Replicating Mouse Movements via Inverse Reinforcement Learning

Anna Brandenberger[†] anna.brandenberger@mail.mcgill.ca
Gavin McCracken[†‡] gavin.mccracken@mail.mcgill.ca
School of Computer Science, McGill University[†]*; MILA*[‡]
For Prof. Doina Precup

1 Introduction

More and more examples of the usage of mouse movements for identity verification have been appearing in the private sector each year [9, 19]. This is because they work quite well; in fact, a paper in 2008 was able to use the support vector machine algorithm to map mouse movements to the person that created them with a 50% improvement over randomly guessing [17]. Inspired by similar successes, the literature has grown to show ample applications in the domains of bot detection and of identity verification.

Bot detection is widely used to protect applications and websites from spamming or other malicious content. For instance, Google's free software reCAPTCHA is a test deployed on websites that aims to tell apart humans from bots [13]. The most recent version of reCAPTCHA, "v3", aims to improve user experience by simply requiring for the user to click on a box, instead of solving an image puzzle – a task which has been defeated by automated software and generative adversarial networks (GANs) [6, 23, 31, 33]. This new reCAPTCHA determines whether the user is a bot by analyzing the user's mouse movement trajectory as they check the box [13, 14] (amongst other information). Another important use of bot detection via mouse movements is in online gaming. Given the negative impact of automated bots or cheating software on fair gameplay, it is essential for online games to be released with cheat detecting software. An additional constraint is that the detection must be done in a non-interactive way so as not to interrupt a users gameplay. This is satisfied by using observational detectors that make decisions by analyzing user actions which are deemed to be difficult for bots to perform like a human, such as mouse movements [11, 25].

Mouse movements have also been used for identity detection in security systems, i.e. to identify a specific user [2, 17, 26, 27, 34]. Online banking is one such use case, where certain banks monitor the user's mouse dynamics to confirm their identity as they log into online banking [7]. Identity detection (and bot detection) algorithms are trained from the mouse movement data via supervised learning classification and clustering algorithms, ranging from very basic k-Nearest-Neighbour [17] to support vector machines (SVM) [17, 26, 34] to various forms of neural networks [2, 24, 27, 29], yielding very accurate discrimination.

Another commonly used behavioural biometric used to identify people is keystroke dynamics [5, 18, 22], i.e. the user's typing rhythm. The arguments presented by Joyce and Gupta [16] and Monrose and Rubin [22] on the use of keystrokes to identify users translate equivalently to mouse movements. Indeed, the neuro-physiological factors that make handwritten signatures unique also manifest themselves in people's keystroke patterns and their manner of moving the mouse. Luc Devroye, an expert on probabilistic processes and typography, agreed that mouse movements "...are like a written signature from someone"; due to their probabilistic nature resulting from various physiological factors i.e. different hand shapes, natural shaking of the hand, blood pressure, different preferences in holding a mouse, etc. [8].

Thus, we propose two interesting questions. Firstly, is it possible to use a GAN or reinforcement learning to train an agent to generate "human-like" trajectories? If it is possible; then

perhaps the movement of software like reCAPTCHA towards mouse movements for bot detection is a waste of developer time and company funding. Secondly, is it possible to "imitate" a human being, i.e. replicate a specific person? This possibility of replication immediately raises ethical questions since companies such as Facebook track and store users' mouse movements without "de-identifying" them [15]. To Facebook, your mouse movement data is something to feed into a deep network in an attempt to optimize their advertising revenue. For users, the importance of mouse movements as an identity marker means that if mouse movement data collected by these companies is leaked and retrieved by people with malicious intent, one further layer of security surrounding online banking is removed (for Facebook, such data breaches regularly occur [30, 20]).

2 Problem Formulation

A mouse movement is a trajectory through time consisting of many points (x, y, t). They can be modelled as a Markov Decision Process (MDP) since they have the Markov property: given a previous state at time t, the next state at time $t+1$ must be both near it and possible to reach using an action from the previous state. It thus makes sense to view the problem of generating such trajectories as an agent that, at each state in its trajectory, chooses the next state in the trajectory by stepping to a nearby state via some action, and receives some reward based on (state, action, next-state) transition. Now with this MDP, it's possible to apply the context of reinforcement learning.

A formulation of the learning task is thus: given an initial point (x_0, y_0) and a target point (x_T, y_T), we want to generate a "human-like" mouse trajectory ζ connecting the points.

We begin by giving a few reasons this problem is challenging. First, to give an idea of the colossal size of the state-space, consider a standard 1920×1080 px monitor and a trajectory lasting 1000 timestamps. The space required for a dynamic solution would be:

$$\begin{aligned}\text{Space} &= (\text{start\_x} \times \text{start\_y}) \times (\text{current\_x} \times \text{current\_y}) \times (\text{target\_x} \times \text{target\_y}) \times (\text{time-stamps}) \\ &= (1920 \times 1080) \times (1920 \times 1080) \times (1920 \times 1080) \times (1000) \times (\text{32-bit}) \\ &= 3.566 \times 10^{22} \text{ bytes} = 35660 \text{ EB (exabytes)}\end{aligned}$$

