MTH 511a - 2020: Lecture 24

Instructor: Dootika Vats

1 Stochastic optimization methods

Last lecture we went over the stochastic gradient ascent algorithm: the merit of this algorithm was its use in online sequential data and for large data set problem.

This lecture focuses on simulated annealing, an algorithm particularly useful for non-concave objective functions. Our goal is the same as before: for an objective function $f(\theta)$, our goal is to find

$$\theta^* = \arg\max_{\theta} f(\theta)\,.$$

1.1 Simulated annealing

Recall that when the objective function is non-concave, all of the methods we've discussed cannot escape out of a local maxima. This creates challenges in obtaining global maximas. This is where the method of *simulated annealing* has an advantage over other methods.

Consider an objective function $f(\theta)$ to maximize. Note that maximizing $f(\theta)$ is equivalent to maximizing $\exp(f(\theta))$. The idea in simulated annealing is that, instead of trying to find a maxima directly, we will obtain samples from the density

$$\pi(\theta) \propto \exp(f(\theta))\,.$$

Wherever there is a maxima, samples collected from $\pi(x)$ are likely to be from areas near the maximas. However, obtaining samples from $\pi(x)$ means there will be samples from low probability areas as well. So how do we force samples to come from areas near the maximas?

Consider for $T > 0$,

$$\frac{\partial e^{f(\theta)/T}}{d\theta} = e^{f(\theta)/T} \frac{f'(\theta)}{T},$$

which has the same roots and direction as $f(\theta)$. Thus,

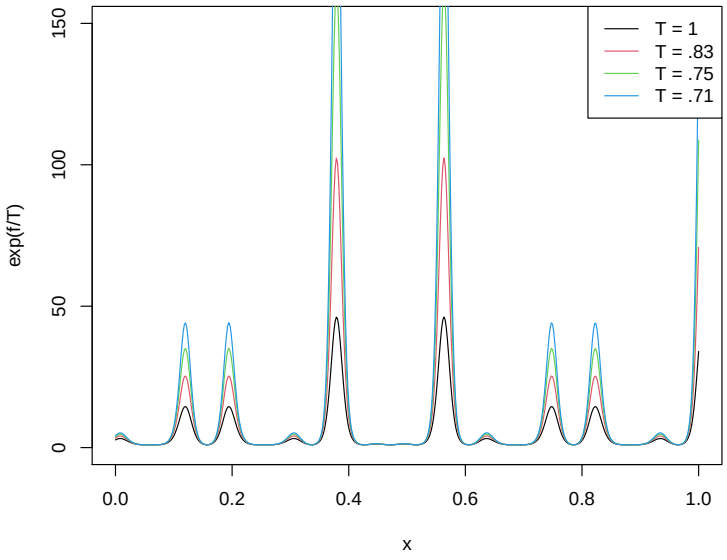
$$\arg\max_{\theta} f(\theta) = \arg\max_{\theta} e^{\{f(\theta)/T\}}.$$

For $0 < T < 1$, the objective function's modes are exaggerated there-by amplifying the maximas.

Example 1. Consider the following objective function

$$f(\theta) = [\cos(50\theta) + \sin(20\theta)]^2 I(0 < \theta < 1)$$

Below is a plot of $e^{f(\theta)/T}$ for various values of T.



In simulated annealing, this feature is utilized so that every subsequent sample is drawn from an increasingly concentrated distribution. That is, at a time point k, a sample will be drawn from

$$\pi_{k,T}(\theta) \propto e^{f(\theta)/T_k}.$$

How do we generate these samples?

Certainly, we can try and use accept-reject or another Monte Carlo sampling method, but such methods cannot be implemented generally.

Note that for any θ', θ

$$\frac{\pi_{k,T}(\theta')}{\pi_{k,T}(\theta)} = \exp\left\{\frac{f(\theta') - f(\theta)}{T_k}\right\}.$$

Let G be a proposal distribution with density $g(\theta'|\theta)$ so that $g(\theta'|\theta) = g(\theta|\theta')$. Such a proposal distribution is a symmetric proposal distribution.

