10

The Blessing and the Curse of the Multiplicative Updates

MANFRED K. WARMUTH

Online learning is a major branch in the computer science field of machine learning. On the surface, online learning seems completely unrelated to evolution, but this chapter will show that there are in fact deep analogies. Our goal is to exploit these analogies to provide new insights into both evolution and online learning.

The basic task in online learning is to process a discrete stream of data. Each piece of the data stream (example) consists of an instance and a label for the instance. The instances are vectors of a fixed dimension and the labels are real valued. Each trial processes one instance vector and its label. The algorithm first receives the instance vector. It then must produce a label for the instance vector. Typically this is done by maintaining one weight per dimension of the instance vectors and predicting the label using the dot product of the instance vector and the current weight vector. After predicting, the online learning algorithm receives the "true" label of the instance vector.

For example, the instance vector might be the predictions of five experts on whether it will rain tomorrow (where 1 stands for rain and 0 for no rain). The algorithm maintains five weights where the ith weight is the current belief that the ith expert is the best expert. The maintained weight vector is a probability vector, and the algorithm's own prediction for rain might be the weighted average (dot product) of the predictions of the experts. The true label is 1 or 0, depending whether it actually rained or not on the next day. The algorithm is constructed to predict the labels "accurately" for each instance as it arrives, and to improve accuracy as more instances accumulate.

The main issue is how to update the weights after receiving the true label for the current instance vector. One choice is to update the weights by a multiplicative update (i.e., each weight is multiplied by a nonnegative factor and then the weights are renormalized). The algebra turns out to be equivalent to the discrete replicator equation seen in earlier chapters. The weights are analogous to meme or gene strategy shares and the multiplicative update factor of each gene is analogous to its fitness.

Multiplicative updates are motivated by a Minimum Relative Entropy principle and we will argue that the simplest such update is Bayes' rule for updating priors. We will give an in vitro selection algorithm for RNA strands that implements Bayes' rule in the test tube where each RNA strand represents a different model. In one liter of the RNA soup there are approximately 10^{15} different kinds of strands and therefore this is a rather high-dimensional implementation of Bayes' rule. It is also a simple haploid evolutionary process.

In machine learning multiplicative updates are investigated for the purpose of learning online while processing a stream of data. The "blessing" of these updates is that they learn very fast in the short term because the good weights grow exponentially. However, their "curse" is that they learn too fast and wipe out other weights too quickly. This loss of variety can have a negative effect in the long term because adaptivity is required when there is a change in the process generating the data stream.

We describe a number of methods developed in machine learning that ameliorate the curse of the multiplicative updates. The methods make the algorithm robust against data that changes over time and prevent the currently good weights from taking over completely. We also discuss how the curse is circumvented by Nature. Surprisingly, some of Nature's methods parallel the ones developed in machine learning, but Nature also has some additional tricks.

Out of this research a complementary view of evolution is emerging. The short-term view deals with mutation and selection (i.e., some sort of multiplicative update). However, the long-term stability of any evolutionary process requires a mechanism for preventing the quick convergence to the currently fittest strategy or gene. We think that the online updates studied in machine learning may have something to say about Nature's evolutionary processes.

10.1 DEMONSTRATING THE BLESSING AND THE CURSE

We begin by giving a high-level definition of *multiplicative updates*. Consider algorithms that maintain a weight vector. In each trial an instance vector is processed and the weights (one per dimension) are updated. Just like strategy shares, the individual weights are non-negative and they are updated by multiplying each weight by a non-negative factor. In the most common case the weights are then rescaled to sum to one. There is a systematic way of deriving such updates by trading off a relative entropy between the new and old weight vector with a loss function that measures how well the weights did on the current instance vector (Kivinen and Warmuth, 1997). See also Appendix B of Chapter 1.

Let us look at one common scenario in more detail. We have a set of *experts* that predict something on a trial by trial basis. These experts can be human or algorithmic, and the instance vector consists of their predictions. The "master algorithm" keeps one weight, $s_i \geq 0$, per expert: s_i represents the "belief" in the ith expert at trial t. At the end of the trial these beliefs are updated as follows:

$$\tilde{s}_i = \frac{s_i \, e^{-\eta \, \text{loss}_i}}{Z}, \tag{10.1}$$

where $\eta > 0$ is a learning rate and $Z = \sum_j s_j \, e^{-\eta \, \text{loss}_j}$ normalizes the weights to 1. Here loss_i is the loss incurred by expert i in the current trial (i.e., some measure of inaccuracy) and the $\tilde{s}_i$ weights are the new or updated weights to be used in trial $t+1$.

The update of weight vector $\boldsymbol{s}$ is the solution to the following minimization problem:

$$\tilde{\boldsymbol{s}} := \operatorname*{argmin}_{\boldsymbol{r} \in \boldsymbol{R}^n : \sum_i r_i = 1} \quad \underbrace{\sum_i r_i \log \frac{r_i}{s_i}}_{\text{relative entropy between } \boldsymbol{r} \text{ and } s} \quad + \quad \eta \quad \underbrace{\sum_i r_i \, \text{loss}_i}_{\text{loss of weight vector } \boldsymbol{r}}. \tag{10.2}$$

The relative entropy term keeps the new share vector $\tilde{\boldsymbol{s}}$ close to the old share vector $\boldsymbol{s}$ and the loss term makes $\tilde{\boldsymbol{s}}$ increase the shares of minimum loss. The non-negative learning rate η is the tradeoff parameter.

The multiplicative update process is exactly the same as discrete replicator dynamics in Chapter 1, where the factor $W_i = \exp(-\eta \; \text{loss}_i)$ is the fitness of strategy i. Thus loss_i is strategy i's fitness rate scaled by minus the learning rate η. Another important connection, discussed at length in Harper (2009), is that Bayes' rule is special case of this update when the loss is the expected log loss and $\eta = 1$.

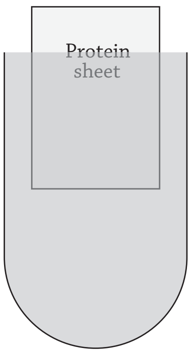
From a machine learning perspective, some basic questions would be: Why are multiplicative updates so prevalent in Nature? Why aren't additive updates more common (i.e., updates that subtract η times the gradient of the loss from the weight vector?)

As we shall soon see these multiplicative updates converge quickly (their "blessing") to the best expert (i.e., to the dominant strategy) when the data stream generating process is static. This means that all but one weight will go to zero. This is a problem when the data process changes with time. In that case the previously selected expert might perform badly on the new data, whereas one of the experts with current weight zero might predict well on the new data. Clearly wiping out the weights of all but one expert is not advantageous when the data changes over time. We call this the "curse" of the multiplicative updates. In terms of populations this means that we have a loss of variety. In the next section we will discuss a number of mechanisms developed in machine learning for preventing this loss of variety and discuss related mechanisms used in Nature.

Next we introduce in-vitro selection of RNA strands as a simple but instructive evolutionary process that connects nicely to online learning. We first show that in-vitro selection can be interpreted as an implementation of the standard Bayes' rule for updating priors to posteriors. We then use this evolutionary process to exemplify the curse of the multiplicative updates.

For example, the goal of the in-vitro selection algorithm might be the very practical problem of finding RNA strands that bind to a particular protein. Binding is a key step in drug design because the drug needs to attach to the surface of the target cell, which is identified by a surface protein. Assume we have a sheet of the target protein available, and maintain an active tube filled with dissolved RNA strands. The algorithm repeats the following three steps twenty times:

1. Dip protein sheet into the active tube.
2. Pull out and wash off RNA that stuck to surface. Discard remaining contents of the active tube.
3. Multiply washed-off RNA back to original amount and use it to fill the active tube used in the next repetition (normalization).



