Reinforcement Reservoirs

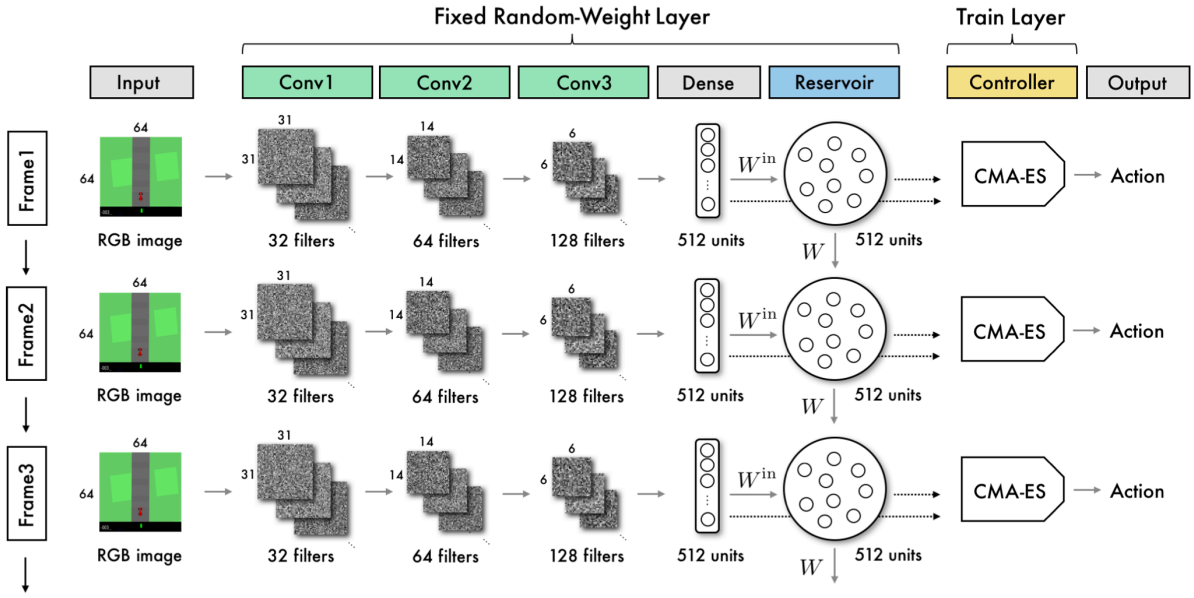
Reinforcement Learning using Reservoir Computing

