华中科技大学

课
程
报
告

院（系） _____

专业班级 _____

学生姓名 _____

课程名称 信息检索

任课教师 魏巍

2018 年 12 月 13 日

信息检索算法实践

目录

摘要

本文旨在回顾并且部分实践信息检索技术在搜索引擎、推荐系统、卷积神经网络等算法中的应用，并进一步延伸、结合作者自身在生成对抗网络（Generative Adversarial Network）方面的研究，实现一种超分辨率图像重建算法。

关键词：信息检索，超分辨率图像重建，神经网络，推荐系统

1. 绪论

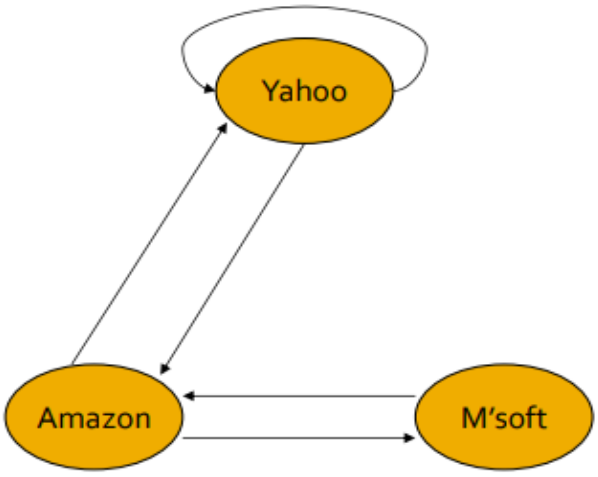
当今，计算机与网络已经发展到一个相当繁荣的阶段，人们通过网络查询头条新闻、点外卖、网购；不仅如此，学生可以通过慕课（MOOC）查询感兴趣的课程、程序员可以用CSDN发布帖子或经验，学者可以通过网络数据库系统查询需要的文献、数据。不论是个人生活还是专业领域，总能看到信息检索的影响：要知道，如此方便快捷的信息获取方式，得益于信息检索技术以及各类算法的发展。信息检索技术早已不是 19 世纪末冷门枯燥的“图书馆文献检索”，而与当下火热的推荐算法、数据挖掘、表示学习等息息相关。

提到信息检索技术，不难想到搜索引擎：百度、Google、360等等。搜索引擎通过全文搜索从网站提取文本与超链接信息，以此给相关的网页进行排序；一种著名网页排序算法叫PageRank算法，它采用网站的“链入”和“链出”量，给网页进行打分，按分值的高低降序展示搜索到的网站。虽然仅用该方法不能够完全适应用户需求，且容易滋生垃圾网站，但仍然是许多现代搜索引擎的算法基础。除了搜索引擎以外，运用机器学习算法提取文本或图像的抽象信息的技术也是当下的热门：机器学习在硬件发展的促进下，能够运用优化算法迭代式地“进化”，最终达到逼近目标数据分布或预测、分类等目的；其效果远优于以信号处理为基础的传统方法。例如推荐算法，便是通过学习用户给商品的评分及文本信息来推测用户的偏好，进而给模拟用户打分、给用户推荐新商品；最新的推荐算法，比如Facebook的算法，甚至将用户的翻页速度、聊天信息也进行了提取，只是为了获得更好的推荐效果。更常见的方法则有集成学习后求平均的方法，可以减小训练的偏差。

2. 搜索引擎

搜索引擎是指能自动从因特网搜索信息，经过整理分析后，提供给用户查询的系统。因特网上的信息量庞大而且没有规律，网站犹如汪洋中的小小岛屿，网页链接则构成小岛之间的桥梁；因此，用户就像一艘轮船，如果没有地图，想要通过不断的网页链接去搜索目的地将会变得十分麻烦。而借助搜索引擎，用户就像拿到了一张信息地图，它可以随时提取各网站的文本信息，建立数据库，检索出符合用户查询条件且得分高的网站记录。

PageRank 算法是一种典型的全文搜索引擎。它的核心思想就是，一个网页的重要性与其它有链接的网页是有关的；因此，用户从任意一个网站开始，不断迭代地随机跳转到从该网页链出的下一个网页，那么不难得到一个矩阵代表从某网页到另一个网页的概率。该矩阵的第 i 行第 j 个元素的值代表第 i 个网页从第 j 个网页所获得的分数。显然，每列的元素求和值为 1，但每行的元素求和值是不确定的。分数越高，则表明该网页越“重要”，在搜索引擎展示查询结果时就越应该放在前列。



	y	a	m
y	1/2	1/2	0
a	1/2	0	1
m	0	1/2	0

如上图 1 所示，有三个网站 Yahoo、Amazon、Microsoft，其中 Yahoo 网页链出端有 Yahoo 自己以及 Amazon，Amazon 链出到 Yahoo 和 Microsoft，而 Microsoft 只链出到 Amazon。假使每个网页能够自身拥有 1 分，并在下一轮重新分配分数时，将这一分平均分配给所有该网页的链出端，即：Yahoo 分给自己 0.5 分，给 Amazon0.5 分；Amazon 给 Yahoo0.5 分，给 Microsoft0.5 分；Microsoft 给 Amazon1 分；正如矩阵所示。一轮迭代后，显然 Amazon 得到 1.5 分，Yahoo 得到 1 分，Microsoft 得到 0.5 分。那么，根据得分高低的降序规则，PageRank 算法将先显示 Amazon 网页，而后是 Yahoo，最后是 Microsoft。

3. 推荐算法

推荐算法的思路有多种，其中较为知名的有：基于内容的、基于协同过滤、潜在语义向量（Latent Factor）的信息推荐方法。其中，潜在语义向量的推荐方式比较难理解，数学解释较为复杂；因此，本文将实践该推荐方法，并将其与大数据分析 MapReduce 思想结合，在 movielens 的无文本两千万次评分数据集上进行测试。（目前还未完成，预计完成时间在一周之内，后续的代码将会发送到 https://github.com/CarterDLaw/Latent-Factor-Model-with-mrjobs-Hadoop-on-movielens-20m，欢迎浏览评论收藏关注。）

3.1 基于内容的推荐算法

基于内容的理论依据主要来自信息检索和信息过滤，根据用户过去的浏览记录向用户推荐用户没有接触过的推荐项。可以从两个方法描述基于内容的推荐方法：启发式的方法和基于模型的方法。启发式的方法就是用户凭借经验来定义相关的计算公式，然后再根据公式的计算结果和实际的结果进行验证，然后再不断修改公式以达到最终目的。基于模型的方法学习以往的数据，期望能够获得有效的信息。

