哈尔滨工业大学

<<模式识别与深度学习>>

实验 5 生成式对抗网络实现

(2020 春季学期)

学院：计算机科学与技术

学号：1170300909

姓名：武磊

指导老师：左旺孟

日期：2020.5.19

目录

一、实验目的

1. 基于 Pytorch 实现生成式对抗网络
   1) 拟合给定的数据分布
   2) 要求可视化训练过程
2. 要求实验中包含 GAN，WGAN，WGAN-GP 的对比和分析

二、实验环境配置

1. 硬件：
   CPU ： Inter Core i7-7500U 2.70GHz
   GPU ： NVIDIA GeForce 940MX
2. 软件：
   操作系统:WIN10
   软件：
   Python 3.6 (基于 Anaconda)
   Pytorch 1.2.0
   CUDA 10.1 V10.1.105
   cudnn v7.6.3 for CUDA 10.1
3. 开发 IDE
   Pycharm

三、实验过程

1. 选择代价函数，超参数，优化器
   1) 模型的超参数
      模型主要的超参数主要有以下因素：
      1. 学习率
      2. Batch size
      3. 网络结构，主要要有网络层数，网络结构设计等等
      4. 其他：
         a) 在 WGAN 中，作者通过 weight clipping 在每次更新判别器网络参数的时候将网络参数裁剪至一定范围，需要设定 weight clipping limit。
         b) 在 WGAN-GP 中，作者通过引入正则化的方式，保证了判别器表征的函数f_w 满足 Lipschitz 条件，惩罚项有超参数λ。

   2) 代价函数选择

- 在 GAN 中可以选择使用二分类交叉熵，也可以自己实现 log_loss 函数
- 在 WGAN 中，作者提出了 Wasserstein distance，具有更加优越的性质。
- 在 WGAN-GP 中，需要在 WGAN 的基础上，添加梯度惩罚项。具体实现和思路见下文。

3) 优化器选择

先前参照论文和网络博客，有以下感性认识：

- 优化器首选 RMSprop，由于 GAN 训练困难，原始论文中设计的 loss 函数被证明是不合理的，所以 loss 的梯度极其不稳定。

在实验中比较 RMSprop，Adam，SGD 等不同优化器，比较对于 GAN 网络收敛的效果。

2. 设计定义网络

本次实验中主要需要尝试用 GAN 网络去拟合指定分布，同时尝试通过 WGAN，WGAN-GP 等方式提升 GAN 模型，从收敛速度，训练的稳定性，loss 曲线，可视化感受等多方面比较。

一个重要的问题是需要考虑用什么指标来衡量生成器生成分布的质量，常见衡量两个分布的方式有 KL 散度，JS 散度等等，还有 WGAN 中提出的 Wasserstein distance。

1) GAN

a) 网络结构设计

本次实验任务很简单，只需要拟合简单的高斯分布，所以网络主体只需要复杂度适当的 MLP 即可。

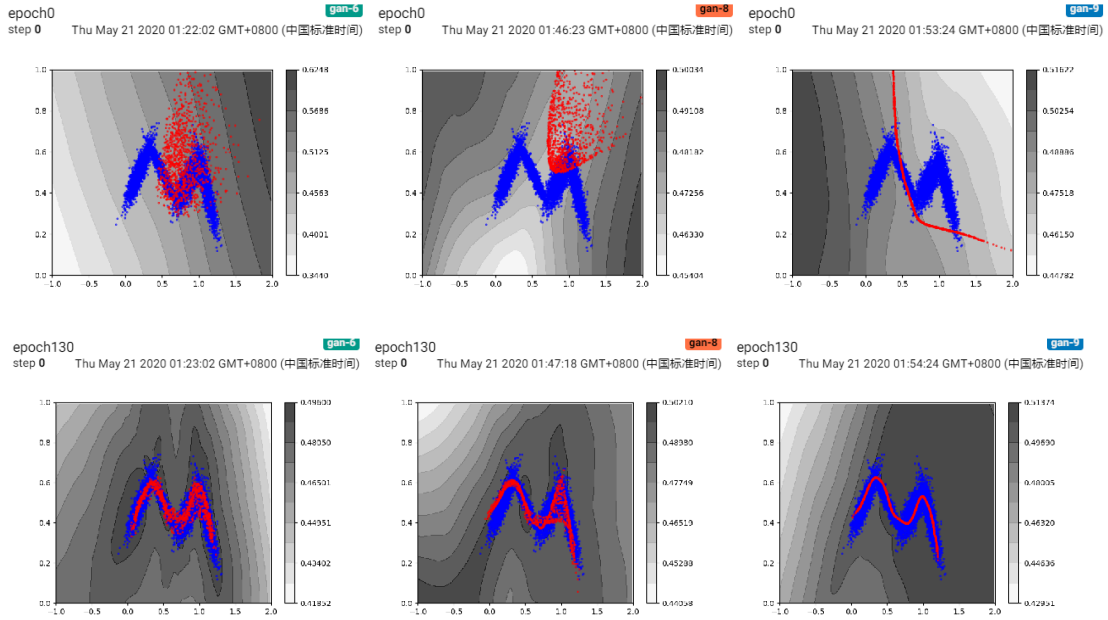
```
1. # GAN
2. discriminator = Discriminator(input_size,hidden_dim,output_size)
3. generator = Generator(noise_dim,hidden_dim,output_size)
```

