Sparse Autoencoders for Interpretability in Reinforcement Learning Models

Coleman DuPlessie

Abstract

Recent work has shown that sparse autoencoders (SAEs) are able to effectively discover human-interpretable features in language models, at scales ranging from toy models to state-of-the-art large language models. This work explores whether the use of SAEs can be generalized to other varieties of machine learning, specifically, reinforcement learning, and what, if any, modifications are necessary to adapt SAEs to this substantially different task. This research investigates both qualitative and quantitative measures of SAEs' ability to represent reinforcement learning models' activations as interpretable features, using a toy reinforcement learning environment to conduct empirical experiments. It finds that SAEs are successfully able to break down deep Q networks' internal activations into human-interpretable features, and, furthermore, that some of these human-interpretable features represent an internal understanding of the underlying task that could not have been discovered from a deep Q network's output alone.

1 Introduction

Recently, there has been great progress in the field of mechanistic interpretability, which studies methods used to make trained machine learning models' decision-making processes understandable to humans. The vast majority of this work has centered around transformers, which provide an ideal testing ground for three reasons: first, their inputs and outputs (natural language) are very easily understood and manipulated by humans; second, natural language tends to be extremely conceptually sparse (i.e. for any concept, the vast majority of text is unrelated to that concept); and third, there are immediate practical uses for interpretability in transformers today (e.g. being used as a method to fine-tune the outputs of Large Language Models (LLMs) without requiring large amounts of human feedback[12]).

This paper focuses on the use of sparse autoencoders (SAEs). SAEs have been successfully applied to transformers to decompose their activations into human-interpretable features[4], which can then be manipulated to change the transformer's output in meaningful, interpretable ways[12].

In this paper, a variety of sparse autoencoders are trained on the activations of deep Q networks (DQNs). Many, though not all, features of these SAEs appear interpretable, and some features represent phenomena that do not improve model performence, but are learned regardless.

1.1 Sparse Autoencoders

Sparse autoencoders are a variety of autoencoders useful for taking features out of superposition. Superposition refers to the theory (proven to exist in toy models, and conjectured to hold in many large models) that "features," independent concepts represented by a machine learning model, are not actually represented independently, one in each neuron. Instead, features are each represented by a linear combination of neuron activations such that the set of features forms an overcomplete basis for the activation space of the model. This means that the model is able to represent more features than it has neurons, at the cost of reduced performance caused by independent features interfering

with each other. Because of the risk of feature interference, superposition is most common when features are very sparse and any given feature is inactive on (i.e. irrelevant to) the vast majority of input data[7].

If a model internally represents features in superposition, this implies that its neuron activations will not be interpretable by themselves, since each neuron is then representing a linear combination of disparate, unrelated features. This is the fundamental problem of mechanistic interpretability, and necessitates some method of taking features out of superposition.

This is achieved by training a sparse autoencoder, a network trained to make its output equal to its input, on the neuron activations of an underlying model. Non-sparse autoencoders are often used to learn efficient codings of arbitrary data, by making their hidden size smaller than their input and output size. SAEs, on the other hand, have hidden sizes substantially larger than their input size (at least 2 times as large, but more commonly anywhere from 4 to 256 times as large). SAEs are trained on some part of the internal state (generally, the activations of one particular layer) of a pre-existing model in order to interpret the model's decision-making process. In order to ensure that SAEs pull features out of superposition (thus rendering them interpretable), SAEs have a mechanism to encourage sparsity (thereby discouraging superposition, since features are sparse, but sets of many features in superposition are generally quite dense). There are two primary mechanisms used for this purpose: sparsity penalties and k-sparse autoencoders. This research uses k-sparse autoencoders.

k-sparse autoencoders allow a fixed number of neurons to fire on any given input. They do this by replacing a reLU unit or other nonlinear activation function with the TopK activation function, which allows the k highest-activating neurons to remain unchanged (similar to all positive neurons with a reLU activation function) while setting all other neuron activations to zero.

This approach has multiple benefits. It is extremely easy to tune the L0 norm (i.e. average feature sparsity) of k-sparse autoencoders, since their L0 norm is forced to be equal to k. Due to this enforced sparsity, the activations of an SAE's hidden layer are referred to as "features" (though some of these features my not be interpretable). k-sparse autoencoders also avoid problems like shrinkage associated with autoencoders that use sparsity penalties and therefore tend to have higher accuracy[8].

