SneakerHeadz263: Using generative adversarial networks to generate fake mouse movements

Jeremy Dohmann and James Gui

Abstract

In this paper, we attempt to use generative adversarial networks (GANs) and long-short term memory neural networks (LSTMs) to generate artificial mouse movements. Rather than other approaches such as [24, 20] our system accepts no human input about where to click and simply generates trajectories across the screen. We also use recurrent neural networks which are more amenable to sequence generation as well as GANs which are well-suited to generative problems. Additionally, GANs help remedy the problem, claimed in [2], that there don't exist any wide-scale publicly available data sets of fake mouse movements, allowing us to instead bootstrap fake bot data using a data set of real human data alone. We validate the generated sequences by visual inspection, calculating statistics of the trajectories using methods described in [2], and by testing the bots on real websites. We found evidence that the models underfit our data and that the GAN suffered from instability due to insufficient compute resources to perform hyperparameter search. Additionally, we found that in the websites we explored (Amazon, Walmart, and Ssense), anti-bot measures seem to first focus on more easily-detected information like browser type rather than mouse movements, but more extensive work is required to make more definitive conclusions.

1 Introduction

In 2018, nearly 40% of all Internet traffic consisted of bot activity, with more than half of that representing malicious bots [22]. These malicious bots are aimed at achieving a variety of goals, including account takeover, content scraping, and fraud [4]. And with the rise of companies like Supreme that use artificial scarcity to drive their products' popularity, another subset of bots has exploded in usage: sneaker bots. According to StockX, a leading re-sale platform, one-third of Gen Z-ers consider themselves "sneakerheads" [26], and this demographic contributes to the growing $6 billion market in re-sale of limited edition products such as sneakers, trading cards, and video game consoles. Bot developers capitalize on this market by using automated browsers like Selenium and Puppeteer to purchase items with superhuman speed. Unlike bots that aim to brute-force their way into accounts or scrape content from websites, however, these bots' primary goal is to cleverly evade detection rather than to send flurries of requests.

Most anti-bot services use browser fingerprinting, along with machine learning and statistical models, to distinguish between bot and human behavior [4]. A browser fingerprint consists of information associated with a user's browsing activity, from the user-agent and local IP address to hardware info and even mouse movements. While browser fingerprinting is often used to track and attribute user data and sometimes seen as a potential privacy concern [1], anti-bot services use the same techniques to ensure the validity of the web traffic going to a particular site. By analyzing browser fingerprints, these anti-bot services can infer whether a certain user is a benign bot, malicious bot, or human user [10]. If a certain request is deemed to be from a malicious bot, the website can block traffic from the bot's IP address, show a CAPTCHA for the user to solve, rate-limit that user, or otherwise try to stop the bot from completing its tasks. This project aims to explore the robustness of anti-bot services by attempting to model human mouse movements using machine learning.

2 Related work

2.1 Browser fingerprinting and anti-botting

Much previous work on browser fingerprinting uses browsing data to uniquely identify users. Eckersley demonstrated that simple queries to Javascript APIs could yield a fingerprint of at least 18.8 bits of identifying information [13]. Since 2010, researchers have added to the suite of potentially-identifying information that fingerprinting algorithms can leverage; Cao et al developed WebGL-based techniques for fingerprinting OS and hardware-level information like graphics card data, using it to track users who browsed across devices [11].

Using browser fingerprints to detect bots has largely been unexplored in academia, however. A 2019 survey by Laperdrix et al discovered only one academic article that used browser fingerprinting to detect bots: a Google research paper that that used canvas fingerprinting to detect bot traffic [18]. Meanwhile, most anti-bot services (Distil Networks, MaxMind, PerimeterX, to name a few) claim to use fingerprinting to detect bots. DataDome claims to use browser fingerprinting to block traffic from bots that use Selenium, Puppeteer, and even libraries that themselves claim to be able to bypass fingerprinting [12]. In the bot vs anti-bot arms race, anti-bot services develop fingerprinting suites that include checks for popular browser automation libraries like Selenium. Conversely, bot developers discover bypasses that disguise their bot activity, changing the browsing data associated with their fingerprint.

As it turns out, these proprietary solutions are often patchy and inadequate. Azad et al investigated 15 popular anti-bot services and were able to bypass their protections by using browsers like Safari with AppleScript that are less popular for automation [4]. In their analysis, constructively whitelisting users known to be "real" is more effective than playing whack-a-mole with bot developers by blacklisting specific browsing contexts.

2.2 User authentication with mouse movements

As mentioned in [4], human mouse movement patterns can be highly idiosyncratic, and some services claim to include mouse movement signals in browser fingerprints. Though it is difficult to know precisely whether and how anti-bot service providers incorporate mouse spoofing detection systems, the reverse engineering experiment performed in [4] suggests that only one of the seven anti-bot services studied incorporates mouse movements in a discernible way.

