Visualizing Neural Network Imagination

May 30, 2024

Abstract

In certain situations, neural networks will represent environment states in their hidden activations. Our goal is to visualize what environment states the networks are representing. We experiment with a recurrent neural network (RNN) architecture with a decoder network at the end. After training, we apply the decoder to the intermediate representations of the network to visualize what they represent. We define a quantitative interpretability metric and use it to demonstrate that hidden states can be highly interpretable on a simple task. We also develop autoencoder and adversarial techniques and show that benefit interpretability.

1. Introduction

When trained to predict the end state of a given sequence, neural networks (NNs) may learn to predict intermediate states even if not explicitly trained to do so. For example, a network trained to predict the end state of an environment where the easiest way is to predict it one step at a time will most likely generate hidden representations of the intermediate steps. Consider a NN trained to predict the 4th state of Conway's Game of Life (GoL) (Conway, 1970) given an initial state. GoL is a cellular automata with simple deterministic rules to obtain the next state from the previous. The network could learn the GoL rules and apply them to the 1st state to obtain a representation of the 2nd state, to the 2nd state to obtain the 3rd state and so on. The states before the 4th state would be represented only internally in the network, since it is only trained to output the 4th state. Another case where the network might represent environment states is when making a decision. For example, if a reinforcement learning (RL) agent needs to decide whether to go left or right in a maze, the NN might predict what the state of the environment will be after going left. The network may represent this state in its hidden activations and use it to decide how to turn. Our goal is to visualize the environment states that a network is representing. In this case, it is like the network is imagining the consequence of its action. When a network is making a decision, visualizing the states it is considering can help evaluate if the network is making a safe and correct decision and for the right reasons. To our knowledge, this interpretability problem has not been explored before.

An important motivation for visualizing the intermediate states represented by a neural network is to better understand mesa optimization (Hubinger et al., 2019). Mesa optimization occurs when a neural network is trained to perform some objective but ends up implementing an internal search process to achieve that objective. The network essentially learns to perform optimization ”inside the model” - hence the term ”mesa” optimization. If the learned internal objective of this optimization does not align with the intended training objective, it can lead to unintended and potentially dangerous behavior. For example, a reward-hacking mesa-optimizer may find an undesirable way to maximize its reward that goes against the intended goal. By visualizing the intermediate states considered by the network during its decision-making process, we may gain insight into whether mesa-optimization is occurring, what internal objective is being pursued, and whether it aligns with the intended objective. This could be an important tool for creating safer and more robust AI systems.

In this paper, we first focus on visualizing the network's intermediate representations when predicting GoL states using a recurrent neural network (RNN). We then apply the technique to an LSTM trained to predict the number of living cells in GoL after a certain number of timesteps. Finally, we experiment with an AlphaZero chess model to see if we can uncover the states the model considers when planning the next move. We use an encoder-decoder architecture, where we train the encoder and decoder as an autoencoder on the input states, and apply the decoder to the intermediate hidden states to visualize what they represent.

The GoL provides a good testbed because we know the ground truth of the intermediate GoL states so we can accurately evaluate our technique. Chess, on the other hand, represents a more challenging case where the model's internal representations may be more abstract and harder to interpret.

1.1. Contributions

Our main contributions in this work are:

1. We introduce the task of visualizing the intermediate environment states represented by a neural network and motivate its importance for understanding mesa optimization and improving AI safety.

2. We develop an autoencoder and adversarial training method for visualizing intermediate representations in an encoder-decoder architecture.

3. We present results on applying this technique to RNNs and LSTMs trained on the Game of Life cellular automata, showing that intermediate states can be recovered in these simple environments.

4. We experiment with an AlphaZero chess model and discuss the challenges in uncovering abstract planning representations in more complex domains.

