UDACITY

PROJECT

Generate Faces

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Brilliant Learner,
By carefully going through the project, it shows a lot of effort, diligence and above all, understanding of the GANs. Well done. You have successfully passed all the rubrics of this project excellently on the second submission and I congratulate you for this. There is still some improvement that can be made to improve model performance and generate clearer faces and I have pointed out some suggestions in this review. Please, feel free to take some time out and improve on the model. It was my pleasure reviewing this wonderfully implemented project and I wish you a very happy new year 2018 🎅.

Notes.

Here are some links to additional resources which I think will be very helpful in further understanding GANs.

- How to Train a GAN? Tips and tricks to make GANs work
- Stability of Generative Adversarial Networks
- Delving deep into GANs
- DiscoGAN in PyTorch

Required Files and Tests

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

The submission included the `dlnd_face_generation.ipynb` file along with the downloaded HTML version, the `helper.py` and the `problem_unittests.py` files.

All the unit tests in project have passed.

Awesome work! The submission successfully passes all the unit tests including the unit tests for:

- `model_inputs()` function showing proper setup of all placeholders.
- `discriminator()` function.
- `generator()` function indicating that the output was compatible with the discriminator input.
- `model_loss()` function.
- `model_opt()` function to show proper implementation of the optimization functions for both generator and discriminator.

Build the Neural Network

The function model_inputs is implemented correctly.

The `model_inputs` have been implemented correctly using `tf.placeholder.` Well done.

The function discriminator is implemented correctly.

The `discriminator()` function was properly built with three convolutional layers and one connected layer yielding an output of the logits and the sigmoid activated values. Great job using batch normalization and leaky Relu activation in the discriminator.

Advice.

- Weight initializers like xavier_initializer could be used to initialize weights as shown below:

```
conv2 = tf.layers.conv2d(relu1, 256, 5, strides=2, padding='same',
                         kernel_initializer=tf.contrib.layers.xavier_initializer())
```

- Adding fully connected(hidden) layers after flattening the conv3 output along with leaky RELU activations could also help improve the performance of the model.

The function generator is implemented correctly.

The submission included a proper implementation of the generator with the output being a tanh activation.

Advice.

- This GitHub thread on "How to Train a GAN? Tips and tricks to make GANs work" has some useful tricks on GAN training.
- Also consider improving the model by adding fully connected.
- Dropout layers are encouraged in the discriminator according to this post to introduce noise. This GitHub example has treated how to add a dropout layer to a model.

The function model_loss is implemented correctly.

Great work implementing the model_loss() function which returns a tuple of the discriminator and generator losses respectively. However, it is very neccessary to apply label smoothing to the discriminator real labels to improve the performance of the model.

Advice.

Applying label smoothing to discriminator loss could be very helpful in model performance.

```
smooth = 0.1#could be varied
 d_real_labels = tf.ones_like(output_real) * (1 - smooth)
   ------
  d_loss_real = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(
        logits=logits_real, labels=d_real_labels))
```

The function model_opt is implemented correctly.

Great work implementing the `model_opt` function to return a tuple of the discriminator training operation and generator training operation. Excellent work using tf.trainable_variables() to get the trainable variables in model and using the variable names to separate discriminator variables from generator variables.

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 work implementing the train function with the `model_inputs , model_loss and model_opt` functions implemented earlier in the notebook. Great work using the `show_generator_output` function to show the generator output after every 10 count of training.

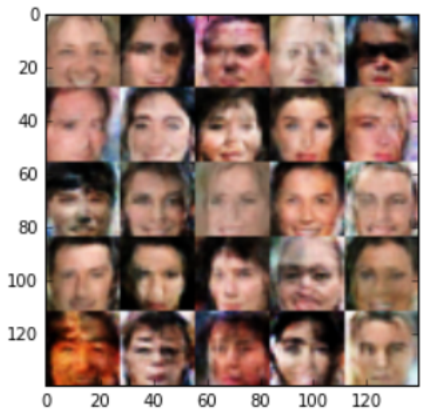
The parameters are set reasonable numbers.

The parameters used to train the model are set to reasonable values.

```
batch_size = 32
z_dim = 100
learning_rate = 0.0001
beta1 = 0.1
```

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

The faces generated are great and look like those of humans, although there is a lot of room for improvement. I have given some advice in this review and I encourage you to take the time and implement these in order to improve model performance.



DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

Student FAQ