如上图所示：

- 判别器接受点集作为输入，经过隐含层，最后输出经过 Sigmoid 函数输出一维向量，标识判别器对输入点集的评价（打分）。

  由于经过 Sigmoid，输出的范围为[0 – 1]，我们认为靠近 0 的输出，表示判别为负例，靠近 1 的输出，表示判别为正例。
- 生成器接受随机噪声作为输入，经过隐含层，生成拟合点集。
  - ✓ 注意：随机噪声的维度可能会对拟合的效果产生很大的影响，因为原始输入在经过映射后能够表示的分布一定程度上受限于原始输入的维度。

    在本问题中由于点集本身的维度就较小，所以 noise_dim 对生成器的生成效果有限，但是在图片生成的过程中，就会有影响。如：1 维随机向量和 100 维随机向量作为原始输入，输出图片的多样性就会产生差异。



上图是 $noise\_\dim = [5,2,1]$ 是 GAN 利用 Adam 优化器拟合出的效果，可以看出生成器的输出分布受限于原始输入。

b) 训练过程

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

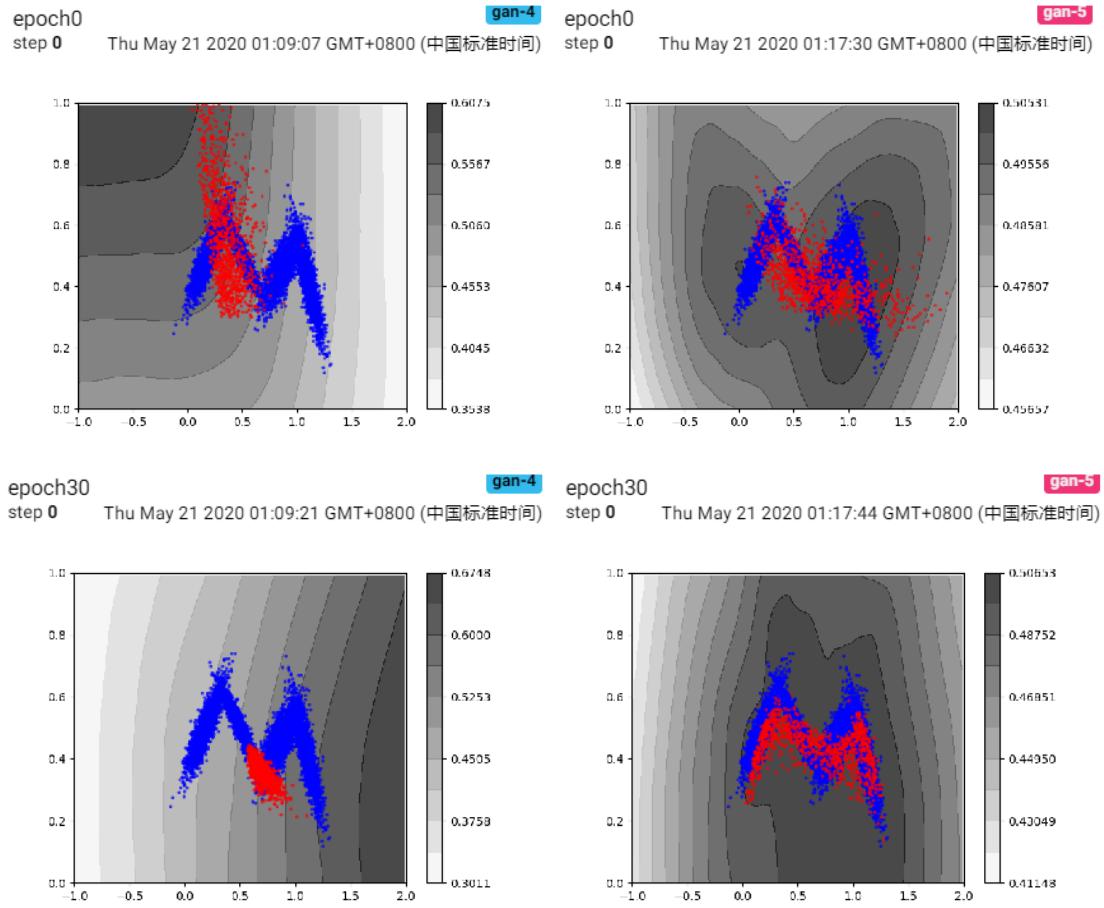
end for

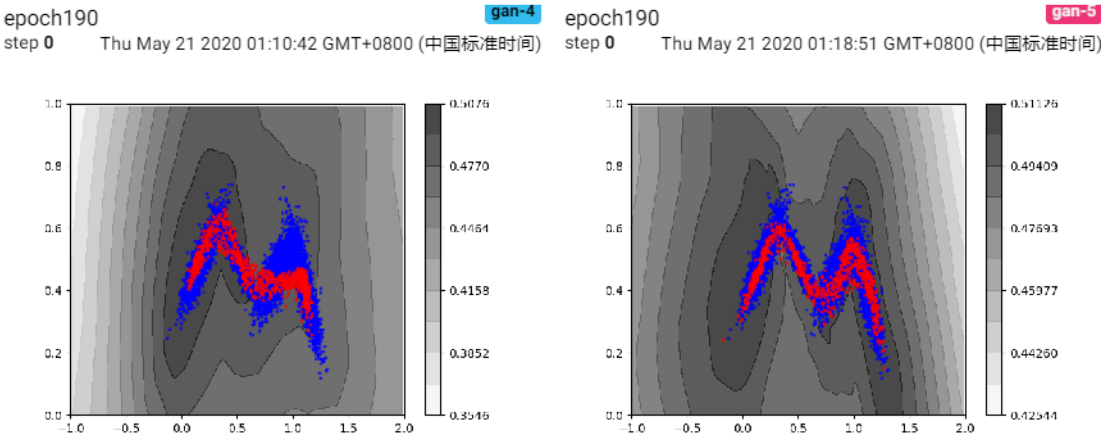
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

• 参考 Ian.J Goodfellow 2014 年的论文，首先训练判别器 D 一次或多次，更新参数；训练生成器 G,更新参数。简单的来说，我们希望判别器能够识别出所有正例（标记为 1），识别出所有的负例（标记为 0）；生成器能够生成足以被判别器识别为正例的点集。

```
#loss 函数是二分类交叉熵
loss_fn = nn.BCELoss()
# 判别器 loss
real_loss = loss_fn(discriminator(real_batch), valid)
fake_loss = loss_fn(discriminator(gen_data.detach()), fake)
d_loss = (real_loss + fake_loss) / 2
#生成器 loss
g_loss = loss_fn(discriminator(gen_data), valid)
```

• 实际上在训练过程中尝试了 first G then D (gan-4) 和 first D then G (gan-5)，效果对比：显然，先训练 G 提前获得一个较好的判别器是有助于收敛过程进行的。





