Neural Matching Pursuit: OMP Meets Reinforcement Learning

Fin Amin, *Member, IEEE*
North Carolina State University
Raleigh, NC
samin2@ncsu.edu

***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) [8] 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 [11], [6], [5], [4]. 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.

The 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 $\mathbf{x}$
1: $\Lambda \leftarrow \emptyset$ {Initialization of selected atoms}
2: $\mathbf{r} \leftarrow \mathbf{y}$ {Initial residual}
3: while $|\Lambda| < m_s$ do
4:     $k \leftarrow \arg\max_{i \notin \Lambda} |\langle \mathbf{d}_i, \mathbf{r} \rangle|$ {Greedy atom selection}
5:     Update $\Lambda$ to include $k$
6:     $\mathbf{z}_\Lambda \leftarrow \arg\min_{\mathbf{z}} \|\mathbf{y} - \mathbf{D}_\Lambda \mathbf{z}\|_2$ {Find coefficients for selected atoms}
7:     $\mathbf{x}_\Lambda \leftarrow \mathbf{z}_\Lambda$ {Update sparse coefficient vector}
8:     $\mathbf{r} \leftarrow \mathbf{y} - \mathbf{D}_\Lambda \mathbf{x}_\Lambda$ {Updating residual}
9: end while
10: 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

[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.

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) [7] in reinforcement learning. The genesis of NMP lies in addressing the limitations of OMP by incorporating the learning and decision-making prowess of neural networks. 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. 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:

Exploration vs. Exploitation: Balancing between exploring the environment to find rewarding actions and exploiting known information to make the best decisions. DQNs are *off policy* algorithms, which means that they use a separate decision policy for training vs. inferencing. In our work, we use the $\texttt{Boltzman}(\tau = 2)$ policy for exploration and GREEDYQ for testing.

Experience Replay: Storing the agent's experiences and randomly sampling from this memory to break the correlation between consecutive learning steps and stabilize training. In our work, we use a simple sequential memory container for this.

Target Networks: Employing a separate copy of the network to estimate the Q-values, which is updated less frequently to further stabilize learning; we refer to the this network as the "target" network and the network which learns from observations as the "online network". We explain when we update our target network in algorithm 3.

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 DQN, $agent$.
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 $\Lambda$ 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 $\texttt{while} - \texttt{loop}$ 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:

- Action0, the typical greedy OMP decision of selecting the atom most correlated with the residual.
  $k \leftarrow \arg\max_{i \notin \Lambda} |\langle \mathbf{d}_i, \mathbf{r}\rangle|$

[2]A source of confusion comes from the way our work and existing literature use the words "states" and "observations" interchangeably. There exist many scenarios where the entirety of the state is not observable, so the observation returned by the environment incomplete. In our context, this difference is not important, so $\mathbf{s} = \mathbf{o}$.

- `Action1`, 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, Λ. 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

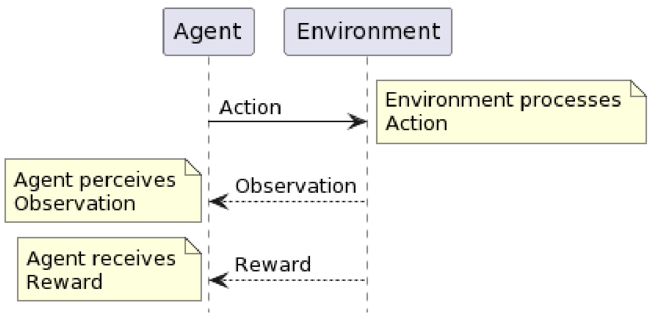

Fig. 1: The flow of information over one time-step of the RL training loop [10]. In our context, there are m_s time steps per *episode*. Here the episode denotes the reconstruction of a single image using m_s atoms.

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 1, 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 Keras-RL [2].

