A Robustness Analysis of Deep Q Networks

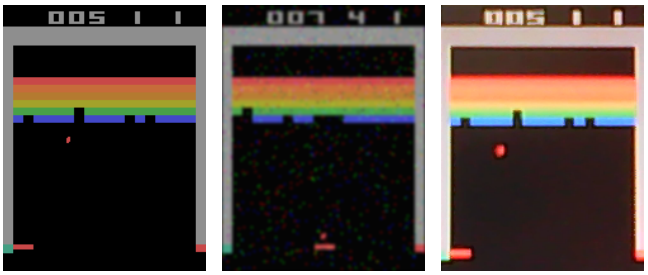
Adam W. Tow, Sareh Shirazi, Jürgen Leitner,
Niko Sünderhauf, Michael Milford, Ben Upcroft
ARC Centre of Excellence for Robotic Vision
Queensland University of Technology
adam.tow@qut.edu.au

Abstract

Deep Q Networks are a type of deep reinforcement learning algorithm that have been shown to be particularly adept at learning a variety of tasks with minimal priors. Specifically, DQN agents have been shown to learn a variety of Atari 2600 video games using only raw images of the game screen and the game score. To leverage DQNs in real world robotics applications, we must first understand how robust these networks are to the perceptual noise common to all robotics domains. In this paper, we present an analysis of the robustness of Deep Q Networks to various types of perceptual noise (changing brightness, Gaussian blur, salt and pepper, distractors). We present a benchmark example that involves playing the game Breakout though a webcam and screen environment, like humans do. We present a simple training approach to improve the performance maintained when transferring a DQN agent trained in simulation to the real world (36% vs. 1% maintained performance - see Table 1). We also evaluate DQN agents trained under a variety of simulation environments to report for the first time how DQNs cope with perceptual noise, common to real world robotic applications.

1 Introduction

The ability to learn how to perform tasks, as opposed to explicitly programming them, is a feature many roboticists seek for their robots. One approach to achieving this is vision based control. Deep Q Networks (DQNs) are a type of deep reinforcement learning algorithm that takes a trial and error approach to learning a mapping between high-level sensory inputs and low-level actions, in order to complete specific tasks. To date, DQNs have demonstrated superior-to-human performance on a number of Atari 2600 video games [Mnih *et al.*, 2015], raising the question of deep reinforcement learning's applicability to robotics in real world environments.



(a) Unchanged (b) Sim. Noise (c) Real World

(d) Robot Experiment

Figure 1: Three different game screen environments were used across the training and testing of both agent DQN[1] and our agent. 1a is an example of the pristine game screen images used in [Mnih *et al.*, 2015]. 1b is an example of the game screen images used to train our agent. 1c is an example of the noise caused by viewing the game screen through a low cost webcam. 1d shows the Baxter robot playing Atari 2600 game Breakout through a webcam's view of the game screen.

The challenge here, is that for an agent to transfer

from ideal environments to real environments, they must be able to cope with perceptual variations to their environment. We find that even small additions of noise (e.g. increase all RGB values in range 0 to 1 by 0.1) to DQN's game screen inputs are enough to degrade performance to random chance.

We present a training regime that introduces noise to the game screen environment that improves the agent's ability to maintain the performance it learned in simulation. The agent we trained with noise will be referred to as 'ours' throughout this paper. In our webcam and screen environment (see right side of Figure 1d), we show our agent maintains 99% of its performance when tested in a noise free environment and 36.4% of its performance when tested in a real environment. The agent we trained in a noise free environment in alignment with [Mnih *et al.*, 2015] is referred to as DQN[1]. DQN[1] when tested in a simulated noise environment maintained less than 1% of its performance. This result remained true when DQN[1] was tested in our real world webcam and monitor environment.

We extend our result by investigating how DQNs cope with various types of noise individually and perform a cross-analysis to find whether training with certain types of noise improve robustness to other types of noise. We embody our agent in a Baxter robot and use prescripted trajectories to allow actions to be sent to the game through a keyboard interface. In particular, we make the following contributions:

- Development of a multi-noise simulated training regime to improve maintained game play performance of a DQN agent trained in simulation but tested in a real world environment.
- Introduction of a benchmark example for evaluating the perceptual robustness of artificial agents to real world noise introduced by low cost cameras.
- Analysis of the perceptual robustness of DQN to various types of noise, including background distractors.

We find that DQN is far less robust to perceptual noise than initially expected and hope this work is useful for people working in this area.