2) WGAN

a) 网络结构设计

GAN 在训练过程中一直存在随缘训练，loss 无法指示训练过程进行，难以收敛等难题。

WGAN 的作者在 GAN 基础上做了一下改进：

• 提出新的 loss 判别函数

• 在此基础上修改了判别器网络结构：去掉的 Sigmoid 输出，改为全连接；在每次更新 D 网络参数，做了 weight clipping 操作。

具体解释可以参照论文原文以及【1】中对于 WGAN 的分析。实际上 WGAN 的作者针对 GAN 中存在的梯度消失和爆炸现象做了细致的数学分析和推导，发现原始 GAN 网络的 loss 函数的不合理性质导致了训练的不稳定。

b) 训练过程

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

训练过程和 GAN 的区别主要在于 loss 函数计算和 weight clipping 操作：

- Loss 函数计算

  在原始 GAN 网络中，判别器的 loss 函数可以表示成：

$$C(G) = -\log(4) + KL\left(p_{\text{data}} \middle\| \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \middle\| \frac{p_{\text{data}} + p_g}{2}\right)$$

$$C(G) = -\log(4) + 2 \cdot JSD\left(p_{\text{data}} \| p_g\right)$$

但是 JS 散度和 KL 散度的特点（如下图所示，参数θ表征两个分布距离）在于真实数据和生成数据分布的相似性会随着拟合分布的变化非线性的跃变，导致梯度消失和爆炸。同时，由于 KL 具有不对称的特点，使得生成器更愿意生成安全，重复的样本（loss 惩罚小），导致的 Mode collapse。

- $JS(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} \log 2 & \text{if } \theta \neq 0\ , \\ 0 & \text{if } \theta = 0\ , \end{cases}$
- $KL(\mathbb{P}_\theta \| \mathbb{P}_0) = KL(\mathbb{P}_0 \| \mathbb{P}_\theta) = \begin{cases} +\infty & \text{if } \theta \neq 0\ , \\ 0 & \text{if } \theta = 0\ , \end{cases}$

作者基于此提出了 Wasserstein 距离，能够更加合理的衡量生成器生成数据和原始数据的距离，同时 Wasserstein 距离具有很好的平滑性，使得在计算梯度时，不会出现梯度消失和爆炸的现象。同时，Wasserstein 距离（不是散度，具有自反性），避免了散度不对称的缺点，有效解决了 Mode Collapse 现象。

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\left[\, \|x - y\| \,\right]\ ,$$

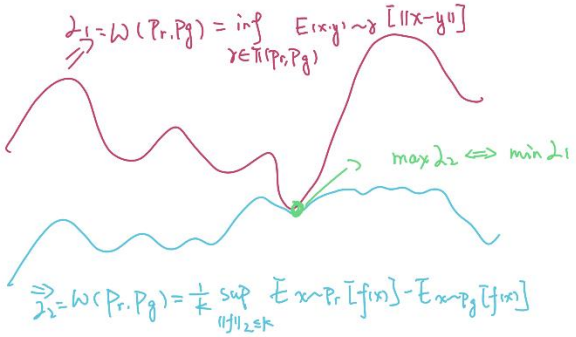
$$W(\mathbb{P}_0, \mathbb{P}_\theta) = |\theta|,$$

Wasserstein 距离的公式含义是指对于所有P_r, P_g所可能的联合分布，计算这两个分布中样本x, y 的的距离期望，求取该期望的下界。但是遍历所有联合分布现在是难求解问题，通过数学变换，我们利用 Lipschitz 连续条件：

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2|$$

我们可以将距离求取上述 W-distance 的下界，转换成求取下式的上界。

$$W(P_r, P_g) = \frac{1}{K} \sup_{||f||_L \leq K} \mathbb{E}_{x\sim P_r}[f(x)] - \mathbb{E}_{x\sim P_g}[f(x)] \quad \text{(公式13)}$$



- Weight clipping
  希望满足上述变换条件，需要判别器 D 网络拟合的映射f_w是 Lipschitz－1 连续，作者直接通过通过权重裁剪，直觉上保证f_w的梯度变化是可控的。
  实际操作，是在每次训练判别器 D 时，取 D 的参数做裁剪：

```
1.  # Clamp parameters to a range [-c, c], c= weight_clipping_limit
2.  for p in discriminator.parameters():
3.      p.data.clamp_(-weight_clipping_limit, weight_clipping_limit)
```

3) WGAN-GP

a) 网络结构

和 WGAN 没有大的区别

b) 训练过程

去掉了 weight clipping 这种粗糙的方式，通过对判别器网络梯度进行正则化的方式，保证了 WGAN 中提出的 Lipschitz－1 连续条件。

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.

while θ has not converged **do**
  for $t = 1, \ldots, n_{\text{critic}}$ **do**
    for $i = 1, \ldots, m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

- Loss 函数

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g} [D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r} [D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}} \left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2 \right]}_{\text{Our gradient penalty}}.$$

