International Journal of Latest Engineering Research and Applications (IJLERA) ISSN: 2455-7137

Volume – 07, Issue – 02, February 2022, PP – 06-12

Rethinking Robustness in Machine Learning: Use of Generative Adversarial Networks for Enhanced Robustness

Ayse Arslan

Abstract: Machine learning (ML) is increasingly being used in real-world applications, so understanding the uncertainty and robustness of a model is necessary to ensure performance in practice. This paper explores approximations for robustness which can meaningfully explain the behavior of any black box model. Starting with a discussion on components of a robust model this paper offers some techniques based on the Generative Adversarial Network (GAN) approach to improve the robustness of a model. The study concludes that a clear understanding of robust models for ML allows to improve information for practitioners, and helps to develop tools that assess the robustness of ML. Also, ML tools and libraries could benefit from a clear understanding on how information should be presented and how these tools are used.

Keywords: ML, AI, robustness, safety, algorithm, software engineering, explainability interpretability

Introduction

Machine learning (ML) agents are increasingly deployed in the real world to make decisions and assist people in their daily lives. The classic definition of ML enables one to conceptualize ML as a growing resource of interactive, autonomous, and often self-learning agency, that can deal with tasks that would otherwise require human intelligence and intervention to be performed successfully. On the other hand,if a model is not robust, there will be a misalignment between the model and expectations of the human-beings. For example, how do models signal when they are likely to make a mistake?

To address these concerns, it is valuable to quantify how robust ML models are. In particular: How likely are ML models to make false statements across a range of contexts and questions?

This study aims to provide a clear understanding of robust models for ML in order to reduce variation as well as to develop tools that assess this robustness of ML. By exploring a hypothetical development model it also offers means to take a first step towards a theory of robust models in ML.

Review of Existing Studies

Machine learning (ML) is used to develop increasingly capable systems targeted at tasks like voice recognition, fraud detection, and the automation of vehicles.

In recent decades, researchers have proven that big data and machine learning algorithms reduce the need for providing ML systems with prior rules and knowledge.

One common argument in favor of ML algorithms is their ability to scale with the availability of data and compute resources. ML models can derive their behavior from data, without the need for explicit rules and prior knowledge. Another advantage of ML is that it can glean its own solutions from its training data, which are often more accurate than knowledge engineered by humans. Yet, most ML applications are based on supervised learning and require training data to be manually labeled by human annotators.

Theoretical justification for almost all ML methods relies upon the equality of the distributions from which the training and test data are drawn, yet, in many real-world applications, this equality is violated. For instance, reinforcement learning (RL) has proven to be particularly efficient at solving complicated problems yet, RL systems require manually defined reward functions or goals before they can learn the behaviors that help accomplish those goals. Also, the RL agent should be able to achieve many different goals by computing counterfactuals, learning causal models, and obtaining a deep understanding of how actions affect its environment in the long term.The combination of self-supervised and offline RL can help create agents that can create building blocks for learning new tasks and continue learning with little need for new data.

While some researchers proposed to use maximum soft max probability to detect misclassified examples others proposed to use ‘Trust Score’ to estimate the confidence in model predictions. Still, some other researchers suggested that the test accuracy of deep networks can be estimated by measuring disagreement rate between a pair of models independently trained via Stochastic Gradient Descent (SGD) and theoretically related this phenomenon to the well-calibrated nature of ensembles of SGD-trained models.

When it comes to robustness, interpretability is important for a variety of reasons (Caruana et al., 2017; Doshi-Velez & Kim, 2017), including safety and trust. Shafto et al. formalizes this intuition in a recursive

Bayesian model of human pedagogical reasoning (Shafto & Goodman, 2008; Shafto et al., 2012; 2014). In their model the probability a teacher selects an example e to teach a concept c is a soft maximization (with parameter α) over what the student's posterior probability of c will be. The student can then update their posterior accordingly.

