UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
PROGRAMA DE POSGRADO EN INGENIERÍA
FACULTAD DE INGENIERÍA
PROCESAMIENTO DIGITAL DE SEÑALES

GENERACIÓN DE AUDIO A PARTIR DE LENGUAJE NATURAL

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN INGENIERÍA

PRESENTA:
JORGE DAVID LÓPEZ AYALA

TUTOR
DR. CALEB ANTONIO RASCÓN ESTEBANÉ
INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y SISTEMAS

CIUDAD DE MÉXICO, DICIEMBRE 2023

Dedicatoria ...

A Ceci e Isra, porque más que un techo me dieron un hogar donde pertenecer.

A María y Jorge, mis padres, por ser ese lugar seguro cuando todo lo demás parecía ser un caos.

A Dalí, te amo, pequeño.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Planteamiento del problema

Dentro del ámbito de creación de piezas musicales, existe un campo llamado *producción musical*, el cual se encarga de analizar los patrones y tendencias que garanticen la aceptación generalizada de dichas obras [1]. Esta disciplina comprende tópicos como la selección de intérpretes, estilo musical, duración de la obra, tipos de sonidos a utilizar, entre otros parámetros relevantes. En este trabajo se busca hacer mención especial y resaltar la importancia de la selección de sonidos a utilizar dentro de una obra. Aunque se cuente con las métricas correctas para el éxito de la pieza, el uso de sonidos que no son acordes con el contexto de la demografía a la cual se enfoca el producto puede ser determinante al momento de la exitosa *viralización* de la pieza.

El tipo de aproximación que se busca hacer aquí pretende un flujo más orgánico para el usuario. Actualmente dentro del campo de la producción musical, en el área del diseño de sonidos [2], se utilizan osciladores digitales en técnicas como la síntesis aditiva, síntesis sustractiva, por modulación, entre otras. Con estas herramientas, adjunto a modulaciones, aplicación de filtros y otro tipo de técnicas, se busca aproximar el sonido deseado. Sin embargo, eso se vuelve prohibitivo, en términos de conocimiento, para las personas inexpertas en el manejo de señales. Esto vuelve larga y pesada a la curva de aprendizaje, lo cual no necesariamente interesa a la persona que tiene ambiciones menos teóricas y más prácticas para con su obra musical.

Otro gran grupo de productores musicales optan por extraer sonidos de distintas fuentes sonoras, como del mismo ambiente, provocados por la interacción entre objetos, o incluso otras obras musicales, e incluirlas en sus propias obras. A esta técnica, dentro del argot de la producción musical, se le conoce como *sampling* [3], donde básicamente el talento del productor se mide por su capacidad de 1) escoger sus *samples* 2) preprocesarlos y 3) modificarlos/adaptarlos acorde al contexto de su obra. Los puristas de esta escena artística típicamente piensan que cada productor debe de *cazar* sus propios *samples*, de modo que se vuelve parte de la producción el obtener tus sonidos. Si bien un productor musical está siempre pendiente de su entorno para obtener dichas muestras de audio, hoy en día, el estándar es obtener estos sonidos de bibliotecas de sonidos curadas por un tercero (empresas o personas)

[4] [5]. Y en esencia, no es una mala idea tener acceso a una biblioteca sónica, donde las muestras de audio están ordenadas, sean de buena calidad y cuenten con una descripción específica del tipo de sonido del que se trata. El único contraargumento a esta opción, es que la mayoría, si no es que todas las bibliotecas con sonidos de buena calidad tienen un costo de admisión. Es decir, hay que pagar por usar estos sonidos, y muchas empresas han optado por un sistema de suscripción mensual donde además de que solo tienes ciertos créditos para "gastar" en sonidos. Adicionalmente, estos sonidos usualmente están categorizados en niveles, donde un sonido de un nivel más alto cuesta más créditos y, en el momento en que se terminen tus créditos tendrás que pagar aun más. Este tipo de prácticas crean una brecha entre los productores con recursos económicos altos y los productores que no pueden o no quieren adherirse a este tipo de prácticas.

De modo que existe la necesidad y oportunidad de crear una herramienta que permita el acceso, o en este caso generación, de sonidos útiles en un contexto de producción musical. Estos sonidos se pueden considerar como muestras de audio *one-shot* de instrumentos musicales, la cuales se pueden definir como una nota sostenida en el tiempo que corresponda con un instrumento musical, y contenga las características propias de una muestra de audio. Con este trabajo se busca dar solución a esta problemática, brindando una solución que genere muestras de audio con base en los requerimientos del usuario. Es decir, con base en una descripción textual, generar un sonido que sea acorde gramatical y semánticamente. Eso permitiría contar con la facilidad de generar los sonidos de instrumentos deseados "a la carta", y con los recursos propios de una computadora personal.

Antes de entrar en las vicisitudes de este trabajo, es necesario establecer el idioma, los modismos o el *slang* (*short-language*) que se maneja en este documento. Esta aclaración tiene cabida para al contextualizar al lector de la lingüística desarrollada en el campo en el que se está por adentrar. La piedra angular en el lenguaje de este documento recae en la definición de ***sample*** **de audio** ya que en el argot de los productores musicales su definición difiere un poco de la proveniente del contexto de estudio de sistemas y señales. La definición formal que se obtiene en un ámbito científico establece que un *sample* es un elemento puntual de información que pertenece a una señal y corresponde a un instante de tiempo determinado [6]. Por otro lado, en el argot musical se considera a un *sample* una pieza de audio de poca duración (algunos segundos) específicamente grabada o es extraída de obras más grandes[3]. Por ejemplo, un *sample* puede ser una trompeta dando una nota sostenida por 5 segundos, un *riff* de guitarra extraído de una canción o el sonido de una puerta golpeando contra la pared. Consecuentemente, dichos audios pueden ser tomados y modificados por un productor musical, y ser utilizados como percusión en una canción, o modificar el tono de una muestra *one-shot* y tener un *piano-roll* entero de un instrumento.

1.2. Motivación

Se busca crear una alternativa que democratice la creación de sonidos que puedan ser utilizados en un contexto de producción musical sin estar limitados a este campo

en específico. Sin conocimientos previos de teoría musical, procesamiento de señales o técnicas de grabación, se podrá acceder a sonidos propios de instrumentos musicales específicos, y el único requerimiento para con el usuario sea un texto descriptivo del sonido que desea (lenguaje natural).

1.3. Objetivos

Generales.

- Desarrollar un sistema capaz de generar muestras de audio *one-shot* que se puedan utilizar para producción musical, diferentes a las muestras con las que fue entrenado, extrayendo las características y combinándolas en una señal resultante, con base en un condicionamiento textual brindado como entrada al sistema.

Específicos.

- Desarrollar un modelo generativo de audio capaz de ser condicionado por *embeddings* de texto.
- Desarrollar las diferentes técnicas de modelos generativos y explorar su contribución en el contexto de campo de *Music Information Retrieval*.

1.4. Contribución

Muy recientemente han salido a la luz proyectos que generan audio a partir de lenguaje natural. Sin embargo, estos proyectos están enfocados a otras tareas como generación de voz, ruidos ambientales, o música con armonías más elaboradas. En este proyecto se propone atacar el problema de la generación de audio a partir de lenguaje natural, para la obtención de muestras *one-shot*. Éstas son de particular importancia en el área de la producción musical, porque hace posible, a partir de una muestra: generar instrumentos enteros; obtener sonidos que no existían previamente (libres de derechos de autor); y brinda una base de desenvolvimiento para la creación de nuevas obras musicales.

1.5. Estructura de tesis

La estructura de esta tesis, brinda un camino fácil de seguir y armado de tal manera que sea natural para el lector envolverse en este campo. En esta introducción, se explicó la necesidad del desarrollo de un trabajo como éste y que es lo que se pretende lograr. El marco teórico, capitulo 2, brinda las herramientas teóricas necesarias para entender la razón de las decisiones tomadas en el desarrollo de las técnicas utilizadas en este trabajo. La metodología, capítulo 3, detalla la manera en la que las señales

fueron representadas en la forma en la que fueron representadas y dibuja un panorama teórico de cada una de las técnicas mencionadas en este trabajo. En el capítulo 4 "conjunto de datos", se explica la estructura de los datos con los que se entrenó a los modelos y se justifica la razón por la que se escogió trabajar con estos datos. En el capítulo 5 "Implementación y entrenamiento", se detallan los parámetros y técnicas de entrenamiento utilizadas para cada una de las técnicas. El trabajo termina con los últimos 2 capítulos, nombrados "Evaluación y Resultados" capítulo 6.1 y "Discusión y Conclusiones" capítulo 7 respectivamente, donde se obtienen y se discuten métricas objetivas y subjetivas con las que se evaluaron los modelos abordados en el desarrollo de este trabajo. También en estos capítulos se analizan los resultados obtenidos y se proponen nuevas perspectivas a la solución del planteamiento de este problema.

Capítulo 2

Marco teórico

Cuando se aborda un proyecto, es necesario conocer el estado del arte y las *materias primas* con las que se va a trabajar, de tal forma que se conozcan sus propiedades, defectos, limitaciones y fortalezas. Para el propósito particular de este trabajo, la *materia prima* es el audio, pero no cualquier tipo de audio, este trabajo hace énfasis en sonidos provenientes de instrumentos musicales, de tal manera que estas señales se pueden clasificar más cerca de la música que de un ruido ambiental. De acuerdo con Meinard Müller en su libro *Fundamentals of Music Processing* [7], la música puede representarse de distintas maneras, donde el autor resalta tres de ellas. Müller comienza hablando de las *representaciones en partituras*, como notas musicales con tono o la notación musical occidental. En segundo lugar nombra a las *representaciones simbólicas*, las cuales describen a la música mediante entidades que tienen un significado explicito musical y que pueden ser analizadas por una computadora cuando se encuentran en un formato digital, tales como representaciones en un piano, representaciones MIDI o representaciones de puntuación. Como última forma de representación se menciona que la música es mucho más que una descripción simbólica, y aborda el sonido desde un punto de vista físico, introduciendo representaciones como ondas, formas de onda, espectrogramas, dinámicas y timbre, entre otras.

Este tipo de representaciones son particularmente relevantes, dado a que tienen una base matemática que describe el comportamiento de la onda de sonido. Un sonido es generado por la vibración de un objeto, como la cuerda de una guitarra o la piel en un tambor. Dichas vibraciones generan desplazamiento y oscilaciones en las moléculas del aire, resultando en regiones locales de compresión y rarefacción. Estas perturbaciones viajan a través del aire en forma de una onda, hasta que llegan a un dispositivo biológico (persona) o electrónico (micrófono) que interpretan estos sonidos.

Para propósitos de este trabajo, se hará especial énfasis en las fuentes sonoras procesadas a través de un medio digital. El sonido al ser una señal analógica, tiene que ser pasada a través de un *ADC* (*Analog to Digital Converter*, por sus siglas en inglés) para convertirla en una señal digital que los dispositivos electrónicos sean capaces de entender y procesar. Una vez se convierte la señal a su representación digital, ya es posible manipular el audio a placer, y extraer la información necesaria para el propósito deseado. El audio cuenta con características y propiedades útiles que

ayudan al procesamiento y generación de algoritmos alrededor de las mismas, de tal forma que se puede hablar de dos grandes tipos de características, las *características en el dominio del tiempo* y las *características en el dominio de la frecuencia*, las cuales se explorarán más a fondo a continuación.

2.1. Propiedades de las señales de audio

2.1.1. Frecuencia y tono (*Pitch*)

De acuerdo con Müller en [7], si se tiene una fuente sonora que se repite de una forma regular y alternante, la señal resultante es llamada periódica. En este caso, el periodo de una onda es el tiempo que se requiere para que se complete un ciclo. La frecuencia, medida en **Hertz**, (Hz) es el recíproco del periodo. En términos de sonido, para una onda sinusoidal, mientas más alta sea la frecuencia, más agudo será el sonido. Una señal sinusoidal puede ser considerada como el prototipo de una realización acústica de una nota musical, de tal forma que suele decirse que una señal sinusoidal es un *sonido armónico* o *tono puro*. La noción de frecuencia está íntimamente relacionada con lo que determina el tono o *pitch* del sonido. El *pitch* es un atributo subjetivo del sonido, dado que en sonidos complejos con distintas frecuencias este termino puede ser especialmente ambiguo, aunque la relación entre ondas con una sola frecuencia y el *pitch* este clara.

2.1.2. Timbre

Además de las características dinámicas y de duración de una fuente sonora, existe el termino de timbre o color del tono. Esta característica permite a la persona que lo escucha distinguir entre instrumentos musicales, incluso si está siendo tocado el mismo tono *pitch* al mismo volumen [7]. Sin embargo, el concepto de timbre es complicado de entender debido a su ambigüedad, ya que un sonido puede describirse como cálido, suave, oscuro, cálido, etc. Una forma de hacer una abstracción de este concepto es el sonido del piano, cuando se toca un tecla, el sonido generado es más que los tonos necesarios para generar ese tono, a esas características que le dan el *carácter* de piano se le llama timbre.

2.2. Propiedades de las señales de audio digitales

De acuerdo con *Analog Devices* en [8], un convertidor analógico a digital (ADC), toma una señal contínua y la convierte en su representación digital. Esto lo hace tomando muestras que representan la amplitud de la señal en puntos específicos del tiempo. De esta manera, dicha señal adquiere propiedades que ayudan a describirla y procesarla. A continuación se describen brevemente algunas de ellas.