With such a large state space, dynamic programming based learning methods are ruled out and the difficulty is further incremented by a continuous action space spanning through time. We additionally remark below on Figure 1, which gives a few intuitions about the complexity of the underlying random function that "generates" authentic human mouse movements and qualitatively shows that mouse movement trajectories have many potential probability modes.
Figure 1:

- The top left is a 2-d histogram recording the number of times the mouse was at each position (x, y). Notice that as expected, most trajectories lie approximately between the start and end points, but many trajectories with curvature exist.
- The top right 2-d histogram shows the average velocity of the mouse in each position (x, y). On the straight line between start and end points, the velocity is comparatively low; but in the surrounding areas we see very high velocities. This supports our intuition that the mouse slows down as near the endpoint, and that trajectories of the mouse that begin away from the end point of the trajectory may have higher speeds.
- The bottom left 2-d histogram is the expected direction of the velocity on a position. Dark red means the mouse is going straight up, and dark blue means the mouse is going straight down.

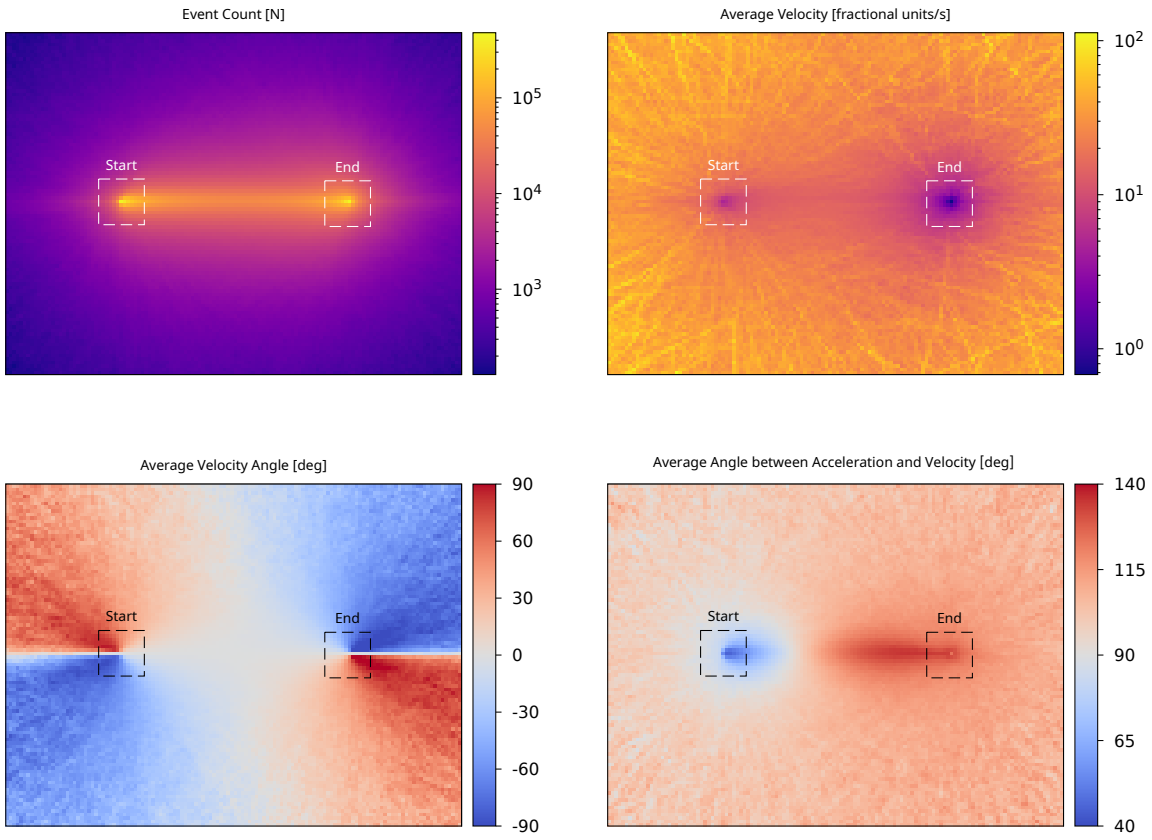

Figure 1: Top left: a histogram of event counts. Top right: average velocity. Bottom left: Expected direction. Bottom right: average angle between acceleration and velocity. This figure was generated from a data-set of recorded human mouse trajectories, which were normalized and rotated such that all trajectories started at the point $(0, 0)$ and ended at $(1, 0)$.

- The bottom right 2-d histogram shows the angle between the velocity and the acceleration. Recall that in perfect circular motion, the acceleration is applied perpendicular to the velocity (at a right angle towards the center of a circle). Note: the white oval around the start point shows where mouse movements are expected to be moving in a circle.

Naturally, given the complexity of the task, hand-designing a reward function encouraging the agent to output reasonable mouse trajectories is very challenging. A few examples of trajectories generated by agents trained on mis-specified reward functions are shown in Figure 2.