Though we could not find any large-scale, publicly available, and *successful* attempts at performing human vs. bot mouse movement classification, [27] presents an unsuccessful attempt. Unfortunately, the paper was a rather brief technical report for a class project and had neither a release of the data collected, a description of the classification system, nor any successful results to report. It appears that they had difficulty generating nontrivial examples of “fake” data to serve as training for their classifier. One of the major issues hindering scholarship on this area is the lack of a publicly available, high-quality data set containing non-human mouse movement trajectories. This may be inherent to the problem space since security is a moving target and the most advanced hackers are always improving in order to avoid detection.

By contrast, there have been some studies on building per-user profiles of mouse movement data and using machine learning systems to perform user authentication. One such example is (very) informally described [23] and another (peer-reviewed) study is given here [2]. This paper will rely heavily on the techniques of [2], in which the authors use elementary physics to hand-design features fed into a random forest classifier. Their approach calculates numerous features including average velocity, angular momentum, jerk, and acceleration. Though they achieve good results (average 80%) on the 10-way user authentication of the Balabit data set [5] they introduce a strong inductive bias into the features used by the learning algorithm, which should result in reduced accuracy overall. Additionally, their classifier is not recurrent, and thus is incapable of accounting for sequential information.

From an architectural perspective, user authentication and bot detection are identical problems (though the former is potentially a multi-class classification). As mentioned in [2], the main motivation for focusing on authentication is that while there is a publicly available data set, the Balabit data set discussed below, containing user-ID’d real web traffic, no such data set exists for state-of-the-art bot mouse movements. Though it could be possible to generate a fake data set using an array of data augmentation techniques, such a project would be a massive endeavor, the quality of which also difficult to validate.

2.3 The Balabit data set

The Balabit data set was released in 2016 by the Romanian software company of the same name. Though it was ostensibly released as part of a data science competition, the Github repository no longer contains any active links to it, nor does it contain any in-depth description of how the data was obtained. According to [2], however, it was the best such public data set available online as of 2018 [5].

The data set consists of multiple sessions for 10 different users of an unnamed website. Each session includes time stamps, mouse coordinates scaled to a 4000 by 4000 pixel window, and button actions (including drags, clicks, releases, and moves). Each user has dozens of sessions associated with them and each session is thousands of button actions long, with the average duration between button actions being .3 seconds.

In total there are approximately 2 million unique mouse movements in the data set.

2.4 Survey of alternative approaches to mouse movement generation

A common way to produce naturalistic looking mouse movements for bots is to use cubic splines or Bezier curves to interpolate between user-provided points [28, 20, 24]. As discussed in [4], it is difficult to reverse engineer exactly what anti-bot services do under the hood, so there haven’t been any public analyses of whether Bezier curves are hard to detect for anti-bot systems. An interesting path for future work would be to train a classifier to distinguish user data from Bezier splines and determine whether they are any harder to detect than linear interpolation. This would also be a difficult task given the unknown provenance of the Balabit data set and would require deep analysis to ensure that the false data hovered at locations drawn from the same distribution as the user data.

Another under-studied and under-documented approach is using neural networks to generate naturalistic interpolations between user chosen points. One such informally documented study uses a simple feed-forward model on an unnamed data set [19] to generate interpolations between user-provided points. Though there is no numerical assessment of the quality of the trajectories produced, they pass a basic visual inspection. It is interesting that a non-recurrent deep learning method would achieve good results, since sequence generation is typically performed using RNNs.

From our survey of the literature, we believe that deep-learning approaches to creating naturalistic mouse movements are in a very nascent state. There is potential for investigations into the impacts of recurrent networks and other more advanced generative architectures such as variational autoencoders (VAEs) and generative adversarial networks (GANs).

2.5 Deep learning approaches to sequence generation

As mentioned in the previous section, there has been little public work on generating naturalistic mouse movements using deep learning and even less on using deep learning to detect bots. Where such studies have been done, they did not investigate more advanced architectures such as recurrent neural networks or GANs. In this section we will briefly cover LSTMs and GANs.

2.5.1 LSTMs. Recurrent neural networks are special architectures designed for sequence prediction. They are typically trained over sequences (e.g. text) to perform a ‘next token’ prediction task. They output both a hidden state used to carry information from one time step to another as well as the identity of the next token in the sequence. RNNs in general are known to suffer difficulties with long range structure, a problem somewhat remedied by the introduction of LSTMs [15] which have explicit mechanisms to maintain long-term memory.