2.2.1. Muestra y *Frame*

De acuerdo con Mathworks [9], una muestra es un valor o un grupo de valores (en el caso de una señal multi-canal) que toma una señal en un momento determinado. Y un marco o *frame* por sus siglas en inglés, es un vector o una matriz (en el caso de una señal multi-canal) de muestras almacenadas pertenecientes a instantes de tiempo consecutivos apiladas una junto a otra.

2.2.2. Tasa de muestreo (*Sample Rate*)

La tasa de muestreo o *sample rate* por sus siglas en inglés es el número de muestras que se toman por segundo. Esta propiedad es particularmente relevante, ya que de no escogerse adecuadamente, la representación digital de la señal es susceptible a un tipo de distorsión llamada *aliasing* [10]. Ésta ocurre cuando la frecuencia de la señal es mayor a la mitad de la frecuencia de muestreo, provocando que las muestras no representen adecuadamente la señal de entrada. Este fenómeno fue descrito por Nyquist y Shannon [11], y otra interpretación que se le puede dar, es que la tasa de muestreo tiene que ser mayor al doble de la frecuencia más alta de la señal.

2.3. Características del audio

Probablemente la palabra *característica* pueda llegar a sonar un poco ambigua, dado que es un término tan general que puede confundirse con las propiedades de las señales. Pero, en el contexto de las señales de audio, y de acuerdo con Valerio Velardo [12], una característica se refiere a representaciones extraídas a partir de la señal, y que nos sirven para describir el sonido, de modo que diferentes características capturan aspectos diferentes de la fuente sonora. Dichas características pueden ser categorizadas de formas tales como su nivel de abstracción, su alcance temporal, aspecto musical, dominio de la señal, etc.

Dada la naturaleza de este trabajo, categorizarlas con base en el dominio de la señal resulta más adecuado, ya que son más descriptivas en un sentido físico y tienen fundamentos matemáticos que son propios del enfoque que se le dio a está investigación.

2.3.1. Características en el dominio del tiempo

En el dominio del tiempo, la señal de audio es representada como una forma de onda, donde se aprecia el cambio de amplitud de la señal a lo largo del tiempo como se puede observar en la figura 2.1, de tal forma que son visibles todos los *eventos* que ocurren en la señal en función del instante de tiempo que se especifique. Algunas de las características del audio en el dominio del tiempo más relevantes para este trabajo son la envolvente de amplitud (*Amplitude Envelope*, por sus siglas en inglés), energía RMS (*Root-Mean-Square Energy*, por sus siglas en inglés) o la tasa de cruce por cero (*Zero-cross rate*, por sus siglas en inglés), de las cuales se hará una breve descripción a

continuación. Cabe mencionar que éstas son solo algunas pocas de las características que se pueden extraer de la señal en el dominio del tiempo.

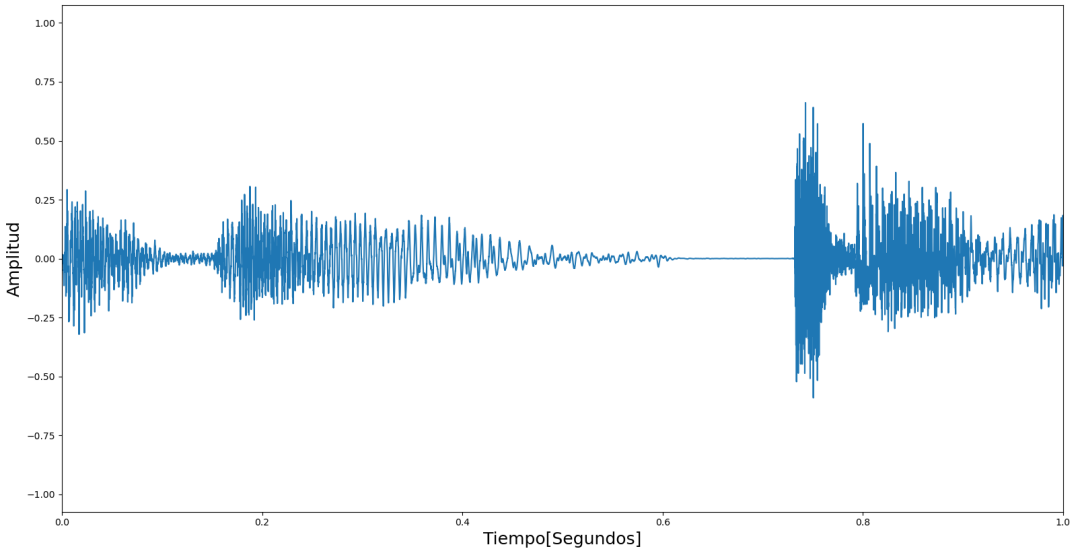


Figura 2.1: Señal de audio en el dominio del tiempo.

2.3.2. Envolvente de amplitud (*Amplitude Envelope*)

Es el valor máximo de amplitud tomado de todas las muestras contenidas en el *frame* [13], la cual se puede expresar matemáticamente como:

$$AE_t = max_{k=t*K}^{(t+1)*K-1} s(k) \tag{2.1}$$

Donde:

- AE_t es la envolvente de amplitud en un *frame* t
- K es el tamaño del *frame*
- $k = t * K$ la primer muestra del *frame* t
- $s(k)$ es la amplitud de la $k - esima$ muestra
- $(t + 1) * K - 1$ es la última muestra del *frame* t

De modo que se obtiene una curva que provee una idea general de la sonoridad de la señal (como se puede observar en la figura 2.2). Sin embargo, es bastante sensible a los valores atípicos (*outliers*, por sus siglas en inglés), lo cual dependiendo de su procesamiento es una mala característica, ya que estas muestras pueden no ser representativas del fenómeno que se tiene. La envolvente de amplitud puede ser usada en tareas de detección de inicio (*Onset detection*), clasificación de genero de música, etc.

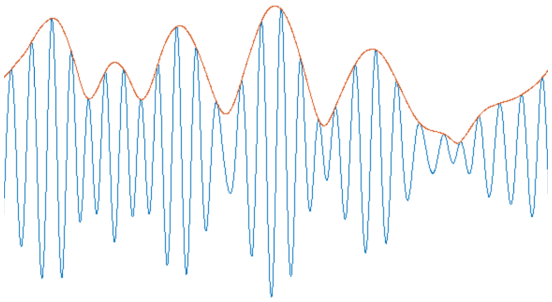


Figura 2.2: En rojo envolvente de amplitud de una señal, en azul señal de interés, imagen tomada de [13].

2.3.3. Energía media cuadrática (*Root-mean-square Energy*)

Esta característica toma el valor medio cuadrático (*RMS*, por sus siglas en inglés) de todos las muestras en un *frame* de la señal [13]. Lo cual se puede denotar con la siguiente formula:

$$RMS_t = \sqrt{\frac{1}{K} \cdot \sum_{k=t \cdot K}^{(t+1) \cdot K-1} s(k)^2} \tag{2.2}$$

Donde:

- RMS_t es el valor medio cuadrático del *frame* t
- $s(k)^2$ es la energía de la k-*ésima* muestra
- $\sum_{k=t \cdot K}^{(t+1) \cdot K-1}$ representa la suma de la energía de todas las muestras en el *frame* t
- K es el número de muestras del *frame*

Dado que es una característica del audio que representa energía, es un buen indicador de la sonoridad de la señal y soluciona el problema de la envolvente de amplitud, ya que es menos sensible a los *outliers* (ver figura 2.3). Esta característica puede ser usada para tareas de segmentación de audio, clasificación de genero de música, etc.

2.3.4. Tasa de cruce por cero (*Zero Crossing Rate*)

Esta característica indica la cantidad de veces que una señal cruza el eje horizontal (que indica cero de amplitud) [13]. Matemáticamente se puede expresar de la siguiente manera:

$$ZCR_t = \frac{1}{2} \cdot \sum_{k=t \cdot K}^{(t+1) \cdot K-1} |sgn(s(k)) - sgn(s(k+1))| \tag{2.3}$$

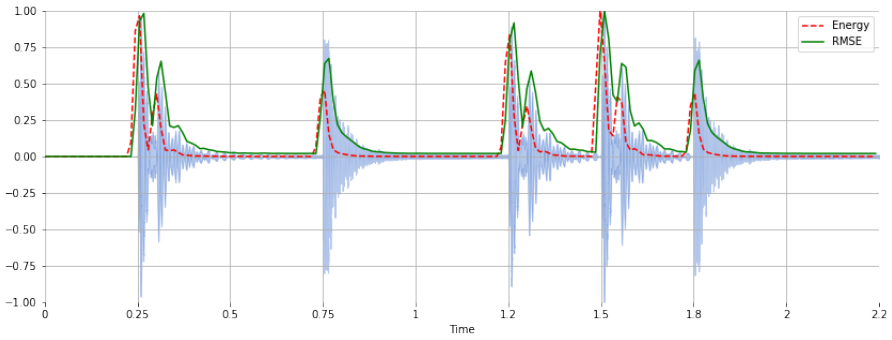


Figura 2.3: Figura compartiva entre la energía de la señal y la señal misma.En rojo la energía, en verde la energía *RMS*, ambas señales tienen una tamaño de ventana de 512 muestras y una longitud de salto (*hop length*) de 256. Imagen tomada de [14].

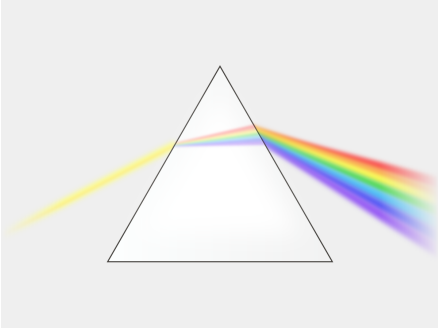
Donde:

- $sgn()$ en la función signo, que regresa un $+1$ si la señal es mayor a cero, -1 si la señal es menor a cero y 0 si la señal es cero
- K es el número de muestras del *frame*

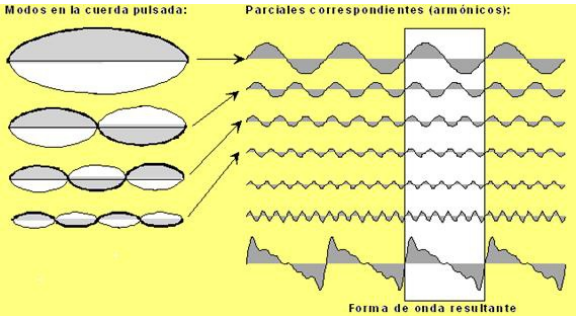
En esta expresión se comparan los valores de amplitud de muestras consecutivas en la señal, de tal manera que, para $|sgn(s(k)) - sgn(s(k+1))|$, si se tienen dos muestras del mismo signo, se obtiene un cero, si se tiene alternancia de signos se obtiene un valor de dos, por lo tanto el valor de normalización es $\frac{1}{2}$. Esta característica es utilizada para tareas de reconocimiento de sonidos de percusión *vs* sonidos con tono, estimación de tono mono fónico, etc.

2.3.5. Señales de audio en el dominio de la frecuencia

Cuando se analiza una señal en el dominio de la frecuencia, se busca obtener una representación matemática y/o pictórica para las componentes en frecuencia (señales sinusoidales) que son contenidas en una señal dada. La recombinación de componentes sinusoidales que sirven para reconstruir la señal original es un fenómeno que puede ser descrito mediante el análisis de Fourier [6]. Para que este concepto quede un poco más claro, se tomará la analogía de un prisma a través del cual pasa un haz de luz, como es presentado en la figura 2.4a. Cuando se descompone una forma de onda en sus componentes sinusoidales, observado en la figura 2.4b, es muy parecido a cuando el prisma separa la luz blanca en diferentes colores. De tal forma que la suma de estás componentes sinusoidales resultan en la forma de onda original. Por otro lado, si cualquiera de esas componentes se excluye, se tendrá una señal completamente distinta.



(a) Prisma separando un haz de luz blanco en sus componentes de color.



(b) Suma de sinusoidales para formar la señal original.

Figura 2.4: Analogía de la descomposición de un haz de luz en sus colores y la descomposición de una señal de audio en sus componentes frecuenciales, imágenes tomadas de [15].

2.3.6. Serie de Fourier y su relación con el sonido

Haciendo un breve repaso al análisis de Fourier, y de acuerdo con [6], podemos recordar que la combinación lineal de exponenciales complejas armónicamente relacionadas de la forma:

$$x(t) = \sum_{k=-\infty}^{\infty} C_k e^{j2\pi k F_0 t} \tag{2.4}$$

Es una señal periódica con un periodo fundamental $T_p = 1/F_0$, por lo tanto se puede pensar a las señales exponenciales

$$\{e^{j2\pi k F_0 t}, k = 0 \pm 1, \pm 2, ...\} \tag{2.5}$$

Como el *bloque constructor* a partir del cual se pueden construir señales periódicas de varios tipos mediante la la elección adecuada de la frecuencia fundamental y los coeficientes $\{C_k\}$. F_0 determina el periodo fundamental de $x(t)$ y los coeficientes $\{C_k\}$ especifican la forma de la onda, y están definidos como:

$$C_k = \frac{1}{T_p} \int_{T_p} x(t) e^{-j2\pi k F_0 t} dt \tag{2.6}$$

En general, los coeficientes C_k de Fourier son coeficientes complejos. Además es fácilmente observable que si la señal periódica es real, C_k y C_{-k} son complejos conjugados. Por lo tanto, si:

$$C_k = |C_k| e^{j\theta k} \tag{2.7}$$

Entonces:

$$C_{-k} = |C_k|^{-j\theta k} \tag{2.8}$$

Consecuentemente, la serie de Fourier también se puede expresar de la siguiente forma:

$$x(t) = C_0 + 2\sum_{k=1}^{\infty} |C_k| \cos(2\pi k F_0 t + \theta_k) \tag{2.9}$$

Donde C_0 es real cuando $x(t)$ es real.