Advances in deep learning over the last few decades have been driven by a few key elements. With a small number of simple but flexible mechanisms (i.e., inductive biases such as convolutions or sequence attention), increasingly large datasets, and more specialized hardware, neural networks can now achieve impressive results on a wide range of tasks, such as image classification or machine translation. However, the use of large models and datasets comes at the expense of significant computational requirements. Yet, recent works suggest that large model sizes might be necessary for strong generalization and robustness, so training large models while limiting resource requirements is becoming increasingly important. One promising approach involves the use of conditional computation: rather than activating the whole network for every single input, different parts of the model are activated for different inputs.

Much recent work has focused on learning emergent communication protocols in deeplearning based agents (Foerster et al., 2016; Sukhbaatar et al., 2016). However, these emergent protocols tend to be uninterpretable (Kottur et al., 2017) which could affect robustness of a model. A number of techniques have been suggested to encourage interpretability, such as limiting symbol vocabulary size (Mordatch & Abbeel, 2017), limiting memorization capabilities of the speaker (Kottur et al., 2017), or introducing auxiliary tasks such as image labelling based on supervision data (Lazaridou et al., 2016). Despite these modifications, the protocols can still be difficult to interpret.

One problem studied in the literature regarding robustness is finding a student-teacher pair such that the student can learn a set of concepts when given examples from the teacher (Jackson & Tomkins, 1992; Balbach & Zeugmann, 2009). However, it is difficult to formalize this problem in a way that avoids some turn-around solutions known as "coding tricks." A coding trick refers to a solution in which the teacher and student simply "collude" on a pre-specified protocol for encoding the concept through examples. Many additional constraints to the problem have been proposed to try to rule out coding tricks such as requiring the student to be able to learn through any superset of the teacher's examples (Goldman & Mathias, 1996), requiring the learned protocols to work for any ordering of the concepts or examples (Zilles et al., 2011), requiring the student to learn all concepts plus their images under primitive recursive operators (Ott & Stephan, 2002), and giving incompatible hypothesis spaces to the student and teacher (Angluin & Krik¸ is, 1997).

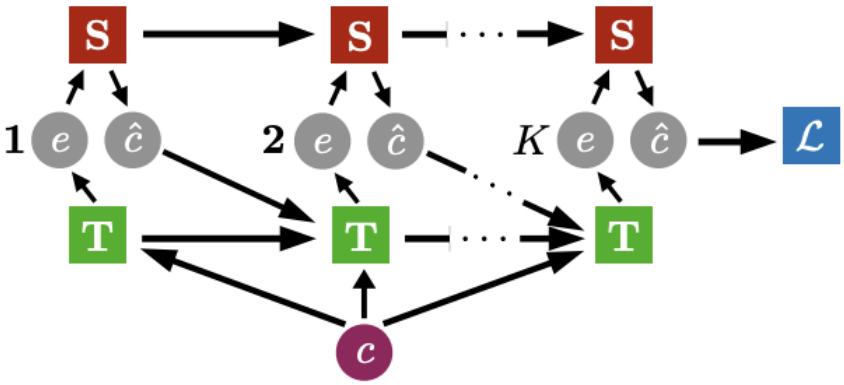

Figure 1. Visualization of Interpretability in Machine Learning based on Teacher-Student (T-S) interaction

Figure 1 includes a visualization of a student-teacher (S-T) interaction win the context of ML. At each step, the teacher T takes in the true concept and the student's S's last estimate of the concept and puts an example for S so that S outputs its new estimate. Some definitions of interpretability in the literature (Doshi-Velez & Kim, 2017; Weller, 2017; Lipton, 2016) include, but are not limited to:

1. Evaluating how similar a teacher's strategies are to intuitive human-designed strategies in each task
2. Evaluating the effectiveness of a teacher's strategy at teaching human-beings.