Introduction

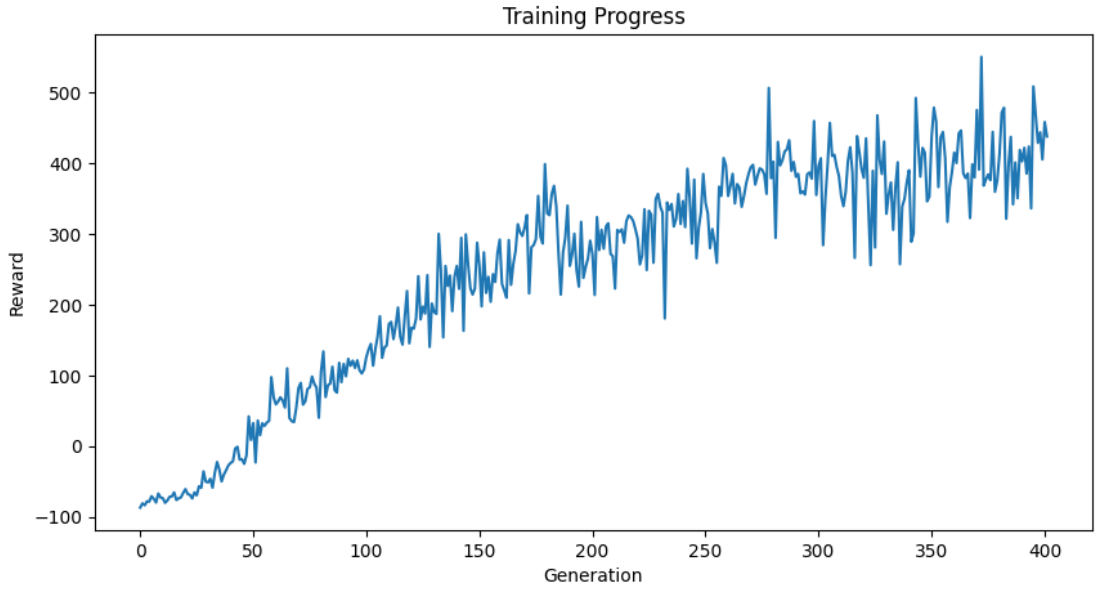
Reservoir Computing (RC) has many advantages over regular Recurrent Neural Networks (RNNs), the most important being that only the readout neuron needs to be trained, saving computational resources. Also, any number of output neurons can be added to the same reservoir without interfering with the already trained output neurons, so the same model can potentially solve several different tasks. Researchers have recently begun to re-focus on reservoir computing, particularly in the design of computer chips, thus creating physical reservoirs, but despite this newfound interest, there remains a neglected area, namely reinforcement learning. In addition to the already mentioned advantages of reservoir computing over regular RNNs, reinforcement learning opens up many new ones, e.g. the introduction of complex dynamics by increasing the spectral radius of the reservoir results in better training phases, described as "the best value for learning is when the dynamics of the reservoir is around the edge of chaos" (Matsuki, 2022), and this chaos also introduces a new field of reinforcement learning called Chaos-based Reinforcement Learning (CBRL), "which exploits the internal chaotic dynamics of the system for exploration" (Matsuki, 2022). This means that the reservoir essentially has a built-in exploration strategy that is often difficult to implement manually, for example by setting the right parameters for a greedy search.
Thus, these described mechanisms make RCs an interesting candidate for solving reinforcement learning tasks, as in the two following papers by Chang and Futagami (2019) and Matsuki (2022).

Chang et al. 2019 -
Reinforcement Learning with Convolutional Reservoir Computing

For the first model, I replicated Chang and Futagami (2019), who trained a model to play a car racing game by using an untrained CNN to preprocess the game frames and then passing them through the reservoir. The authors used a spectral radius of $g = 0.95$ and a leakage rate of $\alpha = 0.8$ for the reservoir. In general, it can be said that a small spectral radius leads to stable dynamics, while a larger radius leads to chaotic dynamics, with the cut-off being somewhere close to $g = 1.0$. The chosen leakage rate of $\alpha = 0.8$ makes the reservoir more prone to 'forgetting', i.e. a rather low recall of previous states, with leakage rate values being defined between 0 and 1, and a small leakage rate meaning a high recall of previous states and vice versa. So the model can be described as somewhat stable, only taking in a few frames before 'forgetting' them.
The task itself was to solve this racing car game by using an untrained Convolutional Neural Network (CNN) to project the game frame into a lower dimensional space. This lower dimensional representation is then fed into the reservoir and then both the CNN output and the reservoir output are concatenated and fed into a readout neuron to determine the action to be taken, e.g. steer left, right, brake etc. Below is an image from the paper (Chang & Futagami, 2019) showing the proposed model architecture:



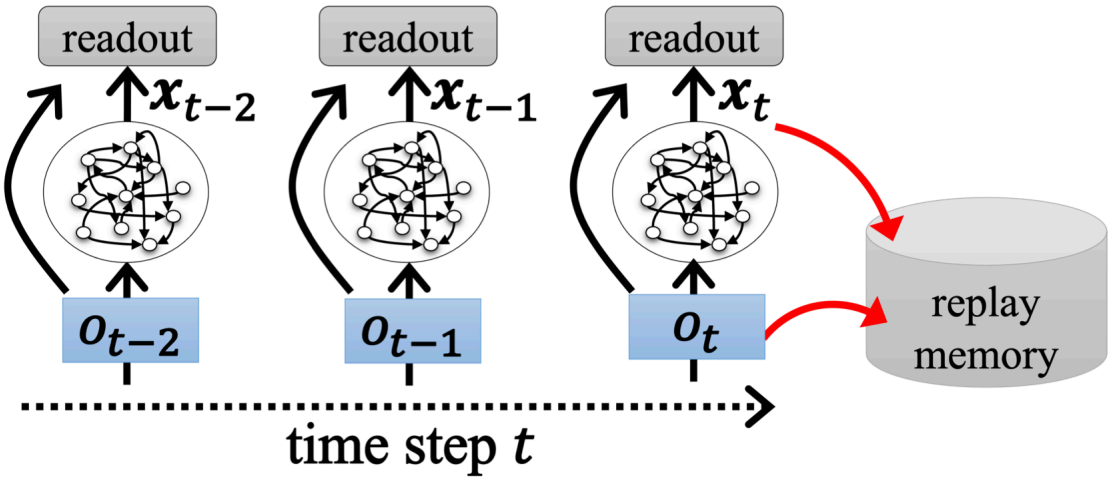
The optimisation of the readout neuron is done by an evolutionary-based method called Covariance Matrix Adaptation Evolution Strategy, or CMA-ES for short, where multiple weight distributions are tried at once and then combined by the weighted rewards achieved by each. As the authors didn't provide any code, the model architecture and training can be found in my GitHub repository. A short demonstration of the trained model can be seen either in this Google Colab notebook or in the corresponding Jupyter notebook from the repository. The general training went quite well and can be seen in this graph :



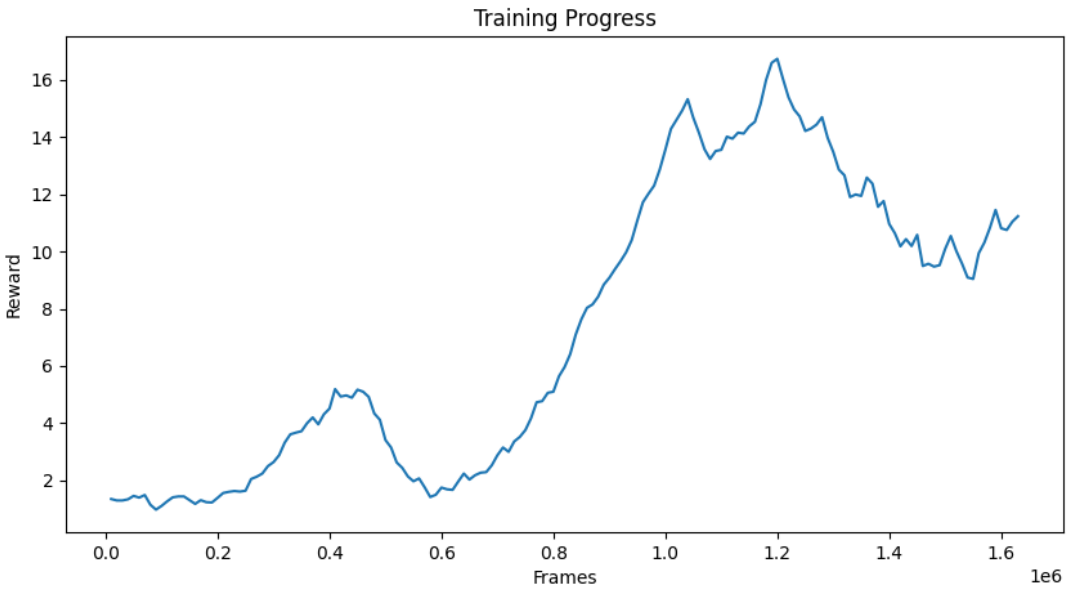
Toshitaka Matsuki 2022 -
Deep Q-network using reservoir computing with multi-layered readout