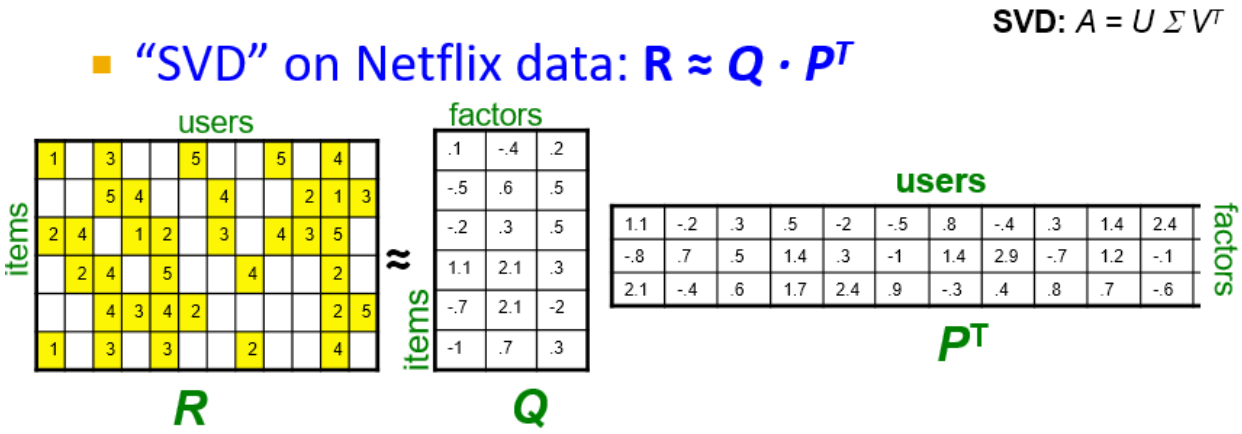
3.2 基于协同过滤的推荐算法

协同过滤(Collaborative filtering)的含义是指利用兴趣相投的群体的喜好来推荐用户感兴趣的信息，个人通过合作的机制给予信息相当程度的回应（比如评分）并记录下来，以达到过滤的目的，进而帮助别人筛选信息。基于协同过滤的算法具有一些优势：能过滤难以基于内容分析的、复杂抽象的概念，推荐个人化、自动化程度高。这种算法需要按照 Jaccard 或 cosine 这类的相似度公式，计算出各种商品之间的相似度。

3.3 基于潜在语义向量（Latent Factor）的推荐算法

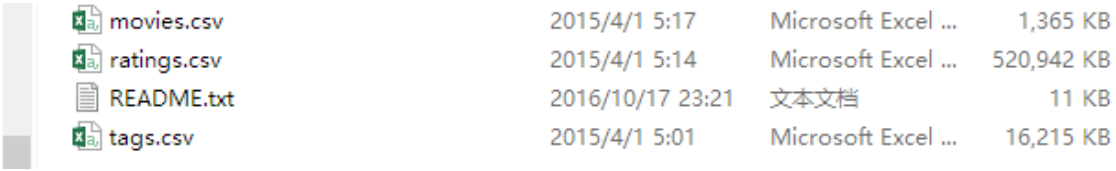
与奇异值分解 SVD 相类似，在得到用户对商品的评分矩阵 R 后，我们可以将其分解为 Q 与 P_Transpose 两个矩阵；这两个矩阵的向量长度作为一个超参数，由人为设定。分解的核心思想在于：认为可以利用 SVD 创建潜空间，使得该两个矩阵的值为参数，只需利用

梯度下降的优化方法，即可找到将具有潜在语义的向量并使 Q 与 P_Transpose 相乘可以逆推回原始的矩阵 R，并补齐用户评分的空缺（并不是所有的商品用户都有进行评分，所以存在空元素；矩阵 R 中展示的商品至少有一个用户评分）。这是一种无文本 SVD 矩阵运算的迭代优化方法，补齐的空缺对于推荐用户没评分但可能感兴趣的商品非常有价值。



当前该算法的实践还在进行中，不过可以先说明编程思路，并且展示部分代码。

首先，要认识数据集：这是一个包含 20000263 个、由 138493 用户给出的关于 27278 部电影的评分，需要用到的数据集有两个：movies.csv 与 ratings.csv，分别包含 MovieID 序列、“UserID、MovieID、rating”。



movies.csv	2015/4/1 5:17	Microsoft Excel ...	1,365 KB
ratings.csv	2015/4/1 5:14	Microsoft Excel ...	520,942 KB
README.txt	2016/10/17 23:21	文本文档	11 KB
tags.csv	2015/4/1 5:01	Microsoft Excel ...	16,215 KB

进行数据清洗，先将 ratings.csv 读入 python，使用 numpy 转化为读写速度更快的 npy 格式；转 npy 格式的代码如下：

```
import csv
import numpy as np
from scipy import sparse

with open("D:/ml-20m/ratings.csv", encoding='utf-8') as csvRating:
    reader = csv.reader(csvRating)
    data = []
    for index, item in enumerate(reader):
        if (index != 0):
            data.append(item)
    for index, item in enumerate(data):
        item[0] = int(item[0])
        item[1] = int(item[1])
        item[2] = float(item[2])
        data[index] = [item[0], item[1], item[2]]
    np.save("rating.npy", data)
```

此外，将 ratings.csv 中的 movieID 转化为从 0 开始连续的整数序列，直到读取（原有的 movieID 在 movie.csv 中保存，但 ratings.csv 中只有用户评分过的 movieID，未评分的 ID 则不会出现，因此其 movieID 并不连续，不利于读取。）因此，可以利用 python 的字典（基于哈希表）数据结构，将 ratings 中有的 movieID 按顺序分别映射为：1，2，…

而后，再次读取 rating.npy，按字典改动第二列 movieID：

```
with open("D:/ml-20m/movies.csv", encoding='utf-8') as csvMovie:
    reader = csv.reader(csvMovie)
    data = []
    Movie = []
    dic = {}
    for index, item in enumerate(reader):
        if (index != 0):
            data.append(item)
    for item in data:
        item[0] = int(item[0])
        Movie.append(item[0])
    for index, item in enumerate(Movie):
        dic[item] = index
    np.save("movie_dict.npy", dic)

import numpy as np

dic = np.load("movie_dict.npy").item()
rating = np.load("rating.npy")
rating_alter = []
for index, item in enumerate(rating):
    item[1] = dic.get(item[1])
    rating_alter.append(item)
    if (index % 100000 == 0):
        print(index)
np.save("rating_alter.npy", rating_alter)
```