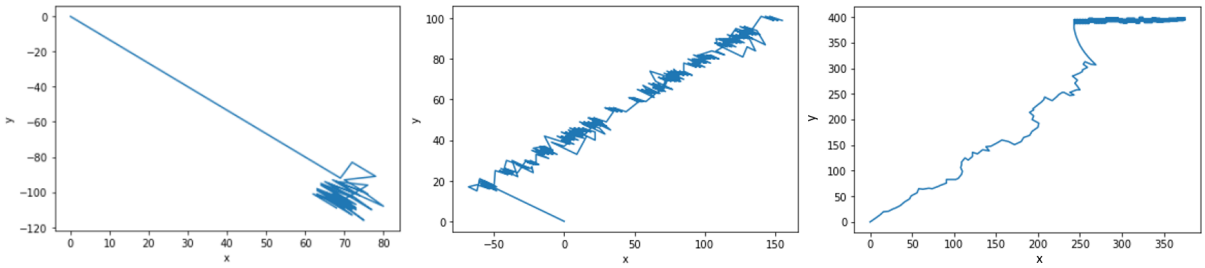

Figure 2: Example mouse movement trajectories generated from incorrectly specified reward functions. For example, in the leftmost plot we can see that the incorrectly specified reward caused the agent to learn to output movements that are as long as possible, and thus it can't hit it's target point (alg: actor-critic policy gradient [32]). In the centre plot, the reward was intended to penalize the agent for not decreasing it's distance towards the target, so it learned to take as many diagonal steps as possible (alg: actor-critic policy gradient [32]). The right plot, was an attempt to make movements move more circularly (alg: Q-Learning [32]).

This gave rise to the idea to attempt using inverse reinforcement learning. Multiple gigabytes of expert mouse movement trajectories were recorded and parsed via root-kit hooks to Windows and Linux operating systems. We use the inverse reinforcement learning paradigm to recover a reward function from these expert policies. We build on the maximum entropy inverse optimal control framework [35] by following literature for applications of IRL to larger state and action spaces (Sections 3.2, 3.3). We train an agent using guided cost learning ([10], detailed in section 3.3) to return a policy for outputting "human-like" mouse movement trajectories.

3 Algorithms

Before discussing the specific implementation of the aforementioned algorithms to our specific setting, we present an overview of the theory, especially focusing on the maximum entropy formulation of inverse reinforcement learning (section 3.2) and the guided cost learning algorithm (section 3.3).

3.1 Generative Adversarial Networks

We began by attempting to solve the trajectory generation problem using generative adversarial networks (GANs) [12]. In short, we train two neural networks simultaneously where one network, the discriminator, estimates the probability that a sample generated by the other neural network, the generator, is actually an expert trajectory.

In practice we found that this did not perform as expected, see section 5. Therefore we move to address this through RL where, as mentioned previously, we attempt to solve the problem of designing a correct reward function by using inverse reinforcement learning.

3.2 Maximum Entropy Inverse Reinforcement Learning

Ziebart et al. [35]'s formulation of the IRL problem relies on the principle of maximum entropy to remove the inherent ambiguity of the IRL problem statement.

Let a Markov Decision Process (MDP) be defined as $M = \langle S, A, T, r, \gamma \rangle$. S denotes the state space, A the action space, and $T : S \times A \to \mathbb{R}$ denotes the transition distribution, where $T_a(s, s')$ is the probability $P_a(s_{t+1} = s' \,|\, s_t = s)$ of transitioning to state $s' \in S$ from state $s \in S$ via action $a \in A$. r denotes the reward function and $\gamma \in [0, 1]$ is the discount factor. An optimal policy π^* is one which maximizes the expected cumulative reward. In the inverse reinforcement learning (IRL) setting, the goal is to recover the reward function given a set of N expert demonstrations $D = \{\tilde{\zeta}_1, \tilde{\zeta}_2, \ldots, \tilde{\zeta}_N\}$ consisting of state action pairs $\tilde{\zeta}_i = \{(s_0, a_0), \ldots, (s_T, a_T)\}$, with T the length of the trajectory.

In most of the original papers presenting theory behind IRL [1, 35], the authors assume a linear reward function parametrized by weights θ, which maps the sum of state features $\mathbf{f}_{s_j} \in \mathbb{R}^k$ along a trajectory ζ, the state feature count $\mathbf{f}_\zeta = \sum_{s_j \in \zeta} \mathbf{f}_{s_j}$, to a reward value

$$R(\theta, \zeta) = \theta^T \mathbf{f}_\zeta = \sum_{s_j \in \zeta} \theta^T \mathbf{f}_{s_j}. \tag{1}$$

Given the set of expert trajectories $D = \{\tilde{\zeta}_i\}$, Abbeel and Ng [1] show that for the learning agent to achieve the same performance as the expert (assuming the expert were indeed solving an MDP with reward function as defined in (1), then it is both necessary and sufficient for the following feature expectation matching to hold:

$$\sum_{\text{paths } \zeta_i} P(\zeta_i) \mathbf{f}_{\zeta_i} = \tilde{\mathbf{f}} \tag{2}$$

where $\tilde{\mathbf{f}} = \frac{1}{N} \sum_i \mathbf{f}_{\tilde{\zeta}_i}$ is the expected feature count of the N expert trajectories.