Finalmente, otra forma de expresar la serie de Fourier se puede obtener expandiendo la función coseno en la ecuación 2.9 como:

$$\cos 2\pi k F_0 t + \theta_k = \cos 2\pi k F_0 t \cos\theta_k - \sin 2\pi k F_0 t \sin\theta_k \tag{2.10}$$

Por lo tanto, se puede reescribir como:

$$x(t) = a_0 + \sum_{k=1}^{\infty} (a_k \cos 2\pi k F_0 t - b_k \sin 2\pi k F_0 t) \tag{2.11}$$

Donde:

$$\begin{aligned} a_0 &= C_0 \\ a_k &= 2|C_k| \cos\theta_k \\ b_k &= 2|C_k| \sin\theta_k \end{aligned}$$

Las expresiones 2.4, 2.9, y 2.11 constituyen tres formas equivalentes para la representación en serie de Fourier de una señal periódica real.

Si bien este tipo de análisis permite escudriñar una señal en términos de sus componentes espectrales (en frecuencia), dicha exploración sirve para señales continuas que cumplan con las condiciones de Dirichlet. Sin embargo, el audio no es una señal periódica. Por lo tanto, si permitimos que el periodo de la señal aumente sin limite, cuando el periodo pasa a ser infinito, la señal se vuelve no periódica y su espectro pasa a ser continuo.

Si se considera una señal no periódica $x(t)$ de duración finita, se puede crear una señal periódica $x_p(t)$ con periodo T_p, donde claramente $x_p(t) = x(t)$ en el limite donde $T_p \to \infty$:

$$x(t) = \lim_{T_p \to \infty} x_p(t)$$

La interpretación implica que se debe de obtener el espectro de $x(t)$ a partir del espectro de $x_p(t)$ simplemente tomando el limite de $T_p \to \infty$.

2.3.7. Transformada de Fourier

Ahora habiendo explicado la relevancia del dominio de la frecuencia y su relación con el audio, es necesario introducir una herramienta que permita llevar las señales no periódicas (audio) al dominio de la frecuencia. Dicha herramienta es conocida como la **Transformada en Tiempo Discreto de Fourier** ($DTFT$, por sus siglas en inglés).

Dada una señal en tiempo discreto con energía finita [6], la transformada de Fourier se define como:

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)e^{-jwn}$$

Donde físicamente, $X(\omega)$ representa el contenido en frecuencia de la señal $x(n)$. Es decir, $X(w)$ es la descomposición de $x(n)$ en sus componentes de frecuencia, como se puede observar en la figura 2.5. Para cada parámetro de frecuencia $\omega \in \mathbb{R}$ se obtiene un coeficiente $d_\omega \in \mathbb{R}_{\geq 0}$ y un coeficiente de fase $\varphi_\omega \in \mathbb{R}$. En el caso de que el coeficiente d_ω sea grande, existe una gran similitud entre la señal y la onda sinusoidal ω, y la señal contiene oscilaciones periódicas en esa frecuencia, como es presentado en la figura 2.6.

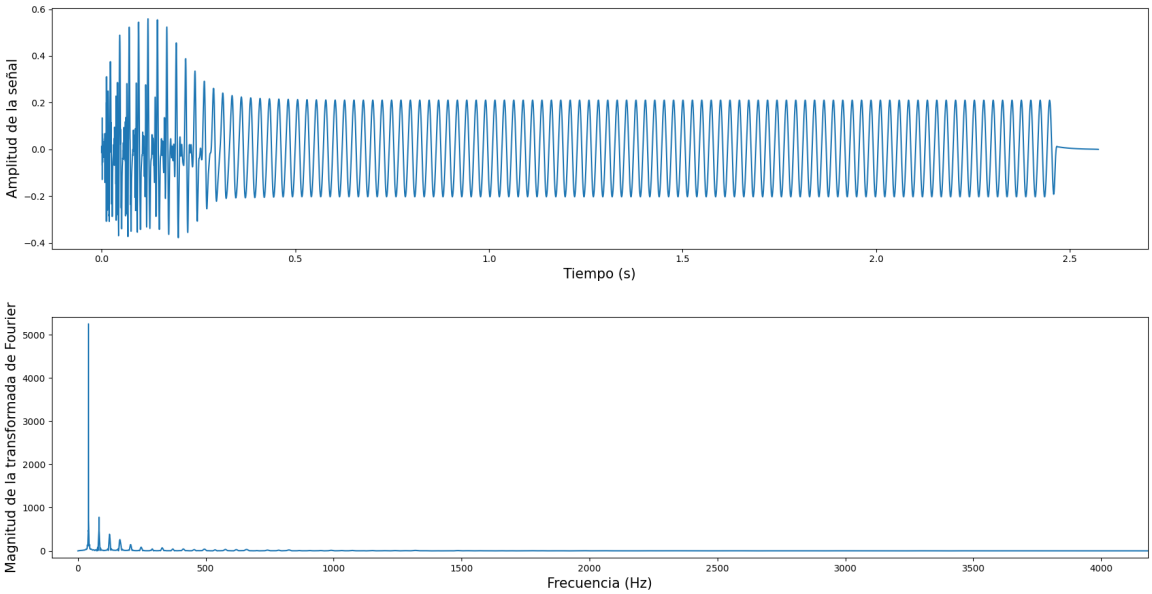


Figura 2.5: Gráfica de señal en el tiempo y su representación en frecuencia.

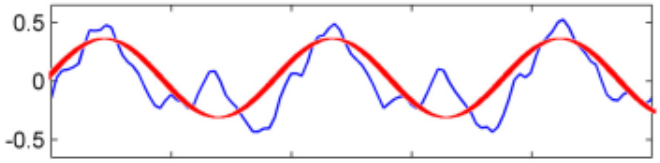


Figura 2.6: Oscilaciones periódicas en frecuencia, imagen tomada de [16].

Con estás figuras y ejemplos se puede ver que la idea principal detrás de la transformada de Fourier es descomponer una señal en sus componentes en frecuencia. En palabras de Hubbard [16], "la señal proporciona información de cuándo ciertas notas son tocadas en el tiempo, pero esconde información acerca de las frecuencias. En contraste, la transformada de Fourier de una obra musical muestra cuales notas (frecuencias) son tocadas, pero esconde información acerca de cuando son tocadas."

2.3.8. Transformada rápida de Fourier (FFT)

Si bien la DFT puede obtener la transformada de Fourier de una señal discreta, la complejidad computacional del producto de una matriz-vector $\mathbf{X} = DFT_N \cdot \mathbf{x}$ requiere $O(N^2)$ sumas y multiplicaciones [7], lo cual implica un coste computacional demasiado grande para la mayoría de aplicaciones. Por ejemplo, si se tiene una señal de 1000 muestras ($N = 10^3$), requeriría un numero de operaciones en el orden de millones ($N = 10^6$). La buena noticia es que la matriz de la DFT es muy estructurada y puede ser explotada cuando se realiza un producto matriz-vector. La idea central consiste en la factorización de la matriz DFT en un producto de $O(logN)$ matrices dispersas, donde cada una pueda ser evaluada con $O(N)$ operaciones. Lo cual llevó a un algoritmo eficiente, llamado Transformada rápida de Fourier (FFT, por sus siglas en inglés), que solo requiere $O(logN)$ sumas y multiplicaciones. El algoritmo fue originalmente encontrado por Gauss en 1805 [17] y luego re-descubierto por Cooley and Tukey en 1965.

El algoritmo de la FFT está basado en la observación de que si se tiene una DFT de tamaño par $N = 2M$ puede ser expresada en términos de aplicar dos $DFT's$ de la mitad del tamaño M. Sea $\sigma_N = \exp(-2\pi i/N)$ la raíz primitiva de la unidad usada en DFT_N de modo que $DFT_N(n.k) = \sigma_N^{kn}$ para $n, k \in [0 : N-1]$. Similarmente, si se define $\sigma_M = \exp(-2\pi i/M)$ de modo que $DFT_M(n,k) = \sigma_M^{kn}$ para $n, k \in [0 : M-1]$. Es evidente que, $\sigma_M = \sigma_N^2$. Sea $\mathbf{x} \in \mathbb{C}^N$ un vector entrada y $\mathbf{X} = DFT_N \cdot \mathbf{x}$ como anteriormente se definió. Entonces para las primeras M entradas $X(k), k \in [0 : M-1]$ se tiene que:

$$X(k) = \sum_{n=0}^{N-1} x(n)\sigma_N^{kn} \tag{2.12}$$

$$= \sum_{n=0}^{M-1} x(2n)\sigma_N^{k2n} + \sum_{n=0}^{M-1} x(2n+1)\sigma_N^{k(2n+1)} \tag{2.13}$$

$$= \sum_{n=0}^{M-1} x(2n)\sigma_M^{kn} + \sigma_N^k \sum_{n=0}^{M-1} x(2n+1)\sigma_N^{k(2n+1)} \tag{2.14}$$

En otras palabras, las primeras M entradas de $\mathbf{X}$ son obtenidas aplicando primero una DFT_M sobre los índices pares de la entrada $\mathbf{x}$, así como también una DFT_M sobre los índices impares de la entrada $\mathbf{x}$. El resulto final se obtiene sumando los dos vectores resultantes , donde el segundo es ajustado por los factores σ_N^k, que también son conocidos como factores de giro (*twiddle factors*). Similarmente, para las últimas M entradas $X(M+k), k \in [M-1]$ se tiene que:

$$X(M+k) = \sum_{n=0}^{N-1} x(n)\sigma_N^{(M+k)n} \tag{2.15}$$

$$= \sum_{n=0}^{M-1} x(2n)\sigma_N^{(M+k)2n} + \sum_{n=0}^{M-1} x(2n+1)\sigma_N^{(M+k)(2n+1)} \tag{2.16}$$

$$= \sum_{n=0}^{M-1} x(2n)\sigma_M^{kn} - \sigma_N^k \sum_{n=0}^{M-1} x(2n+1)\sigma_N^{k(2n+1)} \tag{2.17}$$

Donde se usa $\sigma_N^{(2n+1)} = -1$. Esto muestra que las últimas M entradas de $\mathbf{X}$ son obtenidas bajo el mismo esquema computacional que las primeras $\mathbf{M}$, excepto por el uso de los factores de giro (*twiddle factors*) $-\sigma_N^k$ en vez de σ_N^k. La siguiente factorización de matrices resume el resultado:

$$DFT_M \cdot \begin{pmatrix} x(0) \\ x(1) \\ \vdots \\ x(N-1) \end{pmatrix} = \begin{pmatrix} id_M & \Delta_M \\ id_M & -\Delta_M \end{pmatrix} \begin{pmatrix} DFT_M & 0 \\ 0 & DFT_M \end{pmatrix} \begin{pmatrix} x(0) \\ x(1) \\ \vdots \\ \frac{x(N-2)}{x(1)} \\ x(3) \\ \vdots \\ x(N-1) \end{pmatrix} \tag{2.18}$$

La matriz $id_M = diag(1, 1, ..., 1)$ denota la matriz identidad $(M \times M)$ y $\Delta_M = diag(1, \sigma_N, ..., \sigma_N^{M-1})$ la matriz diagonal $(M \times M)$ que contiene los factores de giro (*twiddle factors*). La reordenación de el vector de entrada en componentes con índice par y componentes con índice impar pueden expresarse con una matriz de permutación adicional. Todo esto junto, lleva a la factorización de la matriz DFT_N en un producto de matrices dispersas y matrices DFT_M de la mitad del tamaño.

2.3.9. Transformada en tiempo corto de Fourier (STFT)

La transformada de Fourier $\hat{f}$ de una señal $f \in \mathbf{l}^2(\mathbb{R})$ describe el contenido en frecuencia de la señal[7]. Comparando la señal con una función exponencial periódica $t \to \exp(2\pi i\omega t)$ resulta en un coeficiente $\hat{f}(\omega)$ que muestra la intensidad global de las oscilaciones en ω Hz que se producen en la señal. Sin embargo, debido a la naturaleza no local de la función de análisis, la información de la frecuencia es siempre es promediada sobre todo el tiempo de la señal. De tal forma que cambios súbitos, variaciones locales tales como comienzo o final de eventos no pueden ser detectados bien por la transformada de Fourier. Para solucionar los inconvenientes de la transformada de Fourier, en 1946 Dennis Gabor presentó una versión modificada de la transformada de Fourier, la **Transformada en tiempo corto de Fourier**(*STFT*, por sus siglas en inglés). Esta transformación es un compromiso entre una representación de tiempo

y frecuencia, determinando el contenido en frecuencia sinusoidal y la fase de secciones locales de la señal mientras cambia en el tiempo.