2. Methodology

We conduct experiments on two main environments: the Game of Life and Alpha Chess. The Game of Life serves as a simple and controlled environment where the ground truth dynamics are known, allowing us to validate our visualization technique and assess the impact of different architectural choices and training procedures. Alpha Chess, on the other hand, represents a more complex and realistic environment where the network's internal planning process may be more sophisticated and challenging to interpret. By applying our visualization technique to both environments, we aim to demonstrate its generalizability and explore its limitations in uncovering planning in more advanced networks.

2.1. Architecture Overview

The architecture used in our experiments consists of three main components: an encoder, a recurrent neural network, and a decoder. The encoder takes the initial state of the environment as input and learns to compress it into a hidden representation. This hidden representation is then processed by a series of recurrent layers, which aim to capture the temporal dynamics and potential planning steps of the network. Finally, the decoder takes the hidden representations from the recurrent layers and attempts to reconstruct the corresponding environment states.

We experiment with different numbers of recurrent layers and timesteps to investigate the impact of network depth and temporal resolution on the interpretability of the intermediate representations. The encoder, recurrent layers, and decoder are all implemented using convolutional neural networks (CNNs) to efficiently process grid-like state representations.

2.2. Training Procedure

The network is trained using pairs of initial and final environment states. The loss function used for training is a combination of the L2 loss for predicting the final state, the reconstruction loss for the autoencoder, and optionally, the adversarial loss from the discriminator.

The primary objective is to minimize the L2 loss between the predicted final state and the ground truth final state. This encourages the network to learn to accurately predict the outcome of the environment's dynamics. Additionally, we employ autoencoder training by minimizing the reconstruction loss between the input state and the decoder's output when applied to the encoder's hidden representation. This helps to ensure that the encoder learns a meaningful and invertible representation of the environment state. In some experiments, we also use adversarial training to encourage the decoder to generate realistic and valid environment states, even when applied to the intermediate hidden representations of the recurrent layers.

2.3. Network Visualization and Evaluation Metrics

To visualize the intermediate states of the network, we apply the decoder network to the intermediate hidden states of the recurrent layers during inference. By doing so, we aim to visualize the states that the network may be representing or considering during its potential planning process. This approach is motivated by the hypothesis that the network may use a similar representation format for the states it considers during planning as it does for the encoder's output. By visualizing these intermediate representations, we can gain insight into the network's internal decision-making process and assess whether it is engaging in some form of planning or state search.

For GoL, we define a metric to determine how closely the network represents the intermediate GoL states. We consider a GoL sequence with m states $\{s_1 \dots s_m\}$ and feed s_1 into the inference network to generate $\{h_1 \dots h_n\}$ outputs from the n hidden states of the network. We threshold each pixel of network outputs at .5 so they become either 0 or 1, since GoL states have binary pixel values. Finally, we count the number of h_i outputs that match any s_j GoL state and divide it by the minimum between n and $m-2$ (not m since we exclude s_1 and s_m) as shown in this equation:

$$\frac{\sum_{i=1}^{n}\sum_{j=2}^{m-1} f(h_i, s_j)}{\min(n, m-2)} \tag{1}$$

f is a function which determines if two states match. If less than 95% of the pixels in the two inputs of $f(h_i, s_j)$ are equal f gives 0. Otherwise, f gives .5 if another generated state already matched the GoL state, and 1 if it did not. This gives credit for states in any order matching, because when the number of model and GoL timesteps are different it is not clear which states should match. Note that the calculated metric could be greater than 1 if $n > m - 2$. Each metric value that we report is the average over ten thousand GoL sequences for each run. See appendix A for examples of metric values of state sequences.

3. Results

3.1. Game of Life (RNN)

We first apply our visualization technique to a recurrent neural network (RNN) trained on GoL.

