UDACITY

PROJECT

Generate Faces

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Requires Changes

3 SPECIFICATIONS REQUIRE CHANGES

Great work on your project !! It was really a great experience reviewing your project. Just make the suggested changes, and you will be done. I am confident that you will do great in your next submission. 😃

Here are some tips to improve your project or learn more:

1. Delving deep into GANs
2. GAN Hacks
3. GAN stability
4. BEGAN and WGAN article
5. DiscoGAN implementation
6. WGAN walk-through
7. BEGAN implementation

Required Files and Tests

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

Perfect all necessary files are here!

All the unit tests in project have passed.

Well done, all units passed! 🎉

Remember that the unit tests do not assure perfect code or results. There could still be some unresolved issues but we (reviewers) will give you feedback on the issues we catch.

Build the Neural Network

The function model_inputs is implemented correctly.

Nice work!

Placeholders are the building blocks in computational graphs of any neural network. To read more about Tensorflow placeholders you can take a look at these links: link1 link2

The function discriminator is implemented correctly.

Amazing work here!

A detailed checklist is down below:

1. Use a sequence of `conv2d` layers with strides ✔
2. `batch_normalization` to avoid "internal covariate shift" ✔
3. LeakyRelu ✔
4. Use `sigmoid` as the output layer ✔
5. Good work on skipping `tf.layers.batch_normalization` on the first layer ✔

TIP:

- **(Highly recommended)** Use Xavier weight initialization to break symmetry, and in turn, help converge faster and prevent local minima. Here are two links to help you understand Xavier initialization: link1 link2. A possible implementation is to pass `tf.contrib.layers.xavier_initializer()` as the value for the `kernel_initializer` parameter in `tf.layers.conv2d`

The function generator is implemented correctly.

Superb work!

- The checklist is the same to the discriminator (above) except that you have to use `tanh` as the output layer and skipping `tf.layers.batch_normalization` on the first layer. ✔
- Using Tanh as the output layer means that we'll have to normalize the input images to be between -1 and 1. ✔
- The tip is also the same as the discriminator (above). Meaning, you can also implement Xavier weight initialization **(Highly recommended here too)**.

The function model_loss is implemented correctly.

Good job! You did a fantastic job here as the loss function for GANs can be very complicated and confusing. 👍

TIP: To prevent the `discriminator` from becoming too strong and to help it generalize better, the discriminator labels can be reduced from 1 to 0.9. This is called label smoothing (one-sided).
A possible TensorFlow implementation is:

```
smooth = 0.1
d_loss_real = tf.reduce_mean(
        tf.nn.sigmoid_cross_entropy_with_logits(logits=d_logits_real, labe
ls=tf.ones_like(d_model_real)*(1.0 - smooth)))
```

A possible TensorFlow implementation is `tf.ones_like(tensor) * (1.0 - smooth)` where smooth is a float value ~ 0.1. The resulting value will multiply the label by ~0.9 and reduce the label from 1 to ~0.9.

The function model_opt is implemented correctly.

Fantastic Job!

You made the necessary changes when using BatchNorm. 👏

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**

Excellent!! You have combined all the parts together in the function to train a GAN!

Perfect, you set the `batch_z` and `batch_images` range from -1 to 1 and to fit the range of the Tanh function output (from the generator) and used `np.random.uniform()` for `batch_z` !!

The parameters are set reasonable numbers.

Nice start! A breakthrough on the details of suggested hyperparameters is down below:

The aggressive hyperparameters are on the left side values, and safe hyperparameters are on the right side values.

Learning Rate: ✔ Appropriate values are **0.001/0.0002**
Beta1: ✔ Appropriate values are **0.3/0.5**
Alpha/ leak parameter: ✔ Appropriate values are **~0.2**
Z-dim: ✔ Appropriate values are **128/100**
Batch Size: ✔ Appropriate batch sizes are **64/32**

I recommend you start with the safe hyperparameters after making the required changes to see how the hyperparameters affect the image quality. Your model was able to get good results with safe but very slow hyperparameters, so you are half way there!

Learning rate and Beta1(The exponential decay rate for the 1st-moment estimates) are inverses of each other. When increasing the learning rate, Beta1 should also decrease, and vice versa for stability purposes. You can learn more about momentum in gradient descent algorithms in this link and Adam in this link

Always set parameters in powers of 2 like 4,8,16,32,64..... This will help Tensorflow to optimize the computation.

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

Great work overall, you are almost there!! Please make the required changes alongside the suggested tips to be able to generate realistic faces. Good luck!

RESUBMIT

DOWNLOAD PROJECT