Learning Permutations with Sinkhorn Policy Gradient

Patrick Emami
Department of Computer and
Information Science and Engineering
Gainesville, FL, 32611
pemami@ufl.edu

Sanjay Ranka
Department of Computer and
Information Science and Engineering
Gainesville, FL, 32611
sanjayranka@gmail.com

Abstract

Many problems at the intersection of combinatorics and computer science require solving for a permutation that optimally matches, ranks, or sorts some data. These problems usually have a task-specific, often non-differentiable objective function that data-driven algorithms can use as a learning signal. In this paper, we propose the Sinkhorn Policy Gradient (SPG) algorithm for learning policies on permutation matrices. The actor-critic neural network architecture we introduce for SPG uniquely decouples representation learning of the state space from the highly-structured action space of permutations with a temperature-controlled *Sinkhorn layer*. The Sinkhorn layer produces continuous relaxations of permutation matrices so that the actor-critic architecture can be trained end-to-end. Our empirical results show that agents trained with SPG can perform competitively on sorting, the Euclidean TSP, and matching tasks. We also observe that SPG is significantly more data efficient at the matching task than the baseline methods, which indicates that SPG is conducive to learning representations that are useful for reasoning about permutations.

1 Introduction

Learning to solve combinatorial optimization problems from data has applications in many fields. As a motivating example, consider planar minimum/maximum weight matching. Given a bipartite graph with vertices represented as points in the plane, the objective is to find a permutation matrix that matches vertices such that the sum of Euclidean distances amongst all match pairs is minimized/-maximized. Other related combinatorial problems include graph matching [4, 27], ranking [5, 1], and data association in multi-object tracking [37, 23, 24]. Data association can be cast as a minimum weight matching problem; given a sequence of raw images containing object detections, the objective is to find the optimal matching between detections in adjacent images to form object tracks.

In this paper, we propose a data-driven algorithm for the task of learning permutations based on a policy gradient method from the reinforcement learning (RL) literature. Data-driven approaches to solving combinatorial problems involve training a model on a dataset of problem instances drawn from a distribution, so that the model is able to score highly with respect to the task-specific objective on a test set of instances drawn from the same distribution. Models trained with supervised learning can perform well by relying on traditional loss functions such as mean squared error [34, 35], cross-entropy error [27], or mean squared error augmented with task-specific objectives [24]. However, obtaining large quantities of solved problem instances to build a labeled dataset is not always feasible.

Recent empirical results indicate that a task-specific objective can be used as the sole learning signal. Most notably, [2] uses an encoder and decoder with attention, trained with REINFORCE [36], to solve combinatorial problems that have a sequential nature (e.g., the Euclidean TSP). This was recently

extended to take advantage of the graph structure of the inputs by replacing the encoder with a graph attention layer [16]. A DQN [25]-inspired algorithm proposed in [8] also learns graph embeddings of problem instances, for the Euclidean TSP, Maximum Cut, and Minimum Vertex Cover problems. We have developed SPG, a policy gradient method designed for the class of combinatorial problems involving permutations. Indeed, the action space of a policy trained with SPG is the discrete set of $N \times N$ permutation matrices ($\mathcal{P}_N$). In contrast to similar learning-based approaches [2, 8, 16], SPG is not restricted to learning policies that emulate a greedy heuristic. We demonstrate that SPG is able to learn to sort integers, produce near-optimal matchings for maximum weight matching (MWM), and find tours of competitive length on the NP-Hard Euclidean Traveling Salesman Problem (TSP). On the MWM task, SPG is more data-efficient and outperforms baseline methods when scaling up to larger problem sizes.

1.1 Learning permutations with the Sinkhorn-Knopp algorithm

Learning permutations is challenging for two main reasons. First, the number of permutations grows factorially in the size of the problem. Second, the non-differentiability of $\mathcal{P}_N$ prevents learning algorithms from directly using backpropagation for end-to-end training. Due to [1], a truncated version of the Sinkhorn-Knopp algorithm [33], which maps a square matrix to a doubly-stochastic matrix (all rows and columns sum to one), was derived for use as a layer in a composable, end-to-end differentiable model. This *Sinkhorn layer* produces continuous and differentiable relaxations of permutation matrices. Recently, [22] built upon this work and introduced Gumbel-Sinkhorn networks for performing inference on probabilistic models that have latent permutation variables. Gumbel-Sinkhorn networks combine Sinkhorn layers with a temperature-controlled softmax gradient estimator for discrete variables [14, 21] to infer permutations from data. We use temperature-controlled Sinkhorn layers to develop a deep neural network architecture for a policy network that can be trained with SPG. We also describe a technique for removing bias induced by the continuous relaxation. The rest of our paper is structured as follows. In the next section, we introduce the requisite background for explaining SPG. In Section 3, we provide details on SPG. In Section 4, we present findings from ablation studies and the main set of experiments. In Section 5, we discuss related works and conclude.

2 Background

Notation. We use capital letters to represent scalars (e.g., N, K), bolded capital letters for matrices (e.g., $\mathbf{M}$), script capital letters for sets (e.g., $\mathcal{S}$), lowercase letters for elements of a set (e.g., $s \in \mathcal{S}$), and bolded lowercase letters for vectors (e.g., $\mathbf{b}$). Occasionally it will be necessary to introduce a concept using notation from another paper; we will point out these instances to maintain clarity.

2.1 Problem setting

The learning framework we most closely follow is the average reward contextual bandits setting. Formally, at each step, there is a context which consists of a state $s \in \mathcal{S}$ and a set of actions $\mathcal{A}$ available to an agent. Here, $\mathcal{A} = \mathcal{P}_N$. States are instances of combinatorial problems of size N drawn from a distribution ρ. As an example, consider the task of sorting ten numbers; $\mathcal{S}$ is the set of all 10! orderings of the ten numbers, ρ is a uniform distribution over $\mathcal{S}$, and $\mathcal{A}$ is the set of all 10×10 permutation matrices. Unlike the standard contextual bandits setting, the number of "bandit arms" scales factorially in the problem size N and we are interested in finding a deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$ that does not take the entire context as input. It might be possible to also formulate this problem with a stochastic policy $\pi(a|s)$ using, e.g., the tractable, differentiable density over permutations based on the reparameterization trick proposed recently in [20]. The recently introduced Gumbel-Sinkhorn distribution does not have a tractable density [22], and hence we leave this line of inquiry for future work.

The immediate reward, or the return, that the agent receives from the environment is $r(s, \pi(s))$. If we parameterize the policy with parameters θ, the agent's objective can be defined as finding θ^* that maximizes the average reward $J(\pi_\theta) = \mathbb{E}_{s \sim \rho}[r(s, \pi_\theta(s))]$. Note that the objective can also be written in terms of the *regret*, or the expected amount of return the agent "missed out on" by selecting sub-optimal actions.

