LEARNING TO RECOVER SPARSE SIGNALS

Sichen Zhong[⋆]*, Yue Zhao*[†⋆]*, Jianshu Chen*[‡]

[⋆] Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY
[†] Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY
[‡] Tencent AI Lab, Bellevue, WA

ABSTRACT

In compressed sensing, a primary problem to solve is to reconstruct a high dimensional sparse signal from a small number of observations. In this work, we develop a new sparse signal recovery algorithm using reinforcement learning (RL) and Monte Carlo Tree Search (MCTS). Similarly to OMP, our RL+MCTS algorithm chooses the support of the signal sequentially. The key novelty is that the proposed algorithm *learns* how to choose the next support as opposed to following a pre-designed rule as in OMP. Empirical results are provided to demonstrate the superior performance of the proposed RL+MCTS algorithm over existing sparse signal recovery algorithms.

Index Terms — Compressed Sensing, Reinforcement Learning, Monte Carlo Tree Search, Basis Pursuit, Orthogonal Matching Pursuit

1. INTRODUCTION

We consider the compressed sensing (CS) problem [1–4], where for a given matrix $A \in \mathbb{R}^{m\times n}$, $m \ll n$, and a (noiseless) observation vector $y = Ax_0$, we want to recover a k-sparse vector/signal x_0 ($k < m$). Formally, it can be formulated as:

$$\underset{x}{\text{minimize}} \ ||x||_0, \tag{1}$$

$$\text{subject to} \ \ Ax = Ax_0 \tag{2}$$

The readers are referred to the seminal papers [1, 2] for a comprehensive survey of the compressed sensing problem.

There is a large collection of algorithms for solving the CS problem. Some foundational and classic algorithms include linear relaxation and convex optimization, matching and subspace pursuit [5–7], as well as iterative thresholding [8, 9] among others. In particular, two well-established methods are (i) Orthogonal Matching Pursuit (OMP) and (ii) Basis Pursuit (BP). OMP recovers x_0 by choosing the columns of A iteratively until we choose k columns [10]. BP recovers x_0 by solving $\min_{Ax=y} ||x||_1$ [2]. Because OMP and BP are extremely well studied theoretically [1, 2] and empirically [11], we use these two algorithms as the main benchmark methods to compare against when evaluating the proposed RL+MCTS algorithm.

Recent advancements in machine learning have opened a new frontier for signal recovery algorithms. Specifically, these algorithms take a deep learning approach to CS and the related error correction problem. The works in [12], [13], [14],and [15] apply ANNs and RNNs for encoding and/or decoding of signals x_0. Modern generative models such as Autoencoder (AE), Variational Autoencoder (VAE), and Generative Adversarial Networks (GANs) have also been used to tackle the CS problem with promising theoretical and empirical results [16–18]. These works involve using generative models for encoding structured signals, as well as for designing the measurement matrix A. Notably, the empirical results in these works all use structured signals in x_0. For example, in [17] and [18], MNIST digits and celebrity images are used for training and testing, while in [16], block sparsity for x_0 is assumed.

Differently from the above learning-based works, our innovation with machine learning is on signal *recovery* algorithms (as opposed to signal encoding or measurement matrix design). We do not assume the signals to be structured (such as images), but cope with general sparse signals. This underlying model for x_0 is motivated by the same assumptions in the seminal work on universal phase transitions by Donoho and Tanner in [11]. Moreover, we assume the measurement matrix A is given. Extending to varying matrices A is left for future investigation.

In this work, we approach the signal recovery problem using reinforcement learning (RL). Specifically, we leverage the Monte Carlo Tree Search (MCTS) technique with RL, which was shown to achieve outstanding performance in the game of Go [19,20]. We further introduce special techniques to reduce the computational complexity for dealing with higher signal sparsities in CS. Experimental results show that the proposed RL+MCTS algorithm significantly outperforms OMP and BP for matrix A of various sizes.