Interpretability of a model can capture a range of different types of concepts such as rule-based, probabilistic, boolean, and hierarchical concepts. It is often difficult to define naturally-occurring concepts via

rules and some examples of a concept can seem more prototypical than others (e.g sparrow vs peacock) (Rosch & Mervis, 1975), and this is not captured by simply modeling the concept as a set of rules that must be satisfied. An alternative approach models concept learning as estimating the probability density of the concept (Anderson, 1991; Ashby & Alfonso-Reese, 1995; Fried & Holyoak, 1984; Griffiths et al., 2008). (Shafto et al., 2014) investigate teaching and learning unimodal distributions. Moreover, an object can have many properties, but only a few of them may be relevant for deciding whether the object belongs to a concept or not. In this case, the purpose of a ML algorithm would be to see what strategy T learns to quickly teach S which properties are relevant to a concept. In addition to this,human-defined concepts are often hierarchical as human-beings are sensitive to taxonomical structure when learning how to generalize to a concept from an example (Xu & Tenenbaum, 2007). In such a case, the ML algorithm aims to test how T learns to teach when the concepts form a hierarchical structure.

When it comes to ensuring robustness of a model, adversarial ML, and more generally the security and privacy of ML , encompasses a line of work that seeks to understand the behavior of models and learning algorithms in the presence of adversaries to ensure robustness. Adversarial examples are examples that are created by making small perturbations to the input designed to significantly increase the loss incurred by a machine learning model (Szegedy et al., 2014; Goodfellow et al., 2015).

Carlini et al. (2019); Stock &Cissé (2018) define adversarial robustness as the minimum distance in the input domain required to change the model's output prediction by constructing an adversarial attack. Carlini et al. (2019), states that easily attackable data are often outliers in the underlying data distribution and then use adversarial robustness to determine an improved ordering for curriculum learning.

In another branch of research, neural networks are shown to lack adversarial robustness – small perturbations to the input can successfully fool classifiers into making incorrect predictions (Szegedy et al., 2014; Goodfellow et al., 2014; Carlini& Wagner, 2017b; Madry et al., 2017; Qin et al., 2020b).

A common misconception is that adversarial training is equivalent to training on noisy examples. Noise is actually a far weaker regularizer than adversarial perturbations. There are some previous works adding random noise to the input and hidden layer during training, to prevent overfitting (e.g. (Sietsma& Dow, 1991; Poole et al., 2013)). Adversarial and virtual adversarial training requires only one hyperparameter, and has a straightforward interpretation as robust optimization.

Generative adversarial networks (GANs) are a recently proposed class of generative models in which a generator is trained to optimize a cost function that is being simultaneously learned by a discriminator. The discriminator is tasked with classifying its inputs as either the output of the generator, or actual samples from the underlying data distribution p(x). The goal of the generator is to produce outputs that are classified by the discriminator as coming from the underlying data distribution (Szegedy et al., 2014; Goodfellow et al., 2015).

In recent years, deep generative models have dramatically pushed forward the state-of-the-art in generative modelling in terms of increased level of robustness (Zhu et al., 2016). Many of the most successful approaches include variational auto encoders (VAEs) (Kingma& Welling, 2014; Rezende et al., 2014), generative adversarial networks (GANs) (Goodfellow et al., 2014), generative moment matching networks (GMMNs) (Li & Swersky, 2015; Dziugaite et al., 2015), and nonlinear independent components estimation (Dinh et al., 2014).

Ho et al. [10, 9] previously presented a GAN-like algorithm for imitation learning, where the goal is to recover a policy that matches the expert demonstrations. The proposed algorithm, called generative adversarial imitation learning (GAIL), has an adversarial structure.

Chen and his colleagues (2019) introduce InfoGAN — an extension of GAN that learns disentangled and interpretable representations for images. A regular GAN achieves the objective of reproducing the data distribution in the model, but the layout and organization of the code space is underspecified in terms of mapping the unit Gaussian to images. The InfoGAN imposes additional structure on this space by adding new objectives that involve maximizing the mutual information between small subsets of the representation variables and the observation.

Moreover, Bachman & Precup [1] suggested that in order to increase robustness, data generation can be converted into a sequential decision-making problem and solved with a reinforcement learning method.

Pfau &Vinyals drew a connection between the optimization problems in GANs and actor-critic methods in reinforcement learning, suggesting how ideas for stabilizing training in one domain could be beneficial for the other [19]. As the authors point out, these optimization tricks could also be useful for imitation learning algorithms with the same two-level optimization structure.

