UDACITY

‹ Return to "Deep Learning" in the classroom

Generate Faces

REVIEW

CODE REVIEW

HISTORY

Requires Changes

1 SPECIFICATION REQUIRES CHANGES

Fantastic Work here!! This is a great submission. I've suggested a few tips, which will help you generate very realistic faces and numbers, **specially the tip that I have provided in generator section.**

You are almost there, experiment with the layers and parameters. I am sure you'll get proper results in no time and create faces of upcoming celebrities :D

All the very best for your future and happy learning!!!

Required Files and Tests

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

The iPython notebook and helper files are included.

All the unit tests in project have passed.

Great work! All the unit tests are passed without any errors. But you need to keep in mind that, unit tests cannot catch every issue in the code. So, your code could have bugs even though all the unit tests pass.

Build the Neural Network

The function model_inputs is implemented correctly.

Correct, you have defined the placeholder tensors, which are the building block in computation graph of any neural net in tensorflow.

The function discriminator is implemented correctly.

Correct implementation of Discriminator, good work!

The function generator is implemented correctly.

Good Job implementing the generator! You have used Tanh as the last layer of the generator output, so you will normalize the input images to be between -1 and 1 in train function.

Make your generator deep to get high quality images, see my first tip below.*

You have met the basic requirements, but I recommend you to work on the below tips and comment on the improvements you see in the generated image.
1) Experiment with more conv2d_transpose layers in generator block so that there're enough parameters in the network to learn the concepts of the input images. DCGAN models produce better results when generator is bigger than discriminator. **Suggestion: 1024->512->256->128->out_channel_dim** (Use stride as 1 to increase the number of layers without changing the size of the output image).

2) Experiment with various values of alpha (slope of the leaky Relu as stated in DCGAN paper) between 0.06 and 0.18 and compare your results.

3) Experiment dropout in generator, so that it is less prone to learning the data distribution and avoid generating images that look like noise. (CONV/FC -> BatchNorm -> ReLu(or other activation) -> Dropout -> CONV/FC)

The function model_loss is implemented correctly.

Correct!

Tip: Experiment with label smoothing for discriminator loss, it prevents discriminator from being too strong and to generalize in a better way. Refer https://arxiv.org/abs/1606.03498
Below is a starter code,

```
d_loss_real = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=d_logits_real, labels=tf.ones_like(d_model_real) * (1 - smooth)))
```

The function model_opt is implemented correctly.

Correct.
To avoid internal covariant shift during training, you use batch norm. But in tensorflow when is_train is true and you have used batch norm, mean and variance needs to be updated before optimization. So, you add control dependency on the update ops before optimizing the network. More Info here http://ruishu.io/2016/12/27/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**

Great work combining all the functions together and making it a DCGAN.

Good job scaling the input images to the same scale as the generated ones using batch_images = batch_images*2.

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

Extra:
1) Talk on "How to train a GAN" by one of the author of original DCGAN paper here..

2) Here is a post on Gan hacks, https://github.com/soumith/ganhacks

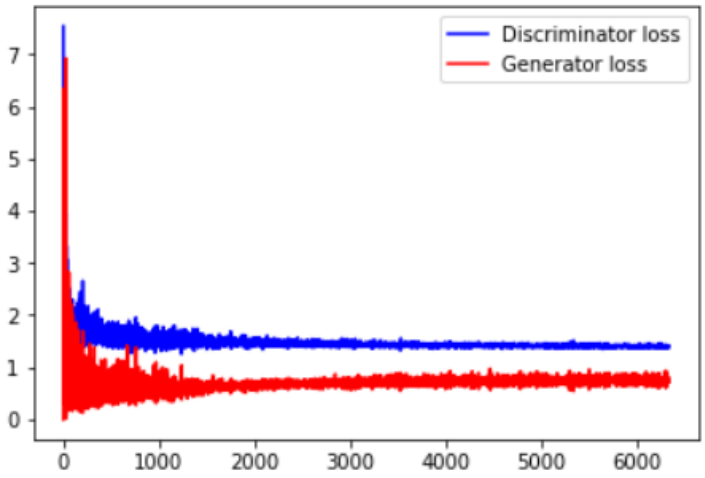
3) Plot discriminator and generator loss for better understanding. You can utilize the below code snippet to plot the loss graph to get a better understanding.

```
d,_ = sess.run(…)
g,_ = sess.run(…)
d_loss_vec.append(d)
g_loss_vec.append(g)
```

At the end, you can include the below code to plot the final array:

```
Discriminator_loss, = plt.plot(d_loss_vec, color='b', label='Discriminator loss')
Genereator_loss, = plt.plot(g_loss_vec, color='r', label='Generator loss')
plt.legend(handles=[ Discriminator_loss, Genereator_loss])
```

You'll be getting a graph similar to the below image,



The parameters are set reasonable numbers.

The hyperparameters chosen are correct. You can further improve the quality of the generated image by experimenting with the parameters and the tips I provided in discriminator and generator. Below are a few extra tips on choosing the hyperparameters for starters...

Tips:

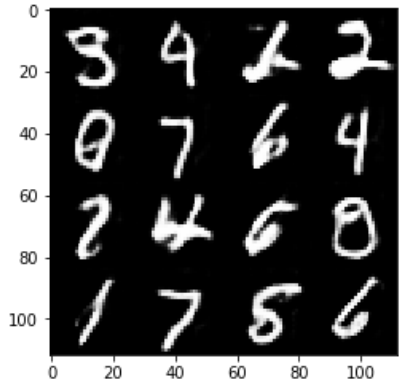
1) Try using different values of learning rate between 0.0002 and 0.0008, this DCGAN architectural structure remains stable within that range.

2) Experiment with different values of beta1 between 0.2 and 0.5 and compare your results. Here's a good post explaining the importance of beta values and which value might be empirically better.

3) 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 and vice versa. Start point for experimenting on batch size would be somewhere between 16 to 32.

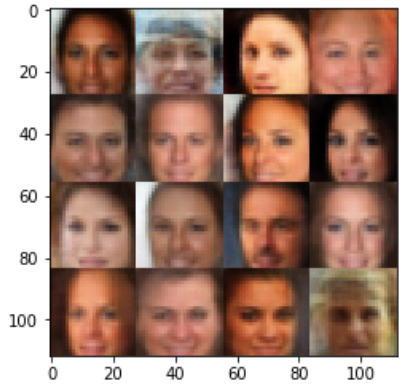
Extra: You can also go through Population based training of neural networks, it is a new method for training neural networks which allows an experimenter to quickly choose the best set of hyperparameters and model for the task.

Below is an output that you can get by modifying based on the tips. Keep experimenting, that's the only way to learn Deep Learning.



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

Your model generates good face images. You can still improve your model to generate realistic faces by following the same tips I provided for you in the above MNIST section.
Tip: If you want to generate varied face shapes, experiment with the value of z_dim (probably in the range 128 - 256).
Below is a reference output generated images that you can get by making changes to your model based on the tips that I have provided...



RESUBMIT

DOWNLOAD PROJECT