In addition to next token prediction, RNNs can also perform sequence generation. As discussed in [14], however, they can struggle with longer sequences, and frequently produce nonsensical (albeit) novel outputs. Thus, they may not be suitable for tasks which are sensitive to noisy or partially incorrect outputs and their output often needs to be either heuristically or hand-edited to pass human inspection (see for example GPT-3’s recent Modern Love essay [21]).

2.5.2 GANs. Generative adversarial neural networks (GANs) are commonly used in computer vision to bootstrap training generative models in contexts where the valid output space is very large. The training procedure works by jointly training a generative model (e.g. a convolutional neural network that produces images, or an LSTM that produces text) with a ‘discriminator’ that tries to identify as false. The GAN architecture is a zero-sum game in which the generator does well only at the discriminator’s expense and vice versa [7].

GANs are useful in contexts where a data set of authentic examples is known but the set of false examples is impossibly large. When appropriately tuned they can create extremely authentic images or videos and they form of the basis of deep fake technology. (See this tutorial generating fake celebrities). Because they are zero-sum, they are extremely challenging to tune and are not provably guaranteed to produce a “good” result. Furthermore, it is hard to quantify the quality of their outputs, and many studies resort to hiring humans to judge them. There are several possible failure modes for GANs, discussed here [8]:

1. As shown in [3], there is always a perfect discriminator in the traditional GAN setting (and thus discrimination is easy). Additionally, when the discriminator becomes perfect the gradient of the generator becomes zero and no updates occur.
2. GANs can suffer from mode collapse, in which they repeat low entropy parts of the output space and appear repetitive.
3. They can be non-convergent, in which neither the generator nor discriminator ever gets the upper hand.
4. The generator can learn to fool the discriminator by producing out of domain examples.

3 Our approach

By contrast to the other approaches reviewed earlier, our model focuses on producing naturalistic cursor movements without receiving input from the user about where to visit. Thus, it is more similar to a random walk than to an interpolation, and not a directly useful approach for botting. The reason for this design decision was to focus on GANs and LSTMs, techniques that we later realized are not directly amenable to the interpolation context. The former would be difficult to apply because we would need to ensure that the hand-programmed click locations are drawn from the same distribution as the underlying human data set and the latter would be difficult because LSTM generation doesn’t typically use hand-programmed anchor points.

We decided try a GAN approach due to the lack of publicly available data on bot mouse behavior. GANs not only provide a way to bootstrap training using a data set of only positive examples but they also (can) provide a good model for the adversarial nature of bots and anti-botters.

Other approaches could have focused on generating good fake data, instead highlighting the importance of discriminating bots as opposed to producing them. Though we could have chosen heuristics for generating such data based on methods popular among script-kiddies, it would be difficult to know what the true underlying distribution is for real-world threats.

Our experiment uses LSTMs as the backbone for both our generator and our discriminator. We tried both pre-training the generator LSTM and fine-tuning it using a GAN training method, and simply naively training the generator LSTM on the human data.

3.1 Data

We used the Balabit data set with the training set partitioned into a 80/20 split with 20% held out for pre-training the LSTM. The validation set had roughly 300,000 unique mouse actions and the training set had roughly 1.5 million. We ignored the timestamp generation from our algorithm and focused simply on producing x,y coordinates and button events, though, as discussed later, we were unsuccessful at producing coherent sequences of button actions and had to resort to ignoring them altogether in our later analysis.

Each session was broken into 200-length chunks, and we trained the LSTMs on 200-length sessions at a time, resulting in, on average one minute of mouse movements. This length was chosen because the data from Balabit had very high granularity and sequences of less than 200 seldom had any interesting structure to them. We also normalized all coordinate values to be between 0 and 1 to control the size of the gradient.

3.2 Long-short term memory RNNs

The generator LSTM was a 4-layer LSTM with 96 units in its hidden state. Its input was a tuple of x,y coordinates as well as an integer encoding one of the 8 possible button states in the Balabit data set. We also had two fully connected sequences, one which produced a pair of coordinates and the other an 8-length vector corresponding to the probabilities assigned to the possible actions. Each of those sequences consisted of two fully connected layers with a ReLU activation between them. The implementation is based on the starter code in [6]. The total number of parameters (roughly 270,000) was chosen to be approximately equal to the number of data points in the validation set used for pre-training. Additionally, we lacked the compute resources to reasonably train larger models.

Because coordinate generation was a regression task and button action selection was a categorical task, we used MSE loss for the coordinate loss and cross entropy for the button action loss. The weightings of these two contributions were calibrated with the MSE loss being weighted by a factor of 100 so that during early epochs both contributed equal amounts to the loss.

