The Journal of Open Source Software

TorchGAN: A Flexible Framework for GAN Training and Evaluation

Avik Pal[1] **and Aniket Das**[1]

1 Indian Institute of Technology Kanpur

DOI: 10.21105/joss.02262

Software

- Review
- Repository
- Archive

Editor: Pending Editor

Submitted: 26 May 2020
Published: 27 May 2020

Abstract

TorchGAN is a PyTorch based framework for writing succinct and comprehensible code for training and evaluation of Generative Adversarial Networks. The framework's modular design allows effortless customization of the model architecture, loss functions, training paradigms, and evaluation metrics. The key features of TorchGAN are its extensibility, built-in support for a large number of popular models, losses and evaluation metrics, and zero overhead compared to vanilla PyTorch. By using the framework to implement several popular GAN models, we demonstrate its extensibility and ease of use. We also benchmark the training time of our framework for said models against the corresponding baseline PyTorch implementations and observe that TorchGAN's features bear almost zero overhead.

Introduction

Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) are a class of deep generative models that formulate the model estimation problem as an adversarial game between two neural networks, a Generator representing an implicit generative distribution, and a Discriminator that differentiates between samples from said implicit distribution and the true data distribution. The implicit distribution recovers the data distribution when the game reaches equilibrium. Apart from being one of the most popular approaches for generative modeling and unsupervised learning tasks in Computer Vision, with diverse applications such as photo-realistic image generation (Brock, Donahue, & Simonyan, 2018; Karras, Aila, Laine, & Lehtinen, 2017), image super-resolution (Ledig, Theis, Huszar, Caballero, & al., 2017), image-to-image translation (Zhu, Park, Isola, & Efros, 2017) and video generation (Clark, Donahue, & Simonyan, 2019; Tulyakov, Liu, Yang, & Kautz, 2018), it has also found applicability in domains such as Natural Language Processing (Zhang et al., 2017) and Time Series Analysis (Esteban, Hyland, & Rätsch, 2017).

GANs generally share a standard design paradigm, with the building blocks comprising one or more generator and discriminator models, and the associated loss functions for training them. TorchGAN makes use of this design similarity by exposing a simple API for customizing these blocks. The interaction between these components at training time is facilitated by a highly robust trainer which automatically adapts to user-defined GAN models and losses. TorchGAN provides an extensive and continually expanding collection of popular GAN models, losses, evaluation metrics, and stability-enhancing features, which can either be used off the shelf or easily extended or combined to design more sophisticated models effortlessly. With the above design principles in mind, we aim to improve upon existing GAN training frameworks such as TFGAN (Shor, 2017), HyperGAN (Community, 2016), and IBM GAN-Toolkit (Raunak Sinha, 2018) on the aspects of extensibility, the richness of the feature set and documentation.

Implementing Models in TorchGAN

The core of the TorchGAN framework is a highly versatile trainer module, responsible for its flexibility and ease of use. The trainer requires specification of the generator and the discriminator architecture along with the optimizers associated with each of them, represented as a dictionary, as well as the list of associated loss functions, and optionally, evaluation metrics. We provide an illustrative example for training DCGAN on CIFAR10. One can either choose from the in-built implementations of popular GAN models, losses and metrics or define custom variants of their own with minimal effort by extending the appropriate base classes. This extensibility is widely useful in research applications where the user only needs to write code for the model architecture and/or the loss function. The trainer automatically handles the intricacies of training with custom models/losses. The trainer also supports the usage of multiple generators and discriminators, allowing training of more sophisticated models such as Generative Multi Adversarial Networks (Durugkar, Gemp, & Mahadevan, 2016). Performance visualization is handled by a customizable Logger object, which, apart from console logging, currently supports the Tensorboard and Vizdom backends.

```
train_dataset = dsets.CIFAR10(root='./cifar10', train=True,
    transform=transforms.Compose([transforms.ToTensor(),
        transforms.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))]
    ), download=True)
train_loader = data.DataLoader(train_dataset, batch_size=128, shuffle=True)
trainer = Trainer(
    {"generator": {"name": DCGANGenerator, "args": {"out_channels": 3,
        "step_channels": 16}, "optimizer": {"name": Adam, "args":
            {"lr": 0.0002, "betas": (0.5, 0.999)}}},
    "discriminator": {"name": DCGANDiscriminator, "args":
        {"in_channels": 3, "step_channels": 16}, "optimizer": {"name": Adam,
            "args": {"lr": 0.0002, "betas": (0.5, 0.999)}}}},
    [MinimaxGeneratorLoss(), MinimaxDiscriminatorLoss()], sample_size=64,
    epochs=20)
trainer(train_loader)
```