2.2 Deterministic policy gradient methods

We now show how the off-policy actor-critic deterministic policy gradient algorithm [32], originally derived for RL, can be formulated for our contextual bandits setting. Let the deterministic target policy be π_θ. The action-value function $Q(s, a)$ induced by policy π_θ, defined as the return for choosing action a in state s under deterministic policy π_θ, is equal to $r(s, \pi_\theta(s))$. In Section 4.2 of [32], the off-policy deterministic policy gradient is defined for an arbitrary behavior policy β (e.g., an ϵ-greedy policy for encouraging exploration) as

$$\nabla_\theta J_\beta(\pi_\theta) \approx \mathop{\mathbb{E}}_{s \sim \rho} \left[\nabla_\theta \pi_\theta(s) \nabla_a Q(s, a) \big|_{a = \pi_\theta(s)} \right], \tag{1}$$

where $Q(s, a)$ is estimated by a differentiable critic $Q_{\theta'}(s, a)$ in practice with parameters θ', fit with off-policy experience from β. Note that throughout this paper, we assume that the regularity conditions for the Markov Decision Process in [32] also hold for our contextual bandits setting. To update the critic parameters θ', we can descend the gradient of the following mean squared error loss:

$$\nabla_{\theta'} \mathcal{L}_{\theta'} = \mathop{\mathbb{E}}_{s \sim \rho} \left[\left(r(s, \beta(s)) - Q_{\theta'}(s, \beta(s)) \right) \nabla_{\theta'} Q_{\theta'}(s, \beta(s)) \right]. \tag{2}$$

Equations 1 and 2 can be derived by setting the RL episode horizon T in [32] to one, i.e., we are equating our contextual bandits setting to 1-step RL. The extension of our proposed policy gradient method to episodic RL is possible and is left for future work. In practice, the policy and action-value functions in Equations 1 and 2 are implemented as deep neural networks, as is the case with the actor-critic paradigm used in the deep deterministic policy gradient (DDPG) algorithm [19]. As a result of using nonlinear function approximation for learning the actor and critic, the policy gradient (Equation 1) is no longer exact and bias is introduced. The critic is not updated with a 1-step temporal difference rule as it is in DDPG; hence, target networks are not needed to stabilize learning. In Section 3, we will provide complete algorithmic details on how we make use of Equations 1 and 2 to implement SPG. In the next section, we introduce the technique used to relax $\mathcal{P}_N$ to a continuous and differentiable set, which is necessary for computing the policy gradient.

2.3 Temperature-controlled Sinkhorn layer

To use the deterministic policy gradient defined in the previous section, we need a continuous and differentiable approximation of $\mathcal{P}_N$. Recently, [22] proposed to combine Sinkhorn layers [1] with a temperature-controlled continuous relaxation inspired by the Gumbel-Softmax trick [14, 21]. We first describe Sinkhorn layers for positive square matrices $\mathbf{X}$, which we hereafter refer to as the Sinkhorn operator $S^L(\cdot)$. Using the notation of [22], the Sinkhorn operator is defined recursively for $1 \leq i \leq L$ as

$$S^0 = \exp(\mathbf{X}) \tag{3}$$

$$S^i = \mathcal{T}_c(\mathcal{T}_r(S^{i-1}(\mathbf{X}))), \tag{4}$$

where $\mathcal{T}_c^{j,k}(\mathbf{X}) = \frac{\mathbf{X}_{j,k}}{\sum_l \mathbf{X}_{l,j}}$ is the column normalization operator and $\mathcal{T}_r^{j,k}(\mathbf{X}) = \frac{\mathbf{X}_{j,k}}{\sum_l \mathbf{X}_{j,l}}$ is the row normalization operator. The gradients with respect to the input can be computed efficiently by unrolling each row and column normalization [1]. One of the main results from [22] is that, by introducing a temperature parameter τ, applying $S^L(\cdot/\tau)$ to $\mathbf{X}$ produces the doubly-stochastic matrix $\mathbf{M}$ whose entries tend to zero or one in the limit as τ approaches zero. In practice, L and τ are treated as hyperparameters; we discuss how they are selected for our experiments in Section 4.2 and Appendix D. Example code for a stable implementation of $S^L(\cdot/\tau)$ in log-scale to mitigate numerical errors is provided in Appendix A. In the next section, we describe $S^L(\cdot/\tau)$ is used within the SPG actor network.

3 Sinkhorn Policy Gradient

In this section, we describe SPG, an off-policy deterministic policy gradient algorithm for the action space of $\mathcal{P}_N$. We also introduce a novel actor-critic neural network architecture for SPG, as well as a technique for reducing the policy gradient bias induced by the continuous relaxation.
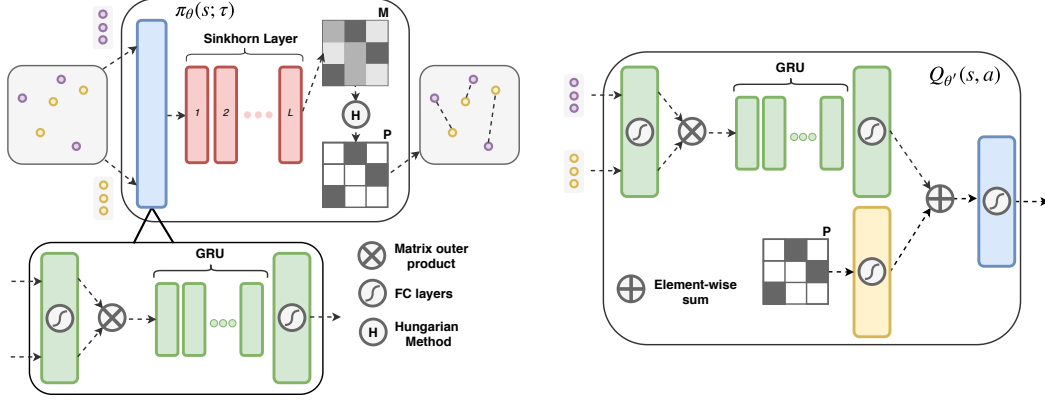
Figure 1: Visual summarization of SPG+Matching's actor (left) and critic (right) network architectures; full details are provided in Appendix A. In the actor network, two sets of objects are processed by a shared embedding layer, after which the matrix outer product of the two embeddings is computed. The fused embedding is fed into a GRU for representation learning over all possible match pairs. The output is passed to a Sinkhorn layer, which produces the doubly-stochastic matrix $\mathbf{M}$. The Hungarian algorithm is used to round $\mathbf{M}$ to a permutation matrix $\mathbf{P}$. The gradient of the actor network is taken with respect to $\mathbf{M}$, "bypassing" $\mathbf{P}$. The critic network uses the same embedding layer for the state, and the action is embedded with a fully-connected layer. These embeddings are fused with an element-wise sum, and mapped via a linear layer to a scalar. Best viewed in color.

We relax $\mathcal{P}_N$ to the continuous set of $N \times N$ doubly-stochastic matrices with $S^L(\cdot/\tau)$. The actor network takes the state as input, typically an $N \times K$ matrix when $\mathcal{S} = \mathbb{R}^{N \times K}$, embeds it into a high-dimensional space, and then uses $S^L(\cdot/\tau)$ to output a doubly-stochastic matrix $\mathbf{M}$. Note that since τ is fixed, the entries of $\mathbf{M}$ are not binary. At the start of training, the entries of $\mathbf{M}$ will be close to $1/N$, and as the actor gets more "confident" in its action selection, $\mathbf{M}$'s values get pushed closer to either zero or one. We round $\mathbf{M}$ to the nearest permutation matrix $\mathbf{P} = \mathrm{H}(\mathbf{M})$ with the $O(n^3)$ Hungarian algorithm [18, 26]. Here, "nearest" technically means the permutation $\mathbf{P}$ that maximizes $\mathrm{trace}(\mathbf{P}^\intercal\mathbf{M})$.

