تمرین سری پنجم

یادگیری ژرف

امیرحسین محمدی

۹۹۲۰۱۰۸۱

مسئله‌ی ۱. (۱۰ نمره)

(آ) یکی از راه‌های معمول برای تخمین گرادیان یک امید ریاضی، استفاده از رابطه‌ی زیر است:

$$\nabla_\theta \mathbb{E}_{z\sim q_\theta(z)}\left[f\left(z\right)\right] \approx \frac{1}{N}\sum_{i=1}^{N} f\left(z^i\right).\nabla_\theta \ln q_\theta\left(z^i\right)$$

که در آن هر z^i نمونه‌ی مستقلی از توزیع $q_\theta\left(z\right)$ می باشد. درستی این رابطه را نشان دهید و بیان کنید که چطور می توانیم از آن در VAE استفاده کنیم. با مراجعه به این مقاله مشکلی که در استفاده از این روش وجود دارد را بیان کنید.

ما اغلب علاقه مندیم که کمیت و معیار های مختلف را بر روی متغیرهای تصادفی محاسبه کنیم. به عنوان نمونه می خواهیم میانگین reconstructions در VAE به ازای یک ورودی محسابه کنیم. روش Monte Carlo Integration به ما کمک می کند تا به جای محاسبات تحلیلی از نمونه گیری تصادفی استفاده کنیم. با این حال ممکن است یک نمونه تصادفی کافی نباشد بنابراین بسیاری از نمونه های تصادفی می توانند کمیت قابل اعتمادی را به ما ارائه دهند. به عنوان مثال، با پرتاب تاس بارها می‌توانیم هیستوگرام احتمالات را برای هر مجموع ممکن بدست آوریم. اگر یک بار تاس پرتاب کنیم، هیستوگرام بسیار اشتباه خواهد بود (فقط یک نوار). اما اگر صدها بار و به طور متوسط تکرار کنیم، می توانیم به مقدار واقعی بسیار نزدیک می شویم.

در روش Monte Carlo Integration ما ورودی ها را به عنوان متغیرهای تصادفی با PDF، p(x) در نظر می گیریم و در نتیجه احتمال y مطلوب ما خروجی است و انتگرال در سرتاسر همه ی x های ممکن:

$$y = \int_x f(x)p(x)dx$$

این انتگرال معادل یک امید ریاضی است:

$$y = \int_x f(x)p(x)dx = \mathbb{E}_{x\sim p(x)}[f(x)]$$

رابطه ی ۱

❖ می‌توانیم این امید ریاضی را با نمونه‌گیری تصادفی و جمع‌بندی تقریب کنیم:

$$y = \mathbb{E}_{x\sim p(x)}[f(x)] \approx \frac{1}{n}\sum_i f(x_i) = \hat{y}$$

به طوریکه x_i ، از توزیع $p(x)$ نمونه برداری شده است. در این رابطه $\hat{y}$ یک تخمین زیرا در واقع یک تقریب تخمینی از y است. اگر بخواهیم کمیتی y را محاسبه کنیم که بتوانیم آن را به عنوان انتگرال یک تابع در فضای احتمالی که دارای یک PDF، $p(x)$ شناخته شده و آسان برای نمونه برداری است بیان کنیم، بنابراین می‌توانیم محاسبات دقیق اما غیرقابل حل (intractable) را با یک تخمین‌گر مثل monte carlo، که tractable جایگزین کنیم.

$$y = \mathbb{E}_{x\sim p(x)}[f(x)] \approx \frac{1}{n}\sum_i f(x_i), x_i \sim p(x)$$ رابطه ی دو

بنابراین طبق قضیه ی Monte Carlo Integration برای محاسبه ی گرادیان امید ریاضی زیر داریم:

$\nabla_\theta \mathbb{E}_{z\sim q_\theta(z)}[f(z)]$ → طبق این رابطه ۱ داریم: $= \nabla_\theta \int_z f(z)q_\theta(z)dz = \int_z f(z)\nabla_\theta q_\theta(z)dz$ → همچنین می توانیم بنویسیم: $\nabla_\theta q_\theta(z) = q_\theta(z)\nabla_\theta lnq_\theta(z)$

$$= \nabla_\theta \int_z f(z)q_\theta(z)dz = \int_z f(z)\nabla_\theta q_\theta(z)dz = \int_z f(z)q_\theta(z)\nabla_\theta lnq_\theta(z)dz$$

$$\nabla_\theta \mathbb{E}_{z\sim q_\theta(z)}\left[f\left(z\right)\right] = \nabla_\theta \int_z f(z) q_\theta(z) dz = \int_z f(z) \nabla_\theta q_\theta(z) dz = \int_z f(z) q_\theta(z) \nabla_\theta ln q_\theta(z) dz$$

بنابراین با توجه به روابط بدست امده و رابطه ی ۲ داریم:

بنابراین با توجه به روابط بدست امده داریم:

$$\nabla_\theta E_{z\sim q_\theta(z)}\left[f(z)\right] = E_{z\sim q_\theta(z)}[f(z)\nabla_\theta \ln q_\Theta(z)] \Rightarrow \nabla_\theta E_{z\sim q_\theta(z)}\left[f(z)\right] \approx \frac{1}{N}\sum_{i=1}^{N} f(z^i).\nabla_\theta \ln q_\Theta(z^i)$$

VAEها مدل متغیر پنهان (latent) را به صورت مشخص می کنند:

$$z_n \sim \mathcal{N}\left(0, I_D\right)$$
$$x_n \sim p_X\left(f_\theta\left(z_n\right)\right)$$

که در آن f_θ تابعی است با پارامتر θ که z را به پارامترهای یک توزیع روی x نگاشت می کند. همانطور که می دانیم VAE از دو بخش encoder و Decoder تشکیل شده است. برای بخش decoder ما فرض کنیم مجموعه داده های ما یک مجموعه داده تصویری است. بیایید فرض کنیم پیکسل های تصاویر ما در D x_n با توزیع (p) برنولی است:

$$p\left(z_n\right) = \mathcal{N}\left(0, \boldsymbol{I}_D\right)$$
$$p\left(\boldsymbol{x}_n \mid \boldsymbol{z}_n\right) = \prod_{m=1}^{M} \operatorname{Bern}\left(\boldsymbol{x}_n^{(m)} \mid f_\theta(\boldsymbol{z}_n)_m\right)$$

به طوریکه $x_n^{(m)}$، m امین پیکسل از nامین تصویر از مجموعه ی D است و $f_\theta : \mathbb{R}^D \to [0,1]^M$ شبکه ی عصبی است که با پارامتر θ که میانگین توزیع های برنولی را برای هر پیکسل در x_n را در خروجی می دهد.

اکنون که مدل را تعریف کرده‌ایم، می‌توانیم یک عبارت برای احتمال log داده D در این مدل بنویسیم:

$$\begin{aligned}\log p(\mathcal{D}) &= \sum_{n=1}^{N} \log p\left(\boldsymbol{x}_n\right) \\ &= \sum_{n=1}^{N} \log \int p\left(\boldsymbol{x}_n \mid \boldsymbol{z}_n\right) p\left(\boldsymbol{z}_n\right) d\boldsymbol{z}_n \\ &= \sum_{n=1}^{N} \log \mathbb{E}_{p(z_n)}\left[p\left(\boldsymbol{x}_n \mid \boldsymbol{z}_n\right)\right]\end{aligned}$$

ارزیابی این بسیار هزینه بر است بنابراین می‌توانیم از Monte Carlo که در قسمت قبل به آن پرداختیم برای تقریب با رسم نمونه‌ها (بردارهای پنهان) z^l از $p(z_n)$ استفاده کنیم و عملا آموزش و به روز رسانی را برای ما تسهیل می کند:

$$\begin{aligned}\log p\left(x_n\right) &= \log \mathbb{E}_{p(z_n)}\left[p\left(x_n \mid z_n\right)\right] \\ &\approx \log \frac{1}{L} \sum_{l=1}^{L} p\left(x_n \mid z_n^{(l)}\right), \quad z_n^{(l)} \sim p\left(z_n\right)\end{aligned}$$

یکی از مشکلات استفاده از این روش این است که واریانس تقریب گرادیان، ممکن است بسیار زیاد باشد .با توجه به نمونه های N در رابطه ی زیر از یک بردار تصادفی X، کوواریانس میانگین نمونه unbiased آن X^- به عنوان $Cov(X^-) = Cov(X)/N$ شناخته می شود .هنگامی که مقادیر قطری Cov(X) بزرگ باشد، نمونه های زیادی مورد نیاز است تا این واریانس را برای تقریب امید ریاضی به زیر سطح دلخواه برسانند.

$$\nabla_\theta \mathbb{E}_{z\sim q_\theta(z)}\left[f\left(z\right)\right] \approx \frac{1}{N}\sum_{i=1}^{N} f\left(z^i\right).\nabla_\theta \ln q_\theta\left(z^i\right)$$

منبع

https://notesonai.com/Variational+Autoencoders
https://notesonai.com/Monte-Carlo+Estimation

(ب) به صورت شهودی بیان کنید که روش Reparameterization چگونه می‌تواند این مشکل را حل کند؟

❖ در VAE تابع هدفی که میخواهیم بهینه سازی کنمی به صورت زیر است:

$$E_{X\sim D}\left[E_{q_\phi(z|x)}[\log p(x \mid z)] - \mathcal{KL}[q(z|x)\|p(z)]\right]$$

برای آموزش مطمئن VAE، باید پارامترهای q را به‌روزرسانی کنیم تا z مناسب برای بازسازی p تولید کند، به این معنی که باید qرا با گرادیان $E_{q_\phi(z|x)}[\log p(x|z)]$ نیز به‌روزرسانی کنیم. مشکل اینجاست که ما در حال نمونه برداری از z_i هستیم که یک عملیات مشتق ناپذیر است و گرادیان ندارد. همچنین SGD می تواند ورودی های تصادفی را مدیریت کند، اما لایه های تصادفی را نه.

بنابراین monte carlo نیز در این حالت امکان پذیر نیست:

$$\nabla_\varphi \mathbb{E}_{z\sim q_\varphi(z|x)}\left[\log p_\theta(x \mid z)\right] = \int_z \nabla_\varphi \left[q_\varphi(z \mid x)\right] \log p_\theta(x \mid z) dz$$

همانطور که می بنیم ما هیچ density نداریم که بتوانیم از آن نمونه بردای کنیم. $\nabla_\varphi \left[q_\varphi(z \mid x)\right]$ گرادیان تابع چگالی و $\log p_\theta(x \mid z)$ لگاریتم تابع چگالی است.

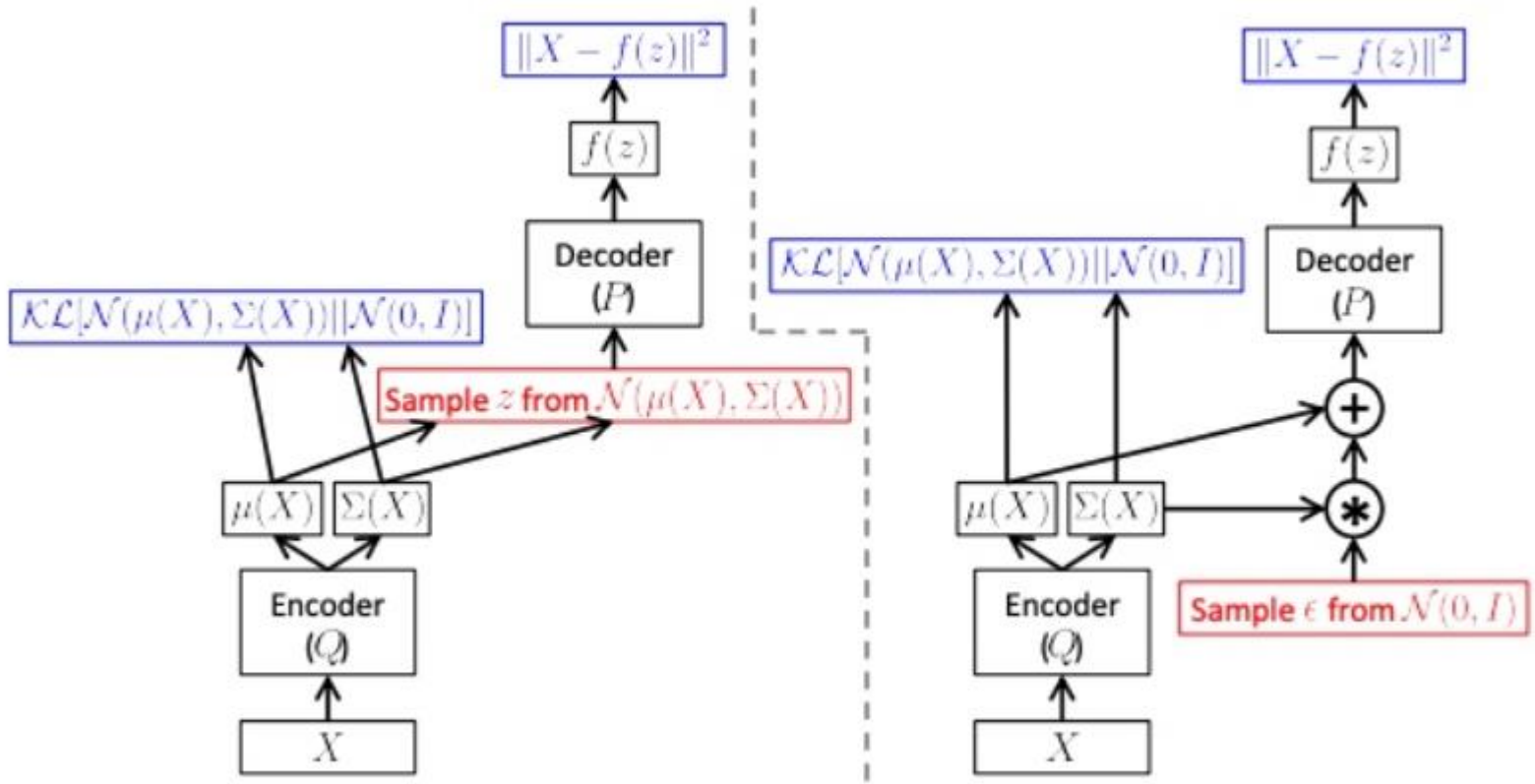
برای حل این مشکل، راه حل این است که از کی توزیع گووسی استفاده کنیم که به این روش Reparameterization Trick می گویند که بر این اساس است نمونه برداری به صورت مداوم انجام شود. سپس می توانیم گرادیان را به صورت زیر بازنویسی کنیم:

$$\nabla_{\boldsymbol{\varphi}} \mathbb{E}_{z \sim q_{\varphi}(z|x)} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \mathbf{z}) \right] = \nabla_{\varphi} \int_{\boldsymbol{Z}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \mathbf{z}) q_{\varphi}(z \mid x) d\boldsymbol{z}$$
$$= \nabla_{\varphi} \int_{\varepsilon} \log p_{\boldsymbol{\theta}}\left(x \mid \boldsymbol{\mu}_{z,\varphi}, \boldsymbol{\sigma}_{z,\varphi}, \varepsilon\right) q(\varepsilon) d\varepsilon$$
$$= \int_{\varepsilon} \nabla_{\varphi} \log p_{\boldsymbol{\theta}}\left(x \mid \boldsymbol{\mu}_{z,\varphi}, \boldsymbol{\sigma}_{z,\varphi}, \varepsilon\right) q(\varepsilon) d\varepsilon$$
$$\approx \sum_{k} \nabla_{\varphi} \log p_{\boldsymbol{\theta}}\left(x \mid \boldsymbol{\mu}_{z,\varphi}, \boldsymbol{\sigma}_{z,\varphi}, \varepsilon_{k}\right), \varepsilon_{k} \sim N(0,1)$$

بنابراین استفاده از این راه حل بدان معناست که نمونه برداری در انتگرال monte carlo دیگر به توزیع رمزگذار بستگی ندارد. نمونه‌برداری مستقیم از $\varepsilon \sim N(0,1)$ منجر به تخمین‌های واریانس پایین در مقایسه با نمونه‌برداری مستقیم از $z \sim N\left(\mu_Z, \sigma_Z\right)$ می‌شود، زیرا شبکه عصبی ما اکنون می‌داند که stocahsticity از یک منبع تصادفی بسیار خاصی از اپسیلون (**ε**) می‌آید. در واقع ما در حال نمونه‌گیری برای z هستیم، همچنین در حال نمونه‌برداری از گرادیان‌ها هستیم. توابع میانگین و stdخروجی های قطعی دارند، بنابراین می توانیم متغیر پنهان را به صورت فرموله کنیم:

$$z = \mu(X) + \Sigma^{1/2}(X) * \epsilon \quad \Rightarrow \quad \epsilon \sim \mathcal{N}(0, I)$$

با این تکنیک تصادفی بودن با شبکه عصبی و پارامترهای آن که باید یاد بگیریم مرتبط نیست .تصادفی بودن در عوض از **ε** خارجی می آید و گرادیان ها می توانند از طریق μ_Z و σ_Z جریان داشته باشند .این باعث می شود که گراف محاسباتی قطعی باشد و اجازه می دهد پس انتشار بدون هیچ مشکلی کار کند.



