Intrinsic Motivation for Reinforcement Learning through Curiosity

Alex Van de Kleut
Department of Neuroscience
Brock University
St. Catharines, Canada
Email: av15fj@brocku.ca

Brandon Cheshire
Department of Neuroscience
Brock University
St. Catharines, Canada
Email: bc14tm@brocku.ca

***Abstract*—Our work presents a novel technique for internally generating reward in a reward-free environment for use in reinforcement learning. Our agent behaves 'curiously', choosing actions that maximize prediction error of the effects of such an action on the environment. The novelty of our approach involves using an autoencoder to compress the input state into an encoded vector. We train a neural network to predict the effect of an action vector on the encoded state rather than on the raw input state. The difference between the actual encoded next state and the predicted encoded next state is used as the reward signal for our agent, who learns to choose actions that maximize this prediction error using a policy network and Q-learning. The policy network is pre-trained by human controllers to guide initial behaviour, which is then allowed to learn behaviours that maximize prediction error. Our model is capable of maintaining high prediction error and successfully exploring an environment with rooms attached to a corridor, which indicates that it has successfully learned to be curious. Our research can be extended by reusing our trained model to operate in a virtual, internally generated model similar to 'dreaming'. Our work presents a way to generate intelligent behaviour without the need for explicitly designed external rewards.**

I. Introduction

One of the three main branches of machine learning is reinforcement learning. This is a technique that was originally inspired by behaviourist psychology, in particular Thorndike's law of effect, which states that actions resulting in a positive outcome are more likely to be repeated and actions resulting in a negative outcome are less likely to be repeated [1]. Reinforcement learning involves an agent who transitions from a state s_t to a state s_{t+1} by performing some actions a_t. The goal of reinforcement learning is to develop a policy $\pi : S \rightarrow A$ that maps states to actions. The policy learns which action a_t maximizes the reward r_t given the state s_t, with the net effect being that the total reward r is maximized.

Reinforcement learning has gained extreme popularity lately with the development of highly successful models that use neural networks to learn π [2]. However, most reinforcement learning studies operate in virtual environments, such as video games. One of the consequences of this is that most reinforcement learning environments (such as the popular OpenAI Gym) naturally provide some kind of continuous reward function from the environment. However, this is at odds with how agents operating in the real world function, where no such feedback is available.

The question then arises: how do we define reward in the real world? This question is studied in the field of motivation theory and is strongly connected with neuroscience. If we examine most organisms, a reductionist approach might stipulate that organisms are motivated by the need for reproduction, which in turn motivates behaviours such as seeking protection and foraging. However, even in complex organisms like humans, such behaviour comes pre-packaged with our brains at birth (for example, the hypothalamus controls hunger, thirts, sleep, sex and aggression to name a few basic functions). We can see then that satisfying these drives provides some kind of initial intrinsic reward. This poses a problem, as it would appear real-life agents do not come *tabula rasa* and already have some predefined policy that drives behaviour. Considering this, and the fact that training agents from scratch to behave intelligently is difficult, one might consider

how to generate an agent that is capable of learning, but already begins with a shaped policy.

What about extrinsic motivation? This type of reward is external to the agent and comes from the environment. Extrinsic reward in the real world is sparse, and some kind of rewards only happen once or twice in a lifetime (for example, graduating or getting married). Traditional reinforcement learning techniques would only update the policy once that reward is experienced. However, we cannot be sure which of our behaviours lead to a rewarded outcome, which is referred to as the "credit assignment problem" [3]. How do humans behave when they are not strictly choosing actions that bring them closer to some sparse extrinsic reward? What drives behaviour more complex than instrincts? One major factor is curiosity [4]. Curiosity drives behaviour that is exploratory in nature and helps us learn about the world around us.

While there are many models of curiosity, one that we found most interesting and applicable was choosing actions that maximize your prediction error. For example, if you are holding a ball, choosing to drop the ball maximizes your prediction error (will it bounce? will it roll? will it break?). There are two advantages to this system. The first is that since our actions maximize our prediction error, they are likely to bring us to novel states. The second is that, by being exposed to novel states, we can better learn an internal model of the world. Additionally, performing actions that have low prediction error will be discouraged, and thus actions like walking into a wall (the outcome of which is easily learnable) will be discouraged.

