

Learning permutations with exponential weights

David P. Helmbold and Manfred K. Warmuth*

Computer Science Department
University of California, Santa Cruz
{dph | manfred}@cse.ucsc.edu

Abstract. We give an algorithm for learning a permutation on-line. The algorithm maintains its uncertainty about the target permutation as a doubly stochastic matrix. This matrix is updated by multiplying the current matrix entries by exponential factors. These factors destroy the doubly stochastic property of the matrix and an iterative procedure is needed to re-normalize the rows and columns. Even though the result of the normalization procedure does not have a closed form, we can still bound the additional loss of our algorithm over the loss of the best permutation chosen in hindsight.

1 Introduction

Finding a good permutation is a key aspect of many problems such as the ranking of search results or matching workers to tasks. In this paper we present an efficient and effective algorithm for learning permutations in the on-line setting called PermeLearn. In each trial, the algorithm probabilistically chooses a permutation and incurs a loss based on how appropriate the permutation was for that trial. The goal of the algorithm is to have low total expected loss compared to the best permutation chosen in hindsight for the whole sequence of trials.

Since there are $n!$ permutations on n elements, it is infeasible to simply treat each permutation as an expert and apply Weighted Majority or another expert algorithm. Our solution is to implicitly represent the weights of the $n!$ permutations with a more concise representation. This approach has been successful before where exponentially many experts have a suitable combinatorial structure (see e.g. [HS97, TW03, WK06] for examples).

We encode a permutation of n elements as an $n \times n$ permutation matrix Π : $\Pi_{i,j} = 1$ if the permutation maps i to j and $\Pi_{i,j} = 0$ otherwise. The uncertainty of the algorithm about which permutation is the target is represented as a mixture of permutation matrices, i.e. a doubly stochastic weight matrix¹ W . The entry $W_{i,j}$ represents the algorithm's belief that the target permutation maps element i to position j . Every doubly stochastic matrix is the convex combination of at most $n^2 - 2n + 2$ permutations (see e.g. [Bha97]). We present a simple

* Manfred K. Warmuth acknowledges the support of NSF grant CCR 9821087.

¹ Recall that a doubly stochastic matrix has non-negative entries and the property that every row and column sums to 1.

matching-based algorithm that efficiently decomposes the weight matrix into a slightly larger than necessary convex combination. Our algorithm PermELearn samples a permutation from this convex combination to produce its prediction.

As the algorithm is randomly selecting its permutation, an adversary simultaneously selects a loss matrix $L \in [0, 1]^{n \times n}$ for the trial. The adversary is allowed to see the algorithm’s doubly stochastic matrix, but not its random choice of permutation. The loss matrix has the interpretation that $L_{i,j}$ is the loss for mapping element i to j and the loss of a whole permutation is the sum of the losses of the permutation’s mappings. This linear decomposition of the loss is necessary to make the algorithm efficient and fits nicely with the algorithm’s weight matrix representation. Section 3 shows how a variety of intuitive loss motifs can be expressed in this matrix form.

Before the next trial, algorithm PermELearn makes a weighted-majority style update to its weight matrix: each entry $W_{i,j}$ is multiplied by $e^{-\eta L_{i,j}}$ where η is a learning rate in $[0, 1]$. After this update, the weight matrix no longer has the doubly stochastic property, and the weight matrix must be projected back into the space of doubly stochastic matrices (called “Sinkhorn Balancing”, see Section 4) before the next prediction can be made. Our method based on Sinkhorn Balancing bypasses a potential issue: if the probability of each permutation Π is proportional to $\prod_i W_{i,\Pi(i)}$ then the normalization constant is the permanent of W , and calculating the permanent is a known #P-complete problem.

We bound (Theorem 1) the expected loss of PermELearn over any sequence of trials by

$$\frac{n \ln n + \eta \mathcal{L}_{\text{best}}}{1 - e^{-\eta}}, \tag{1}$$

where η is the learning rate and $\mathcal{L}_{\text{best}}$ is the loss of the best permutation on the entire sequence. If an upper bound $\mathcal{L}_{\text{est}} \geq \mathcal{L}_{\text{best}}$ is known, then η can be tuned (as in [FS97]) and the bound becomes

$$\mathcal{L}_{\text{best}} + \sqrt{2\mathcal{L}_{\text{est}}n \ln n} + n \ln n. \tag{2}$$

Since $\ln n! \approx n \ln n - n$, this is close to the loss bound when Weighted Majority is run over the $n!$ permutations. We also can show (omitted) a lower bound of $\mathcal{L}_{\text{best}} + \Omega(\sqrt{\mathcal{L}_{\text{best}}n \ln n} + n \ln n)$.

The Exponentiated Gradient family of learning algorithms uses probability vectors as their weight vectors. In this case the normalization is straightforward and is folded directly into the update. PermELearn’s hypothesis is a doubly stochastic matrix. Its update step first multiplies the entries of the matrix by exponential factors and then uses Sinkhorn re-balancing to iteratively re-normalize the rows and columns. We are able to prove bounds for our algorithm despite the fact the re-normalization does not have a closed form solution. We show that the multiplicative update minimizes a tradeoff between the loss and a relative entropy between non-negative matrices. This multiplicative update takes the matrix outside of the set of doubly stochastic matrices. Luckily this un-normalized update already makes enough progress (towards the best permutation) for the loss bound quoted above. We interpret the iterations of Sinkhorn balancing as

projections w.r.t. the same relative entropy. Finally, using Bregman projection methods one can show these projections only increase the progress and thus don't hurt the analysis.