منبع

https://notesonai.com/Variational+Autoencoders
https://notesonai.com/Monte-Carlo+Estimation

(ج) در بسیاری از موارد تابع خطای رمزگشای VAE را خطای MSE در نظر میگیریم. این در حالی است که هدف ما بیشینه کردن تابع $\mathbb{E}_{z\sim q_\theta(z|x)} \ln p_\theta(x|z)$ می باشد. در چه صورتی و با چه فرض هایی این دو کار معادل یکدیگر هستند؟

❖ همانطور که می دانیم VAEیک توزیع را مدل می کند:

$$P(x) = \int P(x|z)P(z)dz$$

❖ وقتی مقدار خروجی پیوسته باشد، آنگاه یک پارامتر مناسب برای P(x|z) برابر است:

$$\mathcal{N}(\mu = f(z;\theta), \sigma^2)$$

❖ همانطور که گفته شد یک VAE با به حداکثر رساندن کران پایینی متغیر (variational lower bound) آموزش داده می شود که می تواند به دو عبارت تقسیم شود:

$$E_{z\sim q}[\log P(x|z)] - \mathrm{KL}(q(z)||p(z))$$

❖ اما اولین عبارت صرفاً لاگ چگالی گاوسی است که عبارت است از یک سری scale و ثابت هایی بنابراین داریم:

$$f^* = \arg\max_{f\in F} \mathbb{E}_{z\sim q_x^*}(\log p(x|z))$$
$$= \arg\max_{f\in F} \mathbb{E}_{z\sim q_x^*}\left(-\frac{||x - f(z)||^2}{2c}\right)$$

⇒ $(x-\mu)^2$ ⇒ MSE

منبع ⇒ https://stats.stackexchange.com/questions/409377/why-is-vae-reconstruction-loss-equal-to-mse-loss

مسئله‌ی ۲. (۱۵ نمره)

(آ) در یک VAE اگر داده ورودی از نوع باینری باشد (تصویری با پیکسل های ۰ و ۱)، می‌توان به جای توزیع گاوسی چندمتغیره روی خروجی کدگشا، از توزیع برنولی چندمتغیره استفاده کرد. توزیع خروجی کدگشا را به شکل برنولی چندمتغیره در نظر بگیرید و تابع فعال سازی آخرین لایه کدگشا را sigmoid در نظر بگیرید. اثبات کنید که در تابع هزینه این شبکه یک جمله ای به شکل Binary Cross Entropy ظاهر می شود.

ابتدا متغیرهای پنهان (latent) را به جای گسسته پیوسته فرض می کنیم، به طوری که $z \in R^k$ در آن K یک ابرپارامتر است که ابعاد فضای پنهان را نشان می دهد. توزیع prior را به صورت یک توزیع گاوسی با میانگین صفر و کوواریانس واحد در نظر می گیریم.

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$$

با توجه به مشاهده x، تقریب postrior، $p(z|x,\theta)$ یک توزیع گاوسی با کوواریانس قطری خواهد بود:

$$q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_e, \text{diag}(\boldsymbol{\sigma}_e))$$

به طوریکه:

$$\boldsymbol{\mu}_e = f_{\phi_\mu}(\mathbf{x})$$
$$\log(\boldsymbol{\sigma}_e^2) = f_{\phi_\sigma}(\mathbf{x})$$

به طوریکه e به بخش encoder اشاره دارد $f_{\emptyset_\sigma}$ و $f_{\emptyset_\mu}$ دو شبکه ی عصبی هستند با وزنهای $\emptyset_\sigma$ و $\emptyset_\mu$. این پارامترها، پارامترهای encoder هستند. همینطور داریم $\emptyset = \{\emptyset\mu, \emptyset\sigma\}$. بنابراین با توجه به روش reparameterization داریم:

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$
$$\mathbf{z} = \boldsymbol{\mu}_e + \boldsymbol{\sigma}_e \odot \boldsymbol{\epsilon}$$

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$
$$\mathbf{z} = \boldsymbol{\mu}_e + \boldsymbol{\sigma}_e \odot \boldsymbol{\epsilon}$$

که در آن ⊙ نشان دهنده ضرب element-wise است. برای بخش prior و تقریب postrior که تعریف شده است، KL divergence به صورت زیر است:

$$\mathrm{KL}(q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi}) \| p(\mathbf{z}|\boldsymbol{\theta})) = \frac{1}{2} \sum_{i=1}^{K} (\sigma_{ei}^2 + \mu_{ei}^2 - \log(\sigma_{ei}^2) - 1)$$

از آنجایی که پیکسل ها در تصاویر باینری هستند، یک مشاهده $x \in R^D$ را با توجه به متغیر پنهان (latent) z، به عنوان یک متغیر تصادفی چند متغیره برنولی با میانگین μd مدل می کنیم. این مربوط به decoder است:

$$p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) = \prod_{i=1}^{D} \mu_{di}^{x_i} (1 - \mu_{di})^{(1-x_i)}$$

همینطور داریم:

$$\boldsymbol{\mu}_d = f_\theta(\mathbf{z})$$

که در آن fθ یک شبکه عصبی با وزن θ است. از آنجایی که خروجی decoder توزیعی را روی یک توزیع برنولی چند متغیره مدل می کند، باید اطمینان حاصل کنیم که مقادیر آن در 0 و 1 قرار دارند .این کار را طبق فرضیات مسئله با یک لایه سیگموئید در خروجی انجام می دهیم:

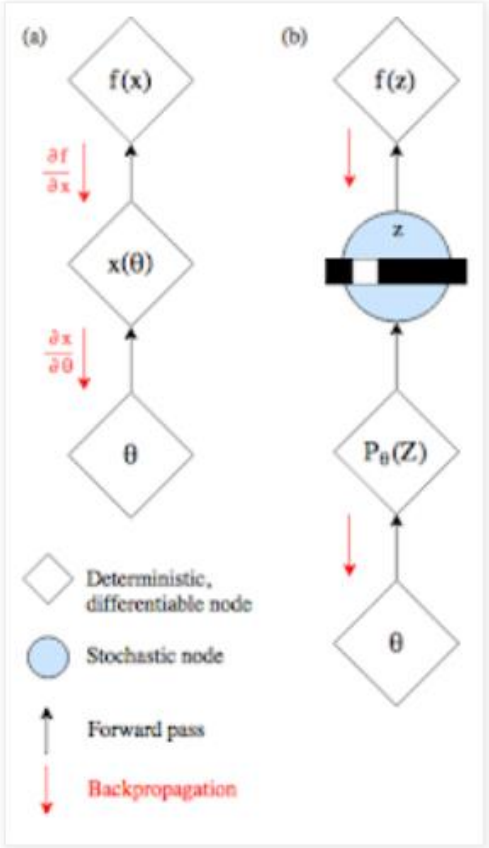
$$\log p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) = \sum_{i=1}^{D} x_i \log \mu_{di} + (1 - x_i) \log(1 - \mu_{di})$$

→ binary cross-entropy

مبنع → https://dfdazac.github.io/01-vae.html

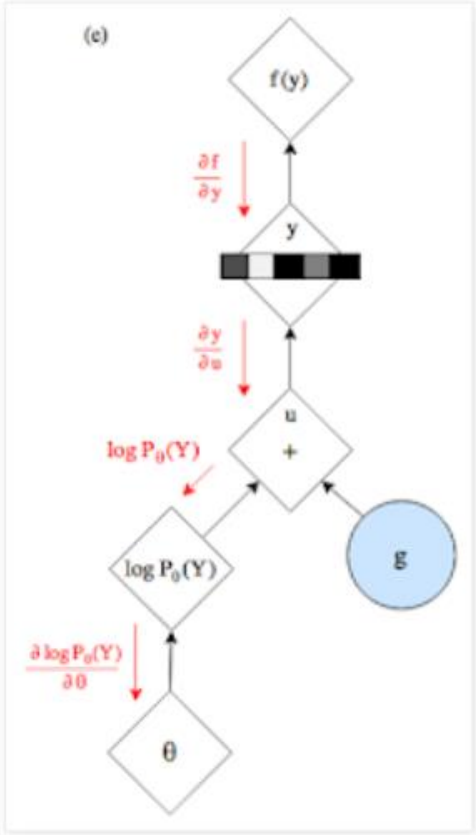
(ب) تکنیک Reparameterization روی بسیاری از توزیع‌های پیوسته قابل اعمال است. تحقیق کنید که چگونه می‌توان از این تکنیک برای یک توزیع categorical استفاده کرد؟

همانطور که می دانیم ما می‌توانیم از شبکه‌های عصبی تصادفی استفاده کنیم، که در آن هر لایه پارامترهای یک توزیع (گسسته) را محاسبه می‌کند و مرحله feed forward آن شامل گرفتن نمونه از آن توزیع پارامتری است .با این حال، مشکل این است که ما نمی‌توانیم از طریق نمونه‌ها عملیات back prob را انجام دهیم. همانطور که در زیر نشان داده شده است، یک گره تصادفی (دایره آبی در شکل زیر) بین f(z) و θوجود دارد.



شکل سمت چپ :در شبکه های عصبی پیوسته، می توان از backprop برای محاسبه گرادیان پارامتر استفاده کرد شکل سمت راست :عملیات back prob از طریق گره های تصادفی امکان پذیر نیست.

برای حل این مشکل ایده ای مطرح شد که اگر بتوانیم نمونه $z \sim p\theta(z)$ را دوباره به نوعی دیگر تعریف کنیم، می‌توان مشکل back prob را از طریق گره‌های تصادفی دور زد، به طوری که گرادیان‌ها می‌توانند از $f(z)$ به θ بدون مواجهه با گره‌های تصادفی جریان پیدا کنند .برای مثال، نمونه‌هایی از توزیع نرمال $z \sim N(\mu,\sigma)$ را می‌توان به صورت $z=\mu+\sigma \cdot \epsilon$ بازنویسی کرد، جایی که $\epsilon \sim N(0,1)$. که همانطور که در قسمت های قبل گفته شد به این روش reparameterization trick می گویند. معمولاً برای آموزش variational autoencoder با متغیرهای latent گاوسی استفاده می شود.



تکنیک reparameterization برای توزیع های پیوسته مناسب است. برا استفاده از تکنیک reparameterization در توزیع های توزیع **categorical** روش Gumbel-Softmax ارائه شده است. ایت روش در واقع این در واقع یک ترفند پارامترسازی مجدد برای توزیعی است که می‌توانیم به آرامی آن را به توزیع طبقه‌ای تغییر شکل دهیم. ما از ترفند Gumbel-Max استفاده می‌کنیم، که روشی کارآمد برای ترسیم نمونه‌های z از توزیع دسته‌بندی با احتمالات کلاس πi است:

$$z = \texttt{one\_hot}\left(\arg\max_i \left[g_i + \log \pi_i\right]\right)$$

argmax مشتق پذیر نیست، بنابراین ما به سادگی از تابع softmax به عنوان یک تقریب پیوسته از argmax استفاده می کنیم:

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau)} \qquad \text{for } i = 1, \ldots, k.$$

از این رو، ما آن را توزیع Gumbel-SoftMax می نامیم. τ یک پارامتر دما است که به ما اجازه می‌دهد تا میزان تقریب نمونه‌های توزیع Gumbel-Softmax را با نمونه‌های توزیع categorical کنترل کنیم. اگر $\tau \to 0$ ، softmax تبدیل به argmax و توزیع Gumbel-Softmax تبدیل به توزیع طبقه‌بندی می‌شود .در طول مرحله ی آموزش، به $\tau > 0$ اجازه می دهیم تا گرادیان ها از نمونه عبور کنند، سپس به تدریج τرا کاهش می دهیم اما نه به طور کامل تا 0، زیرا گرادیان ها منفجر می شوند.

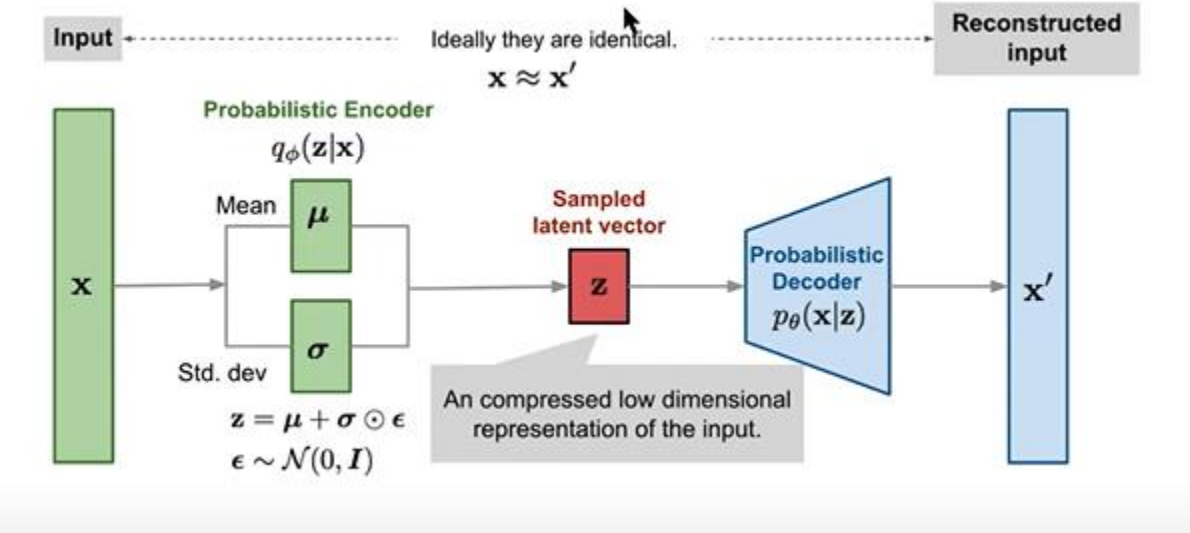
مبنع

https://blog.evjang.com/2016/11/tutorial-categorical-variational.html

(ج) یکی از نسخه‌های تغییریافته VAE مقاله مربوط به beta-VAE می باشد. در این روش یک ضریب beta ای پشت یکی از توابع هزینه قرار میگیرد. درباره این روش تحقیق کنید و بیان کنید:

۱) با انجام چه روندی از محاسبات، این ضریب در تابع هدف این روش ظاهر می‌شود؟

همانطور که گفته شده روش beta-va یک روش مبتنی بر VAE است که به دنبال تحقق ایده ای تحت عنوان disentanglement است. disentanglement ه این معنی است که مدل generative بتواند source of variation بین داده های مشاهده شده را ایزوله کند. به این معنی که مثلا اگر ما در داده های آموزش یک سیب بزرگ قرمز داریم مدل generative بتواند اجزایی مثل سایز (بزرگ)، رنگ (قرمز)، نوع شی (سیب) را ایزوله کند. این ایده سبب می شود که ما بتوانیم نمونه هایی تولید کنیم که در مجموعه ی داده های آموزشی وجود ندارد. برای مثلا م بتوانیم سیب کوچک سیاه تولید کنیم از یک مدل Generative که بر اساس داده هایی مثل سیب بزرگ قرمز و انگور سیاه کوچک آموز دیده است. در واقع ایده disentanglement به ما کمک می کند تا بدانیم هر بخش از فضای latent مربوط به چه بخشی از ویژگی های داده های ورودی است. برای ایجاد همچین ایده ای از VAE استفاده می شود زیرا همانطور که می دانیم این روش برخلاف مدل های Generative دیگر مثل GAN به خوبی لایه ی Latent را یاد می گیرد.



