Learning an Effective Control Policy for a Robotic Drumstick via Self-Supervision

Demonstration

Mason Bretan
Samsung Research America
Mountain View, CA
mason.bretan@samsung.com

Siddharth Sanan
Samsung Research America
Mountain View, CA
s.sanan@samsung.com

Larry Heck
Samsung Research America
Mountain View, CA
larry.h@samsung.com

ABSTRACT

We train a neural network to control a drumstick fastened to a motor. The network takes a temporally arranged sequence of desired strikes, or a rhythm, as input and outputs a sequence of motor velocities controlling the drumstick's physical movement. We use a new method of training, we call *Collaborative Network Training*, in which three networks work together to directly minimize a non-differentiable loss function. In this work, the goal is to minimize the difference between the input sequence and the resulting drumstick strikes on a surface produced by the network outputs. The resulting policy learned by the network works in real-time and has a precision of 10 milliseconds.

KEYWORDS

Robot learning; robot controls

ACM Reference Format:
Mason Bretan, Siddharth Sanan, and Larry Heck. 2019. Learning an Effective Control Policy for a Robotic Drumstick via Self-Supervision. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 3 pages.

1 INTRODUCTION

We use a new method of training neural networks with aims of enabling task objective functions that are not directly differentiable w.r.t. the network output and learning parameters when a process for measuring performance is available, but labeled data is unavailable. Our proposed method allows for direct optimization for achieving a desired task and is particularly suitable for generating continuous-space actions. We demonstrate the efficacy of the technique using a controls task in which a drumstick's movements, controlled by a motor (MX-28 dynamixel), are directed by the outputs of the network. We develop an interactive scenario in which a person provides a rhythmic sequence, the system listens, and the network produces a sequence of actions enabling the robot to mimic the person (see **https://youtu.be/afvHaOw1_EQ**).

The method utilizes a trio of networks and a ranking function that sorts each network according to the performance. This ranking provides a signal for how to update each network's weights using gradient descent. The objective is similar to reinforcement learning (RL) in that network parameters are optimized directly for a task [2, 4]. For a continuous state space the actor-critic method, Deep Deterministic Policy Gradient (DDPG), was introduced in [1]. Unlike DDPG, our method does learn the parameters of a critic network in parallel to the actor, but minimizes a reward function directly by have three networks work collaboratively.

Similarly, ensemble learning typically requires multiple learners that are trained to solve the same problem. During inference each model contributes to a collection of hypotheses which are used to make the final prediction [5]. Our method is different from such methods in that the networks rely on a signal describing their own performance relative to the others, thus, training one another without labeled data

2 COLLABORATIVE NETWORK TRAINING

Collaborative training relies on the ability to describe the outputs of each network as being simply better or worse than the outputs of the other networks within the group. The main premise for collaborative training is that if each network is updated in order to behave more similarly to better performing networks for a single input example, then over time and across many examples the collection will achieve a local optimum for the task. In order to prevent poor convergence the concept of "exploration" is introduced by using reflection.

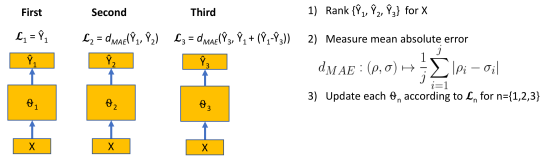

Figure 1: The collaborative training procedure. Three networks are ranked on their performance for a given input, x, and updated according to losses determined by their rankings. In this example, the Second ranked network is optimized to produce output closer to that of the First network, the First network remains unchanged, and the Third network is optimized to move away in a new "exploratory" direction informed by the First.

The procedure, summarized in Figure 1, works as follows: $\mathbb{N}$ is the set containing $\{1, 2, 3, ..., N\}$ where N is the total number of networks. The parameters of each network, θ_n, are randomly initialized for $n \in \mathbb{N}$. In all of our experiments each network within an ensemble have identical architectures. Additionally, we use $N = 3$ and preliminary testing did not find an increase in performance for higher N (though this needs further validation).

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

For a given training input, $x \in X$, where X is the set of all training inputs, the outputs of each network, $\hat{y}_n$, are sorted according to a ranking function $f : (x, \hat{y}_n) \mapsto \mu_n$ where μ_n describes the performance of $\hat{y}_n$ given x. Thus, the best performing network is θ_{w_1} where $w_1 = \arg\max_n(\mu_n)$ and the worst performing network is θ_{w_N} where $w_N = \arg\min_n(\mu_n)$.

The loss for a network in the trio, $\mathfrak{L}_n$, is determined by its ranking. For each of the non-worst networks, $\mathbb{N} \setminus w_N$, the loss is found by computing d_{MAE} between adjacently ranked networks That is,

$$\mathfrak{L}_{w_{n \in \mathbb{N} \setminus w_1}} = d_{MAE}(\hat{y}_{w_n}, \hat{y}_{w_{n-1}}) \quad (1)$$

where $\mathfrak{L}_{w_n}$ is differentiable w.r.t to θ_{w_n}. The goal is for each network to be trained so that a given network, θ_{w_n}, produces an output for x that more closely resembles an output from a better performing network $\theta_{w_{n-1}}$.