2. COMPRESSED SENSING AS A REINFORCEMENT LEARNING PROBLEM

In this section, we formulate the sparse signal recovery problem as a special sequential decision making problem, which we will solve by using RL and MCTS. In reinforcement learning, an agent interacts with an environment by taking actions a based on the observed state s, and the objective is to maximize the long-term cumulative reward by picking the correct actions. The environment is modeled by a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, P, R)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ denotes the action space, $P(s'|s, a)$ denotes the state transition probability (i.e., the probability of transiting to state $s' \in \mathcal{S}$ when taking action a at state s), and R denotes the immediate reward received after the transition.

In the context of compressed sensing, a key challenge is to correctly choose the columns of A, or equivalently, the support of x_0, such that the problem (1) is solved, which is discrete and combinatorial in nature. To address this problem, we formulate it as a sequential decision making problem: an agent sequentially chooses one column of A at a time until it selects up to k columns such that the constraint in (2) holds and the ℓ_0-loss in (1) is minimized. The MDP for compressed sensing can then be defined as follows. A state $s \in \mathcal{S}$ is a pair (y, S), where y is the observed signal generated ac-

cording to x_0, and $S \subseteq [n]$ is the set of the already selected columns of A, where $[n] \triangleq \{1, \ldots, n\}$. In our current setup, we assume the matrix A is fixed, so a state is not dependent on the sensing matrix. Terminal states are states $s = (y, S)$ which satisfy one or more of the following conditions: (i) $|S| = k$ (the maximum possible signal sparsity), or (ii) $||A_S x_s - y||_2^2 < \epsilon$ for some given ϵ. Here, A_S stands for the submatrix of A that is constructed by the columns of A indexed by the set S, and x_s is the optimal solution given that the signal support is S,

$$x_s \triangleq \arg\min_z ||A_S z - y||_2^2. \tag{3}$$

For the action space, we define the set of all feasible actions at state $s = (y, S)$ to be $\mathcal{A}_s = [n] \setminus S$. In other words, a valid action from state s is to choose exactly one additional column from the remaining ones. Note that in compressed sensing, when an action a is taken (i.e., a new column of A is selected) for a particular state $s = (y, S)$, the next state s' is determined; that is, the MDP transition is deterministic. Finally, we define our reward function R. In many applications, one may favor minimizing the residual value instead of minimizing the sparsity. This is due to the fact that observed signals y are usually corrupted by noise, so it is impossible to find an exact solution such that $Ax = y$. Motivated by this, (and granted that our CS problem model is noise-free), we design the reward function as:

$$R(s) := -\alpha||x_s||_0 - \gamma||A_S x_s - y||_2^2 \tag{4}$$

where s is again a state in the form (y, S), and $\alpha, \gamma > 0$ are fixed hyperparameters, and x_s is determined by (3).

Different from existing compressed sensing algorithms, we propose to learn, via RL and MCTS, a policy to sequentially select the columns of A and reconstruct the sparse signal x_0, based on data generated for training. We obtain the training data by generating k-sparse signals x_0 and computing the corresponding vectors $y = Ax_0$ (each k is randomly generated from 1 to m). For each signal y, we then use a "policy network" (to be explained in details later) along with MCTS to choose columns sequentially until k columns have been chosen. The traversed states will be used as additional training data for updating the policy network. Such a strategy allows us to move as much of the computational complexity as possible in testing (i.e., performing the sparse signal recovery task) into training, which shares a similar spirit to the work in [21].

A general challenge for reinforcement learning is that it has high sample complexity; that is, it requires a large amount of training data to learn an effective policy. Note that in compressed sensing, the MDP transition probability is deterministic and known. This allows us to use model-based information to perform MCTS-like planning, which could greatly reduce the sample complexity in reinforcement learning. Such a strategy of exploiting model-based information has been successfully used in AlphaGo [20] and AlphaGo Zero [19] in defeating top human players of Go. In this paper, we will leverage model-based information in CS to effectively learn a sparse signal recovery strategy based on data generated from the model.

