Visualizing Neural Weight Evolution in Reinforcement Learning Agents

Padraic Bergin, Sonny Box

December 12, 2024

Abstract

This work presents a novel approach to visualizing the evolution of neural network parameters during training, with the focus on reinforcement learning models. Through examining changes in weight matrices, we aim to potentially uncover insights into how models learn and adapt over time. In our project we use a reinforcement learning agent trained to play the classic Snake game. During training we extract weights from 3 layers in the neural network to visualize weight distributions across training episodes. This visualization highlights parameter evolution and layer-specific transformations, offering an a way to understand a part of the learning process. Our findings should not only enhance interpretability in reinforcement learning but also may suggest pathways for improving model training and optimization strategies.

1 Introduction

Neural networks are powerful computational models that learn to approximate complex functions through layers of interconnected neurons. Each neuron applies mathematical transformations to input data, and these transformations, combined across layers, enable the network to recognize patterns and make predictions. However, the complexity of these transformations and the interactions of weights connecting neurons pose significant challenges to understanding how models make decisions.

In feedforward neural networks, neurons are organized into layers, with each layer fully connected to the next. Neurons apply linear transformations

followed by non-linear activation functions, progressively refining the features of the input data until the final output is produced. During training, the network's outputs are compared to expected values, and weights and biases are adjusted using backpropagation, guided by the gradient of a loss function. While this process enables models to achieve high accuracy, it obscures the inner workings of the model, leaving the reasoning behind predictions difficult to interpret. This opacity is particularly problematic in reinforcement learning, where models are trained to maximize long-term rewards through trial-and-error interactions with an environment.

Understanding how weights evolve during training can illuminate the learning process and demystify the factors influencing predictions. Visualizing these changes provides insights into the relationships between features, neuron activations, and the model's overall performance. In our project we employ color mapping techniques in order to highlight the evolution of weight distributions, allowing us to track how initially random parameters converge toward meaningful patterns that approximate the underlying function of the problem.

In order to do so, we focus on a reinforcement learning model trained to play the classic Snake game, where the agent must navigate a grid-based environment to consume apples while avoiding collisions. Using rewards to reinforce successful behaviors and penalties for errors, the model learns to improve its performance over time. To better understand this learning process, we extract and analyze the weight matrices of the neural network. These matrices, representing the weighted connections between neurons, are visualized using Min-Max normalization and scalar mapping techniques to reveal their dynamics across training epochs. By interpreting these visualizations, we aim to uncover patterns in parameter updates and gain deeper insights into how reinforcement learning models adapt and refine their decision-making.

2 Previous Work

In our CS 453 Scientific Visualization class [3], we explored various methods to visualize scalar fields, which provided a foundation for our current work on visualizing neural network parameters. In our first project, we learned how to create different types of visualizations, such as gray-scale maps, bi-color maps, rainbow maps, height maps, and combined maps. Each technique had its strengths and weaknesses in representing scalar fields. For instance, 3D

height maps offer more detailed representations through the added dimension, but the challenge is that a single frame of this visualization is still a 2D projection of a 3D object. This loss of dimension means that the relationships between points are only apparent when observing changes across multiple frames.

This limitation became apparent in our report, where the dynamic movement of the visualization was lost in the static nature of screenshots. To mitigate this, we applied color mapping to 3D objects, allowing patterns to emerge that make the depth and relationships of the 3D data more comprehensible to the human eye. This approach of using color mapping to reveal patterns in complex data is a key inspiration for our current project, where we use color mapping to visualize the evolution of neural network weights during training. In our project, we aim to visualize how different weights and biases change as the reinforcement learning model learns to navigate the Snake game, providing insights into how the model adapts and learns over time.