Given $\mathbf{P}$ and $\mathbf{M}$, we can now redefine the policy and critic loss gradients by slightly adjusting Equations 1 and 2. The new policy gradient looks like

$$\nabla_\theta J_\beta(\pi_\theta) \approx \mathop{\mathbb{E}}_{s \sim \rho} \left[\nabla_\theta \pi_\theta(s) \nabla_a Q_{\theta'}(s,a)\big|_{a=\mathbf{M}} \right] \tag{5}$$

and the new critic loss gradient is

$$\nabla_{\theta'} \mathcal{L}_{\theta'} = \mathop{\mathbb{E}}_{s \sim \rho} \left[\left(r(s,\mathbf{P}) - Q_{\theta'}(s,\mathbf{P}) \right) \nabla_{\theta'} Q_{\theta'}(s,\mathbf{P}) \right]. \tag{6}$$

In Equation 5, we need to use the continuous relaxation $\mathbf{M}$ for computing $\nabla_\theta \pi_\theta(s)$ because $\mathbf{P}$ is not differentiable with respect to θ. $\mathbf{M}$ is also used to compute the critic action-gradient because $\nabla_a Q_{\theta'}(s,a)\big|_{a=\mathbf{P}}$ is zero almost everywhere, and furthermore $Q_{\theta'}(s,\mathbf{P})$ has discontinuities. This heuristic of "bypassing" $\mathbf{P}$, i.e., treating it like an identity function when computing the policy gradient, is inspired by the straight-through gradient estimator [3]. The complete training loop for SPG, which resembles the one used by DDPG, is provided in Algorithm 1 in Appendix B. Like DDPG, SPG uses a replay buffer when computing the gradients in the backwards pass. We next describe how we can remove most of the bias that the relaxation introduces into the policy gradient.

De-biasing the policy gradient.   In our initial experiments, we observed that the actor network's improvement in terms of average reward would stagnate quite early during training; however, the critic loss would continue to decrease. Notice that the policy gradient is defined using the critic action-gradient $\nabla_a Q(s,a)|_{a=\mathbf{M}}$ taken with respect to *continuous actions* $\mathbf{M}$, and the critic loss is defined for *discrete actions* $\mathbf{P}$. Since the Q-values for the continuous actions were not being updated by the critic loss, the critic action-gradient was not accurately approximating the direction
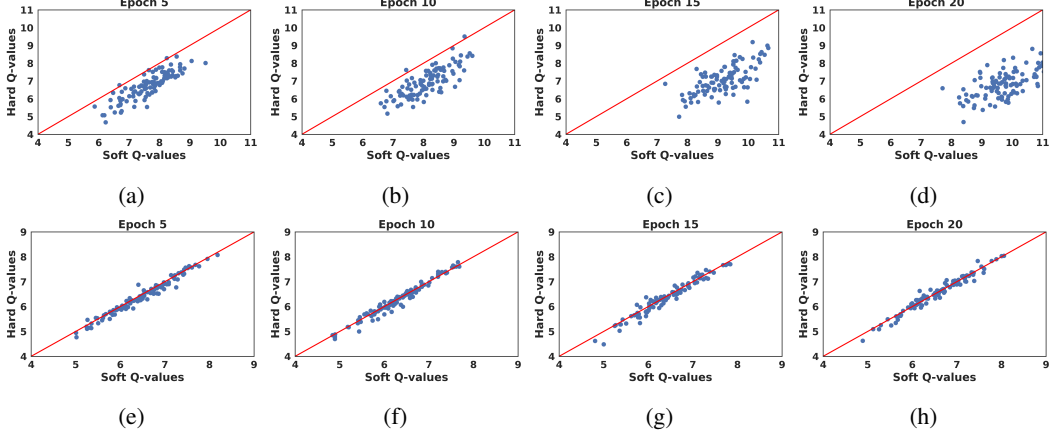
Figure 2: (a-d) The top row compares predicted Q-values for "hard" (discrete) and "soft" (relaxed) permutations from a critic trained *without* the additional penalty term on MWM-10. Each column shows Q-values from a set of 100 random states after [5,10,15,20] epochs of training (left to right). The diagonal line visualizes $y = x$. (e-h) The bottom row compares predicted Q-values for a critic trained *with* the additional penalty term. The linear relationship between the "hard" and "soft" Q-values shows the penalty term's effectiveness.

of maximum reward improvement (see Figure 2a-d). To de-bias the policy gradient, we added an auxiliary term to the critic loss that treats the problem of approximating the Q-values for the discrete actions with the Q-values for the continuous actions as a regression. The critic loss with the penalty for dissimilarity between Q-values is

$$\mathcal{L}_{\theta'} = \text{MSE}\big(r(s, \mathbf{P}), Q_{\theta'}(s, \mathbf{P})\big) + \text{MSE}\big(\text{stop\_grad}\big(Q_{\theta'}(s, \mathbf{P})\big), Q_{\theta'}(s, \mathbf{M})\big), \quad (7)$$

where MSE is short-hand for the mean squared error. Equations 5 and 7 are the key components of the SPG algorithm. We provide a geometric interpretation as to why the penalty term helps the critic action-gradient correctly approximate the direction of maximum reward improvement in Appendix C. In Section 4.2, we show the penalty term's effectiveness with an ablation study.

Actor-critic architecture. SPG uses deep neural networks to implement the actor $\pi_\theta(s; \tau)$ and the critic $Q_{\theta'}(s, a)$. The SPG+Matching architecture is shown in Figure 1. For non-matching combinatorial problems where the only input is a single set of objects, the architecture differs in that there is only a single input to the embedding layer, the GRU is bidirectional, and the matrix outer product is removed. This version of SPG is called "SPG+Sequential". Implementation details for both architectures are in Appendix A.

Exploration. We propose to use an exploration strategy inspired by local search methods from meta-heuristic algorithms like the Greedy Randomized Adaptive Search Procedure (GRASP) [30]. In particular, k-exchange neighborhoods are used to perturb the permutations selected by SPG; this exploration heuristic works by randomly swapping k rows of the permutation matrix. In our experiments, we fix $k = 2$. This is combined with ϵ-greedy exploration to control the amount of exploration over time. We examine the impact of this exploration strategy on SPG's performance in Section 4.2.

4 Experiments

4.1 Experimental setup

To evaluate SPG, we considered three combinatorial optimization problems whose solutions can be represented as permutations: sorting with N=20,50, MWM with N=10,15,20,25, and the Euclidean TSP with N=20.
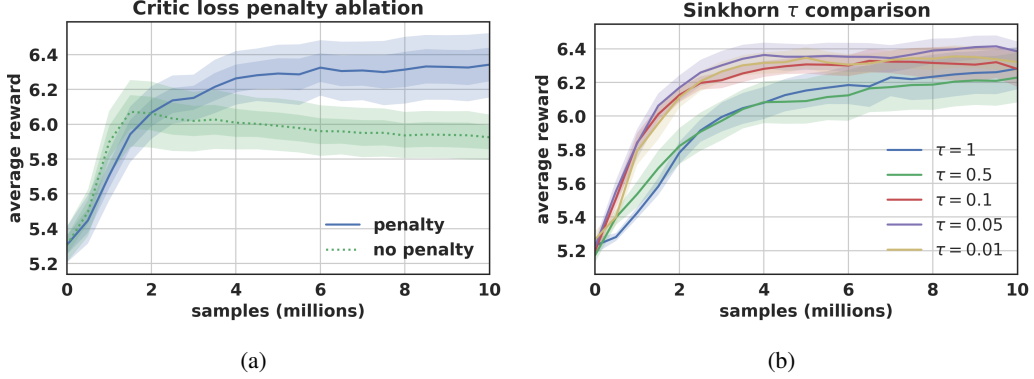
Figure 3: a) Average reward results on MWM-10 with and without the critic penalty term. With the penalty term, the bias from the continuous relaxation is mostly removed, allowing the actor to keep improving as it sees more data. b) Performance on MWM-10 with various Sinkhorn layer temperatures. We observed that as τ decreases, the agent can achieve higher average rewards with lower variance. 68% and 95% CI bands are shown for (a), and 68% CI bands are shown for (b).

For all experiments, 500K problem instances are generated for training, along with 1K held-out samples for evaluating the target policy. The target policy is evaluated after each pass of the training set by computing the mean reward over the test set. We compute the mean and/or median of the best test set scores over 10 random seeds. For hyperparameter tuning and ablation studies, we used two separate validation set of MWM problem instances with N=10 (MWM-10). Full experiment and hyperparameter details can be found in Appendix D.