بنابریان به دنبال این هستیم که چگونه می توانیم مطمئن شویم که لایه ی latent درون VAE، disentanglement است و می توانیم این لایه را کنترل کنیم و نمونه های جدید تولید کنیم.

بنابراین به دنبال این هستیم که چگونه می توانیم مطمئن شویم که لایه ی latent درون VAE، disentanglement است و می توانیم این لایه را کنترل کنیم و نمونه های جدید تولید کنیم. برای تحقق این امر روش beta-VAE که بر اساس روش VAE است، معرفی شد. همانطور که می دانیم تابع هزینه ی روش VAE به صورت زیر تعریف می شود:

$$L_{\text{VAE}} = -\log p_\theta(\mathbf{x}) + D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x}))$$

که این تابع هینه شامل دو بخش است که بخش اول log likelihood را برای تولید دادهایی بر اساس داده های آموزشی ماکسیمم می کند و بخش دوم KL divergence بین approximate posterior و true posterior را مینیمم می کند.

❖ تابع هدف VAE را می توانیم به صورت زیر بازنویسی کنیم :

$$\max_{\phi,\theta} \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z})]$$
$$\text{subject to } D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) < \delta$$

که در این رابطه ما به دنبال این هستیم که log likelihood برای تولید x از z را ماکسیمم کنیم با هدف اینکه approximate posterior و prior در نظر گفته شده برای z، مقدار kl divegence بین آنها تا جایی که می شود کوچک شود. برای قرار دادن شرط کوچک بود، KL را کوچکتر یک مقدار ثابت مثبت مثل Delta قرار می دهیم. بنابراین بر اساس این رابطه احتمال تولید داده ی واقعی ماکسیمم می شود در حالی که فاضله بین توزیع real و approximate posterior کوچک می شود. (در یک ثابت کوچک delat).

$$\max_{\phi,\theta} \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z})]$$
$$\text{subject to } D_{\mathsf{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) < \delta$$

بنابراین این تابع هدف را بر اساس یک ضریب لاگرانژ مثل beta و شرایط KKT (شبیه کاری که در SVM) انجام میدادیم بازنویسی کنیم به صورت زیر:

$$\begin{aligned}
&\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - \beta(D_{\mathsf{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) - \delta) \\
=&\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - \beta D_{\mathsf{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) + \beta\delta \\
\geq&\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - \beta D_{\mathsf{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) \qquad \text{since } \beta, \delta \geq 0
\end{aligned}$$

بنابراین رابطه ی بالا log likelihood را ماکسیمم می کند و این در حالی است که محدودیت تعریف شد ّبر روی KL نیز در نظر می گیرد. حال این تابع هدف بر اساس ماکسیمم نوشته شده است و وقتی می خواهیم آن را مینیمم کنیم داریم:

$$L_{\mathsf{BETA}}(\phi, \beta) = -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) + \beta D_{\mathsf{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}))$$

این تابع هزینه متعلق به Beta-VAE است که اگر Beta=1 باشد دقیقا می شود تابع هزینه ی VAE.

۲) هذف از افزودن ضریب beta چیست و اضافه کردن آن چه تاثیری روی ویژگی‌های فضای نهان یادگرفته‌شده توسط مدل دارد؟

$$L_{\text{BETA}}(\phi, \beta) = -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) + \beta D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}))$$

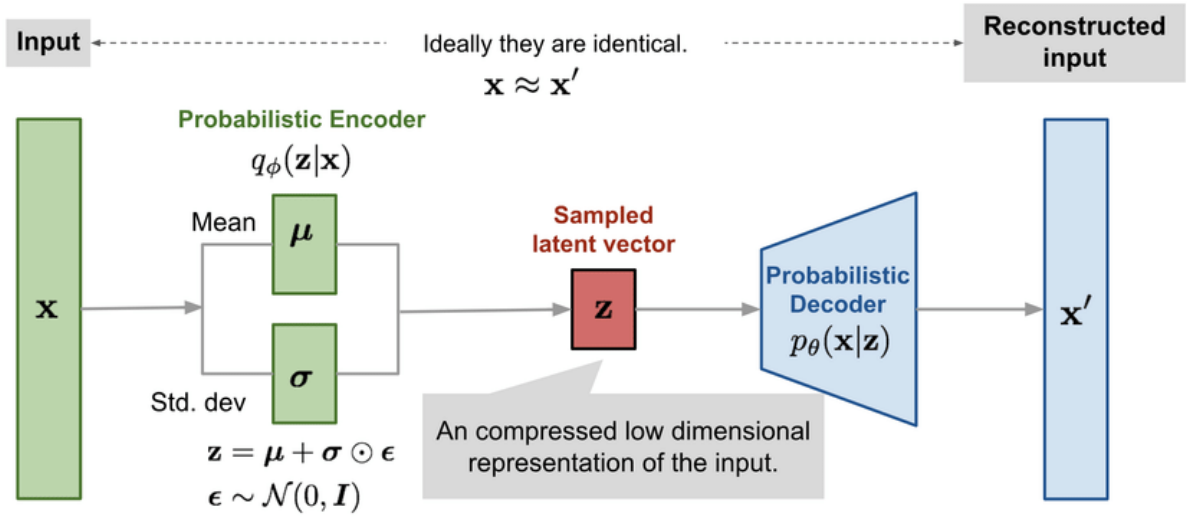
بنابراین با توجه زمانی که beta بزرگ تر از یک باشد به مسئله ی disentanglement در لایه ی latent و در مدل generative مد نظر، بیش توجه کند و قوی تر شود. بنابراین بخش اول این تابع هزینه به بهبود توانایی بازنمایی بخش decoder توجه می کند و بخش دوم به آموزش لایه ی latent در VAE می پردازد که با در نظر گرفتن وزن های بزرگتر سعی می کنیم که لایه یlatent بهتر آموز ش ببیند و مسئله ی disentanglement محقق شود. با این حال ممکن است در نظر گرفتن disentanglement سبب شود که نتوانیم به خوبی داده ها را از z بازنمایی کنیم بنابریان یک trade-off بین reconstruction و disentanglement رخ می دهد.

منبع https://www.youtube.com/watch?v=hq6V5i3Xrgk&t=734s

مسئله‌ی ۳. (۱۰ نمره)

(آ) همانطور که می‌دانیم، دو مدل GAN و VAE از مهم ترین و شناخته شده ترین مدل‌های generative در یادگیری عمیق می‌باشند. یکی از مهم‌ترین کاربردهای آن‌ها، تولید تصاویر و data augmentation می‌باشد. در این صورت، با فرض استفاده از مجموعه داده یکسان و فرآیند آموزش نسبتا کامل، آیا به طور کلی کیفیت تصاویر تولید شده توسط یکی از این مدل‌ها بر دیگری برتری دارد؟ لطفا پاسخ خود را با دلیل و در صورت نیاز اثبات ریاضی تشریح نمایید. می‌توانید از این مقاله استفاده نمایید.

همانطور که می دانیم برای تولید یک نمونه ی جدید از داده های ورودی مدل های Generative معرفی شدند. این مدل ها بر این اساس بودند که سعی بر این داشتند تا با یادگیری فضای مسئله و داده های ورودی، نمونه هایی همچون داده های ورودی تولید کنند. یکی از مهمترین مدل های generative مدل های Auto encoder بودند که سعی بر این داشتن تا با بردن ویژگی های ورودی به ابعاد کمتر و سپس بازنمایی مجدد آن ویژگی های فضای حالت داده های ورودی را یاد بگیرند. اما این روش خیلی کار وقاع نشد و صرفا یاد می گرفت آنچه در ورودی دیده را بازنمای کند و نه چیز دیگری و عملا هیچ اطلاعی از لایه ی latent به ما نمی داد (اینکه چه توزیعی دارد و...) بنابراین عملا ایجاد نمونه های جدید و گوناگون در این مدل های میسر نبود. مدل های بعدی که برای حل این مشکل ارائه شد VAE نام داشت این مدل بر این اساس بود که از دو بخش تشکب شده بود (encoder, decoder) با این تفاوت که سعی می کرد که بر اساس داده های ورودی توزیع این داده را یادبگیرد و سپس از توزیع آن ها نمونه برداری کند و به لایه ی latent ببرد و سپس از با استفاده از لایه ی latent و نمونه برداری از آن توزیع داده های ورودی را بازنمایی کند. این ایده بسیار مفید واقع شد و عملا ما می توانستیم کاملا در مورد توزیع داده های مسئله آگاه باشیم و نمونه های جدید و گوناگون از مجموعه داده های ورودی ایجاد کنیم. ایجاد نمونه های که در دنیا نمونه های واقعی آنها وجود ندارد از ویژگی های این شبکه است زیرا عملا ما با یک توزیع مواجه هستیم و می توانیم با تغییر پارامترهای آن داده های جدید تولید کنیم. همانطور که معماری آن را در شکل زیر می بینم:



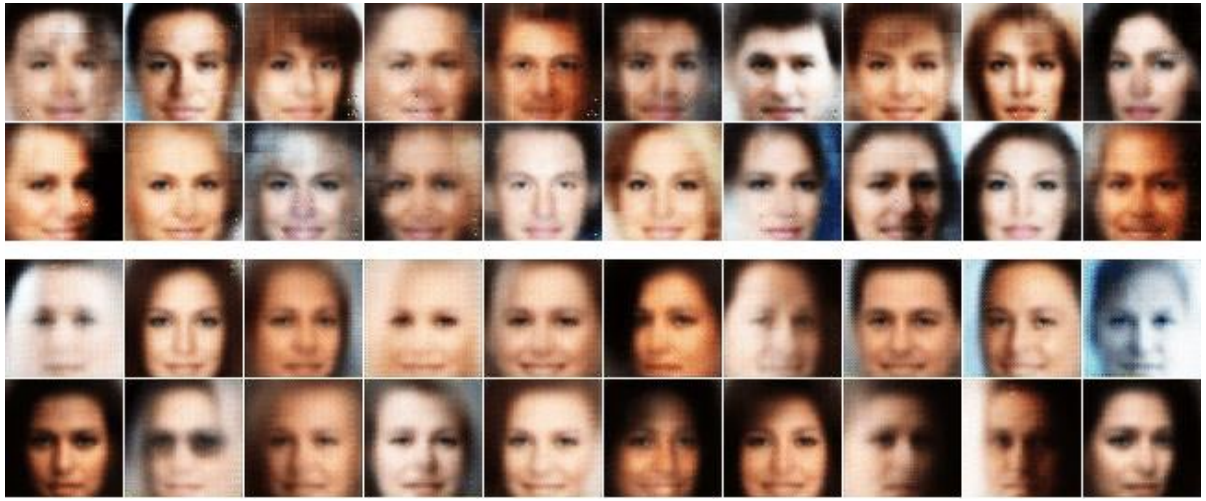
❖ که تابع هزینه ی آن به صورت زیر تعریف می شود:

$$\mathcal{L}(q, \boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_q[\log p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) - \log q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})]$$
$$= \mathbb{E}_q[\log p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta})] - \mathrm{KL}(q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})||p(\mathbf{z}|\boldsymbol{\theta}))$$

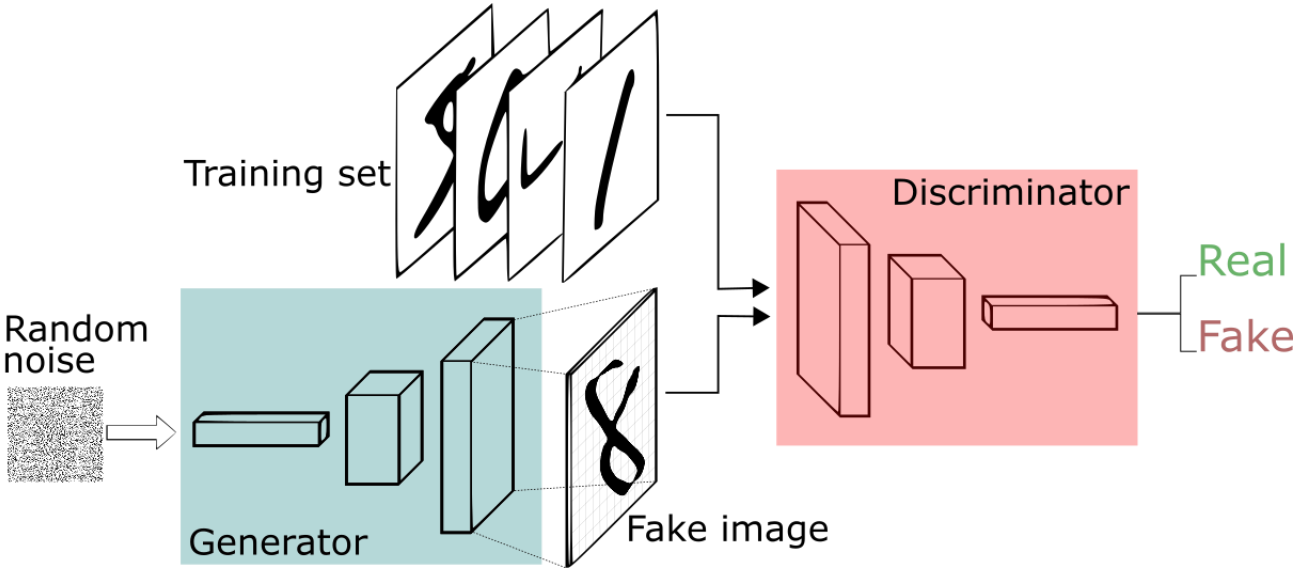
decoder

encoder

اما نکته ای که در این مدل از مدل های generative و تابع هزینه ی آنها وجود دارد این است که این مدل ها نمی توانند نمونه های جدیدی که تولید می کنند را به خوبی ارزیابی کنند. در واقع کیفیت خروجی VAE به خوبی ارزیابی نمی شود در واقع ما نمونه های جدید تولید می کنیم مثل تصاویر اما کیفیت آنها به خوبی در این شبکه ها مورد بررسی قرار نمی گیرد (واحد ارزیاب ندارند) و تصاویر تار هستند و رزولوشن خیلی بالایی ندارند و فقط به دنبال کم کردن تابع هزینه هستند و به کیفیت تصاویر خروجی توجهی ندارد.



یکی دیگر از مدل های generative، GAN نام دارد، این مدل بر این اساس شکل گرفت که از یک لایه ی latent بتواند خروجی high resolution تولید کند. معماری GAN ها از دو بش تشکیل شده است که یکی وظیفه ی تولید نمونه های جدید را دارد (Generator) و دیگری وظیفه ی ارزیابی آن را دارد(discriminator). در این معماری Generator سعی می کند تصاویر تولید که شبیه به تصاویر واقعی باشد و discriminator را فریب دهد که نتواند تشخیص دهد آن تصویر از داده های واقعی است یا داده های تولید شده توسط generator. از سوی دیگر discriminator به دنبال این است که بتواند خود را قدرتمند کند و داده های واقعی را از داه های تولید شده (fake) تشخیص دهد. از این رو پیوسته هر چقدر از روند آموزش می گذرد به دلیل اینکه discriminator قوی تر می شود، generator هم مجبور می شود تصاویر با کیفیت بالاتری تولید کند که بتواند به Discriminator مقابله کند.



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

یکی از مسائلی که در GAN وجود دارد این است که ما توزیع لایه ی Latent را نمی توانیم شناسایی کنیم و صرفا این لایه ی در مرحله ی آموزش بهبود پیدا می کند.

بنابراین یکی از مزیت هایی که GAN نسبت به مدل های دیگر generative دارد این است با تعداد اندکی داده های آموزش می توانند نمونه های با کیفیت و بسیار قابل قبول تولید کنند به طوری که مثلا اگر بخواهیم تصاویر جدید از یک مجموعه داده ی آموزش تولید کنیم GAN به شدت در این مسئله موفق ظاهر شده است و تصاویر high resolution تولید می کند که نمونه های آن را در زیر می بینیم:

همانطور که می بینیم تصاویر تولید شده توسط GAN از کیفیت بسیار قابل قبولی برخوردار هستند و مشکلاتی که در VAE وجود داشت از جمله تاری در خروجی های GAN وجود ندارد و این مسئله به خاطر معماری خصمانه GAN است که generator مجبور می شود تصاویر بسیار با کیفیتی برای فریب دادن Discriminator تولید کند.