3. THE RL+MCTS ALGORITHM

3.1. The Policy/Value Network f_θ

A policy $\pi(a|s)$ in our MDP is defined to be the probability of taking action a at state s. The objective is to learn a policy π that maximizes the long-term cumulative reward. To learn a policy in the above sequential decision making formulation of CS, we employ a single neural network f_θ to jointly model the policy $\pi_\theta(a|s)$ and the state-value function $V_\theta(s)$, where θ is the model parameter (e.g., the weights in a neural network). The policy $\pi_\theta(a|s)$ defines a probability over all actions for a given state s, where the action set includes the possible next columns of A to pick and a stopping action. The value $V_\theta(s)$ defines the long-term reward that an agent receives when we start from the state s and follow the given policy. Our goal is to learn the network f_θ such that it takes a sequence of actions (i.e., pick the columns) that lead to the highest reward.

We design two sets of input features for the policy/value network. The first set of input features is exactly (3), where x_s is then extended to a vector in $\mathbb{R}^n$ with zeros in components whose indices are not in s. In other words, x_s is the "best" solution for signal recovery assuming that the support of the signal is given by s. We abuse the notation x_s to denote both (3) and its extended version with zero padding, and the meaning of it will be clear given the context. The second set of features is motivated by OMP, which is given by

$$\lambda_s := A^T(y - A_S x_s) \in \mathbb{R}^n,$$

where $y - A_S x_s$ is the residual vector associated with the solution x_s. For the root state r in which no columns are chosen, x_r is set to be the n-dimensional zero vector, and $\lambda_r := A^T y$. Note that the OMP rule is exactly choosing the next column index whose corresponding component in $|\lambda_s|$ is the largest, where $|\cdot|$ is the absolute value taken component wise.

The neural network architecture is chosen to be a fully connected network with one hidden layer. We choose the activations of neurons in the hidden layer to be ReLU units. The output layer will be softmax (for $\pi_\theta(\cdot|s)$), and identity (for $V_\theta(s) \in \mathbb{R}$). Since there are two outputs, we use categorical cross entropy and MSE as the loss functions [22].

3.2. RL+MCTS Training Procedure

The goal of the RL+MCTS algorithm is to iteratively train the policy network f_θ. The overall training structure is given in Algorithm 1.

Algorithm 1 High-Level Training Procedure

1: **initialize:** $j = 0$, $\theta = \theta_0$, θ_0 random, fixed matrix $A \in \mathbb{R}^{m \times n}$
2: **while** $j < i$ (where i is a hyperparameter) **do**
3: **1)** generate training samples from each (y, x_0) pair
4: by building a tree using Monte Carlo Tree Search (MCTS)
5: and the current f_θ
6: **2)** train/update neural network parameters to get $\hat{\theta}$ using
7: the training samples from step 1.
8: $\theta \leftarrow \hat{\theta}$
9: $j \leftarrow j + 1$
10: **end while**

Most of the details arise in step 1) of Algorithm 1. Similar to the AlphaGo Zero algorithm [19], the proposed RL+MCTS algorithm uses Monte Carlo Tree Search (MCTS) as a policy improvement operator to iteratively improve the policy network in the training phase. For a randomly generated pair (y, x_0), we use MCTS and the current f_θ to generate new training samples to feed back into the neural network. We note that in the testing phase, MCTS can also be combined with the policy/value network to further boost the performance. Specifically, for each given observation vector y and the desired sparsity k, we run a number of MCTS simulations to construct a search tree [23–25].

In our evaluation (cf. Section 4), we generate fixed random matrices A whose entries are sampled from independent and identically

distributed (i.i.d) standard Gaussian $\mathcal{N}(0,1)$ distributions. We randomly generate x_0 with a sparsity of k and then compute y using $y = Ax_0$. k is randomly generated integer in $\{1, 2, ..., m-1\}$ and the locations of the k nonzero elements in x_0 are also randomly chosen. This constitutes a single (y, x_0) pair.