2 Related Work

Deep learning in recent years has seen widespread attention in many fields including robotics [LeCun *et al.*, 2015]. Prominent applications of deep learning to robotics has entered in the area of robotic vision. Deep learning has allowed robot perception systems to be learned directly from raw data, replacing handcrafted feature detectors suited to just one environment or just one task [Levine *et al.*, 2016b; Finn *et al.*, 2016; Bojarski *et al.*, 2016]. Convolutional Neural Networks (CNNs) are the most commonly applied tool used for learning such perception systems [LeCun *et al.*, 2015]. Specifically, CNNs are able to learn a mapping from high-dimensional image space to a low-dimensional feature space, providing class labels to image proposals as a common example [Krizhevsky *et al.*, 2012]. Agent-based tasks that require vision can be separated into two key approaches: split approaches and end-to-end approaches.

2.1 Split Approaches

Within robotics, a popular approach to leveraging deep learning is to use a neural network to process raw images, using their outputs as inputs to another system. Auto-encoder neural networks provide one way of extracting a low dimensional feature space from high dimensional images [Rumelhart *et al.*, 1985; Baldi, 2012]. In [Lange *et al.*, 2012], a deep auto-encoder network was used to estimate the 2D position of a toy slot car from raw overhead images to drive it around a track. In [Finn *et al.*, 2016] a number of robotic manipulation tasks requiring close hand-eye coordination were performed through the use of Guided Policy Search on a state representation learned with a deep spatial auto-encoder. In [Watter *et al.*, 2015] a deep auto-encoder network allowed linear-quadratic-Gaussian control to operate on the learned low-dimensional state representation. To train these networks, each work relied on the availability of vision-free trajectories of the task being performed by the target robot. Investigations into the robustness of the approaches to changing lighting conditions were not performed.

CNN's also provide a way of extracting low dimensional feature spaces though are often limited by a requirement on expensive to obtain ground truth data [Krizhevsky *et al.*, 2012]. In [Chen *et al.*, 2015], a mapping between raw driver-perspective images and a preselected set of driving-specific measurements was learned but required expensive to obtain ground truth measurements to facilitate supervised training.

In [Yoon *et al.*, 2016], an iterative linear-quadratic-regulator algorithm was used to control the joint angles of a Baxter robots arms to play a game on a mobile phone screen. The CNN component of the DQN algorithm was replaced by a hand-crafted perception system, distancing the approach from work herein.

Tzeng et al., [2015] presented a method to bootstrap simulator to real world transfer of a pose estimation CNN by exploiting known synthetic-real image pairs. While a real image with multiple similar simulated images were explicitly handled by the implemented contrastive loss function, it remains unclear how the occurrence of multiple visually dissimilar views of the same

state (like changing lighting conditions) would influence the performance of the approach. Future work could investigate how the approach might apply to an end-to-end learning system such as DQN.

2.2 End-to-end Approaches

End-to-end learning for robotics encapsulates systems that learn vision and control jointly. Joint learning of perception and control removes the need for hand-crafted perception systems and hand-crafted controllers, replacing them with a single framework that maps raw images to control outputs [Pomerleau, 1989].

End-to-end learning systems have been used across a number of application areas. Some of the areas where end-to-end learning systems have been presented include autonomous driving in both simulated and real environments [Koutník *et al.*, 2013; Bojarski *et al.*, 2016], robot tasks that require close hand-eye coordination [Levine *et al.*, 2016a] and performing a range of simulated tasks [Mnih *et al.*, 2013; 2015; Nair *et al.*, 2015; Lillicrap *et al.*, 2015]. Across these applications, how each system handles perceptual noise, if at all, remains unclear.

Where end-to-end learning systems have been presented in real world environments, they often require one of either supervised pre-training [Levine *et al.*, 2016a], or a significant amount of data [Levine *et al.*, 2016b] to work. An ability to handle perceptual noise is even more crucial in data-driven approaches where training has been shown to take months even with multiple robots operating simultaneously [Levine *et al.*, 2016b].

Using DQN in real world environments was initially investigated in [Zhang *et al.*, 2015] however the results reported a failure to handle the transfer from simulation to the real world. To remedy this, a simulated view of the environment was generated using encoder readings of the real robot to allow control to take place.

2.3 Summary

Both split and end-to-end learning approaches have been applied to robotics applications. Reinforcement learning and especially DQN has much to offer robotics in terms of its ability to train agents to perform a significant number of diverse tasks with minimal priors on the task and capabilities of the agent. Missing across previous work is focussed investigation into the perceptual robustness of reported systems which directly influences their applicability to robotics applications. The work in this paper is motivated by the need to understand and overcome the limitations of DQN agents in adapting to noisy, real-world environments. In this way, we present a simple approach to improve a DQN agents ability to maintain performance when transferring between the same environment exposed to different perceptual noise; ensuring a robot controlled by a Deep Q Network is able to continue to complete a task robustly and reliably.

