Reinforcement Reservoirs

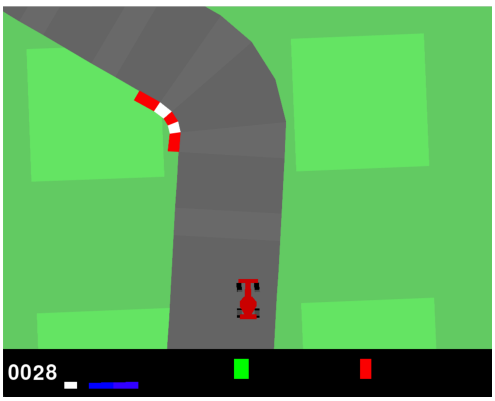
Reinforcement Learning using Reservoir Computing

Introduction

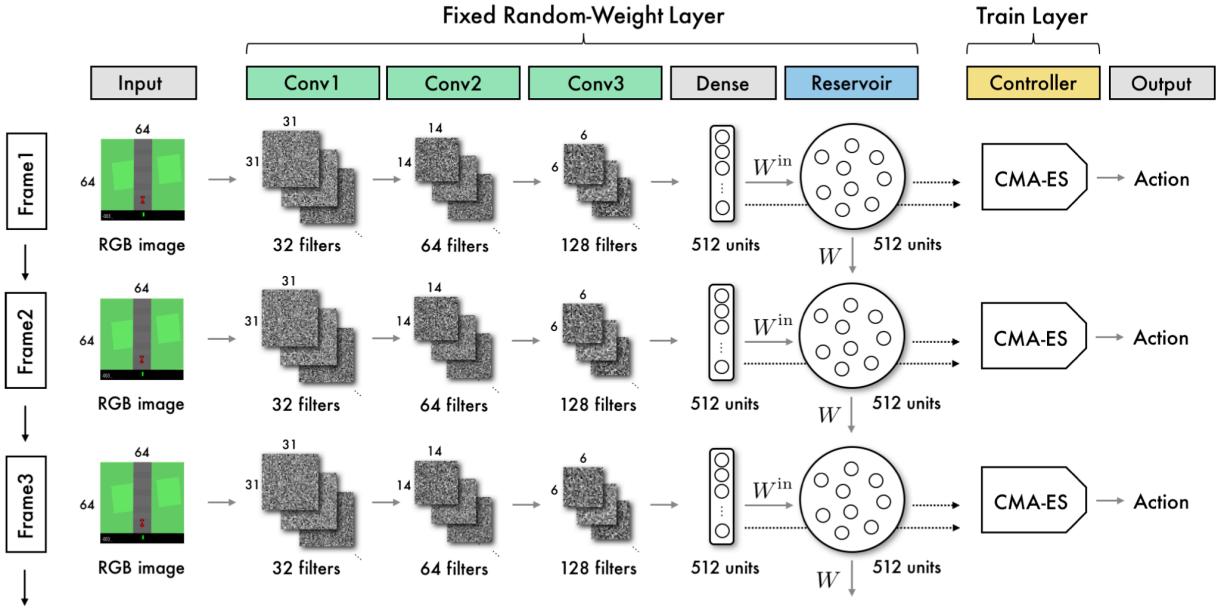
Reservoir computing (RC) has several advantages over regular recurrent neural networks (RNNs), the most important being that only the output neuron needs to be trained, saving computational resources. Also, any number of output neurons can be added to the same reservoir without interfering with the already trained output neurons, so the same model can potentially solve several different tasks, such as training another classifier with a binary output and then looking through the outputs of each output neuron to determine the most likely match. Researchers have recently begun to refocus 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, such as the introduction of complex dynamics by increasing the spectral radius of the reservoir, leading to better training phases, described as "the best value for learning is when the dynamics of the reservoir is at 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 exploration strategy.
Thus, these described mechanisms make RCs an interesting candidate for solving reinforcement learning tasks, as implemented in the following two papers by Chang and Futagami (2019) and Matsuki (2022).