Our new insight of splitting the update into an un-normalized step followed by a normalization step also leads to a streamlined proof of the loss bound of the randomized weighted majority algorithm that is interesting in its own right. On the other hand, the bounds for the expert algorithms can be proven in many different ways, including potential based methods (see e.g. [KW99, CBL06]). We were not able to find a potential based proof for learning permutations with doubly stochastic matrices, since there is no closed form solution for Sinkhorn Balancing. Finally, Kalai and Vempala's "Follow the Perturbed Leader" [KV05] approach can easily be applied to our problem, but it leads to worse bounds.

We introduce our notation in the next section. Section 3 presents the permutation learning model and gives several intuitive examples of appropriate loss motifs. Section 4 describes the details of the PermELearn algorithm, while Section 5 contains the proof of its relative mistake bound and uses the same methodology in an alternate analysis of the Weighted Majority algorithm. In Section 6, we apply the "Follow the Perturbed Leader" algorithm to learning permutations. The concluding section describes extensions and further work.

2 Notation

All matrices will be $n \times n$, and $\mathbf{1}$ and $\mathbf{0}$ denote the all ones and all zero matrices. For a matrix A , $A_{i,j}$ is the element of A in row i , and column j . We use $A \bullet B$ to denote the dot product between matrices A and B , i.e. $\sum_{i,j} A_{i,j} B_{i,j}$. We use single subscripts (e.g. A_k) to identify matrices/permutations from a sequence.

Permutations on n elements are frequently represented in two ways: as a vector, and as a matrix. We use the notation Π (and $\widehat{\Pi}$) to represent a permutation of elements $\{1, \dots, n\}$ into positions $\{1, \dots, n\}$ in either format, using the context to indicate the appropriate representation. Thus, for each $i \in \{1, \dots, n\}$, we use $\Pi(i)$ to denote the position that the i th element is mapped to by permutation Π , and matrix element $\Pi_{i,j} = 1$ if $\Pi(i) = j$ and 0 otherwise.

If L is a matrix with n rows then the product ΠL permutes the rows of L :

$$\Pi = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad L = \begin{pmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \end{pmatrix} \quad \Pi L = \begin{pmatrix} 21 & 22 & 23 & 24 \\ 41 & 42 & 43 & 44 \\ 31 & 32 & 33 & 34 \\ 11 & 12 & 13 & 14 \end{pmatrix}$$

Matrix for $\Pi = (2, 4, 3, 1)$ An arbitrary matrix Permuting the rows

Our algorithm will have some uncertainty about which permutation to predict with. Therefore it maintains a probability distribution over permutations in the form of a $n \times n$ doubly stochastic matrix W as its data structure.

3 The on-line protocol

We are interested in learning permutations in the on-line setting, where learning proceeds in a series of trials. We assume the losses are generated by an (adversarial) process that we will call “nature”. In each trial:

- The learner (probabilistically) chooses a permutation $\widehat{\Pi}$.
- Nature simultaneously chooses a loss matrix $L \in [0..1]^{n \times n}$ for the trial with the interpretation that $L_{i,j}$ is the loss for mapping element i to position j , and the loss of a permutation is the sum of the losses of its mappings, i.e. $\sum_i L_{i,\widehat{\Pi}(i)} = \widehat{\Pi} \bullet L$.
- At the end of the trial the algorithm is given L so that it can adjust its future predictions.

The expected loss incurred by the algorithm on the trial is $\mathbb{E}[\widehat{\Pi} \bullet L]$, where the expectation is over the algorithm’s random choice of $\widehat{\Pi}$.

Since the dot product is linear, $\mathbb{E}[\widehat{\Pi} \bullet L] = W \bullet L$, where $W = \mathbb{E}(\widehat{\Pi})$. The entry $W_{i,j}$ is the probability that the learner chooses a permutation $\widehat{\Pi}$ such that $\widehat{\Pi}(i) = j$. Since permutation matrices are doubly stochastic, the convex combination W is so as well.

It is worth emphasizing that the W matrix is a convenient *summary* of the distribution over permutations used by any algorithm (it doesn’t indicate which permutations have non-zero probability, for example). However, this summary is *sufficient* to determine the algorithm’s expected loss.

Although our algorithm is capable of handling arbitrary sequences of loss matrices L , nature is usually significantly more restricted. Most applications have a loss motif M that is known to the algorithm and nature is constrained to choose (row) permutations of M as its loss matrix L . In effect, at each trial nature chooses a “correct” permutation Π and uses the loss matrix $L = \Pi M$. Note that the permutation left-multiplies the loss motif, and thus permutes the rows of M . If nature chooses the identity permutation then the loss matrix L is the motif M itself. When M is known to the algorithm, it suffices to give the algorithm only the permutation Π at the end of the trial, rather than L itself.

Figure 1 gives examples of loss motifs. The last loss in the table is associated with the competitive analysis of adaptive list structures where the cost is the number of links traversed to find the desired element². Blum, Chawla, and Kalai [BCK03] give very efficient algorithms for this special case. In our notation, their bound has the same form as ours (1) but with the $n \ln n$ replaced by $O(n)$. However, our lower bound shows that the $\ln n$ factors in (2) are necessary in the general permutation setting.

Note that many compositions of loss motifs are possible. For example, given two motifs with their associated losses, any convex combination of the motifs creates a new motif for the (same) convex combination of the associated losses.

