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

This is really a good submission 👍 I enjoyed reviewing this submission.

Your submission shows that you have a good understanding of GANs. Improving performance of the model is bit tricky and I hope suggestions given in comments will improve your learning as well as the performance of the model.

Now, All you have to do is to tweak you hyperparameters with suggested changes and you are done.

Check this paper on loss-sensitive GAN

I would strongly recommend you to check this blog post to get some more intuition about GANs .

You are almost there !! All the best for your next submission. 👍

Required Files and Tests

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

The project submission contains the project notebook file. 📁 ✔

All the unit tests in project have passed.

Tested 🆗 All the unit tests in the project have passed. ✔

Build the Neural Network

The function model_inputs is implemented correctly.

You have perfectly implemented placeholders. ✔

Perfectly implemented placeholders are the backbone of any model as they are used as a handle for feeding values.

💡 Check following external links on placeholders:

> - Check this stackoverflow documentation for more details.
> - Check this tutorial on placeholder.
> - **tf.Variable** vs **tf.placeholder** explained here.

The function discriminator is implemented correctly.

I found following good points you have done here : Applause 👏

> - **Activation function:** The only way by which generator can learn is by receiving gradient from discriminator, hence gradient to flow through entire architecture, it is recommended to use **Leaky ReLU** as the activation function for the convolution layers.
> - **Filters:** Use of same size filters for all the layers.
> - **Normalization:** Use of batch normalization to stabilize GAN training with no use of batch normalization in the **first layer**.
> - **Sigmoid function:** Use of the Sigmoid function for the output layer to produce values between 0 and 1 (probability values).
> - Use of Xavier initialization to makes sure the weights are right across all layers.

💡 Tip:

> - Use **tf.layers.dropout** to prevent overfitting like `tf.layers.dropout(conv, keep_prob, training=is_train)`.
> - Use xavier_initializer instead of random_normal_initializer to makes sure the weights are right across all layers by performing "Xavier" initialization for weights..
>   Eg.-
>   `x1 = tf.layers.conv2d(images, 128, 5, strides=1, padding="SAME",kernel_initializer=tf.contrib.layers.xav`

The function generator is implemented correctly.

Good !! As like discriminator, you have:

> - Correctly used `tf.variable_scope` to reuse variables. ✔
> - It is recommended that there should not be any batch normalization on the last layer and *this is taken care off.* ✔
> - Using `tf.tanh` is a good choice for computing hyperbolic tangent of logits element-wise. ✔

💡 Suggestion

> - Use xavier_initializer instead of random_normal_initializer to makes sure the weights are right across all layers by performing "Xavier" initialization for weights.
> - Check batch normalization for image here

The function model_loss is implemented correctly.

Wow !! 👏

> Using **One-sided label smoothing** with smoothing factor of 0.1 is the best practice for improving performance. One-sided label smoothing was also used at DCGAN project developed by Facebook.

💡 Suggestion:

> - Check out this openAI paper for details on One-sided label smoothing.

The function model_opt is implemented correctly.

Good !!

> AdamOptimizer is recommended optimizer which was also used in DC GAN project, additionally, you have made sure that all updates are computed before running optimizer by use of `control_dependencies`.

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

This looks good !!

Nicely clubbed all the pieces together to form the model to generate realistic faces.

> - It has used all the required functions. ✔
> - Output is shown using `show_generator_output`. ✔
> - Nicely normalized inputs. ✔

💡 Suggestion:

> - While running the discriminator once each iteration, you can optimize generator twice, for every iteration, to make sure that the discriminator loss does not go to zero and this will help the generative model to optimize better.
> - I would suggest you go through this openAI paper for improving GAN training.

The parameters are set reasonable numbers.

Nice effort !!

Although hyperparameters look good. I would suggest tweaking hyperparameters to produce better results. Following are the values which work well in the similar network configuration.

- batch_size : 32 - 64.
- learning_rate : 0.0002 - 0.0003
- beta1 : 0.4 - 0.5
- z_dim : ~100

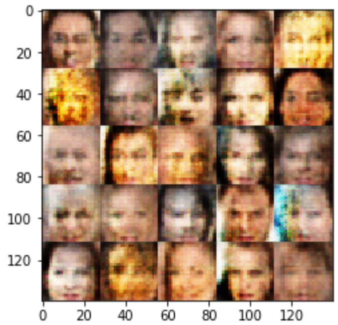
This should improve the performance of the network.

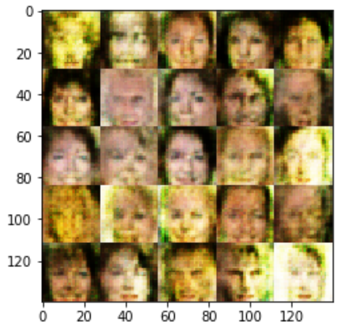
The project generates realistic faces. It should be obvious that images generated look like faces.

Required:

Nice effort !! The model generates faces, however, there are some details missing in that. The generated faces should look like realistic faces.

Output:

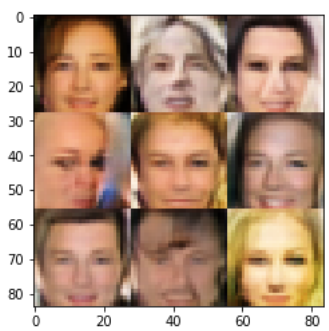


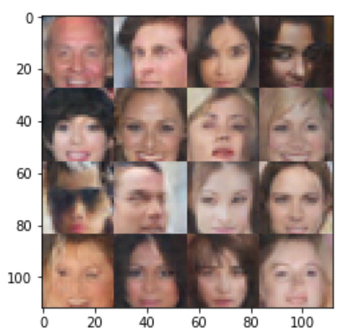


There is a small scope of improvement in the model, All you have to do is to tweak your hyperparameters with the suggested changes and you are done.

For your reference:

Here is the output of the model having **similar network configurations**, which can be considered as a **better model** for generating realistic faces.





RESUBMIT

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

Rate this review