B. DQN Architecture and Training

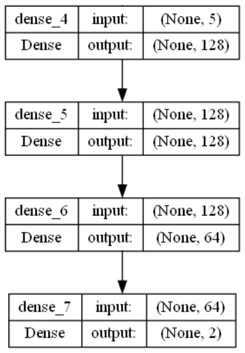
dense_4 | input: | (None, 5)
Dense | output: | (None, 128)
dense_5 | input: | (None, 128)
Dense | output: | (None, 128)
dense_6 | input: | (None, 128)
Dense | output: | (None, 64)
dense_7 | input: | (None, 64)
Dense | output: | (None, 2)

Fig. 2: The network used by the DQN Agent. Each layer aside from the very last is followed by a Tanh activation. The final layer uses ReLu.

The network described in figure 2 is trained to learn the Q-Value given a state. For DQNs, we can not perform the Bellman update (equation 3) directly. Instead, we use $\texttt{SGD}(\eta = \texttt{0.001})$ to minimize the following loss function:

$$L(\theta) = \mathbb{E}\left[\left(Q_{\text{target}}(s, a) - Q(s, a; \theta) \right)^2 \right] \tag{4}$$

where each term represents:

- $L(\theta)$: The MSE loss function, dependent on the parameters θ of the neural network.
- $Q_{\text{target}}(s, a)$: The target Q-value for state s and action a, calculated using the Bellman equation:
$$Q_{\text{target}}(s, a) = r + \gamma \max_{a'} Q(s', a'; \theta^-) \tag{5}$$
where r is the reward received after taking action a in state s, $\gamma = 0.99$ is the discount factor, s' is the new state, a' represents any possible action in the new state, and θ^- are the parameters of the target network.

- $Q(s,a;\theta)$: The predicted Q-value by the current model (with parameters θ) for state s and action a.

We train our network by using a dictionary D produced from a batch of signals, y_{train} in accordance to algorithm 3. We perform this algorithm two times sequentially–in other words, we train over the training signals twice.

Algorithm 3 NMP Agent Training

```
Require: Signal batch $\mathbf{y_{train}}$, Dictionary $\mathbf{D}$, Number of atoms $m_s$, RL environment env, online DQNs $agent_{\pi_b}$, target DQN $agent_t$, memory buffer $M$, exploration policy $\pi_b$.
Ensure: Trained DQN agent
 1: for each signal $\mathbf{y}$ in $\mathbf{y_{train}}$ do
 2:     $\Lambda \leftarrow \emptyset$ {Initialization of selected atoms}
 3:     $\mathbf{r} \leftarrow \mathbf{1}$ {Initial residual}
 4:     $\mathbf{o} \leftarrow \mathbf{env_0}$ {Initial environment observation}
 5:     for $t = 1$ to $m_s$ do
 6:         $a \leftarrow agent_{\pi_b}(\mathbf{o})$ {Agent selects action using exploration policy}
 7:         $k, \mathbf{r}, \mathbf{o}^+, \leftarrow \mathbf{env}(a)$ {Environment processes action}
 8:         Update $\Lambda$ to include $k$
 9:         Store transition $(\mathbf{o}, a, \mathbf{r}, \mathbf{o}^+)$ in $M$.
10:         Get mini batch of transition tuples $m \subset M$.
11:         $\theta_{agent_{\pi_b}} = (\theta_{agent_{\pi_b}} - \eta\nabla_\theta L(\theta_{agent_{\pi_b}}))|_m$ {Update online network using eq. 4 and $m$.}
12:         $\mathbf{o} \leftarrow \mathbf{o}^+$
13:     end for
14:     $agent_t = agent_{\pi_b}$ {Update the target network}
15: end for
```

IV. Experiments

We conduct two experiments, one is intended to analyze the long-horizon decision making of our agent; the second experiment tests robustness. We use the SKLearn [9] library's implementation of OMP and $\mathtt{DictLearning}(\mathtt{fit\_algo} = \mathtt{lars}, \mathtt{pos\_dict} = \mathtt{True})$. Numpy [3] is used to update the sparse coefficient vectors in NMP by using the least squares algorithm. The signals we use for this evaluation are from the DIGITS $\in \mathbb{R}^{64}$ dataset from SKlearn; we do a $50 - 50$ split for training and testing:

1) We measure the MAE of reconstructing our test signals with respect to m_s. By increasing m_s, we force our agent to consider a longer horizon during its decision-making process. For each value of m_s, we train our NMP agent on the training set and measure our performance on the test set.
2) We measure the MAE of reconstructing our test signals with respect to σ^2, the variance of noise added to $\mathbf{y_{train}}$. In other words, we test our algorithm after learning a dictionary and training our agent on $\hat{\mathbf{y_{train}}} = \mathbf{y_{train}} + \mathcal{N}(0, \sigma^2)$, for each value of σ. Note that our test signals are not altered. For this experiment, we fix $m_s = 8$.

We compare our results with OMP as a baseline for both experiments.

A. Experiment 1: Long-Horizon Decision Making

This experiment examines how NMP learns to "think ahead." As m_s inreases, the episodes become longer–this in turn forces the DQN agent to forecast further ahead with respect to which atoms it chooses.

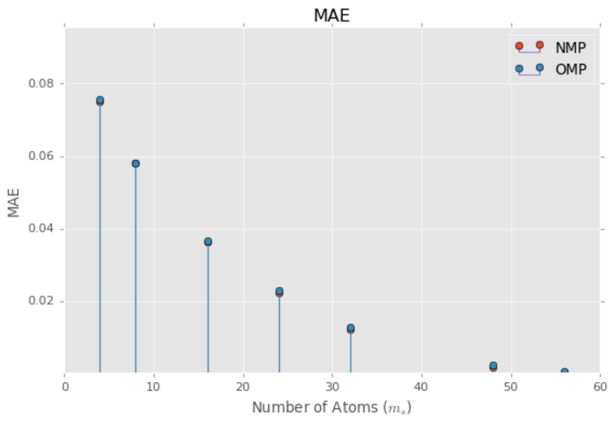

Fig. 3: The results of experiment 1 which analyzes the long-horizon decision making of NMP. The MAE of both OMP/NMP is measured with respect to the number of atoms. NMP very slightly outperforms OMP at $m_s \geq 16$.

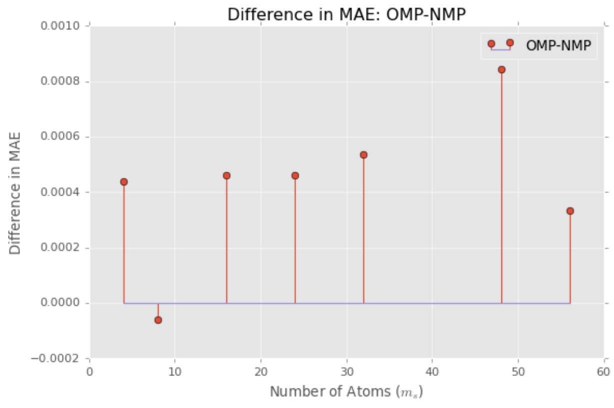

Fig. 4: The difference in MAE between OMP and NMP, in the setting of experiment 1. A positive value indicates that NMP has less error than OMP.

According to figures 3 and 4, NMP offers an incremental upgrade over OMP as the number of atoms increases. Although at first glance this result is not exciting, we note that the maximum possible improvement over OMP *decreases* as $m_s \to m$. The performance of NMP is identical to OMP for $m_s < 16$.

B. Experiment 2: Robustness to a Noisy Dictionary

In this experiment, we simulate the realistic scenario of a dictionary being learned from noisy training samples. The additive noise causes the dictionary to be less optimal with respect to the signals we wish to reconstruct. This scenario appears in many contexts such as image processing, MIMO, and audio signal processing where the training samples are not properly representative of the signals during deployment.