² In the adaptive list problem the searched element can be moved forward in the list for free, but other reorderings of the elements incur a cost not modeled here.

loss $\mathcal{L}(\widehat{\Pi}, \Pi)$	motif M
the number of elements i where $\widehat{\Pi}(i) \neq \Pi$	$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$
$\frac{1}{n-1} \sum_{i=1}^n \widehat{\Pi}(i) - \Pi(i) $, how far the elements are from there "correct" positions (division by $n-1$ ensures that the entries of M are in $[0, 1]$.)	$\frac{1}{3} \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{pmatrix}$
$\frac{1}{n-1} \sum_{i=1}^n \frac{ \widehat{\Pi}(i) - \Pi(i) }{\Pi(i)}$, a position weighted version of the above emphasizing the early positions in Π	$\frac{1}{3} \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1/2 & 0 & 1/2 & 1 \\ 2/3 & 1/3 & 0 & 1/3 \\ 3/4 & 1/2 & 1/4 & 0 \end{pmatrix}$
the number of elements mapped to the first half by Π but the second half by $\widehat{\Pi}$, or vice versa	$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$
the number of elements mapped to the first two positions by Π that fail to appear in the top three position of $\widehat{\Pi}$	$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$
the number of links traversed to find the first element of Π in a list ordered by $\widehat{\Pi}$	$\frac{1}{3} \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

Fig. 1. Loss motifs

Other component-wise combinations of two motifs (such as product or max) can also produce interesting loss motifs, but the combination usually cannot be distributed across the matrix dot-product calculation, and so cannot be expressed as a simple function of the original losses.

4 Algorithm

Our permutation learning algorithm uses exponential weights and we call it PermELearn. It maintains an $n \times n$ dimensional doubly stochastic W as its main data structure, where $W_{i,j}$ is the probability that PermELearn predicts with a permutation mapping element i to position j . In the absence of prior information it is natural to start with the uniform prior, i.e. the matrix $\frac{1}{n}$ in each entry.

In each iteration PermELearn must do two things:

1. Choose a permutation $\widehat{\Pi}$ from some distribution s.t. $\mathbb{E}[\widehat{\Pi}] = W$.
2. Create a new doubly stochastic matrix \widetilde{W} for use in the next trial based on the current W and the loss matrix L .

The first step is described in Algorithm 1. The algorithm greedily decomposes W into a convex combination of at most $n^2 - n + 1$ permutations, and then randomly selects one of these permutations for the prediction.³ By Birkhoff's theorem (see [Bha97]), every doubly stochastic matrix A is a convex combination

³ The decomposition is not unique and the implementation may have a bias as to exactly which convex combination is chosen.

Algorithm 1 PermELearn: Selecting a permutation

Require: a doubly stochastic $n \times n$ matrix W

```
A := W;  
for  $\ell = 1$  to  $n^2 - n + 1$  do  
    Find permutation  $\Pi_\ell$  such that each  $A_{i, \Pi_\ell(i)}$  is positive  
     $\alpha_\ell := \min_i A_{i, \Pi_\ell(i)}$   
     $A := A - \alpha_\ell \Pi_\ell$   
    Exit loop if all entries of  $A$  are zero  
end for {at end of loop  $W = \sum_{k=1}^\ell \alpha_k \Pi_k$ }  
Randomly select  $\Pi_k \in \{\Pi_1, \dots, \Pi_\ell\}$  using probabilities  $\alpha_k$  and return it.
```

Algorithm 2 PermELearn: Weight Matrix Update

Require: learning rate η , non-negative loss matrix L , and doubly stochastic weight matrix W

```
for each entry  $i, j$  of  $W$  do  
    Create  $W'$  where each  $W'_{i,j} = W_{i,j} e^{-\eta L_{i,j}}$   
end for  
Create doubly stochastic  $\tilde{W}$  by re-scaling the rows and columns of  $W'$  (Sinkhorn balancing) and update  $W$  to  $\tilde{W}$ .
```

of permutation matrices. In addition, one can find a permutation Π where each $A_{i, \Pi(i)} > 0$ by finding a perfect matching on the $n \times n$ bipartite graph containing the edge (i, j) whenever $A_{i,j} > 0$. Given a permutation Π where each $A_{i, \Pi(i)} > 0$ we form the new matrix $A' = A - \alpha \Pi$ where $\alpha = \min_i A_{i, \Pi(i)}$. Matrix A' has non-negative entries and A' has more zeros than A . Furthermore, each row and column of A' sum to $1 - \alpha$, so A' is $1 - \alpha$ times a doubly stochastic matrix (and thus $1 - \alpha$ times a convex combination of permutations).

After at most $n^2 - n$ iterations we arrive at a matrix A' with exactly n non-zero entries. Since the row and column sums of A' are the same, A' is just a constant times a permutation matrix. Therefore, we have found a way to express the original doubly stochastic matrix as the convex combination of (at most) $n^2 - n + 1$ permutation matrices (see Algorithm 1).

There are several improvements possible. In particular, we need not compute each perfect matching from scratch. If only q entries are zeroed by a permutation, then that permutation still represents a matching of size $n - q$ in the graph for the new matrix. Thus we need to find only q augmenting paths to complete the perfect matching. The whole process requires finding $O(n^2)$ augmenting paths at a cost of $O(n^2)$ each, for a total cost of $O(n^4)$ to decompose W into a convex combination of permutations.

The second step first multiplies the $W_{i,j}$ entries of the loss matrix by the factors $e^{-\eta L_{i,j}}$. These factors destroy the row and column normalization, so the matrix must be re-normalized to restore the doubly-stochastic property. There is no closed form for the normalization step. The standard iterative re-normalization method for non-negative matrices is called *Sinkhorn Balancing*. This method first normalizes the rows of the matrix to sum to one, and then

normalizes the columns. Since normalizing the columns typically destroys the row normalization, the process must be iterated until convergence [Sin64].