We produced both a pre-trained LSTM trained only on validation data to be used for fine-tuning by the GAN as well as an LSTM trained on the entire training sequence. They were both trained for 60 epochs with a learning rate of 0.0001 and dropout of 0.2. Training on the full train set took roughly 2 hours on a 2 core machine with 6 GB of GPU. Due to insufficient compute resources we did not perform any hyper parameter search on either the architecture shape or the training parameters. The loss curve for the LSTM training on the full training set is shown below:

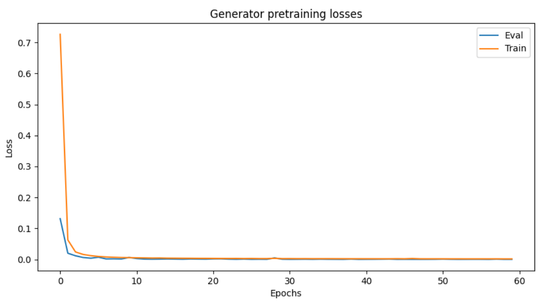


Figure 1. Training losses for the LSTM on the train set show the validation loss consistently below the training loss, a sign of underfitting, suggesting that with more compute resources, a larger model would've been the right choice.

The validation loss being lower than the training loss plus the simplicity of the outputs trajectories suggests that the model under-fit the data and thus we could've seen substantial benefits from increasing the model size.

The discriminator was a 3 layer BiLSTM with hidden state of size 64. It read in full sequences at a time and passed the hidden states into a series of two fully connected layers whose final output size is 2 (corresponding to probability of the input sequence being real or fake, respectively). The input was identical to that of the generator. It was trained with cross entropy loss (as typical in GANs).

3.3 GAN architecture

We used the traditional [17] GAN implementation from the pytorch tutorials but with our pre-trained LSTM as the generator and the discriminator LSTM as our discriminator.

As noted in the related work section, GANs can fall victim to perfect discriminators, mode collapse, non-convergence, and junk outputs from the generator.

To remedy the first problem we followed the recommendation of [3] and used $-log(D(G))$ as our loss for the generator (where $G(x)$ is the output of the generator and $D(G(x))$ is the probability the discriminator assigns to the generator's output being real. Other solutions we adopted to help resolve this, per [9], were to increase dropout for the discriminator to 0.5 and make the discriminator smaller than the generator.

These strategies can result in instability. Per [3] and [16], we added Gaussian noise with variance 0.0005 and mean 0 to the inputs and randomly labeled fake sequences as real with probability 0.2 when training the discriminator. The first strategy is called instance noise and the second strategy is called one-sided label smoothing. We divided the instance noise and the label smoothing probability by the epoch number in order to gradually fade them out.

Training the GAN for a mere 10 epochs took over 12 hours. Due to the computational expense we couldn't perform any of the hyperparameter tuning which likely would've been necessary to ensure good training. We eventually saw deeply convergent loss curves, followed by an overly powerful discriminator. The loss curves are shown below:

The evaluation in the following section will also show that the GANs suffered mode collapse, generating random, jumpy segments and very basic, non naturalistic geometric trajectories as well as generating out of domain data to fool the discriminator. In particular, it generated coordinates outside of the valid ranges leading us to need to clip them

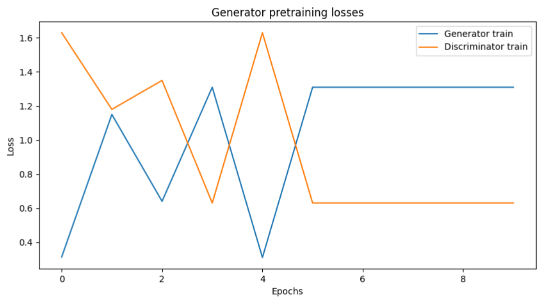


Figure 2. The GAN training curves show early instability followed by the discriminator becoming perfect and the training converging suboptimally.

in order to visualize them or study their statistics. Thus we managed to suffer from all four possible failure modes for the GAN, highlighting just how expensive and complex their training can be (no wonder hackers stick to Bezier curves!)

4 Evaluation

It is challenging to measure the quality of generative models. In studies on generative text models [14] and image synthesis [25] it has been common practice to employ human raters to rate the quality of the model outputs. Such approaches make sense when you have the resources to pay raters and when the output space is one in which human raters have good intuitions about what is good and what is bad. In the case of this project, we didn't have the resources to hire raters, and even if we did it is unclear whether humans would be good at distinguishing real mouse movements from bot mouse movements.

