UDACITY

PROJECT

Generate Faces

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Requires Changes

2 SPECIFICATIONS REQUIRE CHANGES

Fantastic Work here!! This is a great first submission. You have a vivid grasp of DCGAN concepts. I've suggested a few tips which will help you generate very realistic faces and you'll pass this project as well as this Nanodegree.

Looking forward to your next submission 😄

Required Files and Tests

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

All the unit tests in project have passed.

Build the Neural Network

The function model_inputs is implemented correctly.

The function discriminator is implemented correctly.

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

Awesome

- 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

- Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers.
- Implement dropouts with low `drop_rate` in `discriminator` as mentioned here.

The function generator is implemented correctly.

Simply follow the same tips of using dropouts and weight initialisation as that for the `discriminator`.

The function model_loss is implemented correctly.

Nice job implementing here the loss function for GANs.

Tip

- To prevent `discriminator` from being too strong, only the discriminator labels (one-sided) are reduced from 1 to 0.9. This is known as **label smoothing**.
  One can do this as follows: `labels = tf.ones_like(tensor) * (1 - smooth)`
- More Tips: refer GAN Hacks

The function model_opt is implemented correctly.

Neural Network Training

The function train is implemented correctly.

- **It should build the model using `model_inputs`, `model_loss`, and `model_opt`.**
- **It should show output of the `generator` using the `show_generator_output` function**

Great work combining all the parts (functions) together and making it a **Deep Convolution Generative Adversarial Network**.

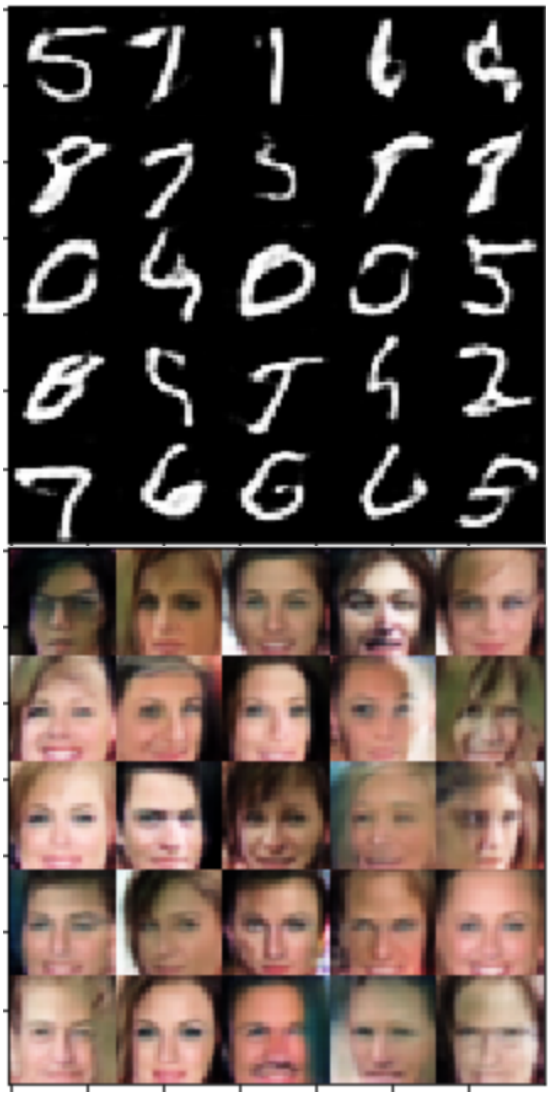
The parameters are set reasonable numbers.

Your choice of hyper-parameters is good. However, I suggest you to go through the Radford model in the original DCGAN paper and then choose your hyper-parameters. Anyway, I am putting forward a few of my suggestions:

- **Batch Size:** Try reducing batch_size to (~16 to 32) 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.
- **Learning Rate**: The current rate is a bit high. The DCGAN with this architectural structure remains stable with lr between `0.0001` and `0.0008`.
- An important point to note is, batch size and learning rate are linked. If the batch size is too small then the gradients will become more unstable and would need to reduce the learning rate.

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

Using a similar network (3 conv blocks in D and 3 deconv blocks in G) with 2 and 1 epochs respectively, I was able to get the following results:



After making the specified changes and considering the suggestions, you will be able to come up with equally good or better results.

Also, for better visualisation purposes, simply use the `n_images` value of around 25 (5x5) in `show_generator_output`.

RESUBMIT

DOWNLOAD PROJECT