Normalizing the rows corresponds to pre-multiplying by a diagonal matrix. The product of these diagonal matrices thus represents the combined effect of the multiple row normalization steps. Similarly, the combined effect of the column normalization steps can be represented by post-multiplying the matrix by a diagonal matrix. Therefore Sinkhorn balancing a matrix A results in a doubly stochastic matrix RAC where R and C are diagonal matrices. Each entry $R_{i,i}$ is the positive multiplier applied to row i , and each entry $C_{j,j}$ is the positive multiplier of column j in order to convert A into a doubly stochastic matrix.

Convergence: There has been much written on the scaling of matrices, and we briefly describe only a few of the results here. Sinkhorn showed that this procedure converges and that the RAC conversion of any matrix A is unique if it exists⁴ (up to canceling multiples of R and C) [Sin64].

A number of authors consider scaling a matrix A so that the row and column sums are $1 \pm \epsilon$. Franklin and Lorenz [FL89] show that $O(\text{length}(A)/\epsilon)$ Sinkhorn iterations suffice, where $\text{length}(A)$ is the bit-length of matrix A 's binary representation. Kalantari and Khachiyan [KK96] show that $O(n^4 \ln \frac{n}{\epsilon} \ln \frac{1}{\min_{i,j} A_{i,j}})$ operations suffice using an interior point method. Linial, Samorodnitsky, and Wigderson [LSW00] give a preprocessing step after which only $O((n/\epsilon)^2)$ Sinkhorn iterations suffice. They also present a strongly polynomial time iterative procedure requiring $\tilde{O}(n^7 \log(1/\epsilon))$ iterations. Balakrishnan, Hwang, and Tomlin [BHT04] give an interior point method with complexity $O(n^6 \log(n/\epsilon))$. Finally, Fürer [Fur04] shows that if the row and column sums of A are $1 \pm \epsilon$ then every matrix entry changes by at most $\pm n\epsilon$ when A is scaled to a doubly stochastic matrix.

We defer further analysis of the imprecision in \tilde{W} to the full paper, and continue assuming that the algorithm produces a doubly stochastic \tilde{W} .

5 Bounds for PermELearn

Our analysis of PermELearn follows the entropy-based analysis of the exponentiated gradient family of algorithms [KW97]. This style of analysis first shows a per-trial progress bound using relative entropy to a comparator as a measure of progress, and then sums this invariant over the trials to obtain an expected total loss of the algorithm. As with the exponentiated gradient family of algorithms, we show that PermELearn's weight update is the solution to a relative entropy-regularized minimization problem.

Recall that the expected loss of PermELearn on a trial is a linear function of its W weight matrix. Therefore the gradient of the loss is independent of

⁴ Some non-negative matrices, like $\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$, cannot be converted into doubly stochastic matrices because of their pattern of zeros. The weight matrices we deal with have strictly positive entries, and thus can always be made doubly stochastic with an RAC conversion.

the current value of W . This property of the loss greatly simplifies the analysis. Our analysis for this relatively simple setting provides a good foundation for learning permutation matrices and lays the groundwork for the future study of more complicated permutation loss functions.

We start our analysis with an attempt to mimic the standard analysis [KW97] for the exponentiated gradient family updates which multiply by exponential factors and re-normalize. Unfortunately, the lack of a closed form for the normalization causes difficulties. Our solution is to break PermELearn’s update (Algorithm 2) into two steps, and use only the progress made to the intermediate un-normalized matrix in our per-trial bound (7). After showing that the normalization to a doubly stochastic matrix only increases the progress, we can sum the per-trial bound to obtain our main theorem.

The per-trial invariant used to analyze the exponentiated gradient family bounds (for every weight vector U) the decrease in relative entropy from a (normalized) U to the algorithm’s weight vector by a linear combination of the algorithm’s loss and the loss of U on the trial. In our case the weight vectors are matrices and we use the following (un-normalized) relative entropy between matrices A and B with non-negative entries:

$$\Delta(A, B) = \sum_{i,j} A_{i,j} \ln \frac{A_{i,j}}{B_{i,j}} + B_{i,j} - A_{i,j} \ .$$

Note that this is just the sum of the relative entropies between the corresponding rows (or equivalently, between the corresponding columns):

$$\Delta(A, B) = \sum_i \Delta(A_{i,*}, B_{i,*}) = \sum_j \Delta(A_{*,j}, B_{*,j}) \ .$$

A Dead End: In each trial, PermELearn multiplies each entry of its weight matrix by an exponential factor, and the rows and columns are normalized using one additional factor per row and column (Algorithm 2):

$$\widetilde{W}_{i,j} := \frac{W_{i,j} e^{-\eta L_{i,j}}}{r_i c_j}, \tag{3}$$

where r_i, c_j are chosen so that row i and column j of the matrix \widetilde{W} sum to one.