而后，将 rating_alter 分为训练集和测试集（数据量比例约为 100：1）；其中测试集是该矩阵的左下部分，而训练集取测试集的互逆集合。

```
train_set = []
test_set = []
rating_alter = np.load("rating_alter.npy")
for index, item in enumerate(rating_alter):
    if (item[0] >= 133493 and item[1] < 1000): # 挖出左下角
        test_set.append(item)
    else:
        train_set.append(item)
np.save("train_set.npy", train_set)
np.save("test_set.npy", test_set)
```

训练时，要将该训练集和测试集均转为稀疏矩阵，便于进行矩阵运算。

该代码在更小的数据集上已经成功运行，但对于一个几百万乘几十万维的超大矩阵，则不能够直接运算，需要运用分治的思想，将其分解为多个任务，汇总得到最终结果，也即 MapReduce 思想。具体实现可以调用 mrjob 的 python 包或使用 hadoop 平台命令。

```
import numpy as np
from scipy import sparse
import tensorflow as tf


def my_init(size):  # 从[-5,5]的均匀分布中采样得到维度是size的输出
    # return tf.random_normal(size, mean=0.0, stddev=1)
    return tf.random_uniform(size, -5, 5)


def regularization(X):
    x = tf.reduce_sum(tf.square(X))
    return x


def replacement(r, c, mat):
    for i in range(r):
        for j in range(c):
            if (mat[i, j] == 0):
                mat[i, j] = 0
    return mat


def compute_loss(r, c, mov, q, p):
    loss = 0
    for index, item in enumerate(r):
        items = int(c[index])
        item = int(item)
        loss += np.square(mov[item,items] - np.dot(q[item,:],p[:,items]))
    return loss


length = 1000 # 超参数 factor长度
iteration = 10000
```

```
Q = tf.Variable(my_init([138493, length]))
P = tf.Variable(my_init([length, 27278]))
variables = [Q, P]

'''
# 恢复上一次训练的参数
Q = tf.Variable(np.load("Q.npy"))
P = tf.Variable(np.load("P.npy"))
'''

# Movie = tf.placeholder(tf.float16, shape=[138493,27278])
row_train = np.load("train_set.npy")[:,0] - 1 # 从一开始。应该减一，使之从零开始
col_train = np.load("train_set.npy")[:,1]
data_train = np.load("train_set.npy")[:,2]
movie_train = sparse.coo_matrix((data_train,(row_train,col_train)),shape=(138493,27278)).tocsr()
# Movie_predict = tf.matmul(Q, P)
Loss_train = compute_loss(row_train, col_train, movie_train, Q, P)


row_test = np.load("test_set.npy")[:,0] - 1 # 从一开始。应该减一，使之从零开始
col_test= np.load("test_set.npy")[:,1]
data_test = np.load("test_set.npy")[:,2]
movie_test = sparse.coo_matrix((data_test,(row_test,col_test)),shape=(138493,27278)).tocsr()
Loss_test = compute_loss(row_test, col_test, movie_test, Q, P)

row_test = np.load("test_set.npy")[:,0] - 1 # 从一开始。应该减一，使之从零开始
col_test= np.load("test_set.npy")[:,1]
data_test = np.load("test_set.npy")[:,2]
movie_test = sparse.coo_matrix((data_test,(row_test,col_test)),shape=(138493,27278)).tocsr()
Loss_test = compute_loss(row_test, col_test, movie_test, Q, P)


global_step = tf.Variable(0, name='global_step', trainable=False)
learn_rate = tf.train.exponential_decay(learning_rate=0.0002, global_step=global_step, decay_steps=50, decay_rate=0.98,
                                        staircase=False)
solver = tf.train.AdamOptimizer(learning_rate=learn_rate).minimize(Loss_train, var_list=variables)


# 创建对话，初始化所有变量
config = tf.ConfigProto(allow_soft_placement=True, log_device_placement=False)
config.gpu_options.allow_growth = True
sess = tf.Session(config=config)
sess.run(tf.global_variables_initializer())

for i in range(iteration):
    _, loss_train = sess.run([solver, Loss_train])
    loss_test = sess.run([Loss_test])
    print('iteration %s, loss_train: %s, loss_test: %s' % (i, loss_train, loss_test))
    global_step = global_step + 1

sess.close()
```

4. 基于神经网络的数据挖掘与信息提取

深度神经网络对于提取抽象信息有非常不可思议的效果。在一些基础知识的介绍过后，下文将展示对于图像超分辨率研究，并展示图像超分辨率 SRGAN 算法的代码实现。

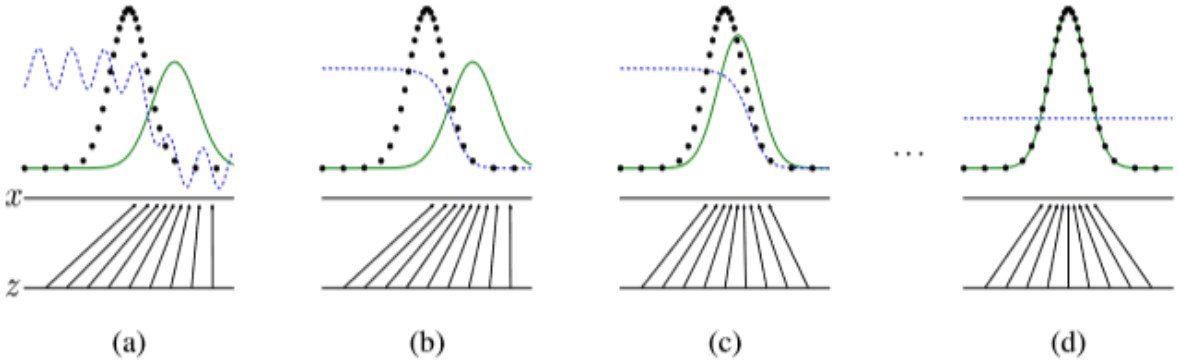
4.1 神经网络基本概念

神经网络是一种运算模型，由大量的节点（神经元）间相互联接构成。每个节点代表一种特定的输出函数，称为激励函数（activation function）。两个节点间的连接都代表一个对于通过该连接信号的加权值，称之为权重。人工神经网络可以看作是对自然界某种算法、数据分布或者函数的逼近，也可以是对一种逻辑思维的表达。神经网络的学习方式一般为梯度下降的一阶优化方法；由于普通的梯度下降速度慢，遇到悬崖等情况不能很好适应，因此衍生出如 Momentum、Adam 等多种自适应学习率的优化方法。