3 Deep Q Network Architecture

Herein we consider a robotic agent interacting with its environment through a sequence of actions and observations. The agent aims at finding a policy, i.e. a sequence of actions, leading to the maximum cumulative future reward. Q learning describes a technique for learning such an optimal policy based on the reward received at state s given a specific action was taken [Watkins and Dayan, 1992]. More formally Q learning aims to approximate the optimal action-value function

$$\mathcal{Q}^*(s,a) = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s, a_t = a, \pi], \tag{1}$$

given the observed state s and taken action a at time t conditioned by the current policy π [Watkins and Dayan, 1992]. The future reward (r_t) is discounted at every time step t by constant factor γ. We are interested in representing the state s with visual observations from camera images only.

Recently [Mnih *et al.*, 2015] have shown the feasibility to use deep, convolutional networks to approximate this Q function for playing emulated Atari games. In this paper we use the same network architecture and hyper-parameter settings as reported by [Mnih *et al.*, 2015]. The CNN begins with an input size of 84x84x4 (four grey-scale images) as in Figure 2. Within the network, each convolutional layer is followed by a rectified linear unit (ReLU) layer [Nair and Hinton, 2010]. The first layer convolves 32 filters of size 8x8 with stride 4. The second layer convolves 64 filters of size 4x4 with stride 2. The third layer convolves 64 filters of size 3x3 with stride 1. The final hidden layer (second last layer) is a fully connected layer with 512 units. The output layer is another fully connected layer with a single output for each action in the game. For Breakout, the actions included no action, move left and move right.

The reward received from the game environment is used to update the weights of the CNN to better approximate the Q-function. The weights are changed using RMSProp stochastic gradient descent [Tieleman and Hinton, 2012].

The DQN architecture interfaces with the open-source Atari 2600 emulator, Stella, through the Arcade Learning Environment (ALE) [Bellemare *et al.*, 2013]. ALE provides a simple interface between DQN and Stella, allowing a DQN agent to send and receive the information needed for training and testing. Specifically, training is informed by retrieval of a reward and corresponding game screen image for a particular action.

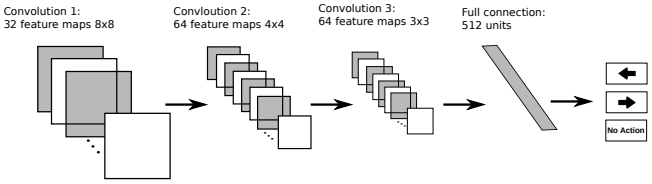


Figure 2: Deep Q Network wraps a convolutional neural network (CNN) in a modified Q-learning algorithm.

4 Experimental Setup

We ran two experiments on our benchmark environment to demonstrate how training our agent with augmented data improved the agent's ability to maintain performance when it's environment changed in appearance. We then embodied our agent within a Baxter robot that viewed the game through a webcam and performed actions with a keyboard (see Figure 1d). Further analysis with an additional six training environments cross-analysed in simulated environments is presented in Section 6.

We trained two DQN agents using the framework in [Mnih *et al.*, 2015]. The first agent was trained on the Atari 2600 game Breakout with identical hyperparameters to those reported in [Mnih *et al.*, 2015]. This agent is referred to as DQN[1]. The second agent was also trained on Breakout however, the game screen images exposed to this agent during training were subjected to simulated noise. This agent is referred to as 'our' agent or 'ours'. To evaluate each agent's performance, both DQN[1] and our agent were tested on three modifications of the Breakout game environment. These included the unchanged game environment as in [Mnih *et al.*, 2015], the simulated noise environment used to train our agent and a real world environment as in Figure 1. The real world environment used a webcam's view of the game screen. This view was unseen by both agents during training. To further demonstrate real world performance, we used the Baxter robot platform to embody our agent. Here Baxter retrieved game screen images through the webcam and selected keys on a keyboard to execute game actions.

4.1 Training

DQN[1] was trained in an unchanged game environment as a baseline performance measure. An example of the Breakout game screen that DQN[1] was exposed to during training can be seen in Figure 1a.

Specifically, the procedure follows: at each time step, the agent selects an action based on the preceding four game screen images. A frame skipping technique is used whereby the selected action is performed in the game four times, corresponding with [Mnih *et al.*, 2015]. After the action has been performed four times, a reward based on the current game score is returned. This process continues for a number of epochs; each epoch composed of 250,000 time steps as in [Mnih *et al.*, 2015].

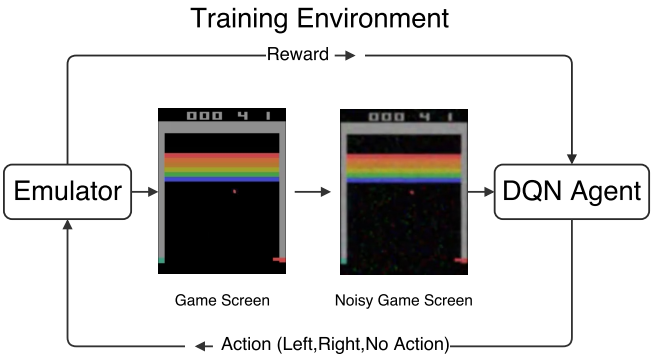


Figure 3: Training flow diagram shows key information transfer between DQN framework sub-systems.