As a practical matter, this experiment can't be simulated with a computer because there are too many different kinds of RNA strands to track (on the order of 10^{15} strands in one liter). The basic scheme of in-vitro selection is always the following: Start with unit amount of random RNA and then in a loop, do a functional separation into "good" and "bad" RNA, and finally re-amplify the good RNA to the unit amount. The bad RNA is discarded.

The amplification in step 3 is done with the Polymerase Chain Reaction (PCR) which was invented by Kary Mullis in 1985. Roughly speaking, PCR proceeds as follows. First the single-stranded RNA is translated to double-stranded DNA. Then the following protocol is iterated for a number of steps: The substrate is heated so that the double-stranded DNA comes apart; then it is slowly cooled so that short primers can hybridize to the ends of the single-stranded DNA, and finally an enzyme (the Taq Polymerase) runs along DNA strand and complements the bases, resulting again in double-stranded DNA. Ideally in each iteration, all DNA strands are multiplied by factor of 2. After a sufficient number of iterations, the double-stranded DNA is translated back into single-stranded RNA.

We next give a mathematical description of the in-vitro selection iteration. Assume the *unknown* kinds of RNA strands in our liter of RNA strands are numbered from 1 to n where $n \approx 10^{15}$. We let $s_i \geq 0$ denote the "share" or fraction of RNA i. The contents of the entire tube are represented as a share vector $\boldsymbol{s} = (s_1, s_2, \ldots, s_n)$. When the tube has a unit amount of strands, then $s_1 + s_2 + \cdots + s_n = 1$. We let $W_i \in [0,1]$ be the fraction of one unit of RNA i that is "good" (i.e., W_i is the fitness of strand i for attaching to protein). So the target protein is represented by a second vector, the fitness vector $\boldsymbol{W} = (W_1, W_2, \ldots, W_n)$. Note that the two vectors $\boldsymbol{s}, \boldsymbol{W}$ are unknown to the researcher; clearly, there is no time to sequence all the strands and determine the fraction of attachment for each strand i. In that sense, the iterations of in-vitro selection represent some sort of "blind computation".

In the definition of the fitness vector we made a strong assumption: we assumed that the fitness W_i is independent of the share vector $\boldsymbol{s}$ (i.e., no frequency dependance). This is of course unrealistic because similar strands might compete for the same limited locations. Therefore W_i may be lower when the share s_i of strand i and the shares of similar strands s_j are high. In biological jargon, there may be some apostatic effects. However, at low concentrations they can be safely ignored, and the independence assumption might be a very good approximation.

To continue with the mathematical description, note that the amount of good RNA in tube $\boldsymbol{s}$ is $s_1 W_1 + s_2 W_2 + \cdots + s_n W_n = \boldsymbol{s} \cdot \boldsymbol{W}$. Similarly there is $s_1(1 - W_1) + s_2(1 - W_2) + \cdots + s_n(1 - W_n) = 1 - \boldsymbol{s} \cdot \boldsymbol{W}$ of bad RNA (which is discarded). During the amplification phase, the good share of RNA strand i (i.e., $s_i W_i$), is multiplied by a factor F. If the amplification is precise, then all good RNA is multiplied by the same factor F (i.e., the factor F does not vary with strand i). At the end of the iteration, the amount of RNA in the tube is again 1. Therefore

$F\,\boldsymbol{s}\cdot\boldsymbol{W}=1$ and $F=\frac{1}{\boldsymbol{s}\cdot\boldsymbol{W}}$. The entire update of the share vector in an iteration is summarized as $\tilde{s}_i=\frac{s_iW_i}{\boldsymbol{s}\cdot\boldsymbol{W}}$.

This update can be seen as an implementation of Bayes' rule: Interpret share s_i of strand i as a prior $P(i)$ for strand i. The fitness $W_i\in[0..1]$ is the probability $P(Good|i)$ and $\boldsymbol{s}\cdot\boldsymbol{W}$ the probability $P(Good)$. In summary, the multiplicative update

$$\tilde{s}_i=\frac{s_iW_i}{\boldsymbol{s}\cdot\boldsymbol{W}}\quad\text{becomes Bayes' rule:}\quad\underbrace{P(i|Good)}_{\text{posterior}}=\frac{\overbrace{P(i)}^{\text{prior}}\ \overbrace{P(Good|i)}^{\text{data likelihoods}}}{P(Good)}.$$

Note the above Bayes' rule is a special case of update (**10.1**) and the relative entropy minimization problem (**10.2**) with learning rate $\eta=1$ and $\text{loss}_i=-\ln P(Good|i)$.

In-vitro selection corresponds to iterating Bayes' rule with the same data likelihoods. Let $i^*=\text{argmax}_i\,W_i$ be the strand (or model or expert) with the highest fitness factor. This strand will gather all the weight. That is, its share s_{i^*} converges to one while the remaining shares converge to zero, as in Figure 10.1. Note that the environment is static in that the data likelihoods remain unchanged. In this case, bringing up the model with highest fitness with a pure multiplicative update (Bayes' rule) increases the average fitness very quickly. That, again, is the blessing.

The standard view of evolution concerns itself with inheritance, mutation, and selection for the fittest with some sort of multiplicative update. However, multiplicative updates are "brittle" because the gradients of the losses appear in the exponents of the update factors. This is problematic when there is noise in the data and the data-generating process changes over time. The problem is compounded when the weights can only be computed up to a constant precision, since that means that weights can go to precisely zero after a few dozen periods, and the corresponding strategies are irretrievably lost. In Nature the curse is extinction—a loss of variety at the level of genes or species. In such cases, the blessing becomes a wicked curse. Some sort of mechanism is needed to ameliorate the curse of the multiplicative update: preventing weights from becoming too large or being set to zero.

10.2 DISPELLING THE CURSE

Of necessity, both Nature and Machine Learning have developed mechanisms that, to some degree, lift or at least mitigate the curse. We begin with a mechanism

