生成对抗网络

本教程源代码目录在book/09.gan，初次使用请您参考Book文档使用说明。

说明:

1. 硬件环境要求： 本文可支持在CPU、GPU下运行
2. Docker镜像支持的CUDA/cuDNN版本： 如果使用了Docker运行Book，请注意：这里所提供的默认镜像的GPU环境为 CUDA 8/cuDNN 5，对于NVIDIA Tesla V100等要求CUDA 9的 GPU，使用该镜像可能会运行失败。
3. 文档和脚本中代码的一致性问题： 请注意：为使本文更加易读易用，我们拆分、调整了dc_gan.py的代码并放入本文。本文中代码与dc_gan.py的运行结果一致，可直接运行dc_gan.py进行验证。

背景介绍

生成对抗网络（Generative Adversarial Network [1]，简称GAN）是非监督式学习的一种方法，通过让两个神经网络相互博弈的方式进行学习。该方法最初由 Ian·Goodfellow 等人于2014年提出，原论文见 Generative Adversarial Network。

生成对抗网络由一个生成网络与一个判别网络组成。生成网络从潜在空间（latent space）中随机采样作为输入，其输出结果需要尽量模仿训练集中的真实样本。判别网络的输入为真实样本或生成网络的输出，其目的是将生成网络的输出从真实样本中尽可能分辨出来。而生成网络则要尽可能地欺骗判别网络。两个网络相互对抗、不断调整参数，其目的是将生成网络生成的样本和真实样本尽可能的区分开[2]）。

生成对抗网络常用于生成以假乱真的图片 [3]）。此外，该方法还被用于生成视频、三维物体模型等。

效果展示

本教程将 MNIST 数据集输入网络进行训练，经过19轮训练后可以看到，生成的图片已经非常接近真实图片的样子，下图中前8行是真实图片的样子，后8行是网络生成的图像效果：

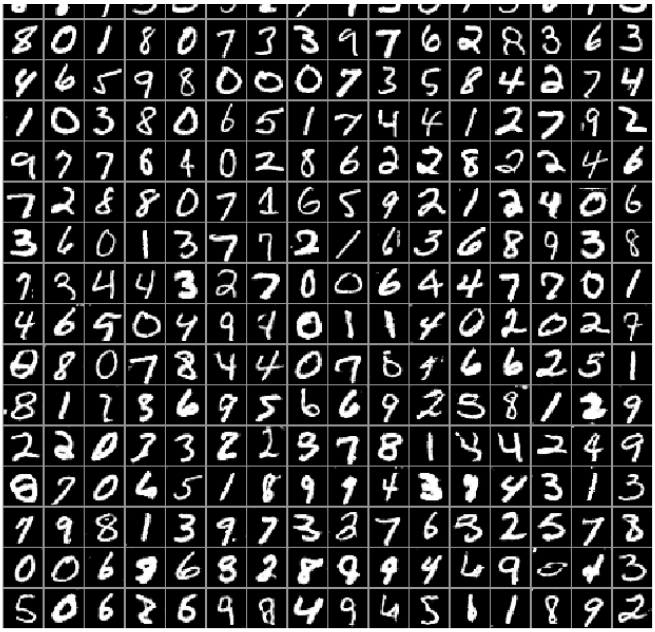

图1. GAN 生成手写数字效果

模型概览

GAN

GAN 网络顾名思义，是一种通过对抗的方式，去学习数据分布的生成模型。其中，“对抗”指的是生成网络（ Generator)和判别网络（ Discriminator)的相互对抗。这里以生成图片为例进行说明：

- 生成网络（ G ）接收一个随机的噪声z，尽可能的生成近似样本的图像，记为G(z)
- 判别网络（ D ）接收一张输入图片x，尽可以去判别该图像是真实样本还是网络生成的假样本，判别网络的输出 D(x) 代表 x 为真实图片的概率。如果 D(x)=1 说明判别网络认为该输入一定是真实图片，如果 D(x)=0 说明判别网络认为该输入一定是假图片。