This feature count matching still is degenerate, as a policy can be optimal for many reward functions, and many policies can lead to the same feature expectations. One manner of dealing with this ambiguity is the maximum entropy model presented by Ziebart et al. [35]. They choose the reward distribution that doesn't exhibit a preference for any path over others that isn't implied by feature expectation matching: equation 2. Under this formulation, we assume that the expert demonstrations all sample trajectories ζ_i from a Boltzmann distribution. This model is parametrized by the reward function weights θ, such that paths with high reward under the current parametrization of the reward function R are exponentially preferred:

$$P(\zeta_i \mid \theta) = \frac{1}{Z(\theta)} e^{R(\theta, \zeta_i)} \tag{3}$$

where $Z(\theta)$ is the partition function serving to normalize the probability $P(\zeta_i \mid \theta)$.

This equation is for the case of a deterministic MDP. In the more general case of non-deterministic MDPs, actions produce non-deterministic transitions according to a state transition distribution T. A similar equation as (3) can be written by making the approximation that the partition function is constant for all action outcomes, and the probability of an action will be weighted by the expected exponentiated reward of all paths beginning with that action. We will not present that case since in the mouse movement setting we are considering, the actions are deterministic.

To recover the reward function, we write the parameter θ as maximizing the likelihood of

the expert data $D = \{\tilde{\zeta}_i\}$ under the maximum entropy distribution (3):

$$\theta^* = \arg\max_\theta L(\theta) = \arg\max_\theta \frac{1}{N} \log P(D \mid \theta) = \arg\max_\theta \frac{1}{N} \log \prod_{\tilde{\zeta}} P(\tilde{\zeta} \mid \theta)$$
$$= \arg\max_\theta \frac{1}{N} \sum_{\tilde{\zeta}} \log P(\tilde{\zeta} \mid \theta) \tag{4}$$

We can show that this function is convex, and thus use gradient-based optimization to find θ^*. We explicitly show this convexity in the appendix (Claim A.1), and calculate the gradient to be

$$\nabla L(\theta) = \frac{1}{N} \sum_\zeta \mathbf{f}_\zeta - \sum_\zeta P(\zeta \mid \theta) \mathbf{f}_\zeta = \tilde{\mathbf{f}} - \sum_{s_i} D_{s_i} \mathbf{f}_{s_i} \tag{5}$$

where $\tilde{\mathbf{f}}$ is the expected empirical feature counts and $D_{s_i} = P(s_i \mid \theta)$ are the expected state visitation frequencies.

The time consuming part of this algorithm is the calculation of the expected state frequencies D_{s_i}. Ziebart et al. [35] present the following dynamic programming based algorithm. It decomposes D_{s_i} into timesteps $D_{s_i,t}$: the probability of visiting state s at time t.

- Backward pass: solve the MDP for $\pi(a \mid s)$ for the current value of the parameter θ using value iteration.
- Forward pass: first set $D_{s,t=1} = P(s = s_i)$, then compute the state frequencies for each time step $t = 1, \ldots, N$

$$D_{s_{i,t+1}} = \textstyle\sum_{a_{i,j}} \sum_k D_{s_k,t} \pi(a_{i,j} \mid s_k) P(s_k \mid a_{i,j}, s_i)$$

- The total state visitation frequency is then the sum over all timesteps $D_{s_i} = \sum_t D_{s_i,t}$

The IRL algorithm to find the optimal cost function parameter θ is then simply a gradient descent algorithm:

MaxEnt-IRL(D)
1 Initialize reward weights θ
2 Repeat:
3 Solve for $\pi(a \mid s)$ for the current θ and compute the state visitation frequencies D_{s_i}
4 Compute the gradient $\nabla L(\theta)$ according to equation (5)
5 Update θ with one gradient step according to $\nabla L(\theta)$

The advantages of the maximum entropy formulation include, aside from of course removing the ambiguity in the feature expectation matching formulation, the ability to handle expert suboptimality and stochasticity, and leading to the formulation of the training procedure as a simple gradient optimization task.

However, in the basic form as presented in [35], the forward pass computation of the policy and state visitation frequencies D_{s_i} is intensive in space and time, and thus this dynamic programming based approach does not scale well to large state and action spaces, such as our mouse movement problem. We therefore look at the guided cost learning method [10] which builds on Ziebart et al. [35]'s ideas of the maximum entropy cost distribution, but is designed to be applicable in complex, high-dimensional systems.

3.3 Guided Cost Learning

Another advantage to guided cost learning is that they remove the assumption that the reward function is linear, which was required in previous work [1, 35]. The cost can now be represented using nonlinear function approximators such as deep neural networks. To do so, Finn et al. [10] generalize the maximum entropy inverse optimal control (IOC) form as seen in section 3.2 to nonlinear cases, with samples used to approximate the partition function Z.

Intuitively, guided cost learning works by using a sample-efficient algorithm to directly optimize a trajectory distribution with respect to the current cost. Instead of solving for the optimal policy during each step in the cost function training as in the previous section, the two are now simultaneously learned. The partition function is estimated using importance sampling, where the sampling distribution is "guided" by the optimization step toward regions where the samples are more useful, to match the maximum entropy cost distribution.

Let's begin by reformulating the negative log-likelihood $L(\theta)$ from equations (4) and (5) to work for a general reward function form. Finn et al. [10] consider a general cost function (as opposed to reward function) $c_\theta(\zeta) = \sum_i c_\theta(s_i, a_i)$ parametrized by weights θ, where s_i and a_i are the state and action at time step i in the trajectory ζ. The cost function form is the only change from the previous section – we are still using the maximum entropy model (3):

$$P(\zeta_i \mid \theta) = \frac{1}{Z} e^{-c_\theta(\zeta_i)}$$

where $Z = \int e^{-c_\theta(\zeta)} d\zeta$ is the partition function which will be estimated via importance sampling.