For the generated (y, x_0) pair, we construct a tree according to the steps given in Algorithm 2. Each node in this tree corresponds to a state s, and each edge (s, a) corresponds to picking action a at state s. Each edge (s, a) in the search tree is associated with two quantities: the action value function $Q(s, a)$ and the visiting count $N(s, a)$. $Q(s, a)$ defines the average reward for selecting the edge (s, a) in the search tree, and $N(s, a)$ denotes the number of times the edge (s, a) has been visited.

Once Algorithm 2 terminates via steps 5) and 6), then all states c traversed are returned. We compute the reward and probability label for each state. The reward label is the same for each traversed state, which is the terminal reward (eq. 4) of the final state c was in. For a given state s which was returned above, the probability label is given by the vector $\frac{1}{\sum_b N(s,b)} N(s, \cdot)$, i.e., the empirical frequencies experienced in the simulated MCTS search.

Algorithm 2 Constructing a complete MCTS tree from a **single** (x_0, y) pair

1: **Initialize**: compute the features of the root state r: x_r and λ_r. The root state corresponds to the empty set with no column chosen. Set current state, $c \leftarrow r$.
2: Run M number of MCTS simulations on c according to PUCT (7) to traverse the tree.
3: Select the action/column a which was visited the most ($N(c, a)$ largest). This leads us to a new state, denoted by c'.
4: Reassign $c \leftarrow c'$.
5: **if** c is a terminal state **then**
6: **return**: all states c traversed
7: **else** c not a terminal state
8: Return to step 2
9: **end if**

3.3. Monte Carlo Tree Search (MCTS)

Step 2 in Algorithm 2 requires clarification. A single MCTS search grows the tree by traversing from the root of the tree to some leaf, expanding the leaf, and updating the edge weights of the traversed path. We describe a single MCTS simulation in Algorithm 3.
Starting from the current root state s_0, we select action a_t according to a variant of the PUCT (Polynomial Upper Confidence bound for Trees) policy [19, 26] until we reach the leaf-state of the tree:

$$a_t = \arg\max_a \left\{ Q(s_t, a) + c_{puct} \cdot \pi_\theta(a|s_t) \frac{\sqrt{\sum_b N(s_t, b)}}{N(s_t, a) + 1} \right\} \quad (7)$$

where s_t denotes the state at time step t and c_{puct} is a hyperparameter which controls the tradeoff between exploration and exploitation. Overall, PUCT treats π_θ as a prior probability to bias the MCTS search; PUCT initially prefers actions with high values of π_θ and low visit count $N(s, a)$ (and hence exploration), but then asymptotically prefers actions with high value $Q(s, a)$ (and hence exploitation). Once the search reaches a leaf-state, we expand the leaf-state and then update $Q(s, a)$ and $N(s, a)$ over the edges that have been traversed according to the rule given in [19]. An MCTS search tree is constructed by running a fixed number of M simulations at each depth. If we perform M Monte Carlo simulations at a fixed depth and state in the tree, then we perform a total of $M \times k$ simulations for a single y.

Algorithm 3 A single MCTS simulation/search

1: **Input**: current MCTS tree, state $c = s_0$, $t = 0$
2: **while** s_t is not a leaf node **do**
3: for each valid action a, compute: $U(s_t, a) := Q(s_t, a) + c_{puct} \cdot \pi_\theta(a|s_t) \frac{\sqrt{\sum_b N(s_t,b)}}{N(s_t,a)+1}$
4: Compute the action a which gives the largest $U(s_t, a)$. Equivalently, let $a_t = \arg\max_a U(s_t, a)$
5: Traverse to the next node from state s_t by taking action a_t. Call this state s_{t+1}.
6: $t \leftarrow t + 1$
7: **end while**
8: Let the traversed states and actions from the above while loop be $\{s_0, a_0, s_1, a_1 s_2, ..., a_{v-1}, s_v\}$, where s_v is a leaf node and $v \in \mathbb{N}$.
9: Construct new neighbor states/nodes of s_v and corresponding edges (s_v, a). For each edge (s_v, a), initialize edge weights to be $Q(s_v, a) = 0, N(s_v, a) = 0$.
10: Update all edge weights traversed. In other words, for every $0 \le t \le v - 1$, update in the following order:

$$N(s_t, a_t) \leftarrow N(s_t, a_t) + 1 \quad (5)$$

$$Q(s_t, a_t) \leftarrow \frac{(N(s_t, a_t) - 1)Q(s_t, a_t) + v(s_v)}{N(s_t, a_t)} \quad (6)$$

11: **Output:** Updated MCTS tree with the leaf node s_v expanded and traversed edge weights updated.