Before presenting results, we will formally introduce the MWM task, which may be less familiar than sorting and the Euclidean TSP. $2N$ points are uniformly sampled in $\mathbb{R}^{[0,1] \times [0,1]}$ to create a bipartite graph $G = (V_1, V_2, E)$, with $|V_1| = N$, $|V_2| = N$, and $|E| = N^2$, and each vertex represented by a point (x_i, y_i). The objective is to find an $N \times N$ permutation corresponding to a matching between pairs of vertices from V_1 and V_2 that maximizes the sum of the Euclidean distances. The action $\mathbf{P}$ selected by the actor is used to permute the vertices of V_2 to create a candidate matching. We define an optimality ratio as $\frac{\text{predicted matching weight}}{\text{optimal matching weight}} \in [0, 1]$ to assess performance; optimal matchings can be computed with the Hungarian algorithm. We emphasize that SPG is designed to solve more complex problems than MWM where labels are not easily available (e.g., data association). In these scenarios, SPG must learn to solve the problem from raw inputs and a reward function that may not accurately specify the desired behavior. MWM is useful as a benchmark not only because it is challenging for SPG and the baseline methods, but also because we can exactly evaluate performance through optimality ratios.

4.2 Ablation and sensitivity studies

To improve our understanding of SPG, we conducted the following studies on MWM-10. First, we compared the performance of SPG with and without the added penalty term to the critic loss. Then, we evaluated SPG with and without exploration. Finally, we considered various values for the temperature parameter τ.

Critic penalty ablation study. We performed an ablation study to investigate the effect of the critic loss penalty term. The Q-values for the discrete actions $\mathbf{P}$, which we called "hard Q-values", accurately predict the immediate rewards. However, without the penalty term, the "soft Q-values" for continuous actions $\mathbf{M}$ diverge. With the penalty term, both the hard and soft Q-values accurately predict the reward (Figure 2, bottom row). The penalty term also allows SPG to train for much longer before learning saturates (Figure 3a).

Exploration study. We evaluated SPG with ϵ-greedy exploration and the 2-exchange neighborhood heuristic to assess its efficacy. In this experiment, ϵ is initialized to either one or zero; when initialized to one, we tried linearly decaying ϵ by 95% and 90% after every epoch until it reached 0.01. We found that the difference in average reward amongst the considered exploration schemes was insignificant,

Table 1: Mean KT on the sorting task.

	N=20	N=50
PN-AC	1.0 ± 0	0.998 ± 0.001
SPG+SEQUENTIAL	0.998 ± 0.002	0.984 ± 0.006

Table 2: Mean tour length on the TSP-20 task.

	AVG. TOUR LENGTH
OPTIMAL	3.83
[16]	3.84
PN-AC	3.89
CHRISTOFIDES	4.30
SPG+SEQUENTIAL	4.62

possibly because the tasks we consider have dense reward functions. However, $\epsilon = 1$ and the decay rate of 0.95 exhibited the lowest variance for the final average reward after 20 epochs compared to no exploration ($F = 4.402$ with $N_1 = N_2 = 10$ and $p = 0.019$), and hence use this exploration strategy in our experiments. Other starting values for ϵ between zero and one were considered, but were strictly worse.

Sinkhorn τ sensitivity analysis. To ascertain SPG's sensitivity to the Sinkhorn temperature parameter τ, we compared SPG's performance on MWM-10 for values in $\{1, 0.5, 0.1, 0.05, 0.01\}$. Results are displayed in Figure 3b. Smaller values of τ corresponded to higher average rewards. However, we observed diminishing returns for $\tau < 0.05$, most likely due to increased variance in the policy gradient estimate. We use $\tau = 0.05$ with $L = 10$ in the remainder of the experiments.

4.3 Main results

Baselines. We compare SPG's performance on sorting and the Euclidean TSP against the greedy RL pretraining model from [2]. In our experiments, this model is referred to as Pointer-Network Actor-Critic (PN-AC). On the Euclidean TSP, we also compare against [16], an extension of [2] that replaces the encoder with a graph attention layer.

We modify PN-AC to use SPG+Matching's bipartite graph embedding layer for the MWM task; we call this model PN-AC+Matching. We also tried removing PN-AC+Matching's autoregressive decoder to create a simple RL baseline, called AC+Matching. It uses a softmax to sequentially sample match pairs directly from the encoder output. Implementation details for all baseline models are provided in Appendix A.

Sorting. SPG is trained to predict the permutation that, when applied to a list of integers, returns them in sorted order. The reward function is the Kendall-Tau (KT) correlation coefficient, which provides dense rewards. A KT score of 1.0 means the list was perfectly sorted. We train SPG+Sequential and PN-AC on lists of size $N = \{20, 50\}$. For PN-AC, we use the same hyperparameters as reported in [2], except that we needed to lower the learning rate to 1e-4.

The mean highest attained KT scores on the test set are presented in Table 1. Both SPG and PN-AC are able to learn to solve this task.

MWM. For this set of experiments, we trained SPG+Matching, PN-AC+Matching, and AC+Matching on MWM with $N = \{15, 20, 25\}$. As a point of reference, the performance of an untrained SPG+Matching policy (SPG+Random) is also provided. Results in Table 3 are the medians of the best optimality ratios achieved on the test set over all 10 random seeds. Unlike PN-AC+Matching, AC+Matching is able to do better than random (Figure 4), from which we conclude that the pointer network decoder is not suitable for this task. SPG+Matching is more data efficient and scales better with larger N than AC+Matching, which implies that the representations learned by SPG+Matching are well-suited to matching problems.

TSP. To demonstrate SPG's performance on an NP-Hard problem, we trained SPG+Sequential on the Euclidean TSP for $N = 20$. To generate each TSP instance, points are uniformly sampled from the unit square. The reward function is the negated sum of the Euclidean distances between each stop along the tour. In Table 2, we present mean tour lengths on the test. The tour lengths for the baselines are reported from [2, 16].
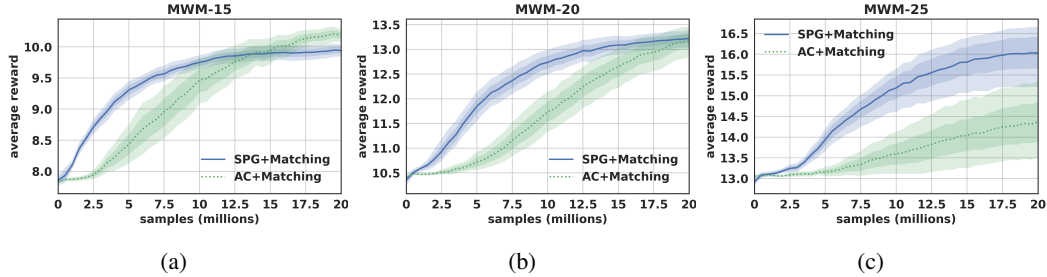
Figure 4: SPG+Matching and AC+Matching average reward curves on the MWM test set. SPG+Matching is more data efficient on all three problem sizes, and achieves a higher average reward on MWM-20 and MWM-25. CI bands for 68% and 95% are shown.

Table 3: Median optimality ratios on the MWM test set.

	N=15	N=20	N=25
PN-AC+Matching	0.714	0.708	0.703
AC+Matching	**0.935**	**0.897**	0.725
SPG+Random	0.698	0.707	0.698
SPG+Matching	0.904	**0.895**	**0.889**

5 Related works

The use of learning algorithms for solving problems involving permutations has a rich history. One of the earliest and most influential papers in this line of research, by Hopfield and Tank [12], describes how to use Hopfield nets to approximately solve instances of the TSP. Due to the ubiquity of permutations, research on this topic continued, with algorithms based on Sinkhorn balancing and continuous relaxations appearing in [17, 9, 11]. The work of [1] laid the groundwork for combining deep learning with learning over permutations [20, 6, 22]. DeepPermNet, proposed in [6], is a convolutional neural network augmented with a Sinkhorn layer for solving problems in computer vision with supervised learning such as comparing sets of images based on their attributes. The DeepPermNet work is complementary to ours, in that a similar convolutional neural network-based architecture can be trained with SPG to solve permutation-based vision tasks.

