UDACITY

‹ Return to Classroom

Face Generation

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations on completing the GAN project. As you might have experienced GANs are advanced and complex topic. In this project, a few more changes such as dropout layers in discriminator, bigger generator, etc., will get you better results. You have done an awesome job so far.

Here some links to know GAN's better:

- What are GANs (Generative Adversarial Networks)?
  https://www.youtube.com/watch?v=TpMIssRdhco
- The paper
  https://arxiv.org/pdf/1511.06434.pdf
- GAN - intro Ian Goodfellow
  https://www.youtube.com/watch?v=YpdP_0-IEOw&t=250s

Section 1: Code quality

Scripts have an intuitive, easy-to-follow structure with code separated into logical functions. Naming for variables and functions follows the PEP8 style guidelines.

Correct Implementation

- Overall, the code is structured and documented in a way that follows Pythonic best practices, making it easier to understand and maintain. The logical separation into functions and classes, along with descriptive naming, contributes to the clarity and readability of the script.

Recommendations

- Remove all the `# TODO: ...` commnents

Additional Resources

- How to Write Beautiful Python Code With PEP 8
  https://realpython.com/python-pep8/

Section 2: Generator and discriminator design

The generator should take a batched 1d latent vector as input and output a batch of RGB images (3 channels).

Generator

- Used a series of strided covolutional transpose layers.
- Use ReLU activation
- Used `tanh` at the output to return in the range -1 to 1.

Suggestion:

- Make the generator at least one more layer bigger than discriminator or reduce the size of the discriminator

The discriminator should take as input a batch of images and output a score for each image in the batch.

Discriminator

- Used a sequence of five convolutional layers. The size(3-4 layers) is appropriate
- Used strided convolution instead of max pooling to downsample, making the network to learn its own pooling function.
- Used batch normalization starting from second layer.
- Leaky relu and batch norm promote healthy gradient flow.

Suggestion:

- Add dropout layers right after each leaky ReLU. I see commented it out. In fact, dropouts in discriminator help it generalize it better.

See example here: https://github.com/udacity/deep-learning-v2-pytorch/blob/master/gan-mnist/MNIST_GAN_Solution.ipynb.

Generator and Discriminator are inheriting from the torch `Module` class. The layers are defined in the `init` method and called in the `forward` method.

- Both the Generator and Discriminator classes are correctly inheriting from the `torch.nn.Module` class.
- The code is well-commented, with explanations for the methods and their parameters.
- It adheres to PEP8 naming conventions and structuring.

Section 3: Data pipeline

(Provide specific details on how the student will fulfill the criteria.)
The `get_transform` function should output a `Compose` of different torchvision (or non torchvision) transforms.

- The function `get_transforms` as defined looks correct for setting up a series of image transformations using PyTorch's `torchvision.transform`. This will ensure that any image passed through this transformation pipeline will be resized, converted to a tensor, and normalized

The custom dataset should have the `__len__` and the `__get_item__` methods implemented and working. The dataset should return a tensor image in the -1 / 1 range.

Suggestions

- Error handling is always a good idea when loading images as files could be corrupted or paths could be incorrect.

Section 4: Loss implementation and training

Both loss functions are accomplishing their roles: the discriminator should be trained to separate fake and real images and the generator should be trained to fool the discriminator. No specific functions are required.

Generator_loss function

- Nice job using a condition to switch between WGAN loss (Wasserstein GAN) and a more traditional loss based on mean squared error (MSE).
- For the WGAN loss, it correctly uses the negative mean of the discriminator's output on the fake images.
- For the MSE loss, it creates labels of ones, which means the generator is trying to get the discriminator to output ones for fake images (interpreted as "real").

Discriminator loss

- Like the generator loss, it supports both WGAN and MSE-based losses.
- For the WGAN loss, it calculates the difference between the means of the real and fake logits, which is the correct approach.
- For the MSE loss, it sets up real and fake labels with an option to "smooth" the real labels. Label smoothing is a common technique to prevent the discriminator from becoming too confident.

Gradient Penalty

- Nice job implementing the gradient penalty. This regularization term is used to enforce the `Lipschitz` constraint on the critic (which is a term often used in the context of WGANs instead of 'discriminator').
- What is Lipschitz constraint and why it is enforced on discriminator? https://ai.stackexchange.com/questions/29904/what-is-lipschitz-constraint-and-why-it-is-enforced-on-discriminator

Two optimizers are created, one for the generator and one for the discriminator. They are both using low learning rates.

- Nice choice of Adam, best optimizer for this scenario. Learning rate in the range of 0.0002-0.0008 works well with a batch size of 16-32. The batch size and learning rate are linked. The smaller batch size should go with low learning rate.
- Check this article to know more about optimization in GAN. https://towardsdatascience.com/understanding-and-optimizing-gans-going-back-to-first-principles-e5df8835ae18

The `discriminator_step` and `generator_step` functions are correctly implemented. The model is training for enough epochs.

- Gradient penalty applied during the discriminator's training to enforce the Lipschitz constraint.

Suggestions

- Please store the losses during the training. You can use it for visualizing trend.
- Also include error handling and model saving checkpoints

The student makes reasonable decisions based on initial results.

The diversity of the input data is significant as well as the number of feature maps and model. The racial bias is a huge concern in the AI community, check out the news articles below:

https://www.forbes.com/sites/parmyolson/2018/02/26/artificial-intelligence-ai-bias-google/#6303f72b1a01

The generated samples should have face attributes (eyes, nose, mouth, hair) and a rough resemblance to a face.

- The generated images look like faces. Well done! In our case, with some tuning such as drop out layers in discriminator, bigger generator, and and more importantly label smoothing, you can improve the results.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START