Finally, the loss for the worst performing network, $\mathfrak{L}_{w_N}$ is computed using

$$\mathfrak{L}_{w_N} = d_{MAE}(\hat{y}_{w_N}, \hat{y}_{w_1} + (\hat{y}_{w_1} - \hat{y}_{w_N})) \quad (2)$$

To summarize, for a given input, the best performing network remains unchanged, the second best network is updated in the direction of the best performing network, and the worst performing network is updated in the direction described by the reflection of the worst past the best.

3 DRUMSTICK TRAJECTORY GENERATION

In this system we train a network to generate a sequence of motor velocities resulting in the production of drumstick trajectories allowing it to strike the drum in a rhythmically controlled manner. We use a custom ranking function that incorporates a method to measure the rhythmic similarity between the resulting drum strikes and the input onset sequence as well as a method to measure "goodness" when the drumstick doesn't come into contact with the drum at all producing no onsets. To measure rhythmic similarity we compute the cosine distance between the input vector and the resulting onset vector generated from the drumstick movements. To evaluate the trajectory we measure the euclidean distance between the tip of the drumstick and two constant values representing either the ideal "rest" position or ideal "strike" position. Therefore, we assume prior knowledge of the positions of the stick and drum in the environment. This allows us to train the models in simulation by replicating the real-world setup.

Another step to better ensure adequate transfer from simulation to real-world is to use a symbolic representation of the desired onset sequence rather than the raw audio signal. In doing this the actual audio does not need to be simulated. Therefore, the input to the networks are a sequence of zeros and ones. Each value in the input vector represents an event a specific time where the temporal resolution is 10ms. We provide the networks with a sequence of 20 events representing 200ms. In the vector a one indicates an onset and zeros indicate no onsets at that time. Additionally, the starting angle of the joint is included in the input vector (see Figure 2).

Frequently a network will produce actions that are not possible. The safety mechanisms within the simulation recognizes illegal actions and prevents them. In the scenario where an action would need to exceed the velocity threshold to successfully reach the target

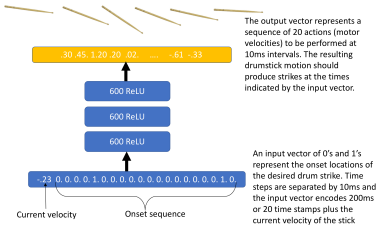

Figure 2: The network is trained to produce a sequence of actions controlling the drumstick's trajectory. The resulting trajectory should produce a rhythm that is identical to the sequence provided by the input vector.

position, the simulation moves as far as it can in the same direction at the thresholded rate. In the scenario where an action would require the robot to pass through the drum the simulation ignores the action all together and does nothing. Because of this behavior, the sequence of performed velocities are likely to be different than the sequence of generated joint angles. This is particularly true during the early stages of training. In order to encourage the networks to learn to generate legal actions instead of using the *generated* joint action sequence from the best and worst performing networks, $\hat{y}_{w_1}$ and $\hat{y}_{w_3}$, we use the *effected* joint action sequence, $e(\hat{y}_{w_1})$ and $e(\hat{y}_{w_1})$, where $e(x)$ is a function representing the simulation and built-in safety constraints.

Results We evaluated the system in both simulation and the real-world. During real-world inference we wanted a user to be able to provide input by drumming. To do this, the drumming audio is recorded and a spectral difference-based onset detection method ([3]) is used to create the input sequence necessary for the network (see Figure 3). This pre-processing audio analysis step introduces the potential for error by either missing onsets or producing false positives. Though, the method is relatively robust achieving roughly 90% accuracy in the controlled environment in which we performed the experiment.

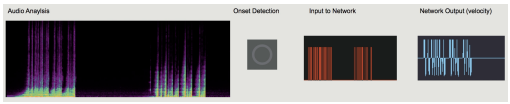

Figure 3: The processing pipeline of the real-time system includes an audio stream which is analyzed for drum onsets. The onset sequence is provided as input to the network which then produces a sequence of motor velocities.

For the real-world evaluation 3 minutes worth of drumming was recorded from a professional drummer and analyzed producing 852 onsets. The resulting onset sequence was then fed into the network (at 200ms intervals) resulting in an approximate imitation of the human drummer. Onset detection was performed on the resulting audio and cosine similarity was measured between the human and robot drummer's onsets. The final cosine similarity for simulation was **1.0** and for real-world was **.86**. The real-world scenario performance declined, but the results are still convincing and much of the decline was likely due to the onset detection.

REFERENCES

[1] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[3] Miller S Puckette, Miller S Puckette Ucsd, Theodore Apel, et al. 1998. Real-time audio analysis tools for Pd and MSP. (1998).

[4] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.

[5] Zhi-Hua Zhou. 2015. Ensemble learning. *Encyclopedia of biometrics* (2015), 411–416.