Homework 2

Christopher Mertin
CS6966: Theory of Machine Learning

March 25, 2017

6. (Convexity basics) For this problem, let f be a convex function defined over a convex set K, and suppose the diameter of K is 1.

   (a) Let $x \in K$, and suppose $f(x) = 2$ and $||\nabla f(x)|| = 1$. Give a lower bound on $\min_{z \in K} f(z)$.

   Solution:

   To be a convex function/set, the following has to hold true

   $$f(y) - f(x) \geq \langle y - x, \nabla f(x) \rangle$$

   We can use this to get a minimum bound on z

   $$f(z) \geq f(x) + \langle z - x, \nabla f(x) \rangle$$

   For z close to x, we can approximate $z = x + h$, giving

   $$f(z) \geq f(x) + \langle x + h - x, \nabla f(x) \rangle$$

   where we can set $h = -\eta \nabla f(x)$, giving

   $$\begin{aligned} f(z) &\geq f(x) + \langle -\eta \nabla f(x), \nabla f(x) \rangle \\ f(z) &\geq f(x) - \eta \, ||\nabla f(x)|| \end{aligned}$$

   plugging in the known values gives

   $$f(z) \geq 2 - \eta$$

   (b) Let x^* be the minimizer of f over K (suppose it is unique), and let x be any other point. The intuition behind gradient descent is that the vector: $-\nabla f(x)$ points *towards* x^*. Prove that this is indeed true, in the sense that $\langle \nabla f(x), x - x^* \rangle > 0$ (*i.e.*, the negative gradient makes an acute angle with the line to the optimum).

   Solution:

   Given the following in the above equation

$$\langle \nabla f(x), x - x^* \rangle > 0$$

If a function is convex, then we have the following is greater than or equal to the previous. As it is greater than or equal to, we can leave the greater than sign

$$f(x^*) - f(x) > 0$$

In order to minimize, we need to take the gradient, giving

$$\nabla f(x^*) - \nabla f(x) > 0$$

If x^* is the minimizer, then the gradient is zero, giving

$$-\nabla f(x) > 0$$

If f is a convex function, then any point $x \neq x^*$ has a negative slope/gradient, meaning the above holds true, thus proving the proposed statement.

(c) Suppose now that the function f is *strictly convex*, *i.e.*, $f(\lambda x + (1 - \lambda)y) < \lambda f(x) + (1 - \lambda)f(y)$ (strictly), for all $x \neq y$, and $0 < \lambda < 1$.

Prove that all the *maximizers* of f over K lie on the boundary of K.

[*Hint:* You may want to use the definition that a point x is not on the boundary iff there exists points $y, z \in K$ such that $x = (y + z)/2$]

Solution:

If the set is *strictly convex*, then no lines can be contained on the boundary. It also has to follow the following three properties:

i. The function f is twice continuously differentable everywhere in its domain
ii. $f'(x)$ is increasing on any subinterval
iii. $f''(x)$ is nonnegative on the interval and never constantly zero on any subinterval