A supervised-learning approach to the MWM problem is proposed in [24]. They use a single LSTM to implement a greedy algorithm that sequentially outputs a probability distribution over potential matches one vertex at a time, not unlike the AC+Matching architecture we used as an RL baseline. They show that it can solve a simple data association task for tracking five targets.

The S2V-DQN algorithm from [8] uses the `struct2vec` [7] algorithm to embed an input graph of a combinatorial problem, which gets passed to a Q-network. For the TSP, we noted that it uses both the edge lengths and well as the vertex coordinates to build its powerful graph representation. PN-AC and SPG both learn their own state representations from just vertex coordinates, and hence both start out producing essentially random tours, unlike S2V-DQN.

6 Conclusion

We introduced the SPG algorithm for solving combinatorial optimization problems involving permutations. Our algorithm is able to solve all tasks under consideration reasonably well and demonstrated its ability to learn strong representations for policies over permutations. A current limitation of SPG is that generalizing to different problem sizes requires retraining. This can partially be addressed with an inductive graph-embedding layer [10] that removes the embedding layer's dependency on the dimension of the problem size, possibly allowing for positive transfer to larger problem sizes. We note that there are many other ways to potentially improve SPG's performance that are orthogonal to the contributions of this paper, such as incorporating prioritized experience replay [31] or parameter noise exploration [29].

References

[1] Adams, Ryan Prescott and Zemel, Richard S. Ranking via sinkhorn propagation. *arXiv:1106.1925 [stat.ML]*, 2011.

[2] Bello, Irwan, Pham, Hieu, Le, Quoc V, Norouzi, Mohammad, and Bengio, Samy. Neural combinatorial optimization with reinforcement learning. In *Workshop Track of the International Conference on Learning Representations*, 2017.

[3] Bengio, Yoshua, Léonard, Nicholas, and Courville, Aaron. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv:1308.3432 [cs.LG]*, 2013.

[4] Caetano, Tibério S, McAuley, Julian J, Cheng, Li, Le, Quoc V, and Smola, Alex J. Learning graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(6):1048–1058, 2009.

[5] Cao, Zhe, Qin, Tao, Liu, Tie-Yan, Tsai, Ming-Feng, and Li, Hang. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning (ICML'07)*, pp. 129–136. ACM, 2007.

[6] Cruz, Rodrigo Santa, Fernando, Basura, Cherian, Anoop, and Gould, Stephen. Deeppermnet: Visual permutation learning. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17)*, 2017.

[7] Dai, Hanjun, Dai, Bo, and Song, Le. Discriminative embeddings of latent variable models for structured data. In *Proceedings of the 33rd International Conference on Machine Learning (ICML'16)*, pp. 2702–2711, 2016.

[8] Dai, Hanjun, Khalil, Elias B, Zhang, Yuyu, Dilkina, Bistra, and Song, Le. Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems*, 2017.

[9] Gold, Steven and Rangarajan, Anand. Softmax to softassign: Neural network algorithms for combinatorial optimization. *Journal of Artificial Neural Networks*, 2(4):381–399, 1996.

[10] Hamilton, Will, Ying, Zhitao, and Leskovec, Jure. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1025–1035, 2017.

[11] Helmbold, David P and Warmuth, Manfred K. Learning permutations with exponential weights. *Journal of Machine Learning Research*, 10(Jul):1705–1736, 2009.

[12] Hopfield, John J and Tank, David W. Neural computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.

[13] Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of 32nd International Conference on Machine Learning (ICML'15)*, pp. 448–456, 2015.

[14] Jang, Eric, Gu, Shixiang, and Poole, Ben. Categorical reparameterization with gumbel-softmax. *arXiv:1611.01144 [stat.ML]*, 2016.

[15] Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv:1412.6980 [cs.LG]*, 2014.

[16] Kool, WWM and Welling, M. Attention solves your tsp. *arXiv:1803.08475 [stat.ML]*, 2018.

[17] Kosowsky, JJ and Yuille, Alan L. The invisible hand algorithm: Solving the assignment problem with statistical physics. *Neural networks*, 7(3):477–490, 1994.

[18] Kuhn, H. W. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. ISSN 1931-9193. doi: 10.1002/nav.3800020109. URL http://dx.doi.org/10.1002/nav.3800020109.

[19] Lillicrap, Timothy P, Hunt, Jonathan J, Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. *arXiv:1509.02971 [cs.LG]*, 2015.

[20] Linderman, Scott W, Mena, Gonzalo E, Cooper, Hal, Paninski, Liam, and Cunningham, John P. Reparameterizing the birkhoff polytope for variational permutation inference. *arXiv:1710.09508 [stat.ML]*, 2017.

[21] Maddison, Chris J., Mnih, Andriy, and Teh, Yee Whye. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017.

[22] Mena, Gonzalo, Belanger, David, Linderman, Scott, and Snoek, Jasper. Learning latent permutations with gumbel-sinkhorn networks. In *International Conference on Learning Representations*, 2018.

[23] Milan, Anton, Rezatofighi, Seyed Hamid, Dick, Anthony R, Reid, Ian D, and Schindler, Konrad. Online multi-target tracking using recurrent neural networks. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*, pp. 4225–4232, 2017.

[24] Milan, Anton, Rezatofighi, Seyed Hamid, Garg, Ravi, Dick, Anthony R, and Reid, Ian D. Data-driven approximations to np-hard problems. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*, pp. 1453–1459, 2017.

[25] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[26] Munkres, James. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[27] Nowak, Alex, Villar, Soledad, Bandeira, Afonso S, and Bruna, Joan. A note on learning algorithms for quadratic assignment with graph neural networks. *arXiv:1706.07450 [cs.LG]*, 2017.

[28] Paszke, Adam, Gross, Sam, Chintala, Soumith, Chanan, Gregory, Yang, Edward, DeVito, Zachary, Lin, Zeming, Desmaison, Alban, Antiga, Luca, and Lerer, Adam. Automatic differentiation in pytorch. 2017.

[29] Plappert, Matthias, Houthooft, Rein, Dhariwal, Prafulla, Sidor, Szymon, Chen, Richard Y, Chen, Xi, Asfour, Tamim, Abbeel, Pieter, and Andrychowicz, Marcin. Parameter space noise for exploration. *arXiv:1706.01905 [cs.LG]*, 2017.

[30] Resende, Mauricio G.C. and Ribeiro, Celso C. *Optimization by GRASP*. Springer New York, 2016. doi: 10.1007/978-1-4939-6530-4.

[31] Schaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. Prioritized experience replay. *arXiv:1511.05952 [cs.LG]*, 2015.

[32] Silver, David, Lever, Guy, Heess, Nicolas, Degris, Thomas, Wierstra, Daan, and Riedmiller, Martin. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML'14)*, pp. 387–395, 2014.

[33] Sinkhorn, Richard. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The Annals of Mathematical Statistics*, 35(2):876–879, 1964.

[34] Vinyals, Oriol, Bengio, Samy, and Kudlur, Manjunath. Order matters: Sequence to sequence for sets. *arXiv:1511.06391 [stat.ML]*, 2015.

[35] Vinyals, Oriol, Fortunato, Meire, and Jaitly, Navdeep. Pointer networks. In *Advances in Neural Information Processing Systems*, pp. 2692–2700, 2015.

[36] Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pp. 5–32. Springer, 1992.

[37] Wojke, Nicolai, Bewley, Alex, and Paulus, Dietrich. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 3645–3649, 2017.