In our CS 434 Machine Learning and Data Mining course [1], we learned about neural networks. The structure of the course allowed us to first understand the process of logistic regression and methods to fit a classifying line in a dataset to separate classes. The fascinating revelation in the neural network section was how a neural network was simply a nonlinear combination of logistic regressor. This idea highlighted the capabilities of combining linear functions in a way to represent an unknown complex function. This course proved to be extremely useful in understanding how our Snake model worked and how we could apply our CS 453 visualization techniques to interpret the model's learning process more effectively.

In our CS 434 Machine Learning and Data Mining course, we also participated in an in-class demonstration using an online neural network playground. This tool provided a dynamic visualization of a neural network during training, showing how each neuron progressively specialized in recognizing specific patterns within the input data. As the network trained, we could observe the evolution of each neuron's response to different aspects of the input, with neurons in the hidden layers gradually becoming more specialized in recognizing particular features of the data.

This hands-on experience demonstrated the potential of visualizing the inner workings of a neural network in an intuitive and accessible way. It emphasized how visualizations can reveal the learning process of a network, offering insights into how neurons adapt and specialize as the model trains.

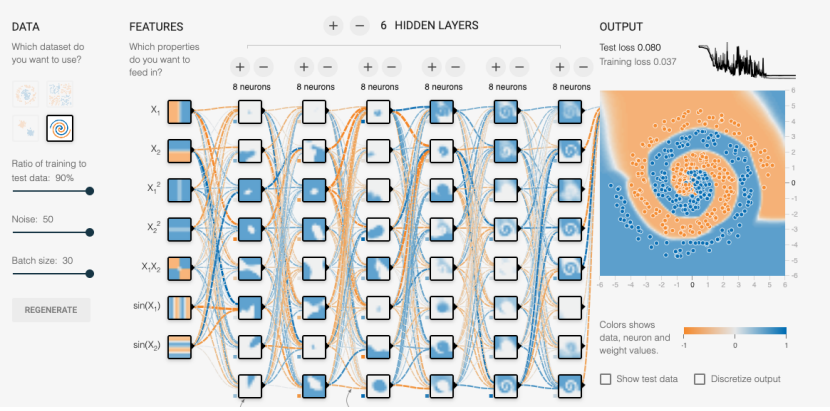


Figure 1: Screen capture of the demo showing how the neurons learn the spiral pattern of the dataset and how to classify orange and blue dots. Adapted from [2].

This experience inspired us to explore similar visualization techniques in our project, where we aim to track the evolution of the weight matrices in our reinforcement learning model.

3 Background

3.1 Deep Q-Network Implementation

The implemented agent is based on a three-layer Deep Q-Network (DQN) architecture:

- **Input layer**: Matches the flattened game board size. For a board of size $n \times n$, the input dimension is n^2.
- **Hidden layers**: Two fully connected layers with ReLU activations, each of size *hidden_size*. These layers transform the raw state representation into increasingly abstract feature representations.
- **Output layer**: Outputs Q-values for each possible action. The dimensionality of this layer corresponds to the number of actions available to the agent in the Snake environment.

3.2 Reward Function

To guide the agent's learning, a shaped reward function is employed:

- A small positive reward (e.g., +0.1) is given when the agent moves closer to the apple.
- A small negative penalty (e.g., -0.1) is given when the agent moves farther from the apple.
- A larger positive reward (+1) is provided when the snake successfully eats an apple.
- A larger negative penalty (-1) is provided when the snake collides with walls or its own body, ending the episode.

This combination of rewards and penalties offers a finer granularity of feedback, encouraging progress toward the apple while discouraging wasteful or harmful movements.

3.3 Replay Buffer and Training Process

To stabilize learning, a replay buffer stores transitions (s, a, r, s'):

- **Sampling Experiences**: During training, mini-batches of past experiences are randomly sampled from the buffer, breaking the temporal correlations present in sequential observations.
- **Loss Function**: The Mean Squared Error (MSE) loss is used to minimize the difference between the predicted Q-values and the target Q-values derived from the Bellman equation.
- **Target Network**: A target network, periodically updated with the parameters of the main network, is employed to stabilize the training. This helps prevent oscillations or divergence in Q-value estimates.