3.1.1. Results with same number of model and GoL timesteps

First, we evaluate our technique with the same number of GoL and model timesteps. For each run, we try our technique with and without adversarial training, and report the maximum of two the metric values. We only include trainings in this analysis where the network reached 99% training accuracy in predicting the final GoL state. We measure the mean metric value and impact of the following hyperparameters in table 1:

- The number of timesteps in GoL and model (*Num timesteps*)
- Whether the network was trained with an autoencoder loss (*Use Autoencoder*)
- Whether corresponding layers of different timesteps share the weights (*Use RNN*)

Num timesteps	Use Autoencoder	Use RNN	Number of Runs	Mean Metric Value
2	True	True	6	1.00
3	True	True	42	0.99
4	True	True	33	0.93
3	False	True	22	0.56
3	True	False	20	0.40

Table 1: The first 3 rows show that interpretability gets slightly worse with more timesteps. The last two rows show that the autoencoder and RNN are helpful. We average over *Number of runs* different training runs of the network.

We also do an evaluation where we train 15 versions of the network with 3 timesteps and measure how many training steps it takes each predicted state to reach 98% similarity with the corresponding ground truth state (the states should correspond since the number of timesteps are equal). On average, the 3rd state reaches 98% accuracy 2773 training steps before the 4th state. The 2nd state reaches 98% accuracy 5533 training steps before the 4th state. The network is learning to predict the intermediate states before the final one, even though it was trained to predict the final state, but not the intermediate ones. We think the network is learning to predict the intermediate states first because they are necessary for predicting the final state.

We also measure the impact of other hyperparameters. We find that having 20 channels in our CNN layers improves the metric by 0.09 over larger numbers of channels. We also find that having 1 layer in the encoder and decoder improves the metric by 0.14 compared to 3 layers.

3.1.2. Results with a different number of model and GoL timesteps

We also measure the results when the number of model and GoL timesteps are different as shown in table 2. This is a more difficult problem because the model cannot simply use each model timestep to predict one GoL timestep.

GoL timesteps	Model timesteps	Number of runs	Mean metric value
2	3	16	0.73
3	2	22	0.74

Table 2: Results where the number of GoL and model timesteps are different. Note that the runs with 2 GoL timesteps and 3 model timesteps can have a metric value of 1.5 if both of their intermediate model states match the single intermediate ground truth state. These runs use the RNN and the autoencoder.

3.1.3. Impact of adversarial training

We measure the impact of our adversarial training technique and show it in table 3. The technique usually improves results, sometimes by about 4x. Adversarial training helps more in the cases where the interpretability score is low without it.

3.2. Game of Life (LSTM)

We also apply our visualization technique to an LSTM trained on the Game of Life. The key differences from the RNN setup are:

- The LSTM is trained to predict the *number* of living cells after a certain number of timesteps, rather than the full game state. This is a more challenging task for our method since it is less obvious that the network will need to predict the intermediate states. To do so, we add a ”count cells” network after the LSTM to output the predicted cell count.
- The decoder is only trained to reconstruct the initial game state from the encoder output, not the final state.

GoL timesteps	Model timesteps	Use auto-encoder	Use RNN	Number of runs	Mean without adversarial	Mean with adversarial
2	3	True	True	16	0.58	0.65
3	2	True	True	22	0.15	0.71
3	3	True	False	20	0.11	0.40
3	3	False	True	10	0.44	0.43

Table 3: Results on a subset of experiments with and without the adversarial training technique. The adversarial training technique improves the results in every case except when we do not use the autoencoder.

We measure the correlation between the ground truth GoL states and the states reconstructed by applying the decoder to the intermediate LSTM activations. We exclude the initial state from the metric since the decoder is explicitly trained on it, but include the final state. Table 4 shows the mean correlation values averaged over multiple runs for different combinations of GoL and LSTM timesteps. The high correlations indicate that, even in this more challenging setup, the LSTM still represents information about the intermediate game states in its hidden activations, and our technique is able to extract that information.

