Label-Free Object Detection in Videos of Physical Interactions with GANs

Alex Barron, Todor Markov, Zack Swafford, Russell Stewart, and Stefano Ermon
Stanford University
{admb, tmarkov, zswaff, stewartr, ermon}@stanford.edu

Abstract

Object recognition is currently one of the most important problems in computer vision. Most approaches to object recognition focus on supervised learning methods, which often need large amounts of labeled data in order to train. In this paper, we pursue an unsupervised learning approach to object recognition by incorporating a physics prior. We attempt to train a Generative Adversarial Network on two tasks: detecting the ball in a Pong game, and detecting the trajectory of a thrown juggling ball. The datasets we use are a simulation of a pong ball bouncing in a box, and a video of a person throwing a juggling ball. Unfortunately, we did not manage to achieve consistent good results on either of the tasks, though in both cases the GAN appeared to learn some relevant features of the movement of the object it was trying to track.

1. Introduction

Previous work [19, 18] has shown that it is possible to detect objects in videos without labels. Such algorithms are extremely desirable since well-labeled object tracking datasets are hard to come by and expensive to create. The goal of this project is to extend current results to new types of object categories. In particular, we use a Generative Adversarial Network (GAN) [6] in which the generator network attempts to extract an object's trajectory from video data and the discriminator network attempts to discriminate if the trajectory seen was given by the generator network or drawn from a pre-specified simulator. Then, newly independent from the discriminator, the trained generator is then able to automatically determine object trajectory from unlabeled videos, as desired. Yet unpublished work from Stewart and Ermon [19] demonstrates that this approach is legitimate in some cases; we will extend this to more cases.

The input to our final algorithm is videos containing the physical object, interaction or principle (in our case, balls bouncing in various environments). The discriminator is a one-layer LSTM and is involved only in training–typical of a GAN architecture–and the generator is a familiar CNN architecture. In application, the generator network is then able to distinguish ball trajectories in general, thereby automatically generating and outputting a label for the video with the object's location.

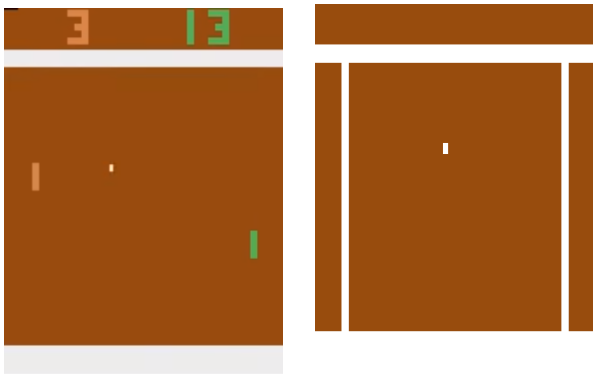

Figure 1: Screenshots from real [11] (left) and simulated (right) games of pong.

1.1. Applications

In particular, we began by investigating this network structure on a game of pong. Using a simple prespecified pong simulator of our own implementation and various unlabeled videos of actual pong gameplay (see figure 1), we trained the GAN. We evaluated our results by looking at the discriminator's ability to differentiate the generator network's trajectory estimation for the ball and the actual ball trajectory. This therefore trains the generator network in the GAN to extract the pong ball's trajectory from an unlabeled video

Results were mixed on the pong dataset, due at least in part to the fact that the underlying physics of the game are variable between game implementations and obviously our own physics implementation as well. We therefore moved to a simpler environment where the physics were completely known: a fully simulated ball bouncing physically in a small room. This problem, of our own design, allows for a reasonable amount of variation (velocities, bounce angles, and overall trajectories varied) while still maintaining

the control necessary to validate the model. We generated the relevant videos and trajectory labels (which are technically unnecessary for the model but helpful for validation) and then used the same physics engine to sample trajectories from the 'simulator'. This approach was somewhat more successful than in the more variable Atari environment.

The final task we solved was object detection during ball throwing. Using a white ball with a dark background, we generated many short videos of a ball thrown in the air. Obviously the physics to describe this environment are relatively well-known and straightforward, so we modeled the ball using a gravity simulator, from which we could draw sample parabolic ball trajectories. This task also proved difficult, but using many recent innovations and tricks that help to train GANs we achieved moderate success.