PermELearn’s update (3) solves the the following minimization problem:

$$\begin{aligned} \operatorname{argmin} \quad & (\Delta(A, W) + \eta (A \bullet L)). \\ \forall i : \quad & \sum_j A_{i,j} = 1 \\ \forall j : \quad & \sum_i A_{i,j} = 1 \end{aligned} \tag{4}$$

Lemma 1. *PermELearn’s updated weight matrix \widetilde{W} (3) is the solution of (4).*

Proof. We form a Lagrangian for the optimization problem:

$$l(A, \rho, \gamma) = \Delta(A, W) + \eta (A \bullet L) + \sum_i \rho_i (1 - \sum_j A_{i,j}) + \sum_j \gamma_j (1 - \sum_i A_{i,j}).$$

Setting the derivative with respect to $A_{i,j}$ to 0 yields $A_{i,j} = W_{i,j} e^{-\eta L_{i,j}} e^{\rho_i} e^{\gamma_j}$ and shows that the normalization factors r_i and c_j are exponentials of the Lagrange multipliers. \square

Since the linear constraints are feasible and the divergence is strictly convex, there always is a unique solution, even though the solution does not have a closed form.

We now examine the progress $\Delta(U, W) - \Delta(U, \widetilde{W})$ towards an arbitrary stochastic matrix U . Using Equation (3) and noting that all three matrices are doubly stochastic (so their entries sum to n), we see that

$$\Delta(U, W) - \Delta(U, \widetilde{W}) = -\eta U \bullet L + \sum_i \ln r_i + \sum_j \ln c_j.$$

Making this a useful invariant requires lower bounding the sums on the rhs by a constant times $W \bullet L$, the loss of the algorithm. Unfortunately we are stuck because the normalization factors don't even have a closed form.

Successful Analysis: We split the update (3) into two steps:

$$W'_{i,j} := W_{i,j} e^{-\eta L_{i,j}} \quad \text{and} \quad \widetilde{W}_{i,j} := \frac{W'_{i,j}}{r_i c_j}, \quad (5)$$

where r_i and c_j are chosen so that row i and column j of the matrix \widetilde{W} sum to one, respectively. Using the Lagrangian (as in the proof of Lemma 1), it is easy to see that these steps solve the following minimization problems:

$$W' = \underset{A}{\operatorname{argmin}} (\Delta(A, W) + \eta (A \bullet L)) \quad \text{and} \quad \widetilde{W} := \underset{\substack{\forall i : \sum_j A_{i,j} = 1 \\ \forall j : \sum_i A_{i,j} = 1}}{\operatorname{argmin}} \Delta(A, W'). \quad (6)$$

The second problem shows that the doubly stochastic matrix \widetilde{W} is the projection of W' onto to the linear row and column sum constraints. The strict convexity of the relative entropy between non-negative matrices and the feasibility of the linear constraints ensure that the solutions for both steps are unique.

We now lower bound the progress $\Delta(U, W) - \Delta(U, W')$ in the following lemma to get our per-trial invariant.

Lemma 2. *For any η , any doubly stochastic matrices U and W and any trial with loss matrix L ,*

$$\Delta(U, W) - \Delta(U, W') \geq (1 - e^{-\eta})(W \bullet L) - \eta(U \bullet L),$$

where W' is the intermediate matrix (6) constructed by PermELearn from W .

Proof. The proof manipulates the difference of relative entropies and applies the inequality $e^{-\eta x} \leq 1 - (1 - e^{-\eta})x$, which holds for any $x \in [0, 1]$ any η :

$$\begin{aligned}
\Delta(U, W) - \Delta(U, W') &= \sum_{i,j} \left(U_{i,j} \ln \frac{W'_{i,j}}{W_{i,j}} + W_{i,j} - W'_{i,j} \right) \\
&= \sum_{i,j} \left(U_{i,j} \ln(e^{-\eta L_{i,j}}) + W_{i,j} - W_{i,j} e^{-\eta L_{i,j}} \right) \\
&\geq \sum_{i,j} \left(-\eta L_{i,j} U_{i,j} + W_{i,j} - W_{i,j} (1 - (1 - e^{-\eta}) L_{i,j}) \right) \\
&= -\eta(U \bullet L) + (1 - e^{-\eta})(W \bullet L). \quad \square
\end{aligned}$$

The relative entropy is a Bregman divergence and \widetilde{W} is the relative entropy projection of W' w.r.t. linear constraints. Therefore, since U satisfies the constraints, we have by the Generalized Pythagorean Theorem (see e.g. [HW01])

$$\Delta(U, W') - \Delta(U, \widetilde{W}) = \Delta(\widetilde{W}, W') \geq 0.$$

Combining this with the inequality of Lemma 2 gives the critical per-trial invariant:

$$\Delta(U, W) - \Delta(U, \widetilde{W}) \geq (1 - e^{-\eta})(W \bullet L) - \eta(U \bullet L) \quad (7)$$

Before presenting our main theorem we introduce some notation to deal with sequences of trials. With respect to a sequence of T trials, define W_t and \widetilde{W}_t to be the initial and updated weight matrices at each trial $t \in \{1, \dots, T\}$ so that $\widetilde{W}_t = W_{t+1}$ for $1 \leq t \leq T$. We now bound $\sum_{t=1}^T W_t \bullet L_t$, the total expected loss of PermELearn over the entire sequence of trials.