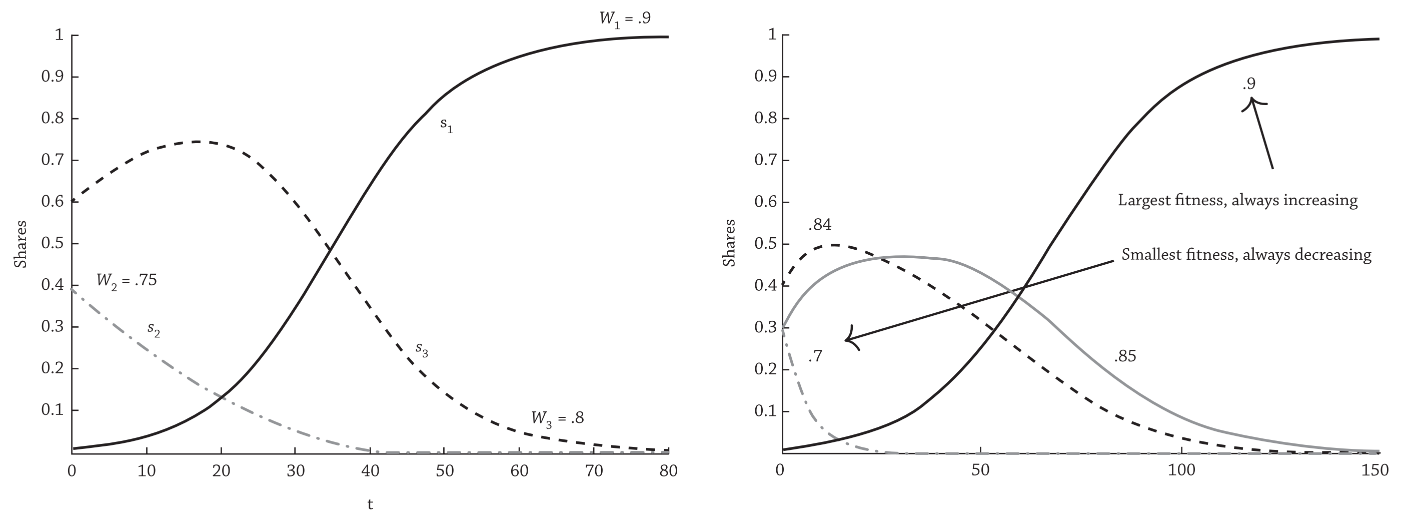


Figure 10.1 Panel A gives the shares of three models as a function of the iteration number t. The initial share vector is $\boldsymbol{s}_0 = (.01, .39, .6)$, which are the starting values of the solid, dot dashed and dashed curves (respectively) at the y-axis when $t = 0$. The fitness vector for the three curves is $\boldsymbol{W} = (.9, .75, .8)$. The shares are updated by iterating Bayes' rule: $\tilde{s}_{t,i} = \frac{s_{0,i} W_i{}^t}{\text{normalization}}$, where t is the iteration number. Panel B gives another example. In the early iterations, the share with fitness .85 (grey curve) beats the share with fitness .84 (dashed curve), but then they are both beaten by the share with highest fitness .9 (black curve), which has a very low share initially.

from Nature, which is illustrated in two experiments. In the first, fast-mutating bacteria species are added to a tube of nutrient solution (Rainey and Travisano 1998). The bacteria diversify, and after a couple of days typically three variants survive: one near the surface (in the oxygen-rich zone), one on the side wall (light-rich but little oxygen) and one in the muck on the bottom of the tube.

This is because three different environments (niches) emerged as the nutrients were used up. Competition is most intense within each niche, and the single fittest variety survives in each. Now if the same experiment is done while agitating the tube, then only one variant survives. The agitation prevents the creation of separate niches, and there is only one homogeneous niche. The competition for the best via the multiplicative update occurs within each niche and the niche boundaries protect diversity. When the boundaries are taken away (e.g., the agitated tube is unified into a single niche), then one variant quickly out-competes the rest via the multiplicative update.

In a second experiment mentioned in Chapter 7, researchers (Kerr et al. 2002) found three species of bacteria that play an RPS game. When started on a Petri dish, colonies of each species develop that slowly chase each other around the dish: R invades S's colonies, S invades P, and P invades R. If all three species are put in a tube with liquid nutrient solution that is continuously shaken, then again only one species survives. So here the local cyclical variation on the gel created by the species themselves makes it possible to maintain the variety. This can't happen in the tube because there is too much mixing. If the nutrient gel is exhausted in the dish experiment, then the local colonies can be transferred to a new dish by pressing a sterile cloth first on the old and then on the new dish. The colonies continue to chase each other on the new dish from where they started. However, if during the transfer the cloth is imprinted twice on the new dish while rotating the cloth between imprints, then this mixes the colonies enough so that only one species survives.

We attribute this first mechanism to Nature:

Nature 1: Niche boundaries help prevent the curse.

It seems that to some extent a loss of variety due to mixing also happens for memes. Humans build ever more powerful modes of connecting the world: roads, airplanes travel, the web. The "roads" allow a mixing of memes over larger areas, and the multiplicative update of the meme shares causes a loss of variety.

We now pose a problem that we claim cannot be solved by a multiplicative update alone.

Key Problem 1. There are three strands of RNA in a tube and the goal is to amplify the mixture while keeping the percentages of the strands unchanged. This is to be done without sequencing the strands and determining their concentrations.

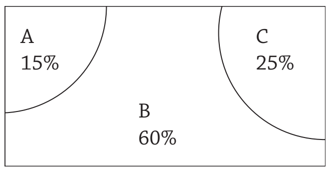


Figure 10.2 How to make more of a mixture for three strands?

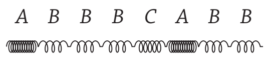


Figure 10.3 Forming long strands with approximately the right percentages.

An example is given in Figure 10.2: We want to make more of a mixture with strand *A* at 15%, *B* at 60% and *C* at 25%. If we simply translate to DNA, apply PCR once to the whole mixture, and translate back to RNA, then this ideally would amplify all three strands by a factor of 2. However in practice, the amplification will be less than 2 and slightly different for each strand. Iterative application of the multiplicative update will favor the strand that has the highest fitness for being replicated and we end up with a large amount of one of the strands.

There is a surprisingly elegant solution to this problem, using standard DNA techniques, that requires no detailed knowledge of the mixture's composition. The solution also tells us something about how to dispel the curse, so it is worth a quick sketch.

Translate to (double stranded) DNA and using an enzyme, add a specific short "end strand" to both ends of all strands in the tube. These end strands function as connectors between strands and make it possible to randomly ligate many strands together into long strands. Now separate out one long strand. With the help of an enzyme, add "primer strands" to both ends of that long strand. Apply PCR iteratively starting with the long selected strand, always making complete copies of the same original long strand that is located between the primers. Stop when you have the target amount of DNA. Now divide the long strands into their constituents by cutting them at the specific end strand that functioned as the connector. Finally, remove all short primer and end strands and convert back to RNA.

Note that for various reasons, the fractions of *A*, *B*, and *C* on the selected long strand will not agree exactly with the fractions in the original tube. However, the initial errors will not be amplified by PCR because the primers function as brackets and assure that only entire copies of the long strands get amplified. The final tube will contain the three strands at the same percentages as they appear on the initial long strand.

We just ran into the second method of Nature for preventing the curse. The long strand functions as a "chromosome." Free floating genes in the nuclei of cells would compete. Some genes would get copied multiple times during meiosis and some would die out (loss of variety). By coupling the genes into a chromosome and by assuring that the entire chromosome is copied each time, Nature prevents the curse.

Nature 2: Coupling preserves diversity.

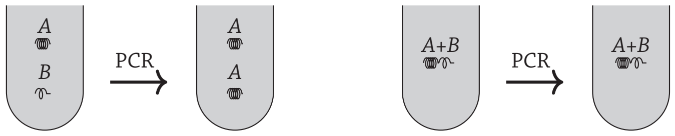
As haploids evolve, the genes on a chromosome are selected for together (i.e., there is one fitness factor per long strand). We are selecting for the best combination, and in some sense the genes must "cooperate" for the sake of the most efficient copying into the next generation, as shown in Figure 10.4.

From a Machine Learning perspective there are already several open questions. What updates to the share vector $\boldsymbol{s}$ can be implemented with in-vitro selection? What updates are possible with "blind computation" (i.e., without sequencing individual strands)? The share vectors here are concentrations and are therefore always non-negative. However, can evolutionary processes use negative weights? Negative weights allow you to express inhibitive effects of certain features, which is very useful.

In machine learning we typically process examples $(\boldsymbol{x},y)$ where $\boldsymbol{x} \in \boldsymbol{R}^n$ are the instance vectors that have the same dimension n as the weight or share vectors $\boldsymbol{s}$, and $y \in \boldsymbol{R}$ is a label. The loss associated with weight vector $\boldsymbol{s}$ on example $(\boldsymbol{x},y)$ often has the form $L(\boldsymbol{s}\cdot\boldsymbol{x},y)$. The simplest example is the square loss: $L(\boldsymbol{s}\cdot\boldsymbol{x},y)= (\boldsymbol{s}\cdot\boldsymbol{x}-y)^2$.

