

## 541 A Environments

542 **Sokoban.** Sokoban is a well-known puzzle where the player must push boxes onto target locations  
 543 within a confined grid. Its high combinatorial complexity and PSPACE-hard nature [12] make it a  
 544 benchmark for both classical planning and deep learning methods. Sokoban challenges algorithms to  
 545 balance search efficiency and long-term planning. In our experiments, we use 12×12 Sokoban boards  
 546 with four boxes.

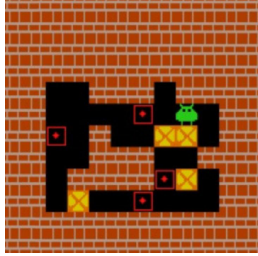


Figure 10: An example instance of Sokoban.



Figure 11: An example instance of Rubik's Cube.

547 **Rubik's Cube.** The Rubik's Cube is a 3D combinatorial puzzle with over  $4.3 \times 10^{19}$  possible  
 548 configurations, making it an iconic testbed for algorithms tackling massive search spaces. Solving  
 549 the Rubik's Cube requires sophisticated reasoning and planning, as well as the ability to navigate  
 550 high-dimensional state spaces efficiently. Recent advances in using neural networks for solving  
 551 this puzzle, such as [1], highlight the potential of deep learning in handling such computationally  
 552 challenging tasks.

553 **N-Puzzle.** The N-Puzzle is a sliding-tile puzzle with variants like the 8-puzzle (3×3 grid), 15-puzzle  
 554 (4×4 grid), and 24-puzzle (5×5 grid). The objective is to rearrange tiles into a predefined order by  
 555 sliding them into an empty space. It serves as a classic benchmark for testing algorithms' planning  
 556 and search efficiency. The problem's difficulty scales with puzzle size, requiring effective heuristics  
 557 for solving larger instances.

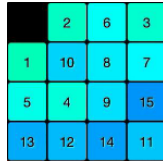


Figure 12: An example instance of N-Puzzle.

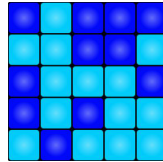


Figure 13: An example instance of Lights Out.



Figure 14: An example instance of Digit Jumper.

558 **Lights Out** The Lights Out is a single-player game invented in 1995. It is a grid-based game where  
 559 each cell (or *light*) can either be on or off. Pressing a cell flips its state and those of its immediate  
 560 neighbors (above, below, left, and right). Corner and edge lights have fewer neighbors and therefore  
 561 affect fewer lights. The goal is to press the lights in a strategic order to turn off all the lights on the  
 562 grid.

563 **Digit Jumper** Digit Jumper is a grid-based game, where the goal is to get from the top-left corner of  
 564 the board to the bottom-right one. At each point, the player can move  $n$  steps to the left, right, up or  
 565 down, where  $n$  is determined by the number written on the board. Digit Jumper is an example of an  
 566 environment with a constant context as is Sokoban.

## 567 B Best-First Search

568 Best-First Search greedily prioritizes node expansions with the highest heuristic estimates, aiming  
 569 for paths that likely lead to the goal. While not ensuring optimality, BestFS provides a simple yet  
 570 efficient strategy for navigating complex search spaces. The high-level pseudocode for BestFS is  
 571 outlined in Algorithm 1.

---

### Algorithm 1 Pseudocode for Best-First Search

---

```

572   while has nodes to expand do
       Take node  $N$  with the highest value
       Select children  $n_i$  of  $N$ 
       Compute values  $v_i$  for the children
       Add  $(n_i, v_i)$  to the search tree
   end while

```

---

## 573 C Training Details

Grid Search: Hidden Size vs Depth vs Repr Dim (600k training steps)