Theorem 1. *For any learning rate η , any doubly stochastic matrices U and initial W_1 , and any sequence of T trials with loss matrices $L_t \in [0, 1]^{n \times n}$ (for $1 \leq t \leq T$), the loss of PermELearn is bounded by:*

$$\sum_{t=1}^T W_t \bullet L_t \leq \frac{\Delta(U, W_1) - \Delta(U, W_{T+1}) + \eta \sum_{t=1}^T U \bullet L_t}{1 - e^{-\eta}}.$$

Proof. Summing Equation (7) over the trials gives

$$\Delta(U, W_1) - \Delta(U, W_{T+1}) \geq (1 - e^{-\eta}) \sum_{t=1}^T W_t \bullet L_t - \eta \sum_{t=1}^T U \bullet L_t.$$

The bound then follows by solving for the loss of the algorithm. \square

When the entries of W_1 are all initialized to $\frac{1}{n}$ and U is a permutation then $\Delta(U, W_1) \leq n \ln n$. Note that the loss $\sum_{t=1}^T U \bullet L$ is minimized when U is a single permutation. If $\mathcal{L}_{\text{best}}$ denotes the loss of such a permutation, then the bound of Theorem 1 implies that the total loss of the algorithm is bounded by

$$\frac{n \ln n + \eta \mathcal{L}_{\text{best}}}{1 - e^{-\eta}}.$$

If an upper $\mathcal{L}_{\text{est}} \geq \mathcal{L}_{\text{best}}$ is known, then η can be tuned as done by Freund and Schapire [FS97] and the above bound becomes

$$\mathcal{L}_{\text{best}} + \sqrt{2\mathcal{L}_{\text{est}}n \ln n} + n \ln n.$$

Splitting the analysis of Weighted Majority: Perhaps the simplest case where the loss is linear in the parameter vector is the “decision theoretic” setting of [FS97]. There are N experts and the algorithm keeps a probability distribution \mathbf{w} over the experts. In each trial the algorithm picks expert i with probability w_i and then gets a loss vector $\boldsymbol{\ell} \in [0, 1]^N$. Each expert i incurs loss ℓ_i and the algorithm’s expected loss is $\mathbf{w} \cdot \boldsymbol{\ell}$. Finally \mathbf{w} is updated to $\tilde{\mathbf{w}}$ for the next trial.

The (Randomized) Weighted Majority algorithm [LW94] or Hedge algorithm [FS97] in this setting updates $\tilde{w}_i = \frac{w_i e^{-\eta \ell_i}}{\sum_j w_j e^{-\eta \ell_j}}$. This update is motivated by a tradeoff between the un-normalized relative entropy and expected loss [KW99]:

$$\tilde{\mathbf{w}} := \operatorname{argmin}_{\sum_i \hat{w}_i = 1} (\Delta(\hat{\mathbf{w}}, \mathbf{w}) + \eta \hat{\mathbf{w}} \cdot \boldsymbol{\ell}).$$

As in the permutation case, we can split this update (and motivation) into two steps: setting each $w'_i = w_i e^{-\eta \ell_i}$ then $\tilde{\mathbf{w}} = \mathbf{w}' / \sum_i w'_i$. These correspond to:

$$\mathbf{w}' := \operatorname{argmin}_{\hat{\mathbf{w}}} (\Delta(\hat{\mathbf{w}}, \mathbf{w}) + \eta \hat{\mathbf{w}} \cdot \boldsymbol{\ell}) \quad \text{and} \quad \tilde{\mathbf{w}} := \operatorname{argmin}_{\sum_i \hat{w}_i = 1} \Delta(\hat{\mathbf{w}}, \mathbf{w}').$$

The following lower bound has been shown on the progress towards any probability vector \mathbf{u} serving as a comparator [LW94, FS97, KW99]:

$$\Delta(\mathbf{u}, \mathbf{w}) - \Delta(\mathbf{u}, \tilde{\mathbf{w}}) \geq \mathbf{w} \cdot \boldsymbol{\ell} (1 - e^{-\eta}) - \eta \mathbf{u} \cdot \boldsymbol{\ell}. \quad (8)$$

Surprisingly the same inequality already holds for the un-normalized update⁵:

$$\Delta(\mathbf{u}, \mathbf{w}) - \Delta(\mathbf{u}, \mathbf{w}') = -\eta \mathbf{u} \cdot \boldsymbol{\ell} + \sum_i w_i (1 - e^{-\eta \ell_i}) \geq \mathbf{w} \cdot \boldsymbol{\ell} (1 - e^{-\eta}) - \eta \mathbf{u} \cdot \boldsymbol{\ell},$$

where the last inequality uses $e^{-\eta x} \leq 1 - (1 - e^{-\eta})x$, for any $x \in [0, 1]$. Since the normalization is a projection w.r.t. a Bregman divergence onto a linear constraint satisfied by the comparator \mathbf{u} , $\Delta(\mathbf{u}, \mathbf{w}') - \Delta(\mathbf{u}, \tilde{\mathbf{w}}) \geq 0$ by the Generalized Pythagorean Theorem [HW01]. The total progress for both steps is again (8).

When to normalize? Probably the most surprising aspect about the proof methodology is the flexibility about how and when to project onto the constraints. Instead of projecting a nonnegative matrix onto all $2n$ constraints at once (as in optimization problem (6)), we could mimic the Sinkhorn balancing algorithm and iteratively project onto the n row and n column constraints until convergence. The Generalized Pythagorean Theorem shows that projecting

⁵ Note that if the algorithm does not normalize the weights then \mathbf{w} is no longer a distribution. When $\sum_i w_i < 1$, the loss $\mathbf{w} \cdot L$ amounts to abstaining (incurring 0 loss) with probability $1 - \sum_i w_i$, and predicting as expert i with probability w_i .

onto *any* convex constraint that is satisfied by the comparator class of doubly stochastic matrices brings the weight matrix closer to *every* doubly stochastic matrix⁶. Therefore our bound on $\sum_t W_t \bullet L_t$ (Theorem 1) holds if the exponential updates are interleaved with any sequence of projections to some subsets of the constraints. However, if the normalization constraint are not enforced then W is no longer a convex combination of permutations, and W can approach $\mathbf{0}$ (so $W \bullet L \approx 0$ for all L).