Para una señal dada, se busca encontrar una transformada que exhiba el contenido en frecuencia de f en un vecindario para cada punto en el tiempo t. La idea básica es considerar solo a una pequeña sección de la señal alrededor del punto t, donde la influencia de un punto dentro de la sección disminuye a la vez que incrementa la distancia de t. Matemáticamente, esta ponderación es modelada con la multiplicación de la señal con una función ventana, que puede ser pensada como una ponderación alrededor de t. En vez de utilizar diferente ventanas para cada punto en t, se usa una sola función que se localiza alrededor del punto $t = 0$. Esta función es entonces recorrida a través del tiempo, como es visto en la figura 2.7. Si $f \in \mathbf{L}^2(\mathbb{R})$ es una señal y $g : \mathbb{R} \to \mathbb{R}$ es una función ventana, entonces $f_{g,t}$ localizada en el punto t está definida como:

$$f_{g,t}(u) := f(u)g(u - t) \tag{2.19}$$

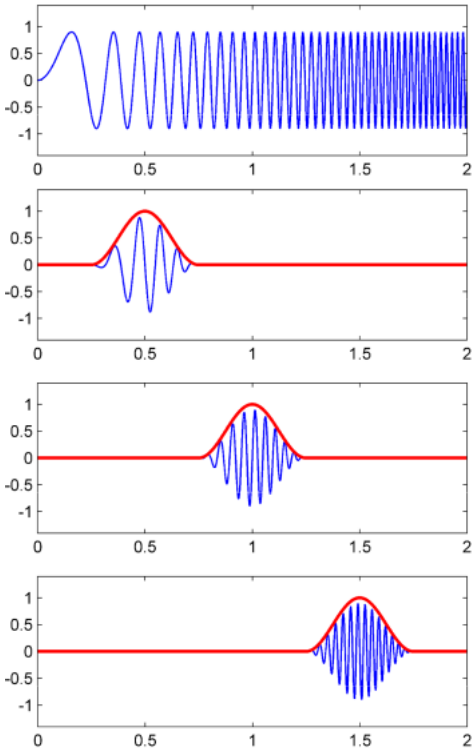


Figura 2.7: Función ventana recorrida en tiempo para hacer la ponderación de datos para la *STFT*, imagen tomada de [7].

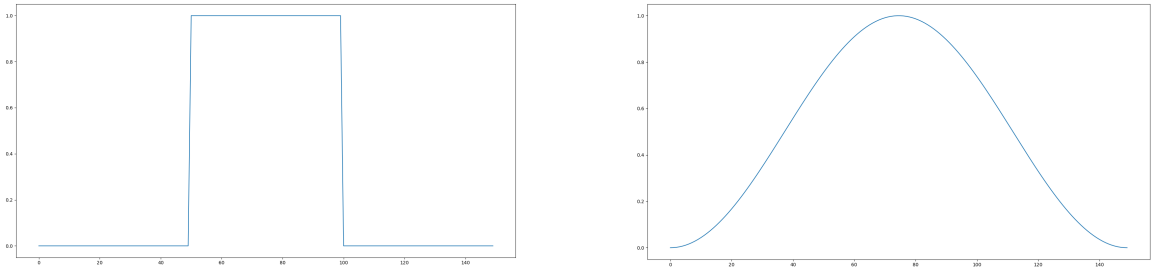
Por lo tanto, dada una señal $f \in \mathbf{L}^2(\mathbb{R})$, así como una función ventana $g \in \mathbf{L}^2(\mathbb{R})$, la Transformada en tiempo corto de Fourier (en tiempo continuo) es una función $\tilde{f}_g : \mathbb{R} \times \mathbb{R} \to \mathbb{C}$ definida por:

$$\tilde{f}_g(t, \omega) := \hat{f_{g,t}}(\omega) = \int_{u \in \mathbb{R}} f(u)\bar{g}(u - t) \exp(-2\pi i \omega u) du \tag{2.20}$$

En otras palabras, $\tilde{f}_g(t, \cdot)$ coincide con la transformada de Fourier de la señal localizada $f_{g,t}$ para una instancia de tiempo fijo $t \in \mathbb{R}$.

2.3.10. Rol de la función ventana

Normalmente, la función ventana se escoge para que tenga un valor de cero fuera de la sección elegida, de tal forma de que cuando la señal se multiplique con la ventana, el producto también será cero fuera de la ventana, puede considerarse como "ver a través de la ventana"[7]. El diseño de la ventana que se toma es fundamental, ya que en función de la ventana que se tenga variará la afectación a la señal. Dada la definición anterior de lo que es una función ventana, la forma más simple de implementar este producto sería con una **ventana rectangular**, mostrada en la figura 2.8a, la cual selecciona la señal en el punto de interés y pone todo lo demás en cero. Sin embargo, usar esta ventana rectangular tiene un inconveniente muy importante. Cuando se hace la suposición de que el periodo de la señal es del tamaño de la señal, es posible siempre y cuando no existan discontinuidades al principio o al final de la señal. Usando la ventana rectangular adhiere estas discontinuidades en los limites de la sección en la señal $f_{g,t}$. Estos cambios abruptos generan *artefactos* debido a interferencias que se extienden a lo largo de todo el espectro de frecuencia, y en lugar de ser parte de la señal original f, estos componentes en frecuencia provienen de las propiedades de la ventana rectangular.



(a) Ventana Rectangular (b) Ventana de Hann

Figura 2.8: Comparación de funciones ventana para procesos de transformación de señales.

Para atenuar estos efectos en los limites, se suele utilizar ventanas que sean no negativas dentro de la sección deseada y que continuamente desciendan a cero hacia la sección de los limites. La ventana que es normalmente utilizada en el procesamiento de señales es la **ventana de Hann** (también conocida como la ventana de Hanning), nombrada así en honor a Julius Von Hann.

La ventana de Hann g es una ventana coseno elevada y se define como:

$$g(u) := \begin{cases} (1 + \cos(\pi u))/2 & \text{si } -0.5 \leq u \leq 0.5 \\ 0 & \text{en otro caso} \end{cases} \tag{2.21}$$

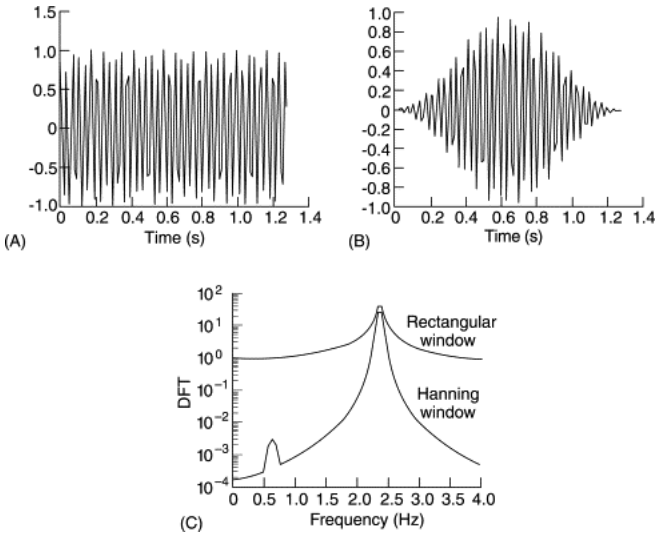


Figura 2.9: Comparación de señal ventaneada con ventana de Hann y ventana rectangular, imagen tomada de [18]

.

Cuando se tiene una caída suave hacia cero en la sección de los limites, los artefactos disminuyen en la transformada de Fourier de la señal ventaneada, como se observa en la figura 2.9. Sin embargo, la desventaja de la ventana de Hann es que introduce frecuencias difusas (*smearing frecuencies*). Como consecuencia, la transformada de Fourier de la sección ventaneada de la señal puede lucir más *suave* de lo que las propiedades de la señal sugieren. En otras palabras, la reducción de los artefactos introducidos por la ventana se consigue a expensas de una peor localización espectral.

2.3.11. Representación de espectrograma

La **STFT** de una señal f, produce para cada punto en el tiempo t y en frecuencia ω un número complejo $\tilde{f}_g(t, \omega)$ [7]. Esta información es, usualmente, representada por medio de un **espectrograma**, el cual es una representación bi-dimensional de la magnitud al cuadrado, denotado de la siguiente manera:

$$Spec(t, \omega) = |\tilde{f}_g(t, \omega)|^2 = |\tilde{f}^g(t, \omega)|^2 \tag{2.22}$$

Cuando se genera una imagen de un espectrograma, el eje horizontal representa el tiempo, el eje vertical es frecuencia, y la dimensión indica el valor del espectrograma de una frecuencia particular en un instante de tiempo específico, representado por la intensidad del color en la imagen. Para enfatizarla relación musical o tonal, el eje de la frecuencia es usualmente representado en una **escala logarítmica** (***log-frequency***

representation). Dicho eje toma en cuenta el hecho de que la percepción humana del tono es logarítmica. Aunque cabe mencionar que la escala específica no es de importancia, ya que solo es útil para propósitos de visualización de la imagen. En este tipo de representación, es más clara la importancia de la ventana de elección, como se ve en la figura 2.10.

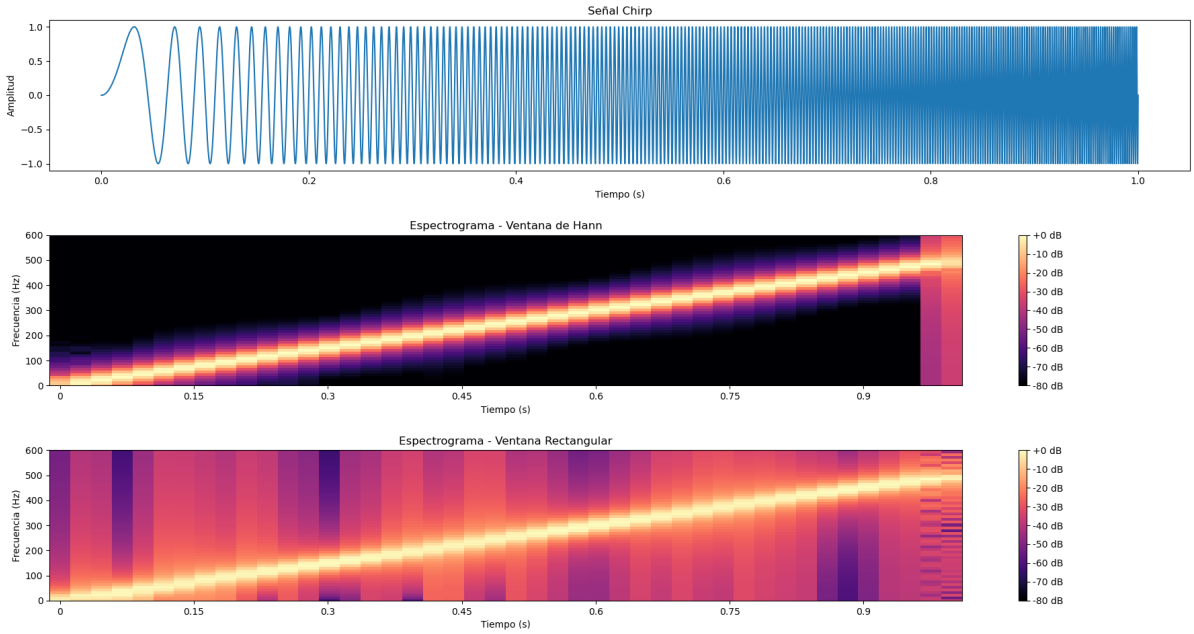


Figura 2.10: Señal chirp. Representación de espectrograma con ventana de Hann y con ventana rectangular. Ambas ventanas con un tamaño de 4096 muestras y un sobrelape de 2048 muestras.

Otro factor a tomar en cuenta es la longitud de la ventana usada en la STFT. Si se escoge una ventana grande resulta en una buena localización de frecuencias, pero una pobre localización en el tiempo; usar una ventana pequeña tiene el resultado opuesto. Incrementar el tamaño de la ventana deriva en una STFT que promedia las frecuencias de la señal sobre un intervalo mayor de tiempo, resultando en una pérdida de información en el tiempo. En el caso dónde se tiene una ventana infinita, se termina teniendo la transformada de Fourier usual, que promedia las frecuencias a lo largo de todo el dominio del tiempo $\mathbb{R}$. En el caso contrario, en donde sucesivamente se va disminuyendo el tamaño de la ventana resulta en una secuencia de Dirac, donde, en el caso en donde g es un impulso, la STFT regresa la señal original, con localización perfecta en el tiempo y ninguna información en frecuencia. La propiedad de localización en el tiempo de la STFT depende de la dispersión temporal de la función g, mientras que la propiedad de localización en frecuencia depende de la dispersión espectral de la transformada de Fourier de de $\hat{g}$.

2.3.12. Espectrogramas Mel (Mel Spectrograms)

Hasta ahora se han discutido los espectrogramas como representaciones bidimensionales del sonido, donde en el eje x se encuentra el tiempo y en el eje y la frecuencia. Sin embargo, en una primera instancia se tomó una escala lineal para representar la frecuencia, y esto es problemático debido a la naturaleza de percepción auditiva que tiene el ser humano. Para explicar mejor este fenómeno se tomará el siguiente ejemplo: sean dos intervalos de notas musicales tales como $C2 - C4$ (Do 2 con 65 Hz y Do 4 con 262 Hz) y $G6 - A6$ (Sol 6 con 1568 Hz y La 6 con 1760 Hz). Ambos intervalos en términos de frecuencia, tienen el rededor de 200 Hz entre la primer nota y la segunda. Al momento de reproducir estos tonos, la percepción que se tiene es que hay una mayor distancia en frecuencia en el primer intervalo que en el segundo (se invita al lector a que realice dicho experimento). Este fenómeno ocurre debido a que los humanos detectamos el sonido de forma logarítmica lo cual, dependiendo de la aplicación, podría suponer un problema con los espectrogramas convencionales de escala lineal.