Algorithm 1 Simulated Annealing algorithm

1: For $k = 1, \ldots, N$, repeat the following:
2: Generate $\theta' \sim G(\cdot|\theta_k)$ and generate $U \sim U(0,1)$
3: Let $\alpha = \min\left\{1, \exp\left\{\frac{f(\theta') - f(\theta)}{T_k}\right\}\right\}$.
4: If $U < \alpha$, then let $\theta_{k+1} = \theta'$
5: Else $\theta_{k+1} = \theta_k$.
6: Update T_{k+1}
7: Store θ_{k+1} and $e^{f(\theta_{k+1})}$.
8: Return $\theta^* = \theta_{k^*}$ where k^* is such that $k^* = \arg\max_k e^{f(\theta_{k+1})}$

Thus, if the proposed value is such that $f(\theta') > f(\theta)$, then $\alpha = 1$ and the move is always accepted. The reason simulated annealing works is because when θ' is such that $f(\theta') < f(\theta)$, even then, the move is accepted with probability α. Thus, there is always a chance to move out of local maximas.

Essentially, each θ_k is approximately distributed as $\pi_{k,T}$, and as $T_k \to 0$, $\pi_{k,T}$ puts more and more mass on the maximas, thus, θ_k will typically be get increasingly closer to θ^*.

- Typically, $G(\cdot|\theta)$ is $U(\theta - r, \theta + r)$ or $N(\theta, r)$ which are both valid symmetrical proposals. The parameter r dictates how far/close the proposed values will be.
- T_k is often called the *temperature* parameter. A common value of $T_k = d/\log(k)$ for some constant d.

Example 2 (continued...). We implement the simulated annealing algorithm for:

$$f(\theta) = [\cos(50\theta) + \sin(20\theta)]^2 I(0 < \theta < 1)$$

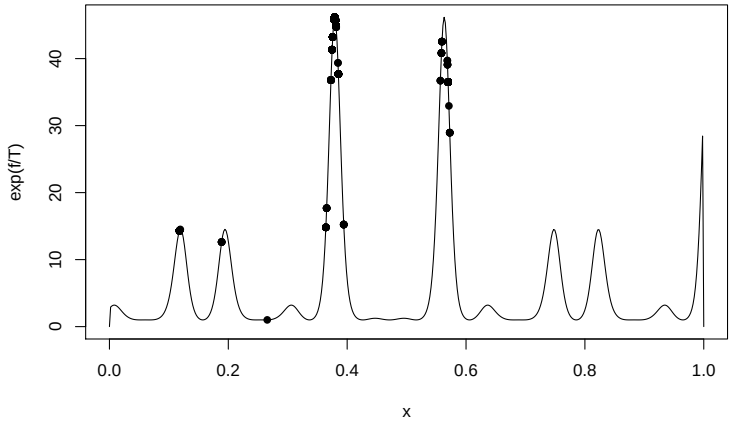
The true $\theta^* \approx .379$.

```
#####################################
## Simulated Annealing
## Demonstrative example
#####################################
fn <- function(x, T = 1)
{
  h <- ( cos(50*x) + sin(20*x) )^2
  exp(h/T)* (0 < x & x < 1)
```

```
}

simAn <- function(N = 10, r = .5)
{
  x <- numeric(length = N)
  x[1] <- runif(1)

  for(k in 2:N)
  {
    # U(x - r, x + r)
    a <- runif(1, x[k-1] - r, x[k-1] + r)
    T <- 1/(log(k))
    ratio <- fn(a,T)/fn(x[k-1], T)
    if( runif(1) < ratio)
    {
      x[k] <- a # accept
    } else{
      x[k] <- x[k-1] # reject, so stay
    }
  }
 return(x)
}
```

Below I implement the algorithm for 500 steps and return the estimate of θ^*. I also plot the values of θ obtained.

```
N <- 500
sim <- simAn(N = N)
sim[which.max(fn(sim))] # theta^*
[1] 0.3792136

x <- seq(0, 1, length = 5e2)
plot(x, fn(x), type = 'l')
points(sim, fn(sim), pch = 16, col = 1)
```

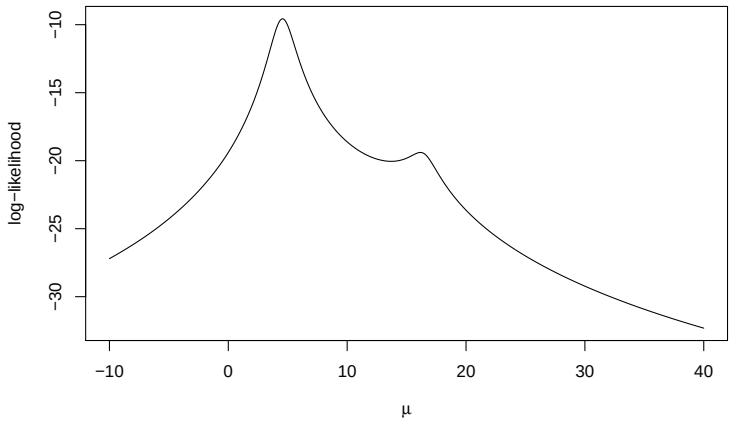


Example 3 (Location Cauchy). Recall the location Cauchy example discussed in Week 6 Lecture 15. The objective function is the log-likelihood of the location Cauchy distribution with mode at $\mu \in \mathbb{R}$. The goal is to find the MLE for μ.

