Neural Matching Pursuit: OMP Meets Reinforcement Learning

Fin Amin, *Member, IEEE*

***Abstract*—In this letter, we introduce Neural Matching Pursuit (NMP), an innovative approach that synergizes the principles of Orthogonal Matching Pursuit (OMP) with the adaptive decision-making capabilities of Deep Q-Networks (DQNs) in reinforcement learning. Traditional sparse coding techniques like OMP, while effective, often face limitations in handling highly correlated features or complex signal structures. NMP addresses these challenges by employing a novel decision strategy within a reinforcement learning framework, providing a more dynamic and context-sensitive selection of atoms for signal reconstruction. Our proposed method, alternates between traditional OMP and a new strategy. This novel strategy diverges from the conventional greedy approach of OMP by selecting atoms based on their minimal correlation with already chosen features, aiming to diversify the feature selection process. This approach not only enhances the robustness of signal reconstruction but also demonstrates potential efficiency improvements, especially in scenarios with intricate feature interdependencies.**

***Index Terms*—sparse-coding, orthogonal matching pursuit, reinforcement learning, dictionary, sparse approximation**

I. Introduction, Background and Motivation

The quest for efficient and accurate signal reconstruction has long been a cornerstone of signal processing, with sparse coding standing out as a particularly powerful technique. Central to this approach is the Orthogonal Matching Pursuit (OMP) algorithm, renowned for its ability to recover signals using a minimal number of samples. However, despite its widespread adoption, OMP encounters limitations when dealing with highly correlated or complex data structures. These challenges become more pronounced in scenarios involving high-dimensional datasets, where traditional greedy selection strategies may not suffice.

A. Problem Background

Dictionary learning and sparse coding are fundamental in representing signals efficiently. The central premise is to express a signal as a linear combination of a few elements (atoms) from a predefined or learned dictionary. Let's define the mathematical foundations of this process.

Consider a signal $\mathbf{y} \in \mathbb{R}^n$. The goal of dictionary learning is to find a dictionary $\mathbf{D} \in \mathbb{R}^{n \times m}$ and a sparse coefficient vector $\mathbf{x} \in \mathbb{R}^m$ such that:

$$\mathbf{y} \approx \mathbf{D}\mathbf{x}$$

Here, $\mathbf{D}$ consists of m atoms (columns), each of length n. The key is to ensure that $\mathbf{x}$ has as few non-zero entries as possible, signifying sparsity.

Preprint awaiting submission
Fin Amin is with North Carolina State University located in Raleigh, NC.

The Orthogonal Matching Pursuit (OMP) algorithm iteratively selects atoms from the dictionary that are best correlated with the residual of $\mathbf{y}$ at each step. The algorithm can be summarized as follows:

Algorithm 1 Orthogonal Matching Pursuit (OMP)

Require: Signal $\mathbf{y}$, Dictionary $\mathbf{D}$, Number of atoms to use m_s
Ensure: Sparse Coefficient Vector $\mathbf{x}$
$\Lambda \leftarrow \emptyset$ {Initialization of selected atoms}
$\mathbf{r} \leftarrow \mathbf{y}$ {Initial residual}
while $|\Lambda| < m_s$ **do**
  $k \leftarrow \arg\max_{i \notin \Lambda} |\langle \mathbf{d}_i, \mathbf{r} \rangle|$ {Greedy atom selection}
  Update Λ to include k
  $\mathbf{z}_\Lambda \leftarrow \arg\min_{\mathbf{z}} \|\mathbf{y} - \mathbf{D}_\Lambda \mathbf{z}\|_2$ {Find coefficients for selected atoms}
  $\mathbf{x}_\Lambda \leftarrow \mathbf{z}_\Lambda$ {Update sparse coefficient vector}
  $\mathbf{r} \leftarrow \mathbf{y} - \mathbf{D}_\Lambda \mathbf{x}_\Lambda$ {Updating residual}
end while
return $\mathbf{x}_\Lambda$

In essence, OMP is a greedy[1] algorithm that aims to approximate $\mathbf{y}$ by iteratively building a sparse representation. The choice of atoms at each step is driven by their correlation with the current residual, ensuring that the most relevant features are captured in the approximation. The algorithm terminates once m_s atoms are selected; m_s can be interpreted as the sparsity parameter of the algorithm. The smaller m_s is, the fewer atoms will be selected to reconstruct the signal. Note that selecting the optimal subset of atoms from a dictionary to minimize the reconstruction error is known to be NP-Hard. An additional remark is that some OMP implementations forgo selecting m_s and instead terminate once r is under some threshold, ie. they select atoms until a tolerable reconstruction error is achieved.