WGAN-GP 的作者认为 weight-clipping 会导致判别器训练困难，通过实验显示了通过 weight-clipping 手段会导致网络参数倾向于向 clipping limit 极值分布，权重分布不连续。

通过加入梯度惩罚项，能够有效避免权重分布不连续的现象，使得训练更加稳定。

想象现在需要将拟合分布平移到真实分布，使得二者的 W-distance 梯度变化尽可能平滑，为了模拟这种过程，最简单的实现就是在P_r, P_g中随机采样，在采样点x_r, x_g的连线均匀采样得到$\hat{x}$，要求f_w对于 $\hat{x}$的梯度变化不超过 1。

四、结果分析

1. 对不同优化器

在这里我们较为系统的比较不同优化器的优劣，以期在后期，面对不同的 loss 函数和网络特点选择有效的优化器和参数值。

深度学习中的优化器主要可以分成两类：

一种是基于动量的方式（SGDM，NAG 等）；一种是自适应化步长的方式（Adagrad，RMSProp 等），而 Adam 优化器则是二者的结合。

Image 2: SGD without momentum

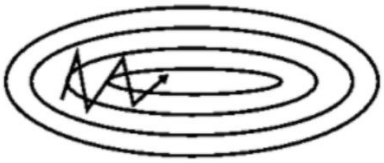


Image 3: SGD with momentum

a) 基于动量

由于 SGD 在上述曲面处很容易出现震荡的现象，原因在于梯度更新过于频繁，但在大多数的下降方向并不正确，梯度更新的效率很低。

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta).$$

$$\theta = \theta - v_t.$$

通过引入之前梯度的累积，可以有效的缓解梯度震荡的问题。下面是示意图。

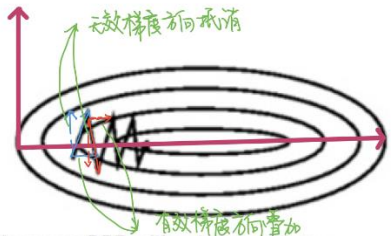


Image 2: SGD without momentum

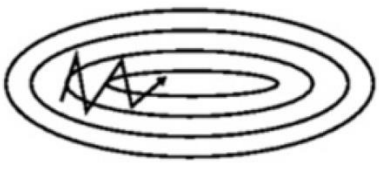


Image 3: SGD with momentum

总的来说：基于动量可以使得梯度方向不变的维度上更新变快，梯度快速变化的方向更新变慢，达到加速收敛和减小震荡的目的。

b) 基于自适应

自适应的出发点**对于频繁更新的梯度，用自适应的计算小步长更新梯度；而稀疏的梯度，那么更新的步长则相对比较大**。

而自适应的过程是通过 RMS（Root Mean Square）,显然，如果当前计算梯度大，则会导致分母中 r 变大，则更新步长减小；如果当前计算梯度小，则同理获得较大的更新步长

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers.
Initialize accumulation variables $\boldsymbol{r} = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho) \boldsymbol{g} \odot \boldsymbol{g}$
Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta+\boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
end while