Para solucionar estos inconvenientes, en 1937 Stevens, Volkmann y Newmann propusieron una unidad de tono tal que distancias iguales en tono sonaran igualmente distantes para el que lo escucha. A esto le llamaron **escala Mel** [19] y es una escala que es un tipo de escala perceptiblemente relevante en términos de tono. La forma en que se puede ir de Mels a Hz se puede representar matemáticamente de la siguiente forma [20]:

$$m = 2595 \cdot \log(1 + \frac{f}{700})$$

Y de igual manera, si se quiere ir de Mels a Hz, la expresión está dada por:

$$f = 700(10^{m/2595} - 1)$$

El mapeo de frecuencia entre Mels y Hertz puede observarse en la figura 2.11, la cual es claramente logarítmica y expresa la relación entre las dos unidades de medida.

De manera que, si se combina esta escala con la representación de espectrograma lo que se obtiene es una representación tiempo-frecuencia, cuya amplitud sea preceptivamente relevante en términos de amplitud y frecuencia. Para poder obtener un espectrograma en escala Mel se siguen los siguientes pasos:

- Extraer la STFT de la señal
- Convertir la amplitud a decibeles
- Convertir las frecuencias a escala Mel

Para poder convertir las frecuencias a escala Mel es necesario seguir los siguientes pasos:

- Escoger el número de filtros de las bandas Mel
- Construir los bancos de filtros Mel

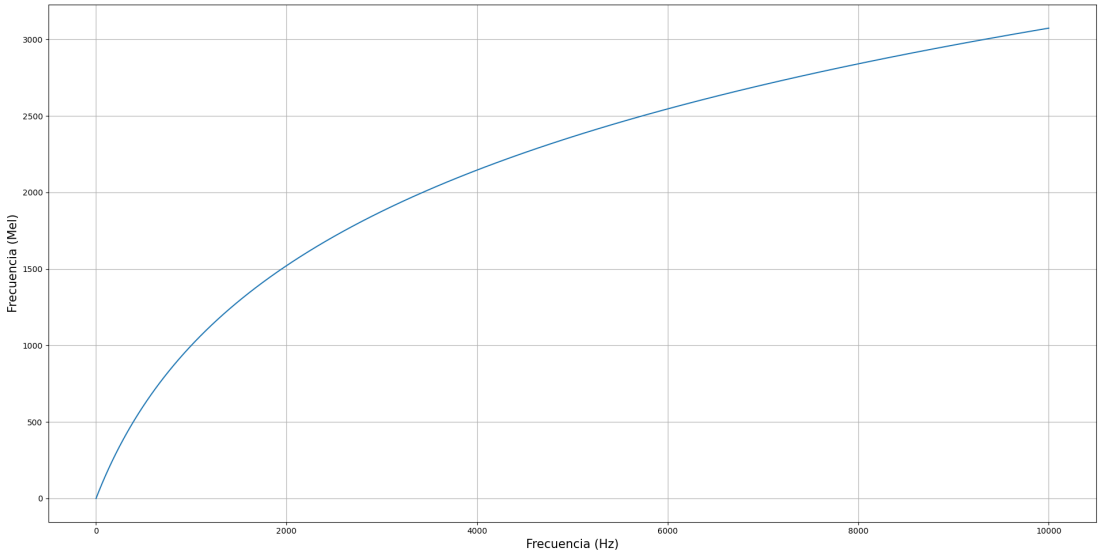


Figura 2.11: Curva de mapeo entre frecuencia en Hz y frecuencia en Mel.

- Aplicar los filtros Mel al espectrograma

El número de filtros de las bandas Mel es un hiper parámetro del modelo o algoritmo que se desee utilizar. Es decir, el número de bandas varía en función del problema que se busque atacar y puede ir desde unas pocas bandas (alrededor de 40 ó menos) y hasta las que sean necesarias. Para construir los bancos de filtros Mel se sigue la siguiente serie de pasos:

- Se convierten a Mel la frecuencia más alta y la más baja.
- Se crean el número de bandas igualmente espaciadas (bandas Mel)
- Se convierten esos puntos de vuelta a Hz
- Se redondea a la frecuencia más cercana
- Crear filtros triangulares

Dichos pasos quedan más claros con la figura 2.12, donde se puede observar que en el eje x se tiene la frecuencia, en la parte inferior está medida en Hz y en la parte superior en Mel. En el eje y está el peso, el cual actuará como filtro, en donde si se tiene valor de 1 la señal no se alterará. Si se cuenta el número de triángulos, se puede observar que se tomaron 6 bandas Mel, en donde los puntos que se muestran en la parte superior de la figura son las frecuencias centrales de cada banda.

Para la construcción de los filtros, el primer paso fue convertir a dominio Mel y tomar espacios igualmente separados. En la figura 2.12, se ve que al convertir de Mel a frecuencia, las bandas que eran igualmente separadas en el dominio Mel, en frecuencia tienen la distribución logarítmica, siendo más pequeños al principio y más grandes al final. Otra cosa a destacar en es que el inicio y el final de cada triangulo, coincide con

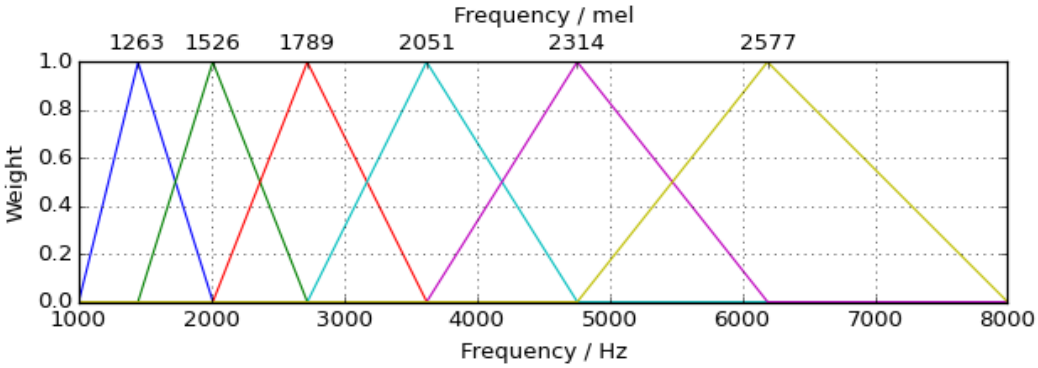


Figura 2.12: Banco de filtros Mel, imagen tomada de [21].

la frecuencia central del triangulo previo y el siguiente, respectivamente. Dicho esto, la forma del banco de filtros estará definida por $(\#\text{bandas}, \text{tamaño del frame}/2 + 1)$. Por lo tanto, la aplicación del banco de filtros al espectrograma, se puede denotar con la siguiente expresión:

$$Mel\ Spectrogram = MY$$

Donde M es el banco de filtros con la forma descrita anteriormente y Y es la representación del espectrograma con la forma $(\text{tamaño del frame}/2 + 1, \#\text{frames})$. Por lo que la forma del espectrograma Mel estará dada por $(\#\text{bandas}, \#\text{frames})$, es decir, se tendrá el número de filas que se escogió para el número de bandas y las columnas estarán dadas por el número de frames que originalmente se escogieron para el espectrograma. La visualización en el espectrograma no cambiará mucho, pero la información será más precisa en términos de percepción, y esto se puede ver en la figura 2.13.

2.4. Aprendizaje automatizado (Machine Learning)

De acuerdo con Burkov en [22], el aprendizaje automatizado o *machine learning* como se le referirá a lo largo de este documento, es un sub campo de las ciencias de la computación. Éste se ocupa de construir algoritmos que, para ser útiles, se basan en una colección de ejemplos sobre un fenómeno. Estos ejemplos pueden provenir de la naturaleza, creados manualmente por humanos o generados por otros algoritmos. El aprendizaje automatizado también puede ser definido como el proceso de resolver un problema práctico mediante: 1) reunir un conjunto de datos, y 2) de forma algorítmica construir un modelo estadístico basado en ese conjunto de datos.

Para ampliar la definición, y como se menciona en [23], el machine learning involucra el análisis e interpretación de estructuras y patrones presentes en lo datos, y de los cuales la máquina (modelo ó computadora) es capaz de aprender, tomar decisiones y razonar con esos patrones. Además, es capaz de obtener soluciones inteligentes sin la necesidad de una supervisión humana. Usando machine learning, un usuario

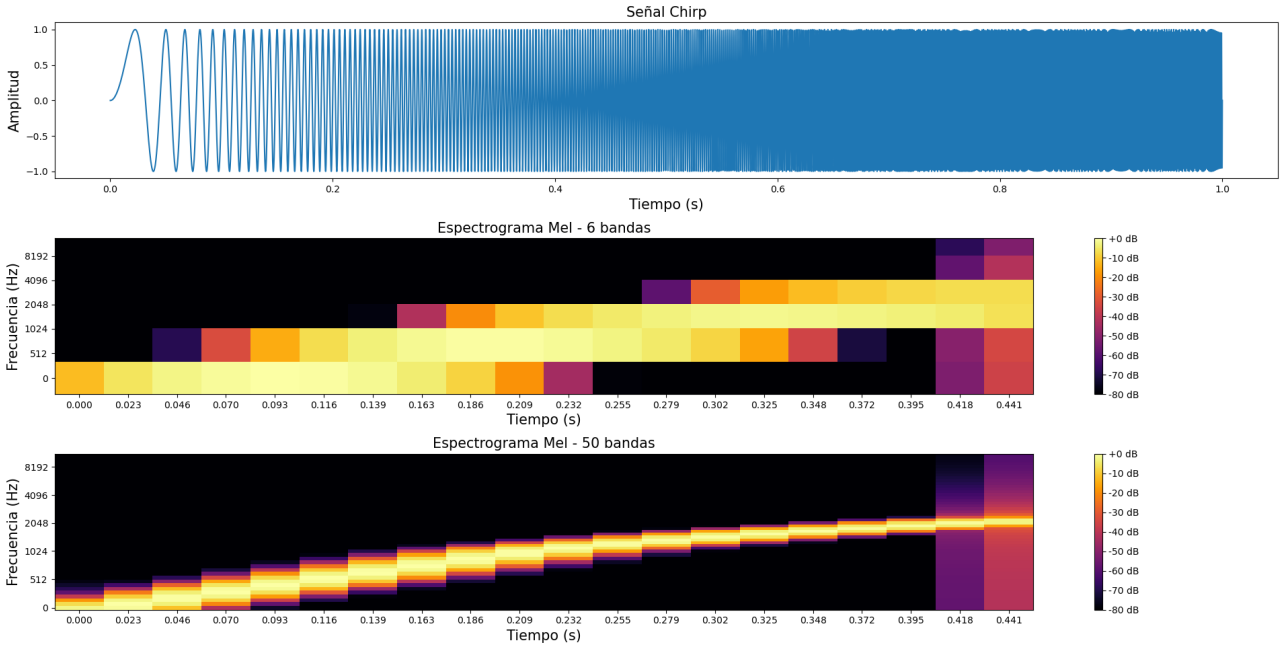


Figura 2.13: Comparación de espectrogramas Mel con múltiples bandas de una señal chirp

puede introducir una gran cantidad de datos, efectuar un entrenamiento, dar datos de prueba y obtener el resultado automático. Si se detecta algún error en los resultados, entonces el algoritmo de ML puede hacer cambios, corregir errores y usar esa información para tomar mejores decisiones en el futuro.

El machine learning consta de tres componentes:

- El algoritmo computacional, experto en efectuar una tarea de machine learning.
- Variables y características con base en las cuales se puede tomar decisiones.
- Un resultado real o esperado por el humano para determinar la precisión (*accuracy*) del sistema.

De modo que el aprendizaje que llevan a cabo estos algoritmos puede ser de cuatro formas: supervisado, semi supervisado, no supervisado y por refuerzo, los cuales se explicaran más a detalle a continuación.

2.4.1. Aprendizaje supervisado

En el aprendizaje supervisado [22], el conjunto de datos (*dataset*, como se le referirá a lo largo de este documento) es una colección de **ejemplos etiquetados** $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. Cada elemento $\mathbf{x}_i$ dentro de N es llamado vector de características. Dicho vector en cada dimensión $j = 1, ..., D$ contiene un valor que describe el ejemplo de alguna manera. Ese valor es llamado característica y se denota como $x^{(j)}$. por ejemplo, si cada ejemplo $\mathbf{x}$ la colección representa una persona, entonces la primer característica, $x^{(1)}$,

podría contener la altura en cm, la segunda característica, $x^{(2)}$, podría contener el peso en kg y así sucesivamente. Esto significa que se $x_i^{(2)}$ contiene el peso en kg de algún ejemplo $\mathbf{x}_i$, entonces $x_k^{(2)}$ también contendrá el peso en kg en cada ejemplo $\mathbf{x}_k, k = 1, ..., N$. Y la etiqueta y_i puede ser un elemento que pertenece a un conjunto finito de clases $\{1, 2, ..., C\}$, o un número real, o una estructura más compleja, como un vector, una matriz, un árbol o un grafo. Por ejemplo, si los ejemplos son correos electrónicos y el problema es detección de *spam*, entonces se tendrán dos clases $\{spam, noSpam\}$. El objetivo de un algoritmo de aprendizaje supervisado es utilizar el *dataset* para producir un modelo que tome el vector de características $\mathbf{x}$ como entrada, y como salida se obtenga información que permita deducir la etiqueta para ese vector de características.