$$f(x|\mu) = \frac{1}{\pi} \frac{1}{(1 + (x - \mu)^2)} .$$

The log-likelihood is

$$l(\mu) := \log L(\mu|X) = -n \log \pi - \sum_{t=1}^{n} \log(1 + (X_i - \mu)^2) .$$



Recall that for the dataset generated, the log-likelihood (above) was not concave and presented many local maxima. This caused Newton-Raphson to possible diverge/converge to minima/local maxima and caused the gradient ascent algorithm to converge

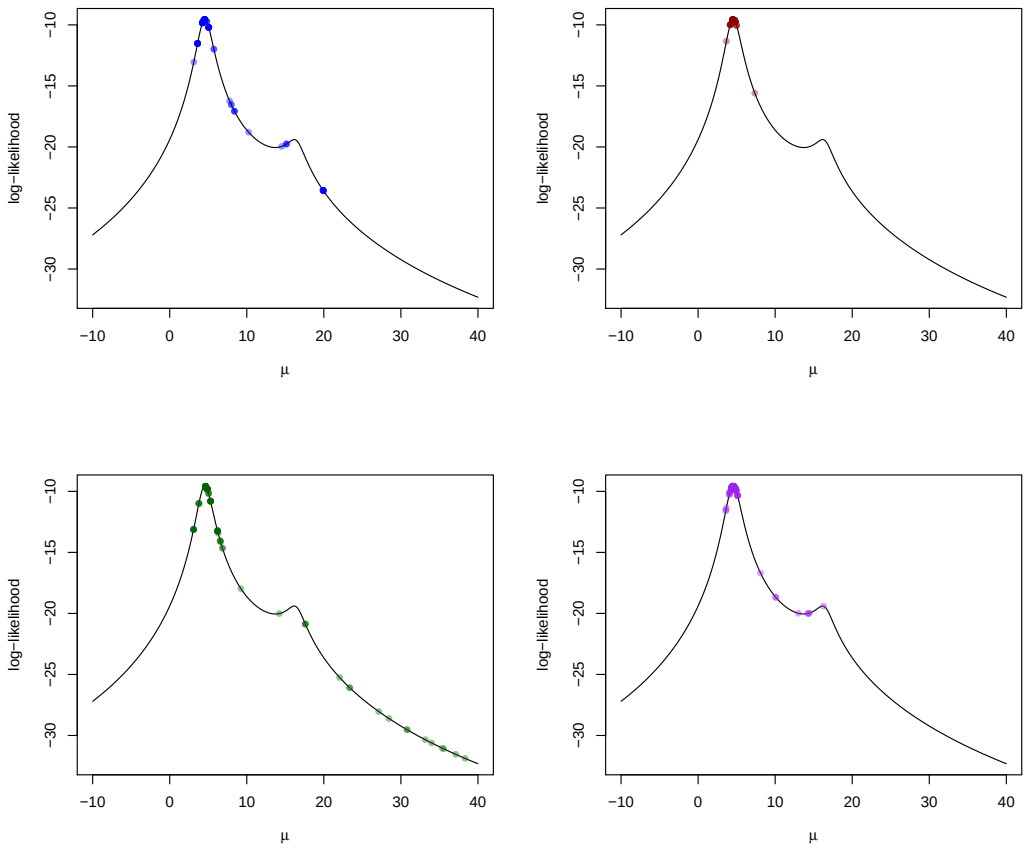
to local maxima. We will implement the simulated annealing algorithm here with $G = U(\theta - r, \theta + r)$.

```
## Function calculates the exp(like/T)
log.like <- function(mu, X, T = 1)
{
  n <- length(X)
  rtn <- -n*log(pi) - sum( log(1 + (X - mu)^2) )
  return(exp(rtn/T))
}

# Simulated annealing algorithm
simAn <- function(N = 10, r = .5)
{
  x <- numeric(length = N)
  x[1] <- runif(1, min = -10, max = 40)
  fn.value <- numeric(length = N)

  fn.value[1] <- log.like(mu = x[1], X, T = 1)
  for(k in 2:N)
  {
    a <- runif(1, x[k-1] - r, x[k-1] + r)
    T <- 1/(1 + log(log(k)))
    ratio <- log.like(mu = a, X, T)/log.like(mu = x[k-1], X, T)
    if( runif(1) < ratio)
    {
      x[k] <- a
    } else{
      x[k] <- x[k-1]
    }
    fn.value[k] <- log.like(mu = x[k], X, T = 1)
  }
  x
  return(list("x" = x, "fn.value" = fn.value))
}
```