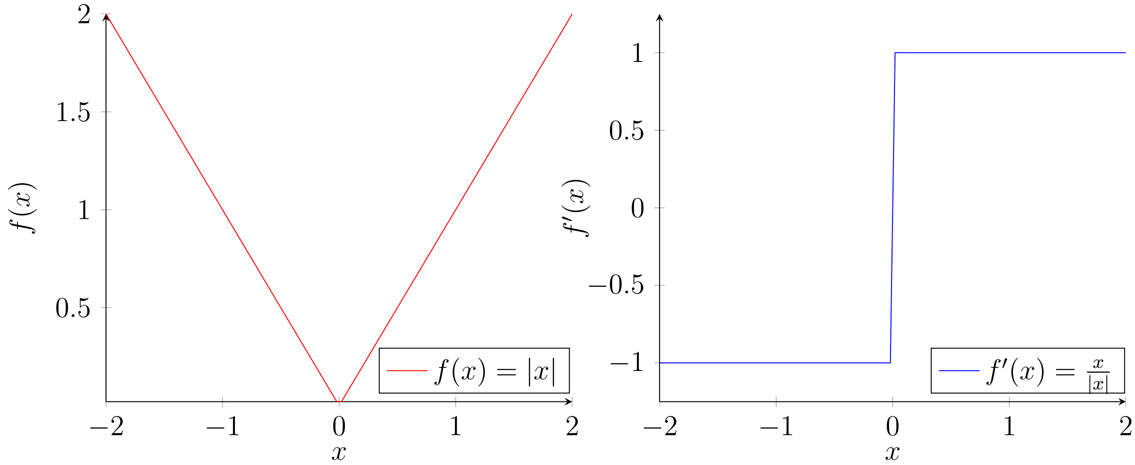
From this we can say that since the function is strictly convex you can "walk along the path" with the largest gradient. As the gradient would be positive, the value of the function would thus be increasing. You can continue to "walk" along this function until you reach the maximum point by following the gradient. As the function is *strictly convex*, you will only ever be increasing on the domain, meaning the largest point will be reached when you reach the boundary. On top of which, since no lines can be contained on the boundary, there must be a single point that is the maximum value, which *has* to be achieved by following the path of steepest ascent.

7. (Gradient Descent Basics)

(a) Give an example of a function defined over $\mathbb{R}$, for which for *any* step-size $\eta > 0$ (no matter how small), gradient descent with step size η oscillates around the optimum point (*i.e.*, never gets to distance $< \eta/4$ to it), for some starting point $x \in \mathbb{R}$.

Solution:

The example function chosen is $f(x) = |x|$ as the derivative is not defined. The plots of $f(x)$ and $f'(x)$ can be seen below.



As the above shows, any value of η will cause the function to oscillate. This is due to the fact that it will overstep the minimum as the derivative is a step function.

(b) Consider the function $f(x, y) = x^2 + y^2/4$, and suppose we run gradient descent with starting point $(1, 1)$, and $\eta = 1/4$. Do we get arbitrarily close to the minimum? Experimentally, find the *threshold* for η, beyond which gradient descent starts to oscillate.

Solution:

$$f(x, y) = x^2 + \frac{y^2}{4}$$

We need to take the gradients with respect to each variable and minimize each resulting function. In doing so gives

$$f_x(x, y) = 2x$$
$$f_y(x, y) = \frac{y}{2}$$

From the above equations, it's easy to see that they minimize at the point $(0, 0)$. We can do a few iterations of gradient descent to see how it performs. This is done below

***i* = 1**

$$f_1(1,1) = \left[1 - \frac{1}{4}\cdot 2(1),\ \frac{1}{4} - \frac{1}{4}\left(\frac{1}{2}\right)\right]$$
$$= \left[\frac{1}{2},\ \frac{1}{8}\right]$$