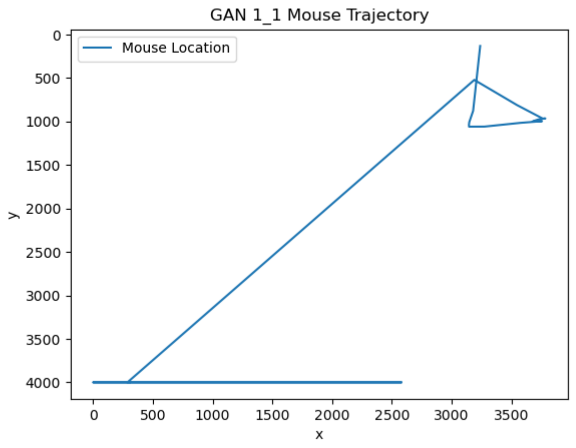
Thus we relied on three approaches: visualizing the trajectories and inspecting them by hand, calculating the physical trajectory statistics from [2] and comparing the distributions resulting from our models to those of the human data, and deploying the bots on real world sites and evaluating whether they were flagged as bots by the sites' security system.

It is important to note that even though we set out to include button actions in our generation algorithm, the data was so sparse that the models struggled to learn coherent button actions and we had to omit them from our analysis.

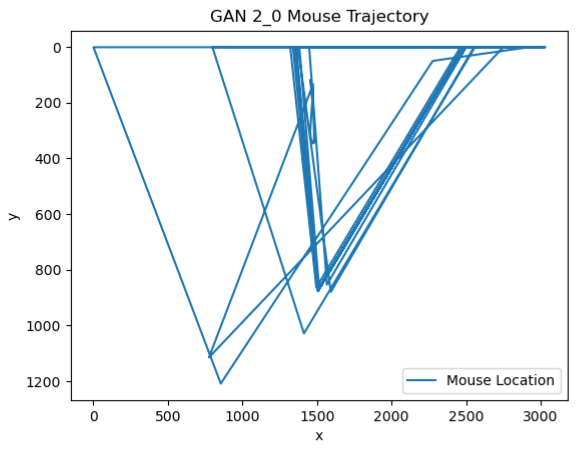
4.1 Visualizing the mouse movements

The LSTM trained on the full training set (see, 3b) outperformed all other approaches upon visual inspection. By contrast with the LSTM pretrained 3c on only the validation set, it had much smoother lines and greater complexity, highlighting the effectiveness of larger data sets. It also, however, showed signs of underfitting, due to the lack of complexity and structure compared to the human data, 3a.

The GANs (figures 4a and 4b) were unequivocally unsuccessful: they displayed major mode collapse, generating repetitive sequences, out-of-domain outputs, and jerky movements.



(a) The GAN at epoch 4 (which minimized its loss against the discriminator) fooled the discriminator by consistently producing trajectories that were outside the window-size, which were clipped to the max pixel value here.



(b) The GAN at the final epoch was consistently producing repetitive triangular motions, looking essentially like random jumps.

4.2 Measuring movement statistics

One quantitative approach to measuring the quality of generated outputs is to calculate the statistics of its underlying distribution. As noted earlier, [2] introduced a framework for parsing mouse movement sessions into discrete actions and calculating physical statistics such as average velocity, acceleration, and many more.

Though we generated 39 total analyses from the features presented in [2], the general pattern for all measures was fairly similar, so we only present the more interesting histograms. The figures in 5 below demonstrate that by and large, the LSTMs were close, but not identical to reproducing the same statistics as the human data. The GAN training procedure only served to cause more divergent generations, with the GAN which was trained for 4 epochs (rendered in blue) produced substantially out of domain data, despite having lower loss of the two GANs, suggesting the instability of the training procedure.

4.3 Real world testing

Our options for testing bots in the real world were limited by our resources. We explored gray-box testing as demonstrated in [4], signing up for a free DataDome trial and setting up a WordPress storefront. However, integration with DataDome's plugins required costly account upgrades, so we were not able to use their dashboard to evaluate our bot's effectiveness against Datadome or other anti-bot services.

Instead, we took some example outputs from our generative models and fed those into Puppeteer, attempting to add items to a cart on various e-commerce platforms. There are a number of limitations to this approach. First, we did not attempt to hide the usage of Puppeteer or otherwise manipulate our browser fingerprint. The original goal of this set-up was to isolate our mouse movements as the only variable that might change the outcome of our attempts. However, our testing based on a few different sites (Amazon, Walmart, Ssense) showed that other aspects of the browser fingerprint would more likely trigger blocks before mouse movement is even taken into account; Walmart, for example, likely blocks any use of Puppeteer, as even human activity from the Puppeteer browser in non-headless mode was blocked from adding items to the cart. On the other hand, Amazon and Ssense allowed users that attempted to purchase items with non-disguised mouse movements to add items to the cart, demonstrating that their anti-bot measures did not take into account mouse movements. A second limitation is that our networks did not take into account source and destination coordinates; they only attempted to create "human like" motions. In a real browsing scenario, users move the mouse between click events with a clear start and stop point. When feeding data into our bot, however, we had no way of controlling where the mouse ended up moving, so the click events would teleport the mouse location in a very non-human way.

