Generate Faces

REVIEW | CODE REVIEW | HISTORY

Meets Specifications

Hi there,

Congratulations for passing the Generate Faces Project of this course 🎉 This was a very solid performance given understanding and complexity of the task, further your code cells are written concisely and effectively, successfully passing the functions tests.

Generally the networks can be improved and more realistic images can be generated by tweaking the models and hyperparameters, for a start in a trial and error fashion, which will lead to intuition over time and the results will be superior, this is a fundamental part of creating and training those type of models.

GANs are amazing and impressive, given that e.g. in this case faces are created out of thin air, however there are numerous real world applications, and more for which a deep dive into this technique is highly recommended and rewarding!

Great work for now, good luck with your next projects! 🔭

Required Files and Tests

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

Thank you for successfully submitting the worked through notebook, and a report in form of a .html version, way to go! 👌

See here for further elaborations on why this is important for cross-platform specific readability/functionality of the project.

✓ All the unit tests in project have passed.

The imported unit_tests for discriminator and generator where executed at destined locations and both passed, well done! ✅

Data Loading and Processing

✓ The function `get_dataloader` should transform image data into resized, Tensor image types and return a DataLoader that batches all the training data into an appropriate size.

The DataLoader is created accordingly and the hyperparameters specify a reasonable batch_size and the image_size = 32 as required, great!

✓ Pre-process the images by creating a `scale` function that scales images into a given pixel range. This function should be used later, in the training loop.

To make the training images compatible with generators tanh output, this rescaling step must scale the images between -1 and 1, what you have done correctly here 👍

Build the Adversarial Networks

✓ The Discriminator class is implemented correctly; it outputs one value that will determine whether an image is real or fake.

The discriminator is implemented correctly and it was opted for 3 convolutional layers, critically leaky_Relu functions where used in the forward pass to recover negative values, tensors where flattened accordingly 💯

✓ The Generator class is implemented correctly; it outputs an image of the same shape as the processed training data.

The Generator successfully processes the input vectors resulting in the output of the image_size as fed to the discriminator, nice! 👌

✓ This function should initialize the weights of any convolutional or linear layer with weights taken from a normal distribution with a mean = 0 and standard deviation = 0.02.

The weight initialisation step is performed with a normal distribution, as required in this model 💯 Feel free to revisit the lecture content linked above, and see the chapter again, for general approaches often used in other models ✓

Optimization Strategy

✓ The loss functions take in the outputs from a discriminator and return the real or fake loss.

Loss functions are successfully returning real, respectively fake losses, given input. 🖼️

[Note] A common source of error is using nn.BCELoss as criterion, which may result to inadequate training because of numerically unstable results, GANs specifically are supposed to be trained with BCEWithLogitsLoss, which you did correctly here, feel free to revisit course content for this.

✓ There are optimizers for updating the weights of the discriminator and generator. These optimizers should have appropriate hyperparameters.

Good choice using Adam optimiser, and setting all parameters, as a suggestion, beta2 is supposed to be set close to 1 for most effective training in computer vision problems, see further elaboration. on this here. 💡

Training and Results

✓ Real training images should be scaled appropriately. The training loop should alternate between training the discriminator and generator networks.

Awesome, models and data moved to cuda, as required for training! 👍

Further the optimisers are set to .zero_grad() this is important since the default setting is accumulating the gradients of the backward passes, which however is only of benefit in RNNs.

✓ There is not an exact answer here, but the models should be deep enough to recognize facial features and the optimizers should have parameters that help wth model convergence.

❕ **[Suggestion]** When observing the training process, keep in mind that the model trains the discriminator and the generator simultanously which is why results should not be read statically from the display of losses, however there are indices, namely when the losses of the discriminator hoover ≈ 0.5, which means that the discriminator hardly can trick the generator any longer, since fake and real images are shown in tandem. Having this in combo with the generator as our endproduct as a relatively low loss before overfitting, can be the optimal stopping point.

```
Epoch [   18/   18] | d_loss: 0.1377 | g_loss: 5.5243
Epoch [   18/   18] | d_loss: 0.0640 | g_loss: 5.4394
Epoch [   18/   18] | d_loss: 0.0452 | g_loss: 4.4711
Epoch [   18/   18] | d_loss: 0.2619 | g_loss: 2.6421
```

Out[26]: "\nDON'T MODIFY ANYTHING IN THIS CELL\n"

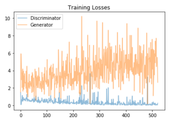
Generator loss is increasing while discriminator loss seems too low, the theoretical optimum was ≈ 0.5 for the discriminator, it can be attempted to optimise this by changing layers in both networks, also experimenting with learning rate and batch size

Training loss

Plot the training losses for the generator and discriminator, recorded after each epoch.

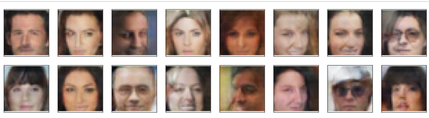
In [27]:
```
fig, ax = plt.subplots()
losses = np.array(losses)
plt.plot(losses.T[0], label='Discriminator', alpha=0.5)
plt.plot(losses.T[1], label='Generator', alpha=0.5)
plt.title("Training Losses")
plt.legend()
```

Out[27]: <matplotlib.legend.Legend at 0x7fe443bb2f28>



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

The models output are recognisable as faces, but some artifacts remain, and pixels are persistent:



To achieve better results the models could be tweaked in terms of hyperparameters as following:

- number of convolutional layers
- number of epochs
- learning rate
- optimizer, and changes in the betas if chosen Adam
- slope of leaky_relu

✓ The question about model improvement is answered.

💡 Suggestions for model improvement where given according to template, see above, additionally, more advanced steps that can be taken are Adversial Debasing, or Rejection Option-based Classification.

DOWNLOAD PROJECT

RETURN TO PATH