Furthermore, In reinforcement learning, state representations are used to tractably deal with largeproblem spaces. State representations serveboth to approximate the value function with few parameters, but also to generalize tonewly encountered states. Their features may be learned implicitly (as part of a neural network) or explicitly (for example, the successor representation of Dayan (1993)). A group of reserachers at Google Brain provide an informative bound on the generalization errorarising from a specific state representationbased on the notion of effective dimension which measures the

degree towhich knowing the value at one state informsthe value at other states. This bound appliesto any state representation and quantifies the natural tension between representations thatgeneralize well and those that approximate well.

It should be emphasized that all of these sophisticated algorithms and techniques can introduce new forms of statistical abuses and while accounting for uncertainty in results is not a panacea, it provides a strong foundation for trustworthy results on which the community can build upon, with increased confidence. As Agarwal et al (2021) assert, ignoring the statistical uncertainty in deep RL results gives a false impression of fast scientific progress in the field.

The next section elaborates on the suggested model more in detail based on these results of existing studies.

Implementation Model

Data Generation

For the sake of practicality, the assumption is that there is some large collection of images, such as the 1.2 million images in the ImageNet dataset (bearing in mind that this could eventually be a large collection of images or videos from the internet or robots). If one were to resize each image to have width and height of 256 (as is commonly done), the dataset would result in one large 1,200,000x256x256x3 (about 200GB) block of pixels. These images could be referred to as "samples from the true data distribution". A generative model to train to generate images like this from scratch can be constructed in this way. As a result, the resulting generative model would be one large neural network that outputs images and referred to "samples from the model".

One such recent model is the DCGAN network from Radford et al. which takes as input 100 random numbers drawn from a uniform distribution (referred to as a code, or latent variables) and outputs an image. As the code is changed incrementally, the generated images do too — which shows the model has learned features to describe how the world looks, rather than just memorizing some examples.

DCGAN is initialized with random weights, so a random code plugged into the network would generate a completely random image. Yet, given the millions of parameters in the network, the goal is to find a setting of these parameters that makes samples generated from random codes look like the training data. In other words, the aim is to make the model distribution match the true data distribution in the space of images.

Training a robust model

We assume that there is a newly-initialized network to generate 200 images, each time starting with a different random code. The question is: how should we adjust the network's parameters to encourage it to produce slightly more believable samples in the future for the sake of robustness? It should be taken into account that there is no simple supervised setting and no explicit desired targets for 200 generated images; the aim is tomake them look real. One clever approach around this problem is to follow the GAN approach. One can introduce a second discriminator network (usually a standard convolutional neural network) that tries to classify if an input image is real or generated. For instance, one could feed the 200 generated images and 200 real images into the discriminator and train it as a standard classifier to distinguish between the two sources. Yet,one can also backpropagate through both the discriminator and the generator to find how to change the generator's parameters to make its 200 samples slightly more confusing for the discriminator. So, while the discriminator is trying to distinguish real images from fake images the generator is trying to create images that make the discriminator think they are real. In the end, the generator network is outputting images that are indistinguishable from real images for the discriminator.

In both cases the samples from the generator start out noisy and chaotic, and over time converge to have more plausible image statistics.

Set-Up

Most generative models can have this basic setup, yet still could differ in the details. Below are some common examples of generative model approaches:

- Generative Adversarial Networks (GANs), pose the training process as a game between two separate networks: a generator network (as seen above) and a second discriminative network that tries to classify samples. Every time the discriminator notices a difference between the two distributions the generator adjusts its parameters slightly to make it go away, until at the end (in theory) the generator exactly reproduces the true data distribution and the discriminator is guessing at random, unable to find a difference.
- Variational Autoencoders (VAEs) allow one to formalize this problem in the framework of probabilistic graphical models where one is maximizing a lower bound on the log likelihood of the data.
- Autoregressive models such as PixelRNN train a network that models the conditional distribution of every individual pixel given previous pixels (to the left and to the top).