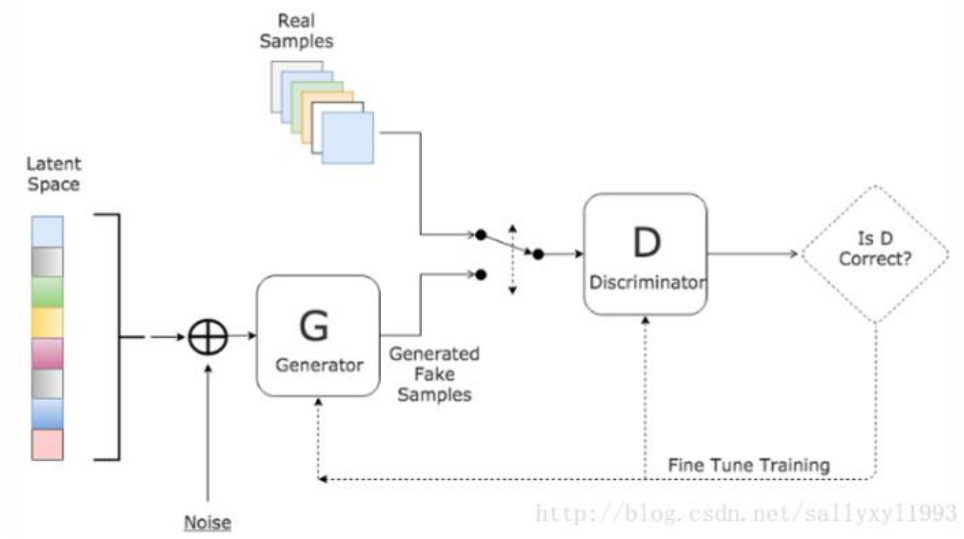
4.2 生成对抗网络基本概念

生成对抗网络（Generative Adversarial Network，GAN）最早由 Google Brain 机器学习科学家 Goodfellow 提出，其基本思想是基于深度学习的二人零和博弈。GAN 拥有两个“博弈者”模型，一个是生成模型（G），负责从某一先验噪声分布$p_z(z)$中学习生成器的输出数据分布p_g，因此可以定义为$\mathrm{G}(\mathrm{z};\theta_g)$，其中$\theta_g$代表生成网络各神经元层的权重；另一个是判别模型（D），$\mathrm{D}(\mathrm{x};\theta_d)$输出一个从 0 到 1 的实数值，代表输入的数据x来自于真实的数据分布而不是p_g的概率。其中，x代表真实数据分布，p_g是生成器输出数据分布。生成对抗网络的目标是：通过大量训练，同时使 D 能最大化分配正确标签给来自 G 的数据与真实数据的概率，并使 G 最小化$\log(1-\mathrm{D}(\mathrm{G}(\mathrm{z})))$。通俗理解，GAN 想通过训练判别网络使之能够准确区分输入数据来自于 G 还是真实数据分布，此时$\mathrm{D}\big(\mathrm{G}(\mathrm{z})\big)$自然趋近于 1，最小化达成。可以将两个目标化为一个方程，即

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z})))]. \tag{1}$$



生成对抗网络同时更新判别分布 D(蓝色的虚线所示)，使之能够区分输入样本来自于真实数据分布p_x（黑色点状曲线）还是生成分布p_g（绿色的实线）。下方的水平线分别代表是噪声 z 与样本 x 的分布（在这个图中为均匀分布），而 z 是通过 G 映射到 x，即$\mathrm{x} = \mathrm{G}(\mathrm{z})$。不断训练后，G 和 D 都无法继续学习而更新参数，因为此时$p_g = p_{data}$；判别器无法区分这两种分布，故而有$\mathrm{D}(\mathrm{x}) = 1/2$。



GAN 的精妙之处在于，通过同时训练 D 与 G，它们可以互相学习并且按照 D 的梯度更新参数，不断升级、变得更强大；最终，就能够产生一个相当逼近于真实数据分布的p_g，以至于很强大的判别网络也无法区分从它中间所产生的数据是否来源于真实数据分布，达到“以假乱真”的地步。换句话说，GAN 能够自动地学习原始真实样本集的数据分布，无论这个分布多么复杂。GAN 的本质是一种无监督学习，因为我们事先并没有对真实数据与生成数据贴上标签。GAN 无需特定的代价函数，学习过程可以学习到很好的特征表示。

4.3 基于生成对抗的超分辨率图像重建算法

超分辨率生成对抗网络（Super-resolution Generative Adversarial Network）的目标是使得 G 生成网络通过低分辨率的图像I^{LR}生成高分辨率图像I^{HR}，由 D 判别网络判断输入 D 的图像是由 G 网生成还是数据库中的原图像。该 SRGAN 的目标函数是：

$$\min_{\theta_G}\max_{\theta_D}\ \mathbb{E}_{I^{HR}\sim p_{\text{train}}(I^{HR})}[\log D_{\theta_D}(I^{HR})]+ \mathbb{E}_{I^{LR}\sim p_G(I^{LR})}[\log(1-D_{\theta_D}(G_{\theta_G}(I^{LR}))] \tag{2}$$

通过批量训练，优化该目标函数，生成网络能够不断学习，以至于生成出于真实高清图片极相似的图片，让判别网络无法成功进行分类。至此，生成网络不但完成生成数据分布与真实图片靠近的任务，而且还成功通过卷积网络的特性对图片分辨率进行放大，最终的代价函数定义为：

$$l^{SR} = {l_X}^{SR} + 10^{-3}{l_{Gen}}^{SR} \tag{3}$$

右边第一项代表内容损失（Content Loss），第二项代表对抗损失（Adversarial Loss）。

Christian Ledig 等人在论文《Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network》中提到，常用于作优化目标的对应像素点之和的 MSE 损失会诱导致生成图片的纹理过于平滑，因此应当采用 VGG 损失[6]，其公式如下：

$$l^{SR}_{VGG/i,j} = \frac{1}{W_{i,j}H_{i,j}}\sum_{x=1}^{W_{i,j}}\sum_{y=1}^{H_{i,j}}(\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2 \tag{4}$$

其中$\varphi_{i,j}$代表由在 VGG19 网络中第 i 个最大池化层前的第 j 个激活后卷积层的特征图。定义 VGG 损失为重建图像$G_{\theta_G}(I^{LR})$与原图像I^{HR}的特征表示间的欧式距离。$W_{i,j}$和$H_{i,j}$分别描述 VGG 网络内特征图的维度。

