Recurrent Control Nets as Central Pattern Generators for Deep Reinforcement Learning

VINCENT LIU, ADEMI ADENIJI, NATHANIEL LEE, JASON ZHAO
STANFORD UNIVERSITY

Central Pattern Generators (CPGs) are biological neural circuits capable of producing coordinated rhythmic outputs in the absence of rhythmic input. As a result, they are responsible for most rhythmic motion in living organisms. This rhythmic control is broadly applicable to fields such as locomotive robotics and medical devices. In this paper, we explore the possibility of creating a self-sustaining CPG network for reinforcement learning (RL) that learns rhythmic motion more efficiently and across more general environments than the current Multilayer Perceptron (MLP) baseline. Recent improvements in CPG modeling introduce the Structured Control Net (SCN), which maintains a standard MLP as the nonlinear module for global control but adds a linear module for local control [12]. SCNs are able to perform well on standard RL metrics, but struggle to produce coordinated locomotion as they are unable to capture time-dependent context. Here, we show that sequential architectures such as Recurrent Neural Networks (RNNs) model CPG-like behavior more effectively. Combining previous work with RNNs and SCNs, we introduce the Recurrent Control Net (RCN), which consists of a linear module for local control and an RNN as the nonlinear module for global control. We find that RCNs match and exceed the performance of baseline MLPs and SCNs across all environment tasks, confirming existing intuitions for RNNs on locomotive tasks and demonstrating the promise of SCN-like structures in control tasks.

Introduction

The ability to model rhythmic outputs given arrhythmic inputs has significant implications in neuroscience, robotics, and medicine. Central Pattern Generator (CPG) networks are a vastly important yet little understood region of reinforcement learning (RL). Existing baselines for locomotive tasks fail to explicitly provide rhythmic intuitions to RL agents.

In RL, the agent makes decisions based on a policy, which is a mapping from states to actions. During training, the model learns the mapping that will maximize reward. The standard policy network used when modeling locomotive tasks is the Multilayer Perceptron (MLP) [7, 9, 10]. OpenAI uses an MLP of two hidden layers and 64 hidden units as a baseline (MLP-64). This simple neural network learns to model tasks of moderate complexity, but fails to generate rhythmic output without rhythmic input [12].

While the MLP models global control relatively well, recent work has shown that explicitly modeling local control in addition to global control improves performance [5, 12]. [12] introduces the Structured Control Net, which outperforms the standard MLP across many environments. The SCN architecture is comprised of a linear module for local control and a nonlinear module for global control (in [12], an MLP-16), the outputs of which sum to produce the policy action. We use this baseline (SCN-16) for our benchmarks.

Previous work has explored the application of Recurrent Neural Networks (RNNs) to the central pattern generation task [15, 16]. RNNs maintain a hidden state that is updated at each timestep and allows RNNs to produce context-informed predictions based on previous inputs [6]. The vanilla RNN allows inputs and previous hidden states to flow between time-states freely and can provide more context to the action policy than the MLP and SCN can,

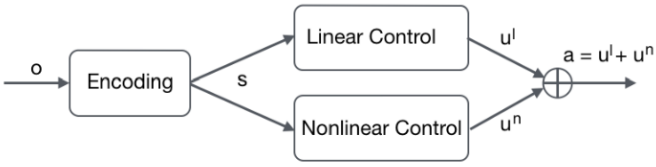

Figure 1. Structured Control Net architecture, adapted from [12].

since both only have access to information at the current timestep. This RNN architecture has many limitations, including loss of information over long time sequences, vanishing and exploding gradients, and complexity of parallelization [6]. We also explore the efficacy of variations to the vanilla RNN intended to mitigate these shortcomings, including Long Short-Term Memory (LSTM) [3, 8] and Gated Recurrent Units (GRU) [17], and provide results on the detrimental effect of increased RNN complexity on our RL environments.

In this paper, we combine the intuition behind SCNs and RNNs to exploit the advantages of modeling global and local control and of invoking global context at each timestep. We adopt the separation of linear and nonlinear modules from [12], which has been shown to improve performance by learning local and global interactions. We also adopt the vanilla RNN as our nonlinear module, which models global interactions more effectively than a MLP. We experimentally demonstrate that this architecture brings together the benefits of both linear, nonlinear, and recurrent policies by improving training sampling efficiency, final episodic reward, and generalization of learned policy, while learning to generate actions based on prior observations. We further validate our architecture with competitive results on simulations from OpenAI MuJoCo,

trained with the Evolution Strategies (ES) optimization algorithm [9, 13, 4].

