UDACITY

‹ Return to Classroom

Generate Faces

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Excellent work with the project!
Here are some resources if you wish to explore GANs more.

- This is a very good resource which covers an application of GANs for image completion and it properly explains the project as well. http://bamos.github.io/2016/08/09/deep-completion/
- Ian Goodfellow's Tutorial on GANs for NIPS 2016 https://arxiv.org/pdf/1701.00160.pdf along with this https://channel9.msdn.com/Events/Neural-Information-Processing-Systems-Conference/Neural-Information-Processing-Systems-Conference-NIPS-2016/Generative-Adversarial-Networks

Required Files and Tests

The project submission contains the project notebook, called “dlnd_face_generation.ipynb”.

All files are submitted!

All the unit tests in project have passed.

All tests passed!

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.

Good work. You correctly implemented the placeholders while ensuring the datatype was correct!

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.

Great implementation!

Build the Adversarial Networks

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

Excellent work on implementing multiple conv layers and appropriately applying the necessary activation functions and using batch normalization.

GANs are difficult to optimize. If the margin by which the discriminator wins is too big, then the generator can't learn well as the discriminator error would be too small. If the generator wins by too much, it won't learn well because the discriminator is too weak to teach it. There is a lot of research in GANs as you might know to solve all kinds of such problems. Some recommended ways would be -

- Use a smaller model for the discriminator relative to generator.
- Using dropout in discriminator so that it is less prone to learning the data distribution.
- I believe this has been referred previously, but do try to read on this - https://arxiv.org/pdf/1606.03498v1.pdf

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

Nicely done! Very impressive implementation.

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.

Great using weight initialization. Here are another research papers recommend Xavier initialization.
https://discuss.pytorch.org/t/how-to-initialize-the-conv-layers-with-xavier-weights-initialization/8419

Optimization Strategy

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

You correctly implemented the loss! Good work especially for using smoothing here!

Simple suggestion for you - Try apply label smoothing for generator loss?

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

Excellent work.

Think about running the optimizer for the generator twice. How do you think that will help?

Training and Results

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

Very well done!
Do you think different learning rates for generator and discriminator optimizers here would help? Try it out :)

- Here is a good resource on some tips and tricks on training GANs - https://github.com/soumith/ganhacks

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.

You selected a good set of hyperparameters.
Some suggestions to try out if you'd like -

- Try different batch sizes for each.
- Do you think different z_dim values are required based on what you are trying to generate? Do you think it should be higher for the faces?
- You selected a good value for `beta1` . Here's a good post explaining the importance of beta values and which value might be empirically better. Do check it out! http://sebastianruder.com/optimizing-gradient-descent/index.html#adam Try to tune your values based on this.

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

Nicely done!
You are getting some really good results.

The question about model improvement is answered.

Thanks for answering the question. I can tell you thoroughly test the GAN, and know the basics quite well now.

I provided some suggestions above, hope those ones help. As for the iterations, it depends on the result you are looking for. Usually more iterations make have better results if you don't have overfitting. Do try dropout to decrease the overfitting issue. When you printed out the loss, it showed you a shape that you can capture the optimal point that you can use an early stop to catch that. If you smooth your loss, you will see the optimal point is around 100 (batch * epoch). If you lower your batch size, you can have decrease epoches to do the training as well.
Hope that helps!

DOWNLOAD PROJECT

RETURN TO PATH