2. Related Work

GANs are one of the newest models in the generative toolbox, first proposed in Goodfellow et al. [7] in 2014. The model has gained popularity for primarily for practical reasons; samples from GANs are simply better than samples from their primary alternative, Variational Autoencoders (VAEs) [12]. Generative models in general are still quite nascent, and improvements to GANs and other models have been quite correspondingly frequent. Improvements like LSGAN [13], DCGAN [15], and WGAN [3] have been suggested and validated, and we applied as best we can the relevant parts and findings of these improved models to our project. In particular, using a least-squares loss function was among the many improvements that we attempted to appropriately train the generator. In our case LSGAN was the only particularly relevant GAN improvement possible and barely affected our outcome; in general, of course, these improvements on GANs are effective.

2.1. Training

Training GANs is notoriously difficult, which is why some papers and other resources have already been compiled specifically to collect tips and tricks [6, 17, 16] from the aforementioned plethora of research on GANs. Many of these optimizations proved fruitful in our work, such as improved techniques for sampling from the generator [21]; adding random noise to the otherwise binary training labels [2, 9]; applying techniques to weak the discriminator such as mitigating its learning rate, using less powerful architectures [5], and using weaker optimizers; and improving the generator by adding more layers.

Stewart and Ermon [19] proposed the general architecture we use for label-free object detection; their work is prefaced by somewhat similar work that is more general [18]. Using LSTMs in the discriminator is generally quite rare; only some few other papers (such as Mogren [14]) have worked with such an architecture. This is known as the SeqGAN problem [22]. In general, however, this is a novel problem with little in the way of state-of-the-art results or research. Currently, when video datasets with object tracking need to be generated, much of it is done with semi-supervised segmentation algorithms. These require additional logic to perfect and much more supervision than our approach.

One difference between our objective and that of traditional applications of GANs is that the generator outputs a simple trajectory and does not need to output an image, which is obviously much more complicated. This obviates the need for other techniques which traditionally complicate applying a GAN, such as upsampling and deconvolution.

3. Methods

In general, GAN architectures attempt to solve the non-convex problem proposed by Goodfellow et al. [7]:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right) \right].$$

This optimization problem is obviously intractable, but progress can be made with gradient ascent, alternating between training the discriminator and training the generator. The discriminator is updated in steps proportionally to

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}\left(x^{(i)}\right) + \log\left(1 - D_{\theta_d}\left(G_{\theta_g}\left(z^{(i)}\right)\right)\right) \right]$$

and the generator is then trained proportionally to

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \log D_{\theta_d}\left(G_{\theta_g}\left(z^{(i)}\right)\right).$$

When appropriately alternated, these gradient ascent schema improve the generator and discriminator in lockstep, thereby finding a solution to the overall optimization problem.

In each of our specific problems, we began by constructing a simulator which produces trajectories sampled according to the environment in which we are testing. We scale these trajectories by the maximum width and height so they are between 0 and 1. This aids the training of generator and we can easily reconstruct trajectories later by rescaling and rounding to the nearest integer. We then sample sets of consecutive frames from the video of the environment, each corresponding to a ground truth trajectory (which we never see during training). The generator then processes these frames to produce a corresponding trajectory which the discriminator will attempt to distinguish from trajectories produced by the simulator.

We use the architectures proposed by Stewart and Ermon [19] for the discriminator and generator of our GAN (figure 2).

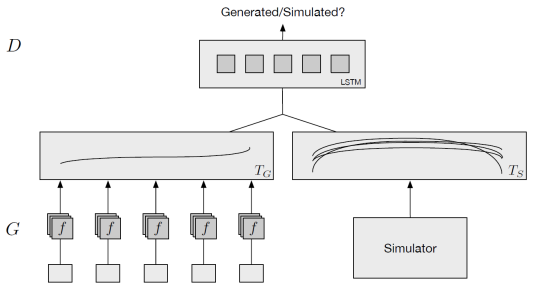

Figure 2: GAN Architecture. G is asked to generate trajectories T_G, which D then discriminates from real (simulated) trajectories T_S. [19]

3.1. Generator

The generator is then fed the raw image frames for each trajectory with mean subtracted and pixel values scaled to be between -1 and 1. We extract image features using a standard convolutional architecture of consisting of two convolutional layers with kernel size 3 and 16 filters each followed by a ReLU activation and batch normalization layer. In our experiments, the outputted trajectory can be either 1 or 2 dimensional depending on whether we are predicting height or position. Denoting this dimension k, we then use a single fully connected layer to project the output of the convolutions to a k-dim vector.

