Arshia Soltani Moakhar

arshia.soltani2@gmail.com LinkedIn GitHub Website

"SPADE: Sparsity-Guided Debugging for Deep Neural Networks,"

A. Soltani Moakhar*, E. Iofinova*, D. Alistarh, *NeurIPS ATTRIB Workshop, Submitted to ICML 2024 Conference*, 2024, (arXiv)(OpenReview).

In this study, we present an innovative approach to enhance the performance of various interpretability methods by introducing sparsity to the network on a selected sample before applying the interpretability method. This novel technique, named SPADE, significantly improves the performance of saliency maps and feature visualizations.

SPADE operates by sparsifying the network using an image and its corresponding augments, as illustrated in Figure 1. This process results in a sample-aware trace (subnetwork) that can be utilized in conjunction with any interpretability method to provide more accurate results.

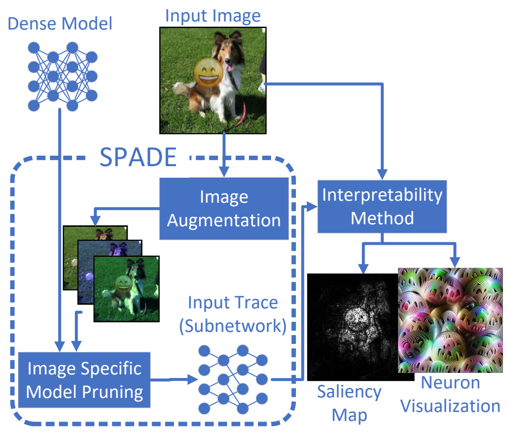


Figure 1: Given an image and a model, SPADE prunes the model using image augmentations.

We demonstrated the effectiveness of SPADE on a backdoored model, where classes respond to both a natural object and certain emojis. In such scenarios, the final neurons exhibit polysemantic behavior, making their interpretation using standard feature visualization challenging. However, as shown in Figure 2, SPADE successfully decouples these concepts, leading to more accurate and interpretable results.

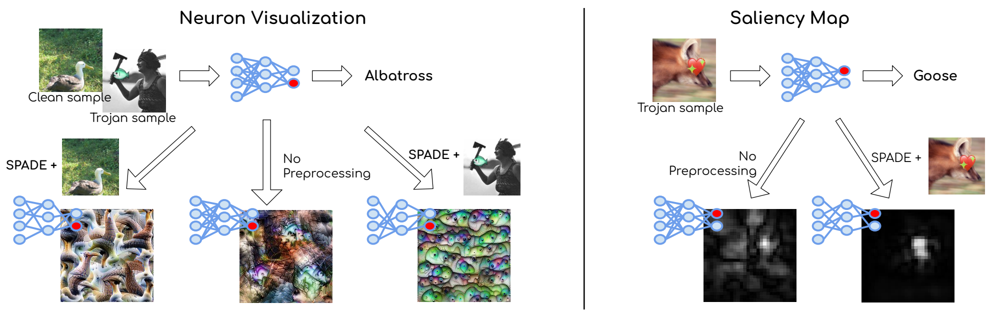


Figure 2: SPADE disambiguates feature visualizations and improves the accuracy of saliency maps. The visualization of the 'Albatross' class neuron consists of a mix of natural and Trojan features, which is difficult to interpret, but preprocessing with SPADE decouples the bird and fish emoji facets. Likewise, preprocessing the model with SPADE before computing a saliency map concentrates it on the Trojan patch, correctly explaining the incorrect prediction.

"Your Out-of-Distribution Detection Method is Not Robust!,"

M. Azizmalayeri, A. Soltani Moakhar, A. Zarei, R. Zohrabi, M.T. Manzuri, M.H. Rohban, *Advances in Neural Information Processing Systems 36*, 2022, (NeurIPS 2022).

Initially, I identified vulnerabilities in existing Robust Out-of-Distribution detection methods to end-to-end adversarial attacks. Subsequently, we proposed an OOD detection algorithm inspired by Generative Adversarial Network (GAN) architecture and adversarial training. In Fig. 3 you can see the overall pipeline of ATD.