All of these approaches have their pros and cons. While VAEs allow one to perform both learning and efficient Bayesian inference in sophisticated probabilistic graphical models with latent variables, their generated samples tend to be slightly blurry. GANs currently generate the sharpest images, yet they are more difficult to optimize due to unstable training dynamics. PixelRNNs have a very simple and stable training process (softmax loss) yet, they are relatively inefficient during sampling and don't easily provide simple low-dimensional *codes* for images.

In addition to generating pretty pictures, one can also use an approach for semi-supervised learning with GANs that involves the discriminator producing an additional output indicating the label of the input. This approach allows one to obtain state of the art results on MNIST, SVHN, and CIFAR-10 in settings with very few labeled examples.

The use case

One can give discriminator an entire minibatch of samples as input, rather than just one sample. Thus, the discriminator can tell whether the generator just constantly produces a single image. With the collapse discovered, gradients will be sent to the generator to correct the problem.

The next step is to prototype the idea on MNIST and CIFAR-10. This requires prototyping a small model as quickly as possible, running it on real data, and inspecting the result. However, deep learning (and AI algorithms in general) must be scaled to be truly impressive — a small neural network is a proof of concept, but a big neural network actually solves the problem and is useful.

Infrastructure

Software

The vast majority of code is written in Python, as engineers mostly use TensorFlow (or Theano in special cases) for GPU computing. Researchers also sometimes use higher-level frameworks like Keras on top of TensorFlow. Below is a sample of a TensorFlow code (Figure 2).

```
with arg_scope([conv2d, ar_multiconv2d]):
    x = tf.nn.elu(input)
    x = conv2d("down_conv1", x, 4 * z_size + h_size * 2)
    pz_mean, pz_logsd, rz_mean, rz_logsd, down_context, h_det = split(x, 1, [z_size] * 4 + [

    prior = DiagonalGaussian(pz_mean, 2 * pz_logsd)
    posterior = DiagonalGaussian(rz_mean + self.qz_mean, 2 * (rz_logsd + self.qz_logsd))
    context = self.up_context + down_context

    if self.mode in ["init", "sample"]:
```

Figure 2. Sample TensorFlow code

Hardware

For an ideal batch job, doubling the number of nodes in a cluster will halve the job's runtime. Top performance also requires top-of-the-line GPUs in addition to the use of a lot of CPU for simulators, reinforcement learning environments, or small-scale models (which run no faster on a GPU).

Provisioning

Infrastructure should present a simple interface, and usability is as important as functionality. It is suggested to use a consistent set of tools to manage all of existing servers and configure them as identically as possible.

```
resource "aws_autoscaling_group" "kubernetes-worker" {
  count = "${length(keys(var.worker_groups))}"
  name = "${lookup(var.worker_groups, count.index)} kubernetes-worker - ${var.cluster_version}"
  launch_configuration = "${element(aws_launch_configuration.kubernetes-worker.*.name, count.index)}"

  tag {
    key = "Name"
    value = "kubernetes-group-${lookup(var.worker_groups, count.index)}"
    propagate_at_launch = true
  }

  tag {
    key = "KubernetesCluster"
```

Figure 3. Sample Terraform config for managing Auto Scaling groups

One can use Terraform (Figure 3) to set up AWS cloud resources (instances, network routes, DNS records, etc). Terraform creates, modifies, or destroys the running cloud resources to match configuration files.

Orchestration

As scalable infrastructure often ends up making the simple cases harderequal effort should be put onto the infrastructure for small- and large-scale jobs.

One can provide a cluster of SSH nodes (both with and without GPUs) for ad-hoc experimentation, and run Kubernetes as the cluster scheduler for physical and AWS nodes.Kubernetes requires each job to be a Docker container, which results in dependency isolation and code snapshotting. However, building a new Docker container can add precious extra seconds to a researcher's iteration cycle, so one can also provide tooling to transparently ship code from a researcher's laptop into a standard image (Figure 4).

