Structured Sparse Recovery using Graph Neural Network and Reinforcement Learning

Kha-Dinh Luong
Department of Computer Science
University of California Santa Barbara
vluong@cs.ucsb.edu

Chinmay Sonar
Department of Computer Science
University of California Santa Barbara
csonar@cs.ucsb.edu

Abstract

The goal of compressive sensing (CS) is to recover a compressible signal from a limited number of measurements. In many cases, one could levitate the existing structure within the signal to improve the efficiency of CS, leading to even fewer measurements [1]. Many existing methods integrate into the CS solver an oracle that, based on the structure of the signal, limits the possible search space of the algorithm [1][5]. However, the oracle is extremely structure-dependent in that it needs to be designed specifically for the structure at hand, often with strong assumptions. Moreover, when the underlying structure is complex, designing a corresponding oracle deems intractable. In this project, we consider solving a CS problem as a sequential decision making process, where in each step, an oracle-agent minimally alters the current estimation of the solver such that it complies with the underlying structure. The structure, generally represented as a graph, is not available to the agent beforehand and therefore needs to be learned.

1 Introduction

The Shannon-Nyquist sampling theorem laid the theoretical foundation of signal recovery from its measurements. The theorem states that a signal can be reconstructed effectively if the sampling rate is at least twice the maximal signal frequency. In many applications such as radar imaging, coping up with the sampling rate required by Shannon-Nyquist theorem is hard as it needs a large storage space, high power consumption or large sensing time.

In transform compressive systems, the signal is effectively transformed into a low dimensional space by re-representing the signal using a sparse and compressible coefficients α in basis expansion $x = \psi\alpha$ where x is the original n-dimensional signal and ψ is the $N \times N$ basis matrix. α is *sparse* if it can be well approximated by a vector with at most $k << N$ non-zero entries. We say that α is *compressible* if in the descending sorted list of coefficients of α the last $n - k$ coefficients rapidly tend to zero.

For sparse or compressible signals, the theory of Compressive Sensing goes beyond the Shannon-Nyquist theorem and states that the signal can be effectively reconstructed using much fewer incoherent (non-periodic) measurements. Mathematically, the measurements $y = \phi x = \phi\psi\alpha$, where the rows of $M \times N$ matrix ϕ are the measurement vectors. Here, the matrix $\phi\psi$ is rank deficient, and loses information in general but for sparse and compressible signals it preserves the distance between the vector representation of such signals known as the restricted isometry property (RIP). It is shown that with a high probability, RIP holds for a large class of random matrices. K-sparse and compressible signals are known to be robustly recoverable with $\mathcal{O}(K \log(N/K))$ measurements. [9] provides a good survey of known algorithmic techniques used in compressive sensing.

[1] introduced Model Based Compressive Sensing where the sparse signal follows a certain predetermined structure. This effectively reduces the space of possible sparse signals and gives a robust

recovery using a small number of samples. For example, [1] considers signals on a rooted connected tree and shows that such signals can be recovered with $\mathcal{O}(M)$ measurements. Several follow-up works [5, 10, 11, 12] considered other structured models such as connected components in a graph, [13, 14]. One issue with such model based compressive sensing techniques is that the structure needs to be hand-crafted with the available domain knowledge about the signals. Such a handmade design might not be possible for complex signals or signals with limited domain knowledge. Hence, in this work, we consider the problem of learning structure in the input signal without a given prespecified model.

2 Related Work

The study of compressive sensing has many impressive applications, such as rapid magnetic resonance imaging [16], single-pixel camera [18], and UAV systems. One of the popular assumptions on the input is sparsity [17]. Several approaches have been proposed for sparsity-based signal recovery. These approaches include alternating minimization [19], methods based on convex optimization [20], and Iterative Thresholding [21]. Although sparsity is the most common input structure, it is natural to study if this notion can be further refined in order to capture more complex structures. Such an approach is applicable in many applications; for example, large wavelet coefficients for natural images can often be organized as a *connected tree*, point sources in the astronomical images tend to form a *cluster*, and active genes can be arranged as a *group*. These structured sparsity models have been shown to achieve faster algorithms along with improved sample complexity for statistical learning and sparse recovery. Hence, recently, several sparse recovery model with distinct structural properties have been studied – block-sparsity [22], tree sparsity [1,23], cluster [5,24], and graph models [1,5,25].

Recently, there has been work on using machine learning and deep-learning models for sparse compressive sensing [3,6, 28]. [3] developed the first generative model, which outputs the unknown signal x. Here, the goal is to model the real-life distribution of x. In follow-up work, [6] developed a *task aware* generative model with a focus on the recovery task and showed a significant improvement over [3]. [29] applied neural gradient descent for compressive sensing. In another line of work, [26, 27] developed classification models on compressed data itself.