c) Adam

Adam 实际上带有 momentum 的 Rmsprop 算法。

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

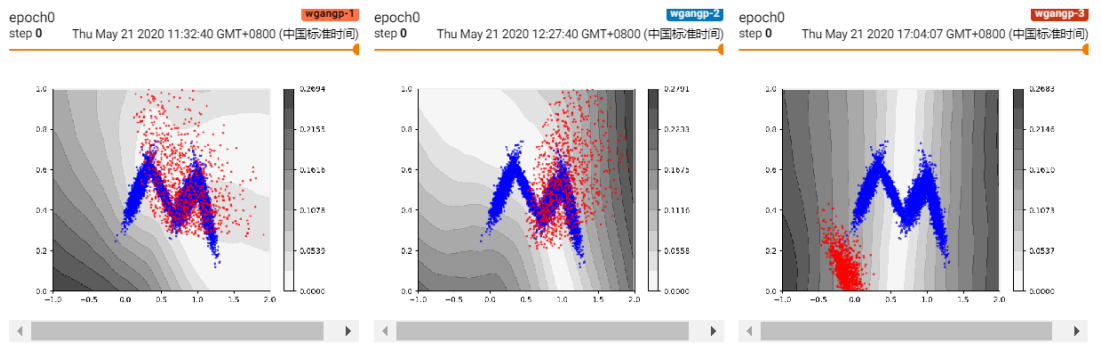
d) 适合 GAN 的优化器

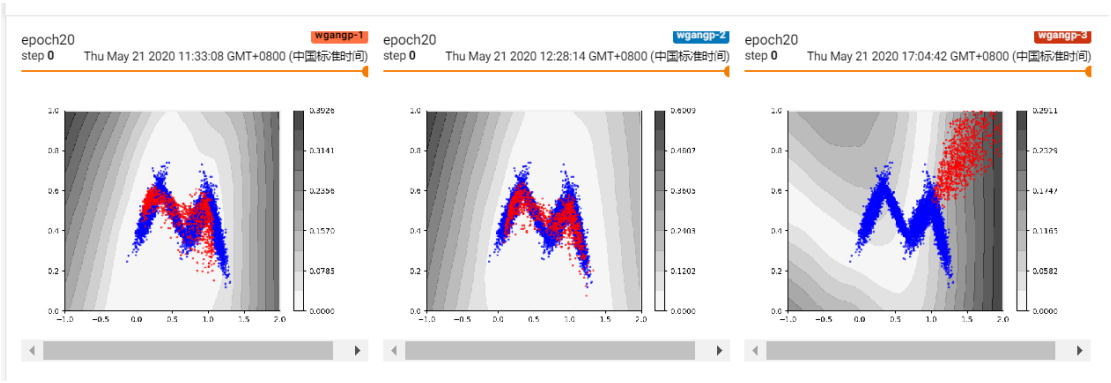
通过之前对 GAN 的分析，我们知道**原始 GAN 中的稀释梯度是普遍现象，而且梯度的变化会有突变的过程**。RMSprop 优化器能够获得较好的效果。

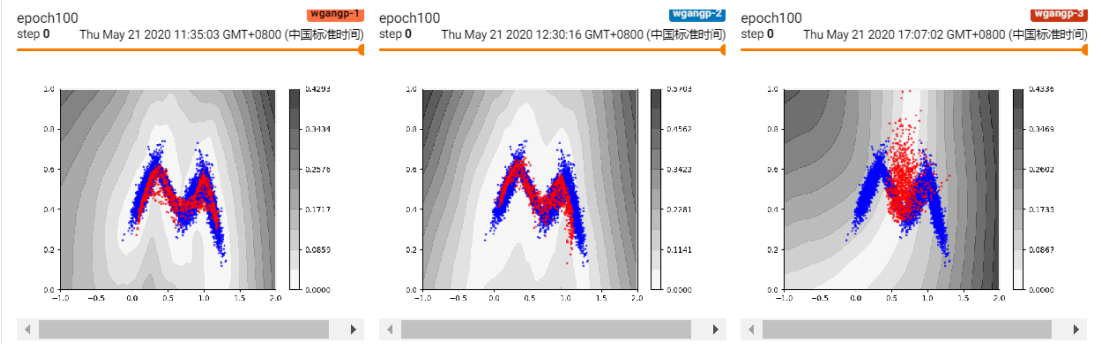
Adam 优化器需要将β_1的值尽可能调小，取$\beta_1 = 0.001, \beta_2 = 0.8$能够获得较好的效果。

而 SGD 和 SGDM 这一类的优化算法，在目前的参数尝试中无法收到较好的收敛效果。

不同优化器对 wgan-gp 收敛效果：（左起分别为 rmsprop\adma\sgd）







但是上述现象是和理论有点矛盾，因为在满足 Lipschitz – 1 条件下，梯度稀疏的现象应该能够得到缓解，可能还有哪里没有理解到位（mark 一下）。

2. 比较不同 gan 网络

参数设置：

```
params = {
"epoches" : 200,
"batch_size" : 500,
"lr" : 0.0002,
"hidden_dim" : 256,
"noise_dim" : 5,
"optim_type":"rmsprop",
"lamda":0.001
}
```

实验有以下发现：

a) 收敛程度：通过图示可以看出

$$wgan_{gp} > wgan > gan$$

b) 计算速度：

$$gan \approx wgan > wgan_{gp}$$

c) 稳定性：

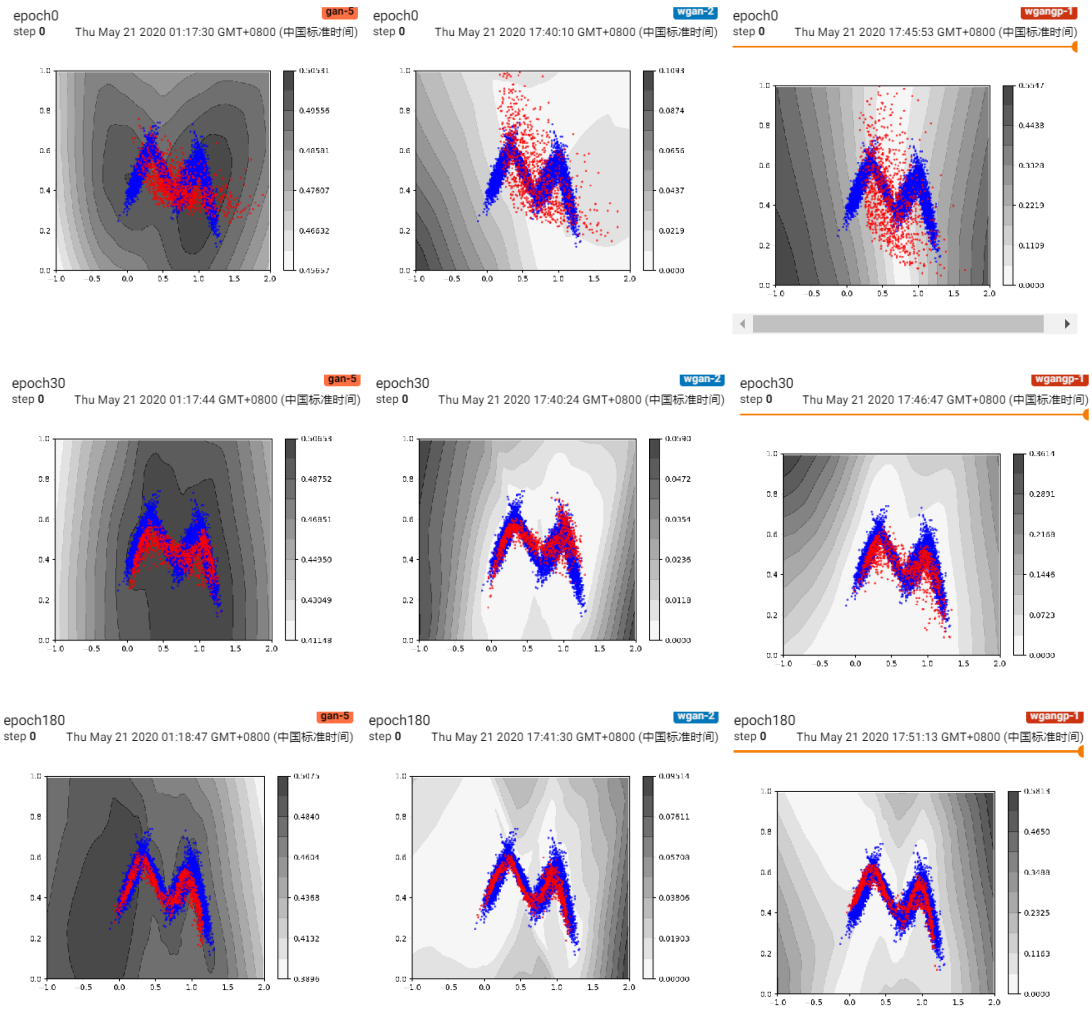
其实三种网络结构的稳定性不能很严格的比较，相对而言。对于拟合分布任务：

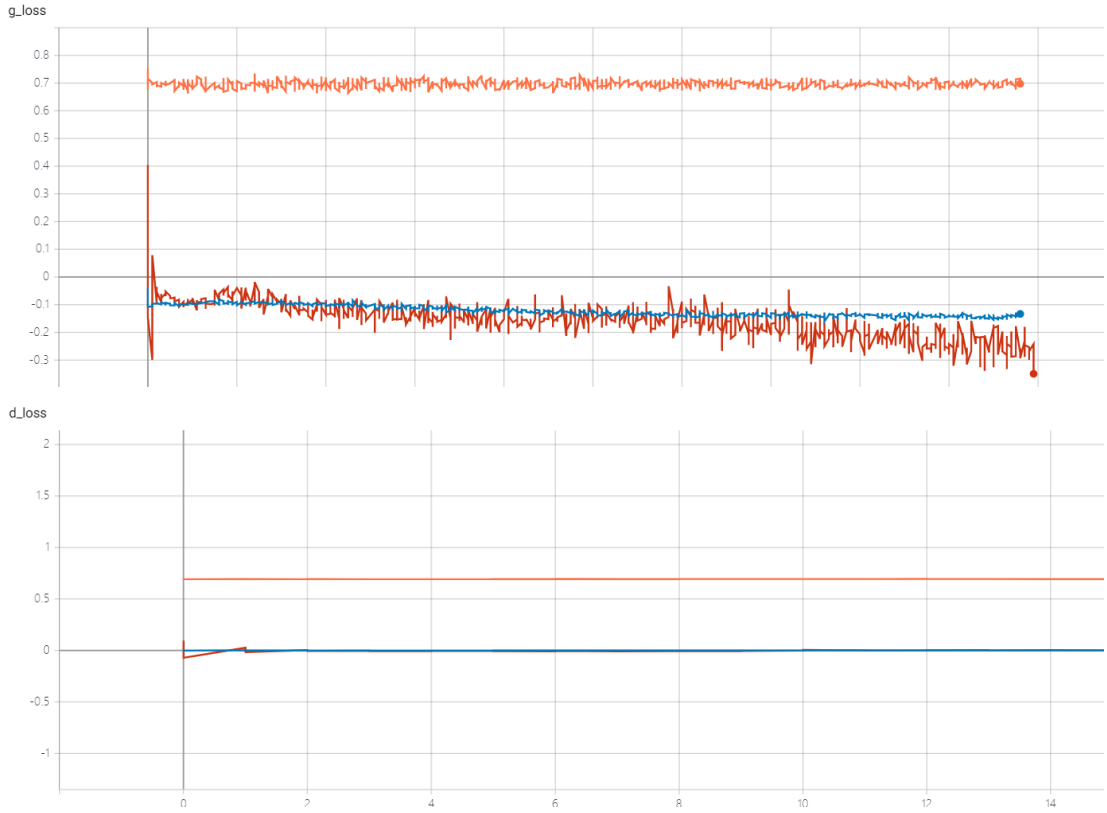
$$wgan_{gp} \approx wgan > gan$$

但是$wgan$ 受到 weight clipping 参数影响很大，比较合适的取值有 0.1,0.01，参数过小会导致模式坍塌。

同理$wgan_{gp}$收到超参数λ影响，理想参数为 0.001

d) 从下面的 loss 图像可以看出，wgan 和 wgan_gp 在训练判别器的时候还是出现了稀疏梯度的现象，Rmsprop 能够帮助其收敛。





五、总结

1. 详细推导理解了各种主流优化器的原理和设计方式，比较了结果的优劣，同时根据实验结果给出了适合训练 gan 的优化器。
2. 比较了 gan，wgan，wgan-gp 的设计原理和改进措施，比较了优劣。
3. 本次实验的任务较为简单，所以拟合效果判别主要考虑的是可视化。但是对于更复杂的生成任务，如何衡量生成的效果，如果设计感知损失函数，定量比较。
4. 可视化过程中如何绘画分界面：
   根据真实数据框出矩形区域，在区域中均匀采样，在可视化的时候，将采样的点送入判别器，进行预测。
   注意，gan 中判别器经过 sigmoid 输出，类似于概率形式。在 wgan 和 wgan-gp 中，我们直接采用距离作为衡量，即和真实分布的距离约接近 0，被判别器标记为正例的可能性越高，图像中注意区分区别（也用同学仍旧使用 sigmoid 处理 wgan 中判别器的输出，获得概率形式的输出。）

六、参考

【1】 WGAN 分析 https://zhuanlan.zhihu.com/p/25071913
【2】 WGAN-GP 分析 https://blog.csdn.net/xiaoxifei/article/details/87542317
【3】 李宏毅 GAN 课程 https://www.bilibili.com/video/BV1JE411g7XF?p=78、
【4】 GAN trick https://machinelearningmastery.com/how-to-train-stable-generative-adversarial-networks/
【5】 优化器对比 https://www.cnblogs.com/guoyaohua/p/8542554.html
【6】 An overview of gradient descent optimization algorithms