Towards adversarial training for mobile robots

Todd Flyr[1] and Simon Parsons[2]

[1] Department of Computer Science, Graduate Center, City University of New York
E-mail: tflyr@gradcenter.cuny.edu
[2] Department of Informatics, King's College London
E-mail: simon.parsons@kcl.ac.uk

Abstract. This paper reports some preliminary work on learning on a physical robot. In particular, we report on an experiment to learn how to strike a ball to hit a target on the ground. We compare learning based just on previous trials with the robot with learning based on those trials plus additional data learnt using a generative adversarial network (GAN). We find that the additional data generated by the GAN improves the performance of the robot.

1 Introduction

While machine learning as a field has advanced considerably in recent years [9], there has yet been relatively little work using neural networks with mobile robots. A significant reason for this is the fundamental difficulty of obtaining a sufficiently large dataset to train a neural network for use in such devices. Traditional techniques for training a neural network require a large corpus of labeled training data [4, 5]. Obtaining such a corpus generally requires thousands to millions of trials and the additional human labor for labeling them. For many machine learning tasks, such as image or speech recognition, large labeled datasets are collaboratively created by various labs which are then shared online for use by anyone wishing to train on them [5].

In mobile robotics, the use of large datasets is a challenge. It is difficult to run a mobile robot in a complex task for enough trials to create such a dataset. In addition, each trial can take a long time to complete. Devices embedded in the real world must deal with a level of uncertainty that is difficult to recreate without the experience of many real world trials. Lastly, mobile robots vary considerably in their design and components, and the tasks they can complete also vary greatly. Thus developing large collaborative datasets has not proved to be practical.

Recent developments in adversarial learning offer a way to solve this problem. Generative Adversarial Networks (GANs) [6], originally introduced for predictive learning and generation of image and video data, can be configured to rely on a much smaller corpus of labeled data than is normally needed for other artificial neural network (ANN) training methods. This paper begins to explore how GANs might be used in the training of a controller for a mobile robot when only a small training dataset is available.

2 Related work

Research into robot learning is extensive. Here we summarise only the most related elements. In [3] Bongard suggests that there are three aspects of adaptive evolution, the last, of which, *prospection*, is most relevant here. Prospection is the ability of an organism or robot to predict future actions, in particular the ability to mentally simulate the actions of future events that have never been previously encountered. Thus a robot equipped with prospection can then adapt to novel situations and can simulate itself to try to understand itself in more detail. Such a robot could rehearse novel actions that can enable it to recover in the face of degraded ability or partial injury or failure. A form of prospection is the ultimate aim of our work.

Pinto *et al.* [11] trained a convolutional neural network (CNN) on image data. However, instead of training images to labels, they trained the images to physical interactions with the robot. Specifically, they record grasping data, pushing data, poke data, and lastly what they label "identity vision", which is a variant of active vision [12]. This method for comprehending objects without labels is proposed as a form of unsupervised learning wherein a robot could be given objects and follow a routine of physical interactions with them to learn what they were in a more intimate way than images. Our work shares the underlying use of neural network methods and the combination of vision data and physical action.

In addition to a recurrent neural network, [8] use a GAN framework to improve on trajectory prediction models for navigating in a crowd with a reduced computational cost. Like the work reported here, [8] shows what can be achieved with a GAN architecture in a learning and simulation environment. Unlike this work, it does not use a physical robot.

Somewhat similar to [11] is the work of [10]. This work developed a method for training a large CNN combined with an autoencoder to learn from first person videos of humans doing a task that involves manipulating objects with their hands. Learning from a sequence of frames allows the network to predict the next frames (future regression) for up to 1-2 seconds. The robot equipped with the neural network can learn to clear space on a table to allow a person carrying a box towards the table to set down the box. The future prediction mechanism from [10] has subsequently been applied to a similar scenario to that in [11]. Here we have a robot engaging in learning in self simulation via a mechanism similar to the proposal in [3]. In [10] a turtlebot is trained to recognize and approach (or avoid) an object in a room by using a process described as a "dreaming model". The robot wanders randomly in the space taking pictures. Then a variational autoencoder with the prediction mechanism from [10] is used to produce sequences of images that are realistic for a path to the object, and a reinforcement learning algorithm learns the correct policies on them. This is done without any real-world trials, and so represents a form of imagining how to achieve a simple goal.