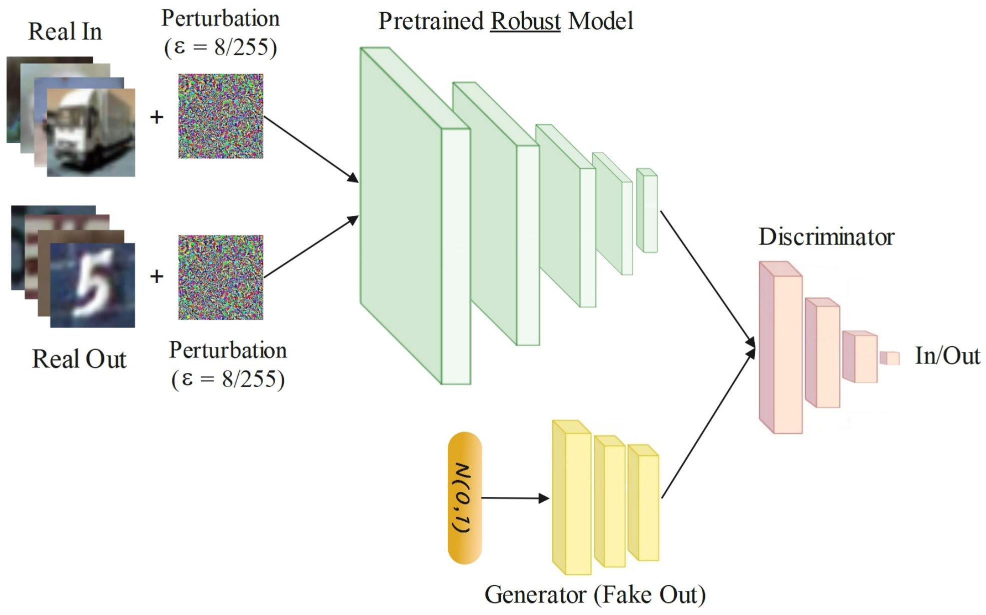


Figure 3: ATD schematic architecture. Generator, discriminator, and robust feature extractor are represented with yellow, pink, and green colors, respectively.

Our out-of-distribution detection model consists of a pre-trained Robust feature extractor and a discriminator. The discriminator network is trained using an Adversarial Generator (GAN-inspired) and adversarial in-distribution and out-of-distribution images (inspired by adversarial training.)

"Seeking Next Layer Neurons' Attention for Error-Backpropagation-Like Training in a Multi-Agent Network Framework,"

A. Soltani Moakhar, M. Azizmalayeri, H. Mirzaei, M.T. Manzuri, M.H. Rohban, *Submitted to ICML 2024 Conference*, 2024, (arXiv).

In this work, I investigated a neural network that has self-interested neurons. It means each neuron is self-interested and tries to maximize its own utility function. The question is *how can we define a **Local** utility function for neurons so that the model learns effectively?* We demonstrated that if neurons try to maximize the norm-2 of the weights of their connections to the neurons in the subsequent layer, and if the neurons in the final layer aim to minimize the model loss, the entire system can learn effectively. We provided proof that an algorithm akin to error-backpropagation represents the Nash equilibrium of this system.

This type of network presents intriguing implications from a collective intelligence standpoint, where the intelligence of the model emerges from a multitude of self-interested entities. Through our experiments, we illustrated that model learning via our algorithm exhibits behaviors typically expected from multi-agent systems. For instance, we demonstrated that the model was capable of adapting to new environments more easily.

“Software 1.0 Strengths for Interpretability and Data Efficiency,”

M. Jabbarishiviari, A. Soltani Moakhar, *ICLR, Tiny Papers*, 2024, .

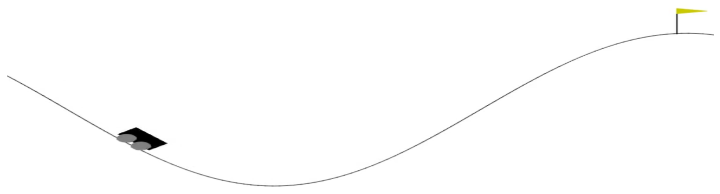
In this collaborative research paper, my colleague and I present a novel interface for reinforcement learning (RL) agents, designed to facilitate seamless integration of reinforcement learning into existing codebases. Our proposed interface shifts the paradigm from the traditional agent-world view, instead positioning RL agents as decision-makers within the code structure.

We introduce the concept of learnable control structures, such as learnable if statements. For training these learnable parts of the code, the programmer provides the library with loss. This loss could be influenced by multiple decisions of numerous agents.

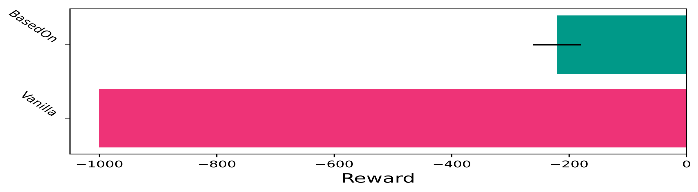
Fig. 4 illustrates a use case of our BasedOn library (a), demonstrating its application in a simple RL task. As anticipated, the programmer’s intuition about the solution can significantly enhance the agent’s performance (c). Moreover, this approach offers a clearer understanding of agent behavior, as opposed to scenarios where the entire decision process is governed by a monolithic neural network. This research aims to make reinforcement learning more accessible and intuitive for programmers, fostering a more symbiotic relationship between human intuition and machine learning capabilities.

```python
def car(state, f):
    x, v = state
    if x < valley and v > 0: # going down
        return right
    if x > valley and v < 0: # going down
        return left
    if x > valley_threshold:
        if f.if_based_on("R", state):
            return left
        else:
            return right
    if x < valley_threshold:
        if f.if_based_on("L", state):
            return left
        else:
            return right
```

(a)



(b)



(c)

Figure 4: **(a)**: Car decision-making function using **BasedOn**. `Valley` is the x value of valley threshold. **(b)**: MountainCar-V0 visualized state. **(c)**: average reward of the algorithm presented in part (a) compared to vanilla policy gradient, showing the usefulness of programmer intuition. Models were trained on 500 episodes each for 1000 steps. Mean and standard deviation are averaged across 20 runs.