In this work, we develop a reinforcement learning based model. In general, reinforcement learning has been employed on a wide variety of optimization problems [32,33]. Notice that our problem of learning the unknown structured input distribution on a graph is a combinatorial optimization problem. Reinforcement learning based models have shown a recent success in a wide variety of such problems [2,4,8]. In particular, models achieve impressive accuracy for combinatorial optimization problems on graphs [4,7,8,30]. Following this path, our work also uses a reinforcement learning approach.

3 Methods

Many existing CS algorithms, such as Iterative Hard Thresholding (IHT), iteratively check the differences between the ground-truth measurements and the measurements obtained with the current estimation of the original signal. The small differences are pruned out and the estimation is updated along the directions of the large differences. In model-based CS, the small differences are also pruned but in a way that complies with the underlying model. Model-based pruning requires prior knowledge of the signal model and designing the pruning function. Our method can be described as "learning to prune", which accounts for the cases where the signal model is not known or well-defined. Recently, there has been a surge in work on machine learning for combinatorial optimization problems, in which reinforcement learning (RL) is a well-adapted learning paradigm due to the lack of labels during training.

We chose to learn pruning functions (step 3, Figure 1) of the Model-based IHT algorithm because of its simplicity. For the signal, we chose to optimize graph signals because of their generality and their ability to represent a variety of structured signals. Note that the pruning itself is an optimization problem, so in the context of machine learning for combinatorial optimization, our method can both be considered a helper function for the overall optimization algorithm (IHT) and an end-to-end optimization function (pruning).

Algorithm 2 Model-based Iterative Hard Thresholding

Inputs: CS matrix Φ, measurements y, structured sparse approximation algorithm $\mathbb{M}_K$
Output: K-sparse approximation $\widehat{x}$ to true signal x
$\widehat{x}_0 = 0$, $d = y$; $i = 0$ {initialize}
while halting criterion false **do**
1. $i \leftarrow i + 1$
2. $b \leftarrow \widehat{x}_{i-1} + \Phi^T d$ {form signal estimate}
3. $\widehat{x}_i \leftarrow \mathbb{M}(b, K)$ {prune residual estimate according to structure}
4. $d \leftarrow y - \Phi\widehat{x}_i$ {update measurement residual}

end while
return $\widehat{x} \leftarrow \widehat{x}_i$

Figure 1: Model based IHT [1]

In this project, we have gone through a total of 3 development cycles, during each of which substantial changes were made to improve the performance of the model. In the last cycle, the model was able to learn to "imitate" the choices made by the IHT algorithm but not able to go beyond and learn the underlying structure of the signals. Following this, we will describe our design choices for each cycle by specifying the corresponding state, the action, and the reward function of the RL model. In all cycles, the graph signal embeddings were learned via the graph neural network (GNN) defined in [31], and the RL model was a Deep Q-learning network.

3.1 Cycle 1

In this cycle, the state of the system is the current state of the IHT optimization step and each pruning step corresponds to a single action. Given a graph $G = (V, E)$, let X be the signal and $\hat{X}_i$ be the estimation of the signal at the i^{th} iteration. An action selects a set $S_i = \{v \in V | \hat{x}_{i,v} \neq 0\}$ to zero out (pruning). IHT follows and updates the next estimation $\hat{X}_{i+1}$ based on the current pruned estimation.

We use a GNN to produce the embedding μ_v for each node $v \in V$, of which $\hat{x}_{i,v}$ is the only input feature. The state embedding $\mu_{\hat{X}_i}$ is $\texttt{SUM}\{\mu_{i,v} | v \in V\}$. Similarly, the action embedding μ_{S_i} is $\texttt{SUM}\{\mu_{i,v} | v \in Si\}$. The concatenation $[\mu_{\hat{X}_i}, \mu_{S_i}]$ is passed into the deep Q-network to output the expected long term return by following the standard RL policy. At each step, the immediate return is $\frac{1}{||d||}$ where d is the residual vector (step 4, Figure 1).

An apparent drawback of the above design is the size of the action space, which scales exponentially with larger graphs. Moreover, given the enormous amount of possible actions, the convergence of the deep Q-network becomes intractable since the number of learning scenarios that we need to generate is many folds larger.

3.2 Cycle 2

In this cycle, we tackled the intractability of the previous design by some major updates to the definitions of states and actions. Instead of choosing a single set of nodes to zero out in one pruning step, we split the pruning into multiple greedy steps. As such, an episode is defined over a single pruning step instead of the whole IHT optimization. We also need to redefine some notations from Cycle 1.