3 Background

A key component of the approach that we use in this paper is a Generative Adversarial Network. GANs are a relatively recent approach to machine learning [6]. They involve one network that is trained to deceive another network by mimicking a dataset:

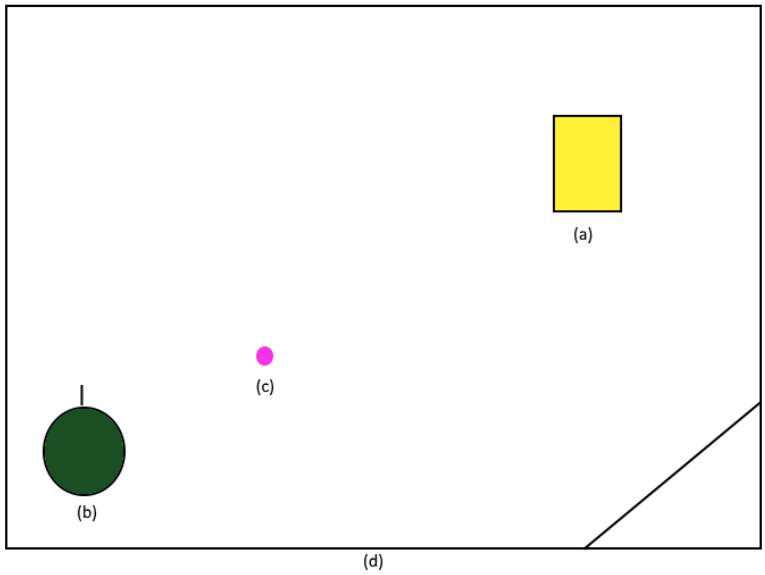
> "We simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake." [7]

In other words, GANs can be a means of generating data that mimics a dataset. Once the GAN is trained, the discriminator cannot tell the difference between real trial data and data from the generator. This means that GANs can potentially be used to increase the amount of data available for training another machine learning method. When that method is a neural network, which (as mentioned above) are typically data hungry, GANs can be used to boost a small dataset, that is unable to do a good job of training a neural network, into a larger dataset that is able to train a neural network well. That is exactly how we used a GAN in the work described here. In particular, we use a Wasserstein GAN (WGAN) [2]. WGANs are a variant of the original GAN, specifically designed to minimize "mode collapse" in which G repeatedly generates the same data accepted by D.

4 Experimental setup

4.1 Overview

A Turtlebot was modified to strike a ball on the ground at a target inside an arena. This is intended to mimic the actions of a simple golf putt. If the hit ball reaches the target, then it is considered a successful shot or score. The overall aim of the work is for a robot to be able to learn a strategy for hitting the ball, compute a set of movements to line up the shot, hit the ball, and score. The experiment is to determine whether the robot will be able to learn this task with only a small number of trials, using adversarial learning techniques to create additional training data. The work reported here involved three sets of experimental trials. In the first set of trials, the robot was controlled by a simple rule-based system. This was intended to both provide a baseline set of results against which robot performance could be compared, and to provide some initial data for learning experiments. In the second set of trials, the robot was controlled by a feed-forward neural network trained on the data from the rule-based trials. In the third set of trials, the robot was under the control of a neural network learnt from a combination of the data generated in the rule-based trials, and additional data generated by a GAN which was itself trained using data from the rule-based trials.



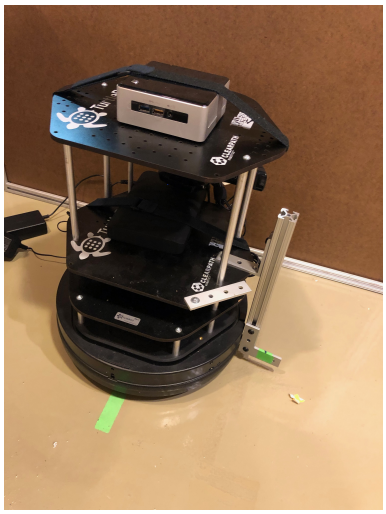


Fig. 1: The experimental setup. At the top is a diagrammatic view of the area *(a)* The target—a piece of colored paper. *(b)* The turtlebot with the putter head attached to the side. *(c)* The ball is placed in an initial position. The robot must locate the ball and the target and then properly position itself to line up the shot. *(d)* The arena walls. On the bottom left is a photograph of the arena. On the bottom right is a photograph of the modified Turtlebot.