4.4 Ethical considerations

Because our real-world tests were so limited in scope, we are confident that our bot traffic did not hinder regular traffic in any way. Furthermore, we simply attempted to add things to a cart without purchasing them, so the impact of our bot activity was also limited. Because our tests did not discover any large vulnerabilities that would not already be known by these websites and their anti-bot providers, we did not feel the need to disclose the results to the e-commerce platforms against which we tested our model.

5 Limitations, conclusions, and future work

In this paper we presented a study on generating naturalistic mouse movements with no human input. This approach differed from others in the field because it emphasized using machine learning, including recurrent networks and GANs, to generate fake movements and didn't use human input (i.e. the human exercises no control over where the mouse goes). We found that LSTMs could produce fairly naturalistic mouse movements. Additionally, our results suggest our LSTMs underfit the data, and thus larger models would likely be able to achieve much greater results. This suggests that recurrent models may be the future of authentic mouse movements. Though modifications would be necessary to ensure that the user can provide desired click locations as input, allowing the LSTM to interpolate between them.

Our results also showed that GANs were extremely resource intensive and difficult to train. Though it may be possible to train a successful GAN with more computer resources, it is unlikely that it would be worth the expense when compared to the quality of the simple LSTM.

From the ML design side, it appears that our work was hindered by two main problems: the LSTMs underfit the existing data and the GANs were improperly tuned. With more computer resources it would've been possible to train better fitting LSTMs for longer, and it would've been possible to hyperparameter search the GANs. Additional improvements could have focused on data augmentation or modifying the data to avoid sparseness issues. As noted earlier, due to the sparseness of button actions (very few button actions were used per session) and the length of the trajectories, it was difficult for LSTMs of our size to learn coherent button action policies. Truncating sequences to emphasize button actions only, augmenting the data set, or increasing the model size would likely have helped with this. Additionally, our model didn't output time stamps and thus all actions were equally-spaced. Larger models could've likely incorporated timing information as well. Being that that would introduce another term to the loss function, this would've necessitated more hyper parameter searches, an expensive procedure we couldn't perform.

On the data side, it is a major shortcoming that there is no current documentation for how the Balabit data set was produced [5].

From the evaluation side, given a larger budget and more time, future work could include gray-box testing as described in [4] could also yield some interesting results. Developing a more robust and comprehensive testing protocol that could isolate the effect of using human vs computer-generated mouse movements would also allow us to more directly evaluate the impact that mouse movements have on determining bot traffic.

Future directions could focus on classification rather than generation. Though Bezier curves are a popular interpolation technique, there is no proof that they are any harder to distinguish from human movements than simple linear interpolations. Though it would require more meticulous construction of the human data set to ensure that we control for click location, path length, movement duration, etc. and focus solely on distinguishing the shape of Bezier curves from human generated trajectories.

References

[1] Gunes Acar et al. "The Web Never Forgets: Persistent Tracking Mechanisms in the Wild". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. Scottsdale, Arizona, USA: Association for Computing Machinery, 2014, pp. 674–689. ISBN: 9781450329576. DOI: 10.1145/2660267.2660347. URL: https://doi.org/10.1145/2660267.2660347.

[2] Margit Antal and Elöd Egyed-Zsigmond. "Intrusion Detection Using Mouse Dynamics". In: *CoRR* abs/1810.04668 (2018). arXiv: 1810.04668. URL: http://arxiv.org/abs/1810.04668.

[3] Martin Arjovsky and Léon Bottou. *Towards Principled Methods for Training Generative Adversarial Networks.* 2017. arXiv: 1701.04862 [stat.ML].

[4] Babak Amin Azad et al. "Web Runner 2049: Evaluating Third-Party Anti-bot Services". In: *17th Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA 2020).* Lisboa, Portugal, 2020.

[5] Balabit. *balabit/Mouse-Dynamics-Challenge.* URL: https://github.com/balabit/Mouse-Dynamics-Challenge.

[6] Domas Bitvinskas. *PyTorch LSTM: Text Generation Tutorial.* URL: https://www.kdnuggets.com/2020/07/pytorch-lstm-text-generation-tutorial.html.

[7] Jason Brownlee. *A Gentle Introduction to Generative Adversarial Networks (GANs).* July 2019. URL: https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/.

[8] Jason Brownlee. *How to Identify and Diagnose GAN Failure Modes.* Aug. 2020. URL: https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/.

[9] Jason Brownlee. *Tips for Training Stable Generative Adversarial Networks.* Sept. 2019. URL: https://machinelearningmastery.com/how-to-train-stable-generative-adversarial-networks/.