2 Related Work

Previous work in this area has primarily focused on the application of sparse autoencoders to large language models, built around the foundation established by Cunningham et al. [6], who showed that sparse autoencoders were fundamentally able to find features in language models. This approach was later refined by other work, such as Bricken et al. [4] and Templeton et al. [12], which introduced performance-enhancing techniques such as resampling and showed that sparse autoencoders scale well as underlying models grow larger.

Makhzani and Frey [8] introduced the k-sparse autoencoder as an alternative to L1 loss penalties, which were previously ubiquitous in regulating overall sparsity. Various other techniques have been proposed to improve model sparsity, such as Rajamanoharan et al. [9], which introduced the gated SAE, or Rajamanoharan et al. [10], which found benefits from using jump ReLU units in SAEs. These three techniques are mutually exclusive and all depend on the model's activation function: only the former was pursued in this paper.

The use of sparse autoencoders was expanded to image models by Surkov et al. [11], who showed that SAEs could reliably find interpretable features in text-to-image models. Researchers such as Abdulaal et al. [1] have been able to apply sparse autoencoders trained on vision-language models to achieve superior results to fine-tuning on practical tasks such as generating radiology reports.

There has been very little pervious research into the application of SAEs to reinforcement learning models. Annasamy and Sycara [2] developed architectures for DQNs designed to aid in interpretability, but, being written before Cunningham et al. [6], did not discuss sparse autoencoders. A preprint by Yin et al. [13] released in November 2024 showed that SAEs compare favorably with other reinforcement-learning-based alignment methods, such as Direct Preference Optimization (DPO) or Reinforcement Learning from Human Feedback (RLHF), when applied to large language models. However, it focused exclusively on the alignment of pretrained large language models.

Figure 1: A game of Ms. Pacman being played by a trained DQN

This paper is the first to apply sparse autoencoders to Deep Q Networks, a more traditional reinforcement learning model architecture.

3 Methods

In this paper, SAEs are trained to allow human interpretation of the features of a small DQN, which is itself trained on a simple reinforcement learning task. This paper uses the game Ms. Pacman for the Atari 2600 as a reinforcement learning environment.

3.1 Environment

This research uses the implementation of Ms. Pacman present in OpenAI Gym, a standardized API for reinforcement learning[5]. When a reinforcement learning agent is being trained, the model's reward is equal to its score, and its input is a live feed of what the player sees. The live feed is represented as a tensor of size 3x210x160.[1] The agent must output one of nine possible directions to push the game's virtual "joystick" (the 4 cardinal directions, 4 diagonals, and the center). Additionally, the model is only queried every fifth frame. This is a common feature of many video-game-based reinforcement learning environments: since the optimal move is likely to remain the same between two adjacent frames, it is minimally helpful to repeatedly query the model on near-identical questions. Instead, it is more efficient to train for more games with a lower frame rate.

Ms. Pacman (see Figure 1) is a useful environment for interpretability research because it is fully deterministic (enemies' movement is pseudorandom based on how the player moves) and relatively simple while also very sparse (i.e. there are many features a model might want to track, but few of them are relevant at any given moment). This is important because SAEs are only useful due to superposition in the base model, and superposition tends to be stronger when features are sparser[7]. Therefore, it is hypothesized that SAEs will be highly effective on models trained in this environment because of its high sparsity. Another benefit of this environment is that it is highly interpretable to humans, and models' decisions (and mistakes) are easily understood, unlike other, more obscure or detailed reinforcement learning tasks.

[1] Due to the physical design of the original Atari 2600, the environment's vertical resolution is greater than its horizontal resolution, meaning that pixels are not square. Depictions of the model in this paper use square pixels, causing the model's input to appear taller than it is wide, despite the original game (and depictions of gameplay in this paper) having a 4:3 aspect ratio.