There is a direct argument that shows that the same final doubly stochastic matrix is reached if we interleave the exponential updates with projections to any of the constraints as long as all $2n$ hold at the end. To see this we partition the class of matrices with positive entries into equivalence classes. Call two such matrices A and B *equivalent* if there are diagonal matrices R and C with positive diagonal entries such that $B = RAC$. Note that $[RAC]_{i,j} = R_{i,i}A_{i,j}C_{j,j}$ so B is just a rescaled version of A . Projecting onto any row (and/or column) sum constraints amounts to pre- (and/or post-) multiplying the matrix by some positive diagonal matrix R (and/or C). Therefore if matrices A and B are equivalent then either projecting one onto a set of row/column sum constraints or multiplying their corresponding entries by the same factor results in matrices that are still equivalent. This means that any two runs from equivalent initial matrices that involve the same exponential updates end with equivalent final matrices even if they use different projections at different times. Finally, for any two equivalent matrices A and RAC , where the entries of A and the diagonal entries of R and C are positive, we have (from the Lagrangians):

$$\begin{array}{l} \operatorname{argmin} \quad \Delta(\widehat{A}, A) \\ \forall i : \sum_j \widehat{A}_{i,j} = 1 \\ \forall j : \sum_i \widehat{A}_{i,j} = 1 \end{array} = \begin{array}{l} \operatorname{argmin} \quad \Delta(\widehat{A}, RAC). \\ \forall i : \sum_j \widehat{A}_{i,j} = 1 \\ \forall j : \sum_i \widehat{A}_{i,j} = 1 \end{array}$$

Since the relative entropy is strictly convex, both minimization problems have the same unique minimum. Curiously enough the same phenomenon already happens in the weighted majority case: Two non-negative vectors \mathbf{a} and \mathbf{b} are *equivalent* if $\mathbf{a} = c\mathbf{b}$, where c is any nonnegative scalar, and again each equivalence class has exactly one normalized weight vector.

6 Follow the perturbed leader

Kalai and Vempala [KV05] describe and bound “follow the perturbed leader” (FPL) algorithms for on-line prediction in a very general setting. Their FPL* algorithm has bounds closely related to WM and other multiplicative weight algorithms. However, despite the apparent similarity between our representations and the general formulation of FPL*, the bounds we were able to obtain for FPL* are weaker than the bounds derived using the relative entropies.

⁶ There is a large body of work on finding a solution subject to constraints via iterated Bregman projections (See e.g. [CL81]).

The FPL setting has an abstract k -dimensional decision space used to encode predictors as well as a k -dimensional state space used to represent the losses of the predictors. At any trial, the current loss of a particular predictor is the dot product between that predictor’s representation in the decision space and the state-space vector for the trial. This general setting can explicitly represent each permutation and its loss when $k = n!$. The FPL setting also easily handles the encodings of permutations and losses used by PermELearn by representing each permutation matrix Π and loss matrix L as n^2 -dimensional vectors.

The FPL* algorithm [KV05] takes a parameter ϵ and maintains a cumulative loss matrix C (initially C is the zero matrix) At each trial, FPL*:

1. Generates a random perturbation matrix P where each $P_{i,j}$ is proportional to $\pm r_{i,j}$ where $r_{i,j}$ is drawn from the standard exponential distribution.
2. Predicts with a permutation Π minimizing $\Pi \bullet (C + P)$.
3. After getting the loss matrix L , updates C to $C + L$.

Note that FPL* is more computationally efficient than PermELearn. It takes only $O(n^3)$ time to make its prediction (the time to compute a minimum weight bipartite matching) and only $O(n^2)$ time to update C . Unfortunately the generic FPL* loss bounds are not as good as the bounds on PermELearn. In particular, they show that the loss of FPL* on any sequence of trials is at most⁷

$$(1 + \epsilon)\mathcal{L}_{\text{best}} + \frac{8n^3(1 + \ln n)}{\epsilon}$$

where ϵ is a parameter of the algorithm. When the loss of the best expert is known ahead of time, ϵ can be tuned and the bound becomes

$$\mathcal{L}_{\text{best}} + 4\sqrt{2\mathcal{L}_{\text{best}}n^3(1 + \ln n) + 8n^3(1 + \ln n)} .$$

Although FPL* gets the same $\mathcal{L}_{\text{best}}$ leading term, the excess loss over the best permutation grows as $n^3 \ln n$ rather than the $n \ln n$ growth of PermELearn’s bound. Of course, PermELearn pays for the improved bound by requiring more time.

It is important to note that Kalai and Vempala also present a refined analysis of FPL* when the perturbed leader changes only rarely. This analysis leads to bounds on weighted majority that are similar to the bounds given by the entropic analysis (although the constant on the square-root term is not quite as good). However, this refined analysis cannot be directly applied with the efficient representations of permutations because the total perturbations associated with different permutations are no longer independent exponentials. We leave the adaptation of the refined analysis to the permutation case as an open problem.

7 Conclusions and Further Work

The main technical insight in our analysis of PermELearn is that the per-trial progress bound already holds for the un-normalized update and that the nor-

⁷ The n^3 terms in the bounds for FPL are n times the sum of the entries in the loss motif. So if the loss motif’s entries sum to only n , then the n^3 factors become n^2 .

malization step only helps. This finesses the difficulty of accounting for the normalization (which does not have a closed form) in the analysis. The same thing already happens in the Weighted Majority setting.