Importantly, we share weights between the convolutions and linear layer across each frame in the trajectory. This ensures that the generator must learn to use variations in the input image to synthesize the trajectory (since the convolutions are deterministic). It is this fact that we exploit to encourage the generator to match a ground truth trajectory for a new video sample, despite never having had access to the labels.

3.2. Discriminator

The discriminator first feeds the simulated or generated trajectory through 3 ReLU fully connected layers of size 64, again using the same weights for every time step. We then use an LSTM with 128 hidden units as suggested by [19] to produce a final output state of for the trajectory which is projected with a final hidden layer to a sigmoid probability. This represents the discriminator's classification of whether the trajectory was simulated or generated.

3.3. GAN Training

For training, we use Adam with learning rate 0.0001 and alternate between optimizing the discriminator and generator loss as is standard with GANs.

As has been extensively documented, training a GAN to produce good results is a non-trivial task. We experiment with a number of common GAN tricks and explored a number of variations on the above architecture to try to improve our results. One problem we often faced was that the LSTM in the discriminator would overpower the generator, leaving the discriminator loss at close to zero and the generator loss very high. In this unfortunate case, the generator outputs are trajectories of a constant height that clearly model nothing.

Following Stewart and Ermon [19], we first introduced one-sided label smoothing to alleviate the problem. This adds noise to the discriminator labels to make it more difficult for the discriminator to learn to differentiate simulated and generated output. After experimentation, we settled on subtracting Gaussian noise with standard deviation 0.3 from the positive labels [9, 2].

We also use loss statistics to balance the loss, running the generator and discriminator training steps up to 10 times in a row if their losses exceed 0.75 and 1.35 respectively [17].

While the above were sufficient to obtain relatively stable training on our more simple, artificially generated experiments, further architectural tricks were required for the real-world dataset where the problem of an overpowered discriminator was even more severe. We tried limiting the power of the discriminator by reducing the number of hidden units in the LSTM and also trying a vanilla RNN architecture in its place. We hypothesized the that since the sequence length was low we were unlikely to run into vanishing gradient problems with the RNN. Unfortunately neither helped training substantially.

In addition, we tried strengthening the generator, adding multiple convolution blocks and trying increased filter sizes to allow it to better construct trajectories. This also was unable to stabilize training, likely as we are trying to distinguish quite simple features from the video (the edges of the ball), and so do not benefit from many layers of filters.

By far the most effective stabilization technique was to train the discriminator with vanilla SGD and the generator with Adam instead of training both with Adam as Stewart and Ermon [19] suggest. This dramatically increased training time since the discriminator learns much more slowly but gave significantly smoother training curves with the discriminator loss slowly decreasing over time. It was this adjustment that allowed us to generate high quality trajectories on the ball throwing dataset. Figures 3 and 4 compare our GAN training curves before and after the change to SGD training for the discriminator.

4. Datasets

We test on two main datasets–a controlled pong environment and a more natural video dataset we create.

4.1. Pong

We initially tested on a video of a pong game we obtained from YouTube [11]. We randomly sampled 10 frame

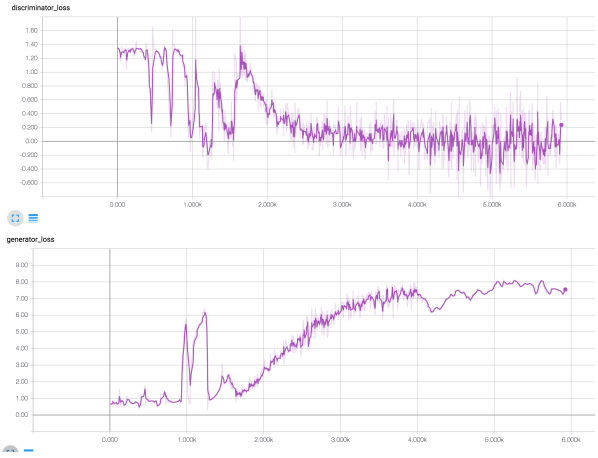

Figure 3: Discriminator and generator loss over time for our original architecture on the ball throwing dataset. The generator begins to produce reasonable samples but by step 4000 the discriminator loss has collapsed to a point where the generator cannot recover. From then on, the generator produces trajectories of constant height.