GoL timesteps	LSTM timesteps	Mean correlation	Number of runs
2	2	0.79	8
2	3	0.81	5
2	4	0.73	9
3	2	0.93	7
3	3	0.76	8
3	4	0.76	4
4	3	0.86	3

Table 4: Results of applying our visualization technique to an LSTM trained to predict the number of living GoL cells after a given number of timesteps.

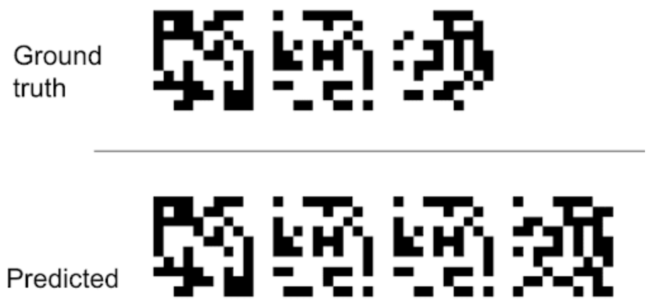


Figure 1: Comparison of the reconstructed states from the LSTM's intermediate activations (top row) and the ground truth GoL states (bottom row). Each column represents a timestep. The first column always matches as the decoder is trained to reconstruct the initial state. The 2nd and 3rd reconstructed states closely match the 2nd GoL state, while the 4th reconstructed state somewhat resembles the final GoL state.

Figure 1 shows an example of the reconstructed states from the intermediate LSTM activations compared to the ground truth GoL states, which has correlation of 0.87.

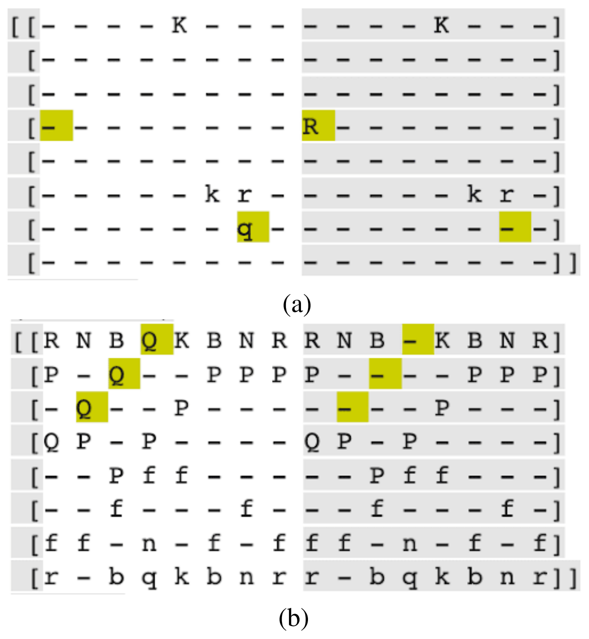
3.3. Alpha Chess

We also apply our visualization technique to a pre-trained AlphaZero chess model to investigate whether it represents potential future board states during its decision-making process. Chess represents a more complex and challenging domain than the Game of Life, and the model's internal planning process, if it exists, may involve more abstract representations.

The AlphaZero model uses a residual network architecture with separate heads for predicting the value and policy (move probabilities) from an input board state. We freeze the pre-trained weights and train a decoder to reconstruct the input state from the activations of the first convolutional layer. We also train the decoder to predict the board state after the model's chosen move, using activations from the final layer before the value and policy heads split. This aims to encourage a consistent state representation throughout the network.

However, when we apply the decoder to activations from the intermediate residual blocks, we do not find consistent evidence of human-interpretable board states or valid moves in these reconstructions. Applying an adversarial loss does not substantially improve the interpretability of the reconstructions.