Given a graph $G = (V, E)$, let X be the signal and $\hat{X}$ be the estimation of the signal at the beginning of the episode, i.e, a single pruning step of IHT. Over multiple greedy steps, we will build a set of nodes S to be pruned out at the end of the episode. In this case, S_i denotes the set of nodes after the i^{th} greedy step.

Similarly to those of Cycle 1, μ_v is the embedding of node $v \in V$, $\mu_X = \texttt{SUM}\{\mu_v | v \in V\}$ is the graph embedding, and $\mu_{Si} = \texttt{SUM}\{\mu_v | v \in Si\}$ is the current pruned-out set's embedding. At each greedy step, we pick a candidate node c such that $c \in V \backslash S_i$ and $\hat{x}_c \neq 0$ to append S_i. The concatenation $[\mu_{\hat{X}}, \mu_{S_i}, \mu_c]$ is passed into the deep Q-network to predict the long-term cumulative

reward for adding c to S_i. After each step, the reward is the reduction of $||d||$ compared to that of the previous step.

Even with the simplification we added in Cycle 2, the performance of our method was still significantly worse than that of IHT. In Cycle 3, we made an adjustment that ensures our method will be at least as good as IHT.

3.3 Cycle 3

In this cycle, we exploited the fact that IHT already makes decent choices and the model should only try to improve based on those. As a result, we initiate S_i to the set of pruned-out nodes chosen by IHT. At each greedy step, the model will swap a node in S_i with another non-zero node. There is an option for the model to pick "no action", which simply means that it agrees with IHT's choices.

Because of the swapping actions, the candidate node c from Cycle 2 does not make sense anymore. Instead, an action is defined by a pair of nodes that includes a to-be-added node a, $a \in V \backslash S_i$, $\hat{x}_a \neq 0$, and a to-be-removed node r, $r \in S_i$. The concatenation $[\mu_{\hat{X}}, \mu_{S_i}, \mu_a, \mu_r]$ is passed into the deep Q-network to predict the long-term cumulative reward for swapping r with a. If "no action" is selected, then $\mu_r = \mu_a = 0$.

4 Experimental Results

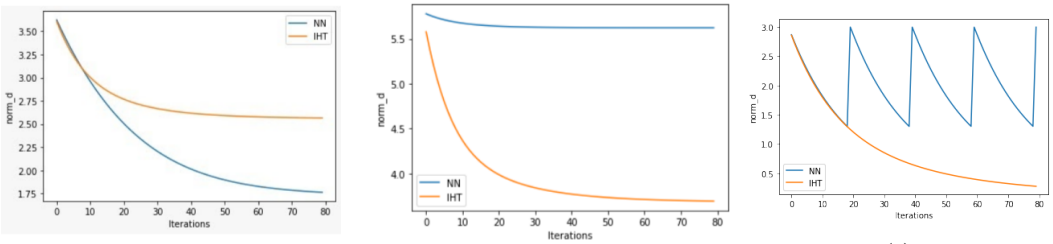
4.1 Dataset

We generated a synthetic dataset of graph signals where the structure of the signals follows a simple predefined model. Each graph was randomly generated according to the Erdős–Rényi model. We randomly selected a small portion of the nodes to assign non-zero values such that the connected components of these nodes form line graphs. The size of each graph, the proportion of non-zero nodes, and the value of non-zero nodes all follow Gaussian distributions. In total, 200 such graphs were generated.

4.2 Recovery performance

We now present our results for cycle 2 and cycle 3 (section 4.3). For cycle 1, due to the large action space, we could not sufficiently train our Q-learning agent.

Training. We train each model for at least 5000 episodes (some of our models are trained for 10000 episodes). Our other default parameters are: $\gamma = 0.9$ (learning parameter), $\epsilon_{start} = 0.9, \epsilon_{end} = 0.05, \epsilon_{decay} = 2500$ (exploration threshold).



(a) (b) (c)

Figure 2: Recovery scenarios for cycle 2. Blue represents plot for the model and olive represents plot for IHT.