3.4. Reducing Computational Complexity during Training by Limiting the Tree Depth

When training the proposed RL+MCTS algorithm, we employ the following technique for reducing the training complexity. First, we remark that using MCTS as a policy improvement operator can potentially be computationally expensive for relatively large matrix A (depending on the available computation resources). To address this challenge, we fix the maximum depth d of the MCTS tree; that is, we build the MCTS tree as described in Section 3.3 until we reach a depth of d. From then on, we roll-out the remaining levels of the tree by simply using the OMP rule to select all remaining columns until a total of k columns are chosen. This technique will be evaluated in the experiments in the next section.

4. EXPERIMENTAL RESULTS

In this section, we present experimental results for evaluating our proposed RL+MCTS algorithm and comparing it against two benchmark methods: (i) OMP and (ii) BP (i.e., ℓ_1 minimization).

4.1. MCTS without Tree Depth Constraint

We begin by presenting results on the proposed RL+MCTS algorithm without limiting the tree depth. In this setting, we will be training and testing on matrices of size 7×15 and 15×50. The training parameters we use in our experiment is given in Table 2. At testing time, we generate observed signals y via the following method. For each sparsity level k between 1 and m, we generate 1000 k-sparse

signals x_0. The k locations of the support of x_0 are chosen randomly, and each entry in x_0 is generated i.i.d $U[0,1]$. We compare the proposed RL+MCTS policy/value network to OMP and BP. With $\hat{x}$ as the predicted sparse vector (by RL+MCTS, OMP, or BP, respectively), we define successful recovery of x_0 as $||\hat{x} - x_0||_2^2 < 10^{-3}$, (i.e., "symbol" recovery, instead of just "bit" recovery).

Figure 1(a) and Figure 1(b) show the recovery success probability of different algorithms. We would like to emphasize that the proposed RL+MCTS results shown in Figures 1(a)–1(b) are obtained using the learned policy $\pi_\theta(a|s)$ only, and *no MCTS has been used in the testing stage* (which, if used, would lead to further improvement). We see that even in this setting RL+MCTS still significantly outperforms OMP and BP. The results imply that starting from a neural network with randomly initialized weights, it is indeed possible for the network to learn an effective policy to choose the columns of A.

We further note a key difference between the proposed RL+MCTS algorithm and AlphaGo Zero: while the latter has finitely many states, the state in the proposed RL+MCTS algorithm depends not only on the current set of columns chosen, but also on the observations y. Hence, the state space in our application is continuous in nature.

4.2. Average Prediction Times

In Table 1, we give the average prediction times per signal in seconds. For OMP and BP, we use python libraries sklearn and cvx respectively. To illustrate the speed during testing, we measure the prediction times on a much less powerful machine than what was used during training. While training was accomplished on a i7 4790 (3.6 GHz) with a single GTX 780, the testing speeds in Table 2 were conducted on a Macbook Air with an Intel i5 clocked at 1.4 GHz and an integrated Intel HD 5000. We predict that the testing speeds can be greatly improved with a more powerful machine and further optimization in the source code. In general, we see that using just the policy/value network for prediction is in general slower than OMP, but on par with or better than BP.