I run the algorithm for 100 steps from four randomly chosen starting points.

```
par(mfrow = c(2,2))

# Four different runs all converge.
sim <- simAn(N = 1e2, r = 5)
plot(mu.x, ll.est, type = 'l', ylab = "log-likelihood", xlab =
    expression(mu))
points(sim$x, log(sim$fn.value), pch = 16, col = adjustcolor("blue", alpha
    = .4))
```

```
sim <- simAn(N = 1e2, r = 5)
plot(mu.x, ll.est, type = 'l', ylab = "log-likelihood", xlab =
    expression(mu))
points(sim$x, log(sim$fn.value), pch = 16, col = adjustcolor("darkred",
    alpha = .4))

sim <- simAn(N = 1e2, r = 5)
plot(mu.x, ll.est, type = 'l', ylab = "log-likelihood", xlab =
    expression(mu))
points(sim$x, log(sim$fn.value), pch = 16, col = adjustcolor("darkgreen",
    alpha = .4))

sim <- simAn(N = 1e2, r = 5)
plot(mu.x, ll.est, type = 'l', ylab = "log-likelihood", xlab =
    expression(mu))
points(sim$x, log(sim$fn.value), pch = 16, col = adjustcolor("purple",
    alpha = .4))
```

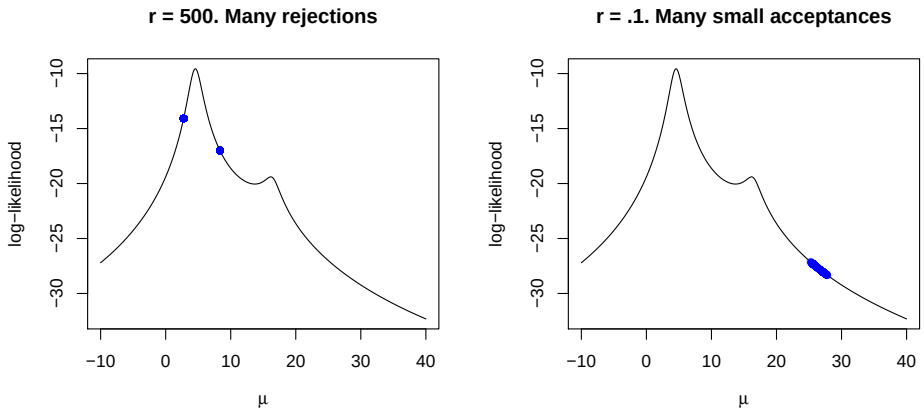


Note the simulated annealing algorithm is able to escape out of local modes and head towards the global maxima. However, the above algorithm is implemented only after tuning r. Tuning r can be challenging.

- Large r: values too far away are proposed where the objective function is very low. These values will get rejected and the algorithm will not move.
- Small r: values too close are proposed where the change in the objective function is small. These values are often accepted, but the algorithm makes very tiny jumps.

Below are runs of the simulated annealing algorithm with r chosen to be too high (500) and too low (.1).

```
par(mfrow = c(1,2))
## Different values of r
# very large r
sim <- simAn(N = 1e3, r = 500)
plot(mu.x, ll.est, type = 'l', main = "r = 500. Many rejections", ylab =
    "log-likelihood", xlab = expression(mu))
points(sim$x, log(sim$fn.value), pch = 16, col = adjustcolor("blue", alpha
    = .2))

#very small r
plot(mu.x, ll.est, type = 'l', main = "r = .1. Many small acceptances",
    ylab = "log-likelihood", xlab = expression(mu))
sim <- simAn(N = 1e3, r = .1)
points(sim$x, log(sim$fn.value), pch = 16, col = adjustcolor("blue", alpha
    = .2))
```



2 Questions to think about

- How do you think this algorithm will scale in higher dimensions? Try implementing simulated annealing for a Lasso optimization problem.
- Is there any benefit to having $T > 1$?