4.2 Physical Setup

A Turtlebot 2 (Kobuki base) running the Robot Operating System (ROS) was placed in a mapped fixed-sized arena of 2×3 meters. The Turtlebot 2 comes equipped with a mounted Microsoft Kinect RGB-D sensor for 3D depth mapped vision/pointcloud generation. Attached to the left side of the Turtlebot is a small putter constructed from 80/20 aluminium. This putter has an approximately 6 cm head that is about 1 cm above the floor (see Figure 1, bottom right). This putter is fixed to the robot and cannot swing independently of the robot base. There are no additional sensors for the putter. A colored sheet of paper representing a target is placed on the floor within view of the robot. A colored ball (pink, about 7 cm diameter) is also placed on the floor within view of the robot. The robot is running the ROS turtlebot navigation stack. It is also running the ROS package *cmvision_3d*, which performs color blob detection and combines it with depth data from the RGB-D sensor. This allows the robot to locate the ball and the target. The robot is placed in a marked, fixed initial position in the corner of the arena, as if in position to approach a putt (see Figure 1).

4.3 Data and training

As described above, data was collected during trials using a rule-based controller. This was then used to train two new controllers. One was a neural network learnt directly from data collected from the rule-based trials. The other was a neural network learnt from the same data augmented with data generated by a GAN (itself trained using data from the rule-based trials).

The data collected during the rule-based trials includes the following components. One is the target location (x, y) where x is the left/right distance and y is the depth of the arena. The coordinate for height, while available, is not needed for a flat target on the floor. The second component is the initial (x, y) position of the ball. The next component is the initial robot location and pose including the (x, y) position and a quaternion for its orientation. ROS uses quaternions for rotations as defined by its *tf* (transform) package. It is composed of (a, b, c, w) and encodes a rotation in three dimensional space [13]. All the coordinates are converted to the coordinates of the map via the ROS transform package. Thus we have a vector of:

$$\langle target(x, y), ball(x, y), robot\_initial\_pose(position(x, y), quarternion(a, b, c, w))\rangle \tag{1}$$

available for training. The robot also tracks the ball when hit and records the closest approach to the centre of the target, recording the inverse, q, of the ball being the output $(1 - (\text{min distance of ball}))$ or $(1 - d_{min})$.

The neural network learnt directly from this data was a fully connected feed forward network with two hidden layers. The network topology was $7 \times 120 \times 84 \times 3$. Note that training only considered the yaw (rotation on the z-axis) of the robot. This makes the inputs

$$\langle target(x, y), ball(x, y), robot\_initial\_pose(position(x, y), yaw))\rangle \tag{2}$$

and the outputs $\langle position(x, y), yaw \rangle$ which are converted into the robot final pose in ROS,

$$\langle position(x, y), quarternion(a, b, c, w) \rangle \tag{3}$$

This is where the robot should be positioned in order to hit the ball to the target. The robot swing rotational distance and swing velocity are currently fixed. Once the neural network was trained, trials were carried out in which it controlled the robot by setting the target point at which the robot tried to strike the ball.

To augment the data from the rule-based trials, we used a GAN to mimic the trial data (which was then used to train a neural network with the same topology as before). The aim in doing so is that the discriminator, D, trains the generator to only generate input parameters and output trajectories that make sense in the context of the actual training corpus, and thus can eliminate nonsensical examples such as the ball going in the opposite direction from where it was hit. To train a GAN G (specifically a WGAN [2]), all of the data from the rule-based trials becomes outputs, as G is trying to simulate the entire system, and generators only accept noise as inputs by definition, thus seven inputs plus three outputs equals ten. The critic, D, accepts G's outputs and only outputs whether or not G looks like a genuine trial run.