Figure 2 shows some examples of board states reconstructed from the intermediate layer activations of the AlphaZero model, along with the corresponding input states. In some cases, pieces would disappear or appear in the reconstructed states (Figure ??). While we occasionally observe reconstructions that resemble valid moves, such as a set of queen moves (Figure ??), these interpretable patterns are not consistently produced across different input states, model layers, or training runs.



```
[[- - - - K - - - - - - - K - - -]
 [- - - - - - - - - - - - - - - -]
 [- - - - - - - - - - - - - - - -]
 [- - - - - - - - R - - - - - - -]
 [- - - - - - - - - - - - - - - -]
 [- - - - - k r - - - - - - k r -]
 [- - - - - - q - - - - - - - - -]
 [- - - - - - - - - - - - - - - -]]
```

(a)

```
[[R N B Q K B N R R N B - K B N R]
 [P - Q - - P P P P - - - - P P P]
 [- Q - - P - - - - - - - P - - -]
 [Q P - P - - - - Q P - P - - - -]
 [- - P f f - - - - - P f f - - -]
 [- - f - - - f - - - f - - - f -]
 [f f - n - f - f f f - n - f - f]
 [r - b q k b n r r - b q k b n r]]
```

(b)

Figure 2: Examples of board states reconstructed from the intermediate layer activations of the AlphaZero model. (a) A reconstructed state where pieces disappear or appear compared to the input state. (b) A reconstructed state resembling a set of valid queen moves.

These results suggest two possible interpretations:

1. The AlphaZero model does not perform explicit internal planning or search, and instead relies on pattern recognition and heuristics learned from its training data.

2. The model does engage in some form of planning, but represents the considered states in a higher-level, abstract format that our decoder network fails to translate into human-interpretable board states.

Human chess players often think in terms of abstract strategies and objectives rather than explicitly visualizing future board positions. It is plausible that the AlphaZero model learns to plan using similarly high-level representations, which would not necessarily map to valid board states when decoded. If this is the case, uncovering the model's internal planning process would require more sophisticated interpretability techniques capable of visualizing abstract, semantic information rather than just raw board states. It is also possible that the model's planning representations are encoded in a format fundamentally incompatible with our decoder-based approach.

4. Related work

(nostalgebraist, 2020) also applies the decoder to intermediate layers to help interpretability. The difference with our work is that they apply the technique to a different architecture and do not have the goal of visualizing intermediate environment states. They also do not use an adversarial technique or have results showing how architecture decisions impact interpretability. See appendix **??** for more related work. NNs have produced unprecedented advances in recent decades in a variety of tasks. However, their inner workings are still unclear. A large number of previous works have focused on visualizing NNs outputs, activation function and hidden layers. For example, (Yosinski et al., 2015) provides a tool to visualize neurons in pre-trained Convolutional NNs (CNNs). Their method allows for deeper local understanding of neuron computations. This tool only works for images and videos.

Others such as (Szegedy et al., 2014) show that CNNs encoding maintains photographically accurate information about the original input several layers in the network. Their approach uses shallow and deep representation to invert an objective function with Stochastic Gradient Descent (SGD) that allows them to reconstruct the original input. Our method also sheds light on what the intermediate layers of a network represent.

Another popular approach for unpacking the inner-workings of NNs has been via visualization. More specifically, the use of saliency maps (Simonyan et al., 2014) highlights an area (map) of the image such that the classifier (e.g. CNN) can discriminate given a class. This method has proved to be a powerful tool for image region segmentation.

More recently, (Mordvintsev et al., 2020) investigated learning a cellular automata update rule. They use a recurrent network with simplified CNN-like layers, with 'per-pixel' dropout. They train it on the task of generating target images. Their visualizations show that the network also learns interpretable intermediate representations. One difference with our method is that we use a standard CNN architecture instead of a modified one. We also use a decoder which we apply to the intermediate representations, where they treat the last 3 channels as the visualization instead of using a decoder. We also apply our technique to a domain with complex intermediate states, while their intermediate states are usually subsets of the target image. We plan to further test our technique by applying it in the domain of generating target images.