Now assume that you restrict yourself to multiplicative updates, which means non-negative weights. Nevertheless, researchers in machine learning circumvented the non-negative weight restriction using the $EG^{\pm}$ algorithm of Kivinen and Warmuth (1997). The algorithm maintains two weight vectors $\boldsymbol{s}^+$ and $\boldsymbol{s}^-$ of dimension n. In the simplest setting it predicts the label of the instance vector $\boldsymbol{x}$ as $(\boldsymbol{s}^+ - \boldsymbol{s}^-)\cdot\boldsymbol{x}$ and incurs the loss $L((\boldsymbol{s}^+ - \boldsymbol{s}^-)\cdot\boldsymbol{x},y) = ((\boldsymbol{s}^+ - \boldsymbol{s}^-)\cdot\boldsymbol{x} - y)^2$. There are now two weights associated with component i of the instance vectors: s_i^+ and s_i^-. The derivative of the loss for s_i^+ and s_i^- are the same except for a sign change. These derivatives appear in the exponents of the fitness factors for s_i^+ and s_i^-. Therefore, if s_i^+ has fitness W_i, then s_i^- has fitness $\frac{1}{W_i}$. So there are two versions of species i, and their fitnesses are reciprocal.

Thus the open question: Is this $EG^{\pm}$ trick used by Nature? Also can such an update be implemented with in-vitro selection?



Uncoupled: *A* and *B* compete. Coupled: *A* and *B* cooperate.

Figure 10.4 Schematic depiction of coupled versus uncoupled.

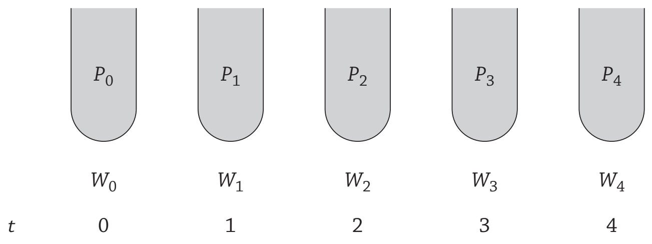
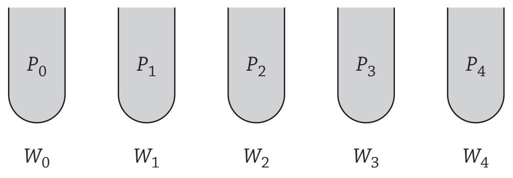


Figure 10.5 The goal is to find a set of strands that attach to $k = 5$ different proteins where each protein P_j has a different fitness vector $\boldsymbol{W}_j$.



	W_0	W_1	W_2	W_3	W_4
$W_{i,1}$	**.9**	**.8**	**.96**	.2	.04
$W_{i,2}$	.1	.01	**.8**	**.9**	**.8**
$u \cdot W_i$	.5	.405	.88	.55	.42

Figure 10.6 The fraction of attachment of strand 1, strand 2, and a tube $\boldsymbol{u}$ with 50% strand 1 and 50% strand 2. High attachment values are marked in **bold**.

We next pose a second problem that cannot be solved by a multiplicative update alone. Informally, the problem is: find a small set of RNA strands that together can bind to q different protein sheets. The previous in-vitro selection discussion assumed that $q = 1$, but $q > 1$ is a much tougher problem.

To see the difficulty, begin with the obvious approach: cycle through the q proteins and do an in-vitro selection step in turn, as in Figure 10.5. Assume protein P_j, for $j = 0, \ldots, q-1$ is characterized by fitness vector W_j. Thus in trial $t = 0, 1, \ldots$ we would do an in-vitro selection step using the fitness vector $\boldsymbol{W}_{j \bmod q}$. We assume here that no one strand exists in the tube that can bind to all the proteins and a set of strands is really needed. See, for example, Figure 10.6: Strand 1 has high attachment rate (marked in bold) on the first three proteins and strand 2 for the last three. So a tube with about 50% of strand 1 and 50% of strand 2 “covers” all five proteins in the sense that it achieves high attachment on all of them. Note that the two strands cooperatively solve the problem and this is related to the disjunction of two variables. More formally, we pose

Key Problem 2. Assume that among all strands in a 1-liter tube of RNA, there is a particular set of two strands such that for each of q proteins, at least one of the two has a high fraction of attachment. Can you use PCR to arrive at a tube

$$(\approx 0.5, \approx 0.5, \approx 0, \ldots, \approx 0)$$

which has high attachment on all q proteins?

Note that initially the two good strands have concentration roughly 10^{-15}. They are indexed by 1 and 2 in Figure 10.6. If we over-train with P_0 and P_1, then the tube will converge to $s_1 \approx 1$ and $s_2 \approx 0$. Similarly, if we over-train with P_3 and P_4, then the tube converges to $s_1 \approx 0$ and $s_2 \approx 1$. We claim this problem cannot be solved with blind computation: when the fitness vectors $\boldsymbol{W}_j$ and the initial share vector are not known, no sequencing of strands is done and the protein structure is not known. Clearly we need some kind of feedback in each trial.

Let us switch to a related machine learning problem and see what additional mechanism was used there to solve this problem. As discussed before, the tube corresponds a share vector $\boldsymbol{s}$ which is an n dimensional probability vector. Proteins correspond to example vectors $\boldsymbol{W}_j$ of the same dimension. In the machine learning setting, the components of the example vectors are binary (i.e., $\boldsymbol{W}_j \in \{0,1\}^N$). Assume there exists a probability vector $\boldsymbol{u} = (0,0,\frac{1}{k},0,0,\frac{1}{k},0,0,\frac{1}{k},0,0,0)$ with k non-zero components of value $\frac{1}{k}$ such that $\forall j : \boldsymbol{u} \cdot \boldsymbol{W}_j \geq \frac{1}{k}$. Our goal is to find a share vector $\boldsymbol{s}$ such that $\forall j : \boldsymbol{s} \cdot \boldsymbol{W}_j \geq \frac{1}{2k}$. (In Key Problem 2 and Figure 10.6, k equals 2.)

Note the we are essentially trying to "learn" a union or disjunction of k variables with a linear threshold function. Typically we are given positive and negative examples of the disjunction. In the above simplification we only have positive examples: that is, all examples need to produce a dot product that lies above the threshold of $\frac{1}{2k}$. An algorithm that achieves this is called Winnow (Littlestone 1988). In normalized form, it goes as follows.

Normalized Winnow algorithm. Let $\boldsymbol{W}_j$ be the current example and $\boldsymbol{s}$ be the current share vector. Then the updated share vector $\tilde{\boldsymbol{s}}$ is:

$$\text{If } \underbrace{\boldsymbol{s} \cdot \boldsymbol{W}_j}_{\text{good side}} \geq \frac{1}{2k} \text{then} \quad \tilde{\boldsymbol{s}} = \boldsymbol{s}, \qquad \text{"conservative update"}$$

$$\text{else} \quad \tilde{s}_i = \begin{cases} s_i & \text{if } \ W_{j,i} = 0 \\ \alpha s_i & \text{if } \ W_{j,i} = 1, \text{ where } \alpha > 1 \end{cases} \tag{10.3}$$

$$\text{and re-normalize the share vector } \tilde{\boldsymbol{s}}.$$

Note that if the good RNA $\boldsymbol{s} \cdot \boldsymbol{W}_j$ is at least as large as the threshold, then the multiplicative update is not executed. Linear threshold updates that are vacuous when the prediction is on the "right side" of the threshold are called "conservative updates". For a batch of examples, Normalized Winnow does passes over all examples until no more multiplicative update steps (10.3) occurred in the last pass.

Does Winnow solve our Problem 2? Yes it does, but it used an additional mechanism discovered in the context of Machine Learning.

Machine Learning 1: Prevent over-training by making the update conservative.

Here the curse of the multiplicative update is avoided by “conservatively” only executing the amplification step when the dot product $\boldsymbol{s} \cdot \boldsymbol{W}_j$ (i.e., the amount of “good” RNA in the functional separation) is too low. So by measuring the concentration of RNA strands on the good side (there are several ways to do this), Normalized Winnow can be implemented with blind computation.

