UDACITY

PROJECT

Generate Faces

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Excellent work with the project! Yours is probably one of the best submissions I have come across till now.

As for your question on the two different batch normalization methods in TF. In all honesty, the `tf.contrib.layers.batch_norm` has some added functionality compared to the other, and I can't seem to find any major difference (even looking at their source code). Their implementations seem to stem from the same paper. The `contrib` indicates it's a contribution that's not part of the official api release per se and might be added into it later. The one main difference I notice is the `updates_collection` parameter. For the `tf.layers.batch_normalization` you would have to define that separately as you already know, but the `contrib` version, I think, gives the option of it being a passable argument to make it more concise. But it's an interesting question, maybe ask more on the forums and other students and mentors can contribute to a better discussion we all could benefit from :)

Here are some resources if you wish to explore GANs more.

- This is a very good resource which covers an application of GANs for image completion and it properly explains the project as well. http://bamos.github.io/2016/08/09/deep-completion/
- Ian Goodfellow's Tutorial on GANs for NIPS 2016 https://arxiv.org/pdf/1701.00160.pdf along with this https://channel9.msdn.com/Events/Neural-Information-Processing-Systems-Conference/Neural-Information-Processing-Systems-Conference-NIPS-2016/Generative-Adversarial-Networks
- Here is an awesome list of implementations in Tensorflow on different GAN related algorithms. https://github.com/wiseodd/generative-models/tree/master/GAN

I hope the review helped you. If you feel there's something more that you would have preferred from this review please leave a comment. That would immensely help me to improve feedback for any future reviews I conduct including for further projects. Would appreciate your input too. Thanks!

Congratulations on finishing the final project! 😄

Required Files and Tests

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

All the unit tests in project have passed.

All tests passed!

Build the Neural Network

The function model_inputs is implemented correctly.

Good work. You correctly implemented the placeholders while ensuring the datatype was correct!

The function discriminator is implemented correctly.

Excellent work on implementing multiple conv layers and appropriately applying the necessary activation functions and using batch normalization.

Couple of suggestions -

- Think about how filter size affects your model learning features for a CNN. As you increase the layers, your model learns more features, so does a larger filter size make sense or a smaller one to learn more features better?
- You currently use `leaky_ReLU3_flat = tf.reshape(leaky_ReLU3, [-1, 4 * 4 * 256])` for the `Dense` layer. Alternatively, you could simplify this step by using `flat = tf.contrib.layers.flatten(leaky_ReLU3)`. Slightly more convenient (I hate trying to reshape things :D)
- Do consider creating a separate function for leaky ReLU here. It's good programming practice.

Also, GANs are difficult to optimize. If the margin by which the discriminator wins is too big, then the generator can't learn well as the discriminator error would be too small. If the generator wins by too much, it won't learn well because the discriminator is too weak to teach it. There is a lot of research in GANs as you might know to solve all kinds of such problems. Some recommended ways would be -

- Use a smaller model for the discriminator relative to generator.
- Using dropout in discriminator so that it is less prone to learning the data distribution. Which you already did, awesome work!
- Use weight initialization. You already have that but some research papers recommend Xavier initialization. https://www.tensorflow.org/api_docs/python/tf/contrib/layers/xavier_initializer
- I believe this has been referred previously, but do try to read on this - https://arxiv.org/pdf/1606.03498v1.pdf

The function generator is implemented correctly.

Nicely done! Very impressive implementation.

One small suggestion -

- For your `logits`, you use `tf.layers.conv2d_transpose()` which has an argument `activation`. So to create `out` you can directly pass `tf.nn.tanh` as a parameter to `conv2d_transpose`. It doesn't make too much of a difference except making it a bit more concise.

The function model_loss is implemented correctly.

You correctly implemented the loss! Good work especially for using smoothing here!

Simple question for you - Why don't you apply label smoothing for generator loss?

The function model_opt is implemented correctly.

Excellent work on using `tf.control_dependencies()` here! I hope you understand the important of using that.

Also, you don't really need to implement that op for the discriminator here because the discriminator isn't used for inference in our case.

Also, instead of the above op, try to run the optimizer for the generator twice. How do you think that will help?

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**

Very well done!

Do you think different learning rates for generator and discriminator optimizers here would help? Try it out :)

Here is a good resouce on some tips and tricks on training GANs - https://github.com/soumith/ganhacks Do check it out!

The parameters are set reasonable numbers.

You selected a good set of hyperparameters.

Some suggestions to try out if you'd like -

- Try smaller batch sizes for each, especially for MNIST.
- Try different batch sizes for each.
- Do you think different `z_dim` values are required based on what you are trying to generate? Do you think it should be higher for the faces?
- You selected a good value for `beta1`. Here's a good post explaining the importance of beta values and which value might be empirically better. Do check it out! http://sebastianruder.com/optimizing-gradient-descent/index.html#adam Try to tune your values based on this.

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

Nicely done!

You are getting some really good results when I run your model. Which is awesome. The suggestions I have provided can help even more to improve upon your model, but I recommend you try to run it your model now for more epochs and then on different datasets :)

As you might remember from P2, there's a lot to experiment when it comes to CNNs, so I encourage you to keep expanding on this model of yours :D