在训练的过程中，两个网络互相对抗，最终形成了一个动态的平衡，上述过程用公式可以被描述为：

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim P_{data}(x)}\left[logD(x)\right] + \mathbb{E}_{z\sim p_z(z)}\left[log(1-D(G(z)))\right]$$

在最理想的情况下，G 可以生成与真实样本极其相似的图片G(z)，而 D 很难判断这张生成的图片是否为真，对图片的真假进行随机猜测，即 D(G(z))=0.5。

下图展示了生成对抗网络的训练过程，假设在训练开始时，真实样本分布、生成样本分布以及判别模型分别是图中的黑线、绿线和蓝线。在训练开始时，判别模型是无法很好地区分真实样本和生成样本的。接下来当我们固定生成模型，而优化判别模型时，优化结果如第二幅图所示，可以看出，这个时候判别模型已经可以较好地区分生成数据和真实数据了。第三步是固定判别模型，改进生成模型，试图让判别模型无法区分生成图片与真实图片，在这个过程中，可以看出由模型生成的图片分布与真实图片分布更加接近，这样的迭代不断进行，直到最终收敛，生成分布和真实分布重合，判别模型无法区分真实图片与生成图片。

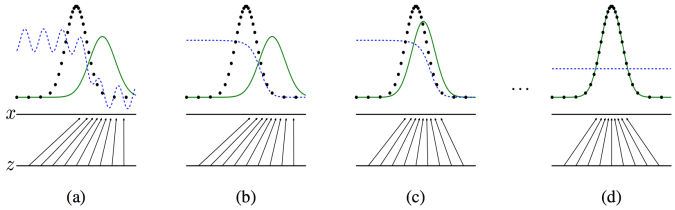

图2. GAN 训练过程

但是在实际过程中，很难得到这个完美的平衡点，关于GAN的收敛理论还在持续不断的研究中。

DCGAN

DCGAN [4] 是深层卷积网络与 GAN 的结合，其基本原理与 GAN 相同，只是将生成网络和判别网络用两个卷积网络（ CNN ）替代。为了提高生成样本的质量和网络的收敛速度，论文中的 DCGAN 在网络结构上进行了一些改进：

- 使用全卷积网络：去掉了FC层，以实现更深的网络结构。
- 激活函数：在生成器（G）中，最后一层使用Tanh函数，其余层采用 ReLu 函数；判别器（D）中都采用 LeakyReLu。

DCGAN中的生成器（G）结构如下图所示：

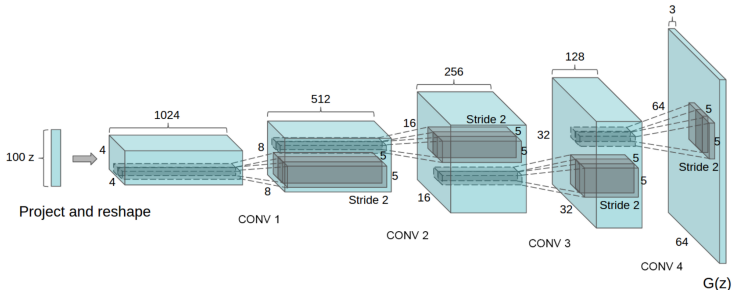

图3. DCGAN中的生成器（G）

数据准备

本次教程使用数据规模较小的 MNIST 训练生成器和判别器，该数据集可通过paddle.dataset模块自动下载到本地。

关于 MNIST 的详细介绍可参考数字识别。

训练模型

`09.gan/dc_gan.py` 演示了训练的整体过程。

加载包

首先加载 PaddlePaddle 的 Fluid 和其他相关包

```
import sys
import os
import matplotlib
import PIL
import six
import numpy as np
import math
import time
import paddle
import paddle.fluid as fluid

matplotlib.use('agg')
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
```