The loss function is now written and estimated as follows

$$\begin{aligned} L(\theta) &= \frac{1}{N} \log P(D_{\text{demo}} \mid \theta) = \frac{1}{N} \sum_{\tilde{\zeta}_i \in D_{\text{demo}}} c_\theta(\tilde{\zeta}_i) + \log Z \\ &\approx \frac{1}{N} \sum_{\tilde{\zeta}_i \in D_{\text{demo}}} c_\theta(\tilde{\zeta}_i) + \log \frac{1}{M} \sum_{\zeta_j \in D_{\text{samp}}} \frac{e^{-c_\theta(\zeta_j)}}{q(\zeta_j)} \end{aligned} \tag{6}$$

where D_{demo} is still the set of N expert trajectories and D_{samp} the set of M samples from the background distribution q. The gradient $\nabla L(\theta)$ is computed by considering $w_j = \frac{1}{q(\zeta_j)} e^{-c_\theta(\zeta_j)}$:

$$\nabla L(\theta) = \frac{1}{N} \sum_{\tilde{\zeta}_i \in D_{\text{demo}}} \frac{\partial c_\theta}{\partial \theta}(\tilde{\zeta}_i) - \frac{1}{Z} \sum_{\zeta_j \in D_{\text{samp}}} w_j \frac{\partial c_\theta}{\partial \theta}(\zeta_j) \tag{7}$$

Similarly to the linear case, we can use this form of the loss to optimize the IOC objective using gradient optimization algorithms. However, if the cost is now represented by a neural network, $\frac{1}{N}$ and $-\frac{w_j}{Z}$ must be backpropagated respectively for trajectories belonging to D_{demo} and trajectories sampled from D_{samp}.

The choice of background distribution $q(\zeta)$ is critical to the success of the algorithm, and many possible choices can be made (e.g. a uniform distribution or a distribution lying near the expert distribution). Finn et al. [10] instead adapt $q(\zeta)$ as training progresses, to generate more samples in regions of trajectory space that are good according to the current cost $c_\theta(\zeta)$. The IOC step (4) of Guided-Cost-Learning is done using stochastic gradient methods by sampling subsets of demonstration and background samples at each iteration. As part of this step, the estimation of the partition function Z is done using importance sampling. At iteration k, we have samples from k distributions $q_1(\zeta), \ldots, q_k(\zeta)$. An estimator of $\mathbb{E}\big[f(\zeta)\big]$ for a function

$f(\zeta)$ can be constructed as $\mathbb{E}\big[f(\zeta)\big] \approx \frac{1}{M}\sum_{\zeta_j}\big(\frac{1}{k}\sum_{\kappa} q_\kappa(\zeta_j)\big)^{-1} f(\zeta_j)$. The importance weights are thus $z_j = \big(\frac{1}{k}\sum_{\kappa} q_\kappa(\zeta_j)\big)^{-1}$, and the objective $L(\theta)$ can be rewritten as

$$L(\theta) = \frac{1}{N}\sum_{\tilde{\zeta}_i \in D_{\text{demo}}} c_\theta(\tilde{\zeta}_i) + \log \frac{1}{M}\sum_{\zeta_j \in D_{\text{samp}}} z_j e^{-c_\theta(\zeta_j)}.$$

The guided cost learning algorithm is thus:

GUIDED-COST-LEARNING(D_{demo})

1 Initialize the background distribution $q_k(\zeta)$
2 **for** iterations $i = 1$ to I:
3 Generate sample trajectories D_{traj} from $q_k(\zeta)$ and append them to D_{samp}
4 Use D_{samp} to update the current cost c_θ according to NONLINEAR-IOC
5 Update $q_{k+1}(\zeta)$ using D_{traj} according to a forward policy optimization algorithm
6 Return: optimized parameters θ and trajectory distribution $q(\zeta)$

NONLINEAR-IOC

1 **for** iterations $k = 1$ to K:
2 sample batch $\hat{D}_{\text{demo}} \subset D_{\text{demo}}$
3 sample batch $\hat{D}_{\text{samp}} \subset D_{\text{samp}}$
4 Append demonstration to background: $\hat{D}_{\text{samp}} = \hat{D}_{\text{demo}} \cup \hat{D}_{\text{samp}}$
5 Estimate $\nabla L(\theta)$ using samples $\hat{D}_{\text{demo}}$ and $\hat{D}_{\text{samp}}$
6 Update θ using $\nabla L(\theta)$
7 Return optimized parameters θ

Note that as required, the guided cost learning algorithm outputs both a learned (potentially nonlinear) cost function $c_\theta(s_i, a_i)$ and a trajectory distribution $q(\zeta)$ – which corresponds to a controller $q(a_i \mid s_i)$ used by the agent to execute the trajectories.

