Generate Faces

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Well Done!!! You have met all the specifications, but don't stop here, keep experimenting. Experimenting is the only way you understand DL.

Go ahead and explore the wonderful world of GANs. Below are a few links for starters...
1) In order to gain more intuition about GANs in general, I would suggest you take a look at this link.

2) Also, if you want to gain intuition about the convolution and transpose convolution arithmetic, I would suggest referring to this [paper] (https://arxiv.org/abs/1603.07285).

3) For more advanced techniques on training GANs, you can refer to this paper.

4) One of the biggest problem GAN researchers face (you yourself must have experienced) with standard loss function is, the quality of generated images does not correlate with loss of either G or D. Since both the network are competing against each other, the losses fluctuate a lot. This problem was solved in early 2017 with introduction of Wasserstein GANs. With WGAN, the loss function directly correlates with how good your model is, and tracking decrease in loss a good idea. Do read it up.

5) Finally, have a look at this amazing library by Google for training and evaluating Generative Adversarial Networks.

6) Here are some other important resources for GAN:
http://www.araya.org/archives/1183 for GAN stability.
https://github.com/yihui-he/GAN-MNIST, https://github.com/carpedm20/DCGAN-tensorflow for DCGAN.
https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7

7) Below are few GAN videos:
https://www.youtube.com/watch?v=dqwx-F7Eits&list=PLkDaE6sCZn6FcbHlDzbVzf3TVgxzxK7lr&index=3
https://www.youtube.com/watch?v=RvgYvHyT15E
https://www.youtube.com/watch?v=HN9NRhm9waY
https://www.youtube.com/watch?v=yz6dNf7X7SA
https://www.youtube.com/watch?v=MgdAe-T8obE

8) Take a look at this Progressive Growing of GANs for Improved Quality, Stability, and Variation, which creates HD quality photos similar to the below image.

All the 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!
**Below are the good points of the architecture chosen:*
1) Leaky ReLU activation function helps with the gradient flow and alleviate the problem of sparse gradients (almost 0 gradients). Max pooling generates sparse gradients, which affects the stability of GAN training. That's the reason, you chose not to use pooling.

2) You have used batch normalization to stabilize GAN training by reducing internal covariant shift. You can go to this link for further understanding Batch norm.

3) You have used Sigmoid as the activation function for the output layer which produces probability-like values between 0 and 1.

4) Good job experimenting with dropout layers for discriminator, applying dropout will decrease hyper learning distrib. If discriminator end up dominating generator, we must reduce discriminator learning rate and increase dropout.

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) Use custom weight initialization. For example Xavier weight initialization to help converge faster by breaking symmetry or you can also use truncated_normal_initializer with stddev=0.02, which improve overall generated image quality, like in DCGAN paper.

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.

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.

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 different slope values for leaky_relu as told in discriminator.

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_image_output) * (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

✓

The parameters are set reasonable numbers.

The hyperparameters chosen are correct and your model generated realistic images, good job!

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

Tips:

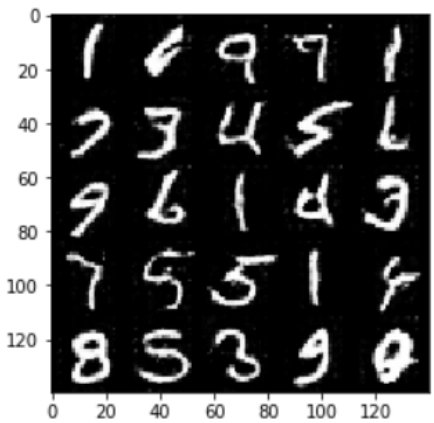
1) Try using different values of learning rate between 0.0002 and 0.0008, this DCGAN architectural structure remains stable within that range.

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 I got by modifying based on the tips. Experiment, that's the only way you learn Deep Learning.



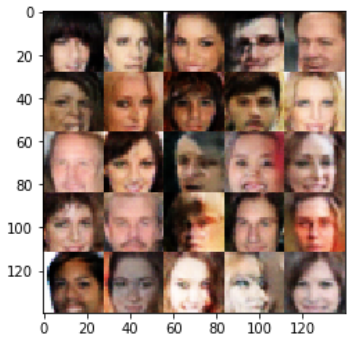
✓

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

Your model generates good face images and hyperparameters are correct. 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 one of the generated images that I got by making slight changes to your model.



DOWNLOAD PROJECT

RETURN TO PATH

Student FAQ