UDACITY

Return to "Deep Learning" in the classroom

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.
Take a look at this amazing paper that was presented in NeurIPS,
https://www.youtube.com/watch?v=kSLJriaOumA
https://arxiv.org/pdf/1812.04948.pdf
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.

Data Loading and Processing

The function `get_dataloader` should transform image data into resized, Tensor image types and return a DataLoader that batches all the training data into an appropriate size.

The function get_dataloader transforms image data into resized Tensor image types and return a DataLoader that batches all the training data into an appropriate size. Good job!
All images are also resized to image_size=32.

Pre-process the images by creating a `scale` function that scales images into a given pixel range. This function should be used later, in the training loop.

Good job scaling the input images to the same scale as the generated ones. Since tanh is used the last layer in the generator, its output lies between -1 and 1. So, the real image should also be normalized so that the input for the discriminator (be it from generator or the real image) lies within the same range.

Build the Adversarial Networks

The Discriminator class is implemented correctly; it outputs one value that will determine whether an image is real or fake.

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 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.

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 negative slopes (slope of the leaky Relu as stated in DCGAN paper) between 0.06 and 0.18 and compare your results.

3) Experiment 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. (CONV/FC -> BatchNorm -> ReLu(or other activation) -> Dropout -> CONV/FC)

The Generator class is implemented correctly; it outputs an image of the same shape as the processed training data.

Good Job implementing the generator!
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 conv_transpose2d 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**

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.

This function should initialize the weights of any convolutional or linear layer with weights taken from a normal distribution with a mean = 0 and standard deviation = 0.02.

This function initializes the weights of any convolutional or linear layer with weights taken from a normal distribution with a mean = 0 and standard deviation = 0.02.
Here's another way of implementing the same...

```
def weights_init_normal(m):
    classname = m.__class__.__name__
    # Apply initial weights to convolutional and linear layers
    if classname.find('Conv') != -1 or classname.find('Linear') != -1:
        # apply a centered, normal distribution to the weights
        m.weight.data.normal_(0, 0.02)
```

Optimization Strategy

The loss functions take in the outputs from a discriminator and return the real or fake loss.

Correct!
Utilizing label smoothing for discriminator loss prevents discriminator from being too strong and to generalize in a better way. Refer https://arxiv.org/abs/1606.03498

There are optimizers for updating the weights of the discriminator and generator. These optimizers should have appropriate hyperparameters.

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.

Training and Results

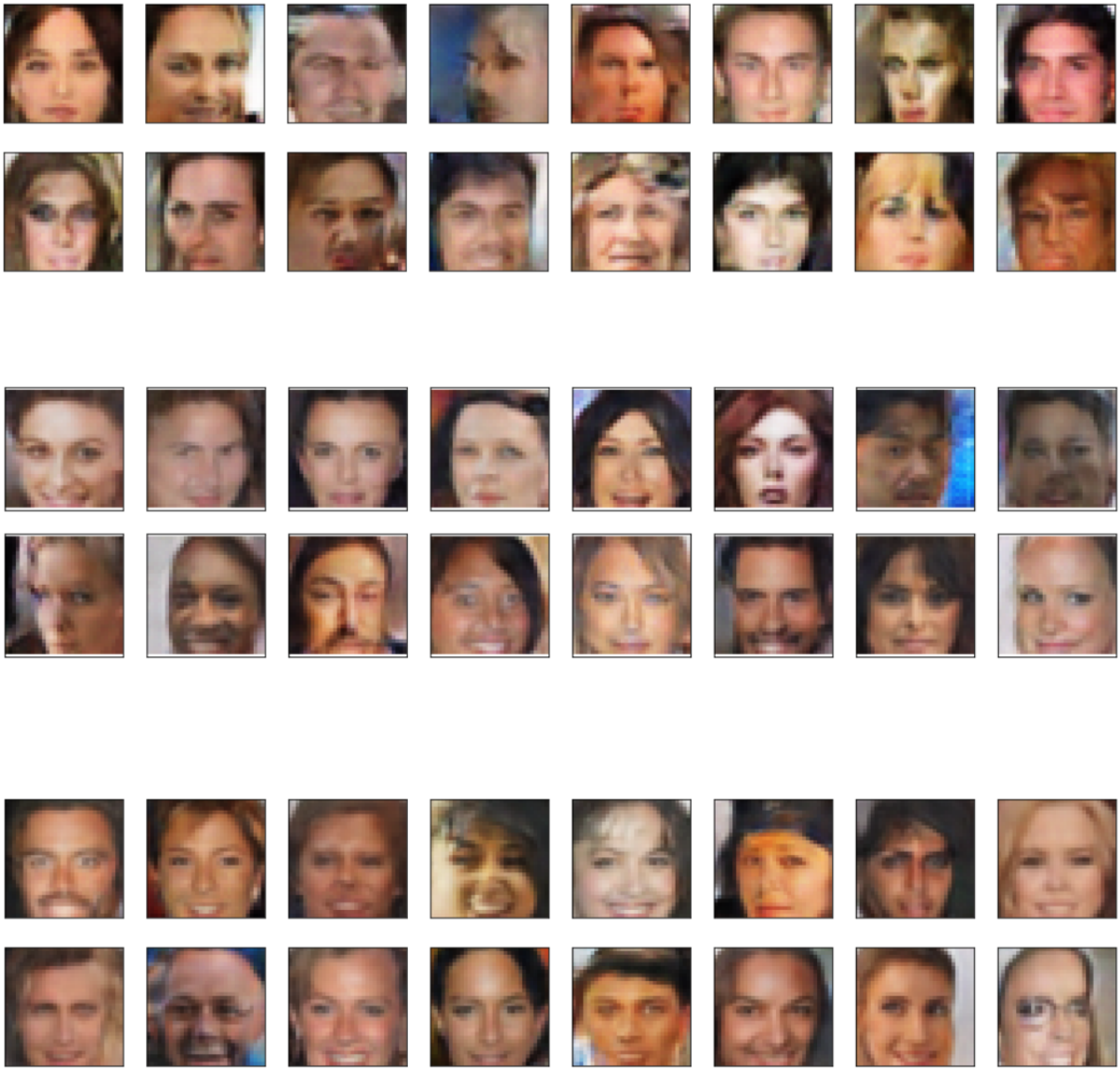
Real training images should be scaled appropriately. The training loop should alternate between training the discriminator and generator networks.

There is not an exact answer here, but the models should be deep enough to recognize facial features and the optimizers should have parameters that help wth model convergence.

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

Your model generates good face images. Good job!

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...



The question about model improvement is answered.

Your thought process is impeccable. All the answers you have provided are correct.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review