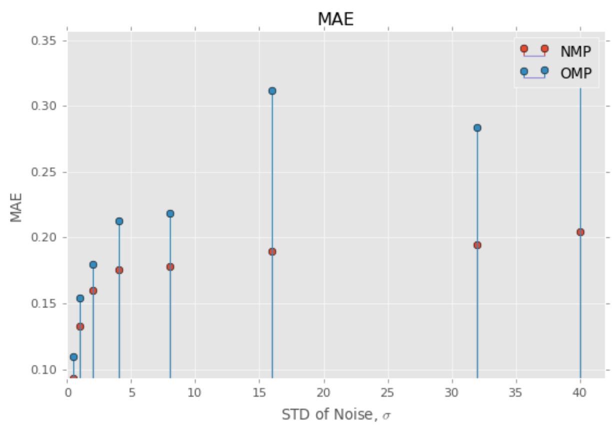

Fig. 5: The results of experiment 2 which analyzes NMP's robustness to a noisy dictionary. The MAE of both OMP/NMP is measured with respect to the noise energy. NMP significantly outperforms OMP.

According to figure 5, NMP significantly outperforms OMP in terms of reconstruction accuracy. There are various explanations to this. One is that the alternative atom-selection strategy, `Action1`, allows NMP to select atoms which are a better fit for the signal. Another explanation is that there is an implementation error within the off-the-shelf OMP algorithm.

V. Conclusion

In this work, we have presented the Neural Matching Pursuit algorithm and evaluated its performance against the traditional OMP in two experiments. Our approach integrates reinforcement learning with sparse coding, aiming to enhance decision-making in atom selection for signal reconstruction. It is important to emphasize that our findings are preliminary and should be interpreted with caution.

The first experiment focused on long-horizon decision-making capabilities of the NMP agent. The results indicate that NMP provides a slight improvement over OMP, particularly when the number of atoms, m_s, is increased. This suggests that NMP might have an advantage in scenarios requiring consideration of a larger set of potential atoms for signal reconstruction, though the extent of this advantage is modest.

The second experiment assessed the robustness of NMP to noise in the training samples, a common scenario in practical applications such as image processing, MIMO communications, and audio signal processing. Here, NMP demonstrated a more significant improvement over OMP, especially at higher noise levels. While these results are encouraging, they also prompt further investigation into whether these improvements stem from the inherent strengths of the NMP algorithm or potential limitations in the OMP implementation used for comparison.

Future work should focus on extensive validation of NMP across various datasets and noise conditions, refinement of the algorithm to enhance its decision-making process, and exploration of potential integrations with other signal processing techniques. Moreover, a deeper analysis into the behavior of NMP, particularly in scenarios with high-dimensional data and extreme noise levels, would be beneficial to fully understand its potential and limitations.

In conclusion, the NMP algorithm represents a novel approach in the realm of sparse coding and signal reconstruction, offering potential improvements over traditional methods. While our results are encouraging, they are but a first step towards a more robust and versatile solution for signal processing challenges.

A. *Limitations*

The results of this study are preliminary and are not decisive. The reasoning behind why NMP performs as well as it does is not yet understood. Our experiments involve only one dataset; our results might not be indicative of global performance. Furthermore, NMP requires computational overhead such as needing to train itself before being able to be deployed.

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.

[2] François Chollet et al. Keras. https://keras.io, 2015.

[3] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[4] Kenneth Kreutz-Delgado, Joseph F Murray, Bhaskar D Rao, Kjersti Engan, Te-Won Lee, and Terrence J Sejnowski. Dictionary learning algorithms for sparse representation. *Neural computation*, 15(2):349–396, 2003.

[5] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th annual international conference on machine learning*, pages 689–696, 2009.

[6] Julien Mairal, Jean Ponce, Guillermo Sapiro, Andrew Zisserman, and Francis Bach. Supervised dictionary learning. *Advances in neural information processing systems*, 21, 2008.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[8] Yagyensh Chandra Pati, Ramin Rezaiifar, and Perinkulam Sambamurthy Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar conference on signals, systems and computers*, pages 40–44. IEEE, 1993.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[10] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[11] Ivana Tošić and Pascal Frossard. Dictionary learning. *IEEE Signal Processing Magazine*, 28(2):27–38, 2011.