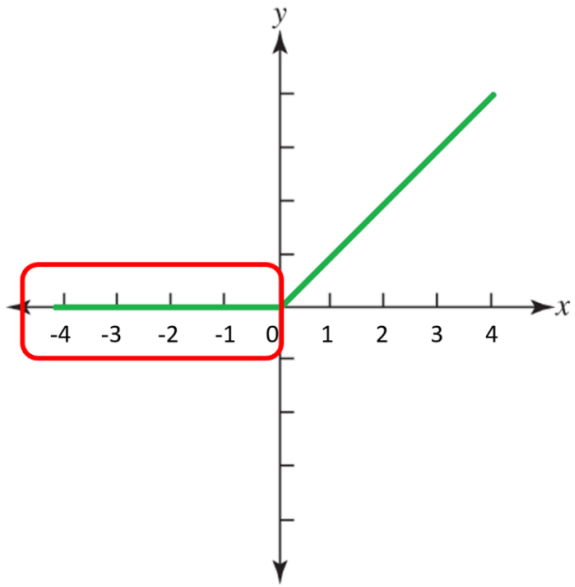
منبع

https://medium.com/@prakashpandey9/deep-generative-models-e0f149995b7c

1706.01807.pdf (arxiv.org)

(ب) تابع ReLU یکی از پرکاربردترین توابع فعالسازی مورد استفاده در شبکه های یادگیری عمیق می‌باشد، اما در برخی کاربری‌های خاص همانند بخش Generative در روش GAN می‌تواند منجر به ایجاد مشکل در فرآیند آموزش شود، به عبارت دیگر شبکه مولد ما آموزش نخواهد دید. در این مورد استثنا، توصیه می‌شود به جای ReLU ، از Leaky ReLU به عنوان تابع فعالسازی استفاده گردد. لطفا علت بروز مشکل به هنگام استفاده از ReLU را به طور کامل توضیح داده و تشریح کنید که به چه علت استفاده از Leaky ReLU می‌تواند مشکل را حل کند.

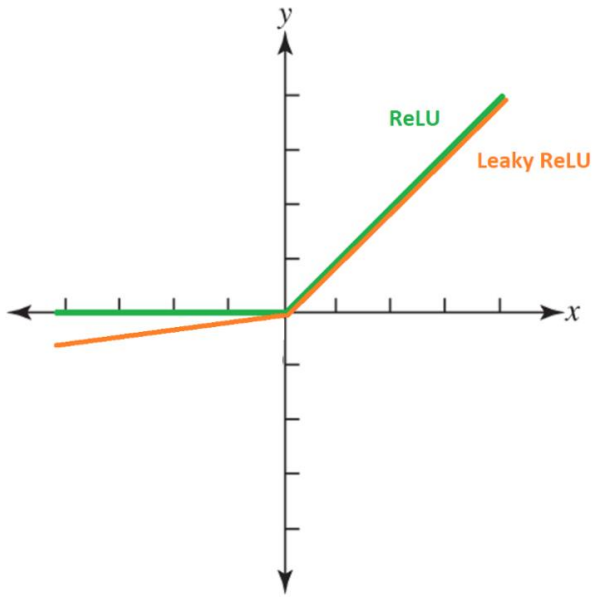
تابع فعال سازی ReLU فقط حداکثر بین مقدار ورودی و صفر را می گیرد .اگر از تابع فعال‌سازی ReLUاستفاده کنیم، گاهی اوقات شبکه در حالتی به نام حالت مرگ (dying) گیر می‌کند، و این به این دلیل است که شبکه چیزی جز صفر برای همه خروجی‌ها تولید نمی‌کند. این به این دلیل که بسیاری از مقادیری که وارد تابع فعالیت می شود منفی هستند.



- High learning rate ❖
- Large negative bias ❖

از جمله دلایل وقع مشکل Dying هستند.

Leaky ReLU با اجازه دادن به برخی از مقادیر منفی از این حالت مرگ (Dying) جلوگیری می کند .کل ایده روش های مولد مثل GAN، این است که Generator مقادیر گرادیان ازDiscriminator، دریافت کند و اگر شبکه در وضعیت در حال مرگ گیر کرده باشد، فرآیند یادگیری اتفاق نمی افتد.



خروجی تابع فعال‌سازی Leaky ReLUدر صورت مثبت بودن ورودی مثبت و اگر ورودی منفی باشد، یک مقدار منفی کنترل‌شده خواهد بود .مقدار منفی کنترل توسط پارامتری به نام آلفا است که با اجازه دادن به برخی از مقادیر منفی، تلورانس شبکه را معرفی می کند.

منبع

https://towardsdatascience.com/the-dying-relu-problem-clearly-explained-42d0c54e0d24#4995

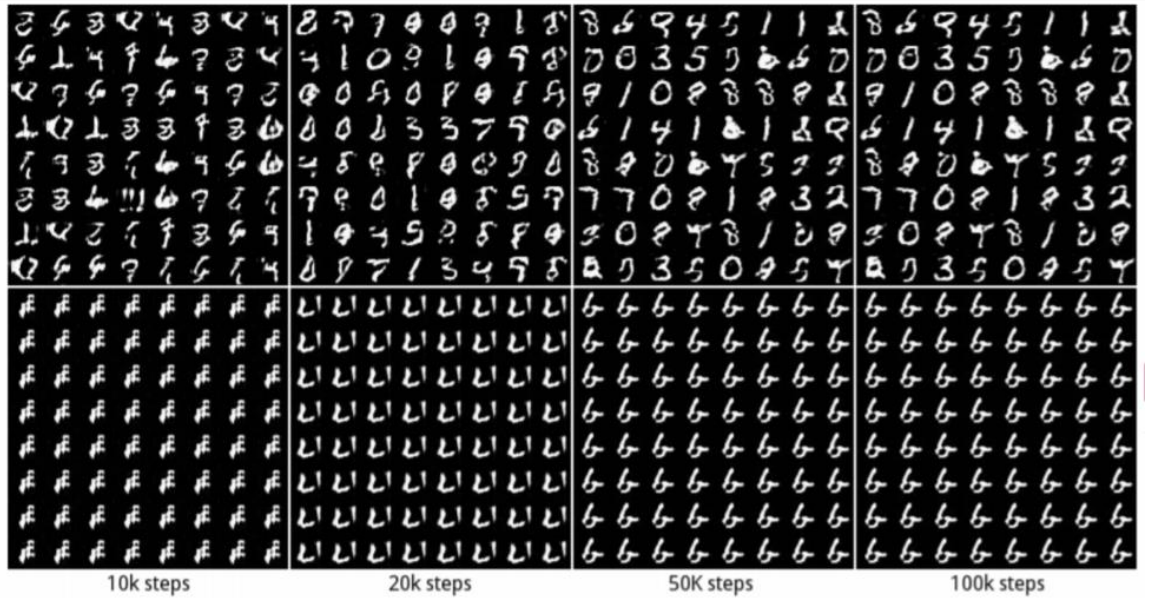
مسئله‌ی ۴. (۱۵ نمره)

(آ) یکی از مشکلات شایع در شبکه های GAN مشکل Mode Collapse می باشد که باعث می شود شبکه GAN به یک حالت عدم آموزش رسیده و به طور متوالی خروجی های یکسان تولید کند. این مشکل را به طور کامل تشریح کرده و راه حل های احتمالی به منجر به غلبه بر این مشکل می شوند را بیان نمایید.

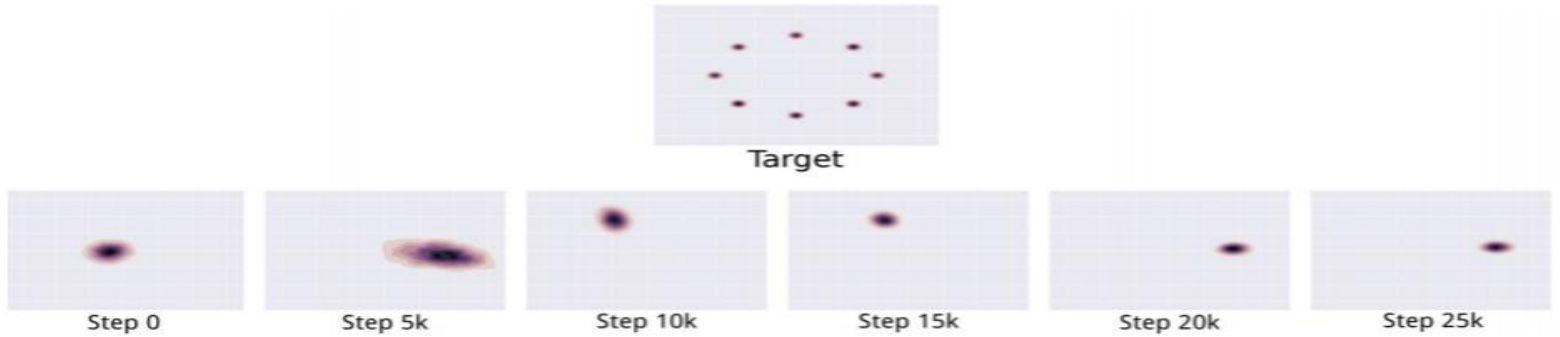
در حوزه ی یادگیری ماشین گاهی لیبل های مجموعه دادگان ما بیش از یک و دو لیبل است، مثلا در مجموعه‌ی دادگان MNIST مثلا در دیتاست ارقام ۱۰ دسته (۱۰ لیبل) وجود دارد. این مسئله زمانی بیشتر جلوه می کند که به طور مثال ما بر روی تصاویر انسان ها کار میکنیم از نظر پراکندگی و گوناکونی زیادی داریم که هر دسته و مجموعه از یک توزیع پیروی می کند. بنابراین ما در یک دیتا ست چند دسته ای مثل مجموعه ی ارقام، با فضای ابعادی (مسئله ای) مواجه هستیم که از چند توزیع با قله های مختلف تشکیل شده است و از تونع زیادی نسبتا برخوردار هستند. تصور ما این است که وقتی از GAN استفاده می کنیم شبکه ی GAN تمامی این توزیع ها را آموزش ببیند و در خروجی به ازای هر دسته ما نمونه ای را آموزش دیده باشد و آن را Generate کند. بنابراین ما توقع داریم که در خروجی مجموعه ی متنوعی از مجموعه دادگان مسئله تولید شود. اما در عمل این اتفاق نمی افتد و در خروجی ما با یک از زیر مجموعه ی کوچکی از فضای مسئله مواجه هستیم. همانطور که در شکل زیر می بینیم:

توقع ما از شبکه GAN

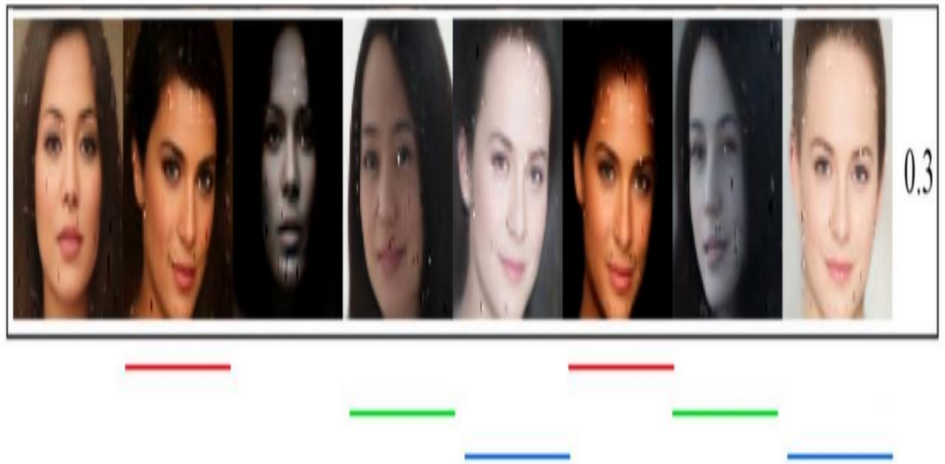
چیزی که در عمل اتفاق افتاده است. و فقط در گام های مختلف (100K) عدد ۶ تولید شده است



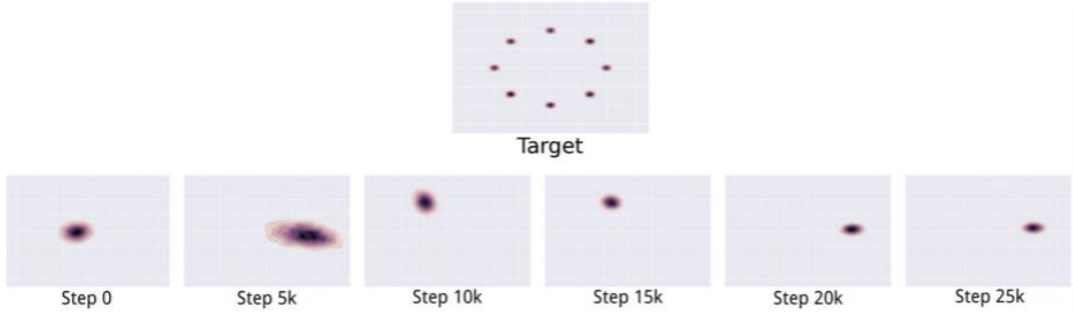
دلیل به وجود آمدن پدیده ی Mode collapse به این دلیل است که اگر Generator قدرتمند باشد خیلی سریع آموزش می بیند در یک MODE (همانطور که از شکل زیر مشخص است) و داده هایی تولید می کند که به راحتی می تواند Discriminator را فریب دهد (زیرا مولد در یک گام آموزش می بیند و تمیز دهند در یک گام دیگر). بنابراین پس از آن بر به صورت تدریجی Discriminator آموزش می بیند که این MODE را به عنوان داده های fake دسته بندی کند. بنابراین زمانی که Discriminator در تشخیص داده های fake به اندازه ی کافی قدرتمند می شود، Generator یک ضعف دیگر در Discriminator پیدا می کند و به یک MODE جدید Collapse می کند و تا زمانی که مجددا Discriminator تنظیم شود و بتواند این MODE را به عنوان fake پیش بینی کند. بنابراین مجددا دوباره Generator به حالت اولیه بر می گردد و همین پروسه مجددا تکرار می شود. همان طور که گفته شد وقع complete collapse خیلی نادر است اما partial collapse که در بخش از توزیع رخ می دهد متداول است.



البته باید اشاره داشت که این مشکل همیشه مانند مثال قبل نیست که فقط یک نمونه به طور کلی در گام مختلف آموزش یاد گرفته شود و تولید شود (complete collapse)و این مسئله کمتر متداول است. این مسئله معمولا به صورت نسبی اتفاق می افتد و مثلا یک بخش از فضای مسئله آموزش می بیند و آن بخش تولید می شود (partial collapse). همانطور که در شکل زیر می بینید تصاویر که رنگ زیرین آنها یکسان است تقریبا از یک توزیع هستند



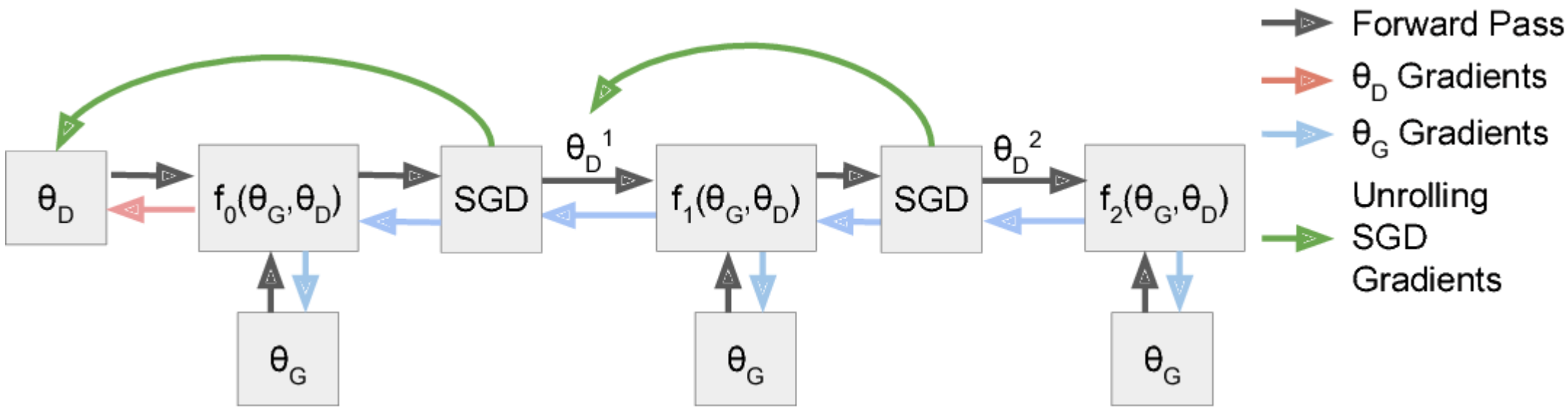
به این مشکل به وجود آمده در GAN اصطلاحا mode collapse می گویند. در واقع همانطور که گفته شد mode collapse زمانی رخ می دهد که که توزیع های متنوعی در فضای مسئله داشته باشیم(که این مسئله مثلا به خاطر افزایش تعداد لیبل های زیاد مسئله ناشی می شود). همانطور که در شکل زیر می بینید.