B. Motivation

The greedy decision to sequentially select the next atom is a heuristic which, although effective, potentially has room for improvement. We surmise that an alternative strategy for selecting atoms can be beneficial. We introduce Neural Matching Pursuit (NMP), an innovative approach that synergizes the robustness of sparse coding with the adaptability of Deep Q-Networks (DQNs) in reinforcement learning. The genesis of NMP lies in addressing the limitations of OMP by incorporating

[1]A greedy algorithm is an algorithm which iteratively makes the locally optimal choice. The hope is that by successively selecting the locally optimal choice, a global optimum is achieved.

the learning and decision-making prowess of neural networks. This integration is not merely a fusion of methodologies but a significant stride towards a more dynamic and context-sensitive approach to feature selection in signal reconstruction. NMP distinguishes itself by introducing a novel decision strategy that complements the traditional OMP method. Instead of solely relying on the greedy selection of the most correlated features, NMP employs a strategy that factors in the overall coherence and diversity of the feature set, aiming to enhance the reconstruction accuracy.

In this section we give the preliminary background of RL, DQNs and the interactions agents make with the *environment*. Afterwards, we introduce the NMP algorithm.

II. Reinforcement Learning, Environments, and Deep Q-Learning

Reinforcement Learning is a type of machine learning where an agent learns to make decisions by performing actions in an environment to achieve a goal. The agent receives feedback in the form of rewards or penalties based on its actions from the environment. In RL, the environment represents the problem or the space in which the agent operates. It provides the agent with states[2], and in response to the agent's actions, it presents the next state and a reward signal. The objective is to learn a policy, π, that maximizes the cumulative reward over time.

The **Value function** $V^\pi(s)$ under a policy π is defined as the expected return (cumulative discounted reward) when starting in state s and following policy π thereafter:

$$V^\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right], \tag{1}$$

where γ is the discount factor, and R_{t+k+1} is the reward at time $t+k+1$.

The **Q-function** $Q^\pi(s,a)$, or action-value function, under policy π, is the expected return after taking an action a in state s and then following policy π:

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]. \tag{2}$$

During training, the Q-function is typically updated using the Bellman equation as follows:

$$\begin{aligned} Q^{new}(s_t,a_t) \leftarrow (1-\alpha) Q(s_t,a_t) \\ + \alpha\left(r_t + \gamma \max_a Q(s_{t+1},a)\right), \end{aligned} \tag{3}$$

where α is the learning rate, s_t and s_{t+1} are the current and next state, a_t is the current action, and r_t is the reward received after taking action a_t in state s_t.

Deep Q-Learning is an advanced form of Q-learning, enhanced with deep neural networks, known as Deep Q-Networks. The key idea is to use a deep neural network to approximate the Q-value function, which represents the expected **cumulative** reward of taking an action in a given state, following a particular policy. Considerations in the DQN algorithm involve:

[2]A source of confusion comes from the way literature treats *Exploration vs. Exploitation*: Balancing between exploring the environment to find rewarding actions and exploiting known information to make the best decisions. This is typically done by an exploration policy. Common choices for exploration policies are epsilon greedy and Boltzmann

Experience Replay: Storing the agent's experiences and randomly sampling from this memory to break the correlation between consecutive learning steps and stabilize training.

Target Networks: Employing a separate network to estimate the Q-values, which is updated less frequently to further stabilize learning.

III. The Neural Matching Pursuit Algorithm

In this section, we first outline the NMP algorithm. Afterwards, we explain the actual structure of the DQN agent along with the environment it interacts with. Furthermore, we explain how the agent is trained.

Algorithm 2 Neural Matching Pursuit (NMP)

Require: Signal $\mathbf{y}$, Dictionary $\mathbf{D}$, Number of atoms m_s, RL environment, $\mathbf{env}$, Trained RL agent, $agent$.
Ensure: Sparse Coefficient Vector $\mathbf{x}$
1: $\Lambda \leftarrow \emptyset$ {Initialization of selected atoms}
2: $\mathbf{r} \leftarrow \mathbf{y}$ {Initial residual}
3: $\mathbf{o} \leftarrow \mathbf{env_0}$ {Initial environment observation}
4: **while** $|\Lambda| < m_s$ **do**
5: $a \leftarrow agent(\mathbf{o})$ {Agent selects action}
6: $k, \mathbf{o}^+ \leftarrow \mathbf{env}(a)$ {Environment selects atom given a}
7: Update Λ to include k
8: $\mathbf{z}_\Lambda \leftarrow \arg\min_{\mathbf{z}} \|\mathbf{y} - \mathbf{D}_\Lambda \mathbf{z}\|_2$ {Find coefficients for selected atoms}
9: $\mathbf{x}_\Lambda \leftarrow \mathbf{z}_\Lambda$ {Update sparse coefficient vector}
10: $\mathbf{r} \leftarrow \mathbf{y} - \mathbf{D}_\Lambda \mathbf{x}_\Lambda$ {Updating residual}
11: $\mathbf{o} \leftarrow \mathbf{o}^+$
12: **end while**
13: **return** $\mathbf{x}_\Lambda$