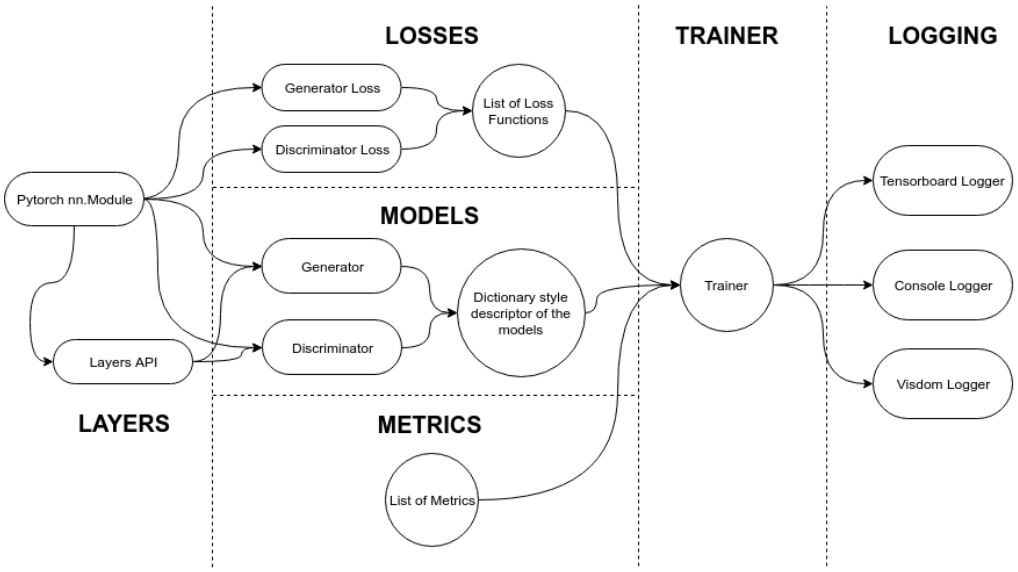

Figure 1: Overview of TorchGAN Design.

Existing Frameworks

TorchGAN provides high-quality implementations of various GAN models, metrics for evaluating GANs, and various approaches for improving the stability of GAN training. We provide an overview of the features that are provided off the shelf by TorchGAN and compare them with the ones provided by other frameworks. Note that the list is not exhaustive as the modular and extensible structure of TorchGAN allows one to extend or modify these features, or use them as building blocks for more sophisticated models.

Table 1: Supported features of different frameowrks. Features officiallt supported are marked "✓", under active development are marked "⋆", and those currently unsupported are left blank.

	TorchGAN	TFGAN	IBM GAN-Toolkit	HyperGAN
Vanilla GAN	✓	✓	✓	✓
DCGAN	✓	✓	✓	✓
Wasserstein GAN	✓	✓	✓	✓
Wasserstein GAN-GP	✓	✓	✓	✓
Inception Score	✓	✓	✓	
InfoGAN	✓	✓		✓
CycleGAN	✓	✓		✓
Least Squares GAN	✓	✓		✓
Auxillary Classifier GAN	✓	✓		
Spectral Normalization GAN	✓	✓	✓	
Self Attention GAN	✓	✓		
Conditional GAN	✓		✓	
Energy Based GAN	✓			✓
Boundary Equilibrium GAN	✓			
DRAGAN-GP	✓			
Binary GAN	✓			
Adversarial Autoencoders	✓			
Historical Averaging	✓			
Feature Matching	✓			
Minibatch Discrimination	✓			
Frechet Inception Distance	⋆	✓	✓	
Progressive GAN	⋆	✓		
Adversarially Learned Inference	⋆			✓
Star GAN		✓		

Table 1 summarizes the features supported by a variety of open-source GAN frameworks. It suggests that TorchGAN supports the widest variety of features among the frameworks being considered. For comparison, we only consider the models present in the official repository of a given framework or an associated officially maintained model-zoo/examples repository. We avoid comparisons with projects like Pytorch-GAN[1], Keras-GAN[2], etc., as these are not frameworks and hence cannot be extended to newer models.