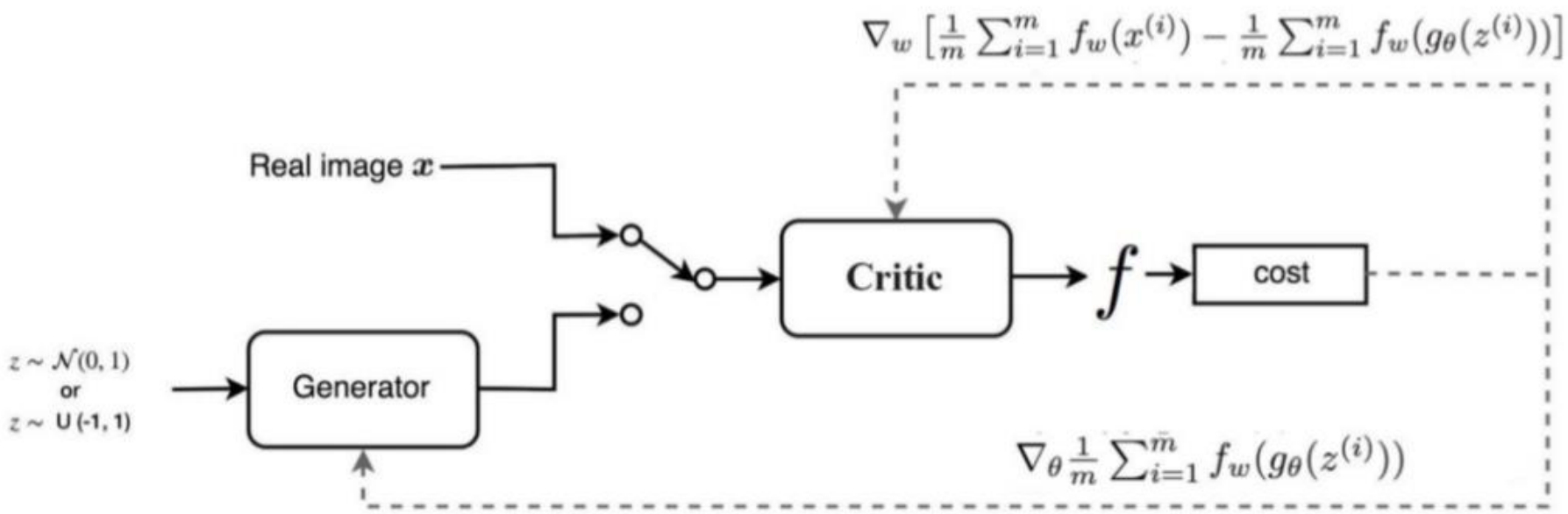
روش های گوناگونی برای مواجه با مشکل Mode collapse ارائه شده است که از مهم ترین های آنها می توان به موراد زیر اشاره کرد:

- Unrolled GAN
- Wasserstein GAN
- Ada GAN
- VEEGAN
- اضافه کردن نویز

از مهم ترین روش های گفته شده می توان به Unrolled GAN اشاره کرد که بر این اساس است که از یک تابع هزینه برای Generator استفاده می کند که علاوه بر دسته بندی های فعلی discriminator، خروجی‌ها و ورژن های بعدی discriminator را نیز در نظر می‌گیرد. بنابراین مولد نمی تواند برای یک discriminator بیش از حد بهینه سازی کند.



از دیگر روش های مهم برای مواجه با پدیده ی Mode Collapse می توان به اشاره کرد Wasserstein GAN که تابع هزینه بر این اساس است که این امکان را فراهم می کند که discriminator بدون نگرانی در مورد ناپدید شدن گرادیان‌ها، بهینه‌سازی شود، و این مسئله Mode Collapse را کاهش می‌دهد. در واقع اگر discriminator در حداقل های محلی گیر نکند، یاد می گیرد که خروجی هایی را که Generator روی آنها ثابت می شود (Collapse) را رد کند. بنابراین Generator باید Mode های جدیدی را امتحان کند.



منبع

https://kunrenzhilu.medium.com/mode-collapse-and-wgans-e3e6e809a6c4

(ب) معماری کلی و تابع هزینه مدل W-GAN را تشریح نمایید و تفاوت های آن با مدل پایه GAN را توضیح دهید. آیا تابع هزینه این مدل کمکی به برطرف شدن مشکل Mode Collapse خواهد کرد؟ توضیح دهید.

همانطور که که گفته شد معماری اولیه ی GAN از اشکالاتی رنج می برد که یکی از مهم ترین های آن ها می توان به:

- **Vanishing Gradients**
- **Mode Collapse**
- **Failure to Converge**

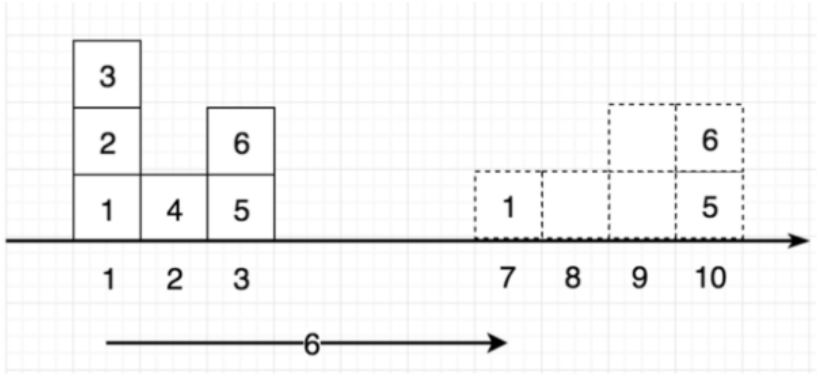
این مشکلات عمدتا به دلیل تابع هزینه ی معماری اولیه ی GAN ارائه شد که به شکل زیر می باشد.

$$\min_G \max_D V(D,G)$$

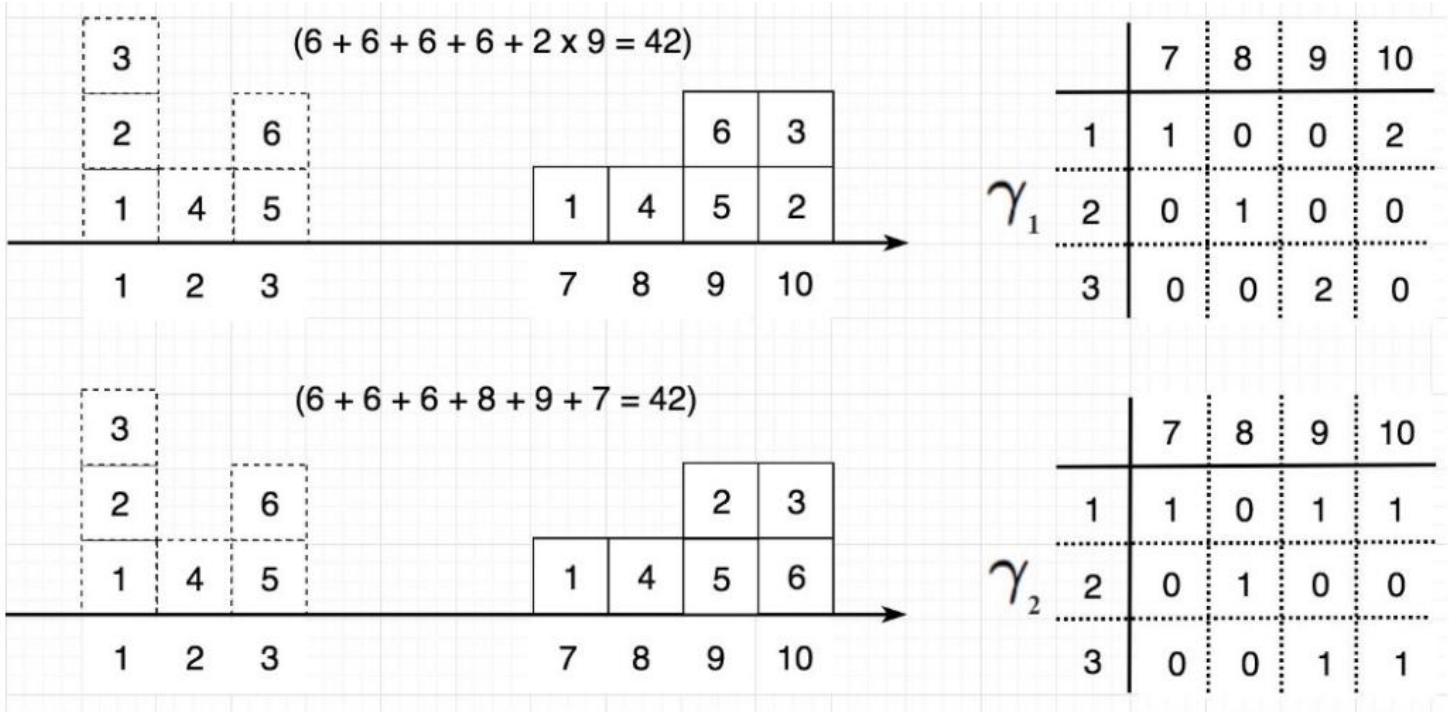
$$V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z))]$$

یکی از راه حل هایی که برای حل مشکلات در معماری اولیه GAN ارائه شد معماری Wasserstein GAN بود. این معماری بر اساس یک معیار به نام Earth-Mover (EM) distance ایجاد شده است.

فرض کنید ما ۶ جعبه دریافت داریم و می خواهیم آنها را از سمت چپ به مکان هایی که با مربع نقطه چین در سمت راست مشخص شده است منتقل کنیم. برای جعبه شماره ۱، آن را از مکان ۱ به مکان ۷ منتقل می کنیم. هزینه جابجایی برابر است با وزن آن ضربدر فاصله. برای سادگی، وزن را ۱ تنظیم می کنیم. بنابراین هزینه جابجایی جعبه شماره ۱ برابر است با ۶ (۱-۷).



شکل زیر دو روش متفاوت برای تحرک ۷ را نشان می دهد .جداول سمت راست نحوه جابجایی جعبه ها را نشان می دهد. به عنوان مثال، در شکل اول، ما 2 جعبه را از مکان 1 به مکان 10 منتقل می کنیم و مقدار $\gamma(1,10)$ برابر دو تنظیم می شود. کل هزینه جابه جایی هر شکل زیر 42 است.



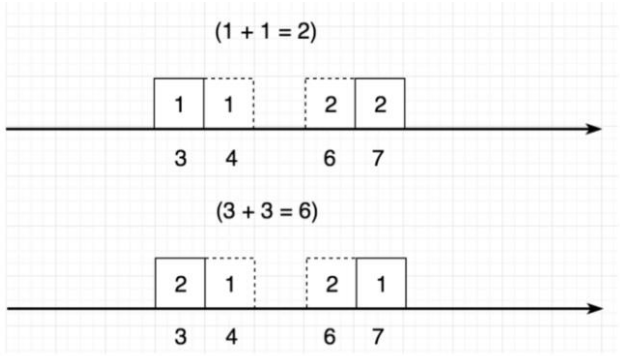
γ_1

	7	8	9	10
1	1	0	0	2
2	0	1	0	0
3	0	0	2	0

γ_2

	7	8	9	10
1	1	0	1	1
2	0	1	0	0
3	0	0	1	1

با این حال، همه حالت های جابه جایی هزینه یکسانی ندارند. از بین این حالت ها Earth-Mover (EM) distance یا Wasserstein Distance حالتی است گه کمترین میزان هزینه ی جابه جایی را دارد.



حال این معیار کمترین فاصله(هزینه) یا همان Wasserstein Distance چه کمکی در واقع به ما می کند. ما در GAN می خواهیم از یک توزیع مخفی (نویزی) گه هیچ اطلاعاتی در مورد آن نداریم، نمونه هایی با کیفیت و نزدیک به توزیع داده های آموزش و اصلی تولید کنیم. اما مشکلی که در این روش وجد دارد این است که ما در واقع هیچ کدام از توزیع های داده های واقعی و داده های تولید شده را نمی شناسیم. بنابریان حتی نمی دانیم این توزیع ها آیا از هم فاصله دارد یا نه و اصلا چقدر فاصله دارند. بنابراین ما باید به نوعی این فاصله و تفاوت دو توزیع را مدل کنیم. این مسئله فاصله ی بین داده های اصلی p_r و داده های تولید شده p_g به نوعی در معماری اولیه GAN نیز در نظر گرفته شده بود به طوری که از معیار های ***KL-Divergence*** و ***JS-Divergence*** استفاده می شد که رابطه ی آنها به صورت زیر است:

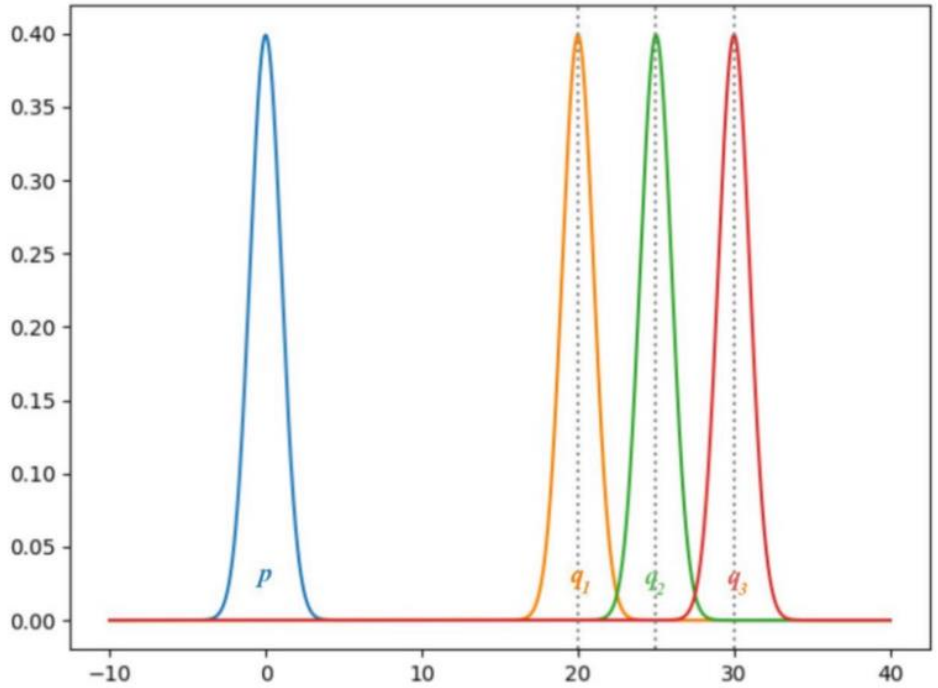
$$D_{KL}(P||Q) = \sum_{x=1}^{N} P(x) \log \frac{P(x)}{Q(x)}$$