Our agent was trained in a modified game environment where the game screen had been subjected to simulated noise. See Figure 3 for a summary of the training scheme. To represent the conditions of blurring, sensor noise and varying lighting conditions that are common in robotic vision tasks, three representative sources of noise were introduced during training. Each game screen image was subjected to a combination of 2% salt and pepper noise, a uniform random brightness level applied by adding a scalar (range [-0.25,0.25]) to each RGB pixel and Gaussian blur. The Gaussian blurring was performed using a 5x5 kernel with standard deviation 0.5 and mean 0.5.

Training is monitored at each epoch with the average action value. The average action value is calculated by computing the average maximum Q-values on a held-out set of game states. Briefly, for each of 32 held-out game states, DQN makes a prediction on the total reward it would expect to achieve by the end of the game if it performed a given action. As the agent improves, its expectation of the total reward it can achieve by the end of the game increases. To avoid local optima, a balance must be struck between choosing to act randomly versus on-policy. Evaluation stages set a 5% chance of choosing a random action to avoid local optima (consistent with [Mnih *et al.*, 2015]).

4.2 Testing

The performance of DQN[1] and our agent was quantified in three distinct environments. These included the unchanged game environment used to train DQN[1] (Figure 1a), the simulated noise environment used to train our agent (Figure 1b) and the real world environment (Figure 1c). Real world environment tests were run solely with the webcam and monitor (right hand side of Figure 1d) to increase the speed of agent evaluations.

To compare our agent with DQN[1], we followed a

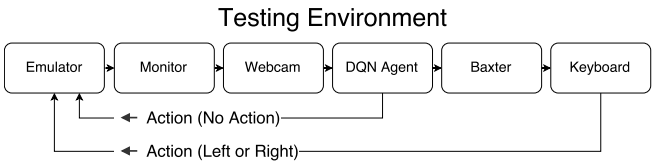


Figure 4: In our real world test setup, Baxter plays Breakout in stop-motion. The emulator waits for left or right actions to be received from Baxter's interaction with the keyboard. No-action commands bypass Baxter.

similar approach to that described in [Mnih *et al.*, 2015]. The approach taken allowed each agent to play through Breakout thirty times. The total score achieved after each play was then recorded.

4.3 Testing on Baxter

The Baxter robot platform was used to demonstrate real world performance. Modelled in Figure 4, our agent visually perceived the Breakout game screen using the webcam. Images captured with the webcam were cropped to the 210x160 game screen dimensions that the emulator outputs. The webcam was aligned such that the game screen image on the monitor exactly filled the webcams view. Using this view, our DQN agent would select an action to perform based on the last four game screen images captured.

On selecting an action to perform, Baxter was instructed to either press the left-side of the keyboard (for action 'left') or the right-side of the keyboard (for action 'right') with a ROS command. To achieve this, a path between two manually selected end-effector positions (key pressed and key not pressed) was calculated using an inverse-kinematics path planner. On receipt of keyboard input, the emulated Breakout game was stepped with the corresponding action. For simplicity, the action 'no action' was directly passed to the emulator, allowing the game to progress more quickly in those times.

5 Results

We found that training a DQN agent in an environment with simulated noise improved maintained performance under perceptually different environments. On average, our agent was able to maintain 99% performance when tested in the unchanged noise free environment and 36.4% in our webcam and screen environment. Note that these environments were not seen during training. A DQN agent trained in the unchanged noise free environment, named DQN[1], maintained less than 1% performance when tested in the noisy-simulated and real world environments. The results in this section cover the training and testing of these two networks. Further analysis on training different DQN agents is presented in Section 6.

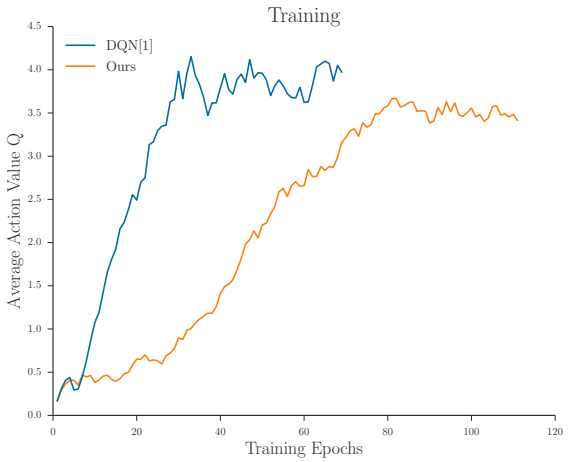


Figure 5: The addition of simulated noise increased the training time required to reach a steady level of performance however, the stability of our agent remained largely unaffected.