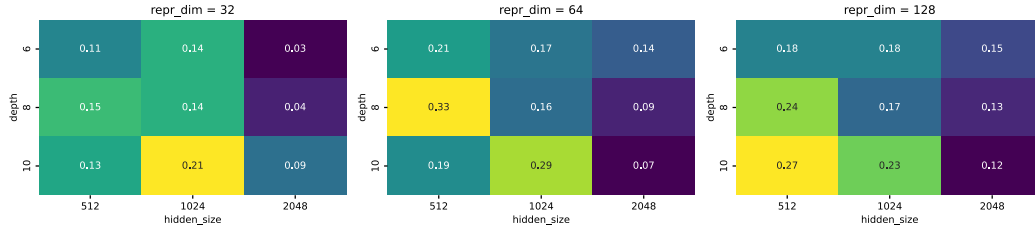


Figure 15: Grid of network’s depth, representation dimension and hidden dimension.

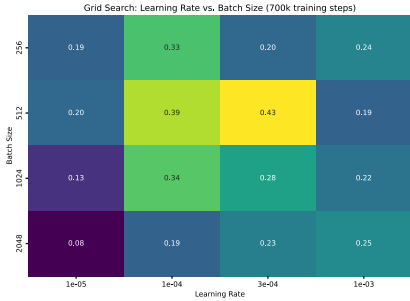


Figure 16: Learning rate and batch size grid for the Rubik’s Cube. Solved rate is investigated on a cube that has been shuffled only 10 times.

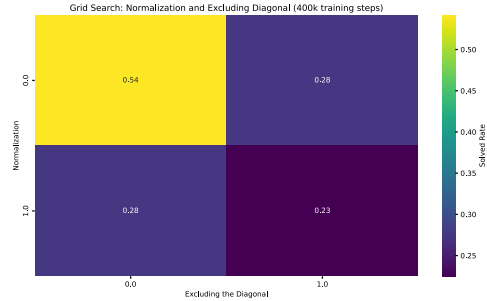


Figure 17: Use of normalization in contrastive learning and whether the distance from positives is divided by sum of all batch elements or only the in-batch negatives.

574 Code to reproduce all results is available in the anonymous repository referenced in the main text.  
 575 Below, we document the training procedures for the supervised baseline, contrastive baseline, and  
 576 CR<sup>2</sup>.

577 **Training Data** For Sokoban, we use trajectories provided by Czechowski et al. [10], and train on  
 578 a dataset of  $10^5$  trajectories. For 15-Puzzle, Rubik’s Cube, and Lights Out, we generate training  
 579 trajectories by applying a policy that performs  $n$  random actions, where  $n$  is set to 150, 21, and 49,  
 580 respectively. In the case of 15-Puzzle, we additionally remove single-step cycles from the dataset  
 581 to improve data efficiency. For Digit Jumper, we generate training data by sampling a random path



Figure 18: Trainings with different metrics for the Rubik’s Cube. The solved rate is for a cube shuffled 10 times.

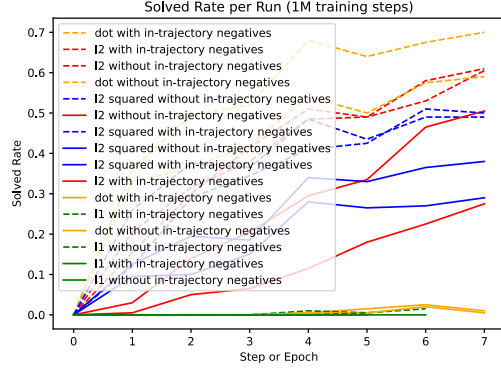


Figure 19: Symmetric contrastive loss vs. backward contrastive loss.

from the upper-left corner to the bottom-right corner on a standard  $20 \times 20$  grid. All grid cells not required for this path are filled by sampling uniformly from the set  $1, \dots, 6$ . The network for Digit Jumper typically saturates in performance after a few hours of training, so we train until convergence is observed. For Sokoban, Rubik’s Cube, Lights Out, and 15-Puzzle, we adopt an unlimited data setup and train all models for two days. This results in the models seeing approximately  $8 \times 10^6$  trajectories for Rubik’s Cube,  $7 \times 10^6$  for 15-Puzzle, and  $9 \times 10^6$  for Lights Out.

