be obtained

$$Q(\theta|\theta^{(t)}) = E_{z|x,\theta^{(t)}} [\log p(x,z|\theta)]$$

Maximization step (M-step)

- Find the parameter that maximize the expected log-likelihood

$$\theta^{(t+1)} = \arg \max_{\theta} Q(\theta|\theta^{t})$$

Local and global optimization methods

Local optimization methods

- "Greedy" optimization methods
  - Can get trapped at local minima
  - Outcome might depend on starting point
- Examples
  - Golden Search
  - Nelder-Mead Simplex Method
  - E-M algorithm

Today

- Simulated Annealing
- Markov-Chain Monte-Carlo Method
- Designed to search for global minimum among many local minima

Summary: The E-M Algorithm

- Iterative procedure to find maximum likelihood estimate
  - E-step: Calculate the distribution of latent variables and the expected log-likelihood of the parameters given current set of parameters
  - M-step: Update the parameters based on the expected log-likelihood function
- The iteration does not decrease the marginal likelihood function
- But no guarantee that it will converge to the MLE
- Particularly useful when the likelihood is an exponential family
  - The E-step becomes the sum of expectations of sufficient statistics
  - The M-step involves maximizing a linear function, where closed form solution can often be found

Local minimization methods

The problem

- Most minimization strategies find the nearest local minimum from the starting point
- Standard strategy
  - Generate trial point based on current estimates
  - Evaluate function at proposed location
  - Accept new value if it improves solution

The solution

- We need a strategy to find other minima
- To do so, we sometimes need to select new points that does not improve solution
- How?
Simulated Annealing

### Annealing
- One manner in which crystals are formed
- Gradual cooling of liquid
  - At high temperatures, molecules move freely
  - At low temperatures, molecules are "stuck"
- If cooling is slow
  - Low energy, organized crystal lattice formed

### Simulated Annealing Strategy
- Consider decreasing series of temperatures
- For each temperature, iterate these steps:
  - Propose an update and evaluation function
  - Accept updates that improve solution
  - Accept some updates that don’t improve solution
    - Acceptance probability depends on “temperature” parameter
- If cooling is sufficiently slow, the global minimum will be reached

Local minimization methods

The problem with hill climbing is that it gets stuck on “local-maxima”


Simulated Annealing can escape local minima with chaotic jumps

Example Applications

- The traveling salesman problem (TSP)
  - Salesman must visit every city in a set
  - Given distances between pairs of cities
  - Find the shortest route through the set

- No polynomial time algorithm is known for finding optimal solution
- Simulated annealing or other stochastic optimization methods often provide near-optimal solutions.

Examples of simulated annealing results

- A good scheme should be able to
  - Connect any two possible paths
  - Propose improvements to good solutions
- Some possible update schemes
  - Swap a pair of cities in current path
  - Invert a segment in current path

Update scheme in Simulated Annealing

- Random walk by Metropolis criterion (1953)
- Given a configuration, assume a probability proportional to the Boltzmann factor
  \[ P_A = e^{-E_A/T} \]
- Changes from \( E_0 \) to \( E_1 \) with probability
  \[ \min \left( 1, \frac{P_1}{P_0} \right) = \min \left( 1, \exp \left( -\frac{E_1 - E_0}{T} \right) \right) \]
- Given sufficient time, leads to equilibrium state
Simulated Annealing Recipes

Key requirements

- Irreducibility: it is possible to get any state from any state
  - There exist \( t \) where \( \Pr(q_t = j | q_0 = i) > 0 \) for all \( (i, j) \).
- Aperiodicity: return to the original state can occur at irregular times
  \[ \gcd \{ t : \Pr(q_t = i | q_0 = i) > 0 \} = 1 \]
- These two conditions guarantee the existence of a unique equilibrium distribution

Using Markov Chains

Markov Chain Revisited

- The Markovian property
  \[ \Pr(q_t | q_{t-1}, q_{t-2}, \ldots, q_0) = \Pr(q_t | q_{t-1}) \]
- Transition probability
  \[ \theta_{ij} = \Pr(q_t = j | q_{t-1} = i) \]

Simulated Annealing using Markov Chain

- Start with some state \( q_t \).
- Propose a changed \( q_{t+1} \) given \( q_t \).
- Decide whether to accept change based on \( \theta_{q_t,q_{t+1}} \)
  - Decision is based on relative probabilities of two outcomes

Equilibrium distribution

- Starting point does not affect results
- The marginal distribution of resulting state does not change
- Equilibrium distribution \( \pi \) satisfies
  \[ \pi = \lim_{t \to \infty} \Theta^{t+1} = (\lim_{t \to \infty} \Theta^t) \Theta = \pi \Theta \]
- In Simulated Annealing, \( \Pr(E) \propto e^{-E/T} \)
Practical issues