Chang and Futagami 2019 - Reinforcement Learning with Convolutional Reservoir Computing

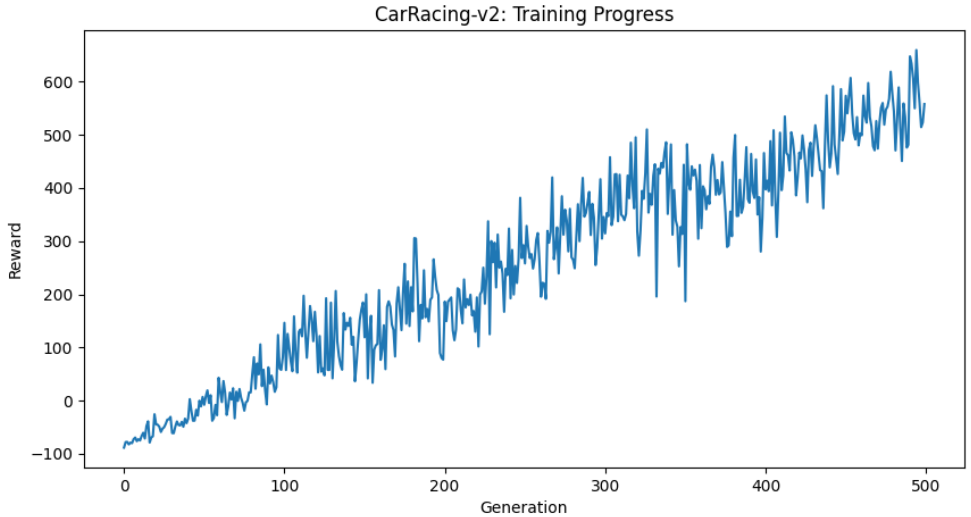
For the first model, I replicated Chang and Futagami (2019), who trained a model to play a car racing game by preprocessing the game frames with an untrained CNN and then passing them through the reservoir. The authors used a spectral radius of $g = 0.95$ and a leakage rate of $\alpha = 0.8$ for the 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 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, taking in only 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. steering left, right, braking etc. Below is an image from the paper (Chang & Futagami, 2019) showing the proposed model architecture:

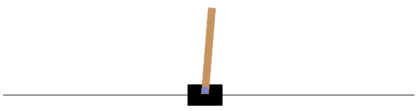


The optimisation of the readout neuron is done by an evolutionary-based method called Covariance Matrix Adaptation Evolution Strategy, or CMA-ES for short, where multiple weight distributions are tried at once and then combined by the weighted rewards achieved by each. As the authors didn't provide any code, the model architecture and training can be found in my GitHub repository. A short demonstration of the trained model can be seen in this Google Colab notebook. The general training went quite well, although I chose fewer parallel and averaging runs than in the original paper to both speed up and reduce the computational complexity of the training. The progress in achieving better scores for each generation can be seen in the graph below:

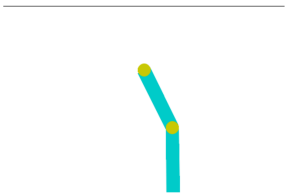


Toshitaka Matsuki 2022 -
Deep Q-network using reservoir computing with multi-layered readout

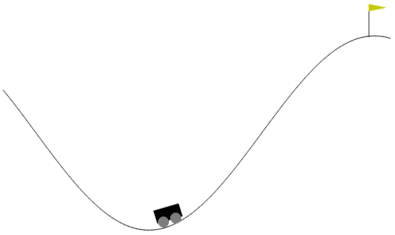
For the second replication, I implemented a model based on the work of Matsuki (2022) that uses a multi-layer perceptron as the readout neuron and is trained using Q-learning to learn a reward function. Q-learning combined with deep neural networks is often referred to as Deep-Q Networks (DQNs), so the readout neuron is a DQN, while the initial setup combines the environmental observation and the reservoir output as inputs to the readout neuron. So, unlike the first paper described above, this model doesn't take raw game frame inputs, but rather gets quite simple and specific environmental information, such as the agent's position and corresponding angles, while using a slightly more sophisticated learning algorithm with Q-Learning instead of CMA-ES.
When replicating the results, I encountered problems with the MountainCar task, as the training failed quite often, so I started to look at the environments used in this study and found that their solution does not necessarily prove a good choice of learning algorithm or model. Thus, both CartPole and Acrobot were identified as "clearly limiting benchmarks, easily solved by traditional agents" (Hare, 2019), while the "MountainCar environment is a difficult benchmark for neural network parametrised agents" (Hare, 2019), as the difficulty of the environment is not based on the complexity of the task to be learned, but rather on the very sparse rewards that challenge the agent's ability to explore. Below are screenshots of the environments in "human rendering mode", the actual input to the model is just a few numbers indicating positions and angles.