A Actor-critic architectures

A.1 SPG+Matching

We begin by explaining the SPG+Matching actor network architecture for $\mathcal{S} = \mathbb{R}^{N \times K} \times \mathbb{R}^{N \times K}$. Let N be the number of objects in each of the two disjoint sets of vertices in a bipartite graph and suppose $\mathbf{X}_1, \mathbf{X}_2 \in \mathbb{R}^{N \times K}$ is a representation of the bipartite graph with feature dimension K. For planar graphs, as in MWM, $K = 2$. The parameters for the nonlinear embedding of $\mathbf{X}_1$ and $\mathbf{X}_2$, $\{\mathbf{W}_e, \mathbf{b}_e\}$ are shared,

$$
\begin{aligned}
\mathbf{E}_1 &= \sigma\big(\mathbf{X}_1 \mathbf{W}_e + \mathbf{b}_e\big) \\
\mathbf{E}_2 &= \sigma\big(\mathbf{X}_2 \mathbf{W}_e + \mathbf{b}_e\big)
\end{aligned}
\tag{8}
$$

with $\mathbf{W}_e \in \mathbb{R}^{K \times 128}$ and $\mathbf{b}_e \in \mathbb{R}^{128}$. All nonlinearities σ are LeakyReLUs with negative slope 0.01. Next, the matrix outer product of the embeddings is computed

$$
\mathbf{E} = \mathbf{E}_2 \mathbf{E}_1^\intercal.
\tag{9}
$$

$\mathbf{E}$ is split in the zeroth dimension to form an N-dimensional sequence of length N. Next, this sequence gets passed to a GRU layer:

$$
\mathbf{h}_N = \mathrm{GRU}(\mathbf{E}, \mathbf{h}_0).
\tag{10}
$$

The GRU has parameters $\mathbf{W}_i \in \mathbb{R}^{3N \times 128}, \mathbf{b}_i \in \mathbb{R}^{3*128}, \mathbf{W}_h \in \mathbb{R}^{3N \times 128}, \mathbf{b}_h \in \mathbb{R}^{3*128}$, and the output of the GRU is $\mathbf{h}_N \in \mathbb{R}^{N \times 128}$.

We map $\mathbf{h}_N$ to an $N \times N$ matrix with a linear layer,

$$
\mathbf{Y}_a = \mathbf{h}_N \mathbf{W}_a + \mathbf{b}_a
\tag{11}
$$

for $\mathbf{W}_a \in \mathbb{R}^{128 \times N}$ and $\mathbf{b}_a \in \mathbb{R}^N$. Next, $\mathbf{Y}_a$ is processed by the Sinkhorn layer (Figure 5); in our experiments, we use $\tau = 0.05$ and $L = 10$. The Hungarian algorithm is applied to round the output of the Sinkhorn layer to a permutation matrix. We parallelize the Hungarian algorithm by splitting the mini-batch across multiple processors.

The critic network for SPG+Matching has a similar architecture as the actor network up to Equation 11. Instead of the linear layer, we map the hidden state of the GRU back to the embedding dimension with a nonlinear layer and BatchNorm [13],

$$
\mathbf{Y}_c = \sigma\big(\mathrm{BN}(\mathbf{h}_N \mathbf{W}_c + \mathbf{b}_c)\big).
\tag{12}
$$

for $\mathbf{W}_c \in \mathbb{R}^{128 \times 128}$ and $\mathbf{b}_c \in \mathbb{R}^{128}$. The critic also takes as input the action $\mathbf{P}$, which we combine with the learned representation of the state $\mathbf{Y}_c$ as follows:

$$
\begin{aligned}
\mathbf{E}_d &= \sigma\big(\mathrm{BN}(\mathbf{P}\mathbf{W}_d + \mathbf{b}_d)\big) \\
\mathbf{Y}_f &= \sigma\big(\mathrm{BN}\big((\mathbf{Y}_c + \mathbf{E}_d)\mathbf{W}_f + \mathbf{b}_f\big)\big).
\end{aligned}
\tag{13}
$$

with $\mathbf{W}_d \in \mathbb{R}^{N \times 128}$ and $\mathbf{b}_d \in \mathbb{R}^{128}$, and $\mathbf{W}_f \in \mathbb{R}^{128 \times N}$ and $\mathbf{b}_f \in \mathbb{R}^N$. We found that BatchNorm helps the critic network combine the state and action effectively. The fused state and action embedding $\mathbf{Y}_f \in \mathbb{R}^{N \times N}$ is mapped to a scalar by

$$
Q = (\mathbf{Y}_f \mathbf{W}_{g_1})^\intercal \mathbf{W}_{g_2}
\tag{14}
$$

where $\mathbf{W}_{g_1}, \mathbf{W}_{g_2} \in \mathbb{R}^{N \times 1}$.

A.2 SPG+Sequential

The SPG+Sequential architecture is used for problems where $\mathcal{S} = \mathbb{R}^{N \times K}$ is a set of N objects, and the agent selects an action that permutes its own input. For the most part, the SPG+Sequential actor network mirrors the actor architecture from SPG+Matching. Rather than reproduce most of Equations 8-12 here, we will just point out the few differences. First, the SPG+Sequential actor only needs to embed one set of objects $\mathbf{X}$, so there is only one embedding computed instead of two (Equation 8). Second, this embedding is passed directly to the GRU; there is no matrix outer product as in Equation 9. The GRU in SPG+Sequential is bidirectional, which we found helped improve the overall performance on TSP-20.

SPG+Sequential's critic network differs from SPG+Matching's critic network in that the action is fused with the state *before* the recurrent layer. This performed significantly better for the tasks we considered; we believe that this is due to the prominent sequential nature of both the states and actions for the sorting and TSP tasks. The critic can be defined for a state and action pair $(\mathbf{X}, \mathbf{P})$ as

$$\mathbf{E}_1 = \sigma\big(\mathrm{BN}(\mathbf{X}\mathbf{W}_j + \mathbf{b}_j)\big) \tag{15}$$

$$\mathbf{E}_2 = \sigma\big(\mathrm{BN}(\mathbf{P}\mathbf{W}_k + \mathbf{b}_k)\big) \tag{16}$$

$$\mathbf{E} = \sigma\big(\mathrm{BN}((\mathbf{E}_1 + \mathbf{E}_2)\mathbf{W}_l + \mathbf{b}_l)\big) \tag{17}$$

$$\mathbf{h}_N = \mathrm{GRU}(\mathbf{E}, \mathbf{h}_0) \tag{18}$$

$$\mathbf{Y}_m = \sigma(\mathbf{h}_N \mathbf{W}_m + \mathbf{b}_m) \tag{19}$$

$$Q = (\mathbf{Y}_m \mathbf{W}_{n_1})^{\mathsf{T}} \mathbf{W}_{n_2}. \tag{20}$$

The critic parameters are $\mathbf{W}_j \in \mathbb{R}^{K \times 128}, \mathbf{b}_j \in \mathbb{R}^{128}, \mathbf{W}_k \in \mathbb{R}^{N \times 128}, \mathbf{b}_k \in \mathbb{R}^{128}, \mathbf{W}_l \in \mathbb{R}^{128 \times 128}, \mathbf{b}_l \in \mathbb{R}^{128}, \mathbf{W}_m \in \mathbb{R}^{256 \times 128}, \mathbf{b}_m \in \mathbb{R}^{128}, \mathbf{W}_{n_1} \in \mathbb{R}^{128 \times 1}, \mathbf{W}_{n_2} \in \mathbb{R}^{128 \times 1}$, plus the GRU parameters with dimension 256 (2x 128 for bidirectional GRU).

Sample code for implementing a Sinkhorn layer. In Figure 5, we show example code for a stable implementation of the Sinkhorn layer. In our experimentation, a naive implementation occasionally caused numerical errors even for τ values as large as 0.1.