3.4 Implementation

For our mouse movement MDP, we registered an MDP of our own design into the gym AI environment; we then implemented guided cost learning using a neural network cost function approximator. Note however, that we plot episode reward on Fig. 4, which we defined as $reward = 100 \times (1 - cost)$, where the cost of a trajectory was normalized to be in [0, 1]. We used a 400×400 square as the state space, which is much smaller than the average monitor size, 1080×1920, but adequate enough. Previous work fooling reCAPTCHA used a divide and conquer method where 100 by 100 squares were solved by reinforcement learning, and "chained" together [3].

4 Results

We reproduced the classic GAN [12], and found a GitHub implementation for the Wasserstein GAN [4]. In our experiments, they could converge on other datasets, such as MNIST and CIFAR 10, but could not converge on the mouse movement data. We feel as though Wasserstein GAN should converge, however, as this is an RL project, we decided to investigate RL methods.

Figure 4 shows a plot of the reward gathered by the agent from the neural network cost function as a function of training, and Figure 5 shows examples of mouse movement trajectories generated by the agent.

5 Discussion

Attempting to hand-engineer a reward function that worked with Q-Learning, deep-Q-networks or actor-critic to generate reasonably "human" trajectories was a fruitless endeavor in our experiments. However, this doesn't mean it's impossible: recent literature shows that reCAPTCHA

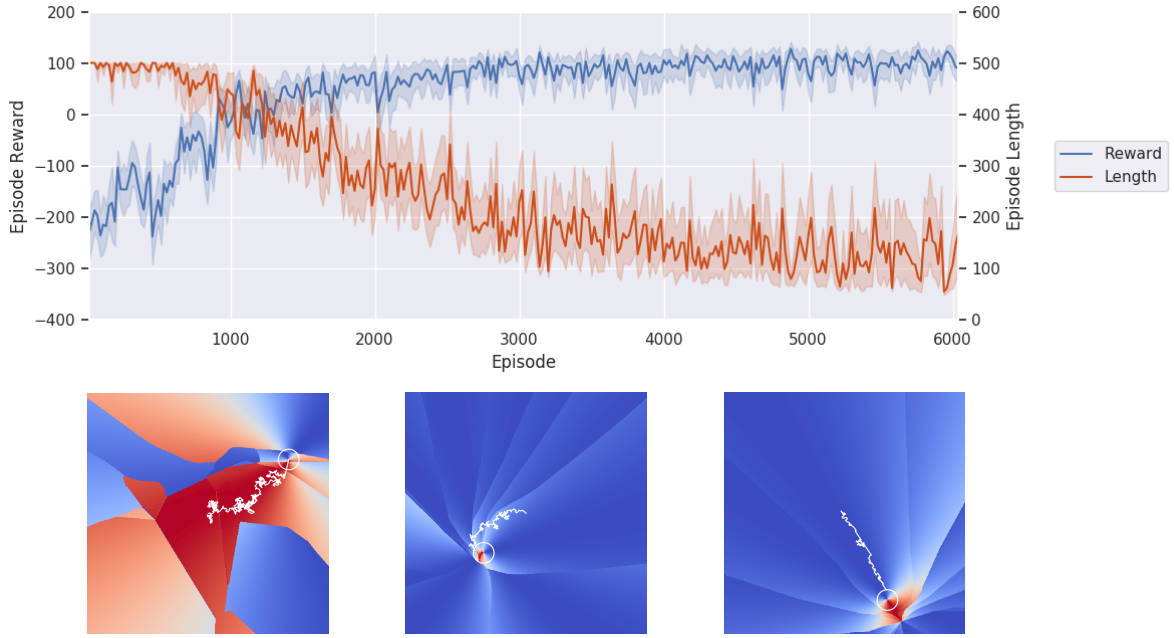

Figure 3: Top: preliminary actor-critic results when learning a simple cost function encouraging the agent to travel towards the target in a straight line. Bottom: plots of selected episode trajectories (white) and angle between actor decision and target versus position (blue towards, red away). Left to right are an early episode for large epsilon, an episode after learning has started, and one close to convergence.

Figure 4: Reward gathered by the guided cost learning agent and trajectory length as a function of training episode.

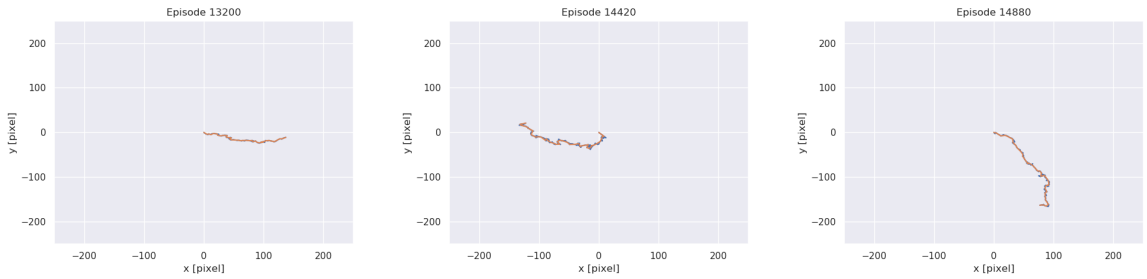

Figure 5: Example mouse movements generated by the guided cost learning method towards the end of its training.

can be tricked with reinforcement learning, without using IRL [3]. Similarly, training a GAN to generate mouse-movements proved to be difficult, and our experiments found that the discriminator always learned too quickly. We suspect that this is because, as discussed in section 2, there exist so many possible probability modes that the generator can't learn anything meaningful before the discriminator has learned to discriminate with such high confidence that the reward signal is lost. Generating data from a distribution with many modes is indeed a known problem in training GANs [21, 28]. Generally, in our experiments, the GAN would generate sets of points that were in regions in which the discriminator had low confidence (it classifies with 50% probability), and not points that were similar to the human data.