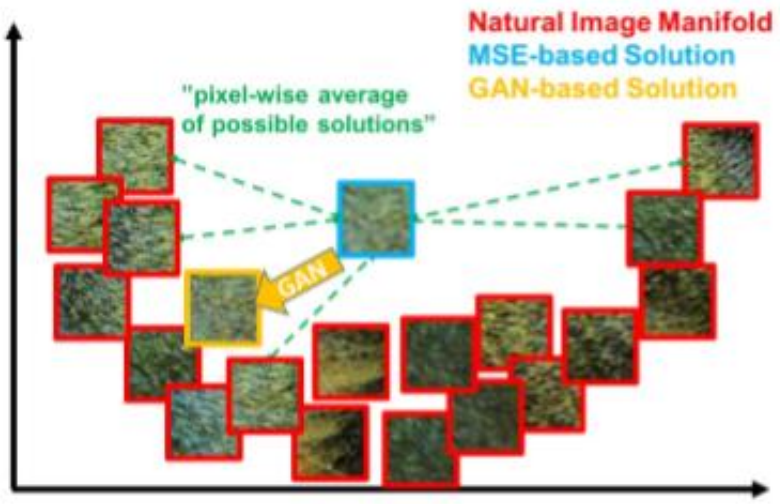
该论文展示了 SRGAN 生成的更高分辨率图像与测试集图像、其它生成方法得到图像的对比。其中，括号中代表的数值分别为 PSNR 与 SSIM 值，分别表征峰值信噪比以及与原图像之间的相似性。可见，SRGAN 在两个指标上的表现相当优异，虽然不及 SRResNet；而且，它的清晰度应当是比 SRResNet 和 bicubic 更高的。

需要注意的是，SSIM 作为一种结构相似性算法，是对两个图像每个像素点的值之间的差别进行平方求和，并不能够很好地表征两个图像每个局部位置的相似性。通过肉眼观察 SRGAN 与 SRResNet 的图像，不难发现，SRGAN 生成的图像清晰度优于 SRResNet。

此外，定义对抗损失为

$$l_{Gen}^{SR} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR})) \tag{5}$$

其中，log 内的项代表重建图像$G_{\theta_G}(I^{LR})$是一个自然高清图的概率；该项损失鼓励网络的图像生成分布更贴近于自然高清图像的流形分布，可以生成更自然的超分辨率重建图像。



上图中，红色边框图片来自于自然图片流形分布，蓝色边框图片为 MSE 得到超分辨率图像集，橘色边框图片为该 GAN 得到的图像集。

4.4 基于生成对抗的超分辨率图像重建代码实现

超分辨率图像重建实现的代码已贴至附录，包括四个函数（主函数 main.py、模型函数 model.py、图像改动函数 utils.py、超参数设置与文件路径函数 config.py），在 python 编程平台上最终运行主函数即可。实现图像重建的数据集来源于 DIV2K - bicubic downscaling x4 competition：http://www.vision.ee.ethz.ch/ntire17/。

未来展望

过去半年在深度学习上的研究中，我越来越感觉到数据以及信息提取的至关重要。有时候，数据获取难易甚至能够直接决定一个好的 idea 能不能被成功做出来。还有一个想法就是，最近一年经常和物理、化学做机器学习交叉，使用全连接网络预测工程问题，经常受到数据来源的限制，比如训练集来源于 A、B、C 类，但最后预测的却是一个和这三类都有点关系但也说不上来关系有多大的第四类——D。毫无疑问，即便是把正则化方法全都套上去，这个问题也是没法解决的；因为正则化解决的是过拟合问题，而通过学习一种分布去预测其它的相关分布却不是过拟合导致的。苦恼的时候，我就想到是否可以引入类似于潜空间的学习方法，学习工程问题的抽象特征，而不只是弄函数上的拟合。这样的方法也有可能直接学习到 D 和 ABC 相关的 mutual 特征。当然，这只是我自己的不成熟的见解。

对于未来，我还是充满期待；希望将来能够在这个领域继续砥砺前行。

致谢

在完成本篇论文的过程中，计算机 15 级刘伟光同学给予了我很大的帮助；我们在一起探讨推荐算法的实践、Hadoop 环境的搭建。魏老师您课上的内容，包括推荐算法、聚类等都给了我许多启发。作为物理学院学生，能够参与到一门关于机器学习的课程里来不容易，（即便只是公选也学到了很多）毕竟学校对于跨专业选课仍然有种种限制。之前我是几乎没有接触过推荐算法以及聚类等，所以就选择推荐算法作为这门课的课程设计。这门课我个人感觉，虽然迫于课时少，无法讲的深入，但是易于消化、能帮忙节省很多课下自学的时间。更重要的是，它能够给予其它没有专业知识但想要了解机器学习与信息检索领域的同学一个窗口、机会。感谢老师辛勤的付出！

附录

SRGAN 代码实现：

“config.py”

```
from easydict import EasyDict as edict
import json

config = edict()
config.TRAIN = edict()

## Adam
config.TRAIN.batch_size = 16
config.TRAIN.lr_init = 1e-4
config.TRAIN.beta1 = 0.9

## initialize G
config.TRAIN.n_epoch_init = 0

## adversarial learning (SRGAN)
config.TRAIN.n_epoch = 1500
config.TRAIN.lr_decay = 0.1
config.TRAIN.decay_every = int(config.TRAIN.n_epoch / 2)

## train set location
config.TRAIN.hr_img_path = 'E:/train_HR/DIV2K_train_HR/'
config.TRAIN.lr_img_path = 'E:/train_LR_bicubic_X4/DIV2K_train_LR_bicubic/'

config.VALID = edict()
## test set location
config.VALID.hr_img_path = 'E:/valid_HR/DIV2K_valid_HR/'
config.VALID.lr_img_path = 'E:/valid_LR_bicubic_X4/DIV2K_valid_LR_bicubic/'

config.VALID.logdir = 'E: /SRGAN_Wasserstein/'
def log_config(filename, cfg):
    with open(filename, 'w') as f:
        f.write("================================================\n")
        f.write(json.dumps(cfg, indent=4))
        f.write("\n================================================\n")
```