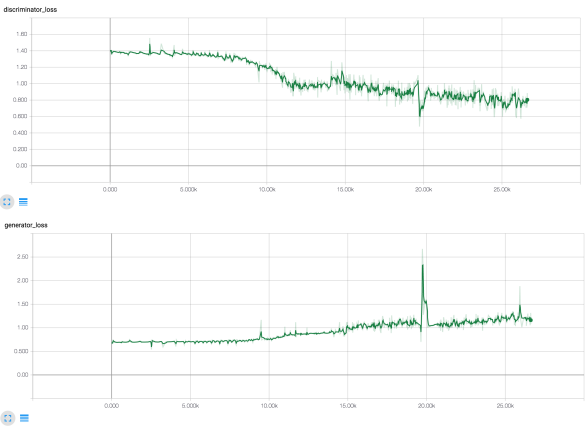

Figure 4: Discriminator and generator loss over time for the modified architecture on the ball training when the discriminator is trained with the less powerful SGD update step. The training curve is far smoother and the final output samples both more similar to simulated trajectories and closer to the ground truth for those frames.

sequences from the 3 minute video which gave us 500 video trajectories on which to train. By analyzing the video, we were able to deduce the parameters require to implement a simulator of the game in Python which mimicked the ball bouncing off the paddles, with a fixed probability of a 'miss' in which the ball would go travel off the window and the game would end. A sample trajectory from this dataset are shown in figure 5. Here we attempted to predict the x and y coordinates of the ball simultaneously.

4.2. Simplified Pong

After testing the architecture on this problem and struggling to get convincing results, we hypothesized that the small size of the ball after downsampling (two pixels), might be interfering with our algorithm's ability to detect it. Thus we implemented a complete simulation of a simplified pong game, including the frame outputs (rather than extracting them from a video). In this simulation we were able to make the ball large and remove the distraction of the moving paddle to try to form an easy toy task for the algorithm to complete. Since we had control of the simulator, we were able to try predicting both the x, y position as above and the y velocity of the ball. We show a sample trajectory for this dataset in figure 6.

4.3. Ball Throwing

The final challenge for our algorithm was to extract trajectories in a far less controlled environment: height tracking of a thrown ball in a video. For this, we filmed someone throwing a ball back and forth between hands for a few minutes, trying to keep the height and time between throws reasonably constant. We then extract 9 frame sequences from the video (this turned out to be approximately a full period).

We tried two methods for the simulation. The first modeled the height of the ball as a sine wave, with the amplitude and period empirically derived from the video. We then perturb the amplitude, period and offset of each generated sine wave by a small amount to simulate the variations in the video and increase the robustness of the generator.

The second modeled the height of the ball as a parabola. To do this, we use least squares to fit a polynomial $p_{i,0}t^2 + p_{i,1}t + p_{i,2}$ to each observed trajectory $\{t_{i,0}, t_{i,1}, \ldots, t_{i,8}\}$. Then, the simulator generates trajectories

$$\{P_0t^2 + P_1t + P_2 \text{ for } t = 0, 1, 2, \ldots, 8\}$$

where $P_j, j = 0, 1, 2$ is a random variable with

$$P_j \sim U(min_i p_{i,j}, max_i p_{i,j})$$

5. Results

5.1. Pong

We have been unable to get the GAN to accurately predict the trajectory of the ball. However, the generator network has begun to mimic some aspects of ball movement - for example the ball trajectories produced are typically smooth, with only a few pixels of movement between each frame as with the real dynamics. Motion also seemed to be generally in the direction of ball, (with significant noise) and further into training predictions tended to be closer to the real location of the ball.