定义辅助工具

定义 plot 函数，将图像生成过程可视化

```
def plot(gen_data):
    pad_dim = 1
    paded = pad_dim + img_dim
    gen_data = gen_data.reshape(gen_data.shape[0], img_dim, img_dim)
    n = int(math.ceil(math.sqrt(gen_data.shape[0])))
    gen_data = (np.pad(
        gen_data, [[0, n * n - gen_data.shape[0]], [pad_dim, 0], [pad_dim, 0]],
        'constant').reshape((n, n, paded, paded)).transpose((0, 2, 1, 3))
                .reshape((n * paded, n * paded)))
    fig = plt.figure(figsize=(8, 8))
    plt.axis('off')
    plt.imshow(gen_data, cmap='Greys_r', vmin=-1, vmax=1)
    return fig
```

```
gf_dim = 64 # 生成器的feature map的基础通道数量，生成器中所有的feature map的通道数量都是基础通道数量的
df_dim = 64 # 判别器的feature map的基础通道数量，判别器中所有的feature map的通道数量都是基础通道数量的
gfc_dim = 1024 * 2 # 生成器的全连接层维度
dfc_dim = 1024 # 判别器的全连接层维度
img_dim = 28  # 输入图片的尺寸

NOISE_SIZE = 100  # 输入噪声的维度
LEARNING_RATE = 2e-4 # 训练的学习率

epoch = 20         # 训练的epoch数
output = "./output_dcgan"    # 模型和测试结果的存储路径
use_cudnn = False  # 是否使用cuDNN
use_gpu=False       # 是否使用GPU训练
```

定义网络结构

- bn 层

调用 `fluid.layers.batch_norm` 接口实现bn层，激活函数默认使用ReLu。

```
def bn(x, name=None, act='relu'):
    return fluid.layers.batch_norm(
        x,
        param_attr=name + '1',
        bias_attr=name + '2',
        moving_mean_name=name + '3',
        moving_variance_name=name + '4',
        name=name,
        act=act)
```

- 卷积层

调用 `fluid.nets.simple_img_conv_pool` 实现卷积池化组，卷积核大小为3x3，池化窗口大小为2x2，窗口滑动步长为2，激活函数类型由具体网络结构指定。

```
def conv(x, num_filters, name=None, act=None):
    return fluid.nets.simple_img_conv_pool(
        input=x,
        filter_size=5,
        num_filters=num_filters,
        pool_size=2,
        pool_stride=2,
        param_attr=name + 'w',
        bias_attr=name + 'b',
        use_cudnn=use_cudnn,
        act=act)
```

- 全连接层

```
def fc(x, num_filters, name=None, act=None):
    return fluid.layers.fc(input=x,
                           size=num_filters,
                           act=act,
                           param_attr=name + 'w',
                           bias_attr=name + 'b')
```

- 转置卷积层

在生成器中，需要用随机采样值生成全尺寸图像，dcgan使用转置卷积层进行上采样，在Fluid中，我们调用 `fluid.layers.conv2d_transpose` 实现转置卷积。

```
def deconv(x,
           num_filters,
           name=None,
           filter_size=5,
           stride=2,
           dilation=1,
           padding=2,
           output_size=None,
           act=None):
    return fluid.layers.conv2d_transpose(
        input=x,
        param_attr=name + 'w',
```

```
        stride=stride,
        dilation=dilation,
        padding=padding,
        use_cudnn=use_cudnn,
        act=act)
```

- 判别器

判别器使用真实数据集和生成器生成的假图片共同进行训练，在训练过程中尽量使真实数据集的输出结果为1，生成的假图片输出结果为0。本教程中实现的判别器由两个卷积池化层和两个全连接层组成，其中最后一个全连接层的神经元个数为1，输出一个二分类结果。

```
def D(x):
    x = fluid.layers.reshape(x=x, shape=[-1, 1, 28, 28])
    x = conv(x, df_dim, act='leaky_relu',name='conv1')
    x = bn(conv(x, df_dim * 2,name='conv2'), act='leaky_relu',name='bn1')
    x = bn(fc(x, dfc_dim,name='fc1'), act='leaky_relu',name='bn2')
    x = fc(x, 1, act='sigmoid',name='fc2')
    return x
```

