Zoom In on Agents: Interpreting Deep Reinforcement Learning Models

Mauricio Baker
Stanford University
maubaker@stanford.edu

Abstract

Despite the successes of deep neural networks for solving reinforcement learning environments, these models' transparency remains highly limited, making it harder to ensure their reliability. In recent years, a number of new approaches to interpreting neural networks have been proposed, yet applications have focused on other machine learning applications. This paper builds on earlier techniques by creating tools for saliency mapping, data sampling, weights visualization, and input optimization, and by applying these tools to interpret a deep Q-learning network pre-trained for playing Atari Breakthrough. Focusing on the first and fourth layers, I find that layer-1 neurons tend to be interpretable as polysemantic representations of one of several distinct objects in the game Breakthrough, identifying particular objects' locations, their direction of motion, and—in some cases—their destruction. Additionally, I find that layer-4 neurons are often used more narrowly, for identifying very specific high-level game states, to be directly used for action selection.

1 Introduction

Many of the most successful reinforcement learning agents—in particular, Deep Q-Learning Networks—have also been among the least transparent. This is a problem for efforts to apply reinforcement learning in safety-critical settings; opaqueness makes it harder for researchers to verify that machine learning models are robust, fair, and otherwise reliable.

A prominent and successful (Mnih et al., 2013) approach to solving reinforcement learning problems is Deep-Q Learning Networks. Like standard (value-based) Q-Functions, DQNs learn to map from environment states to (expected, discounted) cumulative reward, and this function is a key input into the agent's policy. Unlike Q-Functions, which learn these functions as a table, DQNs learn this function through a deep neural network. This has substantial advantages: it makes the number of parameters in the model more manageable, and it allows for learning about some states to transfer into learning about other state-value mappings (Simonini, 2020). However, with these advantages comes a significant downside: the opaqueness of deep neural networks.

Using pixel data from an Atari Breakthrough program, as well as data from the weights and activations of a pre-trained model, this paper combines several interpretability techniques to identify the functions of individual neurons in a successful DQN.

2 Related work

A handful of approaches to interpreting neural networks have been proposed over the last decade, and many are applicable to DQNs.
One prominent approach is creating saliency maps (Simonyan et al., 2013): in this approach, the partial derivative of classification outputs with respect to input components is used to identify inputs that most contribute to the classification. A related approach is occlusion sensitivity (Zeiler and Fergus, 2014), in which one analyzes how a model’s classification confidence changes in response to large, rectangular regions of the input being grayed out. This second approach succeeds in analyzing higher-level functions than saliency maps, but the two approaches have a similar limitation: they do not tell us how inputs are being used to generate outputs, which can be useful information for assessing reliability.

Another branch of work aims to interpret deep Q-learning networks by approximating them with more transparent functions. For example, Liu et al. (2018) train a more simple model (a linear model U-tree) to mimic a DQN, and then they analyze the simpler model. This approach gets closer to understanding neural network functionality, yet it does so with the accuracy loss that comes from approximation—it may leave us without the confidence we want that interpretations of the simpler model map on to the neural networks’ function.

Additionally, some researchers have aimed to advance RL interpretability by forcing models to take more interpretable forms, such as having attention mechanisms (Mott et al., 2019) or hierarchical policies (Shu et al., 2017). These approaches seem promising to some degree, but they leave large parts of models unexplored.

Researchers who are more pessimistic about the interpretability of deep reinforcement learning agents have sought to find alternatives to DRL (e.g. verma et al., 2018), yet the success of DQL may limit the prospects of these research directions.

A further branch of work in interpreting deep neural networks has consisted of efforts to understand the functions of individual neurons and layers, through techniques including gradient ascent (Simonyan et al., 2014), dataset search, and analysis of circuits connecting interpreted features (Olah et al., 2020), as well as weights visualization (Voss et al., 2021). Although these approaches have been challenged by e.g. the difficulty of interpreting certain neurons (Olah et al., 2020), this approach is promising as the one which most directly asks the question, “What are deep neural networks doing?”