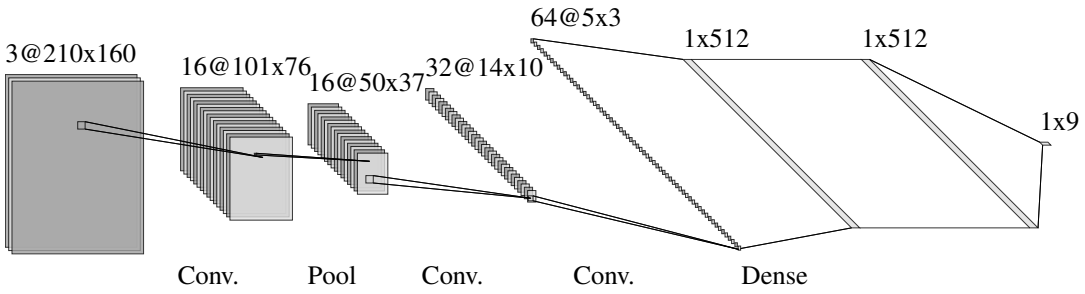

Figure 2: The architecture of the DQN used in this paper.

3.2 Training

Training consisted of two phases: first, training a DQN to perform well in the environment at hand (Ms. Pacman), and next, training a sparse autoencoder to reconstruct the DQN's final layer's activations. In both cases, several different model architectures and hyperparameters were tested to maximize accuracy and, in the case of the SAE, interpretability. The most effective are discussed here.

3.2.1 Deep Q Network

Experimentation began by training a base model, a deep Q network, for 20,000 games. The DQN's architecture consisted of 3 progressively smaller convolutional layers (with one pooling layer) followed by 3 fully-connected layers (See Figure 2). All layers used a reLU activation function.

The DQN is relatively small, with less than 880,000 parameters. This causes it to attain modest success with the task at hand (a much larger model would perform better), but it is large enough to contain interpretable features in superposition, while being small enough to only contain a moderate amount of such features, making it an ideal size for research applications. Prior work applying SAEs to large language models found that SAEs tend to become more interpretable, not less, as the base model grows larger[12]. This suggests that SAEs are likely to also scale well when applied to reinforcement learning models.

Notably, the DQN used in this paper has no recurrent layers or ability to act based on the past, meaning that it has no sense of time and makes decisions entirely based on the current frame. This is not an issue for model performance, because the environment (Ms. Pacman) displays all information that might be necessary to make the optimal decision each frame.

3.2.1.1 Encouraging Randomness

One of the benefits of using Ms. Pacman as an environment is that the game is pseudorandom: enemies' movements begin consistently but later depend unpredictably on the player's movement (in OpenAI Gym's implementation, which this research uses, player input is the only such source of entropy for the pseudo-random number generator). However, this threatens the DQN's ability to generalize: it is well within the abilities of even small reinforcement learning models to memorize the string of actions that best manipulates enemies' movement, and memorizing the enemies' actions replaces the process of making decisions based on the game state, meaning that the model's behavior no longer generalizes to other applications of reinforcement learning (since memorizing the one best sequence of actions is a degenerate case of reinforcement learning optimization)[3].

To counteract this, randomness is added to the model's actions during the training process. Two methods are used to accomplish this, summarized in Table 1.

First, during training, there is some small chance ϵ that any given move is selected randomly instead of being chosen by the DQN. This ensures that the model eventually deviates from the solution it has mapped out up to this point. The value of ϵ can be tuned to strike a balance between the exploration of alternative strategies (when ϵ is large) and the exploitation and refining of the model's current strategy (when ϵ is small). In this research, ϵ begins at a value of 20% and decreases linearly

Source of randomness	DQN train time	SAE train time	Test time
Random action	15%-3.75%	0%	0%
Sticky action	25%	25%	25%

Table 1: Total frequency of random interventions in the DQN's behavior at train and test time.

throughout the first half of the training process to 5%, where it then remains for the second half of training.

This research also uses stochastic action stickiness: in addition to the varying chance that a random action is selected, there is also a constant, 25% chance that the model's previous action is selected.[2] Since the model is only queried every 5 frames to begin with, this means that there is a significant chance the environment goes 15, 20, or more frames without allowing the model to react to changes in the game state. This has two benefits: first, it encourages the model to "look ahead" at the decisions it is likely to make in a few frames and make them now if possible. If the player would be indifferent between two actions (e.g. pushing the joystick left or right while the player is in a vertical corridor), it is beneficial for the model to choose based on which action would be better if it is "sticky" for several frames (e.g. at the next intersection, would it be better to turn left or right?). Second, stochastic action stickiness provides another source of randomness: the model, through no fault of its own, has a chance of missing actions that it would have preferred to take. This ameliorates the problems and bad incentives caused by the environment's deterministic nature.