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q||\frac{p+q}{2})$$

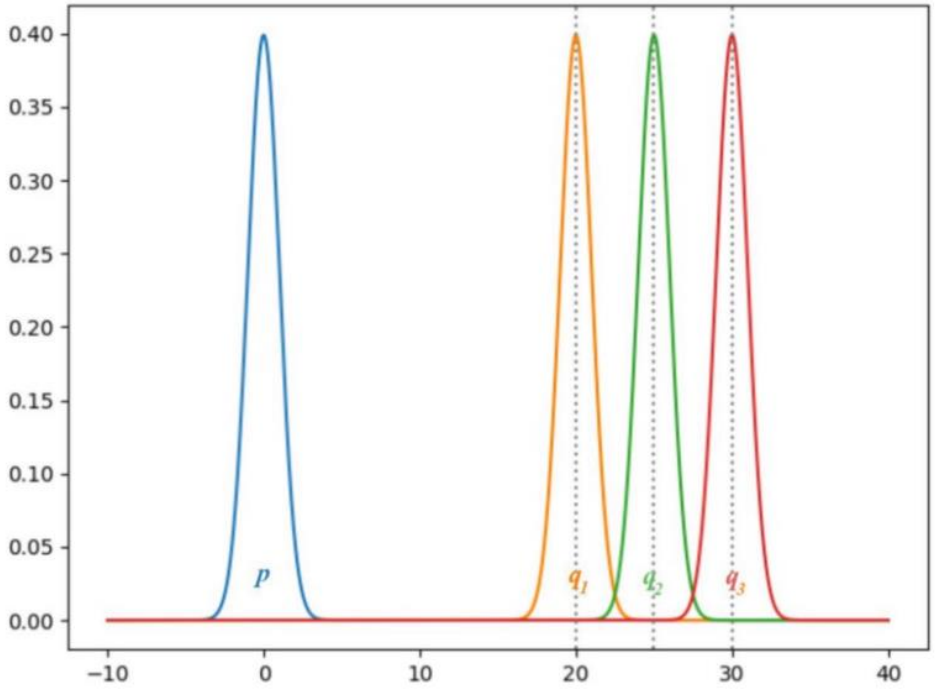
$$D_{KL}(P||Q) = \sum_{x=1}^{N} P(x) \log \frac{P(x)}{Q(x)}$$

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q||\frac{p+q}{2})$$

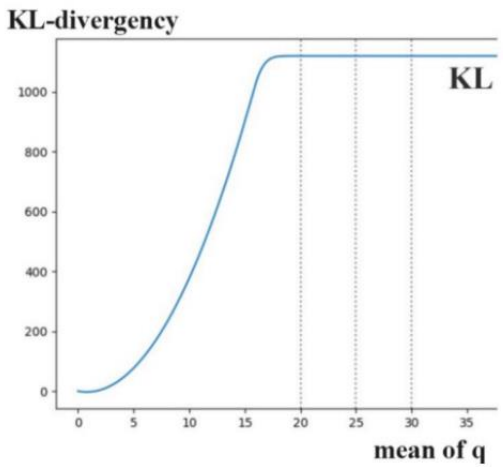
یبق این رابطه p توزیع داده های واقعی در نظر گرفته می شوند و q توزیع داده های تولید شده در نظر گرفته می شوند. اگر فرض کنیم که این داده های از توزیع گووسی پیروی می کنند همانند شکل زیر خواهیم داشت:

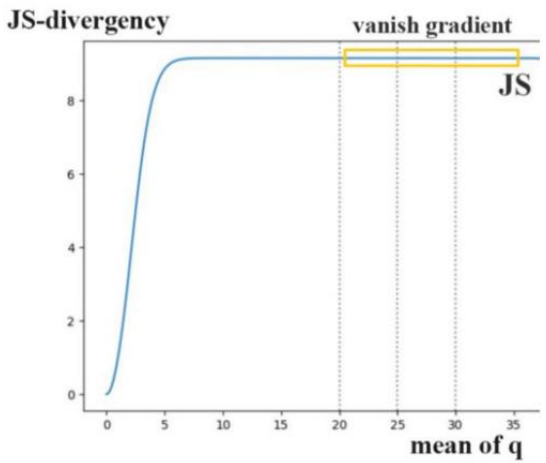


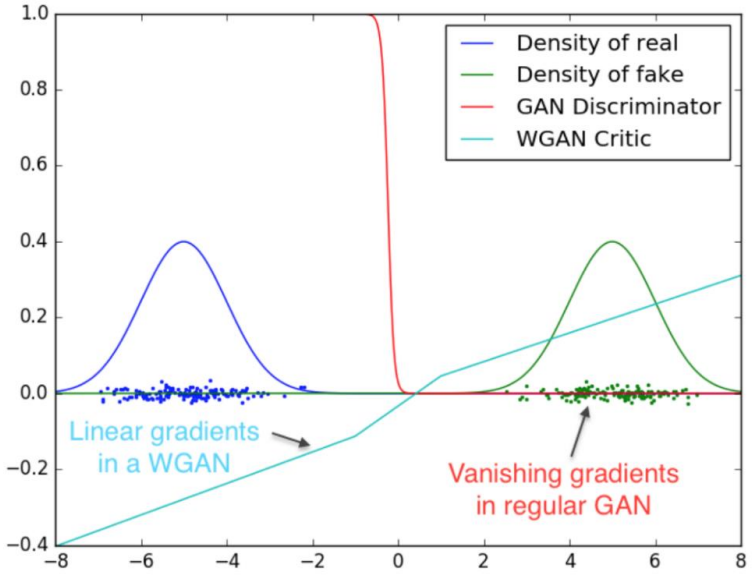
همانطور که در شکل مقابل می بینیم توزیع p که نمایندی داده های اصلی هست با توزیع q1,q2,q3 که نماینده ی داده های تولید شده هستند با میانگین های مختلف هستند فاصله دارد. اگر معیار واگرایی kl و js را برای دو توزیع p و q رسم کنیم مانند شکل زیر داریم.



زمانی که هر دو p و q یکسان هستند، واگرایی ۰ است. با افزایش میانگین q، واگرایی افزایش می یابد. . شیب واگرایی در نهایت کاهش می یابد. ما یک گرادیان نزدیک به صفر داریم، یعنی Generator از نزول گرادیان چیزی یاد نمی گیرد و عملا مشکل Vanishing رخ می دهد



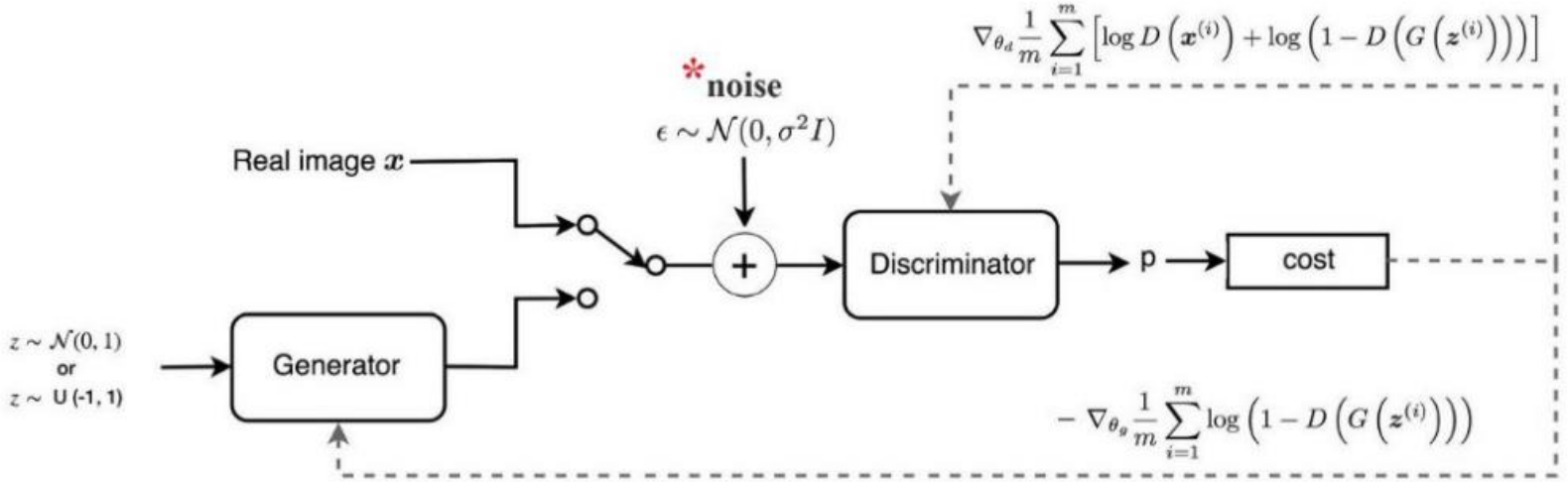




بنابراین همانور که در شکل مقابل می بینیم زمانی که دو توزیع از هم فاصله دارند تابع هزینه معماری GAN اولیه همانند شکل مقابل (نمودار قرمز رنگ) است و عملا بر اساس معیارهایی فاصله ای که در نظر گرفتیم گردایان مناسبی به generator نمی رسید و عملا پیوسته discriminator قوی تر می شود و generator ضعیف باقی می ماند. برای حل این مشکل جایگزین هایی هم ارائه شدن هم چون:

$$\nabla_{\theta_g} \log D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)$$

که این روش هم برای گردایان های با واریانس بزرگ مناسب نیست. یا همچنین روش هایی دیگر همچون اضافه کردن نویز:



با توجه به مشکلاتی که در معماری اولیه ی GAN وجود داشت، سعی شد تا با استفاده از معیار فاصله ی Wasserstein Distance به نوعی تابع هزینه ی GAN را تغییر دهند. معیار Wasserstein Distance به صورت زیر تعریف می شود:

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

probability distribution of x and y

all plans with $\sum_y \gamma(x, y) = p(x)$ & $\sum_x \gamma(x, y) = p(y)$

فاصله Wassersteinحداقل هزینه انتقال جرم در تبدیل توزیع داده qبه توزیع داده p است. فاصله Wasserstein برای توزیع داده های واقعی Pr و توزیع داده‌های تولید شده Pg از نظر ریاضی به عنوان بزرگترین کران پایین یا همان infimum برای هر جابه جایی تعریف می شود. در این رابطه Π(Pr, Pg)مجموعه ای از توزیع های مشترک γ(x,y) را نشان می دهد که حاشیه آنها به ترتیب Prو Pg هستند.
با این حال، معادله فاصله Wasserstein بسیار سخت و تقریبا غیر قابل حل است. برای حل این مشکل با استفاده از با استفاده از دوگان Kantorovich-Rubinstein ، می‌توانیم این محاسبات را ساده کنیم:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)]$$

که در آن sup حداقل کران بالایی است و f تابع 1-Lipschitz است که به دنبال قرار دادن محدودیت زیر است :

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

$$\Rightarrow \quad \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]$$

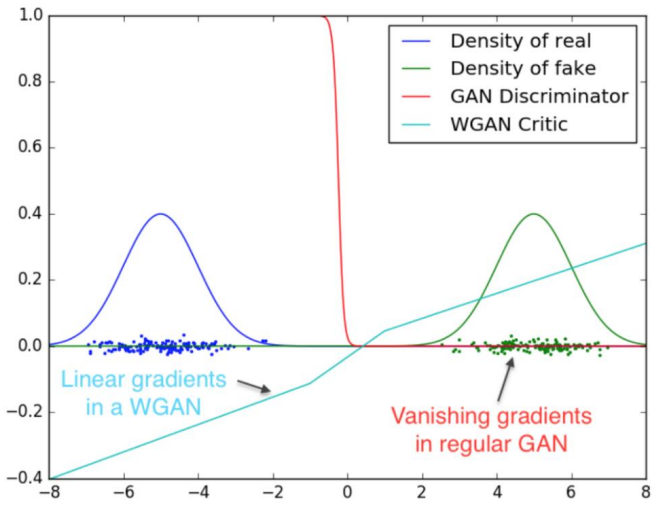
محدودیت Lipschitz continuity به نوعی برای تعیین سرعت تغییر یک تابع در نظر گرفته می شود و برای آن یک آستانه در نظر می گیرد. بنابراین برای محاسبه فاصله Wasserstein، فقط باید یک تابع 1-Lipschitz را پیدا کنیم. مانند سایر مشکلات یادگیری عمیق، ما می توانیم یک شبکه عمیق برای یادگیری آن بسازیم. در واقع، این شبکه بسیار شبیه به متمایز کننده Dاست، فقط بدون تابع سیگموید و به جای یک احتمال، یک مقدار اسکالر را خروجی می دهد. این مقدار را می توان به عنوان واقعی بودن تصاویر ورودی تفسیر کرد. طبق روش Wasserstein GAN نام تمایزکننده (discriminator) را به critic (منتقد) تغییر می‌دهیم.

بنابریان طبق فرضیات گفته شده ما ابع هزینه ی معماری اولیه GAN را در W-GAN به شکل زیر تغییر می دهیم.

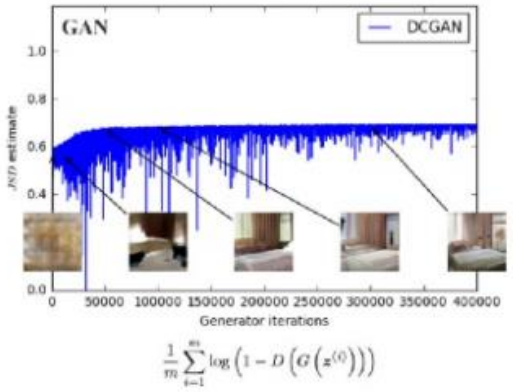
	Discriminator/Critic	Generator
GAN	$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right]$	$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(D\left(G\left(z^{(i)}\right)\right)\right)$
WGAN	$\nabla_w \frac{1}{m} \sum_{i=1}^{m} \left[f(x^{(i)}) - f(G(z^{(i)})) \right]$	$\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f(G(z^{(i)}))$

همانطور که گفته شد تابع f باید یک تابع 1-Lipschitzباشد. برای اعمال این محدودیت، WGAN یک برش بسیار ساده برای محدود کردن حداکثر مقدار وزن در f اعمال می‌کند، یعنی وزن‌های تشخیص‌دهنده باید در محدوده خاصی باشد که توسط ابر پارامتری c کنترل می‌شود.

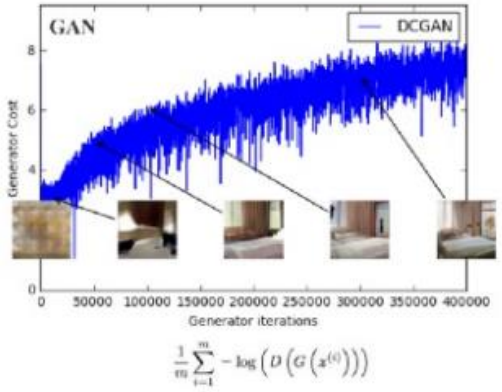
$$w \leftarrow w + \alpha \cdot \mathrm{RMSProp}(w, g_w)$$
$$w \leftarrow \mathrm{clip}(w, -c, c)$$



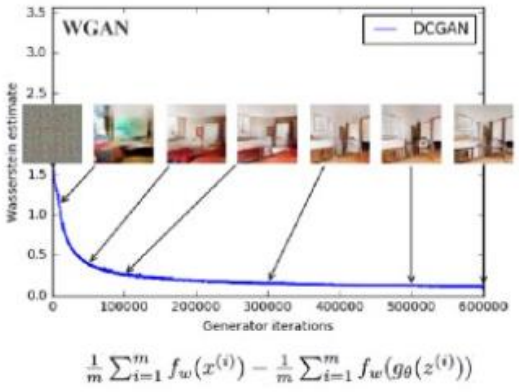
همانطور که در شکل مقابل می بینیم زمانی که تابع هرینه w-gan، (نمودار آبی) یا همان منتقد در w-gan بسیار بهتر عمل می کند و مشکل vanishing که در معماری اولیه وجود داشت را بر طرف می کند.



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

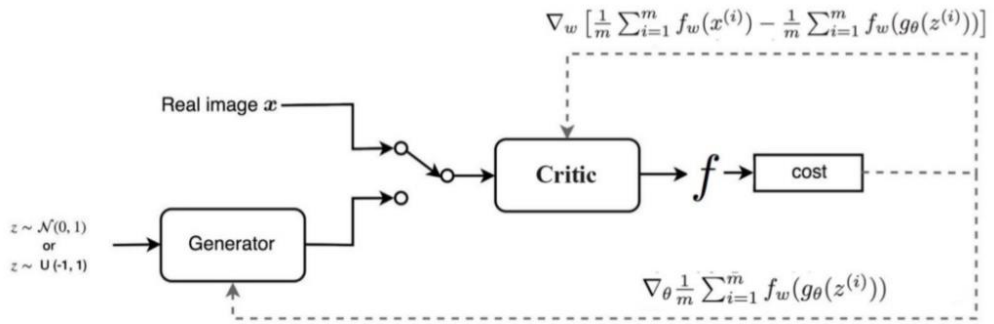


$$\frac{1}{m}\sum_{i=1}^{m}-\log\left(D\left(G\left(x^{(i)}\right)\right)\right)$$



$$\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$$

همانطور که در شکل بالا می بینیم بر خلاف معماری اولیه Gan که تابع هزینه ثبات ندارد و خیلی یکنواح نیست و تابع هزینه ی wgan روند نزولی و یکنواخت دارد. همچنین این تابع هزینه موجود در wgan از مسئله ی mode collapse نیز جلوگیری می کند و حتی زمانی که Critic بسیار خوب عمل می کند همچنان Generatr آموزش می بیند. در واقع مسئله ی Mode Collapse به این علت بر طرف می شود که تابع هزینه wgan بر این اساس است که این امکان را فراهم می کند که discriminator بدون نگرانی در مورد ناپدید شدن گرادیان‌ها، بهینه‌سازی شود، و این مسئله Mode Collapse را کاهش می‌دهد. در واقع اگر discriminator در حداقل های محلی گیر نکند، یاد می گیرد که خروجی هایی را که Generator روی آنها ثابت می شود (Collapse) را رد کند. بنابراین Generator باید Mode های جدیدی را امتحان کند و همواره یاد بگیرد..