4.3. MCTS with Limited Tree Depth

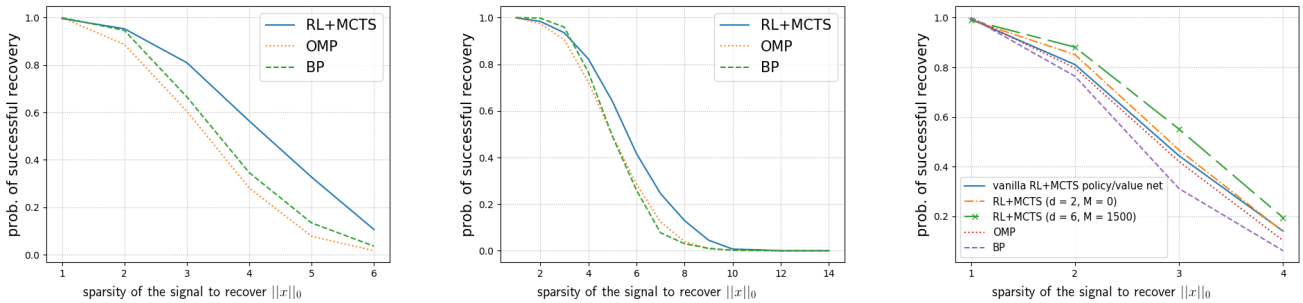
For higher sparsities k, the training time for the proposed RL+MCTS algorithm would be significantly increased. Observe that each (y, x_0) corresponds to a single MCTS tree, and this MCTS tree has depth exactly equal to $k+1$. Since the total number of MCTS simulations we conduct for a single (y, x_0) is kM, we see that an increase in k directly lengthens the training time. We will now show results using the RL+MCTS algorithm with reduced complexity as described in Section 3.4. Specifically, in a single MCTS search, we expand the tree to depth d, and then proceed to follow the OMP rule until a terminal state is reached. We now show the experiment results for this version of the RL+MCTS algorithm. Specifically, we consider the 10×100 matrix in our evaluation.

- **Training Details -** The training details of this experiment can be found in Table 2. We train two models. **A)** First, we train a policy/value network using the vanilla RL+MCTS algorithm without tree depth constraint. **B)** We train a policy/value network by limiting the tree depth $d = 6$, which leads to a 40% reduction in training time per sample.
- **Testing Details (cf. Fig. 1(c)) - A)** We test the policy/value network trained from A) above. This policy/value network will select each column without MCTS. **B)** We test the policy/value network trained from B) above. First, we test the policy/value network to pick the first column; For all subsequent columns up to k, we invoke the OMP rule. This is equivalent to setting the tree depth during testing to $d = 2$ and $M = 0$. Using the same policy/value network, we also conduct an experiment where $d = 6$ and the number of MCTS simulations is set to 1500 during testing.

From Figure 1(c), note that the vanilla RL+MCTS policy $\pi_\theta(a|s)$ still performs slightly better than both OMP and BP. We see that training the RL+MCTS algorithm with a fixed tree depth gives us favorable results versus OMP, vanilla RL+MCTS policy $\pi_\theta(a|s)$, and BP. The green curve shows us that success probability can be improved by increasing the number of MCTS simulations during test time and the depth of the tree. Note that the $d = 2$ and $M = 0$ model is equivalent to the policy network choosing only the first column, and then choosing the rest of the columns via OMP rule. This model performs strictly better than solely using the policy network (blue curve), but performs worse than the $d = 6$, $M = 1500$ model. What these results show is that the policy network learns to take high-level actions (in picking only the first column), and that successful recovery probability can be improved by increasing the number of MCTS M during testing. Finally, the time speed up in training vanilla RL+MCTS (Training model A with max tree depth) decreases from 4.5 seconds per signal to 2.7 seconds (Training model B with tree depth d = 6), a 40% reduction.

5. CONCLUSION

We have shown that the proposed RL+MCTS algorithm is a highly effective sparse signal decoder for the compressed sensing problem assuming no signal structure other than sparsity. Even without using MCTS in testing, the RL+MCTS algorithm's performance exceeds that of existing sparse signal recovery algorithms such as OMP and BP. The flexibility in the RL+MCTS algorithm's design further offers many interesting avenues for future research. For one, it is possible the features chosen in our model can be further improved. Secondly, since the true signal x_0 is known in training, one may be able to leverage the information about x_0 to increase training sample efficiency. The training hyper-parameters may also be further tuned to improve performance. Broader settings of problems such as noisy observations and varying observation matrices A are under active investigation.