A.3 PN-AC

For the sorting task, we used an implementation of PN-AC that is based on the greedy RL pretraining model from [2]. We briefly describe the architecture here, but refer the reader to [2] for more details. The actor network in PN-AC uses an LSTM encoder and decoder with an attention mechanism to sequentially "point" at elements from the input sequence to greedily construct solutions. This algorithm is able to produce solutions that satisfy the permutation constraints by hard-coding a rule that prevents the attention mechanism from selecting the same element from the input twice. Instead of a critic network, we found that an exponential moving average made a suitable baseline for PN-AC's REINFORCE policy gradient estimator. Crucially, PN-AC is designed to perform well on combinatorial problems where the output is a permutation or subset of the input, and both the input and output have an underlying sequential structure.

A.4 PN-AC+Matching

We modified the PN-AC architecture to train it on the MWM task. In PN-AC, the length N input sequence $\mathbf{X} \in \mathbb{R}^{N \times K}$ representing the state gets embedded by a linear transformation and then processed by an encoder network. PN-AC+Matching instead adopts the shared dual-embedding of SPG+Matching to transform two inputs $\mathbf{X}_1, \mathbf{X}_2 \in \mathbb{R}^{N \times K}$ into the embedding $\mathbf{E} \in \mathbb{R}^{N \times N}$ from Equation 9. As in SPG+Matching, $\mathbf{E}$ is split in the zeroth dimension into a length N sequence of N-dimensional vectors, which gets processed by an identical encoder network to the one used by PN-AC. The output of the encoder, a length N sequence of 128-dimensional vectors, is passed to a decoder network as the attention context. The decoder network is mostly unchanged from the one used by PN-AC. As with PN-AC, the initial decoder input is a trainable parameter. We concatenate the embeddings $\mathbf{E}_1$ and $\mathbf{E}_2$ from Equation 8 to use as inputs to the decoder, and hence the decoder

```
def SinkhornLayer(x, tau, L, eps=1e-6):
    # x: input features with shape [N,N]
    # tau: temperature parameter
    # L: number of Sinkhorn iters

    x = x / tau
    for _ in range(L):
        # row normalization
        x = x - LogSumExp(x, dim=1, keepdims=True)
        # column normalization
        x = x - LogSumExp(x, dim=0, keepdims=True)

    # add a small offset 'eps' to avoid numerical
    # errors due to exp()
    return exp(x) + eps
```

Figure 5: Generic Python code for a stable implementation of the Sinkhorn layer. The `dim` argument of `LogSumExp` indicates which axis to sum over.

input dimension is 256. At each time step $t = 1, ..., N$, the pointer mechanism selects the ith node, $i = 1, ..., N$, from $\mathbf{X}_2$. The ith element of $\mathbf{E}_2$ is concatenated with the tth element of $\mathbf{E}_1$ to use as the input to the decoder at the next time step. The order in which the elements of $\mathbf{E}_2$ are selected dictates the permutation applied to $\mathbf{X}_2$ after the decoding is complete, which is used to form the matching solution.

In summary, the main difference between PN-AC and PN-AC+Matching is the embedding process for the input, which is identical to the one used by SPG+Matching.

A.5 AC+Matching

To create a simpler (albeit more effective) baseline than PN-AC+Matching, we experimented with removing the autoregressive decoder. This architecture also uses the same embedding process as PN-AC+Matching and SPG+Matching. Instead of using a decoder, it simply takes the fused embedding $\mathbf{E}$, splits it along the zeroth dimension, and uses each N-dimensional vector as input to an LSTM. At each time step, the output of the LSTM $\mathbf{h}_t \in \mathbb{R}^{128}$ is transformed by

$$\mathbf{l}_t = C \tanh(\mathbf{h}_t \mathbf{W}_p + \mathbf{b}_p), \tag{21}$$

where $C = 10$, $\mathbf{W}_p \in \mathbb{R}^{128 \times N}$, and $\mathbf{b}_p \in \mathbb{R}^N$. The logits $\mathbf{l}_t \in \mathbb{R}^N$ are masked, as is done in PN-AC and PN-AC+Matching, to keep inputs from being selected more than once by the pointer mechanism. A normalized softmax operation is applied to l to create a multinomial distribution over the available inputs, from which one input element is stochastically selected. Similarly to PN-AC and PN-AC+Matching, AC+Matching is trained with REINFORCE and uses an exponential moving-average baseline.

B Algorithm

The training algorithm we use for SPG is given in Algorithm 1. A replay buffer R is maintained for experience replay; we uniformly sample from R to construct mini-batches for computing gradient estimations. Each epoch lasts for a pre-determined number of training steps, after which the target policy is evaluated on a held-out test set.

C Geometric interpretation of the critic loss penalty term

In this section, we provide a geometric interpretation of the penalty term used to remove bias due to the continuous relaxation. First, we define the *Birkhoff polytope* $\mathcal{B}_N$ as the set of all $N \times N$ doubly-stochastic matrices. All points of $\mathcal{B}_N$ lie on the convex hull of $\mathcal{P}_N$, i.e., the vertices of the convex hull are $N \times N$ permutations. Therefore, an SPG policy π_θ outputs an element $\mathbf{M} \in \mathcal{B}_N$, which gets mapped to the nearest vertex of $\mathcal{B}_N$ by the non-differentiable Hungarian method, i.e., $\mathbf{P} = \text{H}(\pi_\theta) = \text{H}(\mathbf{M})$. We provide a visual representation of $\mathcal{B}_3$ in Figure 6a as an example.

Algorithm 1 Sinkhorn Policy Gradient

Initialize actor $\pi_\theta(s; \tau)$ and critic $Q_{\theta'}(s, a)$
Initialize replay buffer R
for $i = 1$ **to** max_train_steps **do**
    Sample state $s \sim \rho$
    $\mathbf{M} = \pi_\theta(s; \tau)$
    $\mathbf{P} = \mathrm{H}(\mathbf{M})$
    Sample $u \sim \mathrm{Uniform}[0, 1]$
    if $u < \epsilon$ **then**
        Make $k = 2$ random row exchanges for $\mathbf{P}$ and $\mathbf{M}$
    end if
    Apply $\mathbf{P}$ to s and observe $r(s, \mathbf{P})$
    Store experience $(s, \mathbf{M}, \mathbf{P}, r)$ in R
    Sample mini-batch $(s_n, \mathbf{M}_n, \mathbf{P}_n, r_n) \sim R$
    Update critic by minimizing:

$$\mathrm{MSE}\big(r_n, Q_{\theta'}(s_n, \mathbf{P}_n)\big) + \mathrm{MSE}\big(\texttt{stop\_grad}(Q_{\theta'}(s_n, \mathbf{P}_n)), Q_{\theta'}(s_n, \mathbf{M}_n)\big)$$

    $\mathbf{M'_n} = \pi_\theta(s_n; \tau)$
    Update the actor policy by ascending the sampled policy gradient:

$$\nabla_\theta \pi_\theta \approx \frac{1}{N} \sum_n \nabla_\theta \pi_\theta(s_n; \tau) \nabla_a Q_{\theta'}(s_n, a)\big|_{a=\mathbf{M'_n}}$$

end for

The Q-values for the discrete actions $Q_{\theta'}(s, \mathbf{P})$ are piece-wise constant over $\mathcal{B}_N$, which is why $\nabla_a Q_{\theta'}(s, a)\big|_{a=\mathbf{P}}$ is zero almost everywhere (and undefined at the discontinuities between the regions dividing $\mathcal{B}_N$). Next, we will describe how we can obtain a useful critic action-gradient that accurately approximates the direction of maximum reward improvement.