### 3 Dataset and Features

As an investigation of a deep reinforcement learning network, this work used as data the state data that an Atari Breakout program fed into a pre-trained DQN. State data took the form of pixel input, pre-processed consistently with the approach taken in Mnih et al. (2013): pixel input was converted to grayscale and downsized to 84x84 pixels. Then, the most recent 4 such pixel screens (“frames”) were all given as input into the model. The pre-trained DQN and the Breakthrough program used were both downloaded from the following public Github repository: https://github.com/SinanGncgl/Deep-Q-Network-AtariBreakoutGame.

Additionally, as an interpretability project, this work used data from the pre-trained DQN itself. I analyzed the activations of individual neurons in hidden layers, as well as in the output layer, in order to identify data samples and synthetic inputs that would highly activate particular neurons and filters. I also used that data to create saliency maps. Lastly, I directly examined the pre-trained weights of the model for visualizing them.

### 4 Methods

I implemented and applied four tools.

The most straightforward was weights visualization: directly mapping the weights that feed into a filter or neuron gives us direct insight into how it works. I created an algorithm that takes a layer and neuron number as an input and displays a normalized, color-coded map of its weights.
An additional technique used was dataset sampling. This involved running the pre-trained model on the game and saving inputs that most activated each neuron in the model. I created a database of the 10 input states that most activate each neuron in the model, as well as their corresponding activations and—in the case of convolutional layers—the corners of the corresponding region. Then, I created a function that takes as input a layer and neuron number and displays the 10 data samples that most activated that neuron, in sorted order, alongside their activation number and with the convolved region outlined.

Next, I created an input optimization tool. This algorithm starts with either a randomly generated image or a 4x84x84 dataset sample. Then, it iteratively modifies the image, optimizing the output of a particular neuron in the pre-trained model. The image is an nxnx4 tensor, where n is selected to be of the size of the input that the neuron examines (which is smaller than the whole image for convolutional layers). In addition, the optimization includes regularization to smooth the output, avoiding extraneous "high-frequency patterns" (see Olah et al., 2017).

Additionally, I created a saliency map tool that displays a saliency map for an arbitrary neuron. This uses gradients calculated through a special case of the input optimization tool.
5 Experiments/Results/Discussion

Experimentation resulted in numerous lessons about how to better implement the above techniques.

For weight visualization, I concluded that colors should be normalized with respect to the range of weights (otherwise a collection of small weights won’t be visible). In addition, colors on opposite ends of the color spectrum should correspond to same-magnitude (opposite-sign) weights, to avoid confusing interpretations. In practice, this means making the center of the color spectrum correspond to a weight of 0. Additionally, I found that colors should have similar brightness at full saturation, so that similarly intense colors are perceived as such. Lastly, I decided that higher magnitudes should correspond to higher intensity, so the center of the color scheme should be black—not a blend of the two extreme colors.

For data sampling, intermediary results suggested I should examine many data samples for each neuron, to avoid overly hasty conclusions. In addition, displaying data samples alongside corresponding activations and sorting samples by activations helped make the display easier to visually interpret, as did drawing boundaries between frames of the same sample. Outlining the region to which the convolution was applied, in the context of the larger image, was also found to be useful for making the data visually intuitive.

For saliency maps, many of the adjustments made to the weight visualization color scheme were also applicable.

For input optimization, it was necessary to modify the ReLu activation functions in the forward propagation of the pre-trained model to leaky ReLu functions, in order to avoid "dead neurons" or zero gradients. A negative slope of 0.001 was useful—higher ones excessively degraded the model’s performance.

In addition, applying regularization in input optimization helped highlight what pixels are critical to the optimization, while blurring those that are less important. This was useful for making a display of the optimized input easier to visually interpret. I took inspiration from Mahendran and Vedaldi (2015). For regularization of discrete inputs, they added the following term to the loss: the sum of the squares of the distance between each pixel and two adjacent pixels. I extended this method to the 3-dimensional input, and I created a tensor-based implementation of it for computational efficiency. This created a problem: outside of a tiny range of values for scaling the regularization term, the input produces would either be drastically over-regularized or under-regularized. I solved this by forcing the regularization term to be zero until late in the optimization process. (This may have worked because it prevented the optimizer from getting stuck early on in a local minimum consisting of an extremely blurred image.)