Note that if the dot product is at least as large as the threshold $\frac{1}{2k}$, then the share vector is not updated and $\tilde{\boldsymbol{s}} = \boldsymbol{s}$. In the context of in-vitro selection this means that RNA strands that attached to the protein sheet are simply recombined into the tube without the PCR amplification step. However, when the dot product is too low, then the multiplicative update is executed and PCR is needed to do the normalization (**10.3**).

One can show that the amplification step occurs at most $O(k \log \frac{N}{k})$ times if there is a consistent k-literal disjunction and this is information theoretically optimal. More general upper bounds on the number of multiplicative update steps are proved for the case when there is no k-literal disjunction that is consistent, and when there are positive and negative examples. The crucial part of these bounds is that they grow logarithmically with the number of variables n. In in-vitro selection, n is the number of different strands (approximately 10^{15}). Nevertheless, the logarithmic dependence on the dimension n makes this a feasible algorithm.

The conservative update would not be necessary if we could make k-fold combinations of RNA strands and then simply select for the best combination of k by doing a multiplicative update *in each* trial. However this “coupling of strands” requires $\binom{n}{k}$ combinations which is too large even for moderate values of k. The brilliant insight of Winnow is that coupling (Nature’s mechanism 2) can be replaced by thresholding and only executing the multiplicative update when you are on the wrong side of the threshold.

The next mechanism we will discuss is to cap the shares/weight from above. Again this will ameliorate the curse of the multiplicative update and in some sense preserve variety. We start with Nature and then discuss how capping is used in machine learning.

In Nature capping can be achieved by a super (apex) predator. Predators need learn how to hunt any particular species. This learning process costs considerable effort and it therefore is advantageous to always specialize on the species that is currently the most frequent. In some sense the super predator nibbles away

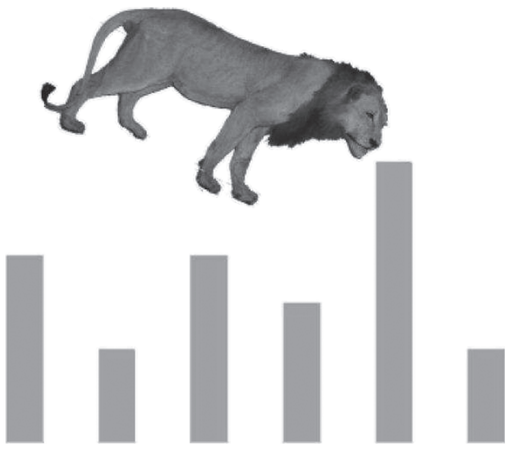


Figure 10.7 The lion pride hunts the most frequent species in the Serengeti, keeping all large grass eating species in check and preserving variety.

at the highest bar of the histogram of possible prey species (see Figure 10.7). Many examples across the animal kingdom are known where removing the super predator causes some species to take over, greatly reducing prey diversity. Sometimes disease can have a similar effect. The more frequent a species, the more opportunity for a disease to spread and this can keep a species from taking over.

Nature 3: A super predator preserves variety.

Perhaps surprising, super-predators loosely correspond to a similar mechanism of machine learning for battling the curse.

Machine Learning 2: Cap the weights from above for the purpose of learning a set of experts.

Assume the current weight vector $\boldsymbol{s}$ is a capped probability vector (i.e., its components lie in the range $[0, c]$, where $c < 1$). There will now be two update steps. The first one applies a vanilla multiplicative update to $\boldsymbol{s}$: $s_i^m = \frac{s_i\, e^{-\eta Loss_i}}{Z}$, where Z normalizes the weights to one. However, because of the normalization, the resulting intermediate weight vector $\boldsymbol{s}^m$ might have some weights larger than c. In a second capping step, all weights that exceed c are reduced to c and the total weight gained is redistributed among the remaining components that lie below c so that their ratios are preserved and the total weight still sums to one. The redistribution of the weights above c might have to be repeated a number of times. However, there are efficient implementations of this capping step. Call the resulting weight vector $\tilde{s}$. Recall that multiplicative updates are motivated

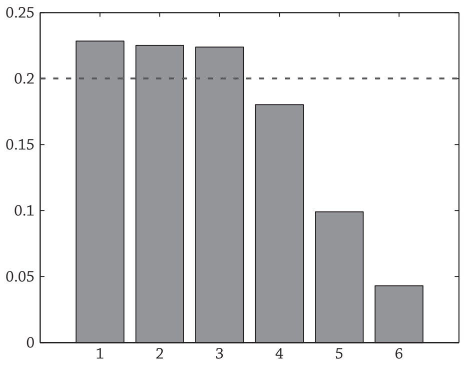


Figure 10.8 Cap and re-scale rest

by using a relative entropy as an inertia term. Similarly, the above method of capping solves a constrained optimization problem. Subject to the capping and total weight constraints, $\tilde{s}$ minimizes the weighted sum of η times the expected loss and the entropy relative to the current weight vector $\boldsymbol{s}$.

What is capping used for in machine learning? Sets of size k from our set $\{1,2,\dots,n\}$ can be encoded as n dimensional probability vectors: $(0,\frac{1}{k},0,0,\frac{1}{k},0,\frac{1}{k})$ with k components at $\frac{1}{k}$ and $n-k$ components at 0. We call these encodings of sets *k-corners*. Note that there are $\binom{n}{k}$ such k-corners. The capped probability simplex we seek is just the convex hull of the corners, $\{\boldsymbol{s} : s_i \in [0,\frac{1}{k}], \sum_i s_i = 1\}$, where we set $c = 1/k$. Since a share vector in the interior of the capped simplex is a convex combination of k-corners, the online algorithm maintains its uncertainty over which set is best by such mixtures.

The curse of the standard multiplicative update is that the weight vector converges to a corner of the simplex, a vector $(0,\dots,0,1,0,\dots,0)$ where the 1 is the position of the best expert. If, however, we run the multiplicative update together with capping at $c = \frac{1}{k}$, then the combined update converges to a k-corner of the capped simplex (i.e., to the best set of k experts). Figure 10.9 depicts the 3- and 4-dimensional simplexes plus their capped versions with $c = \frac{1}{2}$.

Note that the n-dimensional probability simplex has n 1-corners which are the n unit vectors and the capped n-dimensional simplex with $c = \frac{1}{k}$ has $\binom{n}{k}$ different k-corners, an integer much larger than n when $k \geq 2$. The combined update is computationally efficient in that it essentially lets us learn a k-variety with only n weights instead of $\binom{n}{k}$ weights. In the machine learning applications, using n weights is tolerable, but using $\binom{n}{k}$ weights is prohibitive.

An extension of this reasoning may interest technically minded readers. There is the matrix version of the multiplicative update: share vectors are generalized to density matrices which can be viewed as a probability vector of eigenvalues plus the corresponding unit eigenvectors. The "generalized multiplicative update"

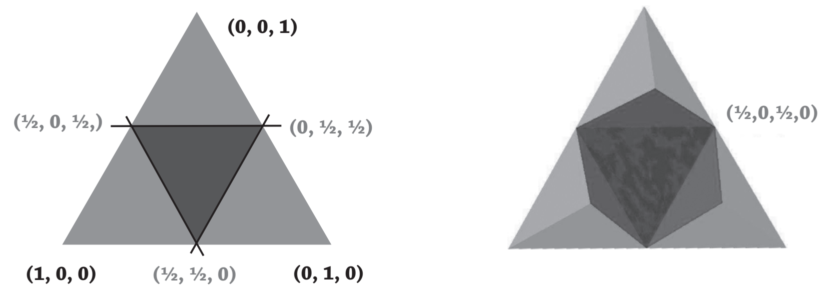


Figure 10.9 The 3-simplex can be depicted as a triangle where the three corners correspond to the single experts. The 2-corners of the capped simplex with $c = \frac{1}{2}$ correspond to the $\binom{3}{2}$ = 3 different pairs of experts from a panel of three experts. The 4-simplex is a tetrahedron and the capped simplex with $c = \frac{1}{2}$ has $\binom{4}{2}$= 6 different 2-corners corresponding to the six different pairs of experts chosen from four candidates.

involves the computation of matrix logs and matrix exponentials (Tsuda et al. 2005). These updates are motivated with the quantum relative entropy instead of the regular relative entropy; which is the core concept in coding theory. When the matrix version of the multiplicative update is combined with a second capping update on the eigenvalues of the density matrix, then we arrive at an online algorithm for principal component analysis: the sets of k experts generalize to a k-dimensional subspace and capped density matrices are convex combinations of k-dimensional subspaces.

