UDACITY

< Return to "Deep Learning" in the classroom

DISCUSS ON STUDENT HUB

Generate Faces

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Awesome submission, you have correctly implemented the DCGAN architecture and have used all the hyperparameters correctly. The results look great! Congratulations on passing this project and all the best 👍

I suggest you read following articles to know how to improve results in GAN:
https://github.com/soumith/ganhacks#how-to-train-a-gan-tips-and-tricks-to-make-gans-work
http://blog.otoro.net/2016/04/01/generating-large-images-from-latent-vectors/

Required Files and Tests

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

Files correctly submitted.

All the unit tests in project have passed.

All the tests have 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.

Dataloaders correctly used.

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.

Well done implementing a general scale function with min and max values.

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 job at

- Using a sequence of convolutional layers using conv2d with strides to avoid making sparse gradients instead of max-pooling layers as they make the model unstable.
- Using leaky_relu and avoiding ReLU for the same reason of avoiding sparse gradients as leaky_relu allows gradients to flow backwards unimpeded.
- Using sigmoid as output layer.
- BatchNorm to avoid "internal covariate shift" as batch normalisation minimises the effect of weights and parameters in successive forward and back pass on the initial data normalisation done to make the data comparable across features.

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

Same as the discriminator function: Excellent job at

- Implementing generator as "deconvolution" network with mostly the same key points as mentioned above.

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

to be between -1 and 1.

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.

Excellent job implementing the loss of the model!

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

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.

The generated faces look good!

The question about model improvement is answered.

DOWNLOAD PROJECT

RETURN TO PATH