Experimentally, I found that it works well to have 13,000 training iterations for the input optimization, starting the regularization at 10,000, and scaling the regularization term by beta=0.5 divided by the total number of input pixels. Beta can be doubled or halved to produce sharper or more blurry images that still balance accuracy and regularization well. Having discussed some considerations from experimentation, I turn to results, which focus on the 1st and the 4th layers of the model. The 1st layer is a convolutional layer that applies 32 filters to pixel input. The 4th layer is a fully connected layer that takes as input the activations of another fully connected layer, and it feeds directly into the output layer.

The combination of data sampling and weights visualization showed that almost all neurons in the first layer identify particular objects, in particular frames. Filters tended to “pay much more attention” (have much higher-magnitude weights in response) to the 4th input channel, which corresponds to the
most recent frame of the game. This makes sense—the current game state is likely most relevant for what one should do next. As examples, the 2nd neuron in this layer primarily activates in response to the left side of the Breakthrough paddle, while the 5th neuron primarily activates in response to the Breakthrough ball being in its most recent frame.

While the object that each neuron was identifying was clear, it was more challenging to interpret more precise functions. Most 1st-layer neurons exhibited substantial degrees of polysemanticity. As described in Olah et al. (2020), this means that the neurons encode multiple meanings. For example, many of the ball-detecting neurons had small, round clusters of high-magnitude weights corresponding to multiple parts of the input image, suggesting that they detect the ball in multiple possible positions. This may be because the 32 filters of the 1st layer are not enough to encode all game states that it is useful to encode, if each neuron is only used to encode one game state.

Next, we turn to considering other layers. A clearly important layer in the model is its output layer. While this layer takes 512 neurons as inputs, weights visualization suggested a surprising result, confirmed by quantitative analysis: a small fraction of input neurons have very outsized effects on the 5th layer’s outputs. More concretely, for the two output neurons, 10% of their weights connecting to the previous layer make up 52.9% and 52.6% of the total (summed, absolute) value of their weights. This suggests the task of interpreting the output neurons is much more feasible than it otherwise may have been: a small fraction of layer 4 neurons may tell us a lot about the output neurons’ behavior. With this motivation, we turn to layer 4.

Data sampling and input optimization (using input optimization with data samples as their starting images) suggest that layer-4 neurons tend to work very differently from those of layer 1. While layer-1 neurons tended to identify small features that were present in a wide range of game states (such as the Breakthrough ball moving in a certain direction), layer-4 neurons often encoded very specific information about the game state as a whole. For instance, neuron 154 in the 4th layer had activations from around 30-70 within a few specific states of the game (which in practice come continuously), and much lower activations (under 3) for every other game state. Neuron 382 overwhelmingly responded to just a single game state, out of the whole game. So did neurons 388 and 399, as additional examples; they respond strongly to one state of the game, and much more weakly to the very similar states immediately preceding and following that state.

This suggests layer 4 encodes very precise, high-level information about the big-picture game state. This is plausible because it has many neurons—making such specificity possible—and it comes after three convolutional layers—making it feasible to identify high-level game states.

6 Conclusion/Future Work

This paper implemented four ML interpretability methods and applied them to a deep Q-learning network. In combination, the four methods—weights visualization, data sampling, input optimization, and saliency mapping—allowed for significant progress toward interpreting the model. Almost all first-layer neurons were robustly interpreted as identifying particular game objects, and often as encoding further details about the state of these objects. In addition, weights visualization revealed that the output layer of the model is primarily sensitive to a small fraction of neurons in the penultimate layer. When applied to these neurons, data sampling and input optimization tended to reveal that these influential neurons were encoding particular high-level game states.

Researchers could build on this work by refining the techniques used. For example, input optimization could be combined with an analysis of clusters in dataset search, to create a range of optimized inputs with different starting points. This may better represent the wide range of meanings encoded by some neurons. Additionally, the methods built could be applied to other DQNs to see what adjustments are necessary, and which findings generalize. For example, do models trained on more diverse sets of games form more human-interpretable neurons?

The significance of the above results is not in Atari, but in supporting a growing foundation of methods and ideas for interpreting DRL networks. While real-life applications of RL remain limited, the rapid rate of progress suggests further applications may be on the horizon. With adequate interpretability techniques prepared in advance, we may be better able to deal with the impacts of these applications as soon as they come, shortening a painful adjustment period.
7 Contributions

This was an individual project by Mauricio Baker.

References