Related Work

MLPs have previously been used to attempt modeling of rhythmic control tasks. The intuition is that the nonlinear fully-connected architecture is an effective function approximation. Although MLPs can generate high episodic rewards on many MuJoCo tasks, they often converge to locomotive behaviors that are jerky and unintuitive to motion.

Structured Control Nets

[12] demonstrates that enhancements can be made to the simple MLP model and yield boosts in performance across many environments. The SCN architecture learns local and global control separately. To model local interactions, the linear module is simply a linear mapping from the observation space to the action space. To model global interactions, the nonlinear module is a MLP, comprised of linear mappings as well as nonlinearities, giving it the ability to learn more complex interactions.

These interactions are specific to locomotion tasks: the agent needs to learn global patterns but also local interactions and movements specific to a task. Intuitively, the explicit modeling of local control is helpful because locomotive actions tend to depend on immediate prior actions. Although the SCN does not model cyclic actions, as it produces outputs given the current observations only, learning local interactions can provide more informative context than strictly learning global interactions. Therefore, we leverage the principle of separate control modules in our architecture

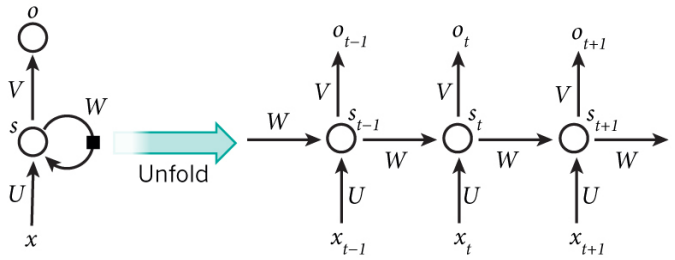

Figure 2. Recurrent Neural Network architecture

Recurrent Neural Networks

RNNs model time sequences well by maintaining a hidden state as a function of priors [6]. Traditionally, RNNs have been intensively used in natural language processing to model sequence prediction by including previous context. Therefore, to leverage the ability to condition current observations on past ones, RNNs have also been explored loosely in RL for quadruped environments. However, RNNs have not been generalized to general locomotive tasks and still remain relatively specific to quadruped-based locomotion tasks [16, 15]. While RNNs are inherently subject to a number of gradient and memory problems, there have been a number of modifications to the vanilla RNN architecture to address these problems. We explore some gated RNN variations in our experiments, but empirically opt for the vanilla architecture and will explain the intuition behind doing so later in this paper. (See Appendix A for a comprehensive description of effective RNN architectures.)

Experimental Setup

To work with locomotive tasks, we used OpenAI Gym [2], a simulated physics environment for RL, and ran our models on Multi-Joint dynamics with Contact (MuJoCo) tasks [14]. We used the MuJoCo '-v2' environments, which were the latest versions at the time of our experiments.

The Gym environment effectively serves as a wrapper to the MuJoCo tasks. At each timestep, the Gym environment returns an observation, which encodes the agent's state (i.e. joint angles, joint velocities, and environment state). The policy takes this observation as input and outputs an action to be executed by the agent. In cyclic fashion, the environment returns the action, reward, and subsequent observation to the policy. Over many episodes and timesteps, the policy learns how to traverse the environment by maximizing the rewards of its actions.

Evolution Strategies

We used a population size per iteration of 20, a sigma noise of 0.1, and a learning rate of 0.01. We annealed the learning rate constantly throughout training with a decay factor of 0.999. We also used an epsilon-greedy policy with an initial random exploration rate of 1. We linearly annealed this probability to 0 over 1 million timesteps.

Recurrent Control Net

We built upon the concept of separate linear and nonlinear modules from [12] and designed our Recurrent Control Net (RCN) in a similar fashion. Our linear module is identical to that of the SCN [12], but our nonlinear module is a standard vanilla RNN with hidden size 32. Intuitively, the linear module provides local control while the nonlinear module provides global control. However, unlike the MLP used in SCN-16 [12], the RNN learns global control with access to prior information encoded in its hidden state. We used this architecture (RCN-32) as our baseline in experiments.

In our experiments, we compared the RCN-32 to a vanilla RNN of hidden size 32 (RNN-32) to test the efficacy of the extra linear module. To reduce the number of trainable parameters for ES, we removed all bias vectors in all models.

