10, Deep Convolutional GAN (DCGAN)

Goal

In this notebook, you're going to create another GAN using the MNIST dataset. You will implement a Deep Convolutional GAN (DCGAN), a very successful and influential GAN model developed in 2015.

Note: here is the paper if you are interested! It might look dense now, but soon you'll be able to understand many parts of it :)

Learning Objectives

1. Get hands-on experience making a **widely used GAN: Deep Convolutional GAN (DCGAN)**.
2. Train a powerful generative model.

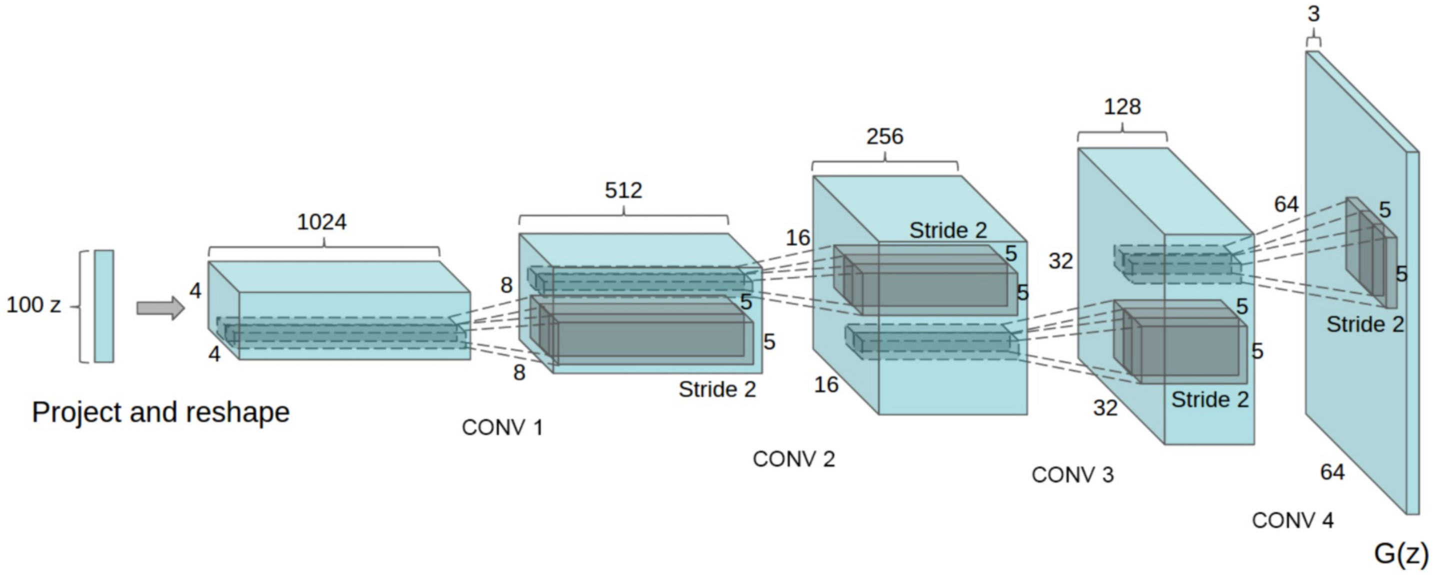


Figure: Architectural drawing of a generator from DCGAN from Radford et al (2016).

Getting Started

DCGAN

Here are the main features of DCGAN (don't worry about memorizing these, you will be guided through the implementation!):

- Use convolutions without any pooling layers
- Use batchnorm in both the generator and the discriminator
- Don't use fully connected hidden layers
- Use ReLU activation in the generator for all layers except for the output, which uses a Tanh activation.
- Use LeakyReLU activation in the discriminator for all layers except for the output, which does not use an activation

You will begin by importing some useful packages and data that will help you create your GAN. You are also provided a visualizer function to help see the images your GAN will create.

```
import torch
from torch import nn
from tqdm.auto import tqdm
from torchvision import transforms
from torchvision.datasets import MNIST
from torchvision.utils import make_grid
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
torch.manual_seed(0) # Set for testing purposes, please do not change!


def show_tensor_images(image_tensor, num_images=25, size=(1, 28, 28)):
    '''
    Function for visualizing images: Given a tensor of images, number of images, and
    size per image, plots and prints the images in an uniform grid.
    '''
    image_tensor = (image_tensor + 1) / 2
    image_unflat = image_tensor.detach().cpu()
    image_grid = make_grid(image_unflat[:num_images], nrow=5)
    plt.imshow(image_grid.permute(1, 2, 0).squeeze())
    plt.show()
```

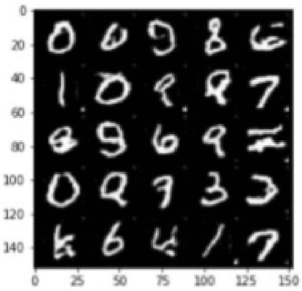
```
def show_tensor_images(image_tensor, num_images=25, size=(1,28,28)):
 image_tensor = (image_tensor + 1) / 2
```

▼ `image_tensor` variable

```
# fake_noise = get_noise(cur_batch_size, z_dim, device=device)
# fake = gen(fake_noise)
# image_tensor = fake
image_tensor = (image_tensor + 1) / 2
```

▼ result (have 25 img and size is (1, 28, 28))

>> example jerr



Step 23000

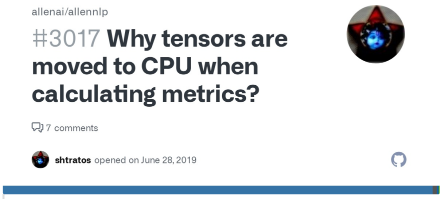
▼ `image_tensor.detach().cpu()`

> using tensors directly and that they are on the CPU.(without the additional layer that store computational graph)

Why tensors are moved to CPU when calculating metrics? · Issue #3017 · allenai/allennlp

System: OS: Ubuntu 18.04.2 Python version: 3.6.7 AllenNLP version: v0.8.4 PyTorch version: 1.1.0 Question When training a simple language model on GPU I get a big computational performance hit when calculating accuracy metrics. I've trac...

https://github.com/allenai/allennlp/issues/3017



allenai/allennlp

#3017 **Why tensors are moved to CPU when calculating metrics?**

7 comments

shtratos opened on June 28, 2019

same with the first GAN, but the difference here is didnt reshape the tensor using `.view(-1, *size)` after `image_tensor.detach().cpu()`

□ Outline of this video

□Notes

□Vocabs

□ things to double confirm

why convolutions without any pooling layers? >> any good?

why Don't use fully connected hidden layers?>> any reason?

why relu not all , output use tanh?

leakyrelu also, why output no need activation?

□Summary