[10] E. Bursztein et al. "Picasso: Lightweight device class fingerprinting for web clients". In: *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices.* 2016.

[11] Yinzhi Cao, Song Li, and Erik Wijmans. "(Cross-) Browser Fingerprinting via OS and Hardware Level Features". In: *24nd Annual Network and Distributed System Security Symposium, NDSS.* 2017.

[12] DataDome. *How to detect, block and manage sneaker bots.* URL: https://datadome.co/bot-detection/how-to-detect-block-manage-sneaker-bots. (accessed: 12.10.2020).

[13] Peter Eckersley. "How Unique Is Your Web Browser?" In: *Privacy Enhancing Technologies.* Ed. by Mikhail J. Atallah and Nicholas J. Hopper. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–18. ISBN: 978-3-642-14527-8.

[14] Alex Graves. "Generating Sequences With Recurrent Neural Networks". In: *CoRR* abs/1308.0850 (2013). arXiv: 1308.0850. URL: http://arxiv.org/abs/1308.0850.

[15] Sepp Hochreiter and Jürgen Schmidhuber. "LSTM Can Solve Hard Long Time Lag Problems". In: *Proceedings of the 9th International Conference on Neural Information Processing Systems.* NIPS'96. Denver, Colorado: MIT Press, 1996, pp. 473–479.

[16] Ferenc Huszar. *Instance Noise: A trick for stabilising GAN training.* Oct. 2016. URL: https://www.inference.vc/instance-noise-a-trick-for-stabilising-gan-training/.

[17] Nathan Inkawich. *DCGAN Tutorial.* URL: https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html.

[18] Pierre Laperdrix et al. "Browser Fingerprinting: A survey". In: *CoRR* abs/1905.01051 (2019). arXiv: 1905.01051. URL: http://arxiv.org/abs/1905.01051.

[19] Cezar Augusto Cordeiro de Lima. *Generating mouse movement with ANN.* Oct. 2018. URL: https://medium.com/neuronio/generating-mouse-movement-with-ann-6659c53a6fe2.

[20] MasterFocus. *RandomBezier() - Move the mouse through a random Bézier path - Scripts and Functions.* Sept. 2012. URL: https://autohotkey.com/board/topic/85043-randombezier-move-the-mouse-through-a-random-bezier-path/.

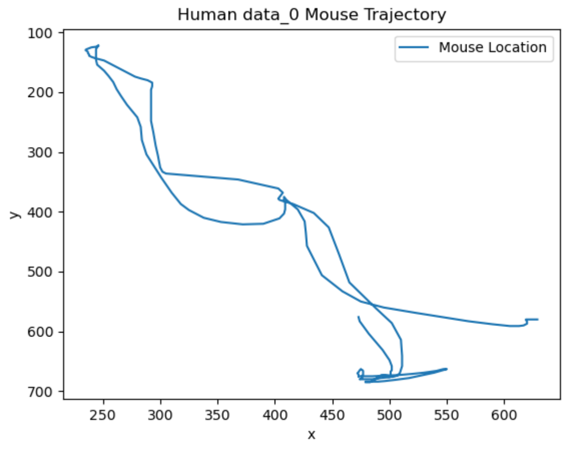
[21] Cade Metz. *When A.I. Falls in Love.* Nov. 2020. URL: https://www.nytimes.com/2020/11/24/science/artificial-intelligence-gpt3-writing-love.html.

[22] Distil Networks. *2019 Bad Bot Report.* URL: https://www.bluecubesecurity.com/wp-content/uploads/bad-bot-report-2019LR.pdf. (accessed: 12.10.2020).

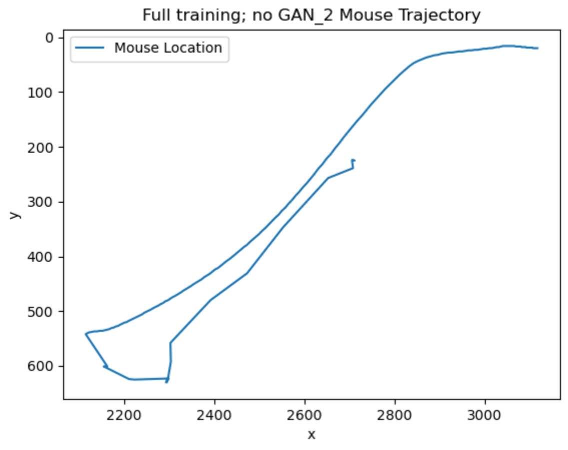
[23] Jose Carlos Norte. *Advanced Tor Browser Fingerprinting.* Mar. 2016. URL: http://jcarlosnorte.com/security/2016/03/06/advanced-tor-browser-fingerprinting.html.

[24] Patrikoss. *pyclick.* Sept. 2018. URL: https://github.com/patrikoss/pyclick.