Finally, the authors of (Zeiler & Fergus, 2013) consider the task of visualizing the intermediate feature layers of a CNN as well as the operation used for classification. Their approach shows significant improvement on ImageNet. Our work differs in that instead of trying to extract specific features from the hidden states, we focus on reconstructing environment states.

5. Conclusion and future work

In our research, we explore a new interpretability problem of visualizing the hidden intermediate states that a network represents. We show that in a simple environment using an autoencoder and adversarial techniques, we can obtain hidden states that closely match the ground truth states of the environment. However, the idea didn't end up working out when we tried it on a larger neural network trained on chess, highlighting the limitations of our approach for more complex models and domains. In future work, we plan to investigate ways to extend our technique to handle such complex scenarios, where the network may represent the consequences of a decision it is considering.

References

John Conway. The game of life. *Scientific American*, 223 (4):4, 1970.

Evan Hubinger, Chris van Merwijk, Vladimir Mikulik, Joar Skalse, and Scott Garrabrant. Risks from learned optimization in advanced machine learning systems. *arXiv preprint arXiv:1906.01820*, 2019.

Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, and Michael Levin. Growing neural cellular automata. *Distill*, 2020. doi: 10.23915/distill.00023. https://distill.pub/2020/growing-ca.

nostalgebraist. Interpreting gpt: the logit lens. 2020. URL https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2014.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014.

Jason Yosinski, Jeff Clune, Anh Mai Nguyen, Thomas J. Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *CoRR*, abs/1506.06579, 2015. URL http://arxiv.org/abs/1506.06579.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. URL http://arxiv.org/abs/1311.2901.

A. Results samples

These are cherry picked examples to illustrate how our metric works.

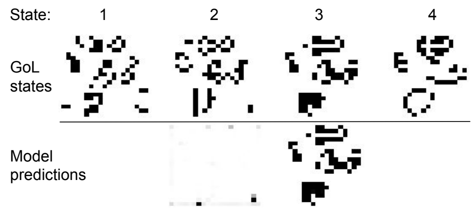


Figure 3: An example with 3 GoL timesteps and 3 model timesteps. The metric gets a value of .5 because the 3rd predicted state matches the ground truth, but the 2nd predicted state does not. The 1st and 4th states are ignored because the network is trained on them.

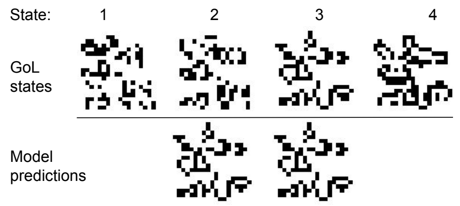


Figure 4: The metric gets a value of .75 because both predicted states match the same ground truth state. So the 3rd predicted state gets a half score.

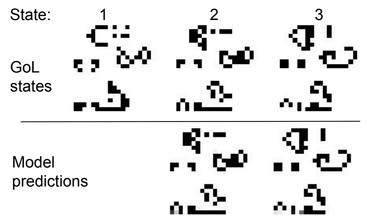


Figure 5: An example with 2 GoL timesteps and 3 model timesteps. The metric gets a value of 1.0. The 2nd model state matches the intermediate GoL state. The 3rd model state matches the 3rd GoL state, but this is ignored because the model was trained to predict the 3rd GoL state. The denominator in the metric is 1 because $min(2, 3-2) = 1$

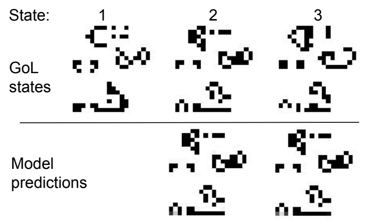


Figure 6: An example with 2 GoL timesteps and 3 model timesteps. The metric gets a value of 1.5. The 2nd model state matches the intermediate GoL state. The 3rd model state also matches the intermediate GoL state, so the metric gives .5 score for this state. Note that we ignore the last GoL state and not the last model state, to this match is a valid one