3.2.2 Sparse Autoencoder

After the DQN has been trained, a sparse autoencoder is trained on the activations of the DQN's final layer. The fully trained DQN plays 20,000 additional games, and the SAE is trained on the DQN's activations during these games.

During these additional games, there is still a 25% chance that the DQN's actions are sticky, but ϵ, the chance of taking a completely random action, is reduced to zero. This is because the DQN's parameters are no longer being updated, and therefore there is no benefit gained from encouraging the DQN to explore all possible paths. Instead, the randomness produced by stochastic sticky actions is preferred, since it produces a distribution of games that are similar, but subtly different from each other. This allows the SAE to train on a diverse training set while ensuring that the SAE's training set closely resembles the DQN's actual gameplay decisions.

The SAE's architecture itself is extremely simple: its input and output are each 512 neurons wide, and they are each connected to one hidden layer that is 2048 neurons wide. The hidden layer itself uses a top-k activation function with $k = 50$. The hidden layer is 4 times larger than the input and output layers: this is a moderately small SAE, but still well within the range of sizes shown to be effective on other tasks (e.g. [4]).

SAEs also use a pre-encoder bias, in which the SAE's decoder layer's bias terms are subtracted from its input. This is motivated by previous work, such as Towards Monosemanticity [4], which finds that pre-encoder biases improve model performance without harming interpretability.

4 Results

4.1 Deep Q Network

A DQN and corresponding SAE were trained as described in Methods. The small DQN (see Figure 2), once fully trained, is able to score an average of 2227 in-game points. When stochastic sticky actions are removed (i.e. there is no randomness and the behavior of the model and environment is deterministic), it scores exactly 2130 points.

[2]This 25% chance of a repeated action is checked before the varying chance of a random action, meaning that the "true" chance of a random action ranges from 15% to 3.75%, not 20% to 5%.

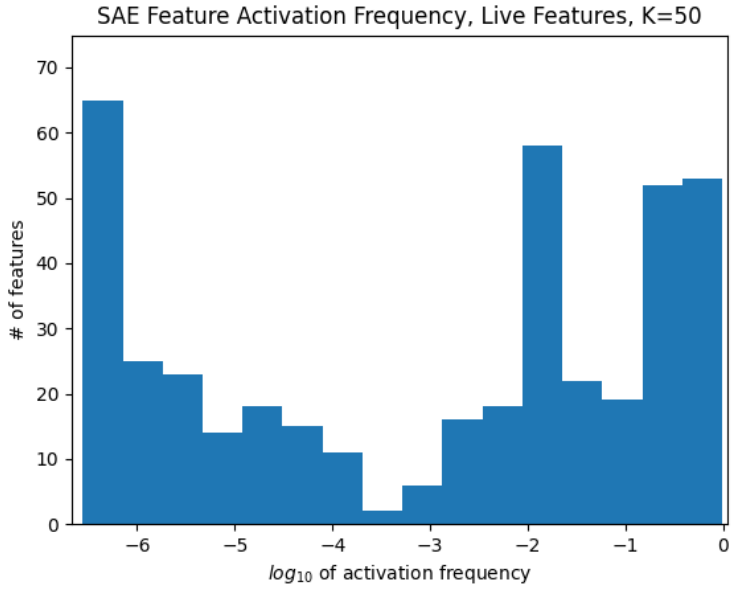

Figure 3: Distribution of activation frequencies of live features. Note that the X axis is a log scale.

4.1.1 Interpretability

Upon inspection, trained DQNs sometimes exhibit human-like strategies (e.g. seeking power-ups when in danger). However, without a SAE, there is no mechanism to draw out these strategies into concrete features that are connected to the model's weights, since features in the DQN are stored in superposition[7]. Such a mechanism would make it possible to arbitrarily modify the DQN's behavior in human-interpretable ways (e.g. no longer seek power-ups when in danger or always seek power-ups) without large side-effects on overall model quality.

