UDACITY

Return to Classroom

Generate Faces

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Good job overall with the training part and choices you made around ideal GAN implementation. Training GANs and achieving good results is a tricky thing and you need to experiment a lot to get decent results.

Also, do try to work with bigger images to get better results and other GANs for multiple of use cases

For instance StyleGAN has some pretty interesting outputs

Here are some resources if you wish to explore GANs more.

- This is an excellent resource which covers an application of GANs for image completion and it properly explains the project as well.
- Ian Goodfellow's Tutorial on GANs for NIPS 2016 link along with this video tutorial
- Here is an awesome list of implementations in TensorFlow if you want to explore TF as well on different GAN related algorithms.

Best of luck and hope you apply all that you have learnt in the nanodegree to some interesting projects at your workplace and/or at your home right now :)

Required Files and Tests

The project submission contains the project notebook, called "dlnd_face_generation.ipynb".

All files present. You can avoid printing the unzipping process as it fills up the output buffer of the first code cell.

All the unit tests in project have passed.

Good job, all the unit tests passed. Quality work!

- You can definitely use more assert statements to check size of images and network shapes if you want. Also, use logging for different warnings/errors so that with version deprecation of functions, it is easy to find the culprit breaking the code within the logs.
- People using different languages may have different opinions about whether to use assertions in production code or not but everyone agrees that assertions are great for catching bugs. This is especially true of a dynamically type-checked language like Python, where a wrong variable type or shape can cause errors at runtime. Here's a short guide on how to use assertions effectively

Data Loading and Processing

The function `get_dataloader` should transform image data into resized, Tensor image types and return a DataLoader that batches all the training data into an appropriate size.

Do remember that if you only pass one integer value to resize function, the output may not be (input,input) as dimension as pytorch's resize works differently.
https://pytorch.org/docs/stable/torchvision/transforms.html#torchvision.transforms.Resize

Pre-process the images by creating a `scale` function that scales images into a given pixel range. This function should be used later, in the training loop.

Do read the full paper if you get time it is filled with insights to train and understand the architecture.
https://arxiv.org/pdf/1511.06434.pdf

Build the Adversarial Networks

The Discriminator class is implemented correctly; it outputs one value that will determine whether an image is real or fake.

Do refer Soumith Chintala's GanHacks guidelines as well while training any GANs ->

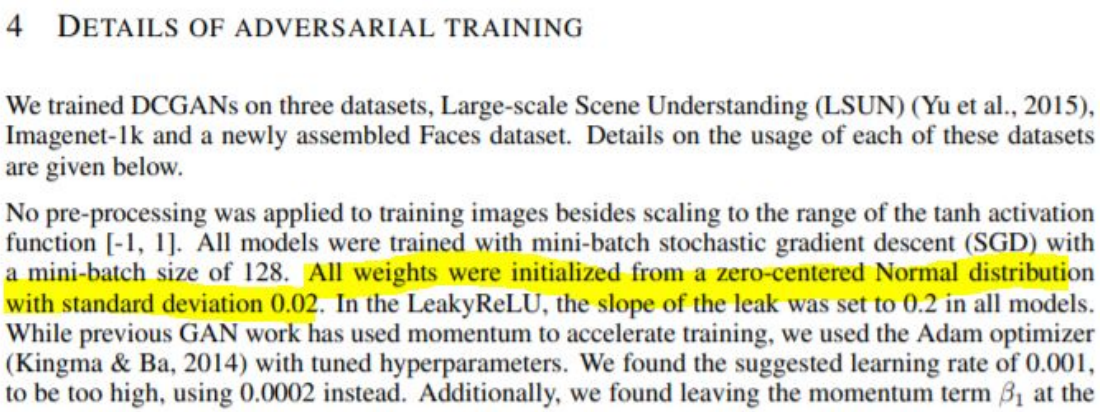
https://github.com/soumith/ganhacks

The Generator class is implemented correctly; it outputs an image of the same shape as the processed training data.

This function should initialize the weights of any convolutional or linear layer with weights taken from a normal distribution with a mean = 0 and standard deviation = 0.02.

- Xavier initialization is also good to use in general but this was just to align with the DCGAN paper.
- Do remember that by doing this we are initializing weights for all the conv and linear linears (including transpose conv (deconv) layers as well.

  Excerpt from the DCGAN paper regarding training



4 DETAILS OF ADVERSARIAL TRAINING

We trained DCGANs on three datasets, Large-scale Scene Understanding (LSUN) (Yu et al., 2015), Imagenet-1k and a newly assembled Faces dataset. Details on the usage of each of these datasets are given below.