Evaluation

We used OpenAI's MLP-64 model and the SCN-16 outlined in [12] as baselines for experimental comparison. We evaluated a model's efficacy by its final reward after 10 million timesteps rather than the rate of convergence. Across all MuJoCo environments, we find that the RNN-32 matches or exceeds both baselines (see Figure 3). We also notice that the RCN-32 consistently improves upon the RNN-32. We only show the results across the Walker2d, Swimmer, Humanoid, and Half-Cheetah environments as they best represent locomotive tasks (as opposed to Humanoid-Standup and Reacher, for example) and provide the most interesting training curves across all models (as opposed to the Ant environment, in which all models converge to negative rewards).

From our experimentation with various recurrent structures, we make several interesting observations. The recurrent structure seems to be inherently conducive to modeling locomotive tasks because its hidden state explicitly encodes past observations, whereas an MLP does so implicitly. We desire this explicit encoding because it facilitates learning of patterns in sequential observations. We also found that the increase in model complexity past a certain threshold is detrimental to ES's randomized training process. Additionally, explicit modeling of linear and global interactions

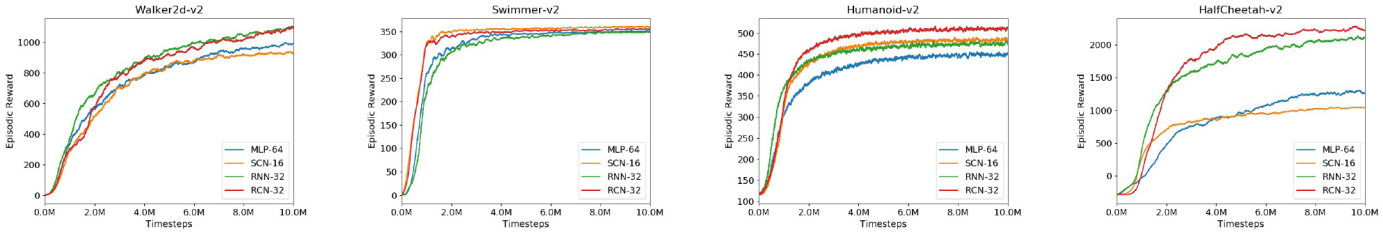

Figure 3. Episodic rewards for MuJoCo environments on baselines MLP-64, SCN-16, RNN-32, and RCN-32 using ES optimization. Average of 5 median trials from 10 total trials.

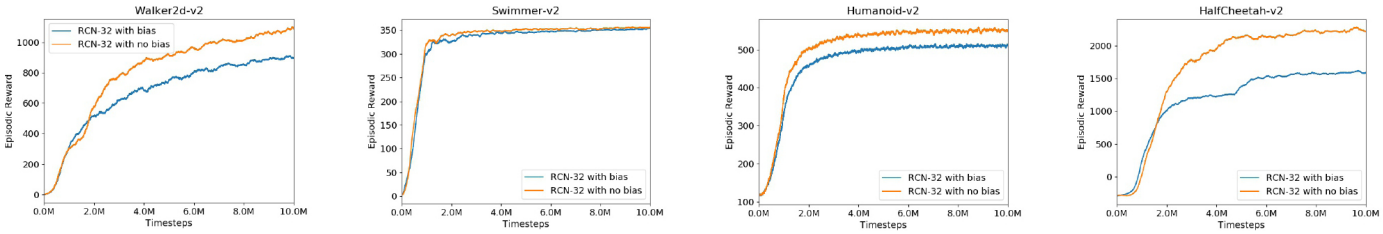

Figure 4. Episodic rewards on MuJoCo environments with RCN-32, with and without biases using ES optimization. Average of 5 median trials from 10 total trials

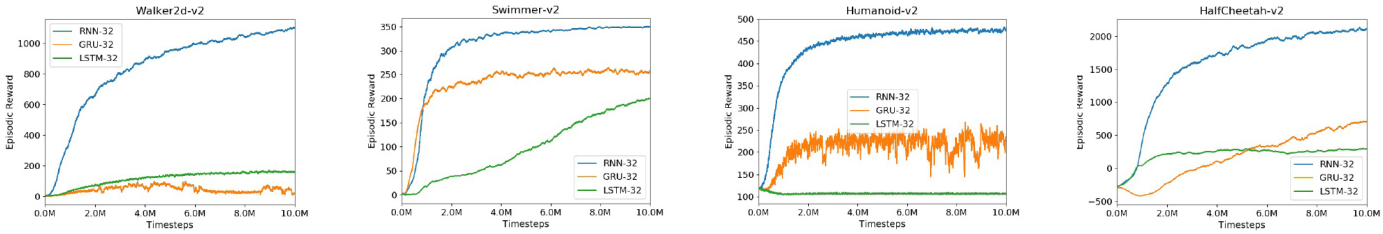

Figure 4. Episodic rewards on MuJoCo environments with RNN-32, GRU-32, and LSTM-32 using ES optimization. Average of 5 median trials from 10 total trials.

with linear and nonlinear modules consistently improves model performance.