Many questions arise in this context: Is there a way to implement capping in the in-vitro selection setting by somehow modifying PCR? How does Nature avoid the $\binom{n}{k}$ combinatorial blowup when it needs to learn a conjunction or disjunction of genes? Is the matrix version of the multiplicative update used somewhere in Nature?

There is a related mechanism for battling the curse that we still need to discuss: lower bounding the shares. After doing a number of iterations of in-vitro selection, some “selfish” strand often manages to dominate without exactly achieving the wanted function. A standard trick is keep a batch of the initial mixture in reserve (which contains the initial variety of approximately 10^{15} different strands). Whenever the current mixture has become too uniform, then a little bit of the initial rich mixture is mixed into the current tube. This essentially amounts to lower bounding the shares.

A good illustration of this mechanism arises from a practical application of online learning: the *disk spindown problem*. It takes energy to shut down and restart a laptop. So if the laptop is likely to be used again very soon, then keeping it running is the best strategy for conserving energy. However, if it is likely to be idle for a long time, then it is better to shut it down immediately. In practice this dilemma is resolved with a fixed timeout: If there was no action within a fixed time τ, then the laptop simply shuts down and powers back up when the next request comes in.

A more clever way is to find a timeout that adapts to the usage pattern of the user by running a multiplicative update on a set of timeouts (Helmbold et al. 2000). So for this setting a suitably spaced set of n fixed timeouts $\tau_1, \tau_2, \ldots, \tau_n$ form our set of *experts*. As before, the *master algorithm* maintains a set of weights or shares $s_1, s_2, \ldots, s_n$ for the experts and applies a multiplicative update after each idle time:

$$\tilde{s}_i = \frac{s_i\, \boldsymbol{e}^{-\eta \text{ energy usage of timeout } i}}{Z}, \quad \text{where } Z \text{ normalizes the shares to } 1.$$

Ideally the shares of the master algorithm will concentrate on the best timeout.

The curse strikes when the data (characterized by the typical length of the idle times) change over time, as in Figure 10.10. Roughly, each typical length of the idle time favors a certain timeout value. The multiplicative update will quickly bring up the shares of the timeout values that use the least energy for the current typical idle time length and wipe out the shares of the remaining timeouts. However, this typical length will occasionally change, for example, due to a shift in workload. Unfortunately the newly needed timeouts are no longer available.

There is a simple mechanism that prevents this problem with a second update step:

Machine Learning 3: Mix in a little bit of the uniform vector.

$$\begin{aligned} \boldsymbol{s}^m &= \frac{s_i\, \boldsymbol{e}^{-\eta \text{ energy usage of timeout } i}}{Z} \\ \tilde{\boldsymbol{s}} &= (1-\alpha)\boldsymbol{s}^m + \alpha \left(\frac{1}{N}, \frac{1}{N}, \ldots, \frac{1}{N}\right), \text{ where } \alpha > 0 \text{ is small.} \qquad \textbf{(10.4)} \end{aligned}$$

In Nature we have a similar situation. The multiplicative update quickly selects for the individuals most fit for the current environment. However, those individuals might not be adapted well for the environment of tomorrow and a mechanism is needed for keeping genes around that might become useful later. At a rough level, mutations function as a mechanism for lower bounding the shares.

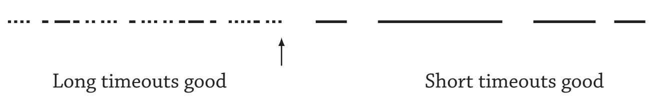


Figure 10.10 Idle times are depicted as black line segments. Bursts of short idle times (before $\uparrow$) favor long timeouts τ_i. For long idle times (after $\uparrow$) small timeouts are better. However, at time $\uparrow$, the multiplicative update has wiped out all short timeouts.

Nature 4: Mutations keep a base variety.

We conclude with the most interesting mechanism discovered in Machine Learning for battling the curse. This mechanism works well when the data shifts once in a while, and some of the shifts are returns back to a type of data seen previously. Natural data often has this form: It shifts between a small number of different "modes": a breakfast mode, a lunch mode, and a dinner mode, and once in a while a holiday mode.

A good online update has to do two things: First, it must have the capability to bring up shares quickly, and second, it must remember experts that did well in the past because they might be needed again. The first is achieved with a multiplicative update which has good-short term properties (i.e., the shares of the good experts are brought up quickly). The mechanism needed for the second part has been dubbed "sleeping" (Adamskiy et al. 2012) because previously good models are essentially put asleep so that they can be woken up quickly when needed again.

Machine Learning 4: Use sleeping to realize a long-term memory.

A simple way to do this is to keep track of the average share vector $\boldsymbol{r}$. Instead of mixing in an α fraction of the uniform weight vector, as done in the second step of (**10.4**), mix in an α fraction of $\boldsymbol{r}$. Finally, at the end of each trial, update the average vector $\boldsymbol{r}$ so that it includes the current share vector. Note that an expert that did well in the past will have a large enough share in the average share vector $\boldsymbol{r}$ and this helps in the recovery when it is needed again.

This two-step update can be implemented within in-vitro selection paradigm (i.e., with blind computation): Maintain two tubes, one for the current share vector $\boldsymbol{s}$ and one for the average share vector $\boldsymbol{r}$. It is easy to implement the $\tilde{\boldsymbol{s}} = (1-\alpha)\boldsymbol{s} + \alpha\boldsymbol{r}$ update by mixing the appropriate fractions of the tube. Similarly, one can update of the average tube at the end of trial t by mixing tubes: $\tilde{\boldsymbol{r}} = \frac{t-1}{t}\boldsymbol{r} + \frac{1}{t}\tilde{\boldsymbol{s}}$. However the totals of the tubes are not preserved in this two-step update: For example, an α fraction of the $\boldsymbol{r}$ tube is moved to the $\boldsymbol{s}$ tube, and now there is less than unit amount in the $\boldsymbol{r}$ tube.

