Generative Adversarial Network (GAN)

C. G. Rodríguez, *Estudiante MCIA, UAQ.*
Deep Learning

***Resumen*— Las redes generativas adversariales pueden ser de utilidad para una gran cantidad de aplicaciones debido a que pueden aprender a imitar distribuciones de datos de casi cualquier tipo.**

Por lo cual en este trabajo se propone la implementación de este tipo de red para la base de datos Fashion MNIST en el entorno de Google Colaboratory, en donde utilizan dos modelos: un generador que muestrea los datos, y un discriminador que clasifica los datos como reales o generados.

***Temas claves*—GAN, Generador, Discriminador, Deep Learning, TensorFlow.**

I. Introducción

LAS redes generativas adversariales (GAN) son arquitecturas algorítmicas que utilizan dos redes, en donde es necesario su debido enfrentamiento, con el fin de generar nuevas instancias sintéticas de datos, las cuales puedan hacerse pasar por datos reales.

La red adversarial, también conocida como el discriminador, realiza la función de discernir entre los datos de entrada, los cuales pueden ser datos reales o datos falsos, es decir, dadas las características de una instancia de datos, se predice una etiqueta a la que pertenecen esos datos.

Por otra parte, la red generativa o bien codificador genera datos falsos a partir del ruido, los cuales se introducen en el discriminador. Así bien, una vez completado el entrenamiento, la red generativa debe ser capaz de generar datos reales.

En la etapa de entrenamiento, la red se entrena en dos fases diferentes: la primera fase se dedica a entrenar al discriminador, así como para la actualización adecuada de sus pesos; para la segunda fase, se entrena el generador mientras se desactiva el entrenamiento del discriminador.

A continuación, se muestra la siguiente función:

$$\min_G \max_D \; E_{x\sim pdata(x)}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))]$$

En donde:

- D(x): es la estimación del discriminador de la probabilidad de que la instancia de datos reales x sea real.
- Ex: es el valor esperado sobre todas las instancias de datos reales.
- G(z): es la salida del generador cuando se da el ruido z.
- D(G(z)): es la estimación del discriminador de la probabilidad de que una instancia falsa sea real.
- Ez: es el valor esperado sobre todas las entradas aleatorias al generador.

C. G. Rodríguez estudia la Maestría en Ciencias en Inteligencia Artificial en la Universidad Autónoma de Querétaro(e-mail: gaby96.flores@gmail.com)

II. Desarrollo

Para la realización del presente trabajo se comenzará primeramente por importar el conjunto de datos Fashion-MNIST, el cual consta de un conjunto de entrenamiento de 60,000 imágenes y un conjunto de prueba de 10,000 imágenes, en donde cada una de las imágenes se encuentran en escala de grises de 28x28, asociada con una etiqueta de 10 clases.

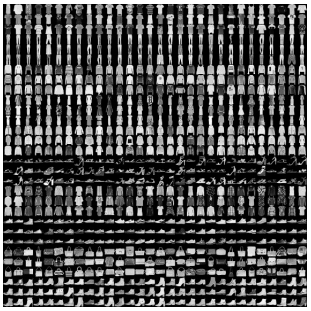

Figura 1. Ejemplos de imágenes de la base de datos propuesta.

A. Modelo

A continuación, se comentará brevemente algunas de las funciones del modelo GAN a implementar.

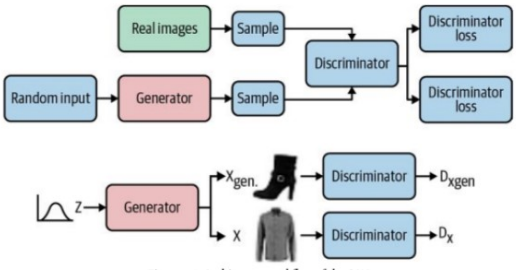

Figura. 2. Arquitectura del modelo GAN.

La función compute_loss calcula siguientes perdidas: la pérdida del discriminador (disc_real_loss) al determinar una imagen real, la pérdida del discriminador (disc_fake_loss) al inferir una imagen generada falsa, la pérdida del discriminador

(disc_loss), la cual es la pérdida total del discriminador; y la pérdida del generador (gen_loss) al medir el error en las imágenes generadas.

```python
def compute_loss(self, x):
    """ passes through the network and computes loss
    """
    # generating noise from a uniform distribution
    z_samp = tf.random.normal([x.shape[0], 1, 1, self.n_z])

    # run noise through generator
    x_gen = self.generate(z_samp)
    # discriminate x and x_gen
    logits_x = self.discriminate(x)
    logits_x_gen = self.discriminate(x_gen)
    ### losses
    # losses of real with label "1"
    disc_real_loss = gan_loss(logits=logits_x, is_real=True)
    # losses of fake with label "0"
    disc_fake_loss = gan_loss(logits=logits_x_gen, is_real=False)
    disc_loss = disc_fake_loss + disc_real_loss

    # losses of fake with label "1"
    gen_loss = gan_loss(logits=logits_x_gen, is_real=True)

    return disc_loss, gen_loss
```

Figura 3. Calculo de pérdidas.

