torchgan Documentation

Release 0.0.1

Avik Pal and Aniket Das

Dec 19, 2018

Getting Started

The `torchgan` package consists of various generative adversarial networks and utilities that have been found useful in training them. This package provides an easy to use API which can be used to train popular gans as well as develop newer variants.

CHAPTER 1

Installation

Follow the following instructions to set up *torchgan*. *Torchgan* is tested and known to work on major *linux distrubutions*. If you face any problem with other *operating systems* feel free to file an *issue*.

1.1 Pip Installation

Installing via *pip* is currently unavailable. It will be available soon.

1.2 Conda Installation

Installing via *conda* is currently unavailable. It will be available once we are at *v0.1*

1.3 Install from Source

```
$ git clone https://github.com/torchgan/torchgan
$ cd torchgan
$ python setup.py install
```

CHAPTER 2

Contributing

Contributions are always welcome. Follow the following guidelines while contributing :-

1. If contributing a *new feature*, first open an issue on github. Describe the feature and provide some references. Also clarify why it shall be a good feature to have in the *core library* and not simply as a *representative example*.
2. If submitting a *bug fix*, file the issue on github. Make sure the bug exists on the *master*.
3. If submitting a *new model*, open a PR in the *model zoo* repository. Follow the contribution guidelines present there.
4. Also fell free to submit *documentation changes*.

For you PR to be merged it must strictly adhere to the style guidelines, we use *flake8* for that purpose. Also all existing tests must pass. No breaking changes will be accepted unless when we are making a change in the *major version*. Also be sure to add *tests* and *documentation* for any code that you submit.

CHAPTER 3

Starter Example

As a starter example we will try to train a *DCGAN* on *CIFAR-10*. *DCGAN* is in-built into to the library, but let it not fool you into believing that we can only use this package for some fixed limited tasks. This library is fully customizable. For that have a look at the *Examples*.

But for now let us just use this as a small demo example

First we import the necessary files

```
import torch
import torchvision
from torch.optim import Adam
import torch.utils.data as data
import torchvision.datasets as dsets
import torchvision.transforms as transforms
from torchgan import *
from torchgan.models import SmallDCGANGenerator, SmallDCGANDiscriminator
from torchgan.losses import MinimaxGeneratorLoss, MinimaxDiscriminatorLoss,
from torchgan.trainer import Trainer
```

Now write a function which returns the *data loader* for *CIFAR10*.

```
def cifar10_dataloader():
    train_dataset = dsets.CIFAR10(root='/data/avikpal', 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)
    return train_loader
```

Now lets us create the *Trainer* object and pass the *data loader* to it.

```
trainer = Trainer(SmallDCGANGenerator(out_channels=3, step_channels=16),
                  SmallDCGANDiscriminator(in_channels=3, step_channels=16),
                  Adam, Adam, [MinimaxGeneratorLoss(), MinimaxDiscriminatorLoss()],
```

(continues on next page)

(continued from previous page)

```
                  sample_size=64, epochs=50,
                  optimizer_generator_options={"lr": 0.0002, "betas": (0.5, 0.999)},
                  optimizer_discriminator_options={"lr": 0.0002, "betas": (0.5, 0.
↪999)})

trainer(cifar10_dataloader())
```

Now log into *tensorboard* and visualize the training process.

CHAPTER 4

torchgan.losses

This losses subpackage is a collection of popular loss functions used in the training of GANS. Currently the following losses are supported:

- *Loss*
- *Least Squares Loss*
- *Minimax Loss*
- *Boundary Equilibrium Loss*
- *Energy Based Loss*
- *Wasserstein Loss*
- *Mutual Information Penalty*

These losses are tested with the current available trainers. So if you need to implement you custom loss for using with the `trainer` it is recommended that you subclass the `GeneratorLoss` and `DiscriminatorLoss`

4.1 Loss

4.2 Least Squares Loss

4.3 Minimax Loss

4.4 Boundary Equilibrium Loss

4.5 Energy Based Loss

4.6 Wasserstein Loss

4.7 Mutual Information Penalty

CHAPTER 5

torchgan.metrics

This subpackage provides various metrics that are available to judge the performance of GANs. Currently available metrics are:

5.1 Metric

5.2 Classifier Score

CHAPTER 6

torchgan.models

This models subpackage is a collection of popular GAN architectures. It has the support for existing architectures and provides a base class for extending to any form of new architecture. Currently the following models are supported:

- *GAN*
- *DCGAN*
- *Conditional GAN*
- *InfoGAN*

You can construct a new model by simply calling its constructor.

```
>>> import torchgan.models as models
>>> dcgan_discriminator = DCGANDiscriminator()
>>> dcgan_generator = DCGANGenerator()
```

All models follow the same structure. There are additional customization options. Look into the individual documentation for such capabilities.

6.1 GAN

6.2 DCGAN

6.3 Conditional GAN

6.4 InfoGAN

CHAPTER 7

torchgan.trainer

This subpackage provides ability to perform end to end training capabilities of the Generator and Discriminator models. It provides strong visualization capabilities using tensorboardX. In most cases you will need to overwrite the `generator_train_iter()` and `discriminator_train_iter()`.

Currently supported Trainers include:

CHAPTER 8

Example

The examples are not ready currently. But rest assured it will be fixed soon.

For now head over to the *model-zoo* repo for seeing how to write your own models.