Our goal is to design an robotic agent (both physically as a robot, and programmatically in software) that evolves in two stages. The first is to train the robot to emulate human control. Recall that organisms come into being with some predefined notion of useful behaviour. By doing this, we can ensure that training of the robot is rapid and begins with already intelligent behaviours. The second is to allow the robot to learn to explore on its own using prediction error as a curiosity reward signal. The robot should learn how to navigate spaces and explore, behaviours that are inherently useful for any mobile organism.

II. Model

Our agent recieves a state s in the form of greyscale pixels ($64 \times 64 \times 1$) and puts out a 4-dimensional action vector a (corresponding to probabilities of increasing or decreasing left or right wheel speeds). Our model is inspired by the intrinsic curiosity module of Deepak Pathak et al. [5]. It consists of two parts.

A. Curiosity Module

The first part is the curiosity module. We discussed rewarding the agent for choosing actions that maximize its prediction error. We might consider some function

$$\psi : S \times A \to S$$

where $\psi(s_t, a_t) = \hat{s}_{t+1}$ predicts the next state given the input state and the action. We might then define reward to be

$$\|s_{t+1} - \hat{s}_{t+1}\|_2$$

the magnitude of the difference between the predicted next state and the actual next state. However, there is a problem with this, namely that many environments are highly stochastic. For example, consider an agent that is exposed to blowing leaves in the wind or static on a television screen. These environments are essentially impossible to predict and thus result in a high prediction error. An agent would be highly rewarded for choosing to stay put and look at this stochastic environment. Since such behaviour is not necessarily intelligent or curious, we want to avoid it. Since the input state s is essentially an image, one might consider simply blurring the image. However, this is not ideal as we are still left with a high-dimensional input where predictions are difficult to make. Indeed, humans do not make predictions about the exact sensory input they will experience next, but rather changes in higher-order internal models. This involves developing a model that can encode the input state s into some low-dimensional vector that contains sufficient detail about s. For this purpose, we employ a function

$$\phi : S \to \mathbb{R}^n$$

where $\phi(s_t)$ produces an n-dimensional vector that represents some kind of 'encoding' of s_t. We also define

$$\phi' : \mathbb{R}^n \to S$$

to be a kind of inverse model that takes the encoded version of s_t and reconstructs $\hat{s}_t$. Then this network is trained to optimize

$$\min \frac{1}{2}\|s_t - \phi'(\phi(s_t))\|_2^2$$

such that $\phi(s_t)$ is sufficient to reconstruct s_t. This choice of distance metric is chosen due to the simplicity of its gradient when performing training, which simplifies to $\|s_t - \phi'(\phi(s_t))\|_2$. This model is referred to as an "autoencoder" [6].

The encoding function ϕ will be helpful since it is in this encoded vector space that we make our predictions about the impact of our actions on the world. Rather than learning a function $\psi : S \times A \to S$ we instead learn a function

$$\theta : \mathbb{R}^n \times A \to \mathbb{R}^n$$

where $\theta(\phi(s_t), a_t) = \hat{\phi(s)}_{t+1}$. In essence, we not trying to predict the actual next state s_{t+1} but rather the *encoded* next state, $\phi(s_{t+1})$. This network is trained to optimize

$$\min \frac{1}{2}\|\phi(s_{t+1}) - \theta(\phi(s_t), a_t)\|_2^2$$

These two networks work together as a single functional unit to produce a network that becomes simultaneously good at encoding the environment as well as making predictions about the encoded environment. The purpose of this entire network is to produce the reward signal for training our policy π. In particular, our reward is some scaled version of the prediction error

$$r_t = \min \frac{\eta}{2}\|\phi(s_{t+1}) - \theta(\phi(s_t), a_t)\|_2^2$$

or a particular action and state.

B. Policy Network

The policy network is a network that takes in the state s_t and produces an action a_t. It is essentially a function

$$\pi : S \to A$$

that is trained to produce action vectors that maximize the reward. To use reinforcement learning with neural networks, we employ a technique known as deep Q learning. Deep Q learning works in environments like ours, with the caveat that the sequence of states $s_0, s_1, \ldots, s_t, s_{t+1}, \ldots, s_n$ is finite [1]. An environment that can be modelled purely on the idea that you have a state s_t and and policy π for choosing an action a_t that lets you transition to another state s_{t+1} is called a Markov decision process (MDP) and in our case, it is a finite MDP (FMDP). We consider a sequence of states, actions, and rewards to represent our MDP, written $s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_n$.