The GAN training plot for the first 2500 iterations is shown in figure 3. The loss for both the generator and

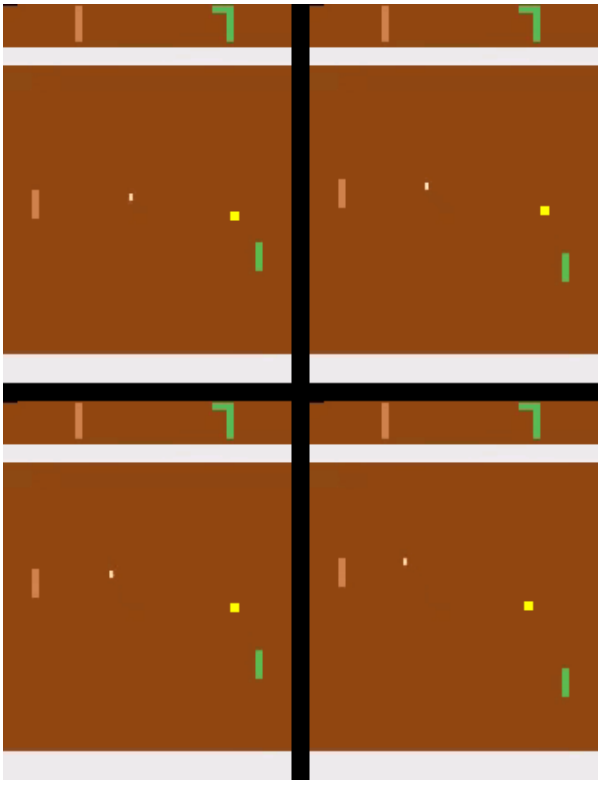

Figure 5: Still frames (in order from left to right, top to bottom) of the real ball and the generated ball (in brighter yellow). It is clear that while the positions of the real and generated balls are different, both balls behave in a vaguely reasonable and ball-like way. Both move in a single direction and bounce on contact with obstacles.

discriminator is quite high relative to what was reported in Stewart and Ermon [19] and our results distinctly suboptimal. We would hope to see a steady decrease in the discriminator loss, with low variance, but that is not present here. Compared to previous experiments in Stewart and Ermon [19], our images are quite large and the frame rate high so we intend to downsample both pixels and frames in future experiments making the signal of ball movement more clear.

5.2. Simplified Pong

We see in figure 6 a series of five stills from a representative video of a ball bouncing (in white) overlayed with our trained GANs annotation (in yellow). Note that the labeling follows a reflection of the balls true trajectory almost exactly–so it is a valid trajectory, although not precisely the desired one.

Such results are much as expected because we imposed no loss on following the true trajectory of the ball; the generator has learned to fool the discriminator by creating a real-looking trajectory inspired by the ball but not actually following the ball itself.

This aspect of the generator could potentially be improved by incorporating into the final loss a term based on any of various segmentation algorithms, which would then give the net an understanding of the physical boundaries and existence of distinct objects whose trajectory is to be tracked. However, this approach becomes less practical when the generator itself would be relied upon more heavily to distinguish objects, e.g. in cases where a vanilla segmentation algorithm could not be applied.

Figure 6: In a series of five still frames (representative of a video with many frames) that proceeds from left to right temporally, we can see the ball (in white) bouncing around the environment with the GAN's generator (whose output is in yellow) attempting to identify and track it. Clearly, the tracking does correspond to the balls movement–although not as closely as we would hope.

5.3. Ball Throwing

We show in figures 7 and 8 series of frames from the ball throwing dataset below with the annotated height predictions. Our best model began to find reasonable trajectories after approximately 15,000 steps of training with vanilla SGD as the discriminator update step and Adam for the generator. A qualitative analysis of many such random samples from the generator showed that when the ball was in flight, we often obtained a reasonably accurate tracking. When the ball is resting in the thrower's hand, the algorithm has a much more difficult time pinpointing the position of the ball. This is likely because we use the same weights for every frame of input and a relatively shallow CNN. Thus if the generator learns to identify the shape of the ball and thus adjust its features when the ball moves in each frame, when the ball is obscured by a hand, it could have any output between frames.

Our choice of background and ball color in hindsight did not make the task as easy as possible for the GAN. A more brightly colored, large ball would likely have been easier for the algorithm to locate but it was interesting nonetheless to see if the architecture could hold up to a more challenging vision problem. Our results show sufficient signs of tracking to suggest that this approach can work for more subtle problems with further tuning.