**Training Hyperparameters** We use the Adam optimizer with a constant learning rate throughout training. A learning rate of 0.0003 was found to perform well across all environments, with the exception of Lights Out, where this setting led to unstable training. For this environment, we instead use a reduced learning rate of 0.0001. In all environments, we use a batch size of 512. The choice of learning rate and batch size was guided by the performance of the contrastive baseline on Rubik’s Cube. Specifically, we evaluated solve rates on cubes shuffled 10 times, as shown in Figure 16.

**Network Architecture** We adopt the network architecture proposed by Nauman et al. [37], using 8 layers with a hidden size of 512 and a representation dimension of 64. This configuration was found to yield optimal performance for the contrastive baseline on Rubik’s Cube, as illustrated in Figure 15. We observed that this architecture also performs well across all other environments, with two exceptions:

- In Sokoban, a convolutional architecture was required to achieve strong performance.
- In Lights Out, the convolutional network was necessary to ensure training stability.

**Test Set** For Sokoban, we construct a separate test set comprising 100 trajectories, which is used to compute evaluation metrics such as accuracy, correlation, and t-SNE visualizations. For all other environments, a separate test set is not required, as we operate in an infinite data regime and train for only a single epoch. In this setting, evaluation is performed directly on unseen data sampled during training.

**Contrastive Loss** We use the backward version of the contrastive loss, which we found to consistently outperform the symmetrized variant on Rubik’s Cube, as shown in Figure 19.

For Rubik’s Cube, we use the dot product as the similarity metric. Performance across different metrics is presented in Figure 18. While the contrastive baseline performs comparably under the  $\ell_2$  metric,  $\text{CR}^2$  achieves significantly better results with the dot product. Based on similar empirical evaluations, we use the following metrics in other environments:

- Lights Out:  $\ell_2$  distance,
- Digit Jumper and 15-Puzzle: dot product,

Table 2: Average solution length of the baselines and CR<sup>2</sup> on Rubik’s Cube and 15-puzzle without using search. Supervised baseline in the Rubik’s Cube solved no of the boards.

Problem	CR <sup>2</sup>	Contrastive Baseline	Supervised Baseline
Rubik’s Cube	448.7	1830.3	NaN
15-puzzle	82.4	119.5	1054.3

- Sokoban: squared  $\ell_2$  distance.

We set the temperature parameter in the contrastive loss to the square root of the representation dimension.

**Supervised Baseline** The supervised baseline takes as input a pair of states and predicts the distance between them by performing classification into discrete bins, where the number of bins corresponds to the maximum trajectory length observed in the dataset.

In all environments—except Lights Out—the supervised baseline uses the same architecture as the contrastive baseline. For Lights Out, however, we employ a different architecture: a dense network, which achieves a performance of 0.7 solved rate, compared to a maximum of approximately 0.2 with a convolutional network. This significant difference in performance motivates our use of a dense architecture for the supervised baseline in this environment.

## D Evaluation Details

We evaluate all networks on 1000 problem instances per environment. For Rubik’s Cube, each instance is a cube shuffled 20 times. For 15-Puzzle, Lights Out, and Digit Jumper, evaluation boards are sampled randomly. For Sokoban, we follow the same instance generation procedure as described by Czechowski et al. [10].

## E Additional Experiments

**No-search results** In the main part of the paper, we limit the maximum solution depth for the no-search results. In this section, however, we remove these constraints and allow arbitrarily long solutions. In such a setup, for both the Rubik’s Cube and 15-puzzle, the contrastive methods achieve a solved rate of over 90%.