(a) (b) (c)

Fig. 1. Signal Recovery accuracies of the 7 by 15 matrix, 15 by 50 matrix, and 10 by 100 matrices (from left to right)

Table 1. Average Prediction Times

7 by 15	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k=11	k=12	k=13	k=14
RL+MCTS	1.2e-3	2.0e-3	3.2e-3	3.7e-3	4.6e-3	5.5e-3								
OMP	2.6e-4	4.2e-4	5.9e-4	6.1e-4	7.2e-4	8.2e-4								
BP	2.4e-3	2.8e-3	3.2e-3	2.8e-3	2.9e-3	2.8e-3								
15 by 50	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k=11	k=12	k=13	k=14
RL+MCTS	1.7e-3	2.3e-3	3.3e-3	3.9e-3	5.2e-3	5.9e-3	7.1e-3	8.7e-3	8.9e-3	1.0e-2	1.1e-2	1.2e-2	1.3e-2	1.4e-2
OMP	3.5e-4	4.7e-4	5.8e-4	6.5e-4	8.0e-4	8.6e-4	9.9e-4	1.1e-3	1.1e-3	1.2e-3	1.4e-3	1.5e-3	1.6e-3	1.7e-3
BP	7.5e-3	6.9e-3	7.2e-3	7.1e-3	7.9e-3	7.6e-3	8.0e-3	8.4e-3	7.6e-3	7.8e-3	8.1e-3	8.1e-3	7.8e-3	8.0e-3
10 by 100	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k=11	k=12	k=13	k=14
vanilla RL+MCTS	1.4e-3	2.3e-3	3.3e-3	3.9e-3	4.9e-3	5.9e-3	6.7e-3	7.8e-3	1.0e-2					
RL+MCTS (d=2, M=0)	2.0e-3	1.8e-3	2.1e-3	2.4e-3	2.6e-3	3.2e-3	3.5e-3	3.6e-3	3.9e-3					
RL+MCTS (d=6, M = 1500)	0.27	0.88	1.76	3.08	4.97	5.73	5.58	5.81	5.94					
OMP	2.8e-4	4.7e-4	5.8e-4	6.4e-4	7.5e-4	8.3e-4	9.3e-4	1.0e-3	1.2e-3					
BP	1.6e-2	2.3e-2	2.5e-2	2.3e-2	2.25e-2	2.24e-2	2.20e-2	2.1e-2	2.7e-2					

Table 2. Training Hyper-Parameters for all matrix sizes

NN hyper-parameters	(7×15)	(15×50)	(10×100)	description
Input	30×1	100×1	200×1	Input(features x_s and λ_s)
Hidden Layer	200 neurons	200 neurons	200 neurons	activation ReLu
Output	17×1	52×1	102×1	Output dimensions($\hat{p}_\theta(\cdot\|s)$ and $\hat{v}_\theta(s)$)
θ	9400 weights	30400 weights	60400 weights	
general hyper-parameters				description
A	$\in \mathbb{R}^{7\times15}$	$\in \mathbb{R}^{15\times50}$	$\in \mathbb{R}^{10\times100}$	entries are i.i.d $\mathcal{N}(0,1)$
k	$\in \{1, 2..., 6\}$	$\in \{1, 2..., 14\}$	$\in \{1, 2..., 9\}$	randomly generated sparsity of x_0 during training
x_0	$\in \mathbb{R}^{15}$	$\in \mathbb{R}^{50}$	$\in \mathbb{R}^{100}$	randomly selected support locations, where each component of x_0, $x_{0,i} \sim U[0,1]$, $\|\|x_0\|\|_0 = k$
i	100	200	100	num. of training iterations
e	400	400	100	num. signals (y, x) pairs generated
M	500	500	1500	num. of MCTS simulations
c_{puct}	2	2	3	exploration/exploitation factor
ϵ	10^{-5}	10^{-5}	10^{-5}	determines threshold of terminal states
d	max	max	max, 6	max tree depth of MCTS tree

6. REFERENCES

[1] D. L. Donoho, "Compressed sensing," *IEEE Transactions on information theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[2] E. J. Candes, "The restricted isometry property and its implications for compressed sensing," *Comptes rendus mathematique*, vol. 346, no. 9-10, pp. 589–592, 2008.