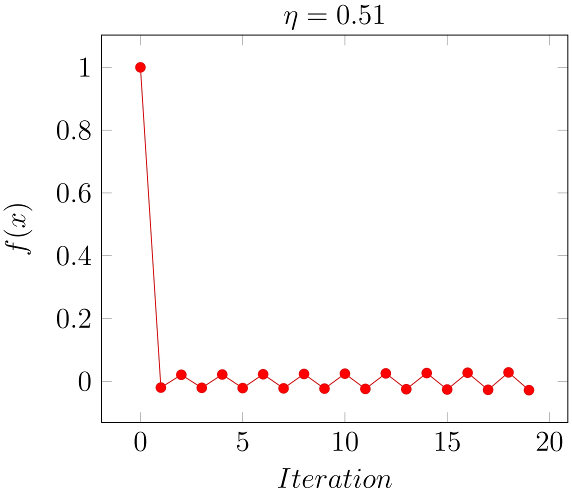
***i* = 2**

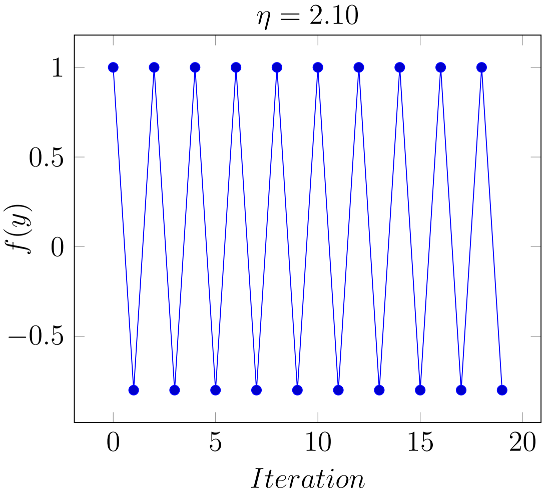
$$f_2\left(\frac{1}{2},\frac{1}{8}\right) = \left[\left(\frac{1}{2}\right)^2 - \frac{1}{4}\cancel{2}\left(\frac{1}{\cancel{2}}\right),\ \frac{(1/8)^2}{4} - \frac{1}{4}\left(\frac{1/8}{2}\right)\right]$$
$$= \left[0, -\frac{3}{256}\right]$$

***i* = 3**

$$f_3\left(0, -\frac{3}{256}\right) = \left[0,\ \frac{(-3/256)}{4} - \frac{1}{4}\left(\frac{-3/256}{2}\right)\right]$$
$$= \left[0,\ -\frac{3}{2048}\right]$$

Thus showing that it approaches the minima $(0,0)$ for $\eta = 1/4$. The bounds on η for oscillations were found, and the figures can be seen below.





If $\eta > 1/2$, then oscillations will occur in the x dimension. The threshold for η for the y dimension is $\eta > 2$. While it does oscillate for any value of $\eta > 1/2$ for x and $\eta > 2$ for y, the values of 0.51 and 2.10 were chosen such that the results were more visible and drastic.

(c) Why is the behavior similar to that in part (a) (oscillation for *every* η) not happening in part (b)?

Solution:

This is because the derivative in (a) is not defined and behaves as a *step function*, however the derivative in (b) is defined, however it is concave, and it can be easy to “overshoot” the minima if the stepsize is too large.

8. (Stochastic Gradient Descent) Suppose we have points $\{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\}$ in the plane, and suppose that $|a_i| \leq 1$, and $|b_i| \leq 1$ for all i. Let $f(x, y) = \frac{1}{n}\sum_{i=1}^{n} f_i(x, y)$, where $f_i(x, y) = (x - a_i)^2 + (y - b_i)^2$.

   (a) What is the point (x, y) that minimizes $f(x, y)$?

   Solution:

   First, we need to calculate the gradient of $f(x, y)$, giving

$$\begin{aligned} f_i(x, y) &= (x - a_i)^2 + (y - b_i)^2 \\ \nabla f_i(x, y) &= [2(x - a_i),\ 2(y - b_i)] \end{aligned}$$

   Where we can set the gradient of $f(x, y)$, rather than $f_i(x, y)$, giving

$$\nabla f(x, y) = \frac{1}{n}\sum_{i=1}^{n} 2(x - a_i) + 2(y - b_i) = 0$$

   $1/n$ can be dropped as both sides can be multiplied by n. As the two values in the summation are independent, we can separate the summation into

$$0 = \sum_{i=1}^{n} 2(x - a_i) + \sum_{i=1}^{n}(y - b_i)$$

   In the above equation, the minimizer for x will be mathematically similar to the one in y, so we can solve for just one and get the other "for free." In solving for the minimizer of x, we get

$$0 = \sum_{i=1}^{n} 2(x - a_i)$$

   Where we can split the sum into two parts, as the sum is independent on x, giving

$$\begin{aligned} 0 &= 2\sum_{i=1}^{n} x - 2\sum_{i=1}^{n} a_i \\ 2\sum_{i=1}^{n} a_i &= 2\sum_{i=1}^{n} x \\ 2\sum_{i=1}^{n} a_i &= 2xn \\ x &= \frac{1}{n}\sum_{i=1}^{n} a_i = \bar{a} \end{aligned}$$

   So we get the minimizer of $f(x, y) = (\bar{a}, \bar{b})$, where $\bar{a}$ is the average over all a values and $\bar{b}$ is the average over all b values.