Due to the small size of the DQN used in this paper, it does often make significant unforced errors such as failing to evade easily-avoidable enemies. Notably, these errors are marginally more common when the environment is in a state not similar to the training distribution (i.e. a state that is very uncommon for the model to arrive at by itself). This is expected, since the model has been trained to perform optimally with only 20,000 games, meaning that it has no experience acting in games that do not resemble these 20,000. Despite this, the model is often, but not always, able to perform well in radically out-of-distribution situations (e.g. if the model is not queried at all for the first 10 seconds of gameplay).

4.2 Sparse Autoencoder Sparsity

In trained sparse autoencoders, sparsity does not necessarily guarantee interpretability, but sparse features are reliably more likely to be interpretable than their dense counterparts. The sparsity of individual features is measured by determining how frequently they activate on a sample of 1,000 games.

Figure 3 shows the sparsity distribution of live features in the SAE trained in this paper. Features that are extremely dense (activating on more than 10% of all inputs) or extremely sparse (activating on less than 0.1% of all inputs) are unlikely to be human-interpretable. State-of-the-art SAEs have many of these features, often referred to as the ultralow density cluster. The SAE trained here, although it does have a number of highly dense features and ultralow density features, also contains a large number of features of the proper sparsity, suggesting that these features are likely to be interpretable.

In addition to the moderate number of highly dense and highly sparse features, SAEs tend to have a number of dead features, which never activate. Measuring the number of dead features in a SAE with certainty is not possible without enumerating every possible input state, which is computationally

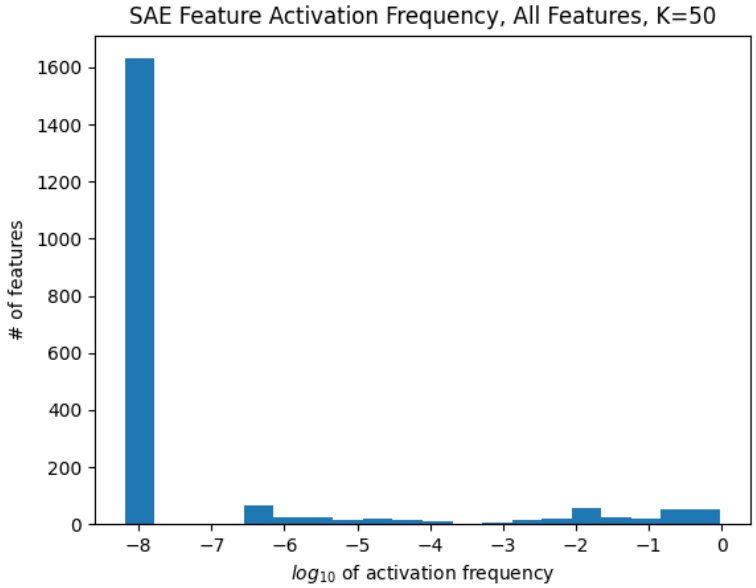

Figure 4: Distribution of activation frequencies of all features. This figure is a scaled version of Figure 3, with the large bar at the left representing dead neurons.

infeasible. However, testing features on a large, but finite, set of input states is reasonably accurate (although it is likely to systematically overestimate the number of dead neurons in a SAE by a moderate amount).

As can be seen in Figure 4, approximately 80% of the trained SAE's 2048 features appear dead. The number of true dead features that never activate (as opposed to features that appear dead because they only activate on states not reached in the 1,000 games of testing) is likely somewhat lower than this, but still significant. While this quantity of dead features is abnormally high, this does not necessarily invalidate the SAE's usefulness: the primary problem with a large number of dead features is that, since the average feature activation frequency is fixed (equal to $\frac{k}{2048}$), a high number of dead features is likely to correspond to an increase in highly dense features, which are generally not interpretable. However, the large number of dead features does not have an impact on the interpretability of the features that remain neither too sparse nor too dense.