“model.py”

```
import tensorflow as tf
import tensorlayer as tl
from tensorlayer.layers import *

def SRGAN_g(t_image, is_train=False, reuse=False):

    w_init = tf.random_normal_initializer(stddev=0.02)
    b_init = None # tf.constant_initializer(value=0.0)
    g_init = tf.random_normal_initializer(1., 0.02)
    with tf.variable_scope("SRGAN_g", reuse=reuse) as vs:
        tl.layers.set_name_reuse(reuse)
        n = InputLayer(t_image, name='in')
        n = Conv2d(n, 64, (9, 9), (1, 1), act=tf.nn.relu, padding='SAME', W_init=w_init,
name='n64s1/c')

        temp = n
        # B residual blocks
        for i in range(5):
            nn = Conv2d(n, 64, (3, 3), (1, 1), act=None, padding='SAME', W_init=w_init, b_init=b_init,
name='n64s1/c1/%s' % i)
            nn = BatchNormLayer(nn, act=tf.nn.relu, is_train=is_train, gamma_init=g_init,
name='n64s1/b1/%s' % i)
            nn = Conv2d(nn, 64, (3, 3), (1, 1), act=None, padding='SAME', W_init=w_init, b_init=b_init,
name='n64s1/c2/%s' % i)
            nn = BatchNormLayer(nn, is_train=is_train, gamma_init=g_init, name='n64s1/b2/%s' % i)
            nn = ElementwiseLayer([n, nn], tf.add, 'b_residual_add/%s' % i)
            n = nn

        n = Conv2d(n, 64, (3, 3), (1, 1), act=None, padding='SAME', W_init=w_init, b_init=b_init,
name='n64s1/c/m')
        n = BatchNormLayer(n, is_train=is_train, gamma_init=g_init, name='n64s1/b/m')
        n = ElementwiseLayer([n, temp], tf.add, 'add3')
        # B residual blacks end

        n = Conv2d(n, 256, (3, 3), (1, 1), act=None, padding='SAME', W_init=w_init, name='n256s1/1')
        n = SubpixelConv2d(n, scale=2, n_out_channel=None, act=tf.nn.relu, name='pixelshufflerx2/1')

        n = Conv2d(n, 256, (3, 3), (1, 1), act=None, padding='SAME', W_init=w_init, name='n256s1/2')
        n = SubpixelConv2d(n, scale=2, n_out_channel=None, act=tf.nn.relu, name='pixelshufflerx2/2')

        n = Conv2d(n, 3, (9, 9), (1, 1), act=tf.nn.tanh, padding='SAME', W_init=w_init, name='out')
        return n

def SRGAN_d(input_images, is_train=True, reuse=False):
```

```
    w_init = tf.random_normal_initializer(stddev=0.02)
    b_init = None # tf.constant_initializer(value=0.0)
    gamma_init=tf.random_normal_initializer(1., 0.02)
    lrelu = lambda x: tl.act.lrelu(x, 0.2)
    with tf.variable_scope("SRGAN_d", reuse=reuse):
        tl.layers.set_name_reuse(reuse)
        in_lay = InputLayer(input_images, name='input/images')
        net_h0 = Conv2d(in_lay, 64, (4, 4), (2, 2), act=lrelu,
                padding='SAME', W_init=w_init, name='h0/c')

        net_h1 = Conv2d(net_h0, 128, (4, 4), (2, 2), act=None,
                padding='SAME', W_init=w_init, b_init=b_init, name='h1/c')
        net_h1 = BatchNormLayer(net_h1, act=lrelu, is_train=is_train,
                gamma_init=gamma_init, name='h1/bn')
        net_h2 = Conv2d(net_h1, 256, (4, 4), (2, 2), act=None,
                padding='SAME', W_init=w_init, b_init=b_init, name='h2/c')
        net_h2 = BatchNormLayer(net_h2, act=lrelu, is_train=is_train,
                gamma_init=gamma_init, name='h2/bn')
        net_h3 = Conv2d(net_h2, 512, (4, 4), (2, 2), act=None,
                padding='SAME', W_init=w_init, b_init=b_init, name='h3/c')
        net_h3 = BatchNormLayer(net_h3, act=lrelu, is_train=is_train,
                gamma_init=gamma_init, name='h3/bn')
        net_h4 = Conv2d(net_h3, 1024, (4, 4), (2, 2), act=None,
                padding='SAME', W_init=w_init, b_init=b_init, name='h4/c')
        net_h4 = BatchNormLayer(net_h4, act=lrelu, is_train=is_train,
                gamma_init=gamma_init, name='h4/bn')
        net_h5 = Conv2d(net_h4, 512, (1, 1), (1, 1), act=None,
                padding='SAME', W_init=w_init, b_init=b_init, name='h5/c')
        net_h5 = BatchNormLayer(net_h5, is_train=is_train,
                gamma_init=gamma_init, name='h5/bn')

        net = Conv2d(net_h5, 128, (1, 1), (1, 1), act=None,
                padding='SAME', W_init=w_init, b_init=b_init, name='res/c1')
        net = BatchNormLayer(net, act=lrelu, is_train=is_train,
                gamma_init=gamma_init, name='res/bn1')
        net = Conv2d(net, 512, (3, 3), (1, 1), act=None,
                padding='SAME', W_init=w_init, b_init=b_init, name='res/c2')
        net = BatchNormLayer(net, is_train=is_train,
                gamma_init=gamma_init, name='res/bn2')
        net_h6 = ElementwiseLayer(layer=[net_h5, net],
                combine_fn=tf.add, name='res/add')
        net_h6.outputs = tl.act.lrelu(net_h6.outputs, 0.2)

        net_ho = FlattenLayer(net_h6, name='ho/flatten')
        net_ho = DenseLayer(net_ho, n_units=1, act=tf.identity,
```

```
                W_init = w_init, name='ho/dense')
        logits = net_ho.outputs
        # Wasserstein GAN doesn't need the sigmoid output
        # net_ho.outputs = tf.nn.sigmoid(net_ho.outputs)
    return net_ho, logits
```

“utils.py”

```
from tensorlayer.prepro import *
import scipy

def get_imgs_fn(file_name, path):
    #Input an image path and name, return an image array """
    # return scipy.misc.imread(path + file_name).astype(np.float)
    return scipy.misc.imread(path + file_name, mode='RGB')

def crop_sub_imgs_fn(x, is_random=True):
    x = crop(x, wrg=384, hrg=384, is_random=is_random)
    x = x / (255. / 2.)
    x = x - 1.
    return x

def downsample_fn(x):
    # We obtained the LR images by downsampling the HR images using bicubic kernel with downsampling
factor r = 4.
    x = imresize(x, size=[96, 96], interp='bicubic', mode=None)
    x = x / (255. / 2.)
    x = x - 1.
    return x
```

