Core Training: Analysis

Small dataset

In the Small dataset, every core must succeed. The probability that this will happen is the product of the success probabilities of the individual cores: $\mathbf{P_0} \times \mathbf{P_1} \times ... \times \mathbf{P_{N-1}}$. How can we assign our training units to maximize that product?

We will show that it is always best for us to spend our units on the core with the lowest success probability. Suppose that $\mathbf{P_0} < \mathbf{P_1}$, and we have some tiny amount Q of units to spend; we will denote $\mathbf{P_2} \times ... \times \mathbf{P_{N-1}}$ as OtherStuff. If we spend the Q units to improve $\mathbf{P_0}$, our success probability becomes $(\mathbf{P_0} + Q) \times \mathbf{P_1} \times$ OtherStuff. If we instead improve $\mathbf{P_1}$, the product becomes $\mathbf{P_0} \times (\mathbf{P_1} + Q) \times$ OtherStuff. Expanding those expressions, we find that they are identical, except that the first has a $Q \times \mathbf{P_1} \times$ OtherStuff term where the second has a $Q \times \mathbf{P_0} \times$ OtherStuff term. Since we know that $\mathbf{P_0} < \mathbf{P_1}$, the first value must be larger. This means that we should improve $\mathbf{P_0}$ first.

The argument above has one issue to iron out: what if increasing $\mathbf{P_0}$ causes it to rise above $\mathbf{P_1}$? By the same argument, we should switch to improving $\mathbf{P_1}$ instead at the instant that $\mathbf{P_0}$ exceeds $\mathbf{P_1}$, and vice versa if they change ranks again, and so on. So, once we have made $\mathbf{P_0}$ equal to $\mathbf{P_1}$, we should increase both of them at the same time. More generally, we should increase the smallest probability until it exactly matches the next smallest probability, then increase those at the same time until they exactly match the next smallest, and so on. We could try to simulate adding tiny units bit by bit, but it is faster to directly calculate how many units are spent at each step. We must take care to correctly handle the case in which we do not have enough units to fully complete a step.

Large dataset

In the Large dataset, **K** of the cores must succeed. Now it is no longer necessarily optimal to improve the smallest probability. For example, suppose that **N** = 2, **K** = 1, and **U** = 0.01, and the $\mathbf{P_i}$ values for the two cores are 0.99 and 0.01. If we spend all of our 0.01 units on the first core, its success probability becomes 1 and we succeed regardless of whether the second core succeeds. There is no reason to consider spending any units on the second core.

Let's extend this strategy of investing most heavily in a subset of the cores and ignoring the rest. We will sort the cores' success probabilities from lowest to highest, and focus on the ones from some index i onward. As in the Small solution, we will start by improving the success probability of the core at index i to match the success probability of the core at index i+1, then improve those two until they match the success probability of the core at index i+2, and so on. (If all of the success probabilities including and beyond index i become 1, then we can improve the core at index i-1, and so on.) We will show that, for some value of i, this is the optimal strategy. We can then try every possibility for i and keep the one that yields the largest overall answer. Notice that, for one optimal i, at most one "previous" core i-1 needs to be improved, because there is at most one i such that capacity is enough to improve cores i, i+1, ..., **N** up to probability 1 and have some left to improve i-1 but not up to 1 (otherwise, we can just choose i-1 instead).

First, let us consider whether it is better to improve core i or core i+1. Let A_i and B_i be the probability of exactly **K**-2 and **K**-1, respectively, of the cores 1, 2, ..., i - 1, i + 2, i + 3, ..., **N**succeeding. Let $P_{i,d}$ be the probability of at least **K** cores succeeding if the success probability of core i is P_i+d and the probability of success of any other core j is P_j. By replacing the definitions and cancelling out some values, we can see that $P_{i,d} - P_{i+1,d} = (A_i - B_i) \times (P_{i+1} - P_i) \times d$. Since $(P_{i+1} - P_i) \times d$ is positive, improving core i+1 is better than improving core i if and only if $B_i > A_i$. Moreover, this doesn't depend on the initial success probabilities of cores i+1 and i, but only on their relative values. So, if we improve core i+1 a little bit, it is still better to keep improving core i+1 if we can instead of switching to improve core i.

Now we want to show that there is some i_0 such that $B_i > A_i$ if and only if $i \geq i_0$. That i_0 is the core where we want to start improving. Notice that A_i depends on **N**-2 probabilities, and A_{i+1} depends on other **N**-2 cores, but **N**-3 of those overlap. Assume fixed **N**-3 core probabilities and let A(p) be the probability that exactly **K**-1 of the **N**-3 fixed cores and an additional core with probability p succeed. Define B(p) similarly. We will show that A(p + d) - B(p + d) > A(p) - B(p) for all p and d. Let U, V and W be the probabilities of having exactly **K**-3, **K**-2 and **K**-1 successes out of the fixed **N**-3 cores. Then, $A(p) = U \times p + V \times (1 - p)$ and $B(p) = V \times p + W \times (1 - p)$. Then, B(p) - A(p) is a linear function on p, which means if it ever changes from positive to negative, it must be B(0) - A(0) > 0 and B(1) - A(1) < 0, which implies W > V and V < U. However, this is impossible, because it is a well known fact in probability theory that the function f(k) defined as the number of successes in a given set of independent experiments is bitonic, so it has no local minimum at **K**-2.

With the above claim established, we can check all possible values of i and then take the largest overall success probability that we find. To compute the probability of at least K successes, we can use a dynamic programming method similar to the one described inthe analysis for last year's Red Tape Committee problem.

It is difficult to prove this method under time pressure in a contest, but we can discover it via brute force simulation, e.g., by dividing up the available units into quanta and exploring all possible ways to partition them among a small number of cores, and seeing which assignment yields the greatest chance of success. It is also possible to arrive at the correct answers via other methods such as gradient descent.