3.4 Exploration Strategy

The agent uses an epsilon-greedy exploration strategy:

- ϵ-greedy: With probability ϵ, the agent chooses a random action to explore the state-action space. Otherwise, it selects the action with the highest Q-value.
- ϵ Decay: Over time, ϵ is decayed, gradually shifting the agent's behavior from exploration toward exploitation as it gains confidence in its learned policies.

3.5 Game Environment

The Snake environment is a grid-based game:

- **State Representation**: The state is represented as a flattened 1D tensor derived from the $n \times n$ game board, encoding the snake's position, apple position, and obstacles.
- **Apple Spawning**: Apples are spawned deterministically at predefined or patterned locations, simplifying the state distribution the agent encounters.
- **Manhattan Distance**:

  $$d_{\text{manhattan}} = |x_1 - x_2| + |y_1 - y_2|$$

  This metric is used to evaluate whether a move brings the snake closer to or farther from the apple, influencing the shaped rewards.

4 Visualization of Neural Weights

4.1 Weight Extraction and Normalization

Weights are extracted from the trained DQN's layers during or after training. To make them suitable for visualization:

- **Normalization**: Normalize the weights to a [0, 1] range using:

  $$\text{normalized\_weights} = \frac{\text{weights} - w_{\min}}{w_{\max} - w_{\min}}$$

  This normalization preserves relative magnitudes while eliminating the influence of arbitrary weight scales.

4.2 PLY File Generation and 3D Visualization

The normalized weights are then mapped to 3D coordinates and stored in PLY files:

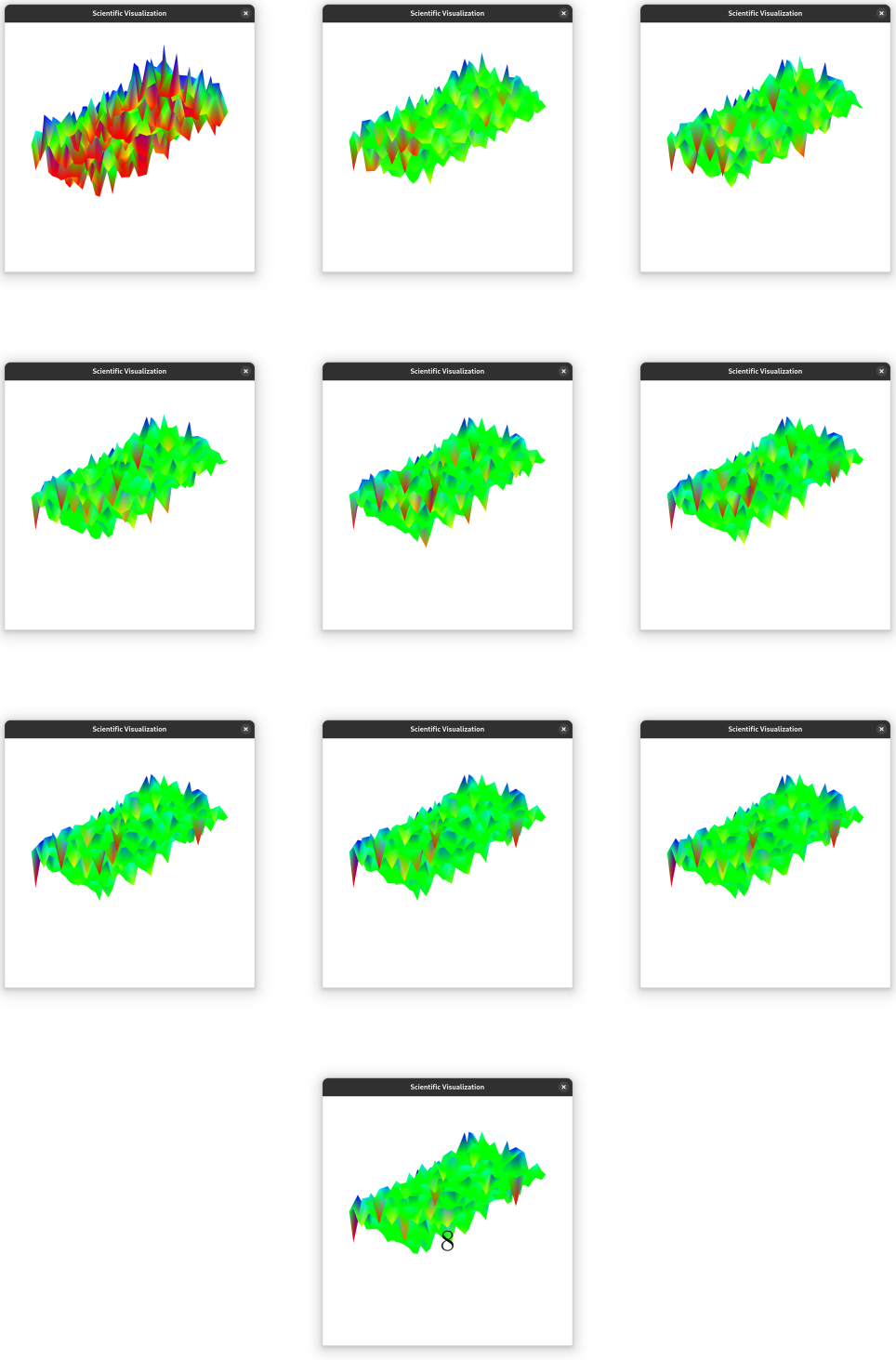
- **Vertex and Face Definitions**: Each neuron or weight can be represented as a vertex in 3D space, with color mappings applied based on the normalized weight values.
- **3D Rendering**: For the final step to render our datasets we used preexisting Project 2 code.