5.1 Training Performance

Plots of the average action value of our agent and DQN[1] are shown in Figure 5. The average action values were computed by evaluating each network in their respective training environment at discrete intervals during training. Our agent was noticeably slower to converge than DQN[1] as a result of the confusion on state introduced by perceptual noise.

5.2 Testing Performance

The game play scores achieved by our agent and DQN[1] across three environments can be seen in both Figure 6 and Table 1. The performance of our agent in both the simulated noise environment and the real world environment is shown to be superior to the performance of DQN[1] in the same environments. Note the median performance of DQN[1] in the simulated noise and real world environments was 0.0.

Further performance results of our agent and DQN[1] have been included in Table 1. While our agent's performance is a significant drop from that achieved by DQN[1] in the unchanged game environment, the maintained performance of our agent across the three environments demonstrates a robustness to environmental condition-variation unseen with DQN[1]. It is important to note that the noisy training environment makes our agent's task fundamentally harder to learn than DQN[1]'s. As an aside, our experiments also revealed that new gameplay strategies can be learnt after the average action value asymptotes.

Game Play Performance Results			
Agent	Environment	Mean Scores (±std)	Normalised (% Human)
DQN	Reported Result [Mnih *et al.*, 2015]	**401.2** (±26.9)	**1327.2%**
DQN[1]	Unchanged Env.	**284.4** (±130.0)	**939.2%**
Ours	Unchanged Env.	29.6 (±11.8)	92.7%
DQN[1]	Sim. Noise Env.	1.0 (±2.2)	-2.3%
Ours	Sim. Noise Env.	**29.4** (±9.9)	**92.0%**
DQN[1]	Real World Env.	2.5 (±4.4)	2.7%
Ours	Real World Env.	**10.7** (±5.6)	**29.9%**

Table 1: While our agent performed poorer than DQN in the unchanged environment, it is important to note that our agent was not trained there. Our agent showed robustness to both the unchanged environment and the real world environment that were not seen during training. While the average game scores of DQN in both the simulated noise and real world environments were just above zero, the median scores in both were 0.0, indicating an inability to recognise state in these environments. The final column normalises the average game score to scores achieved by random action selection and a professional human games tester. [Mnih *et al.*, 2015] reports a score (averaged across 30 episodes of game play) for a random agent as 1.7, and 31.8 for a professional human games tester.

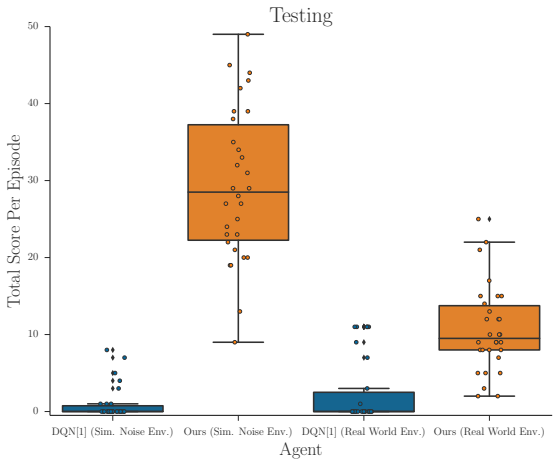


Figure 6: We tested both DQN[1] and our agent in a real world testing environment not seen during training. The setup used a low-cost webcam directed at an LCD monitor to capture images of the Breakout game screen. We find that although you pay a cost in terms of idealistic performance, we can perform in a real world setting better than DQN[1]. Our agent performed near its peak in its simulated noise training environment whereas DQN[1] achieved the same median score of 0.0 as with the real world environment.

5.3 Baxter's Game Play Performance

Our Baxter robot demonstration resulted in a recorded thirty minutes of game play. Thirty minutes was a chosen time with tests indicating that game play could continue for much longer. The performance of our agent on Baxter matched that of the webcam-only setup as our approach ensured synchronisation between actions and webcam images of the game screen was always maintained. Specifically, an action was only performed in the Breakout game after Baxter had successfully selected an action to perform using the keyboard. The action 'no action' was directly performed on the Breakout game, bypassing Baxter.