- 生成器

生成器由两组带BN的全连接层和两组转置卷积层组成，网络输入为随机的噪声数据，最后一层转置卷积的卷积核数为1，表示输出为灰度图片。

```
def G(x):
    x = bn(fc(x, gfc_dim,name='fc3'),name='bn3')
    x = bn(fc(x, gf_dim * 2 * img_dim // 4 * img_dim // 4,name='fc4'),name='bn4')
    x = fluid.layers.reshape(x, [-1, gf_dim * 2, img_dim // 4, img_dim // 4])
    x = deconv(x, gf_dim * 2, act='relu', output_size=[14, 14],name='deconv1')
    x = deconv(x, num_filters=1, filter_size=5, padding=2, act='tanh', output_size=[28, 28],name
    x = fluid.layers.reshape(x, shape=[-1, 28 * 28])
    return x
```

损失函数

损失函数使用 `sigmoid_cross_entropy_with_logits`

```
def loss(x, label):
    return fluid.layers.mean(
        fluid.layers.sigmoid_cross_entropy_with_logits(x=x, label=label))
```

创建Program

```
d_program = fluid.Program()
dg_program = fluid.Program()

# 定义判别真实图片的program
with fluid.program_guard(d_program):
    # 输入图片大小为28*28=784
    img = fluid.layers.data(name='img', shape=[784], dtype='float32')
    # 标签shape=1
    label = fluid.layers.data(name='label', shape=[1], dtype='float32')
    d_logit = D(img)
    d_loss = loss(d_logit, label)

# 定义判别生成图片的program
with fluid.program_guard(dg_program):
    noise = fluid.layers.data(
        name='noise', shape=[NOISE_SIZE], dtype='float32')
    # 噪声数据作为输入得到生成图片
    g_img = G(x=noise)

    g_program = dg_program.clone()
    g_program_test = dg_program.clone(for_test=True)

    # 判断生成图片为真实样本的概率
    dg_logit = D(g_img)

    # 计算生成图片被判别为真实样本的loss
    dg_loss = loss(
```

使用adam作为优化器，分别优化判别真实图片的loss和判别生成图片的loss。

```
opt = fluid.optimizer.Adam(learning_rate=LEARNING_RATE)
opt.minimize(loss=d_loss)
parameters = [p.name for p in g_program.global_block().all_parameters()]
opt.minimize(loss=dg_loss, parameter_list=parameters)
```

数据集 Feeders 配置

下一步，我们开始训练过程。paddle.dataset.mnist.train()用做训练数据集。这个函数返回一个reader——PaddlePaddle中的reader是一个Python函数，每次调用的时候返回一个Python yield generator。

下面shuffle是一个reader decorator，它接受一个reader A，返回另一个reader B。reader B 每次读入buffer_size条训练数据到一个buffer里，然后随机打乱其顺序，并且逐条输出。

batch是一个特殊的decorator，它的输入是一个reader，输出是一个batched reader。在PaddlePaddle里，一个reader每次yield一条训练数据，而一个batched reader每次yield一个minibatch。

```
batch_size = 128  # Minibatch size

train_reader = paddle.batch(
    paddle.reader.shuffle(
        paddle.dataset.mnist.train(), buf_size=60000),
    batch_size=batch_size)
```

创建执行器

```
if use_gpu:
    exe = fluid.Executor(fluid.CUDAPlace(0))
else:
    exe = fluid.Executor(fluid.CPUPlace())

exe.run(fluid.default_startup_program())
```