(b) Suppose we perform gradient descent (on f) with step size $0 < \eta < 1$. Give a geometric interpretation for one iteration.

Solution:

The function to update the weights for gradient descent is defined as follows

$$\theta^{(i+1)} = \theta^{(i)} - \eta \nabla f(x, y)$$

Which can be rearranged as

$$\theta^{(i+1)} = \theta^{(i)} - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x, y)$$

So the "geometric interpretation" would be that, since the cost function is the *average* of the gradients for all the points, and the minimizer is the average over all the points, normal gradient descent would take "small steps" towards the average and would converge in approximately $1/\eta$ iterations, for $\eta < 1$. So for one iteration, it would move to the point $(x - \eta \cdot \bar{a}, y - \eta \cdot \bar{b})$, assuming $\bar{a},\ \bar{b} > 0$.

(c) Now suppose we perform stochastic gradient descent with fixed step size $0 < \eta < 1$, and by picking i at random in $\{1, 2, \ldots, n\}$, and moving along the gradient of f_i (as in SGD seen in class). After T steps, for T large enough, can we say that we get arbitrarily close to the optimum? (Provide a clear explanation)

[*Hint:* Remeber η is fixed]

Solution:

If we have the objective function $J(\boldsymbol{\theta}, \boldsymbol{x}_i)$ is ρ-Lipschitz, defined as

$$\left|\left|\nabla J_\theta \left(\boldsymbol{\theta}^{(t)}, \boldsymbol{x}_i\right)\right|\right| \leq \rho$$

Where we can set $\eta = \epsilon / \rho^2$, and after running SGD for $T = \frac{B^2 \rho^2}{\epsilon^2}$ iterations gives

$$E\left[\boldsymbol{\theta}^{(T)} - \boldsymbol{\theta}^*\right] \leq \epsilon = \frac{B^2}{2\eta T} + \frac{\eta}{2}\rho^2$$

If η is set to the above, and the objective function is ρ-Lipschitz, then you can guarentee convergence up to a certain value of ϵ. And as $T \to \infty$, it will converge. However, if the above does not hold, then it cannot be guarenteed to converge for a fixed value of η.

For example, in the given equation, $\nabla f(x, y)$ is minimized by the averages $(\bar{a}, \bar{b})$. In SGD, you "choose" a random point to "learn with" for each iteration in $\{T\}$. Using the given $f(x, y)$, this can be shown that fixed values of η are not guarenteed to covnerge. Consider m points with values $\{(1, 0), (0, 1), (-1, 0), (0, -1), \ldots\}$. In other words, the first 4 points are unique, and the rest are repeated. There's an even number of each point in the domain.

With a fixed value for η, the given function will be "bouncing around" as the data is shuffled and it won't be able to "learn" as it iterates over the values. However, with a decreasing value for η, it would be able to. This can be seen in the following figures.