“main.py”

```
import os, time, pickle, random, time
from datetime import datetime
import numpy as np
from time import localtime, strftime
import logging, scipy

import tensorflow as tf
import tensorlayer as tl
from model import *
from utils import *
from config import config, log_config

import os
```

```
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

###==================== HYPER-PARAMETERS ========================###
batch_size = config.TRAIN.batch_size
lr_init = config.TRAIN.lr_init
beta1 = config.TRAIN.beta1
## initialize G
n_epoch_init = config.TRAIN.n_epoch_init
## adversarial learning (SRGAN)
n_epoch = config.TRAIN.n_epoch
lr_decay = config.TRAIN.lr_decay
decay_every = config.TRAIN.decay_every
logdir = config.VALID.logdir

ni = int(np.sqrt(batch_size))

def read_all_imgs(img_list, path='', n_threads=32):
    imgs = []
    for idx in range(0, len(img_list), n_threads):
        b_imgs_list = img_list[idx : idx + n_threads]
        b_imgs = tl.prepro.threading_data(b_imgs_list, fn=get_imgs_fn, path=path)
        imgs.extend(b_imgs)
        print('read %d from %s' % (len(imgs), path))
    return imgs

def train():
    ## create folders to save result images and trained model
    save_dir_ginit = "samples/{}_ginit".format(tl.global_flag['mode'])
    save_dir_gan = "samples/{}_gan".format(tl.global_flag['mode'])
    tl.files.exists_or_mkdir(save_dir_ginit)
    tl.files.exists_or_mkdir(save_dir_gan)
    checkpoint_dir = "checkpoint"  # checkpoint_resize_conv
    tl.files.exists_or_mkdir(checkpoint_dir)

    ###==================== PRE-LOAD DATA ========================###
    train_hr_img_list = sorted(tl.files.load_file_list(path=config.TRAIN.hr_img_path, regx='.*.png',
printable=False))
    #train_lr_img_list = sorted(tl.files.load_file_list(path=config.TRAIN.lr_img_path, regx='.*.png',
printable=False))

    ###========================== DEFINE MODEL ============================###
    ## train inference
    t_image = tf.placeholder('float32', [batch_size, 96, 96, 3],
name='t_image_input_to_SRGAN_generator')
    t_target_image = tf.placeholder('float32', [batch_size, 384, 384, 3], name='t_target_image')
```

```
    net_g = SRGAN_g(t_image, is_train=True, reuse=False)
    net_d, logits_real = SRGAN_d(t_target_image, is_train=True, reuse=False)
    _,     logits_fake = SRGAN_d(net_g.outputs, is_train=True, reuse=True)

    net_g.print_params(False)
    net_d.print_params(False)

    ## test inference
    net_g_test = SRGAN_g(t_image, is_train=False, reuse=True)

    # ###======================== DEFINE TRAIN OPS ==========================###
    # Wasserstein GAN Loss
    with tf.name_scope('w_loss/WARS_1'):
        d_loss =  tf.reduce_mean(logits_fake) - tf.reduce_mean(logits_real)
        tf.summary.scalar('w_loss', d_loss)

    merged = tf.summary.merge_all()
    g_gan_loss = - 1e-3 * tf.reduce_mean(logits_fake)
    mse_loss = tl.cost.mean_squared_error(net_g.outputs , t_target_image, is_mean=True)

    g_loss = mse_loss + g_gan_loss

    g_vars = tl.layers.get_variables_with_name('SRGAN_g', True, True)
    d_vars = tl.layers.get_variables_with_name('SRGAN_d', True, True)

    with tf.variable_scope('learning_rate'):
        lr_v = tf.Variable(lr_init, trainable=False)
    ## Pretrain
    g_optim_init = tf.train.RMSPropOptimizer(lr_v).minimize(mse_loss, var_list=g_vars)
    g_optim = tf.train.RMSPropOptimizer(lr_v).minimize(g_loss, var_list=g_vars)
    d_optim = tf.train.RMSPropOptimizer(lr_v).minimize(d_loss, var_list=d_vars)
    clip_D = [p.assign(tf.clip_by_value(p, -0.01, 0.01)) for p in d_vars]

    ###======================== RESTORE MODEL =============================###
    sess = tf.Session(config=tf.ConfigProto(allow_soft_placement=True, log_device_placement=False,
device_count={'gpu':0}))
    loss_writer = tf.summary.FileWriter(logdir, sess.graph)
    tl.layers.initialize_global_variables(sess)
    if tl.files.load_and_assign_npz(sess=sess,
name=checkpoint_dir+'/g_{}.npz'.format(tl.global_flag['mode']), network=net_g) is False:
        tl.files.load_and_assign_npz(sess=sess,
name=checkpoint_dir+'/g_{}_init.npz'.format(tl.global_flag['mode']), network=net_g)
    tl.files.load_and_assign_npz(sess=sess,
name=checkpoint_dir+'/d_{}.npz'.format(tl.global_flag['mode']), network=net_d)
```

```
    ###========================== TRAINING ============================###
    ## use first `batch_size` of train set to have a quick test during training
    sample_imgs = read_all_imgs(train_hr_img_list[0:batch_size], path=config.TRAIN.hr_img_path,
n_threads=32) # if no pre-load train set
    sample_imgs_384 = tl.prepro.threading_data(sample_imgs, fn=crop_sub_imgs_fn, is_random=False)
    print('sample HR sub-image:',sample_imgs_384.shape, sample_imgs_384.min(), sample_imgs_384.max())
    sample_imgs_96 = tl.prepro.threading_data(sample_imgs_384, fn=downsample_fn)
    print('sample LR sub-image:', sample_imgs_96.shape, sample_imgs_96.min(), sample_imgs_96.max())
    tl.vis.save_images(sample_imgs_96, [ni, ni], save_dir_ginit+'/_train_sample_96.png')
    tl.vis.save_images(sample_imgs_384, [ni, ni], save_dir_ginit+'/_train_sample_384.png')
    tl.vis.save_images(sample_imgs_96, [ni, ni], save_dir_gan+'/_train_sample_96.png')
    tl.vis.save_images(sample_imgs_384, [ni, ni], save_dir_gan+'/_train_sample_384.png')

    ###======================== initialize G ====================###
    ## fixed learning rate
    sess.run(tf.assign(lr_v, lr_init))
    print(" ** fixed learning rate: %f (for init G)" % lr_init)
    for epoch in range(0, n_epoch_init+1):
        epoch_time = time.time()
        total_mse_loss, n_iter = 0, 0

        random.shuffle(train_hr_img_list)
        for idx in range(0, len(train_hr_img_list), batch_size):
            step_time = time.time()
            b_imgs = tl.prepro.threading_data(train_hr_img_list[idx : idx + batch_size],
fn=get_imgs_fn, path=config.TRAIN.hr_img_path)
            b_imgs_384 = tl.prepro.threading_data(b_imgs, fn=crop_sub_imgs_fn, is_random=True)
            b_imgs_96 = tl.prepro.threading_data(b_imgs_384, fn=downsample_fn)

            ## update G