6 Further Analysis

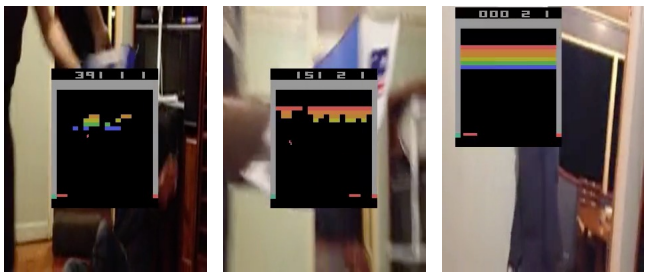
The last section demonstrated how DQN can be successfully transferred onto a real world system by augmenting the training data with simulated noise. In order to increase our understanding and to create more insights into how exactly noise influences DQN's performance, we ran a suite of extensive experiments in simulation. In contrast to the experiments explained in Sections 4 and 5, we now separate the three noise types (Gaussian blur, salt and pepper, varying brightness) and add additional distractors in the form of a real world video sequence (of moving people) in the background surrounding the game screen. The background video addition was chosen to better align with the way humans often play games. In doing so, the input image dimensions were increased such that the game screen maintained the original dimensions of 210x160 px.

We provide answers to the following questions: 1) How important is cropping the image input for DQN? Is DQN distracted by moving background objects? This is important for future robotic applications in a less constrained environment than we showed with Baxter before. 2) How well does DQN generalize when trained to be robust against one type of noise? This generates insights about how the data augmentation should be performed and which noise types and individual noise levels should be included.

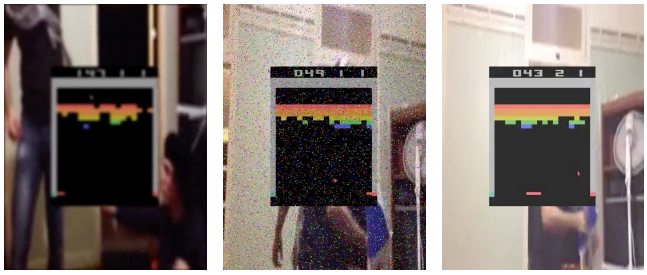
All of these experiments ultimately aim at enabling robotic systems to learn behaviours, instead of relying on fixed scripted behaviours coded by a human expert. The additional agents have been provided with short names that are described in Table 2. Examples of the environments in which these agents were trained have been

DQN Agent Name	Description
DU	*uncorrelated* distractor video in the background
DC	*correlated* distractor video in the background
TR	uniform random game screen position within greater image
BL	Gaussian blur, kernel size $9 \times 9, \sigma = 0.25, \mu = 0.5$
SP	salt & pepper noise, per-pixel probability 6%
BR	brightness change, all rgb pixels changed by uniform random scalar in range $[-0.25, \ldots, 0.25]$

Table 2: DQN agents trained and tested during evaluation. See text for detailed explanation. Noise levels were selected to show noticeable change (to a human) whilst remaining playable (by a human).



(a) Agent DU (b) Agent DC (c) Agent TR



(d) Agent BL (e) Agent SP (f) Agent BR

Figure 7: Six additional DQN agents were trained under varying noise and distractor-laden conditions.

included in Figure 7.

6.1 Training Environments

Each of the six additional agents were exposed to different environments. Across the six, a commonality was the introduction of distractors, i.e. visual information that is unrelated to the task. For a vision-based robot, successfully ignoring such distractors is critical for reliably performing tasks in dynamic, real world environments. To test how DQN coped with distractors, we bordered game screen images with frames from a real world video. The video consisted of one million unique, temporally-coherent frames. To prevent performance changes arising from a reduction in game screen dimensions, the DQN framework was modified to accommodate images of size 400x304 px.

The details of each agents training environment have been itemised below.

- Agent DU was trained solely with the additional background video (see Figure 7a). The background video was sufficient in length to extend past multiple plays through the game, reducing the possibility of feature correlation between the game and the video.
- Agent DC was trained in an environment where the background video was restarted when the game ended. This introduced a small correlation between the state of the background and the state of the game (see Figure 7b).
- Agent TR was trained in an environment where the game screen was given a random position within the background video area on a frame-by-frame basis (see Figure 7c).
- Agent BL was trained with Gaussian blur of kernel size 9x9, standard deviation 0.25 and mean of 0.5 (see Figure 7d).
- Agent SP was trained with salt and pepper noise where every pixel had a 6% chance of being either salt (1) or pepper (0) (see Figure 7e).
- Agent BR was trained with varying brightness levels in the range [-0.25,0.25] (see Figure 7f).

6.2 Testing Results

We cross-analysed the performance of each agent in conditions extending those from training. We compare the robustness of each agent to each type of noise. Agent TR was excluded from this analysis as it failed to learn to play Breakout, representing an example of an environmental condition too complex to learn in. All scores shown in Figure 8, Figure 9 and Figure 10 have been mean normalised to the agents performance in its training environment, to evaluate the relative impact on performance across all of the agents irrespective of the absolute performance they reached during training.

Figure 8 shows the average score achieved by each agent against an increasing intensity of Gaussian Blur.