2.4.2. Aprendizaje no supervisado

En el aprendizaje no supervisado [22], el *dataset* es una colección de ejemplos no etiquetados $\{\mathbf{x}_i\}_{i=1}^N$. De nuevo, $\mathbf{x}$ es un vector de características, y el objetivo de un algoritmo de aprendizaje no supervisado es crear un modelo que tome ese vector de características $\mathbf{x}$ como entrada y lo transforme en otro vector o un valor que puede ser utilizado para resolver problemas prácticos. Por ejemplo, en problemas de agrupamiento (*clustering*), el modelo regresa el *id* del grupo para cada vector de características en el *dataset*.

2.4.3. Aprendizaje semi supervisado

En aprendizaje semi supervisado [22], el *dataset* contiene ejemplos etiquetados y no etiquetados. Normalmente, la cantidad de ejemplos no etiquetados es mucho mayor que el número de ejemplos etiquetados. El objetivo de un algoritmo de aprendizaje semi supervisado es el mismo que el de un algoritmo de aprendizaje supervisado. La expectativa detrás de esto es que usando tantos ejemplos no etiquetados se pueda ayudar al algoritmo de aprendizaje a encontrar (producir) un mejor modelo. Este razonamiento podría sonar contra intuitivo, y pudiera parecer que se introduce más incertidumbre al problema. Sin embargo, cuando se adhieren ejemplos no etiquetados, también se adhiere más información acerca del problema: más cantidad de muestras se refleja en una mejor distribución de probabilidad correspondiente a los datos que se ingresaron.

2.4.4. Aprendizaje por refuerzo

El aprendizaje por refuerzo es un sub-campo del aprendizaje automatizado [22], donde la máquina "vive" en un ambiente y es capaz de percibir el estado de ese ambiente como un vector de características. La máquina puede ejecutar acciones en cada estado y diferentes acciones traen diferentes **recompensas**, y pueden incluso mover a la máquina a otro estado del ambiente. El objetivo de un algoritmo de aprendizaje por refuerzo es aprender una política. Una política es una función (similar a un modelo de aprendizaje supervisado), la cual toma el vector de características de

un estado como entrada y regresa como salida una acción óptima a ejecutar en ese estado. Donde la acción es óptima se maximiza la recompensa promedio esperada.

2.4.5. Parámetros vs. Hiperparámetros

Un hiperparámetro es una propiedad de un algoritmo de aprendizaje [22], normalmente (pero no siempre) teniendo un valor numérico. Este valor influencia la manera en que el algoritmo trabaja. Los hiperparámetros no son aprendidos por el propio modelo a partir de los datos, estos son definidos por el diseñador previo a correr el algoritmo.

Por otro lado los parámetros son variables que define el modelo entrenado. Los parámetros son directamente modificados por el algoritmo de aprendizaje basado en los datos de entrenamiento. El objetivo del entrenamiento es encontrar los valores de los parámetros tales que hagan al modelo óptimo en cierto sentido.

2.4.6. Anatomía de un algoritmo de aprendizaje

Para cada problema donde se aplique un algoritmo de aprendizaje, este puede constar de tres partes [22]:

1. Una función de perdida.
2. Un criterio de optimización basado en la función de perdida (función de costo, por ejemplo).
3. Una rutina de optimización que aproveche los datos de entrenamiento para encontrar un solución al criterio de optimización.

Estos son los bloques constructores de cualquier algoritmo de aprendizaje.

2.4.7. El modelo

De acuerdo con Jung en [24], si se considera alguna aplicación de ML que genere puntos de datos, cada uno caracterizado por características $\mathbf{x} \in X$ y etiquetas $y \in Y$. El principio informal de la mayoría de métodos de ML es aprender un mapa de hipótesis $h : X \to Y$ tal que:

$$y \approx h(\mathbf{x}) = \hat{y} \quad \text{para cada dato.} \tag{2.23}$$

Dicha hipótesis se puede usar para predecir la etiqueta de cualquier dato del que se tiene su vector de características. La predicción $\hat{y} = h(\mathbf{x})$ es obtenida evaluando la hipótesis para el vector de características $\mathbf{x}$ del dato. Parece natural referirse al mapa de hipótesis como un mapa predictor, dado que es utilizado para calcular predicciones para la etiqueta. Para un problema de ML usar un espacio finito de etiquetas Y (por ejemplo, $Y = \{-1, 1\}$) se puede considerar a la hipótesis como un clasificador. Para una Y finita, se puede caracterizar un mapa clasificador particular h usando diferentes regiones de decisión:

$$R^{(a)} = \{\mathbf{x} \in \mathbb{R}^n : h = a\} \subseteq X \tag{2.24}$$

Cada etiqueta valuada en $a \in Y$ es asociado con una región de decisión específica $R^{(a)} := \{\mathbf{x} \in \mathbb{R}^n : h = a\}$ es constituida por todos los vectores de características $\mathbf{x} \in X$ que son mapeados a este valor de etiqueta, $h(\mathbf{x}) = a$. En principio, los métodos de ML pueden usar cualquier mapa posible $h : X \to Y$ para predecir la etiqueta $y \in Y$ a través del cálculo de $\hat{y} = h(\mathbf{x})$, aunque en los métodos prácticos de ML solo pueden evaluar un subconjunto pequeño de todos los posibles mapas de hipótesis. Este subconjunto de mapas de hipótesis (*asequibles*) se denomina espacio de hipótesis o modelo subyacente a un método ML. De tal forma que, la elección del espacio de hipótesis involucra un compromiso entre complejidad computacional y propiedades estadísticas del método de ML resultante.

2.4.8. Función de pérdida

De acuerdo con Jung en [24], cada método de ML usa (de forma más o menos explicita), un espacio de hipótesis H, que consiste en todos los mapas de predicción computacionalmente asequibles h. Para determinar que mapa predictor h de todos los mapas en el espacio de hipótesis H es mejor para el problema que se busca resolver, se utiliza el concepto de **función de pérdida**. Formalmente, la función de pérdida es un mapa:

$$L : X \times Y \times H \to \mathbb{R}_+ : ((\mathbf{x}, y), h) \to L((\mathbf{x}, y), h) \tag{2.25}$$

La cual asigna a un par formado por un punto de datos, con características $\mathbf{x}$ y etiqueta y, y una hipótesis $h \in H$ el número real $L((\mathbf{x}, y), h)$.

El valor de pérdida $L((\mathbf{x}, y), h)$ cuantifica la discrepancia entre el valor real y el valor predicho $h(\mathbf{x})$. Un valor pequeño indica una baja discrepancia entre el valor predicho y el valor esperado. De tal forma que el principio básico de los métodos de ML puede ser descrito como: aprender (encontrar) una hipótesis a partir de un espacio de hipótesis H, que incurra en una mínima pérdida $L((\mathbf{x}, y), h)$ para cualquier punto de datos. Así como la hipótesis del espacio H usada para el método de ML es una elección, también lo es así el de la función de pérdida. La elección de dicha función debe tomar en cuenta la complejidad computacional de buscar en el campo de hipótesis la hipótesis con mínima pérdida. Por ejemplo, si se considera un método de ML que utilice un espacio hipótesis parametrizado por un vector de pesos y una función de pérdida convexa y diferenciable (suave) del vector de pesos. En este caso, buscar por una hipótesis con pérdida pequeña se puede realizar eficientemente usando métodos basados en gradientes (que se trataran más adelante). Además de los aspectos computacionales, la elección de la función de perdida también debe de tomar en cuenta los aspectos estadísticos. Algunas funciones de pérdida resultan en modelos que son más robustos frente a valores atípicos (*outliers*).

2.4.9. Función de pérdida para etiquetas numéricas

Para problemas de ML que involucran puntos de datos con etiquetas numéricas $y \in \mathbb{R}$, como lo son problemas de regresión, una opción ampliamente usada es **la función de pérdida de errores cuadrados**:

$$L((\mathbf{x}, y), h) := (y - h(\mathbf{x}))^2 \tag{2.26}$$

Donde $h(\mathbf{x})$ es la predicción $\hat{y}$ del modelo. La pérdida de errores cuadrados depende de las características $\mathbf{x}$ solo a través del valor de la etiqueta predicha $\hat{y} = h(\mathbf{x})$. Además de la predicción $h(\mathbf{x})$, se requiere de otras propiedades de las características $\mathbf{x}$ para determinar la pérdida.

2.4.10. Función de pérdida para etiquetas categóricas

Los problemas de clasificación involucran puntos de datos cuyas etiquetas toman valores de un espacio discreto de etiquetas Y. En el contexto de problemas de clasificación binaria, se asume que los valores de las etiquetas son $Y = \{-1, 1\}$. El objetivo de los métodos de clasificación es un clasificador que mapee las características $\mathbf{x}$ de los datos a una etiqueta predicha $\hat{y} \in Y$. Este tipo de modelos se implementan teniendo un umbral para el valor $h(\mathbf{x}) \in \mathbb{R}$ de una hipótesis, y que pueda entregar valores de números reales arbitrarios. Entonces se discriminan los puntos de datos como $\hat{y} = 1$ si $h(\mathbf{x}) > 0$ y $\hat{y} = -1$ en cualquier otro caso. Por lo tanto, la etiqueta predicha se obtiene por el signo del valor $h(\mathbf{x})$. Mientras que el signo de la predicción determina el resultado de la clasificación, el valor absoluto $|h(\mathbf{x})|$ se interpreta como la *confianza* en esta discriminación.

En principio, se puede medir la calidad de la hipótesis, cuando se usa para clasificar datos usando la pérdida de error cuadrático. Sin embargo, el error cuadrático es usualmente una medida pobre para la calidad de la hipótesis $h(\mathbf{x})$, que es usada para clasificar datos con una etiqueta binaria $y \in \{-1, 1\}$. Una de la funciones de pérdida es la directa formalización del requisito natural para que la hipótesis resulte en una correcta clasificación ($y = \hat{y}$) para cualquier dato. Esto sugiere aprender una hipótesis $h(\mathbf{x})$ minimizando la pérdida 0/1:

$$L((\mathbf{x}, y), h) := \begin{cases} 1 & \text{si } y \neq \hat{y} \\ 0 & \text{en otro caso} \end{cases} \text{ con } \hat{y} = 1 \text{ para } h(\mathbf{x}) \geq 0, \text{y } \hat{y} = -1 \text{ para } h(\mathbf{x}) < 0$$

La pérdida 0/1 es igual a cero si la hipótesis da como resultado una correcta clasificación $\hat{y} = y$. Para una clasificación incorrecta $\hat{y} \neq y$, la pérdida 0/1 entrega el valor de 1. Este tipo de pérdida es conceptualmente atractiva cuando los datos son interpretados como realizaciones de variables aleatorias independientes e idénticamente distribuidas (i.i.d.) con la misma distribución de probabilidad $p(\mathbf{x}, y)$. Dadas m realizaciones $(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$ de variables aleatorias *i.i.d.*:

$$(1/m) \sum_{i=1}^{m} L((\mathbf{x}^{(i)}, y^{(i)}), h) \approx p(y \neq \hat{y}) \tag{2.27}$$

con alta probabilidad para un tamaño de muestra m suficientemente grande. Una formulación precisa de la aproximación se puede obtener por la ley de los grandes números [25]. Se puede aplicar dicha ley dado que los valores $L((\mathbf{x}^{(i)}, y^{(i)}), h)$ son realizaciones de variables aleatorias *i.i.d.*. La pérdida media 0/1 en el lado izquierdo de la expresión anterior se denomina como precisión (*accuracy*) de la hipótesis h. Esta atractiva característica estadística de la pérdida 0/1 viene con el costo de una alta complejidad computacional. De hecho por cada dato $(\mathbf{x}, y)$, la pérdida 0/1 no es ni convexa ni diferenciable cuando es vista como una función de clasificación h. Por lo tanto, usando esta función para problemas de clasificación típicamente involucra métodos avanzados de optimización para resolver el problema de aprendizaje resultante.

Para evitar la no convexidad de la pérdida 0/1 se puede aproximar mediante una función de pérdida convexa. Y una aproximación popular es la pérdida de la bisagra (*hinge function*):

$$L((\mathbf{x}^{(i)}, y^{(i)}), h) := max\{0, 1 - y \cdot h(\mathbf{x})\} \tag{2.28}$$

Mientras que la función bisagra evita la no convexidad de la pérdida 0/1, sigue siendo una función no diferenciable del clasificador h. Las funciones de pérdida no diferenciables son típicamente más difíciles de minimizar, implicando una complejidad computacional de un método de ML que utiliza esta función de pérdida.

La pérdida logística es utilizada dentro de la regresión logística para medir la usabilidad de una hipótesis lineal $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ y se define como:

$$L((\mathbf{x}^{(i)}, y^{(i)}), h) := log(1 + \exp(-yh(\mathbf{x}))) \tag{2.29}$$

Si se considera un dato específico con el vector de características $\mathbf{x} \in \mathbb{R}^n$ y una etiqueta binaria $y \in \{-1, 1\}$. Si se usa una hipótesis lineal $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, con un vector de pesos $\mathbf{w} \in \mathbb{R}^n$, para predecir la etiqueta basado en las características $\mathbf{x}$ acordes con $\hat{y} = 1$ si $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} > 0$ y $\hat{y} = -1$ de otra forma. Entonces, ambas, la función bisagra y la función logística son funciones convexas del vector de pesos $\mathbf{w} \in \mathbb{R}^n$. La pérdida logística depende suavemente de $\mathbf{w}$. Es una función diferenciable en el sentido en que permite definir un gradiente con respecto a w. En contraste, la función bisagra no es suave, lo cual la hace más difícil de minimizar.

