MATH FOR CS - GANs

Loss functions of General Adversarial Networks (GANs)

Vinh Quang Ngo - 19520354

***Tóm tắt nội dung*—Generative Adversarial Networks (GANs), tiếng Việt dịch là mạng sáng tạo đối nghịch là một kỹ thuật nổi lên trong những năm gần đây, được ứng dụng rộng rãi vào các bài toán học giám sát và học bán giám sát. Cụ thể hơn, GANs thuộc nhóm Generative model, nghĩa là một mô hình có khả năng sinh ra dữ liệu mới, còn về nguồn gốc của từ "đối nghịch" là do mạng GANs được cấu thành từ hai mạng là Generator (G) và Discriminator (D), luôn đối nghịch nhau trong quá trình huấn luyện mạng GANs. Trong đồ án này, em sẽ trình bày chi tiết các vấn đề toán học ẩn sau hàm mất mát của GANs, từ đó tìm ra những ưu, khuyết điểm cùng cách khắc phục cho một số trường hợp.**

I. Giới thiệu Generative Adversarial Networks

GENERATOR (G) và Discriminator (D) có trong GANs có thể được hình dung như là một người chuyên tạo các sản phẩm giả (G) và một chuyên gia giám định (D), người làm giả muốn tạo ra những sản phẩm giả giống thật đến mức có thể qua mắt được chuyên gia giám định, còn chuyên gia sẽ phải phân biệt được đâu là sản phẩm thật và sản phẩm giả.

Trong quá trình huấn luyện, chuyên gia sẽ có hai nhiệm vụ là: học cách phân biệt đâu là sản phẩm giả, đâu là sản phẩm thật và nói cho người chuyên làm giả rằng nó phải làm tốt hơn nữa. Dần dần, người làm đồ giả cũng làm nhái đồ giống thật hơn, chuyên gia cũng phân biệt thật giả tốt hơn, và kì vọng là sản phẩm giả sinh ra từ GANs sẽ đánh lừa được chuyên gia. Trong quá trình này, người làm giả không được tiếp xúc trực tiếp với sản phẩm thật, cách duy nhất để người làm giả học là thông qua nhận xét của người chuyên gia, còn người chuyên gia có thể xem xét được cả sản phẩm thật và sản phẩm giả.

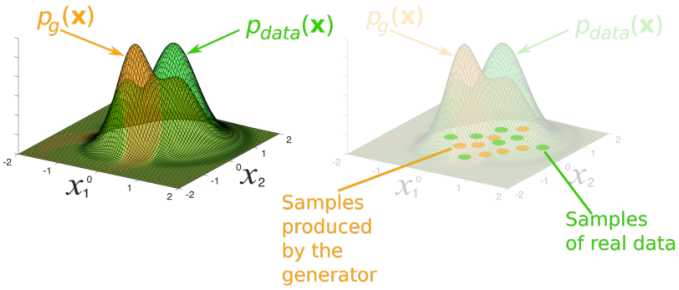
Ý tưởng của GANs bắt nguồn từ Zero - Sum game, hiểu đơn giản là trò chơi đối kháng 2 người (cờ vua, cờ tướng), nếu một người thắng thì người còn lại sẽ thua. Giả sử rằng, người chơi 1 là người luôn muốn dành cực đại hóa khả năng chiến thắng và người chơi 2 là người luôn muốn cực tiểu hóa khả nắng chiến thắng của người chơi thứ nhất (minimax). Khi đạt trạng thái cân bằng, tức là các bước đi tiếp không làm tăng cơ hội thắng của mình hay đối phương. Điều này là tương tự đối với Generator (G) và Discriminator (D).

II. Sơ bộ về Generative Adversial Networks

A. Generator và Discriminator

Trong mạng GANs cơ bản, Discriminator (D) thường là một hàm ánh xạ từ đầu vào (input) tới xác suất đầu vào đó là thật hay giả $D : D(x) \to (0, 1)$. Khi mà hàm Discriminator đạt được nghiệm tối ưu, lúc này Generator (G) sẽ tiếp tục được huấn luyện để làm giảm độ chính xác mà Discriminator có thể dự đoán. Khi mà Discriminator không còn phân biệt được đâu là thật giả nữa (dự đoán 0.5 cho tất cả input đầu vào dù thật hay giả) thì lúc này mạng đã đạt kì vọng.

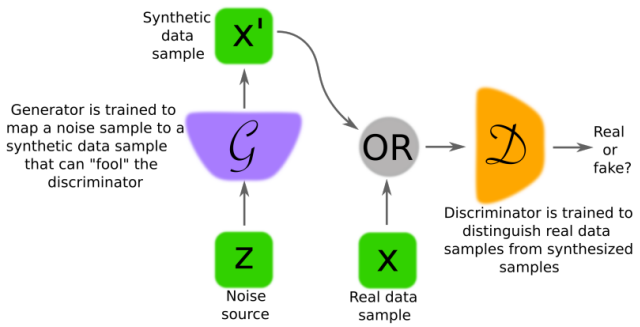
Còn đối với Generator, ta có thể xem như đây là một hàm ánh xạ từ một không gian ẩn (latent space) đến không gian của dữ liệu thực $G : G(\mathbf{z}) \to \mathbf{R}^{|\mathbf{x}|}$, với $\mathbf{z}$ là một phần tử từ không gian ẩn, $x \in R^{|x|}$ là một điểm dữ liệu từ dữ liệu chuẩn, $|x|$ chỉ số chiều của dữ liệu. Generator sẽ học cách bắt chước phân phối xác suất của dữ liệu huấn luyện, từ đó giúp ta có thể tạo ra dữ liệu mới từ phân phối đó.



Hình 1. Phân phối của dữ liệu huấn luyện và dữ liệu được Generator tạo ra

B. Một số ký hiệu, lưu ý

Trong bài viết này, các không gian sẽ được ký hiệu bằng các chữ cái in nghiêng (không gian ẩn: z), $p_{data}(x)$ là hàm phân phối mật độ xác suất của của tập dữ liệu huấn luyện, $p_g(x)$ là hàm phân phối mật độ xác suất được tạo ra bởi Generator, G và D để đại diện cho Generator và Discriminator. Mỗi mạng đều có bộ trong số $\theta_G, \theta D$ để tối ưu trong quá trình huấn luyện. Kí hiệu nabla ∇_{θ_G} dùng để chỉ đạo hàm theo trọng số của Discriminator.



Hình 2. Từ một noise input z thông qua Generator sẽ tạo ra một mẫu giả, mẫu giả này cùng mẫu thật được đưa qua Discriminator để dự đoán

III. HÀM MẤT MÁT CỦA GANS

A. Cross Entropy

Hàm mất mát được đề cập đến trong bài báo khoa học của tác giả GoodFellow được lấy ý tưởng từ Cross entropy. Vậy nên để hiểu được cách hàm mất mát của GANs hoạt động, ta cần phải hiểu được Cross Entropy là gì. Cross Entropy giữa hai phân phối $\mathbf{p}$ và $\mathbf{q}$ được định nghĩa là: $H(p,q) = E_q[log(q)]$. Đối với phân phối liên tục, Cross Entropy có thể biến đổi thành $H(p,q) = \int_x p(x)log(q(x))dx$. Đối với bài toán được nêu trong GANs cơ bản, hàm mất mát có thể được viết dưới dạng binary loss - entropy như sau:

$$L(\hat{y}, y) = [y.log\hat{y} + (1-y).log(1-\hat{y})] \tag{1}$$

Trong đó: y là nhãn ứng với dữ liệu của tập huấn luyện, $\hat{y}$ là xác suất dữ liệu đầu vào được dự đoán là "True" khi thông qua Discriminator.

Khi huấn luyện Discriminator, vì dữ liệu được lấy từ tập huấn luyện nên $P_{data}(x)$ hay $y = 1$ và $\hat{y} = D(x)$. Thay vào (1), ta được:

$$L((D(x), 1) = log(D(x)) \tag{2}$$

Đối với dữ liệu được tạo ra từ Generator, hiển nhiên nhãn của chúng là $y = 0$ và $\hat{y} = D(G(z))$. Thay vào (1), ta được:

$$L((D(G(z)), 0) = log(1 - D(G(z))) \tag{3}$$

Từ phương trình (2) và (3), vì nhiệm vụ của Discriminator là phân biệt được đâu là thật và giả, vậy nên cần phải cực đại hóa phương trình (2) và (3):

$$L^{(D)} = max[log(D(x) + log(1 - D(G(z)))] \tag{4}$$

Còn đối với Generator thì nhiệm vụ là ngược lại, ta cần phải cực tiểu hóa phương trình (2) và (3):

$$L^{(G)} = min[log(D(x) + log(1 - D(G(z)))] \tag{5}$$

Từ (4) và (5), ta có được hàm mất mát tối ưu cho 1 điểm dữ liệu:

$$L = \min_G \max_D [log(D(x) + log(1 - D(G(z)))] \tag{6}$$

Để tổng quát hóa lên cho toàn bộ tập dữ liệu, ta sẽ lấy kì vọng thế vào phương trình (6):

$$\begin{aligned} L = \min_G \max_D (&E_{P_{data}(x)}[log(D(x)] \\ &+ E_{P_z(z)} log(1 - D(G(z)))) \end{aligned} \tag{7}$$

Như vậy, ta đã chứng minh được công thức của hàm mất mát được tác giả GoodFellow nêu ra trong bài báo khoa học.

B. Thuật toán tìm cực trị cho hàm mất mát

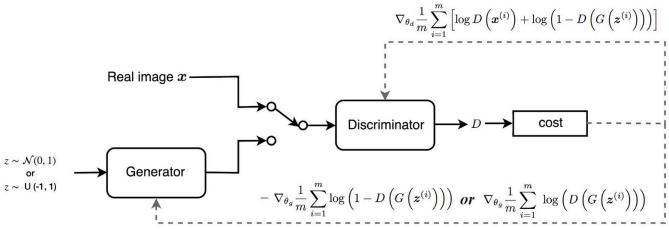
Thuật toán được tác giả GoodFellow giới thiệu trong bài báo để tìm cực trị cho hàm mất mát là minibatch stochastic gradient descent. Ở đây, ta sẽ phải tối ưu hàm mất mát theo G và D với mục đích đối nghịch nhau, vậy nên ta sẽ huấn luyện Discriminator trước với gradient ascent được cập nhật theo công thức sau:

$$\nabla_{\theta_D} \frac{1}{m} \sum_{i=1}^{m} [log(D(x^{(i)}) + log(1 - D(G(z^{(i)})))] \tag{8}$$

Sau khi Discriminator được cập nhật tới điểm gần tối ưu khi Generator bị cố định (Lý do không cập nhật tới điểm tối ưu sẽ được giải thích sau), Generator sẽ được cập nhật lại theo gradient descent công thức sau:

$$\nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^{m} [log(1 - D(G(z^{(i)})))] \tag{9}$$

Ngoài ra, trong bài báo, tác giả cũng đề cập rằng, ta có thể sử dụng các biến thể của Gradient Descent như: momentum, Adam để tối ưu quá trình tìm kiếm điểm cực trị.



Hình 3. Quá trình huấn luyện mạng GANs gồm 2 giai đoạn lặp đi lặp lại là huấn luyện Discriminator và huấn luyện Generator. Thuật toán dừng lại khi đạt cân bằng Nash

C. Cực đại hóa hàm mất mát theo D

Nhắc lại công thức ở mục (6):

$$\begin{aligned} V(G,D) = \min_G \max_D (&E_{P_{data}(x)}[log(D(x)] \\ &+ E_{P_z(z)} log(1 - D(G(z)))) \end{aligned}$$

Đồng thời, ta cũng có công thức của kỳ vọng của một hàm $g(x)$ tùy ý là: $E_{p(x)}[g(x)] = \int_x p_x((g(x))dx$. Áp dụng vào công thức trên ta được:

$$\begin{aligned} V(G,D) = &\int_x p_{data}(x) log(D(x)) dx \\ &+ \int_z p_z(x) log(1 - D(G(z))) dz \end{aligned} \tag{10}$$

Đầu tiên, đặt $x = G(z)$, ta có được các phương trình sau:

- $x = G(z) \Rightarrow z = G^{-1}(x)$
- $P_g(x)dx = P_z(z)dz \Rightarrow p_g(x) = p_z(G^{-1}(x))|\frac{d(G^{-1}(x))}{dx}|$

Thay $z = G^{-1}(x)$ và $p_g(x) = p_z(G^{-1}(x))|\frac{d(G^{-1}(x))}{dx}|$ vào phương trình (10) ta được:

$$V(G,D) = \int_x p_{data}(x) log(D(x)) dx + p_g(x) log(1 - D(G(z)) \tag{11}$$

Để tìm được cực đại theo D, ta cần đến một tính chất của tích phân sau:

$$f(x) \geq g(x) \Rightarrow \int_x f(x)dx \geq \int_x g(x)dx$$

Vậy ta cần đi tìm nghiệm cực đại của phương trình $p_{data}(x)log(D(x))dx + p_g(x)log(1 - D(G(z))$ để phương trình (11) đạt cực đại. Và hiển nhiên phương pháp để tìm cực trị ở đây là giải phương trình đạo hàm bằng 0 như sau:

$$\frac{d}{d(Dx)}[p_{data}log(D(x)) + p_g(x)log(1 - (D(x)))] = 0$$
$$\Leftrightarrow \frac{p_{data}(x)}{D(x)} - \frac{p_g(x)}{1 - D(x)} = 0$$
$$\Leftrightarrow \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} = D^*_G(x)$$

Lấy đạo hàm cấp 2 của phương trình (11), ta dễ dàng chứng minh được $\nabla^2 V(G, D) < 0$. Đồng thời, ta cũng có tính chất, hàm số $f(x)$ đạt cực đại tại x_0 nếu:

$$\begin{cases} \nabla_{x_0} f(x) & = 0 \\ \nabla^2_{x_0} f(x) & < 0 \end{cases}$$

Vậy nên ta có được D tối ưu là:

$$D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \tag{12}$$

D. Cực tiểu hóa hàm mất mát theo G

Trước tiên, ta hãy đi tìm hiểu đôi chút về phân kỳ Kullback-Leibler và Jensen-Shannon là phương pháp để so sánh sự khác biệt giữa hai phân phối khác nhau. Với hai phân phối độc lập P và Q trong cùng một không gian xác suất ($\mathcal{X}$), sự tương quan của Q đối với P được định nghĩa là:

$$D_{KL}(P \parallel Q) = -\sum_{x \in \mathcal{X}} P(x)log(\frac{P(x)}{Q(x}$$

Phân kỳ Jensen là một hàm cải tiến của phân kỳ Kullback-Leibler với nhiều ứng dụng hơn bởi nó luôn đối xứng và xác định (Kull-back-Leibler không xác định tại $P(X) = 0$ hoặc $Q(x) = 0$). Ta định nghĩa phân kì Jensen-Shannon là:

$$JSD(P \parallel Q) = \frac{1}{2}D(P \parallel M) + \frac{1}{2}D(Q \parallel M)$$
$$whereM = \frac{1}{2}(P + Q)$$

Dựa vào thuật toán tìm cực trị cho hàm mất mát đã nêu trên, thay (12) vào (6), ta có được phương trình sau:

$$V(D^*_G, G) = -log4 + KL\left(p_{data} \parallel \frac{p_{data}(x) + p_g(x)}{2}\right) + KL\left(p_g \parallel \frac{p_{data}(x) + p_g(x)}{2}\right) = -log4 + 2.JSD(p_{data} \parallel p_g)$$

Dựa vào định nghĩa, ta dễ dàng chứng minh được phân kỳ Jensen-Shannon luôn luôn không âm, vậy nên ta có:

$$G^* = argminV(D^*_G, G) = -log(4) \tag{13}$$

Dấu bằng xảy ra khi và chỉ khi $P_g = P_{data}$, tương đương với Generator đã hoàn mỹ tạo ra sản phẩm giống với dữ liệu huấn luyện.

E. Sự hội tụ của hàm mất mát

Nhắc lại công thức ở mục (6):

$$V(G, D) = \min_G \max_D (E_{P_{data}(x)}[log(D(x)] + E_{P_z(z)}log(1 - D(G(z))))$$

Giả sử rằng $V(G, D) = U(p_g, D)$ là một hàm theo p_g thì $U(p_g, D)$ là một hàm lồi. Ta lấy supremum (cận trên) của phương trình (6), theo lý thuyết thì đây cũng là một hàm lồi và khả vi tại mọi điểm, vậy nên ta có thể tìm được điểm cực trị bằng các phương pháp như Gradient Descent. Và đó cũng là cơ sở về mặt toán học cho thuật toán của GANs.

IV. Một số vấn đề của GANs trong thực tế

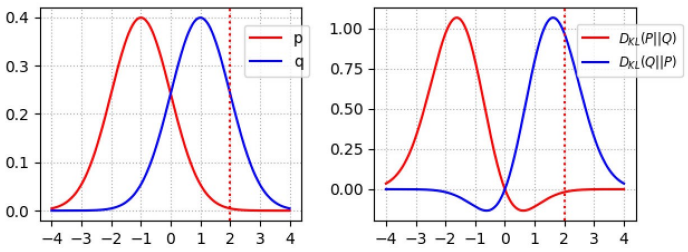
Như đã chứng minh toán học ở trên, mặc dù về mặt lý thuyết thì luôn tồn tại nghiệm tối ưu cho bài toán, nhưng việc huấn luyện GANs trong thực tế vẫn gặp rất nhiều khó khăn. Để khắc phục những khó khăn này buộc phải giải quyết trên cơ sở thực nghiệm.

A. Nguyên nhân không dùng Kullback-Leibler mà lại dùng Jensen-Shannon

Như định nghĩa về phân kỳ Kullback-Leibler đã nêu ở trên, khi ta muốn so sánh sự khác biệt giữa 2 phân phối P và Q, ta có:

$$D_{KL}(P \parallel Q) = -\sum_{x \in \mathcal{X}} P(x)log(\frac{P(x)}{Q(x})$$

Đây là một hàm bất đối xứng bởi $D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P)$. Dựa vào hàm ta dễ thấy rằng, $KL(x)$ sẽ tiến dần đến 0 tại những điểm mà $p(x) \to 0$. Quan sát đồ thị:



Hình 4. So sánh phân kỳ Kullback-Leibler giữa hai phân phối P và Q. Trong ví dụ này ta sẽ giả sử rằng P là phân phối của dữ liệu thật và Q là phân phối của mô hình Generator.

Từ đồ thị trên ta có thể nhận thấy rằng sẽ có 2 trường hợp cho hàm phân kỳ:

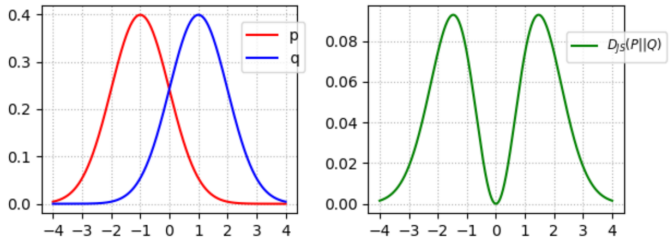
- Nếu ta tính hàm mất mát theo $D_{KL}(P \parallel Q)$, khi $p(x) > 0$ nhưng $q(x) \to 0$ thì điểm phạt rất cao, vậy nên sẽ loại bỏ được những hình không giống thật. Tuy nhiên, khi $q(x) > 0$ nhưng $p(x) \to 0$ thì điểm phạt lại rất thấp. Điều này dẫn đến chất lượng ảnh cho ra rất thấp vì ảnh

Generator tạo ra dù khác với phân phối thực nhưng lại không bị điểm phạt nặng.

- Tương tự trên, khi ta tính hàm mất mát theo $D_{KL}(Q \parallel P)$, khi $p(x) \to 0$ và $q(x) > 0$ thì điểm phạt rất cao, giúp loại bỏ các dữ liễu không giống thật. Nhưng khi $q(x) \to 0$ và $p(x) > 0$ thì lại phạt rất thấp,lúc này phân phối tạo ra có độ chính xác cao hơn nhưng lại kém đa dạng.

Lúc này ta sẽ cần đến cách tính khác để ổn định hơn, người ta bắt đầu áp dụng phân kỳ Jensen-Shannon vào hàm mất mát ứng với Generator. Nhắc lại định nghĩa phân kì Jensen-Shannon:

$$JSD(P \parallel Q) = \frac{1}{2}D(P \parallel M) + \frac{1}{2}D(Q \parallel M)$$
$$where M = \frac{1}{2}(P + Q)$$



Hình 5. So sánh phân kỳ Jensen-Shannon giữa hai phân phối P và Q. Trong ví dụ này ta sẽ giả sử rằng P là phân phối của dữ liệu thật và Q là phân phối của mô hình Generator. Dễ nhận thấy đây là một hàm đối xứng.

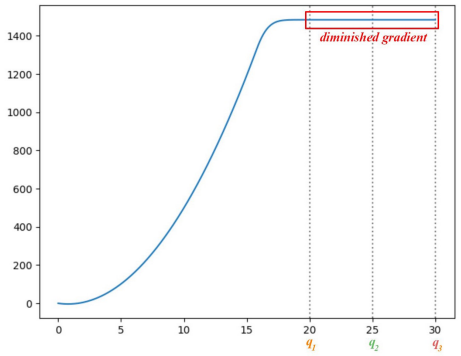
Từ đồ thị trên ta thấy hàm mất mát lúc này là hàm đối xứng $D_{KL}(P \parallel Q) = D_{KL}(Q \parallel P)$, dù với trường hợp ($p(x) > 0$ và $q(x) \to 0$) hay ($q(x) > 0$ và $p(x) \to 0$) thì mô hình đều bị phạt rất nặng, vậy nên phân kỳ Jensen-Shannon được áp dụng vào hàm mất mát bởi trên thực nghiệm lẫn lý thuyết thì hàm mất mát này đều cho ra kết quả tốt hơn.

B. Vấn đề Gradient Vanishing

Việc sử dụng các hàm kích hoạt vào các lớp của mô hình khiến cho đạo hàm của của hàm mất mát tiến dần về 0, làm cho các tham số được cập nhật rất chậm. Đơn cử như việc sử dụng hàm kích hoạt Sigmoid, khiến đầu ra của hàm được giới hạn $[0, 1]$, chính vì vậy, sự thay đổi cực lớn ở đầu vào chỉ thay đổi một lượng rất nhỏ ở đầu ra. Hiện tượng này có nhiều cách chứng minh, tuy nhiên ví dụ này sẽ chứng minh dựa trên hàm mất mát của Generator. Nhắc lại hàm mất mát của Generator:

$$G^* = -log4 + 2.JSD(p_{data} \parallel p_g)$$

Giả sử ta có bốn phân phối $p \sim \mathcal{N}(0, 1)$, $q1 \sim \mathcal{N}(20, 1)$, $q2 \sim \mathcal{N}(25, 1)$, $q3 \sim \mathcal{N}(30, 1)$, ta có đồ thị thể hiện phân kỳ Jensen-Shannon của p so với $q1, q2, q3$ như sau:



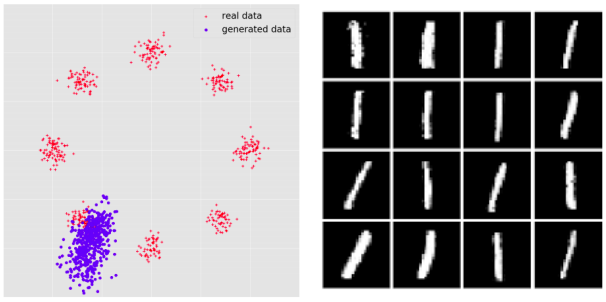
Hình 6. Phân kỳ Jensen-Shannon của $p \sim \mathcal{N}(0, 1)$ so với $q1 \sim \mathcal{N}(20, 1)$, $q2 \sim \mathcal{N}(25, 1)$, $q3 \sim \mathcal{N}(30, 1)$.

Dựa vào đồ thị ta thấy được, biên trên của hàm tạo thành một đường gần như nằm ngang tuyến tính. Vậy nên khi tính đạo hàm của G theo θG, con số này tiến dần tới 0 làm cho Generator rất khó tìm được điểm tối ưu. Hiện tượng này gọi là Gradient Vanishing.

Để tránh hiện tượng hàm logit bão hòa dẫn đến Gradient Vanishing như trên, tác giả của GANs đã đề xuất một phương án thay thế, thay vì chúng ta đi cực tiểu hóa hàm $log(1 - D(G(z)))$, ta có thể đi cực đại hóa hàm $log(D(G(z))$. Trên lý thuyết, hàm mất mát này có thể tìm ra điểm tối ưu tương tự và đồng thời tránh được hiện trượng Gradient Vanishing.

Tuy nhiên, một số bài báo lại cho rằng: phương pháp được đề xuất trong bài báo gốc của Good Fellow lại dẫn đến hiện tượng đạo hàm cực kì không ổn định, vậy nên một cuộc chạy đua toán học đã xảy ra để tìm kiếm một số mô hình toán học mới phù hợp hơn với bài toán như: LSGAN, WGAN, WGAN-GP, BEGAN, ...Nhưng phần lớn đều chưa thực sự vượt qua bài toán gốc, đơn cử như với phân kỳ Kullback-Leibler như trên ta có thể cho ra được một mô hình có độ chính xác cao, tuy nhiên lại kém đa dạng, chưa thực sự phù hợp.

C. Hiện tượng Mode Collapsing



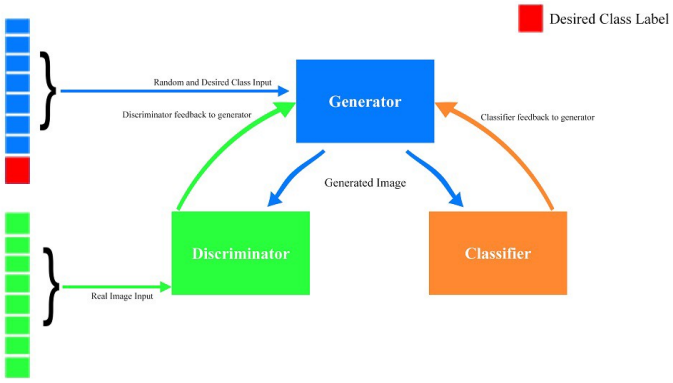
Hình 7. Dữ liệu được tạo ra chỉ tập trung vào một cực trị cục bộ khiến cho các dữ liệu này trở nên khá giống nhau dù với những input đầu vào khác nhau. Hình bên phải là hiện tượng mode collapse khi huấn luyện trên dữ liệu chữ viết tay MNIST

Mode collapse là một trong những vấn đề nan giải nhất của mạng GANs, là hiện tượng là dù đầu vào là gì thì Generator

đều sinh ra những kết quả gần giống nhau. Hiện tượng này có thể hiểu đơn giản là trong quá trình huấn luyện, Generator tạo ra một mẫu có thể đánh lừa Discriminator, lúc này Generator sẽ tiếp tục được huấn luyện để tiến về điểm đó, nhưng không may đó chỉ là một cực trị địa phương. Mô hình hiện tại sẽ cho ra kết quả rất tốt khi chạy trên hàm mất mát, nhưng khi đem ra thực tế thì lại sinh ra những dữ liệu rất kém đa dạng, đây cũng là hiện tượng overfit trong GANs.

Để tránh hiện tượng trên, đã có nhiều phương pháp được đề xuất. Đầu tiên là phương pháp thực nghiệm được nêu ra trong bài báo rằng, với mỗi vòng lặp, cần cập nhật đạo hàm của D và G luân phiên nhau, không nên huấn luyện từng cái tới khi đạt cực trị (Tuy nhiên phương pháp này lại bị bất cập rằng, nếu như hiện tượng Gradient Vanishing xảy ra thì Mode collapse vẫn xảy ra như bình thường)

Một trong các phương pháp được đề xuất kế tiếp là: sử dụng bài toán phân cụm để phân chia dữ liệu huấn luyện và thêm chiều cho không gian ẩn (latent space) để có thể tạo ra dữ liệu giả được phân loại.



Hình 8. Mô hình GANs kèm classifier để giải quyết vấn đề Mode collapse

Bên cạnh đó còn nhiều phương pháp được đưa ra để giải quyết vấn đề Mode collapse như: Thêm một lớp mạng để ánh xạ dữ liệu gốc thành noise sau đó phân tích đánh giá (VEEGAN), huấn luyện mô hình bằng nhiều Generators (MGAN), ...

D. Cân bằng Nash

Như đã đề cập ở phần giới thiệu, ý tưởng của mạng GANs khá giống với bài toán Minimax, thuật toán sẽ dừng lại khi Generator và Discriminator đạt cân bằng Nash. Phát biểu toán học của cân bằng Nash là như sau: với mọi bài toán Minimax dạng $\min_x \max_y f(x,y)$, sẽ luôn tồn tại ít nhất 1 điểm hoặc 1 tập (x^*, y^*) mà $f(x^*, y^*) \geq f(x, y) \forall x \in X, y \in Y$. Hay phát biểu bằng lời là: cân bằng Nash xảy ra khi tình trạng của một người chơi không hề thay đổi dù đối phương có thực hiện những thao tác gì.

Giả sử người chơi A muốn cực đại hóa hàm $f(x,y) = xy$, trong khi đó người chơi B lại muốn cực tiểu hóa nó. Lúc này cân bằng Nash xảy ra khi $x = y = 0$, ở trạng thái này thì dù hành động của đối thủ như thế nào thì cũng không ảnh hưởng đến tình trạng hiện có.

Bài toán đặt ra hiện tại là, với sochastic gradient descent liệu có thể tìm ra cân bằng Nash hay không? (dù biết rằng nó luôn tồn tại). Một trong những phương pháp đang được thử nghiệm và đạt được một số thành tựu là sử dụng các thuật toán Heuristic, để đánh giá lại mô hình huấn luyện nếu nó đi chệch hướng so với trung bình các bước nhảy trước, từ đó giúp tìm được điểm hội tụ của cân bằng Nash.

V. Ứng dụng của GANs trong thực tế

Dựa vào những gì đã được chứng minh trên, Vanilla GANs (GANs đời đầu) rõ ràng có rất nhiều vấn đề về cả mặt toán học và trên thực nghiệm. Tuy nhiên, không thể phủ nhận rằng, các biến thể của GANs dựa trên các hướng đi như Heuristic, thay đổi công thức toán học, thay đổi kiến trúc mạng đã khiến cho GANs có được ứng dụng cực kỳ mạnh mẽ.

Sau đây là một số ứng dụng của GANs tại thời điểm hiện tại:

- **Image synthesis:** một trong những ứng dụng lớn nhất hiện nay của GANs đó là tạo ra những bức ảnh giả rất giống thật, dựa trên đầu vào có thể là noise (như trong bài toán ở trên).
- **Image to Image:** bài toán chuyển đổi ảnh gốc qua một dạng ảnh khác (chân dung $\rightarrow$ anime, ảnh chụp $\rightarrow$ tranh vẽ, ...).
- **Super-resolution:** tạo ra các bức ảnh với độ phân giải cao hơn, ứng dụng vào các lĩnh vực như phục hồi ảnh, điều tra, ...

Hình 9. Ứng dụng image to image của GANs, giúp tạo ra các kiểu hình ảnh từ bức ảnh gốc Mona Lisa theo phong cách các bức họa nổi tiếng khác.

Tham khảo

Bài báo từ các hội nghị:

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in Advances in Neural Information Processing Systems, 2014, pp. 2672–2680. [2] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath, “Generative Adversarial Networks: An Overview,” in IEEE-SPM, APRIL 2017

Bài báo tạp chí:

[3] Duhyeon Bang, and Hyunjung Shim, “Improved Training of Generative Adversarial Networks using Representative Features” [4] YANG WANG, “A MATHEMATICAL INTRODUCTION TO GENERATIVE ADVERSARIAL NETS (GAN)”

Các trang web khoa học:

[5] GAN — Why it is so hard to train Generative Adversarial Networks! web site: https://jonathan-hui.medium.com/

[6] The math behind GANs (Generative Adversarial Networks). web site: https://towardsdatascience.com/