Cycle 2. Figure 2 presents a few scenarios that we have observed when comparing the performance of the model from Cycle 2 with that of IHT on the synthetic dataset. In each plot, the norm of the residue vector is plot against the number of optimizing iteration. The norm is 0 when the signal is perfectly recovered. Figure 2b shows the most dominant scenario in which there is little recovery by the RL model as compared to that of IHT. Since the curve produced by the RL model is considerably

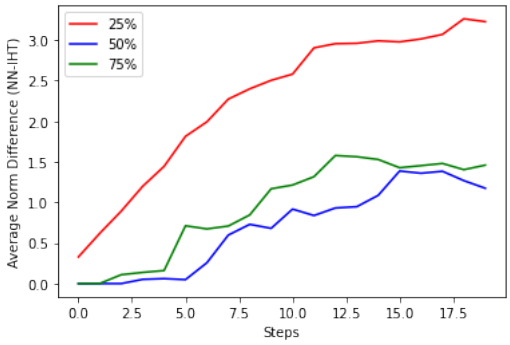


Figure 3: The average difference between the norm of the residual vector produced by the DQN model and the one produced by IHT over 20 steps of optimization. Each curve corresponds to a different level of exploration.

Table 1: The effect of Exploration on Performance

Exploration	**Norm Difference (NN-IHT)**
25%	45.385
50%	**13.714**
75%	18.931

smooth, we could say that the small recovery did not happen by chance but by the model's learning, even though the learning is quite limited. Occasionally, the RL model performed similarly or even better than IHT, as shown in Figure 2a. In a few other cases, the RL models started similarly as IHT did but got off track and stuck in a recovery loop, as shown in Figure 2c.

Cycle 3. Our default exploration rate is set to 50 percent for this case. We observed that our model predominantly imitates IHT, and the two graphs are almost identical. Hence, in terms of available options in each step, the model mostly picks "no action". We omit the figure due to space constraints (and due to the similarity of IHT and model behavior). On the positive side, our model do not behave as model from Cycle 2 in Figure 2b or Figure 2c. But on the other hand, the learning is still limited, and the model outperforms IHT only in a handful of cases. We believe that since we only 'swap' one pair of nodes as our action, the model's capacity to understand graph patterns is limited. A way around this would be to a 'swap' with multiple nodes simultaneously, but such a model would incur high computation cost.

4.3 The effect of exploration

We conducted some experiments on the effect of the exploration-exploitation problem on the learning of our model. The number of exploration episodes within the 5000 training episodes is varied. In particular, we tested the model when 25 percent, 50 percent, and 75 percent of the training episodes are dedicated for exploration and recorded the difference between the norms of the residual vector d produced by the DQN model and IHT. The results are reported in Figure 3 and Table 1.

In general, more exploration is beneficial up to some point. By increasing the percentage of exploration episodes from 25 to 50, we improved the performance of the model significantly. However, the performance slightly deteriorated when we increase the percentage to 75.

5 Conclusion and Future Work

The preliminary results we obtained from this project suggested that, with a certain amount of training, an RL method should perform equally to an existing algorithm for sparse recovery. However, going beyond and learning the underlying model of the signals is still a significant challenge, even on the small and simple synthetic graph signals that we generated. Given that RL is a generally demanding

learning paradigm, multiple factors might have hindered our model from performing at the desired level, suggesting future improvements.

In terms of training, since each episode corresponds to a different graph and the number of actions at each step is quite large and diverse, it is natural that the deep Q-network requires a large number of training episodes. However, due to the project's time constraint, we were only able to train for 5000 to 10000 episodes. Note that it is common for a deep RL model to be trained on millions of episodes.

In terms of representation, at the moment, the complexity of our model is still limited. A more complicated underlying signal model may require a more capable graph representation learning method. The GNN currently used in our project is quite simple in that it may not be able to capture the graph structures effectively. Moreover, we only used a single node feature as the input into the GNN, while successful RL methods for combinatorial optimization on graphs often develop several meaningful features based on the problem.

In conclusion, the outcome of our project is limited yet promising. Further developments that address the limitations mentioned above are necessary to improve the performance.

References

[1] Richard G Baraniuk, Volkan Cevher, Marco F Duarte, and Chinmay Hegde. Model-based compressive sensing. IEEE Transactions on information theory, 56(4):1982–2001, 2010.

[2] Thomas D Barrett, William R Clements, Jakob N Foerster, and AI Lvovsky. Exploratory combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1909.04063, 2019.

[3] Ashish Bora, Ajil Jalal, Eric Price, and Alexandros G Dimakis. Compressed sensing using generative models. In International Conference on Machine Learning, pages 537–546. PMLR, 2017.

[4] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In Advances in Neural Information Processing Systems, volume 32, 2019.

[5] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. A nearly-linear time framework for graph-structured sparsity. In International Conference on Machine Learning, pages 928–937. PMLR, 2015.

[6] Maya Kabkab, Pouya Samangouei, and Rama Chellappa. Task-aware compressed sensing with generative adversarial networks. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 32, 2018.

[7] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In Advances in Neural Information Processing Systems, volume 31, 2018.

