UDACITY

‹ Back to Deep Learning Nanodegree

Generate Faces

REVIEW

CODE REVIEW

HISTORY

Requires Changes

2 SPECIFICATIONS REQUIRE CHANGES

Well Done !!! This is quite a good submission. Regarding your question about the generator and discriminator loss being similar, I would say that you should aim for lower generator loss than the discriminator loss which is what you are getting overall. However, as you scale the input images, you should be able to close some of that gap in the losses. However, the gap will still exist.

In order to gain more intuition about GANs in general, I would suggest you take a look at this blog post. Also, if you want to gain intuition about the convolution and transpose convolution arithmetic, I would suggest referring to this paper. For more advanced techniques on training GANs, you can refer to this paper.

All the best for your next submission. Keep learning.

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.

`model_inputs` has correctly set up the required placeholder tensors required for the model. Well Done !!!

The function discriminator is implemented correctly.

Well Done !!! The function discriminator is implemented correctly. Below are the good points of the architecture chosen:

- You have used Leaky ReLU as the activation function for the convolution layers which helps with the gradient flow. It helps alleviate the problem of sparse gradients.
- You are using same size filters across all the layers.
- You have used batch normalization which stabilizes GAN training.
- You have used Sigmoid as the activation function for the output layer which produces probability-like values between 0 and 1.

However, I would recommend using Xavier Initialization for the weights of the conv layers.

The function generator is implemented correctly.

Well Done Implementing the generator. Also, it was good to see that you have used a generator which is sufficiently larger than the discriminator in terms of depth. Using a generator bigger than the discriminator tends to produce better sample. So, Kudos to you on being thoughtful about the architecture of the generator. However, I would recommend you use Xavier intialization for the weights of the conv transpose layers.

The function model_loss is implemented correctly.

Good Job !!! However, you should have used one-sided label smoothing for the discriminator real loss as discussed in this paper.

The function model_opt is implemented correctly.

Normally, we scale the inputs to a neural network between 0 and 1. However, as the data moves through multiple layers, it starts shifting from that distribution and the deeper layers start getting data in a different range as input. This is known as internal covariate shift. To combat this, a technique known as Batch normalization was introduced, where the inputs to the layer are scaled between 0 and 1. However, batch norm has different behavior during training and testing phases.

Training

- Normalize the activations according to mini-batch statistics.
- Update the population statistics approximation based on the mini-batch statistics.

Testing

- Normalize the activations based on the approximated population statistics.
- **Do not** update the population statistics based on the test mini-batches.

You can think of the above behavior being very similar to the standard scaling concept in normal machine learning. In ml, we scale the training data based on the mean and standard deviation of the training set and use the same mean and standard deviation for the test set normalization. Since in most cases, we perform this preprocessing beforehand on all of training and testing data, we do not need to approximate the mean and variance.

This behavior of updating (and not updating during testing) the population statistics, specifically the moving mean and variance, is controlled by the parameter `training` in the batch norm call. The update ops to calculate and update these statistics are placed in `tf.GraphKeys.UPDATE_OPS`. So, they need to be added as a dependency to the train ops. Since, both generator and discriminator use batch norm and we are already controlling the train and test time behavior using the parameter `training`, we perform this update for both the generator and discriminator.

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**

Well Done !!! The model has been set up correctly. However, there are a few changes that you need to perform.

- You have defined `image_channels` to be equal to `out_channel_dim`. However, you are always setting `out_channel_dim` to be equal to 1 which is incorrect. The `out_channel_dim` should be the same as the images' channel. So, you can directly use data's last dimension as the number of channel output. Please correct this to be dependent on the dataset and not always equal to 1.
- The output from the generator is bound between -1 and 1 as we use `tanh` as the activation function in the output layer. The input batch images are in the range -0.5 to 0.5. You must scale the input images to match the values as in generated image. So, please use `batch_images = batch_images * 2` before passing it to the optimizers.

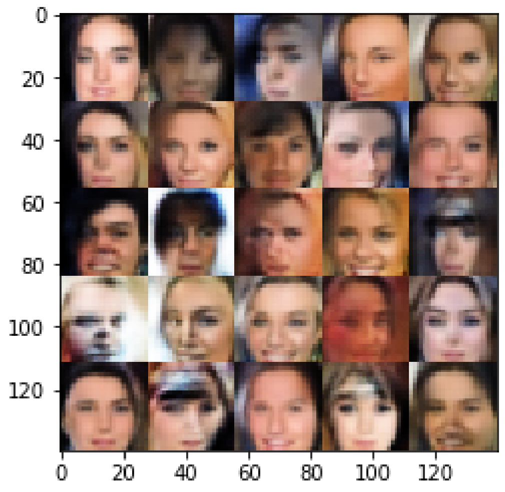
TIP: Execute the optimization for generator twice. This ensures that the discriminator loss does not go to 0 and impede learning.

The parameters are set reasonable numbers.

The parameters are quite good. Well Done !!!

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

Although the images generated do look like faces, they are a little faded out. This is due to the fact that you have not scale your input images to the same range as the output ones. So, correcting that should be able to give you better results. It is possible to achieve images like below within a single epoch:



RESUBMIT

DOWNLOAD PROJECT

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH

Rate this review

Student FAQ