UDACITY

PROJECT

Generate Faces

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

🏆Terrific job with the project! It was really a great experience reviewing your project. I'm impressed with how you leveraged all the main concepts of Generative adversarial networks in your submission.

Congratulations on completing the last project of the Deep Learning Nanodegree! 🎓

Here are some tips to improve your project or learn more.
I Highly recommended number one and two:

1. NVIDIA's high-resolution GAN aka Progressive Growing GAN
2. Fantastic GANs and where to find them
3. Delving deep into GANs
4. GAN Hacks
5. GAN stability

Required Files and Tests

The project submission contains the project notebook, called “dlnd_face_generation.ipynb”.

Perfect all necessary files are here!

All the unit tests in project have passed.

Well done, all units passed! 🎉

Remember that the unit tests do not assure perfect code or results. There could still be some unresolved issues but we (reviewers) will give you feedback on the issues we catch.

Build the Neural Network

The function model_inputs is implemented correctly.

Nice work!

Placeholders are the building blocks in computational graphs of any neural network. To read more about Tensorflow placeholders you can take a look at these links: link1 link2

The function discriminator is implemented correctly.

Amazing work here!

A detailed checklist is down below:

1. Use a sequence of `conv2d` layers with strides ✔
2. `batch_normalization` to avoid "internal covariate shift" ✔
3. LeakyRelu ✔
4. Use `sigmoid` as the output layer ✔
5. Good work on skipping `tf.layers.batch_normalization` on the first layer ✔
6. Use customer kernel ✔

The function generator is implemented correctly.

Superb work!

- The checklist is the same for the discriminator (above) except that you have to use `tanh` as the output layer and skipping `tf.layers.batch_normalization` on the first layer. ✔
- Using Tanh as the output layer means that we'll have to normalize the input images to be between -1 and 1. ✔

The function model_loss is implemented correctly.

Brilliant Work on using (one-sided) label smoothing!! 👍

The function model_opt is implemented correctly.

Fantastic Job!

You made the necessary changes when using BatchNorm. 👏

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

Excellent!! You have combined all the parts together in the function to train a GAN!

Perfect, you set the `batch_z` and `batch_images` range from -1 to 1 and to fit the range of the Tanh function output (from the generator) and used `np.random.uniform()` for `batch_z`!!

The parameters are set reasonable numbers.

You used the appropriate parameters for the type of model you implemented!!

Other suggested values:

Learning Rate: Appropriate values are **0.0002 - 0.0008**
Beta1: Appropriate values are **0.4 - 0.5**
Alpha/leak parameter: Appropriate values are **0.1 - 0.2**
Z-dim: Appropriate values are **100 - 128**
Batch Size: Appropriate batch sizes are **32 - 64**

Learning rate and Beta1(The exponential decay rate for the 1st-moment estimates) are inverses of each other. When increasing the learning rate, Beta1 should also decrease, and vice versa for stability purposes. You can learn more about momentum in gradient descent algorithms in this link and Adam in this link

Always set parameters in powers of 2 like 4,8,16,32,64.....

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

Fantastic work overall! The project generates realistic faces in just one epoch! You should give yourself a pat on the back. 👍

DOWNLOAD PROJECT