- The maximum temperature
- Scheme for decreasing temperature
- Strategy for proposing updates
  - For mixture of normals, suggestion of Brooks and Morgan (1995) works well
  - Select a component to update, and sample from within plausible range

Implementing Simulated Annealing

class normMixSA {
public:
    int k; // # of components
    int n; // # of data
    std::vector<double> data; // observed data
    std::vector<double> pis; // pis
    std::vector<double> means; // means
    std::vector<double> sigmas; // sds
    double llk; // current likelihood

    normMixSA(std::vector<double> _data, int _k); // constructor
    void initParams(); // initialize parameters
    double updatePis(double temperature);
    double updateMeans(double temperature, double lo, double hi);
    double updateSigmas(double temperature, double sdlo, double sdhi);
    double runSA(double eps); // run Simulated Annealing

    static int acceptProposal(double current, double proposal, double temperature);
};

Evaluating Proposals in Simulated Annealing

int normMixSA::acceptProposal(double current, double proposal, double temperature) {
    if ( proposal < current ) return 1; // return 1 if likelihood decreased
    if ( temperature == 0.0 ) return 0; // return 0 if frozen
    double prob = exp(0-(proposal-current)/temperature);
    return (randu(0.,1.) < prob); // otherwise, probabilistically accept proposal
}
### Updating Means

```cpp
double normMixSA::updateMeans(double temperature, double min, double max) {
    int c = randn(0,k);  // select a random integer between 0..(k-1)
    double old = means[c];  // save the old mean for recovery
    means[c] = randu(min, max);  // update mean and evaluate the likelihood
    double proposal = 0-mixLLKFunc::mixLLK(data, pisCopy, means, sigmas);
    if ( acceptProposal(llk, proposal, temperature) ) {
        llk = proposal;  // if accepted, keep the changes
    } else {
        means[c] = old;  // if rejected, rollback the changes
    }
    return llk;
}
```

### Updating Mixture Proportions

- Mixture proportions must sum to 1.0
- When updating one proportion, must take others into account
- Select a component at random
  - Increase or decrease probability by up to 25%
  - Rescale all proportions so they sum to 1.0

```cpp
double normMixSA::updatePis(double temperature) {
    std::vector<double> pisCopy = pis;  // make a copy ofpis
    int c = randn(0,k);  // select a random component to update
    pisCopy[c] *= randu(0.8,1.25);  // update the component
    // normalize pis
    double sum = 0.0;
    for(int i=0; i<k; ++i)
        sum += pisCopy[i];
    for(int i=0; i<k; ++i)
        pisCopy[i] /= sum;
    double proposal = 0-mixLLKFunc::mixLLK(data, pisCopy, means, sigmas);
    if ( acceptProposal(llk, proposal, temperature) ) {
        llk = proposal;
        pis = pisCopy;  // if accepted, update pis to pisCopy
    }
    return llk;
}
```
### Initializing parameters

```cpp
void normMixSA::initParams() {
    double sum = 0, sqsum = 0;
    for(int i=0; i < n; ++i) {
        sum += data[i];
        sqsum += (data[i]*data[i]);
    }
    double mean = sum/n;
    double sigma = sqrt(sqsum/n - sum*sum/n/n);
    for(int i=0; i < k; ++i) {
        pis[i] = 1./k; // uniform priors
        means[i] = data[rand()] * n; // pick random data points
        sigmas[i] = sigma; // pick uniform variance
    }
}
```

### Running examples

```bash
user@host:~/./mixSimplex ./mix.dat
Minimim = 3043.46, at pi = 0.667271,
between N(-0.0304604,1.00326) and N(5.01226,0.956009)
```

```bash
user@host:~/./mixEM ./mix.dat
Minimim = -3043.46, at pi = 0.667842,
between N(-0.030148,1.00478) and N(5.01245,0.91296)
```

```bash
user@host:~/./mixSA ./mix.dat
Minimim = 3043.46, at pi = 0.667793,
between N(-0.030148,1.00478) and N(5.01245,0.91296)
```

### Comparisons

#### 2-component Gaussian mixtures
- Simplex Method : 306 Evaluations
- E-M Algorithm : 12 Evaluations
- Simulated Annealing : ~ 100,000 Evaluations

#### For higher dimensional problems
- Simplex Method may not converge, or converge very slowly
- E-M Algorithm may stuck at local maxima when likelihood function is multimodal
- Simulated Annealing scale robustly with the number of dimensions.
Summary

Today - Simulated Annealing

- Simulated Annealing
- Markov-Chain Monte-Carlo method
- Searching for global minimum among local minima

Next lecture

- More on MCMC Method
- A simple Gibbs Sampler