We used a WGAN because it helps prevent mode collapse, where the GAN output would have zero variability in the generated trials. The GAN we used is a modification of the WGAN code from [1]. Those scripts are written in PyTorch, a python DSL (Domain Specific Language) built on the Torch libraries for machine learning. A separate script takes the dumped YAML from the ROS python scripts and creates and serializes a dataset for use by the PyTorch WGAN script. The data we used was a small corpus of 14 entries from the rule-based trials. In all of these the target was located in basically the same location. The GAN was trained to a loss on the critic of -0.000014 and on the generator to 0.010729 on a network of $8 \times 96 \times 106 \times 106 \times 106 \times 106 \times 96 \times 11$ in size for the generator and a critic of $11 \times 64 \times 74 \times 74 \times 74 \times 74 \times 64 \times 1$ over 6000 iterations of training. After some preliminary work, we added an additional output parameter to the generator to reinforce the importance of being close enough to actually hit the ball. If a ball was missed outright, it was set to zero and if it was close enough to hit, it was set to $+1$ for the trial. Hence the generator output and discriminator input size of eleven. This additional parameter is dropped when the feed-forward network (denoted MLP for "multi-layer perceptron") is trained.

4.4 Experimental procedure

The following section describes how each experimental trial was performed.

The robot, ball, and target are placed in the arena, and the robot controller uses the RGB-D sensor to locate the ball and target via the *cmvision_3d* package. It then computes a location that will position the robot next to the ball with the putter aligned hit the ball to the target. This position is chosen so that base of the robot is in a location where the putter head is perpendicular to the target. Once this position is determined, the robot sets it as a navigation

goal and moves to that location. When the navigation goal is reached, the robot rotates counterclockwise approximately a quarter turn, to prepare for the swing, and then swings the club by quickly executing a clockwise turn at a relatively high rate of rotational velocity. Upon completion of the swing the robot then quickly moves to the forward facing position in relation to the arena. After the ball is hit the robot attempts to track it via the RGB-D sensor. It then records all movement of the ball, whether it hits the target, and the closest approach to the centre of the target.

Note that ROS *tf* is used when the robot tracks the ball after it is hit (and while the robot may still be in motion) as it recovers from its swing to ensure that the ball is accurately tracked on the map. In practice this approach works quite well despite the fast rotational movement of the robot.

The key thing about the setup is that the experiments revolve around setting the location that the robot chooses to move to before beginning its "swing". This is the element that is computed by the rule-based system and the neural networks. All these calculations were based on the relative locations of target and ball, so represent general mechanisms for making this decision, although the results presented here were obtained from experiments in which the ball and target were always in the same place.

5 Experiments

5.1 Trials

As described above, we carried out three sets of trials. First, we ran 14 trials where the robot was under the control of the rule-based system. Second, we took the data recorded in these trials and used it to train a feed forward neural network (MLP) as described in Section 4.3. We then ran trials where the trained MLP determined the location to which the robot would move before attempting to hit the ball to the target. We carried out six trials, stopping when it became apparent that the robot would not approach the ball closely enough to ever strike it. (Recall that the sensors on the robot cannot detect the ball when the robot is close to it — the key skill that the robot is learning here is to pick a location from which to strike the ball from its initial observation of the ball.) Third, we used the GAN described in Section 4.3 to generate data based on the rule-based trials, and used this and the data from the rule-based trials to train another feed-forward neural network (denoted MLP/GAN). We then ran another 14 trials in which the MLP/GAN picked the location to which the robot would move.

5.2 Results

We present results from all three sets of trials. As examples of what data was collected for each trial, see Figure 2. This is data from a trial with the MLP/GAN, but the same data was collected for all trials. We have the initial and final positions of the robot (green and blue squares, respectively), the initial position of

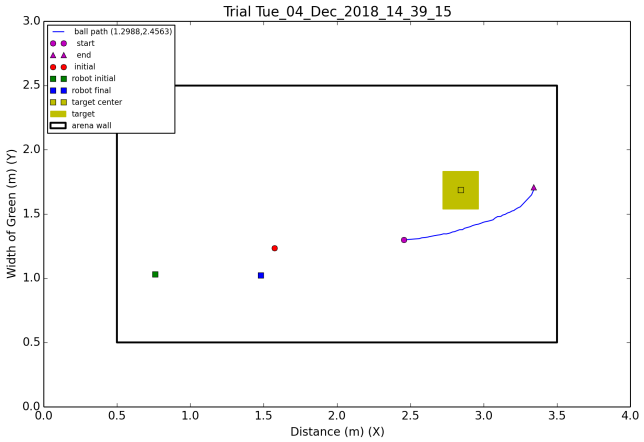


Fig. 2: A trial with the robot controlled by the MLP/GAN trained neural network. This glancing hit caused a spin resulting in the hook seen here and the closest approach was measured at 28.8 cm from the target centre.

the ball (red circle), the target position (yellow-green square), and the tracked location of the ball. Since the ball is tracked by the robot, it is not tracked right from the point at which it is hit because it is within the minimum range of the RGB-D sensor. What is recorded are the earliest point at which the moving ball is observed (magenta circle), around 1m from the robot, where the ball comes to rest (magenta triangle), and the track (blue line). From this we can compute the closest approach of the ball to the centre of the target.