5 Division of Tasks

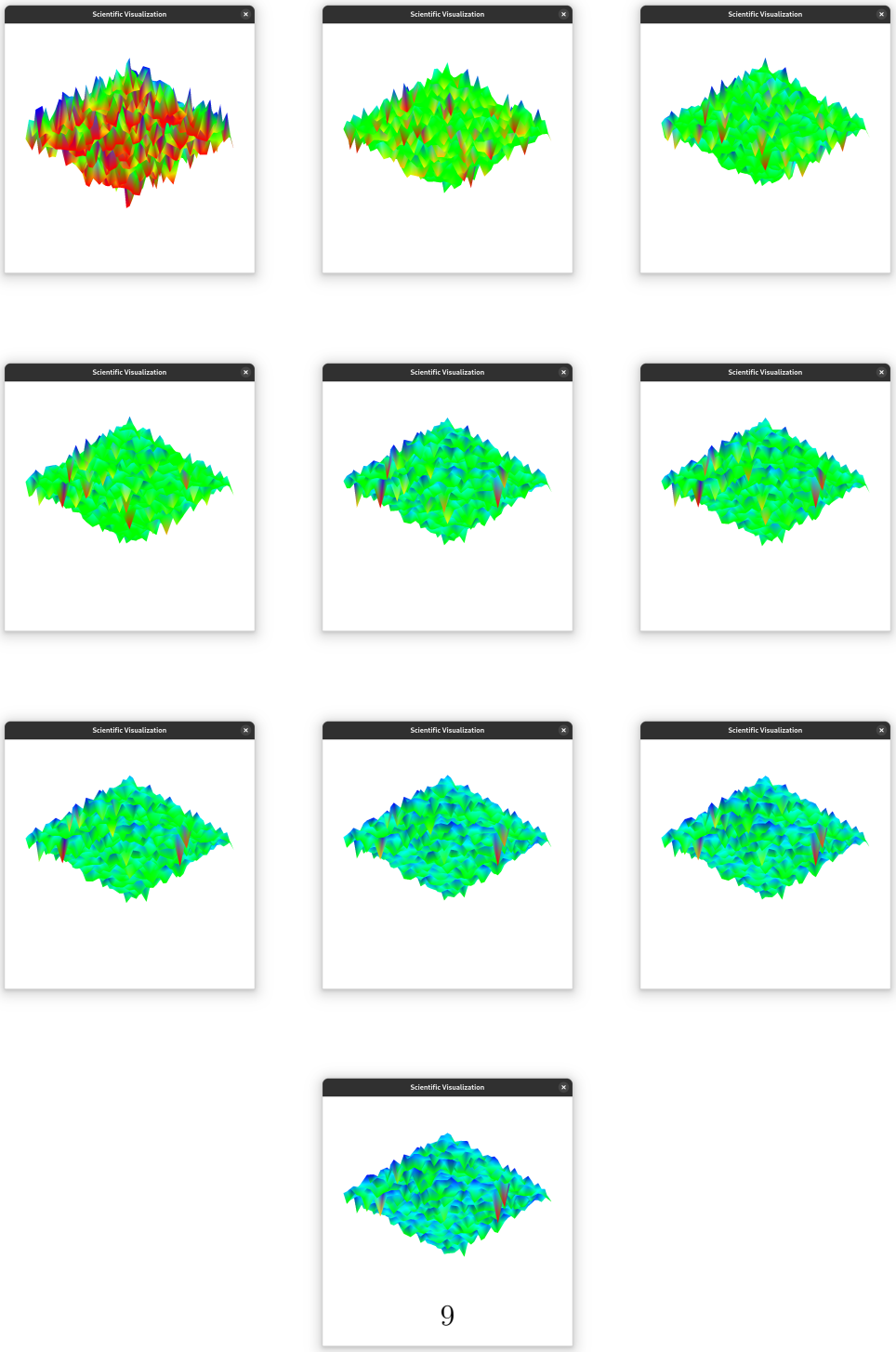
- **Sonny Box**: Generate datasets by setting up the Python environment and Snake game agent.
- **Padraic Bergin**: Generate visualizations using Project 2 code and prepare the paper for final submission.
- **Both**: Discuss findings.