This can be fixed with additional PCR amplification steps, but there is a more elegant way to implement sleeping with in-vitro selection, as described in Figure 10.11. It is essentially a Markov network with two tracks (tubes). The left (awake) tube represents the share vector $\boldsymbol{s}$ and the right (asleep) tube the share vector $\boldsymbol{r}$. Both are initialized to be uniform, with amounts γ and $1-\gamma$, respectively. In each trial, $\boldsymbol{s}$ is updated to $\boldsymbol{s}^m$ via a multiplicative update. The normalization assures that $\boldsymbol{s}$ and $\boldsymbol{s}^m$ have the same total weight. On the asleep side, nothing happens (i.e., $\boldsymbol{r}^m = \boldsymbol{r}$). In the second step, there is an exchange: both tubes send a fraction of

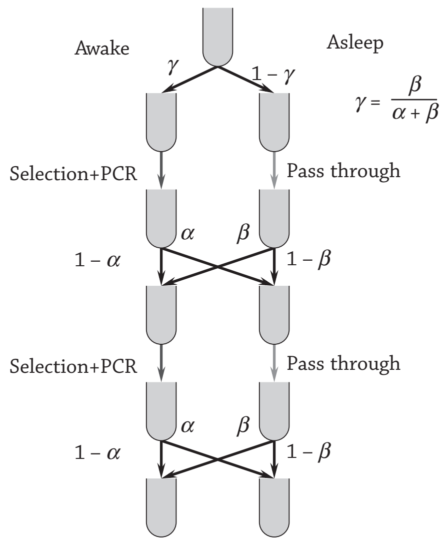


Figure 10.11 2-track method: We start with a unit amount of a rich mixture of RNA strands. A fraction of γ goes to the awake side and $1-\gamma$ to the asleep side. The awake tube participates in the selection process and the asleep tube is just passed through. After that the tubes exchange a small fraction of their content: α from awake to asleep and β from asleep to awake.

their content to the other side:

$$\begin{aligned}\tilde{\boldsymbol{s}} &= (1-\alpha)\boldsymbol{s}^m + \beta\boldsymbol{r}^m \\ \tilde{\boldsymbol{r}} &= \alpha\boldsymbol{s}^m + (1-\beta)\boldsymbol{r}^m,\end{aligned}$$

where the exchange probabilities α and β are small numbers that need to be tuned. If the initial fraction γ of the awake side is $\frac{\beta}{\alpha+\beta}$, then the exchange keeps the fraction on the awake side at γ and the asleep side at $1-\gamma$.

A sample run of this algorithm and some of its competitors is given in Figure 10.12. The repeating segments may be seen as "modes" of the data: There is an A-mode, a D-mode, and a G-mode. Note that the two-track algorithm recovers more quickly in segments where a previously best expert is best again, that is, we return to a previously seen mode (see Figure 10.13). In Figure 10.14 we see that all previously good experts are remembered on the sleeping side (long-term memory).

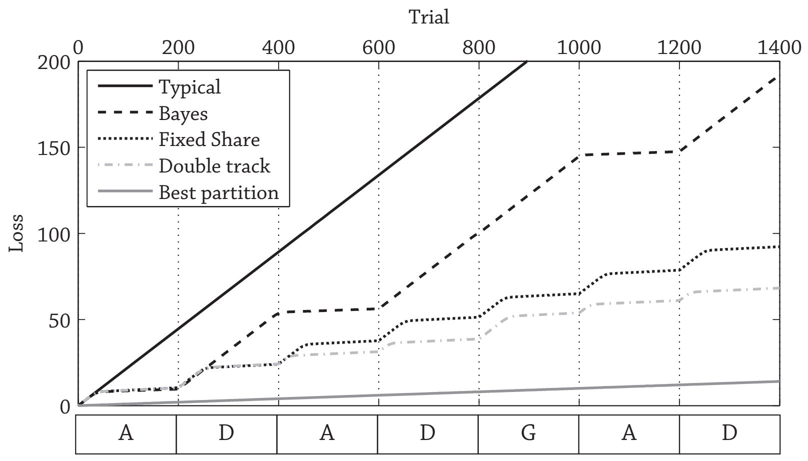


Figure 10.12 We plot the total loss of a number of algorithms when the data shifts recurrently: 1400 trials, 2000 experts, the data shifts every 200 trials. The best experts in the seven segments are A, D, A, D, G, A, D. The grey "Best partition" curve is the total loss with foreknowledge, using the best expert immediately in each segment. The loss of all other experts increase as shown in the black "Typical" curve. The dashed "Bayes" curve is just the multiplicative update: It learns expert A in the first segment and does not adjust in the later segments. However, it has the optimal slope in all later segments when A is best again. The dotted "Fixed Share" update (10.4) mixes in a bit of the uniform distribution in each trial. It learns the best expert in each segment (fixed size bump for each segment). The dash dotted "two-track" algorithm of Figure 10.11 has a long-term memory (i.e., it recovers quicker in segments where the best expert is an expert that was best in a previous segment).

Curiously enough, there is a Bayesian interpretation of the above two methods of mixing in a little bit of the average share vector or the two-track method of Figure 10.11 (Adamskiy et al. 2012): Models are either awake or asleep. For a suitably chosen set of models and prior of the models the two methods can be explained with a Bayesian update with the following caveat: the asleep models predict with the Bayesian predictive distribution (the normalizer of the Bayesian update). This means that Bayes' update for asleep models is vacuous (i.e., the prior equals the posterior). In other words, the asleep models "abstain" while being asleep and their prior is unchanged while they are asleep.

10.3 DISCUSSION

Multiplicative updates converge quickly, which is their blessing. However, multiplicative updates wipe out diversity, which is their curse. Changing conditions

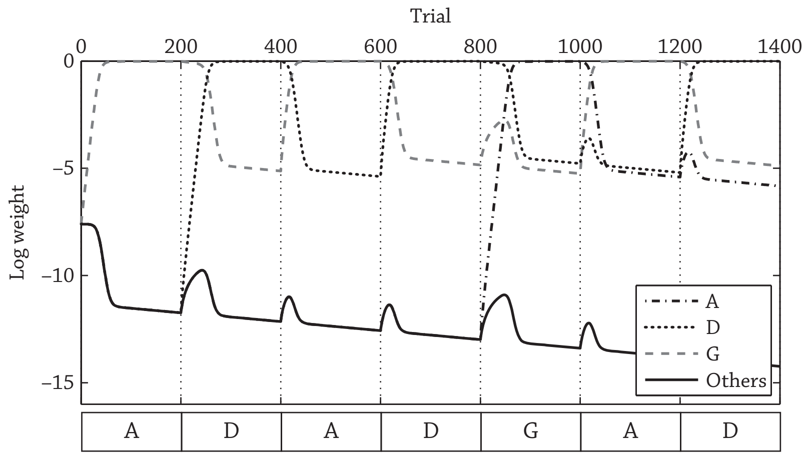


Figure 10.13 Here we plot the log weights of the *awake side* of the "two-track" algorithm. Note that the weights of the best experts are brought up quickly at the beginning of each segment. Most importantly, when an expert is best again, then it is brought up more quickly.

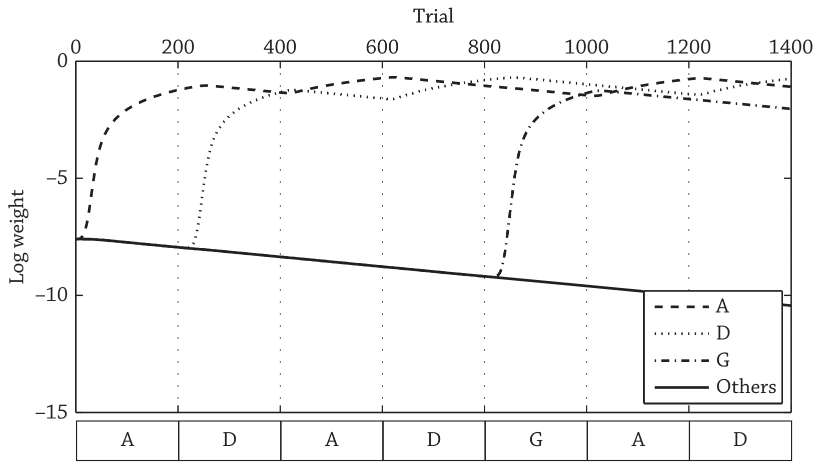


Figure 10.14 Here we plot the log weights of the asleep side of the "two-track" algorithm. Note that the weights of previously good experts decay only slowly. At the end (trial 1400) the weights of the experts A, D, and G are elevated (long-term memory).

require reuse of previously learned knowledge/alternatives and diversity is a requirement for success.