开始训练

训练过程中的每一次迭代，生成器和判别器分别设置自己的迭代次数。为了避免判别器快速收敛到0，本教程默认每迭代一次，训练一次判别器，两次生成器。

```
t_time = 0
losses = [[], []]

# 判别器的迭代次数
NUM_TRAIN_TIMES_OF_DG = 2

# 最终生成图像的噪声数据
const_n = np.random.uniform(
    low=-1.0, high=1.0,
    size=[batch_size, NOISE_SIZE]).astype('float32')

for pass_id in range(epoch):
    for batch_id, data in enumerate(train_reader()):
        if len(data) != batch_size:
            continue

        # 生成训练过程的噪声数据
        noise_data = np.random.uniform(
            low=-1.0, high=1.0,
            size=[batch_size, NOISE_SIZE]).astype('float32')

        # 真实图片
        real_image = np.array(list(map(lambda x: x[0], data))).reshape(
            -1, 784).astype('float32')
        # 真实标签
        real_labels = np.ones(
            shape=[real_image.shape[0], 1], dtype='float32')
        # 虚假标签
        fake_labels = np.zeros(
            shape=[real_image.shape[0], 1], dtype='float32')
        total_label = np.concatenate([real_labels, fake_labels])
```

```
                                feed={'noise': noise_data},
                                fetch_list={g_img})[0]

        total_images = np.concatenate([real_image, generated_image])

        # D 判断虚假图片为假的Loss
        d_loss_1 = exe.run(d_program,
                           feed={
                               'img': generated_image,
                               'label': fake_labels,
                           },
                           fetch_list={d_loss})[0][0]

        # D 判断真实图片为真的Loss
        d_loss_2 = exe.run(d_program,
                           feed={
                               'img': real_image,
                               'label': real_labels,
                           },
                           fetch_list={d_loss})[0][0]

        d_loss_n = d_loss_1 + d_loss_2
        losses[0].append(d_loss_n)

        # 训练生成器
        for _ in six.moves.xrange(NUM_TRAIN_TIMES_OF_DG):
            noise_data = np.random.uniform(
                low=-1.0, high=1.0,
                size=[batch_size, NOISE_SIZE]).astype('float32')
            dg_loss_n = exe.run(dg_program,
                                feed={'noise': noise_data},
                                fetch_list={dg_loss})[0][0]
            losses[1].append(dg_loss_n)
        t_time += (time.time() - s_time)
        if batch_id % 10 == 0 :
            if not os.path.exists(output):
                os.makedirs(output)
            # 每轮的生成结果
            generated_images = exe.run(g_program_test,
                                       feed={'noise': const_n},
                                       fetch_list={g_img})[0]
            # 将真实图片和生成图片连接
            total_images = np.concatenate([real_image, generated_images])
            fig = plot(total_images)
            msg = "Epoch ID={0} Batch ID={1} D-Loss={2} DG-Loss={3}\n ".format(
                pass_id, batch_id,
                d_loss_n, dg_loss_n)
            print(msg)
            plt.title(msg)
            plt.savefig(
                '{}/{:04d}_{:04d}.png'.format(output, pass_id,
                                              batch_id),
                bbox_inches='tight')
            plt.close(fig)
```

打印特定轮次的生成结果：

```
def display_image(epoch_no,batch_id):
  return PIL.Image.open('output_dcgan/{:04d}_{:04d}.png'.format(epoch_no,batch_id))

# 观察第10个epoch，460个batch的生成图像：
display_image(10,460)
```

总结

DCGAN采用一个随机噪声向量作为输入，输入通过与CNN类似但是相反的结构，将输入放大成二维数据。采用这种结构的生成模型和CNN结构的判别模型，DCGAN在图片生成上可以达到相当可观的效果。本案例中，我们利用DCGAN生成了手写数字图片，您可以尝试更换数据集生成符合个人需求的图片，或尝试修改网络结构观察不一样的生成效果。

参考文献

[1] Goodfellow, Ian J.; Pouget-Abadie, Jean; Mirza, Mehdi; Xu, Bing; Warde-Farley, David; Ozair, Sherjil; Courville, Aaron; Bengio, Yoshua. Generative Adversarial Networks. 2014. arXiv:1406.2661 [stat.ML].

[2] Andrej Karpathy, Pieter Abbeel, Greg Brockman, Peter Chen, Vicki Cheung, Rocky Duan, Ian Goodfellow, Durk Kingma, Jonathan Ho, Rein Houthooft, Tim Salimans, John Schulman, Ilya Sutskever, And Wojciech

Techniques for Training GANs. 2016. arXiv:1606.03498 [cs.LG].

[4] Radford A, Metz L, Chintala S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks[J]. Computer Science, 2015.