The no-search approach operates by selecting, at each step, the next state that appears most likely to move toward the solved state—based on the learned representations. If the representation were perfect, this would yield optimal solutions. In practice, however, suboptimal representations cause the agent to spiral quite randomly through the representations far away from the goal state before converging. Thus, the quality of the representation is reflected in the length of these trajectories: the better the representation captures directionality in latent space, the shorter the resulting solutions.

Table 2 reports the average solution length for the no-search approach on Rubik’s Cube and 15-Puzzle. These results suggest that the representations learned by CR<sup>2</sup> are better suited for this approach than those learned by the contrastive baseline, and significantly outperform those derived from the supervised method. This supports the conclusion that CR<sup>2</sup> provides a more reliable notion of direction in latent space. Remarkably, both the average for CR<sup>2</sup> and CRL are smaller than the solutions lengths from the training data, indicating that we observe trajectory stitching.

## F Digit-Jumper Analysis

In Digit Jumper, we observe a similar effect to that seen in Sokoban when comparing CR<sup>2</sup> to standard CRL. As shown in Figure 20, CRL rapidly achieves 100% training accuracy. However, despite this perfect accuracy, the resulting representations exhibit poor correlation with actual temporal structure (Figure 21). This is consistent with the t-SNE representations: the same as for Sokoban,

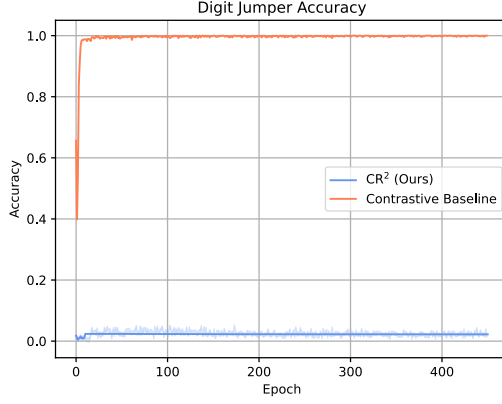


Figure 20: Accuracy of training objectives.

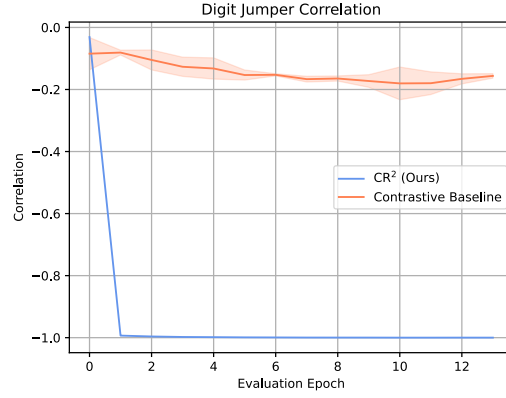


Figure 21: Correlation (Spearman’s  $\rho$ ) between the distance induced by learned embeddings and actual distance.

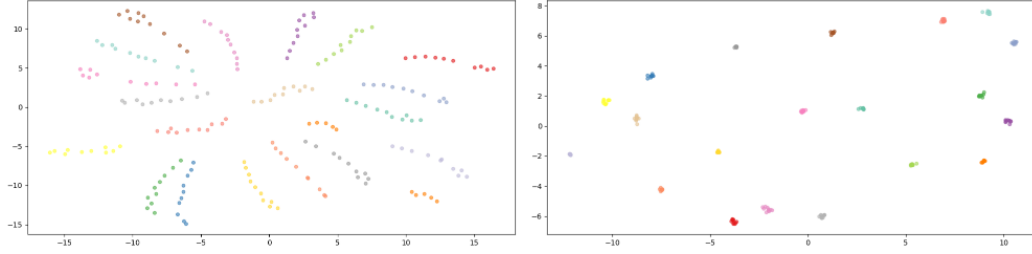


Figure 22: **CR<sup>2</sup> makes representations reflect the structure of the combinatorial task.** t-SNE visualization of representations learned by CR<sup>2</sup> (left) and CRL (right) for Digit Jumper. Colors correspond to trajectories. CRL representations (right) cluster within trajectories, making them useless for planning.

