PROJECT

Generate Faces

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW | CODE REVIEW | NOTES

Requires Changes

SHARE YOUR ACCOMPLISHMENT

2 SPECIFICATIONS REQUIRE CHANGES

Awesome submission ! 🎉 Your understanding and clarity of concepts of how GANs work is vividly depicted here. You've almost done the hard part of building the neural network, all it now needs is some subtle improvements and some more hyperparameter tuning. Complete the changes mentioned in this review and you would see the improvement in the results.

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/
Keep learning ! ✌🏻

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.**

Good job, all the unit tests passed.

Build the Neural Network

✓ **The function model_inputs is implemented correctly.**

You defined the placeholders correctly for the neural network, good job. 👍🏽

✓ **The function discriminator is implemented correctly.**

Excellent job at

- Considering variable scoping
- 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.
  Awesome. 👌🏽

Some suggestions for improvement:

- Use weight initialization: Xavier initialization is recommended as it helps model converge faster. A possible implementation in tensorflow is to pass `tf.contrib.layers.xavier_initializer()` as the value for the `kernel_initializer` parameter in `tf.layers.conv2d`
- Use Dropouts in discriminator so as to make it more robust. A possible implementation in tensorflow can be achieved by simply passing the outputs from the last layer into the `tf.nn.dropout` with a high `keep_probability`.

✓ **The function generator is implemented correctly.**

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.

Suggestions for improvement:

- It is recommended to have a larger generator network than the discriminator network.
- Similar to the discriminator, use Dropouts in the generator at both train and test time with `keep_probability` as 0.5 as suggested here.

✓ **The function model_loss is implemented correctly.**

Excellent job implementing the loss of the model with one-sided label smoothing. It's done to prevent discriminator from being too strong as well as to help it generalise better by reducing labels from 1 to 0.9.

✓ **The function model_opt is implemented correctly.**

Great job implementing the optimization operations!

Neural Network Training

✓ **The function train is implemented correctly.**

- **It should build the model using `model_inputs`, `model_loss`, and `model_opt`.**
- **It should show output of the `generator` using the `show_generator_output` function**

Great job! This function is complex, but you implemented it correctly.

Suggestions for improvement:

- It's recommended to run `g_opt` twice as explained here: http://bamos.github.io/2016/08/09/deep-completion/ and here: https://github.com/carpedm20/DCGAN-tensorflow

🔄 **The parameters are set reasonable numbers.**

For your network architecture, the choice of hyperparameters is mostly reasonable. But the batch size is a bit high. For your reference, a reasonable range of values of parameters in provided below. Try different combinations to generate more realistic faces.

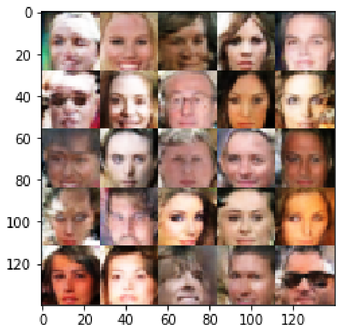
- Learning Rate: The DCGAN with this architectural structure remains stable with learning rate between 0.0001 and 0.0008.
- Beta1: 0.5 seems to be the best choice here as explained in this paper: https://arxiv.org/pdf/1511.06434.pdf
- Z-dim: 100/128 is a reasonably good choice here.
- Batch Size: Batch_size of (~16 to 64) works well because if you choose a batch size too small then the gradients will become more unstable and would need to reduce the learning rate. So batch size and learning rate are linked. Also if one use a batch size too big then the gradients will become less noisy

but it will take longer to converge.

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

As of now, I can see the faces but they are not quite clear and realistic. Please make sure to apply all the suggested changes, and your model would produce much better results. Also, train the model for generating handwritten digits. Also, show 16-25 generated samples so that you have a better idea of how your model is performing.

For your reference, some results from GAN with 3 conv layers in discriminator and 4 deconv layers in generator:



RESUBMIT PROJECT

DOWNLOAD PROJECT

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH

Student FAQ