Cartpole



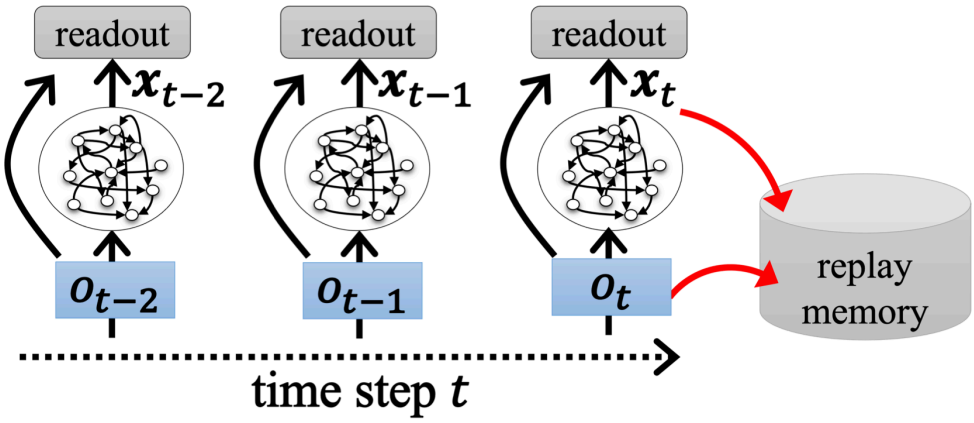
Acrobot



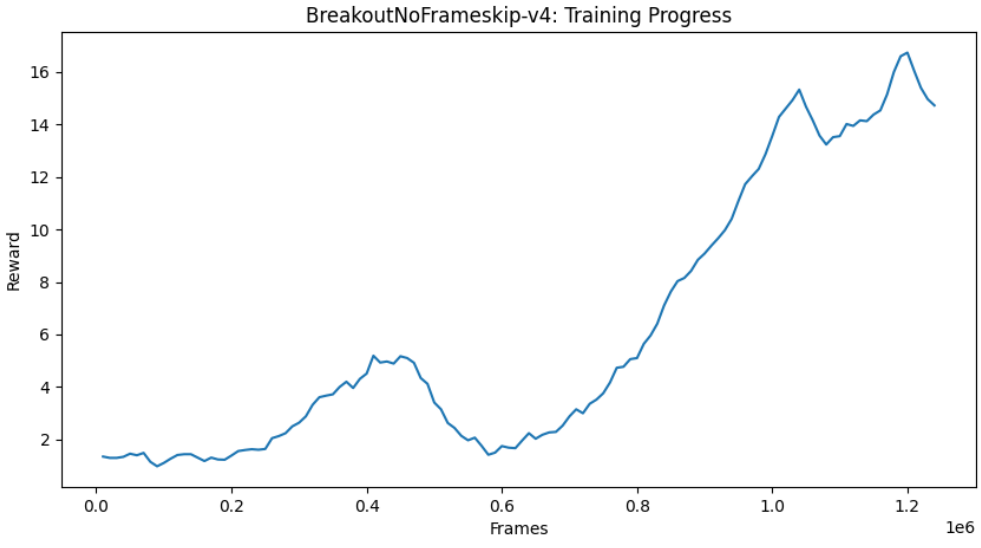
MountainCar

The paper generally had good insights into reservoirs and their influence on reinforcement learning, for example by going through several different reservoir configurations to determine the most suitable parameters, leading to the discovery that a spectral radius of g ~ 1.0 seems to be universally favourable, or that using an MLP instead of a single readout neuron also significantly improves learning. I also successfully replicated the results, but was left unsatisfied because the results may not prove that the proposed model is capable of learning more challenging and complex environments. One reason for these simple environments could be that "reservoir networks have difficulty dealing with high-dimensional inputs such as images", as the author wrote, so studying the influence of DQNs in combination with such data would only make the results more inconclusive, as the method of projecting the image into the reservoir is quite difficult and uncharted territory. The author then said that it might be possible to "overcome the challenges of dealing with high-dimensional inputs by using other techniques to extract features

and then using the reservoir network specifically for learning policy", and so I decided to use the DQN readout neuron approach proposed in this paper in a more complex environment.
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 a mixture of both models described so far, that is, using an untrained CNN to project the game frames into the reservoir, and then using a DQN as a readout neuron to learn to play the Atari Breakout game.
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 shown in the figure in Matsuki's paper (2022):



So 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 the reservoir, resulting in the reservoir states x_t-2, x_t-1, ... which are then concatenated and fed to the readout neuron to take the best action, but as I'm now using a visual environment, the observations of the environment (o_t-2, o_t-1, ...) are first pre-processed using a CNN and then fed into the reservoir and replay memory.
Unfortunately, this model setup was not able to learn to play the game and I was only able to successfully train the original model proposed by Mnih et al. (2013), and at this point I have to agree with Matsuki's (2022) observation that using high-dimensional inputs such as images with reservoirs remains a difficult challenge, and I can understand the reasoning behind choosing less complex environments first to show that the general approach is (potentially) sound. Below is the training progress of the model originally proposed by Mnih et al. (2013), and I stopped training when it was clear that it was working as intended:



As the plot shows, training was stopped at ~1.2 million frames, whereas "good" results are typically achieved at ~10 million frames, and the original model was trained on ~50 million frames. The definition of "good" is rather vague here, but human players are said to have an average score of ~31.0 (Mnih et al., 2015), and half of that was achieved after only 1 million frames, so it is safe to assume that it would outperform most humans at that point. The final model was able to achieve an average score of ~400.0 (Mnih et al., 2015).

While my reservoir computing variant is more efficient and faster because it only trains the readout neuron, uses smaller states, and has a larger replay memory due to the smaller state size, it was barely able to outperform a random model, which has an average score of 1.7 (Mnih et al., 2015). The original paper has states of size 4x84x84 for 4 stacked greyscale images of 84x84 pixels versus a 1024-dimensional vector for the RC variant, where the downsampled game image via the CNN is a 512-dimensional vector and the memory output is also a 512-dimensional vector, which are ultimately concatenated and thus significantly smaller than the states in the original paper.

In total, I spent more than 300 hours training different versions of the RC variant, but never got satisfying results, but the general idea is sound, as the game can be solved using Q-learning and a neural network proven by Mnih et al. (2013), and I suspect, similar to what Matsuki (2022) predicted, that the downsampling method (via the untrained CNN) is responsible for the poor performance, so a different and more sophisticated approach may lead to better results. The reason why a similar setup might have worked for Chang and Futagami (2019) may be that their chosen environment has rich and redundant information, so that even when projecting into a lower dimensional space using an untrained CNN, enough information remains for the readout neuron to make appropriate decisions, whereas in the Breakout environment, very small aspects such as the position of the ball and paddle have to be passed through the network.

Looking to the future, possible solutions could use pre-trained CNNs for feature extraction, or perhaps Principal Component Analysis (PCA) to use only the most important aspects of the image. There are almost infinite ways to approach the problem of projecting the game frame into the reservoir, but choosing the right reservoir hyperparameters is also a challenge that is often only solved by a grid search.

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