6 Results

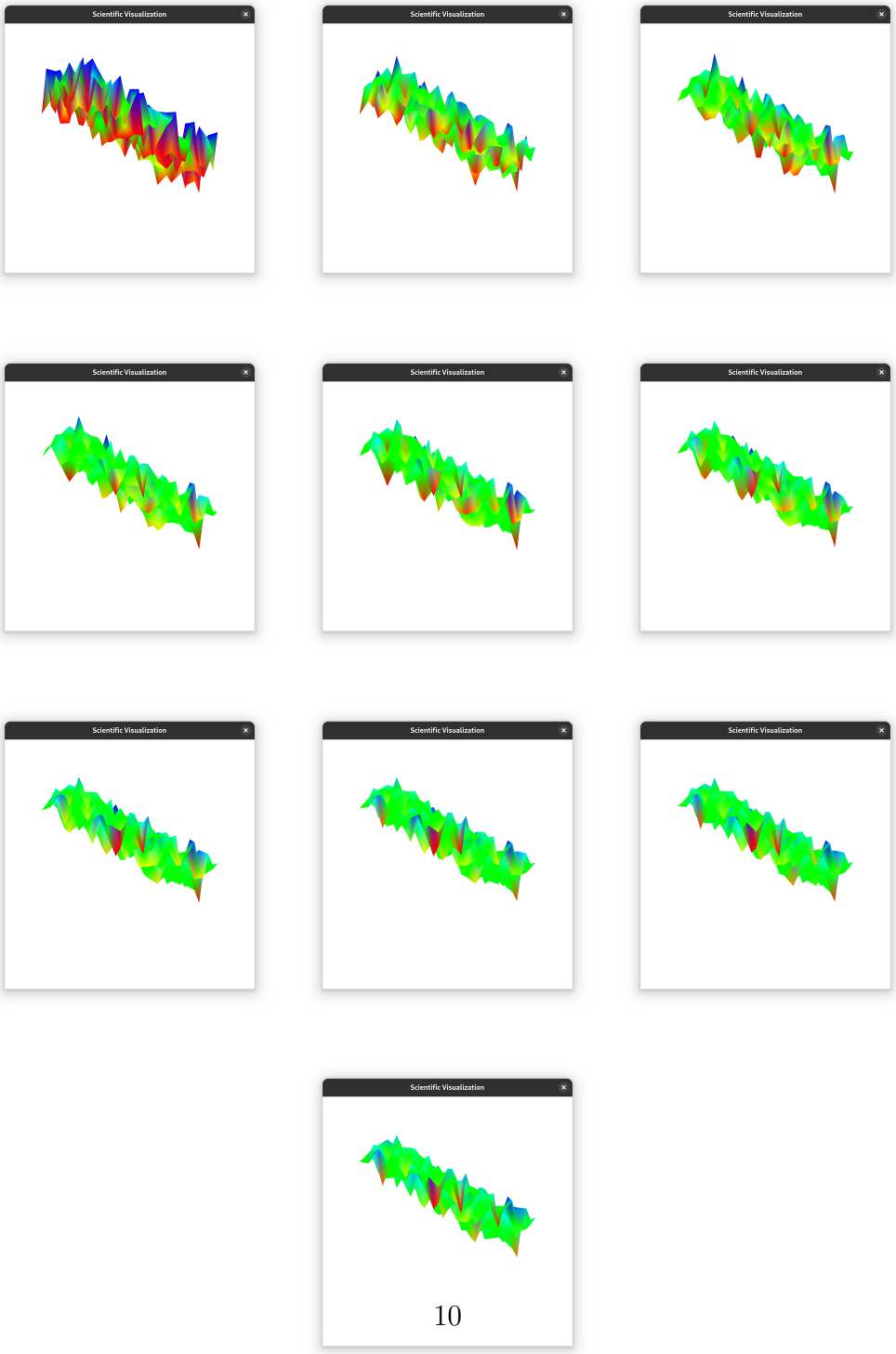
6.1 Layer 1



6.2 Layer 2


Scientific Visualization
Scientific Visualization
Scientific Visualization
Scientific Visualization
Scientific Visualization
Scientific Visualization
Scientific Visualization
Scientific Visualization
Scientific Visualization
Scientific Visualization

6.3 Layer 3


Scientific Visualization
Scientific Visualization
Scientific Visualization
Scientific Visualization
Scientific Visualization
Scientific Visualization
Scientific Visualization
Scientific Visualization
Scientific Visualization
Scientific Visualization