Table 1 summarises the overall performance of the three controllers on the task at hand — hitting the ball to the target. Performance is measured by the closest approach of the ball to the centre of the target. The table includes data for all 14 trials of the rule-based system. In each of these trials the robot hit the ball, and, on average, hit it within 23 centimetres of the target centre. The data from the trials was subsequently used to train the MLP and MLP/GAN controllers, and the performance of the rule-based controller demonstrates that the dataset used to train the MLPs was of reasonable quality with examples of both of successful and failed shots (six hits and eight misses). The table includes data on six trials of the MLP controller. The robot hit the ball in none of these trials, so there was no "closest approach" data to collect. Since it became clear that this controller would not position the robot close enough to the ball to hit it, we abandoned these trials after the six failures. Finally, Table 1 includes data from 14 trials with the MLP/GAN controller. In nine of these the robot managed to hit the ball, and the closest approach data is provided for these trials.

Figure 4 gives plots of six trials of the MLP/GAN controller. As discussed in the caption of Figure 4, note that a bug in the ball-tracking code means that in several of the trials, the closest approach was recorded incorrectly. As

Trial	Rule-based system			MLP			MLP/GAN			
	X	Y	Euclid	X	Y	Euclid	X	Y	Euclid	Corrected
1	0.0928	0.2346	0.2523	-	-	-	0.2164	0.4787	0.5252	
2	0.0308	0.3041	0.3057	-	-	-	0.5646	0.3243	0.6444	
3	0.0047	0.2769	0.2769	-	-	-	-	-	-	
4	0.0665	0.0521	0.0845	-	-	-	-	-	-	
5	0.1738	0.4496	0.4820	-	-	-	0.3099	0.2475	0.3966	
6	0.0539	0.2509	0.2566	-	-	-	-	-	-	
7	0.1467	0.0871	0.1706				-	-	-	
8	0.0444	0.2671	0.2708				0.0145	0.0135	0.0198	0.2098
9	0.1347	0.0742	0.1538				0.089	0.2739	0.288	
10	0.1346	0.0013	0.1346				0.0767	0.1716	0.188	0.378
11	0.0253	0.1649	0.1668				0.0004	0.4972	0.4953	
12	0.0084	0.0682	0.0687				-	-	-	
13	0.2099	0.2069	0.2947				0.4892	0.4299	0.6513	
14	0.0428	0.2611	0.2646				0.5131	0.3819	0.6166	
Average	0.0835	0.1928	0.2273				0.2526	0.3132	0.4672	

Table 1: The closest approach of ball to target for each control system. Distances reported in X and Y directions and *Euclid*ian. Note that the target location bug requires occasional correction by 19 cm (the distance between the corner and the centre of the target). Items with a dash "-" indicate the robot did not get close enough to the ball to hit it. The MLP trials were abandoned after six trials when it became clear it was never going to get close enough to the ball to hit it. The average for the MLP/GAN includes the corrected values. In the rule-based system, trials 4, 7, 9–12 were recorded as hits, in general, any ball within about 18 cm of the target centre will be recorded by the robot as a hit. None of the MLP/GAN trials come that close.

Figure 4 (c) shows, this led to data initally being recorded with a smaller "closest approach" than was actually the case. Table 1 gives the corrected data.

Because the MLP-controlled robot never hit the ball, Table 1 does not provide a quantitative comparison of the performance of the MLP and MLP/GAN controllers. To provide this, Tables 2 and Table 3 provide data on how close the MLP-controlled and MLP/GAN-controlled robots approached the ball.