Gated Information Flow

In all our trials with ES optimization, we noticed that recurrent architectures with gated information flow (GRUs, LSTMs) struggled in training (see Figure 5). We believe that since ES is a random optimizer, it struggles to optimize models with more parameters. A more complex model introduces more local optima, which may cause ES to converge prematurely. Additionally, since MuJoCo tasks are relatively simple and low-dimensional, enhanced memory is unnecessary and the learning of gates in training only burdens the optimization process.

The ES algorithm is inherently hampered by its gradient-free approach. Because it updates weights with random noise, models with more parameters are subject to higher overall noise variance per iteration. This can cause complex models to fail to converge entirely (see Figure 5). However, with simple architectures, we see early convergence in episodic reward (compared to the same models trained with different algorithms). As such, we anticipate that GRUs and LSTMs may achieve higher rewards with an optimization algorithm like PPO (Proximal Policy Optimization, a gradient-based optimizer [11]), where extra parameters from information gates are not heavily penalized.

Linear Control

The RCN-32 consistently outperformed the RNN-32. This finding is consistent with [12], which shows the efficacy of introducing a linear component in addition to the nonlinear component (see Figure 3). This tiered approach accounts both for immediate information provided per observation and for longer-term patterns. As we have mentioned before, individual actions in locomotive tasks are heavily conditioned on immediate observations. The separate linear module allows for a larger emphasis on local information.

Local control is balanced by the RNN, which is responsible for global control. The hidden state is a complex series of nonlinear mappings of past observations, which gives the RNN access to global information. The addition of the linear module to the nonlinear module allows the entire architecture to learn local and global interactions. While the increase in performance is sometimes marginal, the SCN-16 similarly improved the MLP-64.

Incorporating Biases

We also experimented with incorporating biases into the RCN. Doing so immediately decreased performance across all tasks, sometimes even below baseline performances (see Figure 4). We believe that this is because the inclusion of biases burdened the optimizer in training without providing any real value to what the model learns. Just as the gated RNN variations struggled in training, adding parameters to the RCN resulted in higher noise variance per ES iteration. Another possible explanation is the simplicity of the MuJoCo environments, which may not require the additional bias vector to successfully model the task.

Conclusion

We conclude that RNNs model locomotive tasks effectively. Furthermore, we conclude that the separation of linear and nonlinear control modules improves performance. The RCN combines the benefits of both concepts, learning local and global control and patterns from prior sequential inputs. We also note the detriment of increasing model complexity with information gates, though this is probably due to MuJoCo task simplicity and the ES training algorithm. Because ES updates weights randomly, additional gates create more local optima that ES has to overcome.

Since our models have only been trained with the ES algorithm, future investigations would involve exploring the performance of RCNs with an algorithm such as PPO. Additionally, recent practices in natural language processing have successfully replacing recurrent layers with convolutional layers [1]. It would be interesting to explore whether convolution could replace the RNN module for sequential modeling. We hope that our findings open up further investigation into the usage of RCNs for these applications.

Appendix A: Recurrent Architectures

For more context, this section covers in-depth the fundamental recurrent architectures upon which we built our models: Recurrent Neural Networks (RNNs), Gated Recurrent Units (GRUs) and Long Short-Term Memories (LSTMs).

Recurrent Neural Network

The vanilla RNN maintains an internal hidden state to compute future actions, which serves as a memory of past observations. This simple architecture allows all inputs and hidden states to flow freely between timesteps. Standard RNN update equations are below.

$$h^{(t)} = \tanh(W_h h^{(t-1)} + W_x x^{(t)} + b_h)$$
$$o^{(t)} = W_o h^{(t)} + b_o$$

Equation 1: Vanilla Recurrent Neural Network update equations, where h(t), o(t), x(t) denote the hidden state, output (action), and input (observation) vectors, respectively, at timestep t.