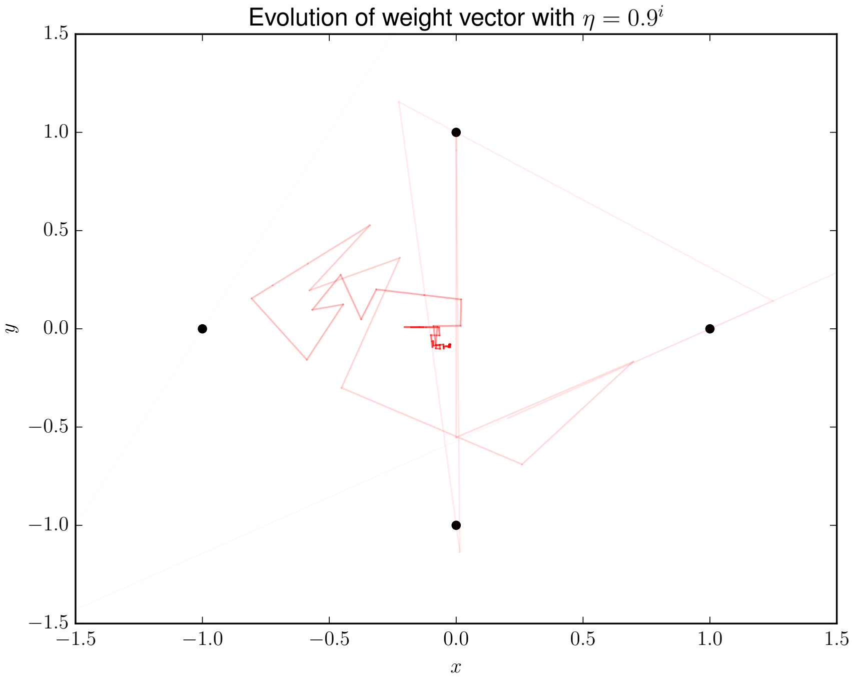


Figure 1: Decreasing value of η

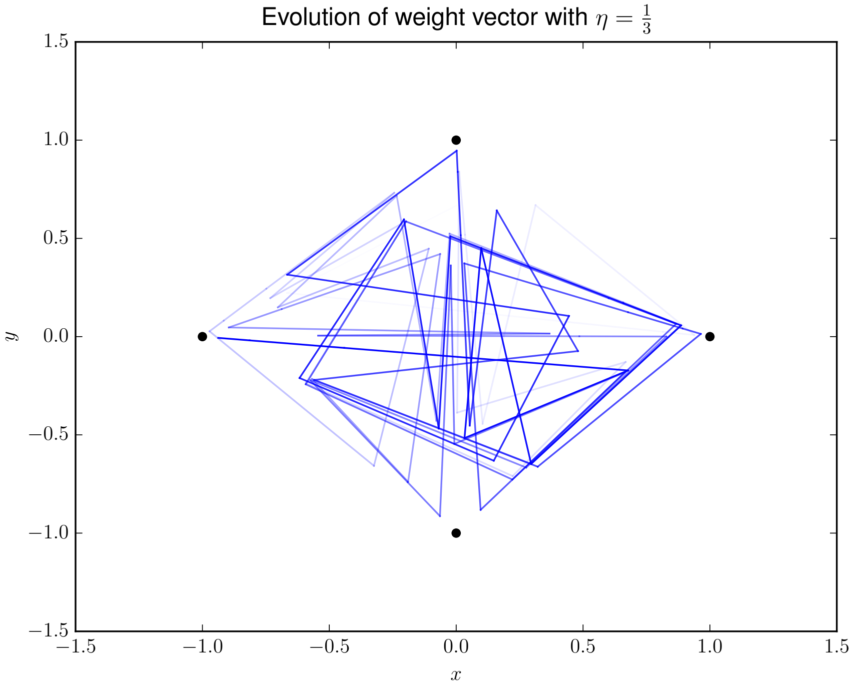


Figure 2: Fixed value of η

In the above figures, the black dots represent the data points. For the lines, the alpha channel is proportional to the iteration such that the earlier the iteration the more faint, and as the iterations progressed the alpha channel was increased. The above figures are for 20 points each for the given domain. They also used the exact same shuffled list.

As Figure 1 shows, even though η started off with a value of 0.9, with each iteration it was updated to be η^i for the i^{th} data point. In this instance, it converged rather quickly. However, with the fixed case in Figure 2, it is easily seen that it keeps diverging from one point to the next. This is due to the *randomness* of SGD with the points being shuffled, and choosing the points randomly from the dataset.

(d) Pick $n = 100$ random points in $[-1, 1]^2$ (uniformly), and run SGD for fixed $\eta = 1/2$, as above. Write down what the distance to optimum is, after $T = \{10, 100, 1000\}$ iterations (if you want to be careful, you should average over 5 random choices for the initialization). Now consider dropping the step size $\eta_t = 1/t$, and write down the result for T as above.