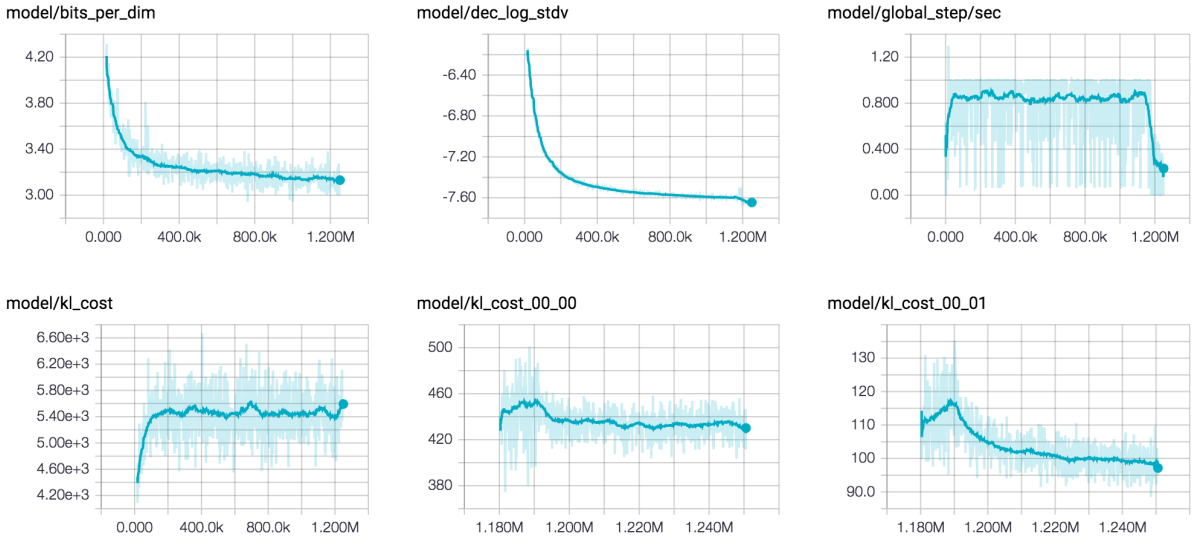

Figure 4. Sample model learning curves in Tensor Board

One can expose Kubernetes's flannel network directly to researchers' laptops, allowing users seamless network access to their running jobs. This is especially useful for accessing monitoring services such as TensorBoard.

The Kubernetes ecosystem provides low-friction tooling, logging, monitoring, ability to manage physical nodes separately from the running instances.

Release of kubernetes-ec2-autoscaler includes a batch-optimized scaling manager for Kubernetes. It runs as a normal Pod on Kubernetes and requires only that worker nodes are in Auto Scaling groups. The autoscaler works by polling the Kubernetes master's state, which contains everything needed to calculate the cluster resource ask and capacity. If there's excess capacity, it drains the relevant nodes and ultimately terminates them. If more resources are needed, it calculates what servers should be created and increases Auto Scaling group sizes appropriately (or simply uncordons drained nodes, which avoids new node spinup time).

'kubernetes-ec2-autoscaler' handles multiple Auto Scaling groups, resources beyond CPU (memory and GPUs), and fine-grained constraints on jobs such as AWS region and instance size. Additionally, bursty workloads can lead to Auto Scaling Groups timeouts and errors, so that even AWS does not have infinite capacity. In these cases, kubernetes-ec2-autoscaler detects the error and overflows to a secondary AWS region.

Such an infrastructure aims to maximize the productivity of deep learning researchers, allowing them to focus on the science of building robust ML models.

Conclusion and Future Work

A clear understanding of robust models for ML allows to improve information for practitioners, and helps to develop tools that assess the robustness of ML. However, a wide range of subsequent research towards an encompassing theory of robust models in ML might still be required. This includes how ML models are shared or documented as well as work on threat specific taxonomies. Also, ML tools and libraries could benefit from a clear understanding on how information should be presented and how these tools are used.

References

[1]. Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. Hindsight Experience Replay. In Advances in Neural Information Processing Systems, pp. 5048–5058, 2017.