[3] E. J. Candes and T. Tao, "Decoding by linear programming," *IEEE Transactions on Information Theory*, vol. 51, no. 12, pp. 4203–4215, Dec 2005.

[4] M. Lustig, D. Donoho, and J. M. Pauly, "Sparse MRI: The application of compressed sensing for rapid MR imaging," *Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine*, vol. 58, no. 6, pp. 1182–1195, 2007.

[5] D. Needell and R. Vershynin, "Uniform uncertainty principle and signal recovery via regularized orthogonal matching pursuit," *Foundations of computational mathematics*, vol. 9, no. 3, pp. 317–334, 2009.

[6] D. Needell and J. A. Tropp, "Cosamp: Iterative signal recovery

from incomplete and inaccurate samples," *Applied and computational harmonic analysis*, vol. 26, no. 3, pp. 301–321, 2009.

[7] W. Dai and O. Milenkovic, "Subspace pursuit for compressive sensing signal reconstruction," *IEEE transactions on Information Theory*, vol. 55, no. 5, pp. 2230–2249, 2009.

[8] I. Daubechies, M. Defrise, and C. De Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, vol. 57, no. 11, pp. 1413–1457, 2004.

[9] M. Fornasier and H. Rauhut, "Iterative thresholding algorithms," *Applied and Computational Harmonic Analysis*, vol. 25, no. 2, p. 187, 2008.

[10] J. A. Tropp and A. C. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Transactions on Information Theory*, vol. 53, no. 12, pp. 4655–4666, 2007.

[11] D. Donoho and J. Tanner, "Observed universality of phase transitions in high-dimensional geometry, with implications for modern data analysis and signal processing," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 367, no. 1906, pp. 4273–4293, 2009.

[12] A. Mousavi, A. B. Patel, and R. G. Baraniuk, "A deep learning approach to structured signal recovery," in *Proc. 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2015, pp. 1336–1343.

[13] A. Mousavi and R. G. Baraniuk, "Learning to invert: Signal recovery via deep convolutional networks," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 2272–2276.

[14] A. Adler, D. Boublil, M. Elad, and M. Zibulevsky, "A deep learning approach to block-based compressed sensing of images," *arXiv preprint arXiv:1606.01519*, 2016.

[15] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Beery, "Deep learning methods for improved decoding of linear codes," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 119–131, 2018.

[16] S. Wu, A. G. Dimakis, S. Sanghavi, F. X. Yu, D. Holtmann-Rice, D. Storcheus, A. Rostamizadeh, and S. Kumar, "Learning a compressed sensing measurement matrix via gradient unrolling," *arXiv preprint arXiv:1806.10175*, 2018.

[17] A. Bora, A. Jalal, E. Price, and A. G. Dimakis, "Compressed sensing using generative models," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 537–546.

[18] Y. Wu, M. Rosca, and T. Lillicrap, "Deep compressed sensing," *arXiv preprint arXiv:1905.06723*, 2019.

[19] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[20] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[21] Y. Zhao, J. Chen, and H. V. Poor, "A Learning-to-infer method for real-time power grid topology identification," *arXiv preprint arXiv:1710.07818*, 2017.

[22] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics New York, NY, USA:, 2001, vol. 1, no. 10.

[23] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*. Springer, 2006, pp. 282–293.

[24] H. S. Chang, M. C. Fu, J. Hu, and S. I. Marcus, "An adaptive sampling algorithm for solving markov decision processes," *Operations Research*, vol. 53, no. 1, pp. 126–139, 2005.

[25] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[26] C. D. Rosin, "Multi-armed bandits with episode context," *Annals of Mathematics and Artificial Intelligence*, vol. 61, no. 3, pp. 203–230, Mar 2011.