Los métodos de ML que usan la función logística, tales como la regresión logística, pueden aplicar métodos basados en gradiente, tales como descenso por gradiente (GD) para minimizar la pérdida promedio. En contraste, no se puede utilizar métodos basados en gradiente para minimizar la función bisagra, dado que no es diferenciable.

2.4.11. Otras funciones de pérdida populares

Error medio cuadrático (Mean Squared Error, MSE)

Esta es otra función de pérdida usualmente empleada en modelos de regresión, se calcula la diferencia entre la predicción del modelo y el valor real, se eleva al cuadrado y se promedia con todo el conjunto de datos [26]. Dado que se están elevando al cuadrado los errores, nunca se tendrá un resultado negativo y se define como:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.30}$$

donde N es el número de muestras con las que se está probando. Y la función se puede ver en la figura 2.14.

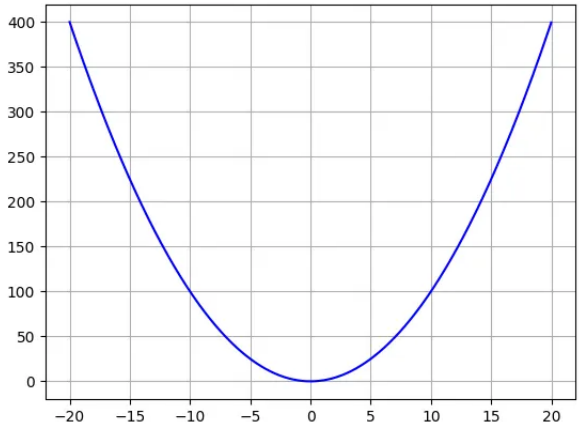


Figura 2.14: Función de error medio cuadrático. En el eje x se coloca el valor de y, y en el eje x el valor del MSE, imagen tomada de [26].

Error medio cuadrático (Mean Absolute Error, MAE)

Este tipo de pérdida se utiliza en problemas de regresión, y se calcula tomando la media de la diferencias absolutas entre el valor predicho y el valor esperado, como se observa a continuación:

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y - \hat{y}| \tag{2.31}$$

donde n es el número total de puntos (o muestras), y es el valor esperado y $\hat{y}$ es el valor predicho por el modelo.

2.4.12. Función de pérdida vs. función de costo

El término *pérdida* se refiere al error en la predicción de un algoritmo de aprendizaje. Una función de pérdida, por otro lado, es una función que calcula la pérdida

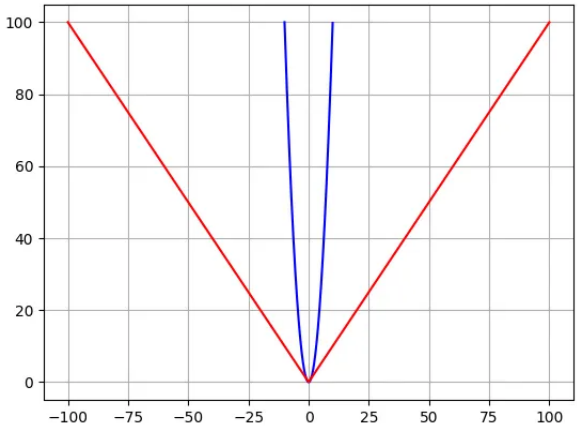


Figura 2.15: Función de error medio cuadrático (azul) comparada con función de error absoluto cuadrático (rojo). En el eje x se situa el valor de y y en el eje y los valores que toma la función de MSE y MAE respectivamente, imagen tomada de [26].

para una predicción en concreto. La función de pérdida es requerida por el algoritmo de aprendizaje para decidir que pasos tiene que tomar para minimizar la pérdida. Mientras que la función de pérdida calcula el error para un solo punto de dato (muestra), la función de costo calcula la pérdida para todo el conjunto de datos. El costo de un modelo no es mas que la suma de las pérdidas individuales sobre las muestras individuales de entrenamiento. Aunque es usual que el término pérdida y costo se utilicen indistintamente.

2.4.13. Descenso por gradiente (Gradient Descent, GD)

Recordando algunos conceptos de regresión lineal, la formula de la pendiente de una linea está descrita como $y = mx+b$, donde m es la pendiente y b es la intersección con el eje y. Y al observar el gráfico de dispersión se buscaba encontrar la recta que mejor se ajustara, lo cual requiere calcular el error entre el valor real y y el valor predicho $\hat{y}$. El descenso por gradiente funciona de manera similar, pero basado en una función convexa.

Similar a una regresión lineal, donde lo que se busca encontrar es la línea que mejor se ajuste, el objetivo del descenso por gradiente, o GD como se le referirá en este documento, es minimizar la función de costo, o el error entre el valor real y y el valor predicho $\hat{y}$ [27]. Para poder realizar esto, son necesarios dos puntos, una dirección una tasa de aprendizaje (*learning rate*), estos factores determinan los cálculos de la derivada parcial de futuras operaciones, permitiendo gradualmente llegar al mínimo local o global, nombrado punto de convergencia.

Los métodos basados en gradientes son métodos iterativos que construyen una secuencia de vectores de pesos $\mathbf{w}^{(1)}, ..., \mathbf{w}^{(r)}$ tales que los correspondientes valores objetivos $f(\mathbf{w}^1), ..., f(\mathbf{w}^r)$ convergen al valor mínimo $min_{\mathbf{w}\in\mathbb{R}^n} f(\mathbf{w})$. Las actualizaciones $\mathbf{w}^r \rightarrow \mathbf{w}^{r+1}$ entre vectores consecutivos de pesos están basados en los denominados pasos del GD (*GD steps*).

2.4.14. Pasos del descenso por gradiente

Sea un método de ML que usa un espacio de hipótesis parametrizado una función de pérdida suave, esto resulta en un problema de optimización suave tal que:

$$min_{\mathbf{w}\in\mathbb{R}^n} f(\mathbf{w}) \tag{2.32}$$

La función suave objetivo $f : \mathbb{R}^n \to \mathbb{R}$ es el riesgo empírico incurrido por una hipótesis con pesos $\mathbf{w} \in \mathbb{R}^n$ [24]. Donde la meta principal es encontrar un vector de pesos $\hat{w}$ que minimiza $f(\mathbf{w})$, $f(\hat{\mathbf{w}}) = min_{\mathbf{w}\in\mathbb{R}^n} f(\mathbf{w})$. Dado que que $f(\mathbf{w})$ es una curva suave, mediante cálculo elemental se puede aproximar localmente a un punto $\mathbf{w}^r$ usando un hiper plano tangente que pase a través del punto $(\mathbf{w}^r, f(\mathbf{w}^r))$. El vector normal de este hiper plano está dado por $\mathbf{n} = (\nabla f(\mathbf{w}^r), -1)$, como se observa en la figura 2.16. La primera componente del vector normal es el gradiente de la función $f(\mathbf{w})$ en el punto $\mathbf{w}^r$. Alternativamente se puede definir el gradiente $\nabla f(\mathbf{w}^{(r)})$ como [24]:

$$f(\mathbf{w}) \approx f(\mathbf{w}^{(r)}) + (\mathbf{w} - \mathbf{w}^r)^T \nabla f(\mathbf{w}^{(r)}) \text{ para una } \mathbf{w} \text{ suficientemente cerca de } \mathbf{w}^{(r)} \tag{2.33}$$

Y recordando que lo que se busca es encontrar un nuevo vector de pesos $\mathbf{w}^{(r+1)}$ que tenga un valor menor $f(\mathbf{w}^{(r+1)}) < f(\mathbf{w}^{(r)})$ que la suposición actual $\mathbf{w}^{(r)}$. La aproximación 2.33 sugiere escoger la siguiente suposición $\mathbf{w} = \mathbf{w}^{(r+1)}$ tal que $(\mathbf{w}^{(r+1)} - \mathbf{w}^{(r)})^T \nabla f(\mathbf{w}^{(r)})$ sea negativo. Lo cual se puede conseguir con el **paso del gradiente**:

$$\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} - \alpha \nabla f(\mathbf{w}^{(r)}) \tag{2.34}$$

con un tamaño de paso $\alpha > 0$ lo suficientemente pequeño. El tamaño del paso α debe de ser lo suficiente pequeño para asegurar la validez de la aproximación lineal. En el contexto del ML el parámetro del tamaño del paso del GD α también es conocido como **tasa de aprendizaje** (**learning rate**). De modo que, el tamaño del paso α determina la cantidad de progreso durante el descenso por gradiente hacia el aprendizaje del vector de pesos $\hat{\mathbf{w}}$ óptimo.

Sin embargo, la interpretación de la tasa de aprendizaje solo es útil para tamaños de paso lo suficientemente pequeños, ya que cuando se incrementa el tamaño del paso más allá de un valor crítico, las iteraciones se alejan del valor óptimo del vector de pesos $\hat{\mathbf{w}}$.

Los métodos basados en gradiente repiten el proceso de GD por un número suficiente de iteraciones para obtener una aproximación al valor óptimo del vector de pesos $\hat{\mathbf{w}}$. De modo que para una función objetivo $f(\mathbf{w})$ que sea convexa y diferenciable con un tamaño de paso lo suficientemente pequeño α, la iteraciones $f(\mathbf{w}^{(r)})$ obtenidas por la repetición de los pasos del descenso por gradiente convergen a un mínimo, es decir:

$$\lim_{r\to\infty} f(\mathbf{w}^{(r)}) = f(\hat{w}) \tag{2.35}$$

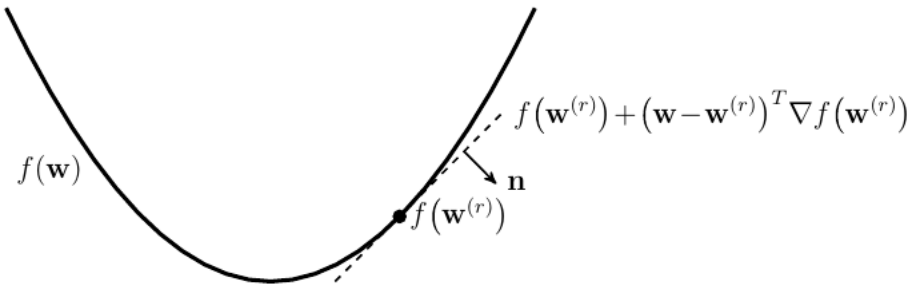
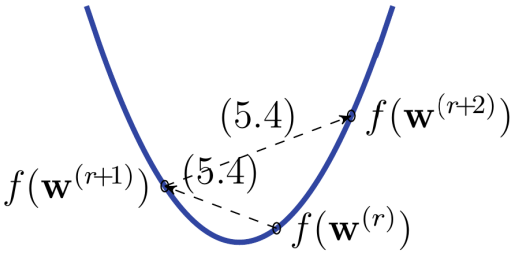


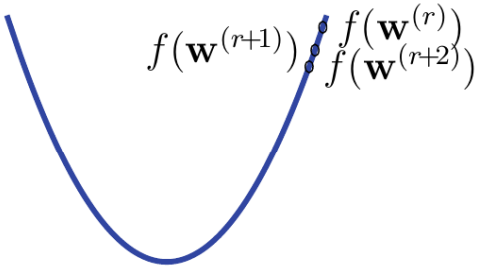
Figura 2.16: Una función suave $f(\mathbf{w})$, puede ser aproximada localmente alrededor del punto $\mathbf{w}(r)$ usando un hiper plano cuyo vector normal $\mathbf{n} = (\nabla f(\mathbf{w}^r), -1)$ esté determinado por el gradiente $\nabla f(\mathbf{w}^{(r)})$, imagen tomada de [24].

2.4.15. Escogiendo el tamaño del paso

La elección del tamaño del paso α en la técnica del descenso por gradiente tiene un impacto significativo en el desempeño del algoritmo [24]. Si se escoge un α demasiado grande, los pasos del GD divergen, como se puede observar en la figura 2.17a, y resulta en un algoritmo que falla en entregar aproximaciones satisfactorias para el vector de pesos $\mathbf{w}^{\text{opt}}$. Por otro lado, si se escoge un α muy pequeña, sucede lo mostrado en la figura 2.17b, donde las actualizaciones harán un progreso muy pequeño hacia la aproximación del vector de pesos $\hat{\mathbf{w}}$ óptimo.



(a) GD con un α muy grande, imagen tomada de [24].



(b) GD con un α muy pequeña, imagen tomada de [24].

Figura 2.17: Comparación de α en técnica de descenso por gradiente sobre una curva convexa y diferenciable.

La elección correcta de una buena α es una tarea complicada. Y pese a que varios sofisticados algoritmos para la sintonización del valor de α ha sido propuestos, el método empírico no es una mala aproximación si se conoce el estado del arte en relación a problemas similares. Sin embargo es útil conocer la condiciones suficientes del tamaño del paso que garanticen la convergencia de las iteraciones del GD.