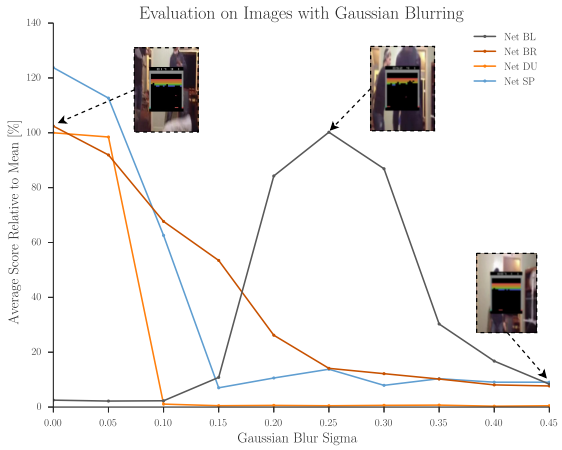


Figure 8: Four agents were tested on game screens with increasing gaussian blur. Agent BL was trained with gaussian blur at sigma 0.25 and is shown to no longer cope in an environment free of blur. Other agents show small level of robustness to Gaussian blur.

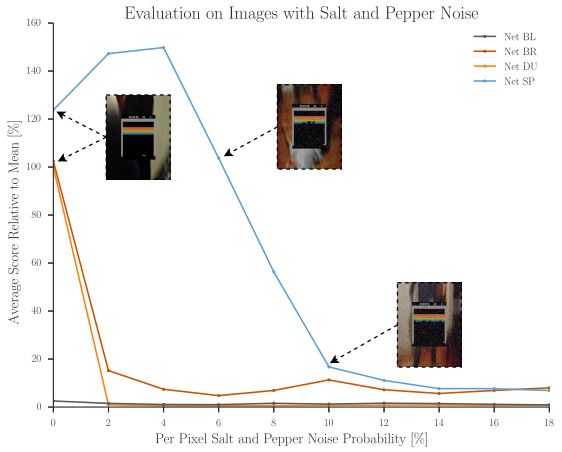


Figure 9: Four agents were tested on game screens with increasing salt and pepper noise. Agent SP who was trained with 6% salt and pepper noise performed better with less of that noise.

Each evaluation increased the sigma of a 9x9 Gaussian Kernel from 0.05 to 0.45 in intervals of 0.05 (sigma=0 is no blur). Agent BL was shown to maintain some performance directly above and below the sigma 0.25 condition where it was trained, showing some resilience outside of training conditions. Agent BL, different to the other three networks, failed to cope with little to no blurring.

Figure 9 shows the average score achieved by each agent against an increasing intensity of Salt and Pepper noise, a distribution which increased from 0% to 18%.

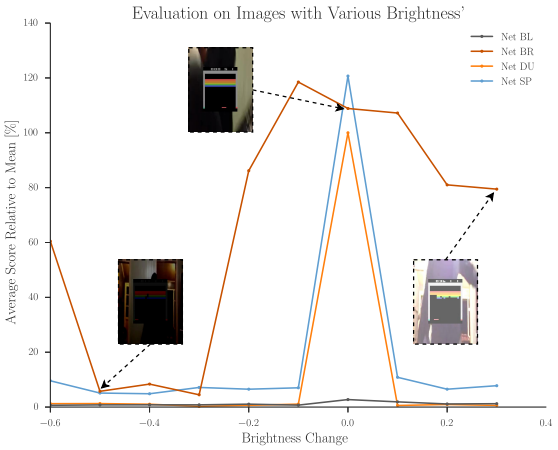


Figure 10: Four agents were tested on game screens with varying brightness levels. Agent BR who was trained on under random brightness values within [-0.25,0.25] performed well.

The most robust agent to the salt and pepper noise was agent SP, who was trained at 6% salt and pepper noise. Agent SP's performance improved with slightly less salt and pepper noise than was experienced during training. Salt and Pepper noise of 2% was adequate to reduce the performance of agents BR and DU to less than 20% of the performance they reached in their training environments.

Figure 10 shows the average score achieved by each agent against a range of brightness. Each test applied a constant brightness, which was changed by adding value in the range [-0.6 to 0.3] to every pixel in each image. The most robust agent to changes in brightness was agent BR, who was trained with random brightness levels in the range [-0.25 to 0.25]. Agents BL, DU and SP were unable to cope with any changes to the brightness.

6.3 Summary

We found that the environment transferability of DQN agents is not clear cut. In particular, we found that DQN agents are far less robust to perceptual noise than initially presumed. Agent BR (trained on images of varying brightness) was the only agent that showed a significant increase in robustness to Gaussian blur; a noise type it did not experience during training. Agents BL, SP and BR, all showed some robustness to more and less of the same type of noise that they were trained for. In the case of agent BL (trained on blurry images), performance dropped significantly when provided with unchanged game screen images (zero noise).