Performance

In order to demonstrate that TorchGAN incurs zero training overhead despite the high level of abstraction it provides, we compare the training time of TorchGAN with vanilla PyTorch

[1] https://github.com/eriklindernoren/PyTorch-GAN
[2] https://github.com/eriklindernoren/Keras-GAN

implementations of DCGAN (Radford, Metz, & Chintala, 2015), CGAN (Mirza & Osindero, 2014), BEGAN (Berthelot, Schumm, & Metz, 2017) and WGAN-GP (Gulrajani, Ahmed, Arjovsky, Dumoulin, & Courville, 2017). Table 2 reports the training time for TorchGAN and Pytorch for 1 epoch, averaged over 8 runs.

Table 2: Average Training Time: TorchGAN vs Pytorch Baselines. DCGAN is trained on CIFAR-10, and the other models are trained on MNIST.

	DCGAN	CGAN	WGAN-GP	BEGAN
TorchGAN	**15.9s ± 0.64s**	**21.8s ± 0.43s**	**30.6s ± 1.35s**	**86.0s ± 0.62s**
Pytorch	16.7s ± 0.24s	22.4s ± 0.52s	31.1s ± 0.97s	87.0s ± 0.27s

For a fair comparison, we disable any form of logging and compute the training time using the $\%timeit$ magic function. We train the models on the CIFAR10 (Krizhevsky, 2009) and MNIST datasets, with a batch size of 128, on an Nvidia GTX Titan X GPU.

Conclusion and Future Work

We present the features of the TorchGAN framework and demonstrate its extensibility, ease of use and efficiency. Future work and extensions under active development include, integration of GAN models for video generation, generalization of the training loop to support Inference GAN models, such that they can be conveniently modified and extended, addition of features such as Adaptive Instance Normalization layers, and expanding the model zoo and documentation to cover more sophisticated examples such as Multi Agent-GAN training. We also envision the extension of the framework to domains beyond Computer Vision by adding support for NLP and Time Series GAN models.

References

Berthelot, D., Schumm, T., & Metz, L. (2017). BEGAN: Boundary Equilibrium Generative Adversarial Networks. Retrieved from http://arxiv.org/abs/1703.10717

Brock, A., Donahue, J., & Simonyan, K. (2018). Large scale gan training for high fidelity natural image synthesis. Retrieved from http://arxiv.org/abs/1809.11096

Clark, A., Donahue, J., & Simonyan, K. (2019). Efficient video generation on complex datasets. Retrieved from http://arxiv.org/abs/1907.06571

Community, K. (2016). HyperGAN. *GitHub repository*. https://github.com/HyperGAN/HyperGAN; GitHub.

Durugkar, I., Gemp, I., & Mahadevan, S. (2016). Generative Multi-Adversarial Networks. Retrieved from http://arxiv.org/abs/1611.01673

Esteban, C., Hyland, S. L., & Rätsch, G. (2017). Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., et al. (2014). Generative Adversarial Nets. In *Advances in neural information processing systems* (pp. 2672–2680).

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. (2017). Improved Training of Wasserstein GANs. Retrieved from http://arxiv.org/abs/1704.00028

Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation. Retrieved from http://arxiv.org/abs/1710.10196

Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. In. Citeseer.

Ledig, C., Theis, L., Huszar, F., Caballero, & al. (2017). Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. doi:10.1109/cvpr.2017.19

Mirza, M., & Osindero, S. (2014). Conditional Generative Adversarial Nets. Retrieved from http://arxiv.org/abs/1411.1784

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. Retrieved from http://arxiv.org/abs/1511.06434

Raunak Sinha, A. S., Naveen Panwar. (2018). IBM GAN Toolkit. *GitHub repository*. https://https://github.com/IBM/gan-toolkit; GitHub.

Shor, J. (2017). Tensorflow GAN. *GitHub repository*. https://github.com/tensorflow/models/tree/master/research/gan; GitHub.

Tulyakov, S., Liu, M.-Y., Yang, X., & Kautz, J. (2018). MoCoGAN: Decomposing Motion and Content for Video Generation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhang, Y., Gan, Z., Fan, K., Chen, Z., Henao, R., Shen, D., & Carin, L. (2017). Adversarial feature matching for text generation. In *Proceedings of the 34th international conference on machine learning-volume 70* (pp. 4006–4015). JMLR. org.

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. *2017 IEEE International Conference on Computer Vision (ICCV)*. doi:10.1109/iccv.2017.244