We implemented Max entropy IRL; however, Ziebart et al. [35] use a dynamic programming approach, meaning that even a tiny state space requires terabytes of RAM:

$$\begin{aligned} \text{RAM} &= (x, y)^3 \times 32\text{bits} \times (actions) && (8)\\ &= (100 * 100)^3 \times 32\text{bits} \times 16 && (9)\\ &= 64 \text{ TB} && (10) \end{aligned}$$

This led us to try using Guided Cost Learning, Fig. 4, which was able to learn "human-like" mouse movements with neural network cost. Our implementation involved expert trajectories from only one human, with the cost function being a neural network that was shown to be powerful enough to distinguish between humans and cubic splines being fit through human data in previous unpublished work (COMP 652 class project).

6 Future Direction

We are very interested in determining if it is possible to "mimic" a specific human individual to the extent that identity verification neural networks can be fooled [29]. Our preliminary results with guided cost learning imply that it might be possible, however, Professor Precup advised us that we would need to get ethics approval before performing tests on other peoples mouse movements. A future experiment will involve using data from several humans and attempting to train an agent for each human expert, that is capable of imitating them. These agents will then be tested against an identity verification neural network that was trained on real human data.

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[2] Ahmed Awad E. Ahmed and Issa Traore. A New Biometric Technology Based on Mouse Dynamics. *IEEE Transactions on Dependable and Secure Computing*, 4(3):165–179, jul 2007. ISSN 1545-5971. doi: 10.1109/TDSC.2007.70207. URL `http://ieeexplore.ieee.org/document/4288179/`.

[3] Ismail Akrout, Amal Feriani, and Mohamed Akrout. Hacking google recaptcha v3 using reinforcement learning. *arXiv preprint arXiv:1903.01003*, 2019.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[5] Francesco Bergadano, Daniele Gunetti, and Claudia Picardi. User authentication through keystroke dynamics. *ACM Trans. Inf. Syst. Secur.*, 5(4):367–397, November 2002. ISSN 1094-9224. doi: 10.1145/581271.581272. URL http://doi.acm.org/10.1145/581271.581272.

[6] Kumar Chellapilla and Patrice Y Simard. Using machine learning to break visual human interaction proofs (hips). In *Advances in neural information processing systems*, pages 265–272, 2005.

[7] Stacy Cowley. Banks and Retailers Are Tracking How You Type, Swipe and Tap. *The New York Times*, August 2018. URL https://www.nytimes.com/2018/08/13/business/behavioral-biometrics-banks-security.html.

[8] Luc Devroye. Private Discussion, April 2019.

[9] eSportbet.com. eSports Anti-Cheating Software. eSportsanti-cheatingsoftware, 2017. Accessed 25 April 2019.

[10] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization. *arXiv:1603.00448 [cs]*, March 2016.

[11] Steven Gianvecchio, Zhenyu Wu, Mengjun Xie, and Haining Wang. Battle of botcraft: fighting bots in online games with human observational proofs. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 256–268. ACM, 2009.

[12] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *arXiv e-prints*, art. arXiv:1406.2661, Jun 2014.

[13] Google. reCAPTCHAv3. https://developers.google.com/recaptcha/docs/v3, 2019.

[14] Andy Greenberg. Google Can Now Tell You're Not a Robot with Just One Click. *Wired*, December 2014. URL https://www.wired.com/2014/12/google-one-click-recaptcha/.

[15] Facebook Inc. Official Answers to USA Senator Questions. https://www.commerce.senate.gov/public/_cache/files/ed0185fb-615a-4fd5-818b-5ce050825a9b/62027BC70720678CBC934C93214B0871.senate-judiciary-combined-7-.pdf, 2018.

[16] Rick Joyce and Gopal Gupta. Identity authentication based on keystroke latencies. *Communications of the ACM*, 33(2):168–176, 1990.

[17] Ryan Kaminsky, Miro Enev, and Erik Andersen. Identifying game players with mouse biometrics. *University of Washington, Tech. Rep*, 2008.

[18] M. Karnan, M. Akila, and N. Krishnaraj. Biometric personal authentication using keystroke dynamics: A review. *Applied Soft Computing*, 11(2):1565–1573, mar 2011. ISSN 1568-4946. doi: 10.1016/J.ASOC.2010.08.003. URL https://www.sciencedirect.com/science/article/pii/S156849461000205X.

[19] Evan Lahti. Valve has 1,700 CPUs working non-stop to bust CS:GO cheaters. https://www.pcgamer.com/vacnet-csgo/, March 2018. Accessed 25 April 2019.

[20] Louise Matsakis and Issie Lapowsky. Everything We Know about Facebook's Massive Security Breach. *Wired*, September 2018. URL https://www.wired.com/story/facebook-security-breach-50-million-accounts/.

[21] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *CoRR*, abs/1611.02163, 2017.