[2]. Bahdanau, D., Hill, F., Leike, J., Hughes, E., Kohli, P., and Grefenstette, E. Learning to Follow Language Instructions with Adversarial Reward Induction. arXiv:1806.01946, 2018.

[3]. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI Gym. arXiv:1606.01540, 2016.

[4]. Christiano, P., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep Reinforcement Learning from Human Preferences. In Advances in Neural Information Processing Systems, pp. 4299–4307, 2017.

[5]. Co-Reyes, J., Gupta, A., Sanjeev, S., Altieri, N., DeNero, J., Abbeel, P., and Levine, S. Guiding Policies with Language via Meta-Learning. arXiv:1811.07882, 2018.

[6]. Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. OpenAI Baselines. https://github. com/openai/baselines, 2017.

[7]. Fu, J., Singh, A., Ghosh, D., Yang, L., and Levine, S. Variational Inverse Control with Events: A General Framework for Data-Driven Reward Definition. In Advances in Neural Information Processing Systems, pp. 8538–8547, 2018.

[8]. Hermann, K. M., Hill, F., Green, S., Wang, F., Faulkner, R., Soyer, H., Szepesvari, D., Czarnecki, W. M., Jaderberg, M., Teplyashin, D., et al. Grounded Language Learning in a Simulated 3D World. arXiv:1706.06551, 2017.

[9]. Ibarz, B., Leike, J., Pohlen, T., Irving, G., Legg, S., and Amodei, D. Reward learning from human preferences and demonstrations in Atari. In Advances in Neural Information Processing Systems, pp. 8022–8034, 2018.

[10]. Lakkaraju, S. H. Bach, and J. Leskovec. Interpretable decision sets: A joint framework for description and prediction. In KDD, 2016.

[11]. Mohan Zhou, Yalong Bai, Wei Zhang, Tiejun Zhao, and Tao Mei. Look-into-object: Self-supervised structure modeling for object recognition. In CVPR, 2020.

[12]. Muhammad Abdullah Jamal, Matthew Brown, Ming-Hsuan Yang, Liqiang Wang, and Boqing Gong. Rethinking class-balanced methods for long-tailed visual recognition from a domain adaptation perspective, 2020.

[13]. Nair, A., Pong, V., Dalal, M., Bahl, S., Lin, S., and Levine, S. Visual reinforcement learning with imagined goals. In Advances in Neural Information Processing Systems, pp. 9208–9219, 2018.

[14]. Pomerleau, D. Efficient Training of Artificial Neural Networks for Autonomous Navigation. Neural Computation, 3(1):88–97, 1991.

[15]. Reddy, S., Dragan, A. D., and Levine, S. Shared Autonomy via Deep Reinforcement Learning. arXiv:1802.01744, 2018.

[16]. Ross, S., Gordon, G., and Bagnell, D. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 627–635, 2011.

[17]. Sadigh, D., Dragan, A., Sastry, S., and Seshia, S. Active Preference-Based Learning of Reward Functions. In Robotics: Science and Systems, 2017. Human Interaction and Interpretability Paper

[18]. Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal Value Function Approximators. In International Conference on Machine Learning, pp. 1312–1320, 2015.

[19]. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal Policy Optimization Algorithms. arXiv:1707.06347, 2017.

[20]. Singh, A., Yang, L., Hartikainen, K., Finn, C., and Levine, S. End-to-End Robotic Reinforcement Learning without Reward Engineering. arXiv preprint arXiv:1904.07854, 2019.

[21]. Sutton, R., Precup, D., and Singh, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial intelligence, 112(1-2): 181–211, 1999.

[22]. Yuliang Zou, Zizhao Zhang, Han Zhang, Chun-Liang Li, Xiao Bian, Jia-Bin Huang, and Tomas Pfister. Pseudoseg: Designing pseudo labels for semantic segmentation. ICLR, 2021

[23]. Zhao, X., Robu, V., Flynn, D., Salako, K., Strigini, L.: Assessing the safety and reliability of autonomous vehicles from road testing. In: the 30th Int. Symp. on Software Reliability Engineering. pp. 13–23. IEEE, Berlin, Germany (2019)