Let us consider R to be the total reward of such an FMDP

$$R = \sum_{i=0}^{n} r_i$$

and the total reward after some time t to be

$$R_t = \sum_{i=t}^{n} r_i$$

. If we use this to model the total expected reward of choosing an action a_t until the end of our FMDP, we may run into problems, namely that future rewards may not be exactly as expected. With this in mind, we may add a 'discount factor' $0 \leq \gamma \leq 1$ that minimizes the impact of future reward predictions the further they are into the future

$$R_t = \sum_{i=t}^{n} \gamma^{i-t} r_i$$

By considering the first term of the sum separately, we get

$$\begin{aligned} R_t &= \sum_{i=t}^{n} \gamma^{i-t} r_i \\ &= r_t + \gamma \sum_{i=t+1}^{n} \gamma^{i-t-1} r_i \\ &= r_t + \gamma R_{t+1} \end{aligned}$$

The Q function from Q-learning is simply a function

$$Q : S \times A \to \mathbb{R}$$

such that

$$Q(s_t, a_t) = \max R_{t+1}$$

which can be thought of as the maximum cumulative reward at the end of our FMDP when choosing

[1]Note here that n in the subscript is not the same as the dimension n of the encoded state vector

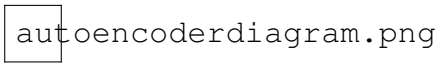

Fig. 1. The simplified diagram showing how the convolutional autoencoder compresses the imput state ($64 \times 64 \times 1$ into an encoded vector, and then deconvolves it to recreate the output.).

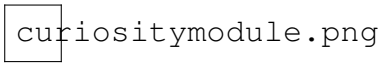

Fig. 2. The simplified diagram showing how the convolutional autoencoder compresses the imput state ($64 \times 64 \times 1$ into an encoded vector, and then deconvolves it to recreate the output.).

an action a_t. Then a policy π can simply defined to be

$$\pi(s_t) = \arg\max_{a_t} Q(s_t, a_t)$$

a greedy solution that always picks the action that maximizes the Q function.

Let us consider a single transition s_t, a_t, r_t, s_{t+1}. We can rewrite the Q function to be

$$Q(s_t, a_t) = r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

a form known as the *Bellman equation*. We could simply store the value $Q(s, a)$ in a table indexed by s and a and update it iteratively as we reach new states and choose new actions. However, considering the dimensionality of our input and output vectors, such a table would be impossibly large and sparse. In addition, it is useful to have an idea about the Q value for unexplored state-action pairs. For this reason, we decided to use a deep neural network to learn the Q function.

The advantage of using a neural network to represent the Q function is that, if the output dimensionality of the network is the same as the number of actions available to us, the network can simply take in the state s_t and produce a vector $\vec{Q}(s_t, a_t)$ whose entries are the predicted Q value for each action a_t. We can then train this network according to

$$\min \frac{1}{2} \| \underbrace{r + \gamma \max_a Q(s, a)}_{target} - \overbrace{Q(s, a)}^{prediction} \|_2^2$$

Recall our single transition s_t, a_t, r_t, s_{t+1}. We can use such a transition to train the network via the following:

1) Pass s_t into Q to get the predicted Q values for each action a_t.
2) Pass s_{t+1} into Q and select the action corresponding to $\arg\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$
3) Let the target $r + \gamma \max_a Q(s, a)$ that Q is minimizing be equal to the Q value of the best action found in step 2 for the chosen action a_t, and equal to the prediction for all other actions (so that their errors are zero and we only update our predictions for the given action).
4) Perform backpropagation.

One trick used by researchers when implementing Q-learning is to use 'experience replay', whereby all transitions s_t, a_t, r_t, s_{t+1} are stored in memory and the network is trained on randomly selected and shuffled minibatches from memory. The end result is that the Q network is able to not only predict the Q value for the chosen action, but also becomes better at predicting it for other actions as well.

Together, the curiosity module and policy network work together to simultaneously learn about the environment and to learn to choose actions that facilitate this.

III. Training and Results

We mentioned earlier that we begin by training the robot to emulate human performance so that it initially begins by selecting intelligent behaviours. To do this, we programmed the robot to be human controlled and trained the Q network to mimic human performance. Target action vectors were simply 1 when the user chose that action and 0 for all others. Since the policy π simply chooses the action corresponding to the highest output from the network, switching over from training on user input to training on Q values allows the robot to initially

Fig. 3. The simplified diagram showing how the convolutional autoencoder compresses the imput state ($64 \times 64 \times 1$ into an encoded vector, and then deconvolves it to recreate the output.).

make decisions that imitate those of humans, but then begin to learn Q values instead.