[22] Fabian Monrose and Aviel D. Rubin. Keystroke dynamics as a biometric for authentication. *Future Generation Computer Systems*, 16(4):351 – 359, 2000. ISSN 0167-739X. doi: https://doi.org/10.1016/S0167-739X(99)00059-X. URL `http://www.sciencedirect.com/science/article/pii/S0167739X9900059X`.

[23] Greg Mori and Jitendra Malik. Recognizing objects in adversarial clutter: Breaking a visual captcha. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 1, pages I–I. IEEE, 2003.

[24] Youssef Nakkabi, Issa Traore, and Ahmed Awad E. Ahmed. Improving Mouse Dynamics Biometric Performance Using Variance Reduction via Extractors With Separate Features. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 40(6):1345–1353, nov 2010. ISSN 1083-4427. doi: 10.1109/TSMCA.2010.2052602. URL `http://ieeexplore.ieee.org/document/5524019/`.

[25] Hsing-Kuo Pao, Kuan-Ta Chen, and Hong-Chung Chang. Game bot detection via avatar trajectory analysis. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(3):162–175, 2010.

[26] Hsing-Kuo Pao, Junaidillah Fadlil, Hong-Yi Lin, and Kuan-Ta Chen. Trajectory analysis for user verification and recognition. *Knowledge-Based Systems*, 34:81–90, 2012.

[27] Maja Pusara and Carla E. Brodley. User re-authentication via mouse movements. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security - VizSEC/DMSEC '04*, page 1, New York, New York, USA, 2004. ACM Press. ISBN 1581139748. doi: 10.1145/1029208.1029210. URL `http://portal.acm.org/citation.cfm?doid=1029208.1029210`.

[28] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *NIPS*, 2016.

[29] B. Sayed, I. Traore, I. Woungang, and M. S. Obaidat. Biometric Authentication Using Mouse Gesture Dynamics. *IEEE Systems Journal*, 7(2):262–274, jun 2013. ISSN 1932-8184. doi: 10.1109/JSYST.2012.2221932. URL `http://ieeexplore.ieee.org/document/6416916/`.

[30] Jason Silverstein. Hundreds of millions of Facebook user records were exposed on Amazon cloud server. *CBC*, April 2019. URL `https://www.cbsnews.com/news/millions-facebook-user-records-exposed-amazon-cloud-server/`.

[31] Suphannee Sivakorn, Iasonas Polakis, and Angelos D Keromytis. I am robot:(deep) learning to break semantic image captchas. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 388–403. IEEE, 2016.

[32] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

[33] Guixin Ye, Zhanyong Tang, Dingyi Fang, Zhanxing Zhu, Yansong Feng, Pengfei Xu, Xiaojiang Chen, and Zheng Wang. Yet another text captcha solver: A generative adversarial network based approach. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 332–348. ACM, 2018.

[34] Nan Zheng, Aaron Paloski, and Haining Wang. An efficient user verification system via mouse movements. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 139–150. ACM, 2011.

[35] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, pages 1433–1438. AAAI Press, 2008. ISBN 978-1-57735-368-3. URL `http://dl.acm.org/citation.cfm?id=1620270.1620297`.

Code is available on https://github.com/gavinMcCrackun/RL_Submission.

A Appendix: Proof Details

Claim A.1. *The function* $\theta^* = \arg\max_\theta L(\theta)$ *is convex and has gradient*

$$\nabla L(\theta) = \tfrac{1}{N}\textstyle\sum_\zeta \mathbf{f}_\zeta - \sum_\zeta P(\zeta \mid \theta)\mathbf{f}_\zeta$$

Proof.

$$\begin{aligned}
\theta^* &= \arg\max_\theta \frac{1}{N}\sum_\zeta \log P(\zeta \mid \theta) = \arg\max_\theta \frac{1}{N}\sum_\zeta \log \frac{1}{Z} e^{R(\theta,\zeta)} \\
&= \arg\max_\theta \frac{1}{N}\sum_\zeta \left(R(\theta,\zeta) - \log Z\right) = \arg\max_\theta \frac{1}{N}\sum_\zeta R(\theta,\zeta) - \log \sum_\zeta e^{R(\zeta,\theta)} \\
&= \arg\max_\theta \frac{1}{N}\sum_\zeta \theta^T \mathbf{f}_\zeta - \log \sum_\zeta e^{\theta^T \mathbf{f}_\zeta}
\end{aligned}$$

Both of these terms are convex, so the function is convex.

$$\begin{aligned}
\nabla L(\theta) &= \frac{1}{N}\sum_\zeta \mathbf{f}_\zeta - \frac{1}{\sum_\zeta e^{\theta^T \mathbf{f}_\zeta}} \sum_\zeta \frac{\partial}{\partial \theta} e^{\theta^T \mathbf{f}_\zeta} = \frac{1}{N}\sum_\zeta \mathbf{f}_\zeta - \frac{1}{\sum_\zeta e^{R(\theta,\zeta)}} \sum_\zeta \mathbf{f}_\zeta e^{R(\theta,\zeta)} \\
&= \frac{1}{N}\sum_\zeta \mathbf{f}_\zeta - \sum_\zeta \mathbf{f}_\zeta P(\zeta \mid \theta) \quad \text{as we wanted to show.}
\end{aligned}$$

□