For the second replication, I implemented a model based on Matsuki (2022), which uses a multi-layer perceptron as the readout neuron and is trained using Q-learning to learn a reward function. Q-learning combined with deep neural networks is often referred to as Deep-Q Networks (DQNs), so the readout neuron is a Deep-Q model, while the initial setup combines the environmental observation and reservoir output as inputs to the readout neuron. While replicating the results, I encountered problems with the MountainCar task, so I started looking into the environments used in this study and found that, on the one hand, it proved to be almost obsolete with the chosen tasks of CartPole & Acrobot, as they were identified as "clearly limiting benchmarks, easily solved by traditional agents" (Hare, 2019), on the other hand, the "MountainCar environment is a difficult benchmark for neural network parametrised agents" (Hare, 2019), as the difficulty of the game is not based on the complexity of the task to be learned, but rather on the very sparse rewards that challenge the agents' ability to explore. Since the highest success rate was achieved with a spectral radius of g = 1.0, the authors did not take advantage of the natural ability of reservoirs to explore when a high spectral radius is chosen near the edge of chaos. Since the first introduction of Deep-Q networks was done by Mnih et al. (2013) at DeepMind by playing all the Atari 2006 games at a superhuman level, I decided to use this model proposed by Matsuki (2022) to successfully play the Atari Breakout game instead of a more or less redundant replication of the study.

To play the breakout game, we had to make some adjustments to the model architecture, mainly by introducing (untrained) CNNs to project the game frame into a lower dimensional space, similar to Chang and Futagami (2019). Everything else remained the same, except for the units in both the reservoir and the deep-q MLP, which now represent the more complex game environment. For Q-learning to work, each state must be stored in a replay memory, which can then be used by the network to update its reward function by going through certain previous states and trying to predict the best move, as seen in the figure in the paper by Matsuki (2022):



Thus, the original model architecture in the paper is presented as one where each observation of the environment (o_t-2, o_t-1, ...) is fed into memory, resulting in the memory states x_t-2, x_t-1, ... which are then concatenated and fed into the readout neuron to take the best action accordingly.
Unfortunately, I was only able to train the original model proposed by Mnih et al. (2013), and I stopped training once it was clear that it was working as intended, and the training process can be seen here:



Typically, good results are achieved at around 10 million frames, and the original model was trained on around 50 million frames.
While the RC variant is more efficient/faster as it only trains the readout neuron and smaller states, it also has a larger replay memory due to the smaller size of the states. The original paper has states of size 4*84*84 for 4 stacked greyscale frames of 84x84 pixels vs. a 1024 dimensional vector for the RC variant, with both the downsampled game frame via the CNN and the reservoir output being a 512 dimensional vector that is ultimately concatenated and thus substantially smaller than the states of the original paper.
In total, I spent >300 hours training different versions of the RC variant, but never got results higher than ~4.0, so it is only able to outperform a full random model, which has an average score of 1.7 and is worse than a linear model or a human player, with average rewards of 5.2 and 31.8 respectively, as noted by Mnih et al. (2015).
However, the general idea is sound as the game can be solved using Q-learning and a neural network, and I suspect that the downsampling method (via CNN) is responsible for the poor performance, so a different approach may lead to better results.
So I hope this project has given a good and quick introduction to Reservoir Computing and Reinforcement Learning, and given some motivation to tackle the task of creating the RC-DQN model that will be able to successfully play the Atari Breakout game in the future.

Chang, H., & Futagami, K. (2019, December 5). *Reinforcement Learning with Convolutional Reservoir Computing*. arXiv.org. https://arxiv.org/abs/1912.04161

Matsuki, T. (2022, March 3). *Deep Q-network using reservoir computing with multi-layered readout*. arXiv.org. https://arxiv.org/abs/2203.01465

Hare, J. (2019, October 21). Dealing with Sparse Rewards in Reinforcement Learning. arXiv.org. https://arxiv.org/abs/1910.09281

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013, December 19). *Playing Atari with Deep Reinforcement Learning*. arXiv.org. https://arxiv.org/abs/1312.5602

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236