Human training involved 4 2-hour sessions (total 2,000 epochs $\times$ 32-sample minibatches $\times$ 15,000 memories in experience replay). It used 4 different humans who were given time to get used to the robot controls, and then instructed to try to cover the most ground possible and see the most things possible with the robot (ie; explore). The robot was trained in various environments, including hallways in a building and inside two houses. We let the robot make predictions at the same time to collect information about the error between the user control and the prediction error and it would train in the background while the operator drove (see figure 4).

Fig. 4. Training loss from the Q network during human training. Loss is calculated as the average over 100 epochs. Every 500 epochs we would update the person driving the robot and recharge the battery. The loss increases when the user changes due to potentially different driving styles (though many things stay the same such as driving straight down the hallway). The network decreases in loss over time and becomes better at predicting human commands over time.

We trained an autoencoder using images saved during the training process, partitioning saved images into training sets and testing sets. Overall, 50,000 images were saved at a $64 \times 64 \times 1$ greyscale resolution and used for training the network for 1600 epochs. Human-controlled behaviour resulted in successful exploration and thus allowed for the collection of a wide variety of state samples. This is important since we cannot be sure that the robot will explore enough states sufficiently to learn good representations of the environment. By the time the robot actually begins using curiosity to drive behaviours, the autoencoder will already be trained enough such that predictions about encoded states actually make sense and lead to intelligent behaviour sooner. Figure 5 displays the autoencoder loss over time.

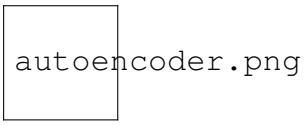

Fig. 5. The training and testing loss from the autoencoder. Over time, the training loss appears to slightly stabilize, tapering off to a loss that plateaus around 300-500. Such loss is unavoidable due to find details or slight differences in input that cannot be perfectly recreated. The testing loss does not stabilize completely, but is definitely reduced over time.

The autoencoder testing loss does not really converge, with performance departing from the training loss at around 300 epochs and oscillating around the approximate loss of 700. This is possible evidence of our model overfitting as the training loss continues to decrease. Adjusting the size of the encoded vector (default of 32) did not improve performance and the testing loss does not seem to converge in general (data not shown). This is possibly due to the fact that our environments are very different from eachother except for some basic geometric similarities. Indeed, autoencoders are best at forming compressed representations only for highly structured and similar data, and perform poorly over wide ranges of inputs. We attempted to account for this originally by switching our input from RGB (3 channels) to greyscale (1 channel), but this was apparently not enough to produce highly successful results. Additionally, because we used a convolutional autoencoder and due to how such autoencoders perform upsampling, we tend to lose fine resolution resulting in an image that looks invariably blurred, which may have contributed to the constant training loss and oscillating testing loss.

We trained the robot in 8 2-hour sessions. Similarly to the human-controlled learning trials, the curiosity learning phase was restricted to 2 hour sessions due to the battery life of the robot. It

essentially consisted of releasing the robot while keeping an eye on it to ensure it didn't encounter any dangerous scenarios. We used a total of 1600 epochs. It trained on-line using an experience replay memory size of 128. Figure 6 shows the training loss from the policy network.

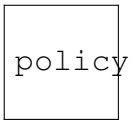

Fig. 6. The average training loss for the policy network over 100 epoch intervals. It shows that over time this network gets better at successfully predicting the reward of choosing different actions. Eventually the network learns which actions are the best to take (ie; maximize prediction error).

The network loss seems to continually decrease over time. This is interesting as it indicates that it is possible that the network might be able to continue learning the Q function if we continue to train it. This is a possible future direction of study.

Part of the difficulty of analysing the outcome of our curiosity-driven agent is defining a quantitative notion of success. There are two ways of measuring success that we discussed. The first is the ability of the robot to maintain high prediction error over time. Our reasoning is twofold. First, the change in encoded state that the robot observes is highly causal, depending heavily on the action taken. Thus, when the prediction network is training to reduce prediction error in encoded states, it is using structured data and is converging towards something when it updates weights. Thus, high prediction error is not due to a poor network. This leads to the second reason; that if the network is continually learning about the world and using encoded states, it must be choosing actions that maximize prediction error. Thus, over the course of training, maintenance of high prediction error indicates that the agent is behaving curiously. Figure 7 demonstrates that such a metric shows our agent is being curious.