We can measure how close the Q-values for the relaxed permutations $\mathcal{M}$ match those for $\mathcal{P}$ with the following:

$$\epsilon = \int_\mathcal{S} \big(Q_{\theta'}(s, \mathbf{P}) - Q_{\theta'}(s, \mathbf{M})\big)^2 \mathrm{d}s.$$

Figure 6b shows the Q-values corresponding to $\mathbf{P}$ and $\mathbf{M}$ for fixed $s \in \mathcal{S}$ in $\mathcal{B}_3$. By treating this as a regression problem with loss function ϵ and targets $Q_{\theta'}(s, \mathbf{P})$, we fit a smooth approximation to the piece-wise constant action-value function induced by $\mathbf{P}$. The critic action-gradient $\nabla_a Q_{\theta'}(s, a)\big|_{a=\mathbf{M}}$, depicted in Figure 6b, is well-defined everywhere in $\mathcal{B}_N$ since $Q_{\theta'}(s, \mathbf{M})$ is continuous and differentiable by construction. Furthermore, the critic action-gradient clearly approximates the direction of reward improvement at the region boundaries in $\mathcal{B}_N$, and the quality of the approximation improves as $\epsilon \to 0$.

In practice, we can minimize ϵ by minimizing the mean squared error between $Q_{\theta'}(s, \mathbf{P})$ and $Q_{\theta'}(s, \mathbf{M})$, which we estimate with a mini-batch of samples s drawn from ρ as $\mathrm{MSE}\big(\texttt{stop\_grad}(Q_{\theta'}(s, \mathbf{P})), Q_{\theta'}(s, \mathbf{M})\big)$. Updating θ' with the gradient of ϵ and a small enough step size guarantees that ϵ will decrease and eventually settle in some local minima. Even though the targets use the same parameters as $Q_{\theta'}(s, \mathbf{M})$, we hold them fixed with $\texttt{stop\_grad}$ and we observe that they become stable as the critic converges.

D Experiment details

Details about the deep network architectures for SPG and the baselines are provided in Appendix A. We use Adam [15] to optimize all models. We arrived at the hyperparameters listed in Table 4 using the SigOpt Bayesian optimization service (except for Adam's parameters, which are the defaults) on
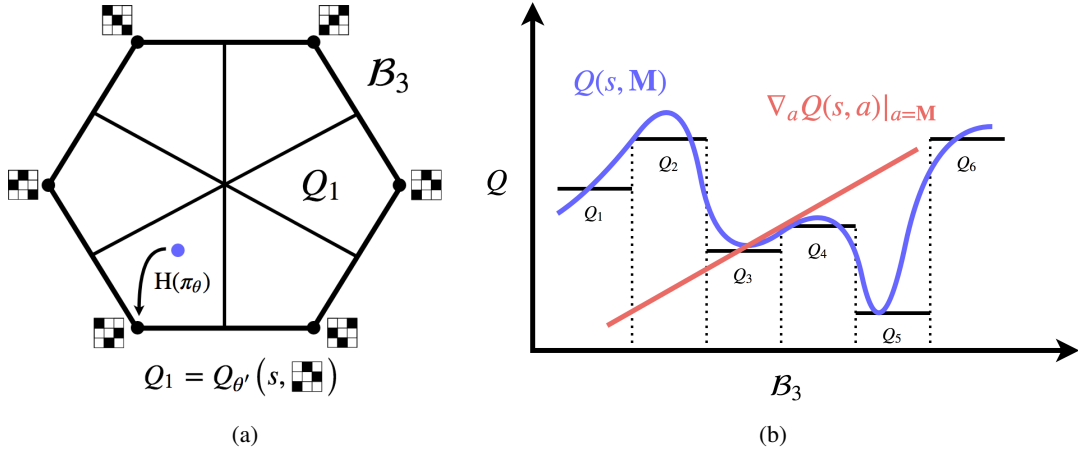
Figure 6: a) Visual representation of the Birkhoff polytope with $N = 3$ ($\mathcal{B}_3$) as a hexagon, for some fixed $s \in \mathcal{S}$. The points in $\mathcal{B}_3$ are 3×3 doubly-stochastic matrices; the vertices of the $\mathcal{B}_3$/hexagon are 3×3 permutation matrices. All elements of $\mathcal{B}_3$ within a region are sent to the vertex contained in that region by the Hungarian method, and hence have the same Q-value, e.g., Q_1. b) In this figure, the x-axis is divided into the 6 regions of $\mathcal{B}_3$, as if the hexagon from (a) was cut to the center along one of the region boundaries and laid out along the real line. This is not a technically accurate depiction of the Q-values over $\mathcal{B}_3$, but it is useful to acquire intuition. The Q-values for discrete actions, i.e. permutations, are piece-wise constant (Q_{1-6}) and the Q-values for the continuous actions form a smooth, differentiable surface. The penalty term tries to minimize the squared error between the two so that the action-gradient of the smooth surface provides an accurate indication of the direction of maximum reward improvement at region boundaries.

a validation set of 500K instances of MWM-10. These hyperparameters are shared across all tasks in our experiments for both SPG+Matching and SPG+Sequential.

For the number of Sinkhorn iterations L, we considered $L \in \{5, 10, 15\}$ and found that 10 offered the best trade-off between having too few iterations (and hence the matrix $\mathbf{M}$ doesn't satisfy the permutation sum-to-one constraints for both the rows and columns), or too many iterations which causes the gradients to vanish.

All experiments are run with pure PyTorch [28] and a single NVIDIA GPU (we used a GTX 1080 and a server with Titan Xp's and a Titan V). We note that we observed a significant speed up when splitting the Hungarian algorithm computation over across multiple cores. In our implementation, we split batches of problem instances of size 128 across four cores. Code for reproducing the results from this paper will be available online.

E Analysis

We observed that MWM becomes increasingly difficult to solve as N increases and the points get more densely crowded; basically, the differences in the matching weights between candidate matchings get arbitrarily small. Additionally, SPG can find a matching that is within 0.003% of the optimal weight but use none of the correct pairs from the optimal solution (Figure 7). This suggests that reward shaping in the form of adding auxiliary objectives may be necessary to help solve these issues of identifiability for certain tasks.

15

Table 4: Hyperparameters used in our main experiments.

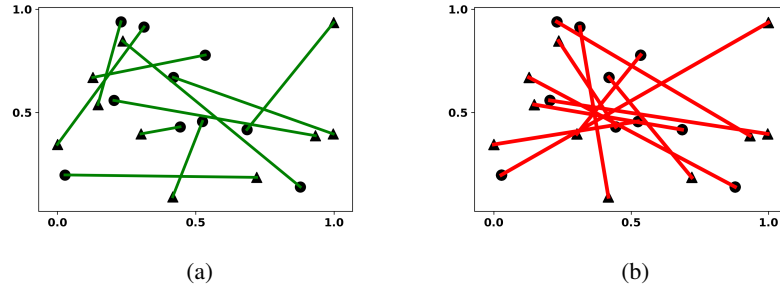| NAME	VALUE
OPTIMIZATION	
ACTOR LR	1E-5
ACTOR LR DECAY RATE	5%/5K STEPS
CRITIC LR	2E-4
CRITIC LR DECAY RATE	5%/5K STEPS
ADAM β_1	0.9
ADAM β_2	0.999
ADAM ϵ	1E-8
EXPLORATION	
k	2
ϵ-START	1.0
ϵ-END	0.01
ϵ-DECAY	5%/EPOCH
SINKHORN LAYER	
τ	0.05
L	10
TRAINING	
MINI-BATCH SIZE	128
REPLAY BUFFER SIZE	1E6
L2 GRADIENT NORM CLIPPING	1

(a)   (b)

Figure 7: (a) The optimal matching for an instance of MWM-10. (b) A matching produced by SPG where the matching weight has an optimality ratio of 0.997. However, none of the selected match pairs are found in the optimal solution.