No pre-processing was applied to training images besides scaling to the range of the tanh activation function [-1, 1]. All models were trained with mini-batch stochastic gradient descent (SGD) with a mini-batch size of 128. All weights were initialized from a zero-centered Normal distribution with standard deviation 0.02. In the LeakyReLU, the slope of the leak was set to 0.2 in all models. While previous GAN work has used momentum to accelerate training, we used the Adam optimizer (Kingma & Ba, 2014) with tuned hyperparameters. We found the suggested learning rate of 0.001, to be too high, using 0.0002 instead. Additionally, we found leaving the momentum term β_1 at the

Optimization Strategy

The loss functions take in the outputs from a discriminator and return the real or fake loss.

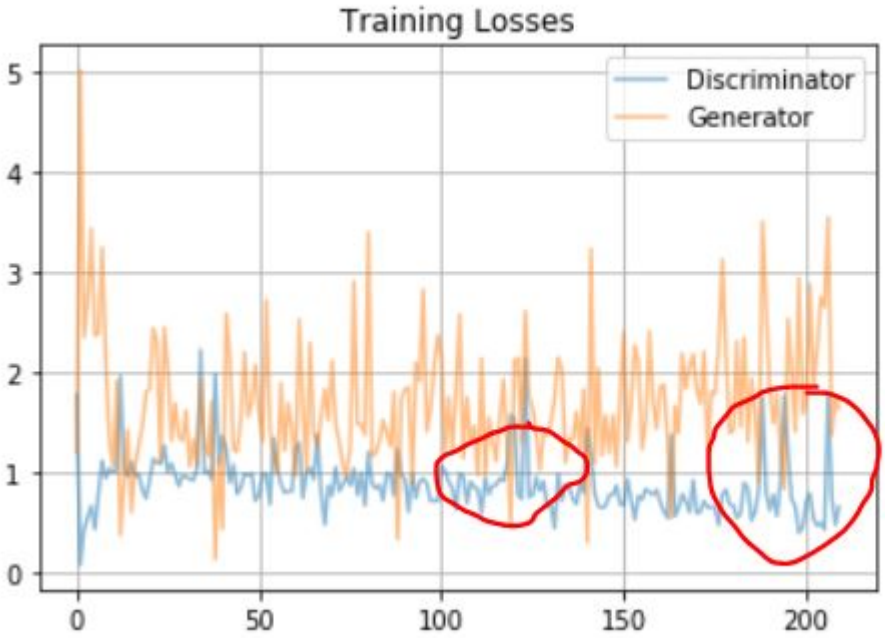
There are optimizers for updating the weights of the discriminator and generator. These optimizers should have appropriate hyperparameters.

In general, while training DCGAN on different datasets, do follow this article for understanding what can be stable training params ->
https://machinelearningmastery.com/how-to-train-stable-generative-adversarial-networks/

Training and Results

Real training images should be scaled appropriately. The training loop should alternate between training the discriminator and generator networks.

```
Epoch [ 14/ 14] | d_loss: 0.6460 | g_loss: 1.6290
```



- You have trained for a decent amount of epochs (14) which is decent in hindsight as the convergence is present in early stages and as and when you increase the epochs you see the generator loss going haywire which doesn't mean that the faces will not be generated in a proper way but you can definitely train for more epochs and use things like exponential learning decay and more dense networks for improving generation.
- As I said, you can also apply learning rate decay here if you are training for more epochs (50+) in general. https://discuss.pytorch.org/t/how-to-do-exponential-learning-rate-decay-in-pytorch/63146

There is not an exact answer here, but the models should be deep enough to recognize facial features and the optimizers should have parameters that help wth model convergence.

The project generates realistic faces. It should be obvious that generated sample images look like faces.

- Some of them are still a bit shaky/blurry, you can definitely train more, apply exponential learning rate decay and see if any other parameters need fine-tuning to the loss not exploding.
- We have only trained on smaller sized images so this is bound to happen as quality input == quality output in any ML algorithm.
  https://paperswithcode.com/task/face-generation

The question about model improvement is answered.

Some more explanations regarding the data -

- Thinking about the variety of faces and ethnicities should be done rightly as there needs to balance in terms of gender, race and not induce human bias as this faces dataset is limited to celebrity faces is a good step in looking at ethics and fairness in AI systems. This what-if tool is pretty decent to test for 5 different kinds of fairness.
- Also discussing the quality of the image in terms of size of the image should be done more as 32x32 is quite small and architectural restrictions happen because of it, given that if you increase the size and also have deep NNs, the training time would jump up significantly if you don't have the right GPU or memory so if you have good hardware or cloud GPUs you can definitely take a bet on training on high-resolution images.
- There is this comparative paper as well which you can read - https://arxiv.org/pdf/1801.04406.pdf

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START