Solution:

The algorithm implemented for SGD can be seen below

Algorithm 1 Stoichastic Gradient Descent

```
Input: $m$ values of $\boldsymbol{x}_i \in \mathbb{R}^d$, $\boldsymbol{\theta} \in \mathbb{R}^d$, $T$
Output: $\boldsymbol{\theta}$ minimized over the data
Randomly shuffle dataset
Initialize values of $\boldsymbol{\theta}$
for $i = 1, 2, \ldots, T$ do
    $\theta_j \leftarrow \theta_j - \eta \nabla_\theta J(\boldsymbol{\theta}, \boldsymbol{x}_i)$        ▷ for every $j = \{0, 1, \ldots, \mathbb{R}^d\}$
end for
return $\boldsymbol{\theta}$
```

The following tables have the results of how far away from the minima the resulting weight vector was. It is an average taken over 5 different examples for each value of T, where the data was shuffled before each iteration.

Table 1: Distance from the minima with $\eta = 1/2$

$T = 10$	$T = 100$	$T = 1,000$
0.944	0.592	0.769

Table 2: Distance from minima with $\eta = 1/i$

$T = 10$	$T = 100$	$T = 1,000$
0.218	0.076	0.063

As the above shows, and as is to be expected, the trials with the fixed value of η were all over the place and a larger value of T did not guarentee convergence.

However, with a decaying value of η, as seen in Table 2, the values did come a lot closer to the minima, and did moreso with increased values of T.

9. (Numeric accuracy in MW updates) Consider the randomized experts setting we saw in class (we maintain a distribution over experts at each time, and the loss of the algorithm at that time is the expected loss over the distribution). Consider the simple setting where the experts predict 0/1, and the loss is either 0 or 1 for each expert. We saw how to update the probabilities (multiply by $e^{-\eta}$ if an expert makes a mistake, keep unchanged otherwise, and renormalize). One of the common problems here is that numeric errors in such computations tend to compound if not done carefully.

   Suppose we have N experts, and we start with a uniform distribution over all of them. Let $p_t^{(i)}$ denote the probability of expert i at time t, for the "True" (infinite precision) multiplicative weight algorithm, and let $q_t^{(i)}$ denote the probabilities that the "real life" algorithm uses (due to precision limitations).

   (a) Consider one simple suggestion: Say we zero out weights that are "too small," specifically, suppose we set $q_t^{(i)} = 0$ if $q_t^{(i)} / \max_j q_t^{(j)} < \epsilon$, for some precision parameter ϵ (such changes frequently occure due to roundoff). Other than this, suppose that the $q_t^{(i)}$ are updated accurately. Prove that in this case, we cannot hope to achieve any non-trivial regret bound. Specifically, for large enough T, the algorithm can have error $T(1 - \mathcal{O}(1))$, while the best expert may have error $\mathcal{O}(T)$.

   [*Hint:* In this case, we are "losing" all information about an expert]

   Solution:

   To prove this bound we can set up the following example. We know that the equation for regret is as follows

$$\text{Regret} = \text{total loss of algorithm} - \min_{i=1,\ldots,N} \sum_{t=1}^{T} \ell_{i,t}$$

   where $\ell_{i,t}$ is the loss of each expert for a given time step, so the values are $[0, 1]$. In our proof, we can assume that there is an adversary that makes it such that each prediction is wrong. As each expert is chosen with a given probability, there's no way for the adversary to know which expert is going to be chosen, but for large enough T it can be made such that *most* of the experts have been zeroed out. This can be seen in the equation to calculate the probabilities

$$p_t^{(i)} = \frac{q_t^{(i)}}{\sum_{i=1}^{N_t} q_t^{(i)}}$$

   Where N_t represents the number of experts at time t. As can be seen here, with experts being zeroed out, it allows for the adversary to force the chosen adversary to fail as each expert has a larger probability of being chosen – as there are less experts.