652 CRL collapses each trajectory into a single point in the representation space, discarding temporal  
 653 information. In contrast, CR<sup>2</sup> preserves a clear temporal structure within the latent space (see  
 654 Figure 22).

## 655 G Negatives

656 We explored alternative methods for incorporating in-trajectory negatives into the contrastive loss. The  
 657 first approach mimics the standard addition of hard negatives: given a batch  $\mathcal{B} = (x_i, x_{i+})_{i \in \{1..B\}}$ ,  
 658 we sample additional negatives,  $(x_{i-})_{i \in \{1..B\}}$ , and compute the loss as

$$\mathcal{L} = \frac{1}{B} \sum_i \log \left( \frac{\exp(f(x_i, x_{i+}))}{\sum_{j \neq i} \exp(f(x_i, x_{j+})) + \exp(f(x_i, x_{i-}))} \right),$$

659 .

660 We considered three strategies for selecting in-trajectory negatives: sampling uniformly at random,  
 661 selecting the first state, or selecting the last state of the trajectory. For Rubik’s Cube, instead of  
 662 choosing the last state—which is identical for all trajectories—we sample a random state further  
 663 away from the solution to serve as a negative.

As shown in Figures 23 and 24, training with this approach failed to achieve strong performance. We hypothesized that the large error introduced by the in-trajectory negatives ( $x_{i-}$ ) caused excessively large gradients, destabilizing training. To mitigate this, we applied a normalization scheme: so that

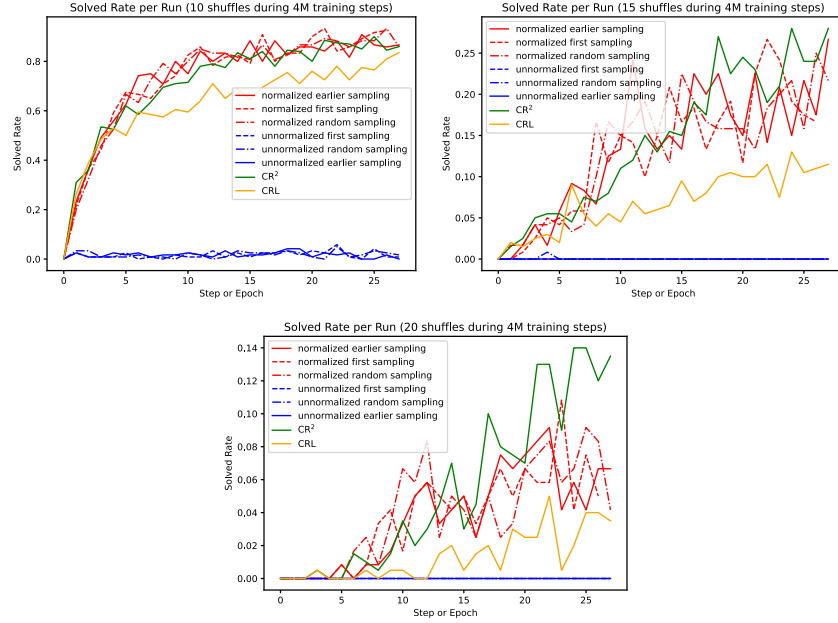


Figure 23: Comparison of different methods of introducing in-trajectory negatives in the Rubik's Cube environment, with increasing number of shuffles of the cube. While normalized negatives perform similarly to CR<sup>2</sup> for a small number of shuffles, their performance fails to be as good for more shuffles.

