Grad Decent Write Up

Tomas Rivera and Emmett Forrestel

May 2024

Our earliest iterations were on the right track, as we experimented with exponentially decreasing and logarithmically decreasing functions. Unfortunately none of them got better than 5% accuracy since we were using the gradient provided to us in the contest document, which was wrong at the time. After speaking with Professor Geman at office hours, we sorted out the issue with the gradient and additionally gained some insight on the optimal functions for epsilon and T.

After that office hours, our T and epsilon took the form:

$$T = (1 - \alpha)^t$$

We began by only varying T with time, which Professor Geman referred to as annealing. We picked an arbitrary α such that $T \to 0$ as $t \to \infty$. Since we were iterating over $1,000,000$ steps, we picked $\alpha = \frac{1}{1,000,000}$. This gave us some decent results, with our best run having a median of 98.47% and a mean of 79.875%. Upon inspection of the data we were printing out, we were never able to fully isolate the metaphorical ball in the absolute minima, which makes sense since the constant ϵ meant that there would always be some random noise.

Thus, in search of some better results, we started varying epsilon and Temperature with each step.

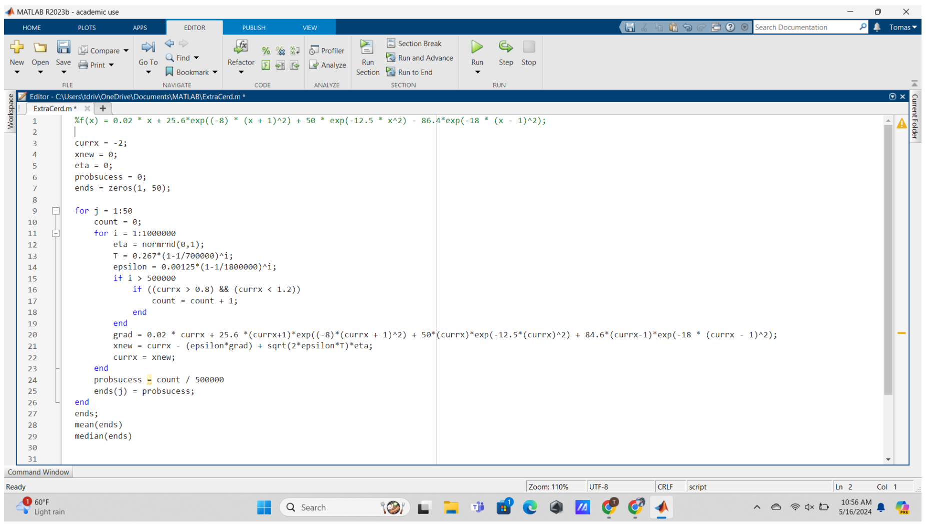
Grid Searching:

In order to compute optimal alpha values to subtract for T and epsilon we ran a 2x2 grid search varying epsilon by 1/100,000 from 1/10,000 to 1/2,500,000 while also varying T in the same manner. This left us with an alpha for T of 1/700,000 and an alpha for epsilon of 1/1,800,000.

Then with those alpha's for T and epsilon we varied what coefficient $(1-\alpha)^t$ would be multiplied by 0.001 for T from 0 to 0.5 and by 0.00005 from 0 to 0.1. Our best coefficients for T and epsilon were 0.267 and 0.00125, respectively.

In order to drastically speed up this grid search, we converted our code from MatLab into C++, and ran a multi-threaded program.

Results:



```matlab
%f(x) = 0.02 * x + 25.6*exp((-8) * (x + 1)^2) + 50 * exp(-12.5 * x^2) - 86.4*exp(-18 * (x - 1)^2);

currx = -2;
xnew = 0;
eta = 0;
probsucess = 0;
ends = zeros(1, 50);

for j = 1:50
    count = 0;
    for i = 1:1000000
        eta = normrnd(0,1);
        T = 0.267*(1-1/700000)^i;
        epsilon = 0.00125*(1-1/1800000)^i;
        if i > 500000
            if ((currx > 0.8) && (currx < 1.2))
                count = count + 1;
            end
        end
        grad = 0.02 * currx + 25.6 *(currx+1)*exp((-8)*(currx + 1)^2) + 50*(currx)*exp(-12.5*(currx)^2) + 84.6*(currx-1)*exp(-18 * (currx - 1)^2);
        xnew = currx - (epsilon*grad) + sqrt(2*epsilon*T)*eta;
        currx = xnew;
    end
    probsucess = count / 500000
    ends(j) = probsucess;
end
ends;
mean(ends)
median(ends)
```

Figure 1: Our Final Code

After our search was complete, our gradient descent always completely failed or completely succeeded. Our program always stopped just before or just after falling into the global minimum valley. We achieved 44 out of 50 perfect trials in our best run with a mean success rate of 0.88 and a median success rate of 1.0.