As our main contribution we showed that the problem of learning a permutation is amenable to the same techniques as learning in the expert setting. This means that all techniques from that line of research are likely to carry over: lower bounding the weights when the comparator is shifting, long-term memory when shifting between a small set of comparators [BW02], capping the weights from the top if the goal is to be close to the best set of comparators [WK06], adapting the updates to the multi-armed bandit setting when less feedback is provided [ACBFS02], PAC Bayes analysis of the exponential updates [McA03].

Our analysis techniques rely on Bregman projection methods. This means that the bounds remain unchanged if we add convex side constraints on the parameter matrix because as long as the comparator satisfies the side constraints, we can always project onto these constraints without hurting the analysis [HW01]. With the side constraints we can enforce relationships between the parameters, such as $W_{i,j} \geq W_{i,k}$ (i is more likely mapped to j than k).

We also applied the “Follow the Perturbed Leader” techniques to our permutation problem. This algorithm adds randomness to the total losses and then predicts with a minimum weighted matching which costs $O(n^3)$ whereas our more complicated algorithm is at least $O(n^4)$ and has precision issues. However the bounds provable for FPL are much worse than for the WM style analysis used here. The key open problem is whether we can have the best of both worlds: add randomness to the loss matrix so that the expected minimum weighted matching is the stochastic matrix produced by the PermELearn update (3). This would mean that we could use the faster algorithm together with our tighter analysis. In the simpler weighted majority setting this has been done already [KW05, Kal05]. However we do not yet know how to simulate the PermELearn update this way.

Acknowledgments: We thank David DesJardins, Jake Abernethy, Dimitris Achlioptas, Dima Kuzmin, and the anonymous referees for helpful feedback.

References

- [ACBFS02] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The non-stochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.
- [BCK03] A. Blum, S. Chawla, and A. Kalai. Static optimality and dynamic search-optimality in lists and trees. *Algorithmica*, 36:249–260, 2003.
- [Bha97] R. Bhatia. *Matrix Analysis*. Springer, Berlin, 1997.
- [BHT04] Hamsa Balakrishnan, Inseok Hwang, and Claire Tomlin. Polynomial approximation algorithms for belief matrix maintenance in identity management. In *43rd IEEE Conference on Decision and Control*, pages 4874–4879, December 2004.
- [BW02] Olivier Bousquet and Manfred K. Warmuth. Tracking a small set of experts by mixing past posteriors. *Journal of Machine Learning Research*, 3:363–396, 2002.

- [CBL06] N. Cesa-Bianchi and Gabor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.
- [CL81] Y. Censor and A. Lent. An iterative row-action method for interval convex programming. *Journal of Optimization Theory and Applications*, 34(3):321–353, July 1981.
- [FL89] Joel Franklin and Jens Lorenz. On the scaling of multidimensional matrices. *Linear Algebra and its applications*, 114/115:717–735, 1989.
- [FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.
- [Fur04] Martin Furer. Quadratic convergence for scaling of matrices. In *Proceedings of ALENEX/ANALCO*, pages 216–223. SIAM, 2004.
- [HS97] D. P. Helmbold and R. E. Schapire. Predicting nearly as well as the best pruning of a decision tree. *Machine Learning*, 27(01):51–68, 1997.
- [HW01] Mark Herbster and Manfred K. Warmuth. Tracking the best linear predictor. *Journal of Machine Learning Research*, 1:281–309, 2001.
- [Kal05] Adam Kalai. Simulating weighted majority with FPL. Private communication, 2005.
- [KK96] Bahman Kalantari and Leonid Khachiyan. On the complexity of nonnegative-matrix scaling. *Linear Algebra and its applications*, 240:87–103, 1996.
- [KV05] Adam Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *J. Comput. Syst. Sci.*, 71(3):291–307, 2005. Special issue Learning Theory 2003.
- [KW97] J. Kivinen and M. K. Warmuth. Additive versus exponentiated gradient updates for linear prediction. *Information and Computation*, 132(1):1–64, January 1997.
- [KW99] Jyrki Kivinen and Manfred K. Warmuth. Averaging expert predictions. In *Computational Learning Theory, 4th European Conference, EuroCOLT '99, Nordkirchen, Germany, March 29-31, 1999, Proceedings*, volume 1572 of *Lecture Notes in Artificial Intelligence*, pages 153–167. Springer, 1999.
- [KW05] Dima Kuzmin and M. K. Warmuth. Optimum follow the leader algorithm. In *Proceedings of the 18th Annual Conference on Learning Theory (COLT 05)*, pages 684–686. Springer, June 2005. Open problem.
- [LSW00] Nathan Linial, Alex Samorodnitsky, and Avi Wigderson. A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents. *Combinatorica*, 20(4):545–568, 2000.
- [LW94] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Inform. Comput.*, 108(2):212–261, 1994.
- [McA03] D. McAllester. PAC-Bayesian stochastic model selection. *Machine Learning*, 51(1):5–21, 2003.
- [Sin64] R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The Annals of Mathematical Statistics*, 35(2):876–879, June 1964.
- [TW03] Eiji Takimoto and Manfred K. Warmuth. Path kernels and multiplicative updates. *Journal of Machine Learning Research*, 4:773–818, 2003.
- [WK06] M. K. Warmuth and D. Kuzmin. Randomized PCA algorithms with regret bounds that are logarithmic in the dimension. In *Advances in Neural Information Processing Systems 19 (NIPS 06)*. MIT Press, December 2006.