These findings highlight that exposing a DQN agent to different types of noise during training can lead to

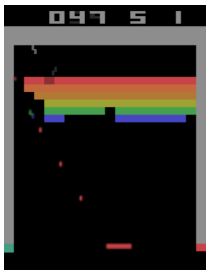


Figure 11: The 'trick' refers to a game play strategy where the player burrows through the brick layers to bounce the ball into the cavity above. This strategy leads to a significant increase in score.

different traits in the features being learnt. As such, care must be made when training such artificial agents to ensure that the environmental conditions in which you want it to operate are accommodated with an appropriate data augmentation process.

7 Discussion

We found that our agent trained with three types of noise was able to cope with changes to the perceptual environment. On average, our agent was able to maintain 99% performance when transferred into a zero noise environment and 36.4% in our real world environment. DQN[1] trained in the noise free environment maintained less than 1% performance when tested in the simulated-noise and real world environments. The performance of our agent in our real world environment demonstrates that Deep Q Networks are capable of training task-specific agents in simulation, that can successfully make the jump to a real world environment without any additional modifications.

While our maintained performance in the real world environment improved over DQN[1], our agent was both slower to train and performed poorer overall in terms of average game score. We believe this can be explained by the way DQN agents are trained. Specifically, making action decisions informed by previous state-action pairs breaks down when identical states appear different. Recall each agent's state is entirely estimated from the game screen.

The performance disparity between DQN[1] that we trained and that reported in [Mnih *et al.*, 2015] can be explained by the shorter time we trained the agent for. An advanced game-specific strategy exists with Breakout that when learnt, leads to a significant boost in game score. If you have ever played Breakout, you may recall breaking through the brick layers to bounce the ball into the cavity above. We refer to this strategy as the 'trick' and display a depiction in Figure 11. Where our agent had only learnt to bounce the ball as often as possible, DQN[1] had just uncovered the 'trick' and in the reported results, their agent had mastered it. We believe that our agent trained with noisy game screen images would eventually uncover the 'trick' if provided enough training time.

8 Conclusion

The perceptually-rich, dynamic environments in which we live are environments largely untested with leading deep reinforcement learning approaches. This work has demonstrated the perceptual robustness of DQN agents and presented a simple approach that improves maintained performance from simulation to real world environments. We presented a benchmark task for evaluating how simulated agents can be evaluated under real world perceptual noise. We also performed an analysis of DQN agents to various types of noise and distractors to provide insights into the appropriateness of Deep Q Networks to robotics applications.

Summary By introducing various types of noise during training, it is possible to train a DQN agent in simulation that can transfer to a real world environment with no modifications. The perceptual robustness of DQN agents to various types of noise is not clear cut.

Acknowledgements

Chris Lehnert for assistance operating the Baxter robot. This research was supported by an APA scholarship and by the Australian Research Council Centre of Excellence for Robotic Vision. Computational resources and services used in this work were provided by the HPC and Research Support Group, Queensland University of Technology, Brisbane, Australia.

References

[Baldi, 2012] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. *JMLR*, 2012.

[Bellemare *et al.*, 2013] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 06 2013.

[Bojarski *et al.*, 2016] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[Chen *et al.*, 2015] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[Finn *et al.*, 2016] C. Finn, Xin Yu Tan, Yan Duan, T. Darrell, S. Levine, and P. Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519, May 2016.

[Koutník *et al.*, 2013] Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1061–1068. ACM, 2013.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[Lange *et al.*, 2012] Stanislav Lange, Martin Riedmiller, and Arne Voigtlander. Autonomous reinforcement learning on raw visual input data in a real world application. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8. IEEE, 2012.

[LeCun *et al.*, 2015] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[Levine *et al.*, 2016a] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[Levine *et al.*, 2016b] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *arXiv preprint arXiv:1603.02199*, 2016.

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

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

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Nair and Hinton, 2010] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[Nair *et al.*, 2015] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

[Pomerleau, 1989] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. Technical report, DTIC Document, 1989.

[Rumelhart *et al.*, 1985] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[Tieleman and Hinton, 2012] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2), 2012.

[Tzeng *et al.*, 2015] Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Xingchao Peng, Sergey Levine, Kate Saenko, and Trevor Darrell. Towards adapting deep visuomotor representations from simulated to real environments. *arXiv preprint arXiv:1511.07111*, 2015.

[Watkins and Dayan, 1992] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[Watter *et al.*, 2015] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems*, pages 2746–2754, 2015.

[Yoon *et al.*, 2016] Wonjun Yoon, Sol-A Kim, and Jaesik Choi. An end-to-end robot architecture to manipulate non-physical state changes of objects. *arXiv preprint arXiv:1603.01303*, 2016.

[Zhang *et al.*, 2015] Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft, and Peter Corke. Towards vision-based deep reinforcement learning for robotic motion control. In *Australasian Conference on Robotics and Automation 2015*, Canberra, A.C.T, September 2015.