Así bien, para cada una de las perdidas anteriores se utiliza la función gan_loss, la cual se calcula en función de la coincidencia de la salida, es decir, se utiliza una prueba de verdadero/falso: 1 para verdadero y 0 para falso. Esta prueba se utiliza para devolver la diferencia de verdad, 0 o 1.

```python
def gan_loss(logits, is_real=True):
    """Computes standard gan loss between logits and labels
    """
    if is_real:
        labels = tf.ones_like(logits)
    else:
        labels = tf.zeros_like(logits)

    return tf.compat.v1.losses.sigmoid_cross_entropy(
        multi_class_labels=labels, logits=logits
    )
```

Figura 4. Función gan_loss

A continuación, se muestran las arquitecturas para cada una de las subredes.

```python
generator = [
    tf.keras.layers.Dense(units=7 * 7 * 64, activation="relu"),
    tf.keras.layers.Reshape(target_shape=(7, 7, 64)),
    tf.keras.layers.Conv2DTranspose(
        filters=64, kernel_size=3, strides=(2, 2), padding="SAME", activation="relu"
    ),
    tf.keras.layers.Conv2DTranspose(
        filters=32, kernel_size=3, strides=(2, 2), padding="SAME", activation="relu"
    ),
    tf.keras.layers.Conv2DTranspose(
        filters=1, kernel_size=3, strides=(1, 1), padding="SAME", activation="sigmoid"
    ),
]
```

Figura 5. Arquitectura Generador.

```python
discriminator = [
    tf.keras.layers.InputLayer(input_shape=DIMS),
    tf.keras.layers.Conv2D(
        filters=32, kernel_size=3, strides=(2, 2), activation="relu"
    ),
    tf.keras.layers.Conv2D(
        filters=64, kernel_size=3, strides=(2, 2), activation="relu"
    ),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=1, activation=None),
]
```

Figura 6. Arquitectura del discriminador.

B. Resultados del entrenamiento

En la siguiente tabla se puede visualizar los resultados obtenidos en la etapa de entrenamiento.

TABLA 1
RESULTADOS OBTENIDOS.

Época	Pérdida discriminador	Pérdida generador
49	0.7572035789489746	1.6796249151229858

Figura 7. Resultados obtenidos.

C. Análisis de los resultados

Al analizar la tabla anterior, se pude observar que se obtuvo un mayor valor de perdida por parte del generador. Esto puede significar que el discriminador pudo discernir correctamente entre las imágenes reales y las imágenes falsas.

D. Conclusiones

La realización de esta tarea se enfocó en aprender un poco más acerca de las redes generativas adversariales (GAN) por medio de la implementación del modelo propuesto, en donde se pudo observar la importancia de mantener el equilibro entre el discriminador y el generador, debido a que es posible que si no se cuenta con un equilibrio adecuado el discriminador termine aprendiendo más rápido que el generador o viceversa. Por el contrario, si se cuenta con un buen equilibrio se obtendrá un discriminador realmente bueno, así como un buen generador.

Así bien, las redes generativas adversariales (GAN) ayudan a mejorar la variación de los datos de salida y evitan que el modelo se quede estancado en un solo tipo de imagen.

III. Referencias

[1] Kashif, Moda-MNIST. Recuperado el día 19 de Mayo de 2022 de https://github.com/zalandoresearch/fashion-mnist

[2] Lanham M. (2020) Practical AI on the Google Cloud Platform. Recuperado el día 20 de Mayo de 2022 de https://books.google.com.mx/books?id=wDgEEAAAQBAJ&pg=PT283&lpg=PT283&dq=generator+%3D+%5B+tf.keras.layers.Dense(units%3D7+*+7+*+64,+activation%3D%22relu%22),+tf.keras.layers.Reshape(target_shape%3D(7,+7,+64)),+tf.keras.layers.Conv2DTranspose(+filters%3D64,+kernel_size%3D3,+strides%3D(2,+2),+padding%3D%22SAME%22,+activation%3D%22relu%22+),+tf.keras.layers.Conv2DTranspose(+filters%3D32,+kernel_size%3D3,+strides%3D(2,+2),+padding%3D%22SAME%22,+activation%3D%22relu%22+),+tf.keras.layers.Conv2DTranspose(+filters%3D1,+kernel_size%3D3,+strides%3D(1,+1),+padding%3D%22SAME%22,+activation%3D%22sigmoid%22+),+%5D&source=bl&ots=GDJjC0KeEh&sig=ACfU3U2JDLEvhVZBeuPUktrZvJzJGeqKtQ&hl=es&sa=X&ved=2ahUKEwiwkc3EmO_3AhWvDkQIHYIcACgQ6AF6BAgDEAM#v=onepage&q&f=false

[3] Martínez Heras, J. (2020) Redes Neuronales Generativas Adversarias (GANs) Recuperado el día 19 de Mayo de 2022 de https://www.iartificial.net/redes-neuronales-generativas-adversarias-gans/