Gated Recurrent Unit

A GRU improves upon the vanilla RNN by learning to retain context for the next action by controlling the exchange of inputs and previous hidden states between timesteps [17]. GRUs have a reset gate r after the previous activation to forget part of the previous state and an update gate u decides how much of the next activation to use for updating.

$$u^{(t)} = \sigma(W_{u,x} x^{(t)} + W_{u,h} h^{(t-1)} + b_u)$$
$$r^{(t)} = \sigma(W_{r,x} x^{(t)} + W_{r,h} h^{(t-1)} + b_r)$$
$$h^{(t)} = u^{(t)} \circ h^{(t-1)} + (\vec{1} - u^{(t)}) \circ \tanh(W_{h,x} x^{(t)} + W_{h,h} h^{(t-1)} + b_h)$$
$$o^{(t)} = W_{o,h} h^{(t)} + b_o$$

Equation 2: Gated Recurrent Unit update equations, where h(t), c(t, and x(t) denote the hidden state, cell state, and input (observation) vectors, respectively, at timestep t. The output is produced with a linear mapping of h(t) to the output (action) vector.

Long Short-term Memory

An LSTM learns a "memory" of important locomotion context via input, forget, and output gates [3, 8]. The input gate i regulates how much of the new cell state to keep, the forget gate f regulates how much of the existing memory to forget, and the output gate o regulates how much of the cell state should be exposed to the next layers of the network.

$$f^{(t)} = \sigma(W_{f,x} x^{(t)} + W_{f,h} h^{(t-1)} + b_f)$$
$$i^{(t)} = \sigma(W_{i,x} x^{(t)} + W_{i,h} h^{(t-1)} + b_i)$$
$$o^{(t)} = \sigma(W_{o,x} x^{(t)} + W_{o,h} h^{(t-1)} + b_o)$$
$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tanh(W_{c,x} x^{(t)} + W_{c,h} h^{(t-1)} + b_c)$$
$$h^{(t)} = o^{(t)} \circ \sigma(c^{(t)})$$

Equation 3: Long Short-Term Memory update equations, where h(t), c(t), and x(t) denote the hidden state, cell state, and input (observation) vectors, respectively, at timestep t. The output is produced with a linear mapping of h(t) to the output (action) vector.

Acknowledgements

This research on modeling central pattern generators originally began as a class project for CS 229, Machine Learning. We thank our TA advisor, Mario Srouji, for giving us insight on his Structured Control Net [12].

References

[1] Eric Battenberg, Jitong Chen, Rewon Child, Adam Coates, Yashesh Gaur Yi Li, Hairong Liu, Sanjeev Satheesh, Anuroop Sriram, and Zhenyao Zhu. Exploring neural transducers for end-to-end speech recognition. In Automatic Speech Recognition and Understanding Workshop (ASRU), 2017 IEEE, pages 206– 213. IEEE, 2017.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. arXiv preprint arXiv:1606.01540, 2016.

[3] Yang-Hun Cha, Dang-Yang Lee, and In-Kwan Lee. Path prediction using lstm network for redirected walking. In 2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR), pages 527–528. IEEE, 2018.

[4] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In Advances in Neural Information Processing Systems, pages 5032–5043, 2018.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[6] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. arXiv preprint arXiv:1506.00019, 2015.

[7] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In International conference on machine learning, pages 1928– 1937, 2016.

[8] Francisco Javier Ordóñez and Daniel Roggen. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. Sensors, 16(1):115, 2016.

[9] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864, 2017.

[10] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In International Conference on Machine Learning, pages 1889–1897, 2015.

[11] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[12] Mario Srouji, Jian Zhang, and Ruslan Salakhutdinov. Structured control nets for deep reinforcement learning. arXiv preprint arXiv:1802.08311, 2018.

[13] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. arXiv preprint arXiv:1712.06567, 2017.

[14] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pages 5026–5033. IEEE, 2012.

[15] Duc Trong Tran, Ig Mo Koo, Yoon Haeng Lee, Hyungpil Moon, Sangdeok Park, Ja Choon Koo, and Hyouk Ryeol Choi. Central pattern generator based reflexive control of quadruped walking robots using a recurrent neural network. Robotics and Autonomous Systems, 62(10):1497–1516, 2014.

[16] Duc Trong Tran, Ig Moo Koo, Gia Loc Vo, Segon Roh, Sangdeok Park, Hyungpil Moon, and Hyouk Ryeol Choi. A new method in modeling central pattern generators to control quadruped walking robots. In Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on, pages 129–134. IEEE, 2009.

[17] Yuan Wang, Kirubakaran Velswamy, and Biao Huang. A novel approach to feedback control with deep reinforcement learning. IFAC-PapersOnLine, 51(18):31– 36, 2018.