Figure 7: A representative selection of four images where the thrown ball is clearly visible (i.e. not occluded by hands or against a similarly colored background). The ball height that the generator network predicts, shown with a yellow line in each case, is clearly quite accurate.

Figure 8: A representative selection of four images where the thrown ball is quite difficult to see with the human eye (i.e. where it is in the thrower's hand or is in front of the door, which has a similar color). In these cases, the ball height that the generator predicts, shown as a yellow line in each case, is quite variable and does not appear very related to the height of the ball.

6. Conclusion

We observe in the results that the unsupervised approach we propose can produce good results for simple models (ball throwing with parabolic simulator), but the object detection becomes worse for more complicated ones (ball throwing with sinusoid simulator; pong ball dynamics). This seems to be caused due to the increased difficulty of precisely specifying a simulator that accurately matches the behavior of the object we're interested in tracking. This issue implies that our method might be difficult to apply successfully to more complicated models. Future work can explore this issue in more detail and verify the extent of this limitation.

As GANs become more commonplace, and as they are applied more frequently to SeqGAN problems [22] (i.e. problems which produce a sequence of output; as mentioned in section 2 these models are still quite rare) they will become more tractable for this and similar problems. GANs have been used primarily to generate fixed outputs, and even to do this most architectures have required a lot of tuning and specialized techniques. With similar techniques available for SeqGAN-style architectures, solving this problem more accurately may become more tractable. Without such advances, we suggest that architectures in this avenue will be difficult to pursue.

References

[1] Martín Abadi et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467* (2016).

[2] Martin Arjovsky and Léon Bottou. "Towards principled methods for training generative adversarial networks". In: *NIPS 2016 Workshop on Adversarial Training. In review for ICLR*. Vol. 2016. 2017.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein gan". In: *arXiv preprint arXiv:1701.07875* (2017).

[4] Alex Barron, Todor Markov, and Zack Swafford. *Label Free Object Tracking*. 2017. URL: `https://github.com/barronalex/Label-Free-Object-Tracking` (visited on 2017-06-12).

[5] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM". In: (1999).

[6] Ian Goodfellow. "NIPS 2016 Tutorial: Generative Adversarial Networks". In: *arXiv preprint arXiv:1701.00160* (2016).

[7] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[8] John D Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95.

[9] Ferenc Huszár. *Instance Noise: A trick for stabilising GAN training*. 2016. URL: http://www.inference.vc/instance-noise-a-trick-for-stabilising-gan-training/ (visited on 2017-06-12).

[10] Eric Jones, Travis Oliphant, and Pearu Peterson. “{SciPy}: open source scientific tools for {Python}”. In: (2014).

[11] Andrej Karpathy. *Pong AI with Policy Gradients*. 2016. URL: https://www.youtube.com/watch?v=YOW8m2YGtRg (visited on 2017-06-12).

[12] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[13] Xudong Mao et al. “Least squares generative adversarial networks”. In: *arXiv preprint ArXiv:1611.04076* (2016).

[14] Olof Mogren. “C-RNN-GAN: Continuous recurrent neural networks with adversarial training”. In: *arXiv preprint arXiv:1611.09904* (2016).

[15] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint arXiv:1511.06434* (2015).

[16] Tim Salimans et al. “Improved Techniques for Training GANs”. In: *arXiv preprint arXiv:1606.03498* (2017).

[17] Soumith Chintala (soumith). *How to Train a GAN? Tips and tricks to make GANs work*. 2016. URL: https://github.com/soumith/ganhacks (visited on 2017-06-12).

[18] Jost Tobias Springenberg. “Unsupervised and semi-supervised learning with categorical generative adversarial networks”. In: *arXiv preprint arXiv:1511.06390* (2015).

[19] Russell Stewart and Stefano Ermon. “Label-Free Supervision of Object Detectors with Adversarial Constraint Learning”.

[20] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. “The NumPy array: a structure for efficient numerical computation”. In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30.

[21] Tom White. “Sampling Generative Networks: Notes on a Few Effective Techniques”. In: *arXiv preprint arXiv:1609.04468* (2016).

[22] Lantao Yu et al. “Seqgan: sequence generative adversarial nets with policy gradient”. In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.