Note that to facilitate our explanation, we refer to the iterations of the `while − loop` in line 4 of algorithm 2 as the "time-steps", t, of the RL algorithm. The key difference between NMP and OMP lies in step 5 of the NMP algorithm, where the agent selects between two actions:

- Action 0, the typical greedy OMP decision of selecting the atom most correlated with the residual.
  $k \leftarrow \arg\max_{i \notin \Lambda} |\langle \mathbf{d}_i, \mathbf{r} \rangle|$
- Action 1, the new decision of selecting the atom which is on average least correlated with the already selected atoms.
  $k \leftarrow \arg\min_{i \notin \Lambda} \left(\frac{1}{|\Lambda|} \sum_{j \in \Lambda} |\langle \mathbf{d}_i, \mathbf{d}_j \rangle|\right)$

These two actions are generally distinct, especially in the context of an over-complete dictionary. While action 0 focuses on immediate correlation with the residual, potentially leading to a local optimum, action 1 looks beyond immediate correlation to enhance the overall representational quality of the selected atom set. This dual-action approach allows NMP to potentially outperform traditional OMP in complex scenarios, offering a more balanced and nuanced method for atom selection in sparse coding.

A. The NMP Environment

insert pic here

The NMP environment is what takes an action from the agent and returns and observation, $\mathbf{o}$ and reward $\mathbf{r}$. It is also responsible for computing which atom to add to Λ with respect to the input action. We abuse notation and declare the reward returned after each step as the scaled-norm of the residual, $\mathbf{r} = \frac{\|r\|_2}{\|y\|_2}$. The observation returned by the environment is more complex. $\mathbf{o}$ is the following 5-tuple:

- **Residual Norm**: The scaled-norm of the current residual, calculated as $\frac{\|r\|_2}{\|y\|_2}$. This represents the magnitude of the difference between the original signal and the current approximation.
- **Determinant of the Gram Matrix**: The determinant of the Gram matrix formed by the selected atoms. It is set to 0 when no atoms are selected, and for selected atoms, it is calculated as $\det(\Lambda^T\Lambda)$.
- **Mutual Coherence of Selected Atoms**: The maximum absolute value of the off-diagonal elements of the normalized Gram matrix of the selected atoms, quantifying the maximum similarity between any two selected atoms. Mathematically, it is represented as:

$$\max_{i \neq j} \left| \frac{\langle \mathbf{d}_i, \mathbf{d}_j \rangle}{\|\mathbf{d}_i\| \|\mathbf{d}_j\|} \right|, \text{ for } i, j \in \Lambda$$

- **Mutual Coherence of Unselected Atoms**: The maximum absolute value of the off-diagonal elements of the normalized Gram matrix of the unselected atoms, representing the maximum similarity between any two unselected atoms. This can be expressed as:

$$\max_{i \neq j} \left| \frac{\langle \mathbf{d}_i, \mathbf{d}_j \rangle}{\|\mathbf{d}_i\| \|\mathbf{d}_j\|} \right|, \text{ for } i, j \in \mathbf{D} - \Lambda$$

- **Progress Metric**: The ratio of the number of selected atoms to the total number of non-zero coefficients allowed, calculated as $\frac{|\Lambda|}{m_s}$. This metric indicates the progress of the atom selection process.

Note that although the dimensions of the signal and the dictionary are arbitrary, the dimension of the observation is five scalar values. This allows the agent to generalize across dictionaries/signals of various size. As shown in algorithm 2 and figure **xyz**, the agent and environment exchange information until the the signal is reconstructed. The environment sends the aforementioned observations and reward signals to the agent. The agent uses the observations returned by the environment to make a decision on which atom to select. This decision is then returned to the environment so that the selected atoms are updated. In our implementation we use Tensorflow [1] and **cite keras rl**.

B. DQN Training

The network described in figure 1 is trained to estimate the Q-Value given

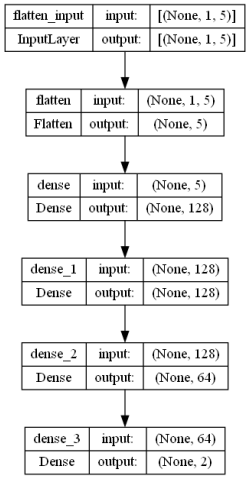

Fig. 1: The network used by the DQN Agent. Each layer is followed by a ReLU activation.

IV. Conclusion

Acknowledgment

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.