4.3 Sparse Autoencoder Interpretability

To determine the interpretability of individual features, the activations of several features were observed manually. This is in line with the prior state-of-the-art with SAEs, which uses human ratings to gauge the interpretability of features (e.g. [12]). The interpretability of features depends both on the input states that cause them to activate (if most states are similar and share a human-interpretable concept, the feature is likely interpretable) and the effect their activations have on the model's chosen action (if the effect is congruent with the input states, the feature is considered interpretable). Since this research trains a SAE on the final layer of a DQN, after all nonlinearities, it is trivial to calculate the effect of a feature activation on the final output of the DQN, because the effect is a constant that scales only with the magnitude of the feature activation (this is not the case when features are calculated from earlier layers in the model).

Two representative interpretable features are discussed below.

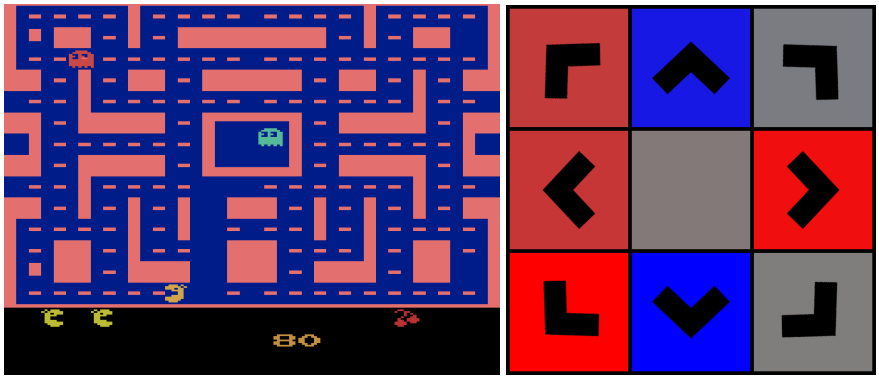

Figure 5: Left: A representative frame that causes feature 55 to activate strongly. Right: The effect feature #55 has on the DQN's action selection, where blue represents encouraging the corresponding action.

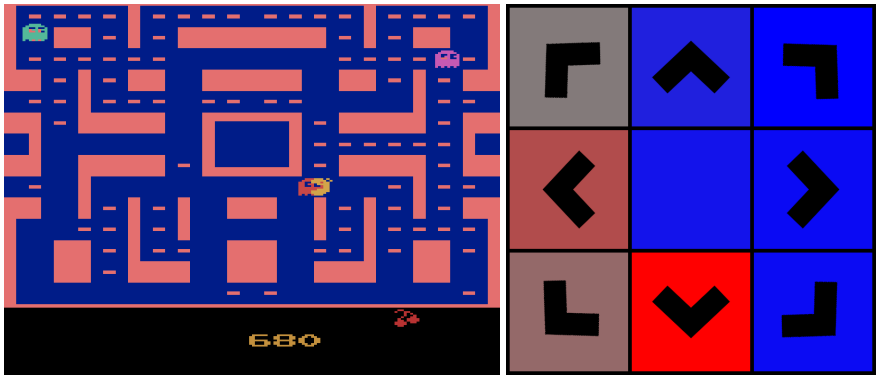

Figure 6: Left: A representative frame that causes feature 1410 to activate strongly. Right: The effect feature #1410 has on the DQN's action selection, where blue represents encouraging the corresponding action.

4.3.1 Feature 55

Feature 55 activates on 1.05% of all input frames. It consistently activates when the player is traveling leftward across the bottom of the screen (see Figure 5). When activated, its effect is to encourage the model to move up or to continue moving left[3] and to discourage the model from turning around.

This is an excellent example of an interpretable feature. It clearly serves one purpose[4], and that purpose seems to align with the model's overall goal of scoring the most points: turning back the way the player came is almost never useful, while continuing or turning up both might be useful, depending on the activation of other features (this feature, considered alone, weakly prefers moving the joystick down, which, since the player is already at the bottom of the screen, has the effect of continuing in a straight line).

4.3.2 Feature 1410

Feature 1410 activates on 2.39% of all input frames. Like feature 55, its activation has a clear effect on the model's decision-making (pushing the model up and to the right, as seen in Figure 6). However, feature 1410 only activates at a very specific point in time: when the player is losing its last life, regardless of where the player is when this happens. At this point, the environment will play a death animation for several frames and then reset for the next game. The game completely

[3]When the model moves the joystick in an illegal direction (e.g. moving down when already at the bottom of the screen), it has the effect of continuing to move in the current direction.

