UDACITY

‹ Return to "Deep Learning" in the classroom

DISCUSS ON STUDENT HUB

Generate Faces

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations! 🎉 You've passed this project. Fantastic Work here!! This is a great submission. Your concepts of DCGAN are crystal clear. I've suggested a few more tips.

Also, keep studying about the topic as this is just the beginning. I have also given some more **tips** to further improve your project.

Moreover, here're a few resources to help you continue this wonderful journey:

1. Wasserstein GAN as it's GAN with an objective and thus has a convergence criterion.
2. Generative Models in general
3. DiscoGAN, Discover Cross-Domain Relations with Generative Adversarial Networks
4. GAN stability
5. MNIST GAN with Keras which shows that how much concise can one be when modelling neural networks.
6. How to Train a GAN: https://github.com/soumith/ganhacks
7. Making pixel art with GANs.
8. Also go through current standard technique for making high resolution images: Progressive Growing of GANs for Improved Quality, Stability, and Variation
9. BigGANs: Large-Scale GAN Training is one of very influential papers of last year in generative modelling. TLDR: Twitter thread by the author.
10. Another interesting read is the Style-Based GAN. You can also see its results on the website thispersondoesnotexist.com.
11. A blog going through variety of GANs and comparing them

I really hope you enjoyed studying Deep Learning, the hottest topic in AI right now, here with Udacity 😄

Until next time! Have an amazing time working with neural nets.

Required Files and Tests

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

All the unit tests in project have passed.

Well done, all units passed! 😄

However, unit testing does not assure perfect code or results. There could still be some unresolved issues. The purpose of unit testing is to help avoid silly mistakes and improve code quality.

Moreover, unit testing(tutorial) is very good practice to ensure that our code is free from bugs & prevents us from wasting a lot of time while debugging minor problems as well as improves our code standards.

I recommend you to continue using unit testing in every module that you write in future, to keep it clean and speed up your development. Unit testing is highly motivated in industries.

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.

- The batch size used is a bit too large. Try values like 16 to 32 for better results because
    - If you choose a batch size too small then the gradients will become more unstable and would need to reduce the learning rate. So batch size and learning rate are linked.
    - Also if one use a batch size too big then the gradients will become less noisy but it will take longer to converge.
- Also consider using SpectralNorm or InstanceNorm or GroupNorm as they are independent of the batch-size and work well in improving image generation results.

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.

Build the Adversarial Networks

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

Nice job of implementing the `discriminator` as a sequence of `conv` layers with the following points to note:

Pros

- implemented `discriminator` using `conv2d` + `strides` to avoid making sparse gradients instead of max-pooling layers.
- used `leaky_relu` instead of ReLU for the same reason of avoiding sparse gradients as `leaky_relu` allows gradients to flow backwards unimpeded.
- used `sigmoid` as output layer
- implemented BatchNorm to avoid "internal covariate shift".

Tips

- Implement dropouts with low `drop_prob` in `discriminator` as mentioned here.

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

- Similar to implementing dropouts in D, as mentioned in section above, use dropout after each deconv layer except the last.

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.

Optimization Strategy

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

Nice job implementing here the loss function with **one-sided label** smoothing for GANs.

- For more Tips: refer GAN Hacks

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

- The beta1 is a bit too high. Values like 0.1 to 0.3 have shown to get best results.

- The current learning rate is okay but can be a bit high. The DCGAN with this architectural structure remains stable with lr between `0.0002` and `0.001`.

Training and Results

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

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.

The question about model improvement is answered.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review