5.3 Analysis

Comparing the MLP and the MLP/GAN, it is clear that the MLP/GAN has better performance. Not only did the MLP/GAN manage to hit the ball in a number of its trials, unlike the MLP, the MLP/GAN also approaches the ball more closely. The average distance of the MLP controlled robot was 14 cm from the X coordinate of the ball and 25 cm from the Y coordinate while the MLP/GAN controlled robot approached within 11 cm (X) and 24 (Y) cm. The latter result is obviously correlated with the former since stopping too far away

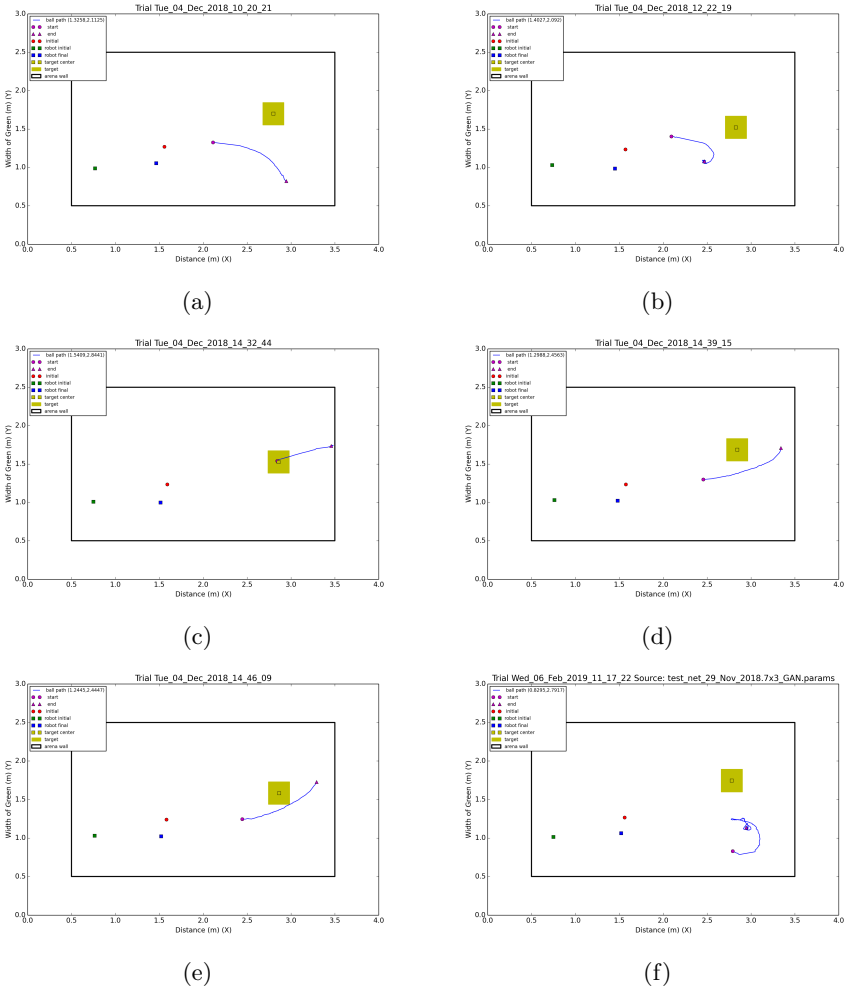


Fig. 4: Plots of putts with the robot under control of the MLP/GAN controller. Note that the robot position is consistently too far behind the ball to hit it effectively. *(a)* - A miss (Closest: 0.5252 m), *(b)* A glancing hit sent the ball spinning (Closest: 0.3966 m), *(c)* Target shift bug suggests a hit but it was not (Closest: 0.0198 m corrected in Table 3), *(d)* Close. (Closest: 0.288 m), *(e)* Close but not a hit (target shift bug) (Closest: 0.188 m corrected in Table 3), *(f)* A hard hook due to spin. (Closest: 0.4953 m)

	Initial Ball Position		Final Robot Position		Robot Distance from Ball		
Trial	X	Y	X	Y	X	Y	Euclidian
1	1.5811	1.2869	1.4499	1.021	0.1312	0.2661	0.2967
2	1.5821	1.2516	1.4529	1.0016	0.1292	0.2450	0.2813
3	1.5804	1.2722	1.4496	1.0395	0.1308	0.2327	0.2669
4	1.5565	1.2737	1.3996	1.0416	0.1569	0.2321	0.2801
5	1.5759	1.2439	1.4151	0.9737	0.1608	0.2702	0.3144
6	1.5754	1.2548	1.4422	0.9738	0.1332	0.2819	0.3110
Average					0.1403	0.2553	0.2914

Table 2: Summary of the six trials where the robot was controlled with the MLP that was trained on a dataset that consisted *only* of data from physical trials with the robot under control of the rule-based system.