            errM, _ = sess.run([mse_loss, g_optim_init], {t_image: b_imgs_96, t_target_image:
b_imgs_384})
            print("Epoch [%2d/%2d] %4d time: %4.4fs, mse: %.8f " % (epoch, n_epoch_init, n_iter,
time.time() - step_time, errM))
            total_mse_loss += errM
            n_iter += 1
        log = "[*] Epoch: [%2d/%2d] time: %4.4fs, mse: %.8f" % (epoch, n_epoch_init, time.time() -
epoch_time, total_mse_loss/n_iter)
        print(log)

        ## quick evaluation on train set
        if (epoch != 0) and (epoch % 10 == 0):
            out = sess.run(net_g_test.outputs, {t_image: sample_imgs_96})#; print('gen sub-image:',
out.shape, out.min(), out.max())
```

```
            print("[*] save images")
            tl.vis.save_images(out, [ni, ni], save_dir_ginit+'/train_%d.png' % epoch)

        ## save model
        if (epoch != 0) and (epoch % 10 == 0):
            tl.files.save_npz(net_g.all_params,
name=checkpoint_dir+'/g_{}_init.npz'.format(tl.global_flag['mode']), sess=sess)

    ###========================== train GAN (SRGAN) =========================###
    for epoch in range(0, n_epoch+1):
        ## update learning rate
        if epoch !=0 and (epoch % decay_every == 0):
            new_lr_decay = lr_decay ** (epoch // decay_every)
            sess.run(tf.assign(lr_v, lr_init * new_lr_decay))
            log = " ** new learning rate: %f (for GAN)" % (lr_init * new_lr_decay)
            print(log)
        elif epoch == 0:
            sess.run(tf.assign(lr_v, lr_init))
            log = " ** init lr: %f  decay_every_init: %d, lr_decay: %f (for GAN)" % (lr_init,
decay_every, lr_decay)
            print(log)

        epoch_time = time.time()
        total_d_loss, total_g_loss, n_iter = 0, 0, 0

        random.shuffle(train_hr_img_list)
        for idx in range(0, len(train_hr_img_list), batch_size):
            step_time = time.time()
            b_imgs_list = train_hr_img_list[idx : idx + batch_size]
            b_imgs = tl.prepro.threading_data(b_imgs_list, fn=get_imgs_fn,
path=config.TRAIN.hr_img_path)
            b_imgs_384 = tl.prepro.threading_data(b_imgs, fn=crop_sub_imgs_fn, is_random=True)
            b_imgs_96 = tl.prepro.threading_data(b_imgs_384, fn=downsample_fn)

            ## update D
            errD, summary, _, _ = sess.run([d_loss, merged, d_optim, clip_D], {t_image: b_imgs_96,
t_target_image: b_imgs_384})
            loss_writer.add_summary(summary, idx)

            ## update G
            errG, errM, errA, _ = sess.run([g_loss, mse_loss, g_gan_loss, g_optim],{t_image: b_imgs_96,
t_target_image: b_imgs_384})
            print("Epoch [%2d/%2d] %4d time: %4.4fs, W_loss: %.8f g_loss: %.8f (mse: %.6f adv: %.6f)"
                  % (epoch, n_epoch, n_iter, time.time() - step_time, errD, errG, errM, errA))
            total_d_loss += errD
```

```
            total_g_loss += errG
            n_iter += 1

        log = "[*] Epoch: [%2d/%2d] time: %4.4fs, d_loss: %.8f g_loss: %.8f" % (epoch, n_epoch,
time.time() - epoch_time, total_d_loss/n_iter, total_g_loss/n_iter)
        print(log)

        ## quick evaluation on train set
        if (epoch != 0) and (epoch % 10 == 0):
            out = sess.run(net_g_test.outputs, {t_image: sample_imgs_96})#; print('gen sub-image:',
out.shape, out.min(), out.max())
            print("[*] save images")
            tl.vis.save_images(out, [ni, ni], save_dir_gan+'/train_%d.png' % epoch)

        ## save model
        if (epoch != 0) and (epoch % 10 == 0):
            tl.files.save_npz(net_g.all_params,
name=checkpoint_dir+'/g_{}.npz'.format(tl.global_flag['mode']), sess=sess)
            tl.files.save_npz(net_d.all_params,
name=checkpoint_dir+'/d_{}.npz'.format(tl.global_flag['mode']), sess=sess)

def evaluate():
    ## create folders to save result images
    save_dir = "samples/{}".format(tl.global_flag['mode'])
    tl.files.exists_or_mkdir(save_dir)
    checkpoint_dir = "checkpoint"

    ###====================== PRE-LOAD DATA ===========================###
    valid_hr_img_list = sorted(tl.files.load_file_list(path=config.VALID.hr_img_path, regx='.*.png',
printable=False))
    valid_lr_img_list = sorted(tl.files.load_file_list(path=config.VALID.lr_img_path, regx='.*.png',
printable=False))

    valid_lr_imgs = read_all_imgs(valid_lr_img_list, path=config.VALID.lr_img_path, n_threads=32)
    valid_hr_imgs = read_all_imgs(valid_hr_img_list, path=config.VALID.hr_img_path, n_threads=32)

    ###========================== DEFINE MODEL ============================###
    imid = 64
    valid_lr_img = valid_lr_imgs[imid]
    valid_hr_img = valid_hr_imgs[imid]

    valid_lr_img = (valid_lr_img / 127.5) - 1   # rescale to [-1, 1]

    size = valid_lr_img.shape
    t_image = tf.placeholder('float32', [None, size[0], size[1], size[2]], name='input_image')
```

```
    net_g = SRGAN_g(t_image, is_train=False, reuse=False)

    ###====================== RESTORE G ============================###
    sess = tf.Session(config=tf.ConfigProto(allow_soft_placement=True, log_device_placement=False))
    tl.layers.initialize_global_variables(sess)
    tl.files.load_and_assign_npz(sess=sess, name=checkpoint_dir+'/g_srgan.npz', network=net_g)

    ###====================== EVALUATION ============================###
    start_time = time.time()
    out = sess.run(net_g.outputs, {t_image: [valid_lr_img]})
    print("took: %4.4fs" % (time.time() - start_time))

    print("LR size: %s /  generated HR size: %s" % (size, out.shape))
    print("[*] save images")
    tl.vis.save_image(out[0], save_dir + '/valid_gen.png')

    out_bicu = scipy.misc.imresize(valid_lr_img, [size[0]*4, size[1]*4], interp='bicubic', mode=None)
    tl.vis.save_image(out_bicu, save_dir + '/valid_bicubic.png')

if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('--mode', type=str, default='srgan', help='srgan, evaluate')
    args = parser.parse_args()

    tl.global_flag['mode'] = args.mode
    if tl.global_flag['mode'] == 'srgan':
        train()
    elif tl.global_flag['mode'] == 'evaluate':
        evaluate()
    else:
        raise Exception("Unknow --mode")
```