With a large enough T, we can bound the best expert by $\mathcal{O}\left(T/N\right)$ as more and more experts are "lost," the adversary can better predict to make the expert lose. This makes our regret bound become $R = T - T/N$ as the algorithm has $\mathcal{O}\left(T\right)$ losses (for large enough T), and the best expert has an error bound of $\mathcal{O}\left(T\right)$. For $T \gg N$, we can change the expert to have error $\mathcal{O}\left(T\right)$ and the regret bound to be $T(1 - \mathcal{O}\left(1\right))$.

(b) A simple way to avoid this (in this setting) is to avoid storing probabilities, but instead maintaining only the number of mistakes $m_t^{(i)}$. Prove how this suffices to recover the probabilities $p_t^{(i)}$ (assuming infinite precision arithmetic).

Solution:

By storing the number of mistakes, we get the following equation for the update at time t as

$$w_t^{(i)} = w_1^{(i)} e^{-\eta m_t^{(i)}}$$

Where w_1 is the initial value of the weight. By calculating it in this format, we don't impose numerical errors in the calculation. This leaves the probabiltiy as being

$$p_t^{(i)} = \frac{w_1^{(i)} e^{-\eta m_t^{(i)}}}{\sum_{i=1}^{N} w_1^{(i)} e^{-\eta m_t^{(i)}}}$$

which removes the above problem in (a) of causing experts to zero below some ϵ, for infinite precision.

(c) Suppse we use the idea in part (b) to construct a distribution q_t that differs from p_t by $< \epsilon$ in the ℓ_1 norm, *i.e.*, $\sum_i \left| p_t^{(i)} - q_t^{(i)} \right| < \epsilon$. Then, assuming we construct such a q_t at time t to sample, show that the expected number of mistakes of the algorithm is bounded by $(1+\eta)\min_i m_T^{(i)} + \mathcal{O}\left(\log(N)/\eta\right) + \epsilon T$.

Solution:

First we can calculate the *total loss* on the algorithm. Before doing so, we need to define a potential function to compare time steps. Our potential function is

$$\Phi_t = -\log\left(\sum_{i=1}^{N} w_t^{(i)}\right)$$

Using this, we can get the total loss of the algorithm by comparing the potential from $t = 1$ and $t = T$. This is done below, where $*$ is the "best performing expert," giving

$$
\begin{aligned}
\sum_{i=1}^{T} p_t^{(i)} \ell_t^{(i)} &\leq \frac{1}{1-e^{-\eta}} \left(\Phi_T - \Phi_1\right) \\
&= \frac{1}{1-e^{-\eta}} \left[\log(N) - \log\left(\sum_{i=1}^{N} e^{-\eta m_t^{(i)}}\right)\right] \\
&\leq \frac{1}{1-e^{-\eta}} \left[\log(N) - \log\left(e^{-\eta m_t^*}\right)\right] \\
&= \frac{1}{1-e^{-\eta}} \left[\log(N) + \eta m_t^*\right] \\
&\approx (1+\eta) m_t^* + \frac{\log(N)}{\eta}
\end{aligned}
$$

As we want to bound the probability distribution such that the ℓ_1 norm is $< \epsilon$, we also get a term of ϵT added in to the above loss. This comes about as being the expected error from the distribution q when compared to p, due to the fact that the best expert mistakes is bounded by $\mathcal{O}(T)$, so with changing the probabilities there are bound to be ϵT instances where another expert is chosen rather than the "best expert" as you're altering the probabilities by a range of ϵ.

(d) The bound above is not great if there is an expert who makes very small number of mistakes, say a constant (because we think of ϵ as a constant, and T as tending to infinity). Using the hint that we are dealing with binary predictions, and say by setting $\eta = 1/10$, can you come up with a way to run the algorithm, so that it uses comptuations of "word size" only $\mathcal{O}(\log(NT))$, and obtain a mistake bound of $(1+1/5)\min_i m_T^{(i)} + \mathcal{O}(\log(N))$?

[*Note*: Using word size k means that every variable used is represented using k bits; *e.g.*, the `C++` "double" uses 64 bits, and so if all probabilities are declared as doubles, you are using a word size of 64 bits.]

Solution: