Instituto Tecnológico y de Estudios Superiores de Monterrey

Generative adversarial network

Materia:

Inteligencia artificial avanzada para la ciencia de datos II (Gpo 50)

Integrantes:

José Antonio Miranda Baños

Campus: Querétaro.

Fecha: 1 Nov de 2024

Generative adversarial network for D&D characters

José Antonio Miranda Baños
IRS, Instituto Tecnológico y de Estudios Superiores de Monterrey (Campus QRT)
Qrt, Mx
A01611795@tec.mx

Abstract—

Keywords— ML, IA, Convolución, Generativo, Pytorch

I. INTRODUCCIÓN

El presente reporte tiene como objetivo principal examinar y documentar el proceso de creación de un avatar de personaje aleatorio para el juego de rol Dungeons & Dragons (D&D) mediante el uso de una red Generative Adversarial Network (GAN). Este proyecto aborda un desafío significativo, ya que la creación de avatares en D&D implica una gran variedad de posibilidades y combinaciones de características, reflejo de la rica diversidad en las razas, clases, personalidades y aspectos visuales que definen a los personajes en este juego. La complejidad de representar esa diversidad a través de un modelo GAN hace que este proyecto sea tanto ambicioso como técnicamente desafiante. Para este reporte nos basaremos en la implementación que muestra Pytorch [1]

Las GAN, conocidas por su capacidad para generar imágenes sintéticas a partir de distribuciones de datos, no son modelos triviales de entrenar. Requieren un equilibrio delicado entre el generador y el discriminador, dos redes que compiten entre sí para mejorar en cada iteración. Entrenar una GAN con éxito demanda un enfoque cuidadoso para ajustar hiperparámetros, gestionar la calidad de las imágenes y evitar problemas comunes como el colapso de modo, donde el modelo puede estancarse y generar imágenes repetitivas. En este sentido, la meta de este reporte no solo consiste en crear un avatar específico, sino en explorar las capacidades de la GAN para representar fielmente la diversidad de personajes en D&D, evaluando el rendimiento y las limitaciones de la red en este contexto.

A lo largo de este trabajo, documentaré los diferentes experimentos realizados, los ajustes en el modelo y los resultados obtenidos, reflejando los avances y obstáculos encontrados. Cabe mencionar que, aunque hasta el momento no he logrado obtener un avatar completamente satisfactorio para usar como mi propio personaje en la campaña semanal de los sábados, los resultados alcanzados aportan perspectivas valiosas para mejorar el modelo. Este proceso iterativo busca no solo perfeccionar el avatar final, sino también contribuir al entendimiento de cómo las GAN pueden utilizarse en aplicaciones creativas complejas y orientadas a un público específico, como es el caso de los jugadores de D&D.

II. ¿QUE ES UNA GAN?

Una GAN (Generative Adversarial Network) es un framework de Deep Learning que se enfoca en aprender una distribución de datos para luego generar nuevos ejemplos que se ajusten a esa distribución, de modo que los resultados producidos puedan ser indistinguibles de los datos reales. Este tipo de modelo es ampliamente utilizado en la generación de imágenes sintéticas, creación de arte digital, mejora de resolución de imágenes, y otras aplicaciones en las que es importante que los datos generados se asemejen a muestras reales. La arquitectura de una GAN está compuesta por dos modelos que interactúan entre sí: el **Generador** y el **Discriminador**, cuya dinámica competitiva es esencial para el proceso de aprendizaje.

A) Generador

El Generador es la parte de la GAN responsable de crear nuevos datos, en este caso, imágenes que simulan las del conjunto de entrenamiento. Su objetivo es generar muestras que imiten de manera realista las características visuales de los datos originales, de tal forma que resulten creíbles para el Discriminador. Al principio del entrenamiento, las imágenes generadas suelen ser de muy baja calidad y fácilmente reconocibles como "falsas" por el Discriminador. Sin embargo, a medida que el entrenamiento avanza y el Generador recibe retroalimentación, sus imágenes van mejorando y se vuelven progresivamente más parecidas a las reales.

B) Discriminador

E El Discriminador, por su parte, actúa como un "juez" que clasifica las imágenes que recibe como "reales" o "falsas". Su tarea es determinar si una imagen pertenece o no al conjunto de entrenamiento original. En términos prácticos, el Discriminador evalúa cada imagen generada por el Generador y asigna una probabilidad de que sea auténtica. Durante el entrenamiento, el Discriminador también se entrena con imágenes reales para mejorar su habilidad de distinguir entre los datos del conjunto de entrenamiento y las falsificaciones generadas por el Generador.

Para este punto se intuye la dirección que toma este modelo. Esta dinámica competitiva entre el Generador y el Discriminador es lo que define la esencia de una GAN. A medida que el Generador intenta crear imágenes suficientemente convincentes para "engañar" al Discriminador, este último se vuelve cada vez más efectivo en identificar imágenes falsas. Este ciclo de mejora mutua se convierte en un juego en el que el Generador y el Discriminador compiten hasta alcanzar un punto de equilibrio. En teoría, cuando ambos modelos han alcanzado un estado de aprendizaje óptimo, las imágenes generadas son tan realistas que el Discriminador tiene dificultades para distinguirlas de las reales.

Sin embargo, debido a esta interdependencia entre los dos modelos, entrenar una GAN puede ser muy complejo y delicado. Existen varios desafíos típicos, como el **colapso de modo**, que ocurre cuando el Generador produce solo un pequeño subconjunto de imágenes similares y no representa toda la diversidad de los datos de entrenamiento. También puede suceder que uno de los modelos, ya sea el Generador o el Discriminador, se vuelva demasiado efectivo, lo cual desequilibra el sistema y frustra el entrenamiento. En algunos casos, el modelo no logra converger y ambos submodelos oscilan sin llegar a un equilibrio, produciendo resultados inconsistentes. Estos problemas comunes hacen que el entrenamiento de una GAN requiera una cuidadosa optimización y monitoreo constante para evitar que las imágenes generadas se vuelvan aleatorias o irreales.

III. DATA SET

Ahora que sabemos cómo funcionan las GAN, el primer paso para construir una, seria encontrar un conjunto de datos de preferencia no aleatorios para entrenar nuestra GAN, mientras más mejor. En nuestro caso ocuparemos un data set del proyecto “Dungeons and Diffusion”.[2]

Este data set viene con alrededor de 2.5k imágenes de 512x512 de diversas especies del juego. Nuestro objetivo será poder crear un personaje que pueda encajar dentro de este set de entrenamiento por lo que podemos hacer una lista de las cosas que esperamos que tenga este personaje basado en los rasgos comunes entre todas las imágenes.

- 2 extremidades
- 1 cabeza
- Color diferente al fondo
- 1 o varias armas

Si por el contrario te gustaría generar un personaje mas especifico te recomendamos generar GAN con sets de entrenamiento más cercanos a lo que estas buscando o revisar la implementación de ‘Justin’ en ‘Dungeons and Diffusion’.

IV. IMPLEMENTACIÓN

En esta sección encuentras la arquitectura de los modelos del generador y el discriminador y los hiper-parámetros iniciales para el entrenamiento, al igual que varias problemáticas que surgieron a la hora de implementarlos. Esta implementación, en Pytorch, la podrás encontrar en la carpeta de ‘Code’ dentro del repositorio. [Repositorio]

A. Hiper-Parámetros

- Workers: 20

  Numero de núcleos que se dedicaran a cargar y transformar las imágenes del set de entrenamiento. (Esto nos ayudo a subir la velocidad de entrenamiento sin embargo 20 puede ser un numero excesivo, te recomendamos usar 4 o 6 y revisar las especificaciones de tu procesador).

- Batch size: 128

  El número de imágenes por Batch para el entrenamiento, Tener un numero alto se ve reflejado en mas uso de memoria y menos iteraciones. Por lo que tener un buen balance entre el Batch size y el Num Epoch para entrenar por varias iteraciones. Un buen balance da como resultado imágenes más variadas.

- Image size: 128 px

  Este parámetro sirve para escalar las imágenes del set de entrenamiento. Principalmente ocupamos un tamaño de 64 pixeles, decidimos cambiarlo buscando mejorar la calidad de las imágenes. Cambiar este numero implica cambiar la arquitectura de tu Generador y Discriminador. Recomendamos ocupar 128 pixeles para no perder tanto detalle, sin embargo, esto se verá reflejado en la memoria, por lo que ocupar tamaños más pequeños sería útil en caso de poca memoria.

- Nc: 3

  El número de canales de las imágenes, en nuestro caso es RGB, este afecta principalmente a la hora de hacer la convolución.

- Nz: 100

  La entrada del Generador, en este caso es un vector de ruido, realmente no hemos experimentado tanto con este valor, creemos que aumentarlo nos dará imágenes mas detalladas. En Futuras iteraciones jugaremos más con este valor.

- Ngf: 128

  El tamaño del mapa de Features del generador, aumentarlo genera imágenes mas detalladas y un poco más nítidas, sin embargo el costo en memoria es muy alto y realmente no ganamos tanto detalle, por lo que te recomendamos dejarlo entre 128 y 64.

- Ndf: 128

  El tamaño del mapa de Features del discriminador, aumentarlo hace que el discriminador se enfoque en mas detalles, del mimo modo aumentarlo tiene un costo muy alto en memoria, por lo que dejarlo en 128 o 64 es lo ideal.

- Num Epochs: 250

  El número de épocas por las cuales se va a entrenar, aumentar el número de épocas aumenta el numero de Iteraciones, lo cual permite que le modelo genere imágenes que podrían pasar por imágenes dentro del set de entrenamiento. Sin embargo tener muchos Epochs tiende a quemar las imágenes (Generar imágenes en gris).

- LrD: $2e^{-5}$

  Este es el learning rate del discriminador, en nuestras primaras iteraciones vimos que el discriminador era muy bueno y el generador no podía seguirle el paso, por lo que decidimos bajar su learning rate al punto en que hubiera un balance entre ambas redes. Tener un discriminador muy demasiado bueno hace que el generador no pueda aprender.

- LrG: $2e^{-4}$

  Este es el learning rate del Generador, es más alto que el del discriminador para compensar la ventaja que este tiene. Subir este valor hace oscilar la perdida de la red del generador lo que genera imágenes 'raras' que se confunden más con ruido que con otra cosa.

- Beta: 0.5

  Hiper parámetro del optimizador Adam.

- Ngpu: 1

  Especificamos usar la GPU para el entrenamiento.

B. *Discriminador*

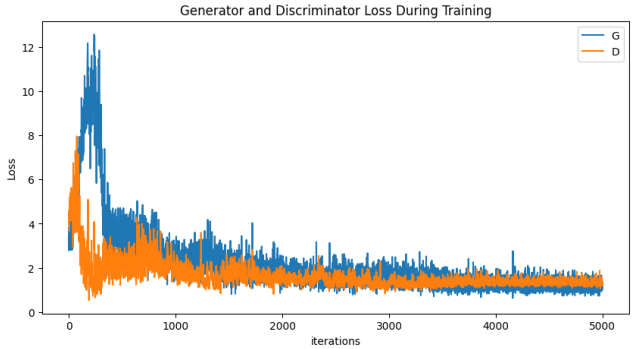
El discriminador se compone de 6 capaz convolucionales, entre cada una hay una capa de LeakyReLu y un DropOut para evitar que el modelo se vuelva muy bueno. Termina con una capa de activación Sigmoide que determina la probabilidad de que una imagen este o no dentro del set de entrenamiento. Esta red toma la salida de la red del Generador por lo que las capaz convolucionales se colocan de forma ascendente.

C. *Generador*

El generador se compone de 6 capaz convolucionales descendente, entre cada capa hay una capa de activación ReLu y una capa de DropOut para evitar que el modelo llegue a engañar demasiado bien al discriminador. Esto no significa que el generador sea bueno, solo que el discriminador es muy malo. Por último una función Tangente hiperbólica para generar valores entre -1 y 1 para generar los valores de los pixeles.

V. Resultados del Entrenamiento

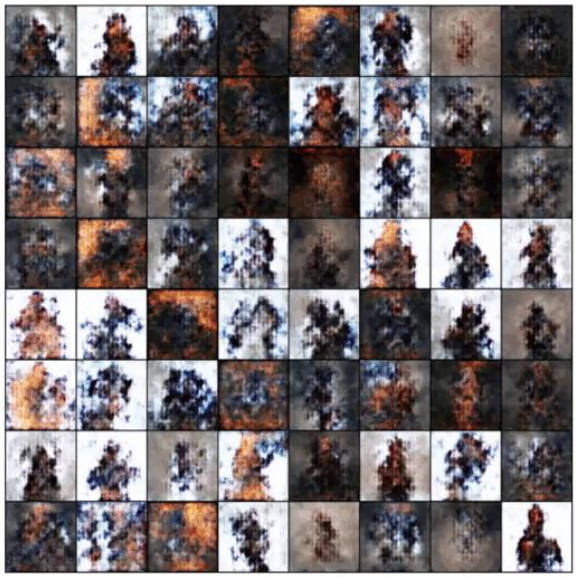
Se entreno según los hiper-parametros descritos en la parte de arriba y obtuvimos resultados relativamente buenos. Estos resultados solo nos indican el desempeño del modelo durante el entrenamiento no significa que las imágenes que estamos generando sean las imágenes que estamos buscando.



Nuestra principal métrica en este caso es la perdida de ambos modelos, podemos interpretar esta como el desempeño de ambos modelos, podemos observar como la red del discriminador en un inicio es superior al generador, pero esta última va mejorando con forme aumentan las iteraciones. Al final el generador logra generar imágenes que logran confundir al discriminador. Esto no significa que las imágenes nos logren confundir a nosotros, sin embargo, a los ojos del discriminador las imágenes generadas podrían formar parte del conjunto de entrenamiento.

VI. RESULTADOS

¿Tal vez la parte más importante de este documento, se logró o no se logró? Es difícil responder esta pregunta, el modelo claramente logra generar imágenes, aun que estas no llegan a tener un parecido visual a las imágenes del conjunto de entrenamiento, para mejorar esto podemos probar varias estrategias. Desde ocupar un escalado de imágenes más amplio, hasta incluir más imágenes o especializar el conjunto de entrenamiento. Una recomendación es hacer transfer learning al modelo de discriminador para que las imágenes queden más claras.



Podemos ver que el modelo logra capturar ciertas características de las imágenes, como su posición en el centro y la silueta humanoide de estas formas, incluso viéndolas desde lejos dan la impresión de ser personajes complejos. El modelo tiene mucho espacio para mejorar.

VII. Mejoras

Esta parte se enfoca en la implementación de varias mejoras para ver como reacciona el modelo. Estas mejoras deberían de mejorar los resultados de este, aunque no es una garantía, puesto que recordemos que el entrenamiento de GANs no es algo tan sencillo. Como adelanto estas mejoras no dieron el resultado que se esperaba y en general empeoraron el modelo.

A. *Image Size :256 px*

Es lógico pensar que si queremos imágenes con mayor detalle valdría la pena generar imágenes con mejor resolución. Esto como un primer enfoque es correcto, sin embargo, el proceso de entrenamiento de GANs mas grandes implica agregar capaz convolucionales para poder abstraer más información de estas imágenes. Esto genera un modelo mas pesado y que consume mas recursos a la hora del entrenamiento.

Implementamos una MidGAN, que vendría a ser la arquitectura mejorada, con una capa convulucional extra para poder procesar imágenes de 256x256. También generamos una BigGAN que realmente no pudimos echar andar por los tiempos de entrega y la falta de recursos, sin embargo esta debería de ser capaz de generar imágenes de 512x512.

B. NZ:100

Mejorar las dimensiones del ruido, genero imágenes un poco mas claras por si solo, aunque aumentar este valor tiene un costo en RAM muy alto por lo que no es recomendado si no se tienen mas de 32 GB de RAM.

Implementar esta mejora junto a la MidGAN, nos presento mas problemas que soluciones. Principalmente por la falta de recursos decidimos dejar este valor en 100. Lo cual nos debería de entregar imágenes de menor calidad, que se verá compensado con la resolución de la MidGAN.

C. Filtros: 64

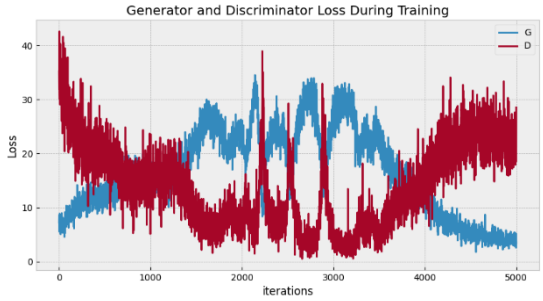
Aumentar los filtros debería aumentar las features que se capturan para la generación y para la discriminación de las imágenes. Sin embargo, duplicar estos filtros es muy pesado para el entrenamiento y por lo que aumentar los filtros de forma indiscriminada requiere un equipo con mayor memoria en su tarjeta gráfica.

Para la MidGAN tuvimos que reducir la cantidad de filtros a 64 para poder mantener un batch size alto de 128 imágenes para poder capturar la diversidad de nuestro dataset de DnD.

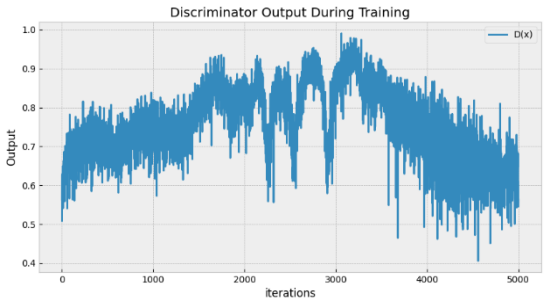
Estas fueron las mejoras que se aplicaron al modelo y a los parámetros del entrenamiento con el fin de generar imágenes mas detalladas.

VIII. Resultados de Mejoras

Entrenamos el modelo por 250 épocas y creamos un programa que nos permitiera correr predicciones haciendo uso del mejor modelo determinado por las gráficas de entrenamiento. Como resultados del entrenamiento obtuvimos las siguientes métricas.

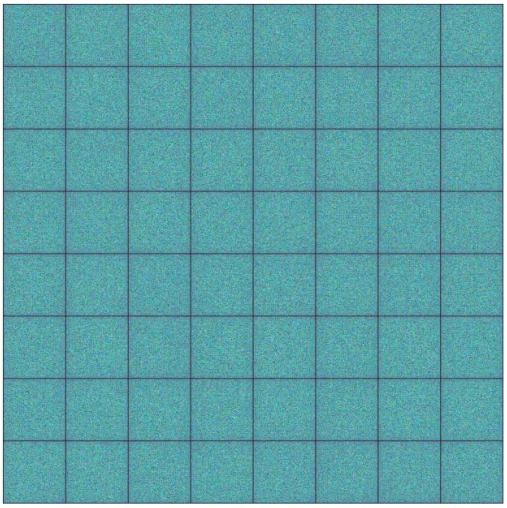


Las Graficas de Perdida durante el entrenamiento nos muestran que tan dominantes eran las redes una sobre la otra. Esta grafica nos muestra un desbalance, al inicio el generador logro engañar de forma exitosa al discriminador aun que este logra aprender y mejorar alrededor de las mil iteraciones. El generador se vuelve a sesgar alrededor de las cuatro mil iteraciones.



La grafica de la salida del Discriminador nos indica que tan buen rendimiento tiene el discriminador, en otras palabras, la probabilidad de que este sepa si la imagen es falsa o verdadera. Una grafica ideal de este tipo debería mantenerse en valores de 0.5 lo que quiere decir que la imagen tiene un 50% de probabilidades de pertenecer a la data set. Esta grafica se mantienen alta la mayor parte del entrenamiento lo que quiere decir que las imágenes generadas por el Generador no logran engañar del todo al Discriminador, Sin embargo con forme pasaban las iteraciones la gráfica se acercaba al 0.5 lo cual podría sugerir que más épocas mejorarían la calidad de las imágenes generadas en el modelo.

Las imágenes generadas por la mejor versión de este modelo nos dan como resultados imágenes como estas.



Las cuales son mayormente ruido, pero logran cumplir con las dimensiones que deseábamos.

IX. Conclusión

El entrenamiento de redes generativas adversarias (GANs) implica un alto costo computacional, especialmente al trabajar con imágenes de alta resolución. Esto se debe a la complejidad inherente del modelo y al considerable poder de procesamiento requerido para equilibrar el entrenamiento del generador y el discriminador.

El modelo que mostró los mejores resultados tanto en el entrenamiento como en la comparación de imágenes, fue el primer modelo. Una posible mejora para futuros trabajos podría ser reestructurar el proyecto dividiendo la generación de imágenes en secciones más pequeñas y posteriormente unirlas. Este enfoque permitiría crear imágenes de mayor

tamaño sin comprometer de manera significativa el uso de recursos computacionales

X. Referencias

[1] DCGAN Tutorial — PyTorch Tutorials 2.5.0+cu124 documentation. (n.d.). https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

[2] *0xJustin/Dungeons-and-Diffusion · Datasets at Hugging face*. (n.d.). https://huggingface.co/datasets/0xJustin/Dungeons-and-Diffusion

[3] Radford, A., Metz, L., & Chintala, S. (2015, November 19). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv.org. https://arxiv.org/abs/1511.06434

[4] Deepak, D. (2023, April 9). DCGAN implementation from scratch on FMNIST dataset in PyTorch. *Medium*. https://medium.com/@deepeshdeepakdd2/dcgan-implementation-from-scratch-on-fmnist-dataset-in-pytorch-eeb3481a8ec2

[5] Lornatang/DCGAN-PyTorch: PyTorch implements a Deep Convolution GAN neural network structure. (n.d.). GitHub. https://github.com/Lornatang/DCGAN-PyTorch