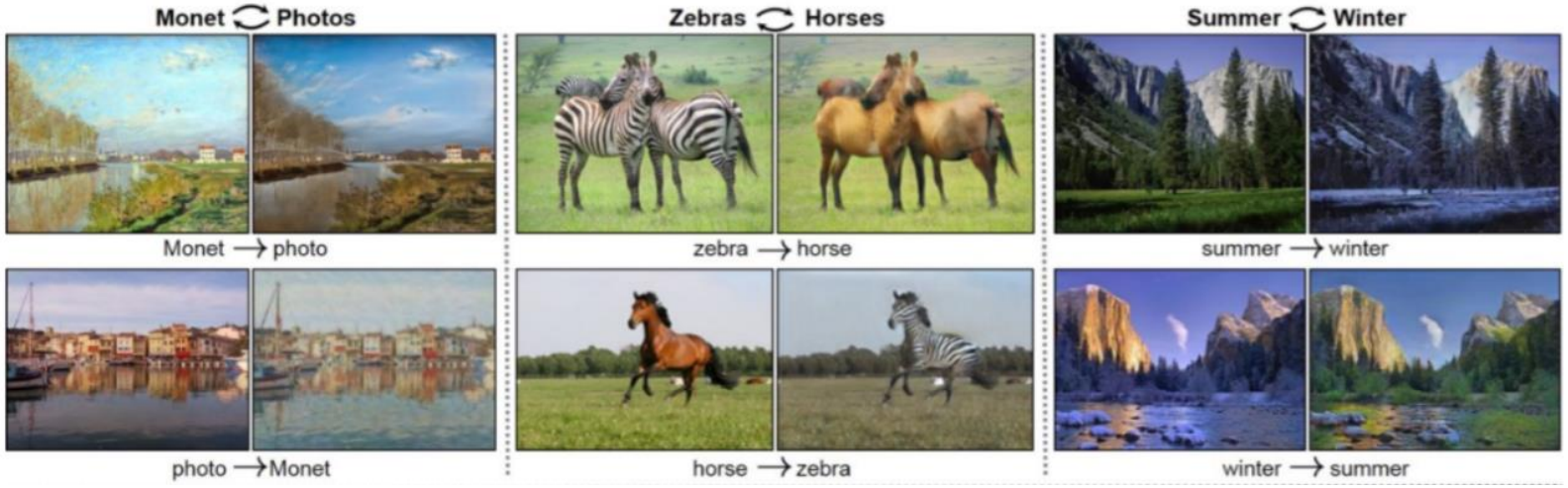


منبع https://blog.paperspace.com/wgans/

مسئله‌ی ۵. (۵ + ۱۰ نمره)

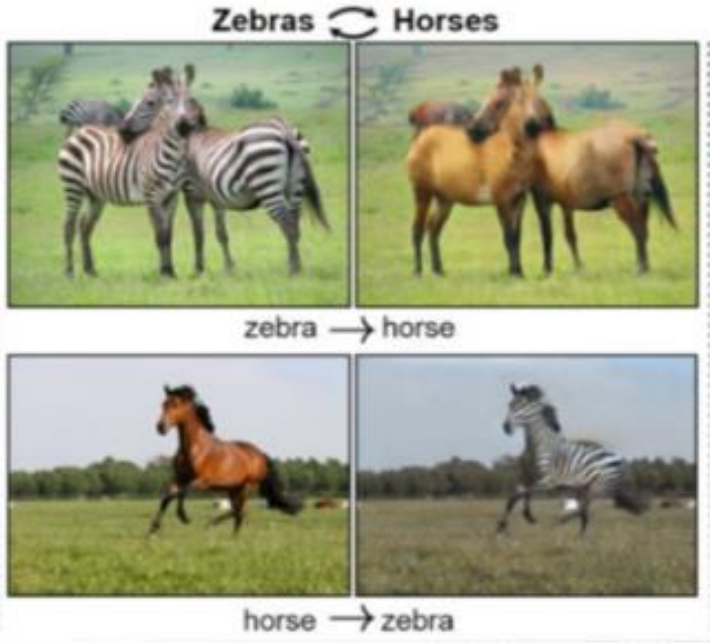
(آ) یکی از چالش های مهم که مدل های GAN تاثیر به سزایی در برطرف شدن آن ها دارند موضوع image2image translation می باشد. یکی از مدل هایی که به طور خاص برای این امر توسعه داده شده است مدل Cycle GAN می باشد. معماری دوگانه این مدل را تشریح کرده و مزایای آن را در ارتباط با چالش مذکور نسبت به مدل پایه GAN بیان نمایید.

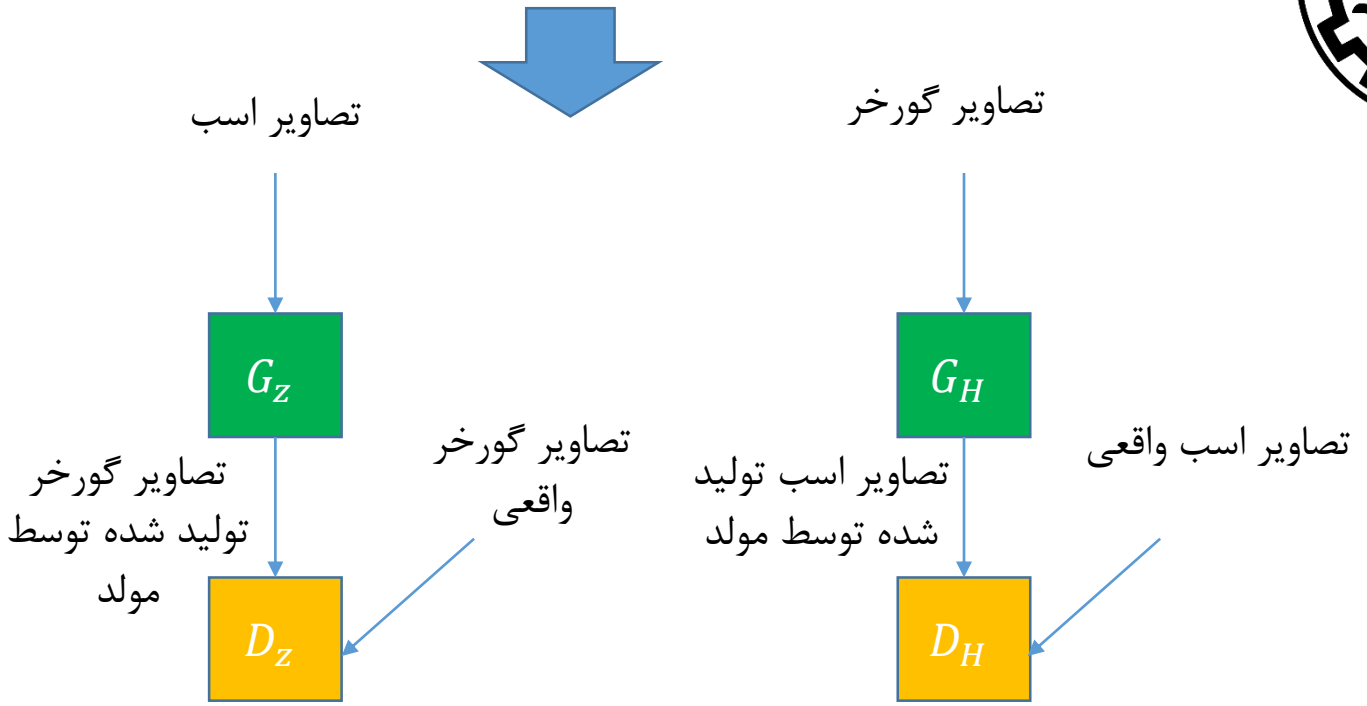
یکی از مسائل مطرح در حوزه ی شبکه های مولد، ترجمه تصویر به تصویر یا همان imagetoimage translation است. بر این اساس که ما به دنبال این هستیم که یک تصویر از یک دامنه توزیع مثلا تصاویر گورخرها را به یک دامنه توزیع دیگر مثلا تصاویر اسب تبدیل و نگاشت می کنیم و در حالت ایده آل به دنبال این هستیم که سایر ویژگی های تصویر مانند پس زمینه به طور قابل تشخیصی ثابت باقی بماند. همانطور که در تصویر زیر می بینیم:



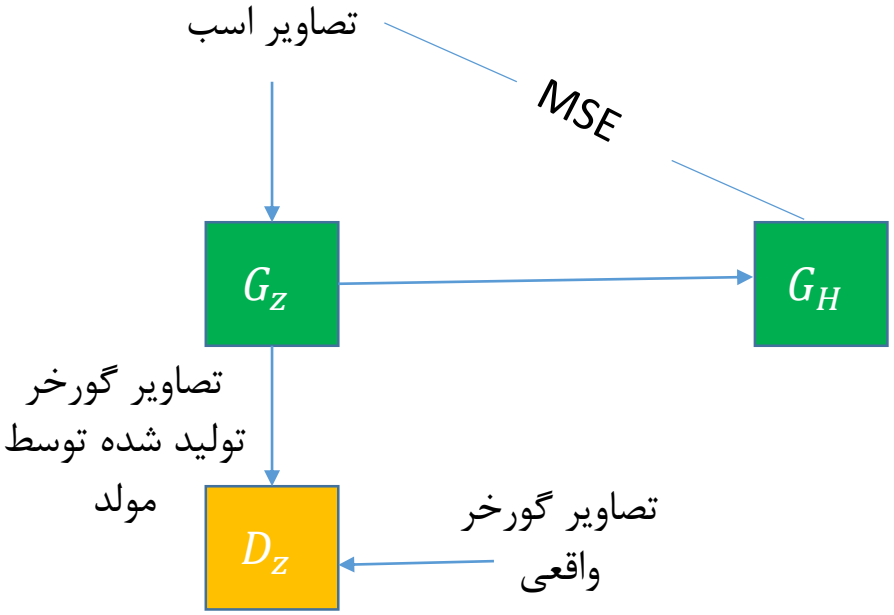
یکی از معماری ها و شبکه هایی که برای حل این مسئله معرفی شده است Cycle GAN ها هست. برخلاف سایر روش ها مانند Pix2Pix network که به تصاویر paired نیاز دارند CycleGAN فقط به دو مجموعه از تصاویر unpaired نیاز دارد و تمام تلاش خود را می کند تا mapping بین آنها را بیابد و نیاز به تصاویر paired ندارند که این مسئله از ویژگی های این نوع از شبکه ها است. از سوی دیگر این شبکه های توانایی انجام این کار را با حجم کمی داده ی آموزشی را دارند.

معماری cycle GAN بر این اساس است که از دو شبکه ی generator متفاوت تشکیل شده است. به عنوان نمونه مثلا بخواهیم تصاویر اسب را به گورخر تبدیل کنیم یا بالعکس. برای این مسئله همانطور که گفته شد ما دو شبکه ی generator در نظر می گیریم به صورت زیر:



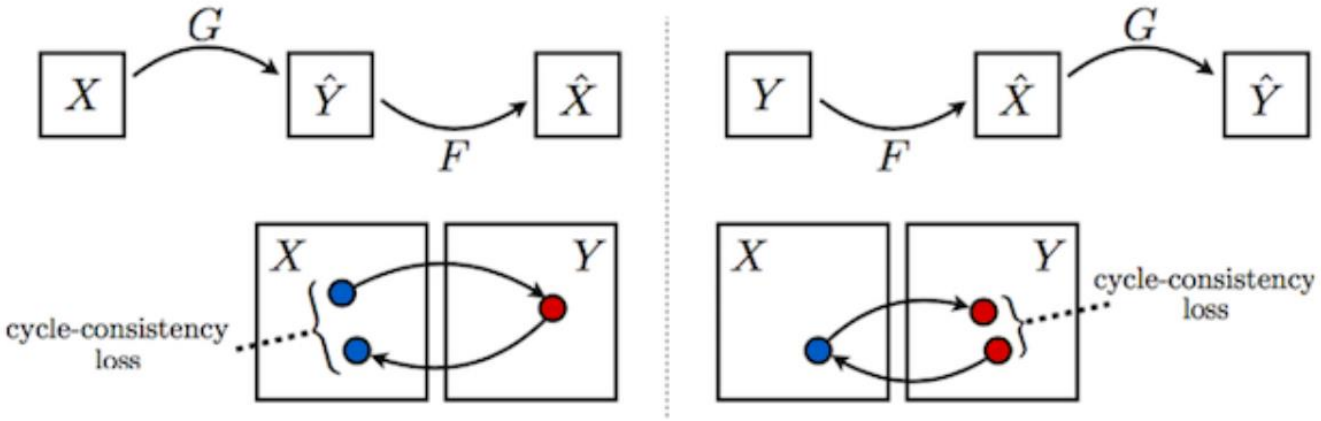


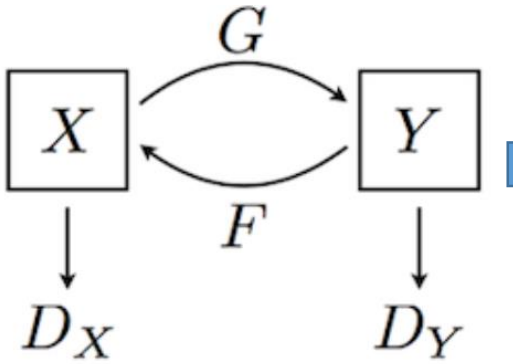
همانطور که از معماری بالا مشخص است، در این معماری ورودی های واقعی اسب و گورخر به ترتیب به شبکه های مولد G_Z و G_H داده می شوند و تصاویر تولید شده توسط یکی این شبکه های مولد همراه با نمونه های واقعی به ترتیب وارد شبکه های Discriminator، D_Z و D_H می شوند. نکته ای که در این معماری وجود دارد با اینکه عملا شبکه های مولد G_Z و G_H به ترتیب آموزش می بینند که از ورود های تصاویر اسب تصاویر گوخر تولید کنند و بالعکس اما این مسئله را در نظر نمی گیرند که ویژگی های اولیه تصاویر ورودی مثل جایگاه اسب در تصویر اولیه، پس زمینه ی تصویر و سایر اشیا داخل تصویر را ثابت نگه دارد و اما صرفا تولید کردن اسب از گورخر و یا بالعکس را آموزش می بنید و به بقیه ویژگی های تصویر توجهی نمی‌کند (در واقع ما فقط به دنبال این هستیم که گورخر در یک تصویر جایگزین اسب شود و یا بالعکس و نمیخواهیم تغییر دیگری ایجاد شود). بنابراین تابع هزینه ی هزینه ای که برای GAN در نظر گرفته می شود برای Cycle GAN مناسب نیست.



بنابراین ایده ای که برای حل این مشکل ارائه شد به صورت بود که تابع هزینه را طوری تغییر دهیم که ویژگی های تصویر از جمله پس زمینه حفظ شود. برای این منظور این ایده مطرح شد که خروجی شبکه مولد G_Z به عنوان ورودی شبکه مولد G_H داده شود و سپس توصیر خروجی با ورودی واقعی تصویر اسب مقایسه شود با استفاده از تابع MSE. در واقع این ایده از اینجا شکل گرفت که ما تابع هزینه طوری طراحی کنیم که علاوه بر اینکه به تبدیل تصاویر اسب به گورخر (یا بالعکس) تمکز کند به حفظ ویژگی های تصویر اولیه نیز تمرکز کند و یاد بگیرد ویژگی های دیگر تصوی را حفظ کند.

