UDACITY

PROJECT

Generate Faces

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Fantastic Work here!! This is a great submission I've seen. 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:

- How to Train a GAN: https://github.com/soumith/ganhacks
- Stability of GANs: http://www.araya.org/archives/1183
- MNIST GAN with Keras: https://medium.com/towards-data-science/gan-by-example-using-keras-on-tensorflow-backend-1a6d515a60d0
- https://blog.openai.com/generative-models/
- https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7

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.

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".
- Used 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.

Tips

- 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.

- For 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.

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

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

Student FAQ