[8] Sahil Manchanda, Akash Mittal, Anuj Dhawan, Sourav Medya, Sayan Ranu, and Ambuj Singh. Gcomb: Learning budget-constrained combinatorial algorithms over billion-sized graphs. In Advances in Neural Information Processing Systems, volume 33, 2020.

[9] Stanković, Srdjan, Cornel Ioana, Xiumei Li, and Vladan Papic. In Algorithms for Compressive Sensing Signal Reconstruction with Applications (2016).

[10] Jagatap, Gauri, and Chinmay Hegde. "Fast, sample-efficient algorithms for structured phase retrieval." Proceedings of the 31st International Conference on Neural Information Processing Systems. 2017.

[11] Cevher, Volkan, et al. Sparse signal recovery using markov random fields. RICE UNIV HOUSTON TX DEPT OF ELECTRICAL AND COMPUTER ENGINEERING, 2009.

[12] Hegde, Chinmay, Piotr Indyk, and Ludwig Schmidt. "Approximation-tolerant model-based compressive sensing." Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2014.

[13] Huang, Junzhou, Tong Zhang, and Dimitris Metaxas. "Learning with Structured Sparsity." Journal of Machine Learning Research 12.11 (2011).

[14] Jagatap, Gauri, and Chinmay Hegde. "Sample-efficient algorithms for recovering structured signals from magnitude-only measurements." IEEE Transactions on Information Theory 65.7 (2019): 4434-4456.

[15] Cevher, Volkan, et al. Recovery of clustered sparse signals from compressive measurements. RICE UNIV HOUSTON TX DEPT OF ELECTRICAL AND COMPUTER ENGINEERING, 2009.

[16] Lustig, Michael, et al. "Compressed sensing MRI." IEEE signal processing magazine 25.2 (2008): 72-82.

[17] Lustig, Michael, David Donoho, and John 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 58.6 (2007): 1182-1195.

[18] Duarte, Marco F., et al. "Single-pixel imaging via compressive sampling." IEEE signal processing magazine 25.2 (2008): 83-91.

[19] Netrapalli, Praneeth, Prateek Jain, and Sujay Sanghavi. "Phase retrieval using alternating minimization." IEEE Transactions on Signal Processing 63.18 (2015): 4814-4826.

[20] Jaganathan, Kishore, Samet Oymak, and Babak Hassibi. "Sparse phase retrieval: Convex algorithms and limitations." 2013 IEEE International Symposium on Information Theory. IEEE, 2013.

[21] Cai, T. Tony, Xiaodong Li, and Zongming Ma. "Optimal rates of convergence for noisy sparse phase retrieval via thresholded Wirtinger flow." The Annals of Statistics 44.5 (2016): 2221-2251.

[22] Eldar, Yonina C., Patrick Kuppinger, and Helmut Bolcskei. "Block-sparse signals: Uncertainty relations and efficient recovery." IEEE Transactions on Signal Processing 58.6 (2010): 3042-3054.

[23] Hegde, Chinmay, Piotr Indyk, and Ludwig Schmidt. "A fast approximation algorithm for tree-sparse recovery." 2014 IEEE International Symposium on Information Theory. IEEE, 2014.

[24] Cevher, Volkan, et al. Recovery of clustered sparse signals from compressive measurements. RICE UNIV HOUSTON TX DEPT OF ELECTRICAL AND COMPUTER ENGINEERING, 2009.

[25] Hegde, Chinmay, Piotr Indyk, and Ludwig Schmidt. "Approximation-tolerant model-based compressive sensing." Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2014.

[26] Lohit, Suhas, et al. "Reconstruction-free inference on compressive measurements." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. 2015.

[27] Lohit, Suhas, Kuldeep Kulkarni, and Pavan Turaga. "Direct inference on compressive measurements using convolutional neural networks." 2016 IEEE International Conference on Image Processing (ICIP). IEEE, 2016.

[28] Mardani, Morteza, et al. "Deep generative adversarial neural networks for compressive sensing MRI." IEEE transactions on medical imaging 38.1 (2018): 167-179.

[29] Mardani, Morteza, et al. "Neural proximal gradient descent for compressive imaging." arXiv preprint arXiv:1806.03963 (2018).

[30] Dai, Hanjun, et al. "Learning combinatorial optimization algorithms over graphs." arXiv preprint arXiv:1704.01665 (2017).

[31] Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016)

[32] Andrychowicz, Marcin, et al. "Learning to learn by gradient descent by gradient descent." arXiv preprint arXiv:1606.04474 (2016).

[33] Cappart, Quentin, et al. "Combining reinforcement learning and constraint programming for combinatorial optimization." arXiv preprint arXiv:2006.01610 (2020).