UDACITY

‹ Return to "Deep Learning" in the classroom

DISCUSS ON STUDENT HUB

Generate Faces

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Kudos ! I think you've done a perfect job of implementing a Deep Convolutional GAN to Generate Faces. It's very clear that you have a good understanding of the basics.

Further Reading: One of the biggest problem GAN researchers face (you yourself must have experienced) with standard loss function is, the quality of generated images do not correlate with loss of either G or D. Since both the network are competing against each other, the losses fluctuate *a lot*. This was solved in early 2017 with introduction of Wasserstein GANs. Do read it up.

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

And Congratulations 🎉 once again, as you've completed this project for the Deep Learning Nanodegree.

Required Files and Tests

The project submission contains the project notebook, called "dlnd_face_generation.ipynb".
The `iPythonNB` and helper files are included.

All the unit tests in project have passed.

Great work. Unit testing is one of the most reliable methods to ensure that your code is free from all bugs without getting confused with the interactions with all the other code. If you are interested, you can read up more and I hope that you will continue to use unit testing in every module that you write to keep it clean and speed up your development.

But always keep in mind, that unit tests cannot catch every issue in the code. So your code could have bugs even though 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.

Correct.
Dataloader is essential to PyTorch, for working with any mode of data.

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 work normalizing inputs using the scale function.
You might want to watch up this talk on How to train a GAN by one of the author of original DCGAN paper and corresponding write-up.

Build the Adversarial Networks

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

Overall you did a fine job implementing the `Discriminator` as a simple convolution network.

Let me illustrate the pros of the architecture you chose.

Pros

- You chose not to use pooling layers to decrease the spatial size. Max pooling generates sparse gradients, which affects the stability of GAN training. We generally use Average Pooling or Conv2d + stride.
- Used Batch normalization. We initialize the BatchNorm Parameters to transform the input to zero mean/unit variance distributions but as the training proceeds it can learn to transform to `x` mean and `y` variance, which might be better for the network. This post is an awesome read to understand BatchNorm to it's core.
- Correctly used Leaky ReLU. As explained above we never want sparse gradients (~ 0 gradients). Therefore, we use a leaky ReLU to allow gradients to flow backwards through the layer unimpeded.

Tips

- Using dropout in discriminator so that it is less prone to learning the data distribution.
- Try another custom weight initialization. Xavier init is also proposed to work when working with GANs.

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

Most of the suggestions are same for both Generator and Discriminator.

Let me (again) illustrate the pros of the architecture you chose.

Pros

- `Tanh` as the last layer of the generator output. This means that we'll have to normalize the input images to be between -1 and 1.

Tips

- Try decreasing the width of layers from 512 -> 64. In context of GANs, a sharp decline in number of filters for Generator helps produce better results.

Further Resource: Keras has a neat API for visualizing the architecture, which is very helpful while debugging your network. `pytorch-summary` is a similar project in PyTorch.

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.

Optimization Strategy

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

Perfect.
Now that was the trickiest part (and my personal favorite in GAN :)

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

Given your network architecture, the choice of hyper-parameter are reasonable.

Tips

- 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. Also try lowering it even further, ~0.3 might even produce better results.
- 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.

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.

Woah ! Now that is something 👍
This simple DCGAN model is generating faces too close to real human faces. Wonder what will happen when trained on even a bigger and diverse dataset?

Fun Read: Ever heard about the analogy "king – man + woman = queen"? Read this up on how to perform the same analogies on images, using simple arithmetic in latent space.

The question about model improvement is answered.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review