	Initial Ball Position		Final Robot Position		Robot Distance from Ball		
Trial	X	Y	X	Y	X	Y	Euclidian
1	1.5588	1.2697	1.4643	1.0543	0.0945	0.2154	0.2352
2	1.6091	1.2413	1.5431	1.0365	0.0660	0.2048	0.2152
3	1.5804	1.1979	1.4607	0.9319	0.1197	0.2660	0.2917
4	1.5767	1.2362	1.4508	0.9860	0.1259	0.2502	0.2801
5	1.5691	1.236	1.4506	0.9854	0.1185	0.2506	0.2772
6	1.6021	1.2792	1.4798	1.0339	0.1223	0.2453	0.2741
Average					0.1062	0.2387	0.2623

Table 3: Summary of the six trials where the robot was controlled with the MLP that was trained on a dataset that consisted data from physical trials with the robot under control of the rule-based system augmented with GAN generated data.

from the ball will mean that the robot cannot hit it. However, it is gratifying to see this so clearly in the data.

The robot controlled by the MLP/GAN failed to hit the ball to the target in 14 trials. It did, however, hit the ball in nine of those trials. As a result, there is "closest approach" data for the MLP/GAN, and this shows that the MLP/GAN controller does not perform as well as the rule based system. The average closest approach for the MLP/GAN is 47 cm, as opposed to 23 cm for the rule-based system. Examining the full set of plots of the trials (a subset of which are in Figure 4), it seems that to hit the target it is necessary for the robot's final position to be facing the target before attempting its swing. The MLP/GAN was not as effective as the rule-based system at doing that. When the MLP/GAN-controlled robot hit the ball, it generally sent the ball straight ahead, as opposed to the

angle necessary to hit the target. This misalignment resulted in the relatively high average closest approach. However, the best shots were within 20 cm. These shots tended to be glancing shots which put a slight counterclockwise spin on the ball, which hooked it towards the target (see Figure 4 (e)).

6 Conclusion

This preliminary work indicates that given a small dataset on which to train a neural network to control a robot, the performance of that network can be significantly improved by augmenting the dataset with the output of a GAN. This is possible given an initial training set with as few as 14 trials. While the performance of the neural network trained with data from the GAN does not outperform the system that generated the initial data, we have yet to make a serious attempt to achieve this. The performance of systems that make use of GANs in other domains suggest that they have the potential to further boost the performance of this system, and can make it more robust overall by generating data that is more varied than the original trial data, while still being realistic. Future work will examine if this is possible in our domain.

References

1. M. Arjovsky, S. Chintala, and L. Bottou. https://github.com/martinarjovsky/WassersteinGAN, 2017. [Online; accessed 10-Jan-2019].
2. M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. 2017. arXiv:1701.07875.
3. J. C. Bongard. Using robots to investigate the evolution of adaptive behavior. *Current Opinion in Behavioral Sciences*, 6:168–173, 2015.
4. A. Coates, P. Baumstarck, Q. V. Le, and A. Y. Ng. Scalable learning for object detection with gpu hardware. *IROS*, 2009.
5. J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.
6. I. J. Goodfellow. Generative adversarial networks. *CoRR*, abs/1701.00160, 2017.
7. I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014.
8. A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi. Social GAN: socially acceptable trajectories with generative adversarial networks. *CoRR*, abs/1803.10892, 2018.
9. Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521, 2015.
10. J. Lee and M. S. Ryoo. Learning robot activities from first-person human videos using convolutional future regression. *CoRR*, abs/1703.01040, 2017.
11. L. Pinto, D. Gandhi, Y. Han, Y.-L. Park, and A. Gupta. The curious robot: Learning visual representations via physical interactions. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *ECCV*, 2016.
12. M. Swain and M. Stricker. Promising directions in active vision. *International Journal of Computer Vision*, 11:109, 1993.
13. A. Zelenak. TF2 ROS quaternion basics. http://wiki.ros.org/tf2/ Tutorials/Quaternions, 2019. [Online; accessed 10-Jan-2019].