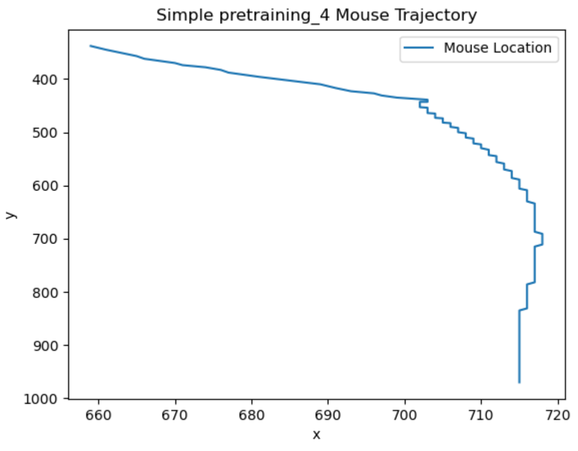
[25] Scott E. Reed et al. "Generative Adversarial Text to Image Synthesis". In: *CoRR* abs/1605.05396 (2016). arXiv: 1605.05396. URL: http://arxiv.org/abs/1605.05396.



(a) Human trajectories show complexity, smoothness, and long range structure.



(b) LSTM trained on full data set shows smoothness and little complexity; indicating underfitting.

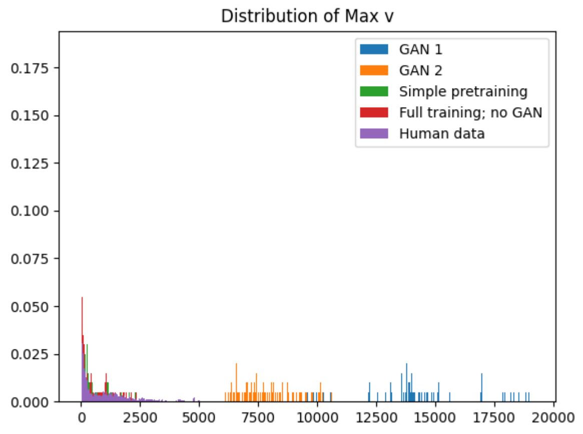


(c) LSTM trained only on the validation set shows rough movements, suggesting the benefits of large data sets.

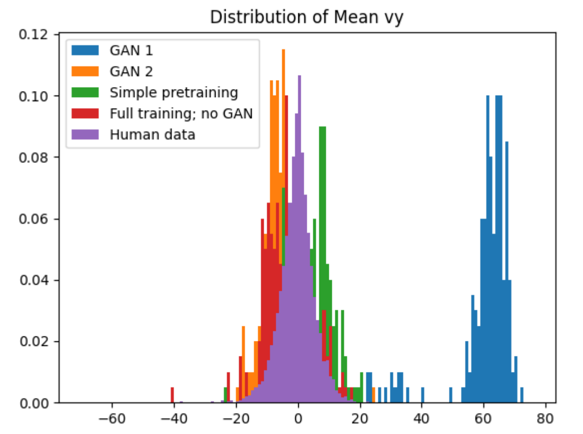
[26] Stockx. *StockX Snapshot: The State of Resale.* URL: https://stockx.com/news/state-of-resale/. (accessed: 12.10.2020).

[27] Evan Winslow. *Bot Detection Via Mouse Mapping.*

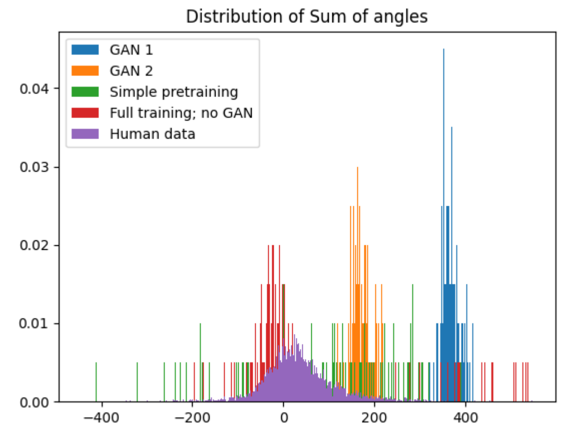
[28] Xvol. *BezMouse.* Aug. 2017. URL: https://github.com/xvol/bezmouse.



(a) Max velocity shows that the LSTMs achieved high quality.



(b) Mean y velocity shows that all models except for GAN trained for 4 epochs achieved decent velocity distribution.



(c) Sum of angles shows that the LSTMs had much higher variance in the sum of the angles in a trajectory than human data. It is unclear why exactly this is, but training a largely model would likely close the gap.

Figure 5. Histograms of physical trajectory statistics calculated from [2]