6.4 Plot of results data

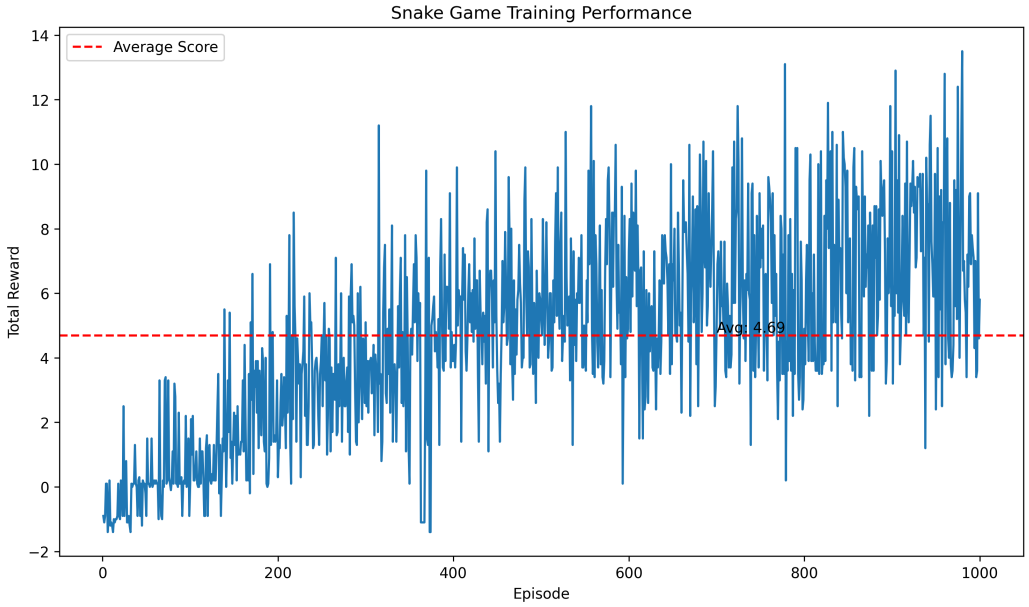


Figure 2: The agent scores on average 4.09 reward points across 1000 episodes, with a maximum of approximately 13 points.

6.5 Evaluation

6.5.1 Layer 1 Observations

In the initial episodes, the visualization of Layer 1 shows a diverse range of data, with distinct red, green, and blue regions clearly visible. This indicates high variability in the weights during the early stages of training. By the 200th episode, much of the irregularity has diminished, with the visualization predominantly displaying green values.

Notably, certain red peaks remain consistent across all episodes, suggesting that some patterns in the input data are preserved throughout training. This observation aligns with the design of the environment, where the snake and apple consistently spawn in the same locations. These consistencies in the visualization reflect the model's ability to adapt and generalize features from the game board's initial configuration.

6.5.2 Layer 2 Observations

Layer 2 is responsible for forming relationships between the input layer and the output layer, thereby playing a critical role in decision-making. Early in training (e.g., the first 100 episodes), the visualization is characterized by high noise, with many sharp peaks and valleys spanning the red, green, and blue color ranges. As training progresses, the visualizations transition to predominantly green tones by approximately the 400th episode, and ultimately to a more uniform aquamarine surface with prominent blue peaks by the 900th episode.

The consistent red peaks observed in Layer 2 suggest biases in the decision-making process, stemming from the fixed spawning positions of the snake and apple. These biases indicate that the agent has evolved a stable and consistent strategy in the later episodes. This evolution is further supported by the uniformity in the visualization and the trends observed in the overall performance metrics.

6.5.3 Layer 3 Observations

The visualization of Layer 3 reflects the simplicity of the action space in the snake game, where the agent is constrained to three possible movement directions (excluding the opposite of its current direction). Early in training (e.g., the first 100 episodes), the visualization shows significant noise, indicative of the agent's exploration of the action space. By the 300th episode, this noise dissipates, and the visualizations evolve into a simple and consistent landscape, reflecting the agent's convergence to a deterministic and efficient strategy.

6.6 Conclusion

The weight visualizations across the three layers of the Deep Q-Network provide valuable insights into the training process and the agent's learning dynamics:

- **Layer 1**: Captures the game board's consistent features, with notable patterns reflecting the fixed initial configurations of the environment.
- **Layer 2**: Demonstrates the model's ability to form relationships between the input and output layers, transitioning from high noise to a uniform and structured representation that reflects a stable strategy.

- **Layer 3**: Highlights the simplicity of the action space, with rapid convergence to a consistent decision-making pattern.

To conclude, the visualizations confirm the model's ability to adapt, generalize, and develop an efficient strategy for the snake game. The progressive simplification and stabilization of the weight landscapes indicate effective learning and convergence during training.

References

[1] Stefan Lee. *CS 434: Machine Learning and Data Mining*. Course taught by Dr. Stefan Lee at Oregon State University. 2024.

[2] Google Research. *TensorFlow Playground*. Accessed: 12 Dec. 2024. n.d. URL: `https://playground.tensorflow.org/`.

[3] Eugene Zhang. *CS 453: Scientific Visualization*. Course taught by Prof. Eugene Zhang at Oregon State University. 2024.