در واقع همانطور که در شکل سمت چپ مشخص است ما به دنبال این هستیم ورودی های مسئله را در معماری Cycle GAN با استفاده از یک تابع مولد به به یک توزیع و فضای حالت دیگر ببریم و سپس با یک تابع مولد دیگر به فضا و توزیع اولیه برگردانیم و ببینم که داده اصلی و داده منتقل شده به فضای دیگر توسط مولد و سپس برگشته به فضای اولیه چقدر اختلاف دارد. بر این اساس اطلاعات موجود توسط تابع هزینه ی MSE به عقب بر گردانده می شود و عملا بر روی دو تابع G_Z و G_H تاثیر می گذارد و همه ی این موارد برای تبدیل تصاویر گورخر به تصاویر اسب نیز وجود دارد.





بنابریان تابع هزینه ی معماری Cycle GAN را به صورت زیر می توانید بازنویسی کنیم:

$$LossGAN = -\log\left(D_Z\left(G_Z(Hourse)\right)\right) + Minimise\ round - tripe\ MSE$$

بخش اول این تابع برای تولید تصویر گورخر از اسب قرار داده شده است که توسط Discriminator انجام میشود

بخش دوم برای این مسئله قرار داده شده است که ویژگی های دیگر تصویر مانند تصویر زمینه و جایگاه اسب و اشیاع دیگر را ثابت نگه دارد

یا به شکل عمومی تر داریم:

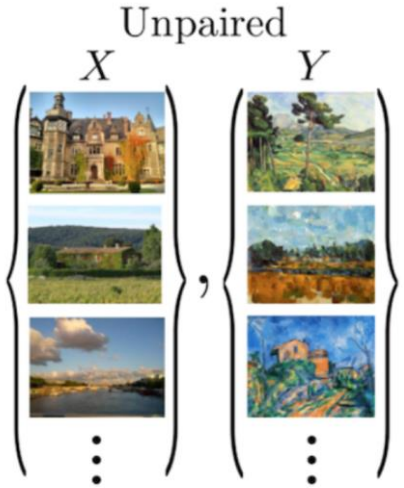
$$Loss_{adv}(G, D_y, X) = \frac{1}{m}\sum_{i=1}^{m}(1 - D_y(G(x_i)))^2$$

$$Loss_{adv}(F, D_x, Y) = \frac{1}{m}\sum_{i=1}^{m}(1 - D_x(F(y_i)))^2$$

$$Loss_{cyc}(G, F, X, Y) = \frac{1}{m}\sum_{i=1}^{m}[F(G(x_i)) - x_i] + [G(F(y_i)) - y_i]$$

$$Loss_{full} = Loss_{adv} + \lambda Loss_{cyc}$$

یکی از مهم ترین مزایای روش Cycle GAN عدم نیاز به تصاویر paired برای یادگیری است. زیرا عملا ما در بسایر از مواقع تصاویر paired نداریم و یا تهیه ی آن سخت است و یا تعداد آن کم است. بنابراین شکبه می تواند به راحتی با تصاویر Unpaired کار کند. همچنین از دیگر مزایای روش Cycle GAN عدم نیاز به تعداد زیادی تصاویر ورودی است.



منبع https://www.youtube.com/watch?v=6FkMCmPPEq4&t=674s

(ب) فرض کنید تحت شرایطی بسیار خاص از شما درخواست شده با تعداد داده محدود (در حدی که معماری پایه GAN به سختی به کمک آن train می شود) یک مدل Cycle GAN را train نمایید. در صورتی که الزام به استفاده از Cycle GAN باشد، راه حل پیشنهادی شما چیست؟ (دقت شود سوال ابتکاری بوده و پاسخ کاملا یکتا ندارد، اما بایستی هر راه حل پیشنهادی با تحلیل کامل و حتی استفاده از روابط ریاضی در صورت نیاز تشریح گردد).

با توجه به فرضیات مسئله همانطور که گفته شده به دلیل اینکه ما با حجم داده ای کمی به عناون داده های آموزش مواجه هستیم که این مسئله می تواند در عملکر شبکه های مولد همچون GAN یا نسخه های مختلف آن مثل Cycle GAN تاثیر بگذارد. یکی از مهم ترین ایده هایی که برای حل مشکل کمبود داده و بهبود عملکر شبکه های Cycle GAN پیشنهاد می شود استفاده از روش DATA Augmentation است که می توانیم با استفاده از آن داده های مجموعه ی آموزشی را افزایش دهیم. با این حال استفاده از روش DATA Augmentation مشکلات خاص خود را دارد از جمله اینکه ممکن است توزیع داده های Augment شده با توزیع داده های اصلی متفاوت باشد بنابراین این مسئله نه تنها صبب بهبود عملکرد شبکه نمی شود بلکه سبب گمراهیی generative می شود. در این راستا برای اینکه بتوانیم از روش Data Augmentation استفاده کنیم و در عین حال مشکلات گفته شده را نداشته باشیم معماری معرفی شده است در سال ۲۰۲۰ به نام Data Augmentation Optimized for GAN یا به اختصار DAG که نشان می دهد با استفاده از این روش، می توانیم زمانی که مجموعه داده های ما کم است عملکرد شبکه ی Cycle GAN و دیگر نسخه های GAN را بهبود دهیم. ایده ی این روش بر این اساس که Jensen–Shannon (JS) divergence را بین توزیع داده های اصلی و توزیع یادگرفته شده توسط مدل هب خداقل برسانیم. همانطور که می دانیم تابع هدف GAN به صورت زیر تعریف می شود:

$$\mathcal{V}(D,G) = \mathbb{E}_{\mathbf{x}\sim P_d} \log\Big(D(\mathbf{x})\Big) + \mathbb{E}_{\mathbf{x}\sim P_g} \log\Big(1 - D(\mathbf{x})\Big)$$

بنابر رابطه ی بالا اگر D^* تمییز دهنده ی بهینه باشد $minV(D^*,G)$ برابر است با Jensen-Shannon (JS) divergence، $JS(P_d,Pg)$ که در این رابطه P_d توزیع داده های اصلی و P_G توزیع داده هایی است که توسط generator تولید می شود. بنابراین به صورت تجربی نشان داده شده است که هرچقدر مجموعه داده های آموزشی بزرگ تر باشد Jensen-Shannon (JS) divergence بهبوبد پیدا می کند. اما زمانی که داده های کم باشند با استفاده از روش هایی مثل data Augmentation می توانیم داده ها را زیاد کنیم اما مشکلی که در این روش وجود دارد این است که اغلب روش های DATA Augmentation مبتنی بر چرخش داده filp و ..... عملا ترنسفورم کردن داده ها هستند که این مسئله سبب می شود که لزوما توزیع داده های Augment شده با داده های اصلی برابر نباشد و از سوی دیگر بسیار از داده ها قابلیت ترنسفروم ندارند.

به عنوان مثال در مقاله ای که معماری DAG ارائه شده ثابت می کند که روش Data Augmentation به صورت معمولی که بر اساس روش های ترنسفورم مثل rotate ،filliping ، Cliping و... نه تنها دقت را بهبود نمی دهد بلکه نسبت به روش baseline که GAN معمولی بود دقت کمتری دارند. این مسئله زمانی که داده های augmented بیشتر می شوند باعث می شود که generator کاملا دچار گمراهی شود و توزیعی که آموزش می بیند با توزیع داده های اصلی کاملا متفاوت است. در این مقاله ترنسفورم ها به دو دسته تقسیم می شوند invertible و Non-invertible که در جدول زیر آورده شده اند

Methods	Invertible	Description
Rotation	✓	Rotating images with 0°, 90°, 180° and 270° degrees.
Flipping	✓	Flipping the original image with left-right, bottom-up and the combination of left-right and bottom-up.
Translation	✗	Shifting images N_t pixels in directions: up, down, left and right. Zero-pixels are padded for missing parts caused by the shifting.
Cropping	✗	Cropping at four corners with scales N_c of original size and resizing them into the same size as the original image.
FlipRot	✓	Combining flipping (left-right, bottom-up) + rotation of 90°.

بای مینیمم کردن Jensen-Shannon (JS) divergence داریم:

$$\mathcal{V}(D^*, G) = -\log(4) + 2 \cdot \mathrm{JS}(P_d^{'} \,||P_g)$$

❖ فرض کنیم که ما مجموعه ترنسفورم های زیر را داشته باشیم:

$$\mathcal{T} = \{T_1, T_2, \ldots, T_K\}$$

❖ فرض می کنیم که مجموعه داده های مسئله با استفاده مجموعه ترنسفورم های بالا augment شود. فرض می شود T1 یک ترنسورم همانی است و خروجی آن عینا داده های اصلی است.

❖ تابع js به صورت زیر بازنویی می شود:

$$\mathcal{V}(D^*, G) = -\log(4) + 2 \cdot \mathrm{JS}(P_d^{\mathcal{T}} || P_g)$$

یک قضیه ای در انی مقاله مطرح می شود که اگر توابع ترنسفور ما که برای augment کردن استفاده می کنیم invertible باشد در این حالت تابع js که بر اساس داده های اصلی مینیمم می شود با حالتی که داده های augment میشود یکسان نیست:

$$\bar{\mathrm{JS}}(P_d || P_g) = \mathrm{JS}(P_d^T || P_g^T)$$

$$\bar{\mathrm{JS}}(P_d||P_g) = \mathrm{JS}(P_d^T||P_g^T)$$

بنابراین با این تکنیک مشکلاتی که در augment کردن داده ها به وجود می امد بر طرف می شد. بنابراین معماری اولیه ی GAN به شکل زیر تغییر می کند:

تابع هزینه آن به صورت زیر تعریف می شود:

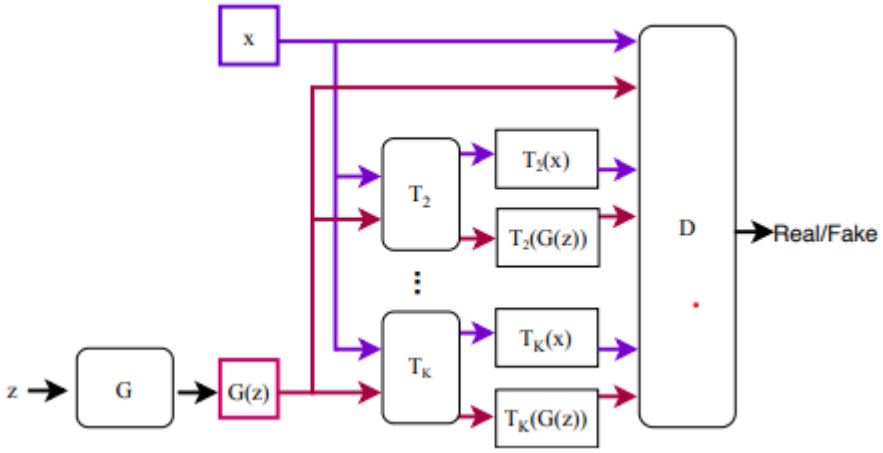
$$\mathcal{V}(D_k, G) = \mathbb{E}_{\mathbf{x}\sim P_d^{T_k}} \log\Big(D_k(\mathbf{x})\Big) + \mathbb{E}_{\mathbf{x}\sim P_g^{T_k}} \log\Big(1 - D_k(\mathbf{x})\Big)$$

بنابریان با این تکنیک تمییزه دهنده بهینه نیز مانند معماری اولیه GAN میشود که داریم:

$$\begin{aligned} D_k^*(T_k(\mathbf{x})) &= \frac{p_d^{T_k}(T_k(\mathbf{x}))}{p_d^{T_k}(T_k(\mathbf{x})) + p_g^{T_k}(T_k(\mathbf{x}))} \\ &= \frac{p_d(\mathbf{x})|\mathcal{J}^{T_k}(\mathbf{x})|^{-1}}{p_d(\mathbf{x})|\mathcal{J}^{T_k}(\mathbf{x})|^{-1} + p_g(\mathbf{x})|\mathcal{J}^{T_k}(\mathbf{x})|^{-1}} \\ &= \frac{p_d(\mathbf{x})}{p_d(\mathbf{x}) + p_g(\mathbf{x})} = D^*(\mathbf{x}) \end{aligned}$$

$$\mathcal{V}(D_k^*, G) = -\log(4) + 2 \cdot \mathrm{JS}(P_d^{T_k}||P_g^{T_k})$$

بنابر این طبق این معماری ما داده های واقعی و داده هایی تولید شده توسط generator که Augment شده اند با ترنسفورم های مختلف را (که در این ترنسفورم ها ترنسفورم همانی نیز هست) به عنوان ورودی به یک discriminator می دهیم.



❖ اما مشکلی که در این روش نیز وجود داشت بود که به دلیل اینکه تمامی ترنسفور ها به یک discriminator داده می شوند عملا رابطه ی زیر همچنان صادق نمی ماند.

$$\bar{\mathrm{JS}}(P_d||P_g) = \mathrm{JS}(P_d^T||P_g^T)$$

برای اثبات این قضیه فرضیات را در این مقاله در نظر گرفته شده است به این صورت که دو توزیع p وq را در نظر بگیرید:

$$q = \sum_{m=1}^{K} w_m q^m$$

$$p = \sum_{m=1}^{K} w_m p^m$$

$$\Rightarrow \sum_{m=1}^{K} w_m = 1$$

❖ اگر دو توزیع q^m و p^m، دو توزیع q^0 و p^0 باشند که بر اساس ترنسفورم های Tm invertible ترنسفورم شده باشند داریم:

$$\mathrm{JS}(p||q) \leq \mathrm{JS}(p^0||q^0)$$

که اثبات آن بر اساس قضیه ای است که در چند اسلاید قبل گفتیم داریم، اگر Tm یک ترنسفورم باشد که invertible است آنگاه داریم:

$$\mathrm{JS}(p^0||q^0) = \mathrm{JS}(p^m||q^m)$$

بنابراین داریم:

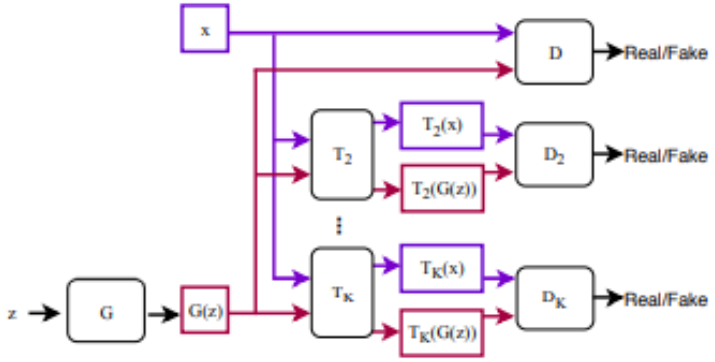
$$\mathrm{JS}(p||q) \leq \sum_{m=1}^{K} w_m \mathrm{JS}(p^0||q^0) = (\sum_{m=1}^{K} w_m)\mathrm{JS}(p^0||q^0) = \mathrm{JS}(p^0||q^0)$$

❖ بنابراین این ایده ارائه شده که این معماری را باید تغییر دهیم، برای این کار به ازای هر ترنسفورمر یک discriminator در نظر میگیریم به صورت زیر:

$$\mathcal{T} = \{T_1, T_2, \ldots, T_K\}$$

$$\{D_k\} = \{D_2, D_3, \ldots D_K\}$$

بنابریان بر اساس ایده ی جدید معماری زیر ارائه شد:



❖ این معماری بسیار شبیه معماری قبلی است با این تفاوت که برای هر ترنسفورم یک discriminator در نظر گرفته می شود

که تابع هدف آن به صورت زیر تعریف می شود:

$$\max_{D,\{D_k\}} \mathcal{V}(D,\{D_k\},G) = \mathcal{V}(D,G) + \frac{\lambda_u}{K-1}\sum_{k=2}^{K}\mathcal{V}(D_k,G)$$

$$\min_{G} \mathcal{V}(D,\{D_k\},G) = \mathcal{V}(D,G) + \frac{\lambda_v}{K-1}\sum_{k=2}^{K}\mathcal{V}(D_k,G)$$

$$\mathcal{V}(\{D_k^*\},G) = \text{const} + 2\sum_{k=2}^{K}\text{JS}(P_d^{T_k}||P_g^{T_k})$$

منبع

مقاله:On Data Augmentation for GAN Training