This indicates that the robot has high prediction error and is thus behaving curiously. We initially considered the fact that prediction error may be due to the fact that the autoencoder is not encoding the input states in a useful manner and thus the output is unpredictable. However, since the states s_t and s_{t+1} are very similar, their encoding should also be

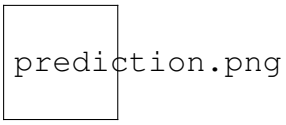

Fig. 7. The average training loss for the prediction network over 100 epoch intervals. Initially it appears that the prediction error decreases, but it rapidly stops decreasing and ends up staying relatively constant throughout the entire training period.

similar (since the neural network is a differentiable function, this is the case [2]) and as such the prediction network should be able to learn to produce outputs that are at least similar to the inputs, such that their loss is smaller than what was observed.

The second, less quantitative notion of successfully curious behaviour is to count the number of rooms explored in an environment with open rooms coming from a corridor. The robot was reset to an initial starting position every 100 epochs for a total of 16 runs. Out of 6 similarly sized rooms, all with open doors and lights on, all adjacent to the same hallway the robot is initially placed in, our robot consistently explored at least 4 rooms for a total of 76 rooms ($M = 4.75,\ SD = 0.683$). Let us assume we have two distributions, one a binomial distribution with $p_{entering} = 0.5,\ n_{trials} = 96$ [3] and one our sample distribution with $p = \frac{76}{96} = 0.792,\ n = 6{*}16 = 96$. A z test for proportions tells us that our robot performs better than randomly with $p < 0.0001$ ($z = 4.2$). This is a strong indication that our robot performs very well (especially since a random probability of entering a room of 0.5 is generous).

IV. Conclusions and Future Work

We designed and implemented a series of neural networks that together encode the environment, learn about how actions affect the environment, and learn to choose actions that help us learn about the environment. We made a robot that is curious and behaves as a reflexive agent with a brain capable of making intelligent decisions. The curious behaviour

[2]All continuous, differentiable functions have the property that a small change in input results in a small change in output, as formalized in the espilon-delta notion of a limit.

[3]Note however that the assumptions for a binomial distribution are not entirely met, namely the events are not independent. If you enter a room and get stuck, it is harder to enter another room.

of the robot was demonstrated through its ability to maintain a high prediction error in its curiosity module, despite the fact that it was learning simultaneously.

Two comparisons that might have strengthened our argument are to allow the curiosity module to operate during human training, to show that under simple controls the robot is capable of getting better at predicting things in general. This would make our argument that maintained prediction error indicates curiosity much stronger. Also, we could have observed the robot behaviour with and without pretraining and with and without curiosity reward to see whether the robot can learn to behave curiously without human guidance, and whether human guidance alone is enough for the robot to behave well.

One very interesting future direction that we intended to work on for this project but never finished motivated the use of an autoencoder for our state dimensionality reduction. By predicting the encoded next state, one could apply the decoder ϕ' to recover $\phi'(\theta(\phi(s_t), a)) \approx s_{t+1}$ and feed this back into the input of the networks. This could let the robot operate in an internally generated model of the world, without having to interact with the environment. For robots that are mobile, this is useful since robots can be dangerous and expensive to operate in the real world. This proposed model for internally learning using an internal world model we refer to as 'dream learning' and is where we intend to further our research in this area.

REFERENCES

[1] E. L. Thorndike, "Animal intelligence: An experimental study of the associative processes in animals," *Psychological Review Monograph Supplement*, no. 2, pp. 1–109, 1901.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidgeland, and G. Ostroski, "Human-level control through deep reinforcement learning," *Nature*, no. 518, pp. 529–533, 2015.

[3] G. Dam, K. Kording, and K. Wei, "Credit assignment during movement reinforcement learning," *PLOS ONE*, no. 8.

[4] D. E. Berlyne, "The arousal and satiation of perceptual curiosity in the rat," *Journal of Comparative and Physiological Psychology*, no. 48, pp. 238–246.

[5] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," 2015.

[6] J. C. H. C Y Liou and W. C. Huang, "Modelling word perception using the elman network," *Neurocomputing*, no. 71, pp. 3150–3157, 2008.