A mechanism is needed to ameliorate the curse and machine learning and Nature each have developed a number of them.

Machine Learning mechanisms:

1)	conservative update	for learning multiple goals
2)	upper bounding weights	for learning multiple goals
3)	lower bounding weights	for robustness against change
4)	sleeping	for realizing a long-term memory.

Nature’s mechanisms:

1)	coupling	for preserving variety
2)	boundaries	for preserving variety
3)	super-predators	for preserving variety
4)	mutations	for keeping a base variety.

We are convinced that studying the long-term stability of evolutionary processes is an important research topic and we believe that important insights can be gained by contrasting Nature’s stabilizing mechanisms with the ones developed in machine learning for multiplicative updates.

Regarding the sleeping mechanism, it seems that the two-track algorithm models evolution at different scales. The awake track represents the short-term evolution and the asleep track the long-term evolution. An interesting question is whether a third track would be useful. It is reasonable to expect that Nature makes use of the sleeping mechanism as well. At a macro level if you sow a handful of wild seed, then not all sprout immediately. Some lay dormant, and this guards against changing weather conditions. However, the deep question is how is sleeping realized at the genetic level. Does junk DNA or sex play a role?

NOTES

We use the term “multiplicative updates” specifically for updating the weights or shares by non-negative scalar factors. Note that the mutation update on a population represented by a share vector is described by a different type of multiplication: In this case the share vector is multiplied with a stochastic matrix.

Some early work on online learning can be found in Vovk (1990); Littlestone and Warmuth (1989, 1994). For the motivation of multiplicative updates that uses the relative entropy as a divergence measure see Kivinen and Warmuth (1997). A survey of in-vitro selection is given in S. Wilson and Szostak (1999).

For the observation that Bayes' rule is related to the discrete replicator equation see, for example, Harper (2009).

The relationship between various disturbance regimes and diversity has been discussed extensively in the literature. The "intermediate disturbance hypothesis" proposes that diversity will be highest at intermediate levels of disturbance (see Miller et al., 2011 for an overview).

Problems related to the replication of a mixture of strands while preserving their relative frequencies (Key Problem 1) are often discussed in the context of the evolution of early life (see, for example, Zintzaras et al. 2002; Vasas et al. 2012).

The original Winnow algorithm appeared in Littlestone (1988). The normalized version presented here was analyzed in Helmbold et al. (2002). The stabilizing effect of a super/apex predator has been well studied. See for example J. A. Estes et al. 2011 and F. Sergio et al. 2008 and the references therein, as well as the predator/prey model on the prism developed in Chapter 7.

The multiplicative updates together with capping and their application to learning principal components online were developed in Warmuth and Kuzmin (2008). The optimality of these algorithms has been shown in Jiazhong et al. (2013). The matrix versions of the multiplicative updates were developed in Tsuda et al. (2005) and Warmuth and Kuzmin (2006, 2011).

The method of making an online algorithm robust against time changing data by mixing in a bit of the uniform distribution was first analyzed in Herbster and Warmuth (1998). The idea of mixing in a bit of the average share vector was introduced in Bousquet and Warmuth (2002). The long-term memory properties of this method were also analyzed in that paper.

Finally, the Markov type two-track update given in Figure 10.11 was introduced in Adamskiy et al. (2012). Also, a Bayesian interpretation of the long-term memory updates based on the mechanism of sleeping was given in that paper.

BIBLIOGRAPHY

Adamskiy, D., M. K. Warmuth, and W. M. Koolen. "Putting Bayes to sleep." In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds., *Advances in Neural Information Processing Systems 25 (NIPS '12)*, pp. 135–143. Curran Associates, Inc., 2012.

Bousquet, O., and M. K. Warmuth. "Tracking a small set of experts by mixing past posteriors." *Journal of Machine Learning Research* 3 (2002): 363–396.

Estes, J. A. et al. "Trophic downgrading of planet earth." *Science* 333, no. 6040 (2011): 301–306.

Harper, M. "The replicator equation as an inference dynamic." arXiv:0911.1763 [math. DS], November 2009.

Helmbold, D. P., D. D. E. Long, T. L. Sconyers, and B. Sherrod. "Adaptive disk spin-down for mobile computers." *ACM/Baltzer Mobile Networks and Applications (MONET)*, pp. 285–297, 2000.

Helmbold, D. P., S. Panizza, and M. K. Warmuth. "Direct and indirect algorithms for on-line learning of disjunctions." *Theoretical Computer Science* 284, no. 1 (2002): 109–142.

Herbster, M., and M. K. Warmuth. "Tracking the best expert." *Machine Learning* 32 (1998): 151–178.

Jiazhong, N., W. Kotłowski, and M. W. Warmuth. "On-line PCA with optimal regrets." In *Proceedings of the 11th International Conference on Algorithmic Learning Theory (ALT 24)* 8139 (2013): 98–112. Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin.

Kerr, B., Margaret A. Riley, Marcus W. Feldman, and Brendan J. M. Bohannan. "Local dispersal promotes biodiversity in a real-life game of rock-paper-scissors." *Letters to Nature* 418 (2002): 171–174.

Kivinen, J., and M. K. Warmuth. "Additive versus exponentiated gradient updates for linear prediction." *Information and Computation* 132, no. 1 (1997): 1–64.

Littlestone, N. "Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm." *Machine Learning* 2, no. 4 (1988): 285–318.

Littlestone, N., and M. K. Warmuth. "The weighted majority algorithm." In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, 1989.

Littlestone, N., and M. K. Warmuth. "The weighted majority algorithm." *Information and Computation* 108, no. 2 (1994): 212–261.

Miller, A. D., S. H. Roxburgh, and K. Shea. "How frequency and intensity shape diversity-disturbance relationships." *Proceedings of the National Academy of Sciences USA* 108, no. 14 (2011).

Rainey, P. B., and M. Travisano. "Adaptive radiation in a heterogeneous environment." *Nature* 394 (1998): 69–72.

Sergio, F. et al. "Top predators as conservation tools: Ecological rationale, assumptions, and efficacy." *Annual Review of Ecology, Evolution, and Systematics* 39 (2008): 1–19.

Tsuda, K., G. Rätsch, and M. K. Warmuth. "Matrix exponentiated gradient updates for on-line learning and Bregman projections." *Journal of Machine Learning Research* 6 (2005): 995–1018.

Vasas, V., C. Fernando, M. Santos, S. Kauffman, and E. Szathmary. "Evolution before genes." *Journal of Biology Direct* 7, no. 1 (2012): 1–14.

Vovk, V. "Aggregating strategies." In *Proceedings of the Third Annual Workshop on Computational Learning Theory* (1990): 371–383.. Morgan Kaufmann.

Warmuth, M. K., and D. Kuzmin. "Online variance minimization." In *Proceedings of the 19th Annual Conference on Learning Theory (COLT '06)* (2006). Pittsburg, Springer-Verlag.

Warmuth, M. K., and D. Kuzmin. "Randomized PCA algorithms with regret bounds that are logarithmic in the dimension." *Journal of Machine Learning Research* 9 (2008): 2217–2250.

Warmuth, M. K., and D. Kuzmin. "Online variance minimization." *Journal of Machine Learning* 87, no. 1 (2011): 1–32.

Wilson, D. S., and J. W. Szostak. "In vitro selection of functional nucleic acids." *Annual Review of Biochemistry* 68 (1999): 611–647.

Zintzaras, E., M. Santos, and E. Szathmary. "'Living' under the challenge of information decay: The stochastic corrector model vs. hypercycles." *Journal of Theoretical Biology* 217, no. 2 (2002): 167–181.