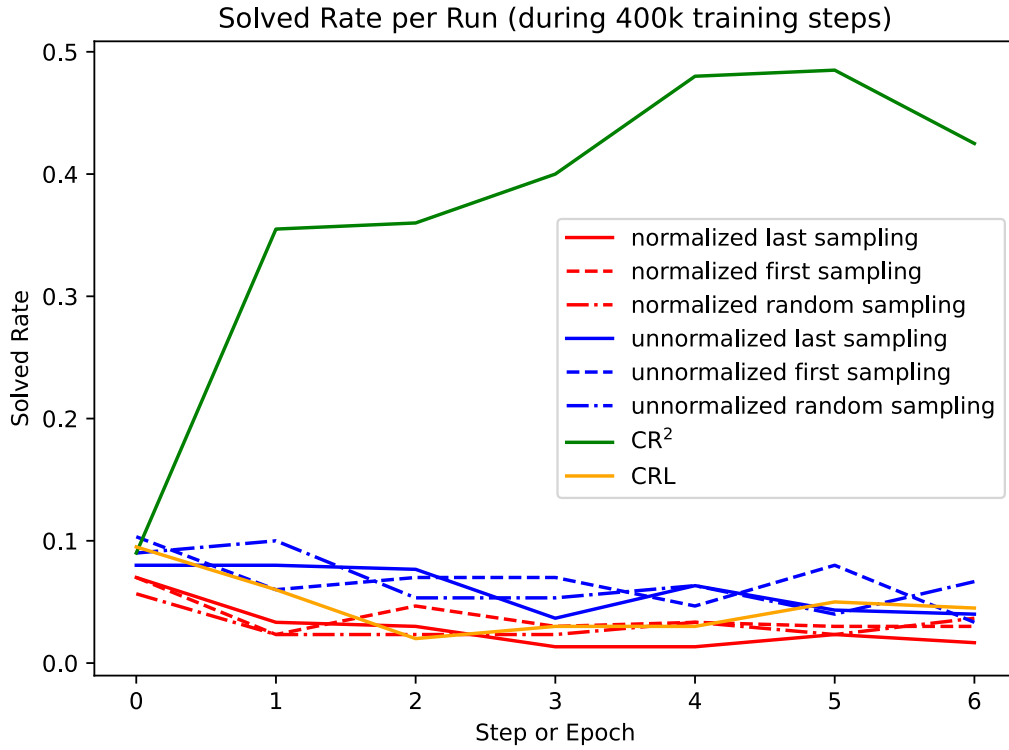


Figure 24: Comparison of different methods of introducing in-trajectory negatives in the Sokoban environment. The only negative sampling strategy that works is CR<sup>2</sup>.

the vector

$$\begin{bmatrix} f(x_1, x_{1-}) \\ \vdots \\ f(x_B, x_{B-}) \end{bmatrix}$$

has the same norm as the norm (when viewing the matrix as a vector of size  $B^2$ ) of the matrix

$$\begin{bmatrix} f(x_1, x_{1+}) & f(x_1, x_{2+}) & \dots & f(x_1, x_{B+}) \\ \vdots & \vdots & \ddots & \vdots \\ f(x_B, x_{1+}) & f(x_B, x_{2+}) & \dots & f(x_B, x_{B+}) \end{bmatrix}$$

664 . This normalization enabled achieving comparable performance to  $\text{CR}^2$  on Rubik’s Cube shuffled 10  
 665 times (Figure 23). However,  $\text{CR}^2$  still outperforms all negative sampling strategies on cubes shuffled  
 666 15 and 20 times.

667 For Sokoban, the only approach that consistently improved performance is  $\text{CR}^2$ , as demonstrated in  
 668 Figure 24. We hypothesize that this is because removing contextual information is more challenging  
 669 in Sokoban than in Rubik’s Cube. In the latter, the context is more local and changes gradually over  
 670 time, making it *softer*, while the context in Sokoban is constant through a trajectory. This is discussed  
 671 in detail in Section 4.2.

## 672 H Computational Resources

673 All training experiments were conducted using NVIDIA A100 GPUs and took between 5 and 48  
 674 hours each. The solving runs ranged from 10 minutes to 10 hours. In total, the project required  
 675 approximately 30,000 GPU hours to complete.