[4]If it served many different purposes (i.e. represented several concepts in superposition), the set of input states that activate it would be a multimodal distribution, which is not the case.

disregards the direction the model moves, and it is too late for the model to possibly increase its reward any more.

This is an interesting feature because it shows that the original DQN is able to distill the concept of "losing the player's last life," even though learning that concept carries no benefit for the model during training. Nevertheless, despite the fact that the model's response to this feature never affects the environment's state (since the model's action selection is ignored during the final frames after it loses its last life), the SAE is able to distill the feature out of the trained DQN, meaning that the original feature must be represented within the original DQN in superposition.

5 Conclusion

In this work, a sparse autoencoder was trained to find human-interpretable features in a deep Q network that was itself trained to play the game Ms. Pacman. While some of the sparse autoencoder's features were dead, highly dense, or otherwise not interpretable, many features were interpretable, shedding light on the inner decision-making processes of the DQN. Some of these features carry seemingly no benefit to the model's performance, but are learned and distilled into features anyway, suggesting that the sparse autoencoder is accurately capturing the DQN's inner model of the environment.

Sparse autoencoders are a promising tool for interpreting the decisions of trained machine learning models. They can be used to easily steer models' activations, allowing precise, directed control over model output. This research proves that sparse autoencoders can provide insight into the interpretability of traditional reinforcement learning models.

6 Acknowledgements

I would like to thank my mentor, Andrew Gritsevskiy, for his guidance and support, as well as MIT PRIMES for making this research project possible.

This research used the Delta advanced computing and data resource which is supported by the National Science Foundation (award OAC 2005572) and the State of Illinois. Delta is a joint effort of the University of Illinois Urbana-Champaign and its National Center for Supercomputing Applications.

References

[1] Ahmed Abdulaal, Hugo Fry, Nina Montaña-Brown, Ayodeji Ijishakin, Jack Gao, Stephanie Hyland, Daniel C. Alexander, and Daniel C. Castro. An x-ray is worth 15 features: Sparse autoencoders for interpretable radiology report generation, 2024. URL https://arxiv.org/abs/2410.03334.

[2] Raghuram Mandyam Annasamy and Katia P. Sycara. Towards better interpretability in deep q-networks. In *Proceedings of 33rd National Conference on Artificial Intelligence (AAAI '19)*, pages 4561 – 4569, January 2019.

[3] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012. URL http://arxiv.org/abs/1207.4708.

[4] Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nicholas L Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Chris Olah. Towards monosemanticity: Decomposing language models with dictionary learning, 2023. URL https://transformer-circuits.pub/2023/monosemantic-features.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[6] Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models, 2023. URL https://arxiv.org/abs/2309.08600.

[7] Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition, 2022. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/.

[8] Alireza Makhzani and Brendan Frey. k-sparse autoencoders, 2014. URL https://arxiv.org/abs/1312.5663.

[9] Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Tom Lieberum, Vikrant Varma, János Kramár, Rohin Shah, and Neel Nanda. Improving dictionary learning with gated sparse autoencoders, 2024. URL https://arxiv.org/abs/2404.16014.

[10] Senthooran Rajamanoharan, Tom Lieberum, Nicolas Sonnerat, Arthur Conmy, Vikrant Varma, János Kramár, and Neel Nanda. Jumping ahead: Improving reconstruction fidelity with jumprelu sparse autoencoders, 2024. URL https://arxiv.org/abs/2407.14435.

[11] Viacheslav Surkov, Chris Wendler, Mikhail Terekhov, Justin Deschenaux, Robert West, and Caglar Gulcehre. Unpacking sdxl turbo: Interpreting text-to-image models with sparse autoencoders, 2024. URL https://arxiv.org/abs/2410.22366.

[12] Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, Alex Tamkin, Esin Durmus, Tristan Hume, Francesco Mosconi, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/.

[13] Qingyu Yin, Chak Tou Leong, Hongbo Zhang, Minjun Zhu, Hanqi Yan, Qiang Zhang, Yulan He, Wenjie Li, Jun Wang, Yue Zhang, and Linyi Yang. Direct preference optimization using sparse feature-level constraints, 2024. URL https://arxiv.org/abs/2411.07618.