Si se asume que la función objetivo $f(w)$ es convexa y suave. Entonces, los pasos del GD convergen a un mínimo para cualquier tamaño de paso α que satisfaga [28]:

$$\alpha \leq \frac{1}{\lambda_{\max}(\nabla^2 f(\mathbf{w}))} \text{ para todo } \mathbf{w} \in \mathbb{R}^n \tag{2.36}$$

En esta expresión se utiliza una matriz Hessiana $\nabla^2 f(\mathbf{w}) \in \mathbb{R}^{n \times n}$ de una función $f(\mathbf{w})$ cuyas componentes son las derivadas parciales de segundo orden $\frac{\partial f(\mathbf{w})}{\partial w_i \partial w_j}$ de la función $f(\mathbf{w})$. Es importante notar que garantiza la convergencia para cada posible inicialización $\mathbf{w}^{(0)}$ de las iteraciones del GD.

2.5. Aprendizaje profundo (Deep Learning)

Las redes neuronales artificiales son técnicas populares de machine learning que simulan los mecanismos de aprendizaje de organismos biológicos [29]. El sistema nervioso humano contiene células llamadas neuronas. Estas neuronas están conectadas una con la otra mediante el uso de *dendritas* y *axones*, y las regiones conectadas mediante estos elementos se les conoce como *sinapsis*.

Los mecanismos biológicos son simulados en las redes neuronales artificiales, las cuales contienen unidades de computo a las que se les denomina neuronas. Estas unidades computacionales están conectadas unas con las otras mediante *pesos*, los cuales desempeñan el mismo rol que la fuerza de conexión sináptica en los organismos biológicos. Cada entrada de una neurona es escalada (ponderada) con un peso, la cual afecta la función calculada en esa unidad.

Una red neuronal artificial calcula una función de las entradas propagando los valores de las entradas en las neuronas hacia las salidas de las neuronas, y usa los pesos como parámetros intermedios. Y el aprendizaje ocurre cambiando el peso de de las conexiones entre neuronas. Los datos de entrenamiento proveen una retroalimentación para corregir los pesos en la red neuronal dependiendo de que tan bien haya predicho la salida. El objetivo de cambiar los pesos es modificar la función calculada, para hacer mejores predicciones en iteraciones futuras. La habilidad de calcular acertadamente funciones de entradas no vistas con anterioridad a partir de un entrenamiento sobre un conjunto de datos finito de pares *entrada-salida*, se le conoce como **generalización del modelo**.

El objetivo principal de una red neuronal es *aprender* una función que relacione una o más entradas a una o más salidas mediante el uso de ejemplos de entrenamiento. En otras palabras, teniendo solo ejemplos de entradas y salidas, el modelo *construirá* una función que relacione las entradas con las salidas.

2.5.1. El perceptrón

La red neuronal más simple se conoce como el perceptrón [29]. Esta red neuronal contiene una sola capa de entrada y un nodo de salida. Dado que el conjunto de entrenamiento tiene d entradas y una sola salida, se asume que la red neuronal tiene d entradas, denotadas como el vector $\overline{X}$ (el cual es el argumento de la función a ser

aprendida) y una sola salida y. La capa de entrada contiene d nodos que transmiten las d características $x_1, x_2, ..., x_d$ contenidas en el vector $\overline{X}$. Se utiliza vectores fila de características (en lugar de los vectores columna usuales) debido a que los vectores son usualmente extraídos de matrices filas de matrices. La capa de entrada son las aristas incidentes de pesos $w_1, w_2, ..., w_d$ contenidos en el vector columna $\overline{W} = [w_1, w_2, ..., w_d]^T$, y todas sus aristas son incidentes en un solo nodo de salida. La capa de salida no ejecuta ninguna operación por si misma. La función lineal $\overline{W} \cdot \overline{X}^T = \sum_{i=1}^{d} w_i x_i$ es ejecutada en el nodo de salida. Subsecuentemente, el signo de este valor real es usado para predecir la variable dependiente de $\overline{X}$. Por lo tanto, la predicción $\hat{y}$ es calculada como:

$$\hat{y} = f(\overline{X}) = sign\{\overline{W} \cdot \overline{X}^T\} = sign\{\sum_{j=1}^{d} w_j x_j\} \tag{2.37}$$

La función signo mapea el valor real a $+1$ ó -1, lo cual es apropiado para clasificación binaria. Las aristas de la entrada hacia la salida contienen pesos w_d, son multiplicados con cada característica y sumados en el nodo de salida, como se observa en la figura 2.18. Subsecuentemente, la función signo es aplicada para convertir el valor agregado a la etiqueta de una clase. La función signo sirve el rol de *función de activación*. El valor de la variable en un nodo de una red neuronal, es en ocasiones llamado *activación*.

En muchas ocasiones hay una parte invariante de la predicción, a la cual se le refiere como *sesgo* (**bias**). El sesgo puede ser introducido como un peso de una arista utilizando una neurona de peso. Esto se logra sumando una neurona que siempre transmita el valor de uno al nodo de salida. El peso de esta arista que conecta la neurona de sesgo con el nodo de salida provee la variable sesgo, en la figura 2.18 se puede observar los diagramas del perceptrón simple, con y sin neurona de sesgo (*bias nueron*).

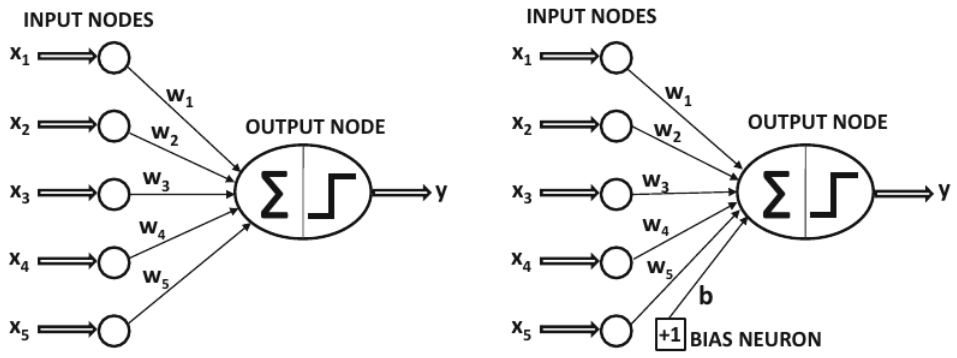


Figura 2.18: A la izquierda diagrama de un percetrón simple, se tienen los nodos de entrada (*input nodes*), e insidien todos en el nodo de salida (*output node*), donde se suman y el resultado pasa a través de la función signo para obtener el valor de la predicción. De lado izquierdo es también un perceptrón simple, pero agrega la neurona de sesgo (*bias neuron*), imagen tomada de [29].

2.5.2. Función de activación

La elección de la función de activación es una parte crucial del diseño de las redes neuronales. Diferentes tipos de funciones no lineales tales como *sign*, *sigmoide*, *tangente hiperbólica*, etc, son comúnmente usadas. La notación Φ se utiliza para denotar la función de activación. Una red de una sola capa con un vector columna $\overline{W}$ de pesos y un vector (fila) $\overline{X}$ tendrían una predicción de la forma:

$$\hat{y} = \Phi(\overline{W} \cdot \overline{X}^T) \tag{2.38}$$

La función de activación más básica $\Phi(\cdot)$ es la activación lineal, que también es referida como activación identidad:

$$\hat{y} = \Phi(v) = v \tag{2.39}$$

La función de activación lineal es usualmente utilizada cuando en el nodo de salida, el objetivo es un valor real. Las funciones clásicas de activación son la función signo, sigmoide,tangente hiperbólica y ReLU:

$\Phi(v) = sign(v)$ función signo ,figura 2.19a.
$\Phi(v) = \dfrac{1}{1+e^{-v}}$ función sigmoide, figura 2.19b.
$\Phi(v) = \dfrac{e^{2v}-1}{e^{2v}+1}$ función tanh, figura 2.19c.
$\Phi(v) = \text{máx}\{v, 0\}$ función ReLU (Rectified Linear Unit), figura 2.19d.

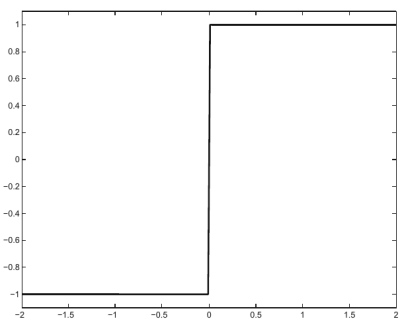
2.5.3. Función de activación Softmax

La función de activación softmax es usualmente usada en la capa de salida para mapear los k valores reales en k probabilidades de sucesos discretos. Esta función de activación mapea valores reales a probabilidades que sumen uno. Específicamente, la función para la $i - esima$ salida está definida como:

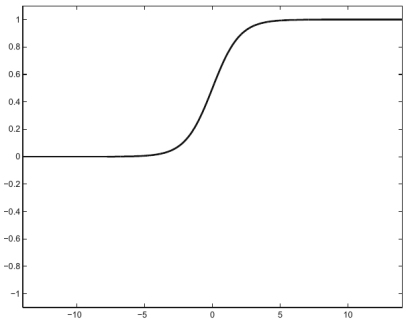
$$\Phi(\overline{v})_i = \frac{\exp(v_i)}{\sum_{j=1}^{k} \exp(v_j)} \forall i \in \{1, ..., k\} \tag{2.40}$$

2.5.4. Redes neuronales multicapa

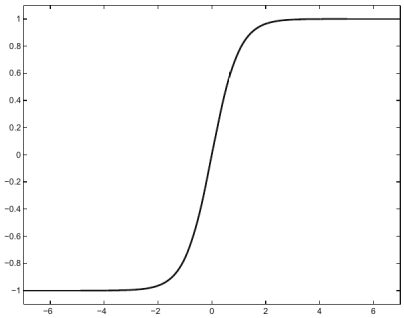
Las redes neuronales multicapa contiene más de una capa computacional. El perceptrón contiene una capa de entrada y una de salida, de las cuales solo la capa de salida realiza cálculos. La capa de entrada transmite los datos a la capa de salida, y todos lo cálculos son visibles al usuario. Las redes multicapa contienen múltiples capas computacionales [29], las capas intermedias adicionales (entre la capa de entrada y la de salida) se les conoce como *capas ocultas*, mostradas en la figura 2.20, debido a que sus cálculos no son visibles al usuario.



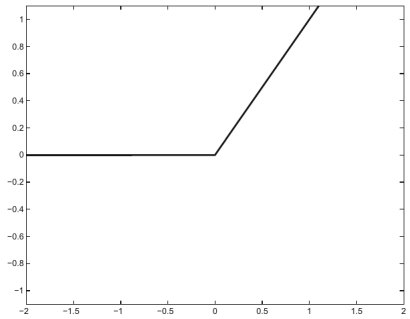
(a) Función sign



(b) Función sigmoide



(c) Función tanh



(d) Función ReLU

Figura 2.19: Funciones de activación

La arquitectura específica de una red multicapa se les conoce como redes *feed-forward*, esto debido a que las capas sucesivas se alimentan unas a otras en dirección de la entrada hacia la salida. L arquitectura estándar de las redes *feed-forward* asume que todos los nodos en una capa están conectados con todos lo nodos de la capa siguiente. Así como en el perceptrón, la función de pérdida penaliza desviaciones indeseables de salidas predichas por la red neuronal. Si una red neuronal contiene $p_i, ..., p_k$ unidades en cada una de sus k capaz, entonces el vector (columna) de estas salidas, denotadas como $\overline{h_1}...\overline{h_k}$ tienen dimensionalidades $p_i, ..., p_k$. Por lo tanto, el número de unidades en cada capa se le conoce como dimensionalidad de la capa.

2.5.5. Algoritmo de retropropagación (Backpropagation)

Para este punto, conociendo las bases teóricas, ya es posible realizar el entrenamiento de una red. Para lograr esta tarea, un método comúnmente utilizado es el conocido como *backpropagation*, desarrollado por David E. Rumelhart, Geoffrey E. Hinton y Ronald J. Williams en 1986 [30]. De tal forma que, la idea detrás del algoritmo de retropropagación se basa en que no se conoce lo que las capas ocultas están haciendo, pero se puede calcular que tan rápido cambia el error cuando cambia la actividad en las capas ocultas [31]. De ahí que se pueda determinar que tan rápido cambia el error cuando se cambia el peso de una conexión individual. Esencialmente, se está buscando el camino con el descenso con mayor pendiente, donde el

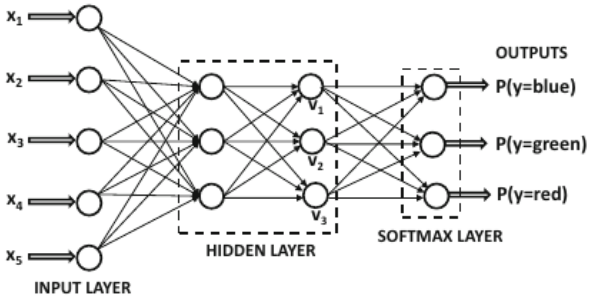


Figura 2.20: Ejemplo de una red neuronal con múltiples salidas en las capas ocultas (*hidden layers*, se usa una capa softmax *softmax layer* parta clasificación multi etiqueta), imagen tomada de [29].

inconveniente es que se trabaja en un espacio con una alta dimensionalidad. Se inicia calculando las derivadas del error con respecto a un solo ejemplo de entrenamiento.

Cada capa oculta puede afectar a varias capas ocultas. Por lo tanto es necesario combinar muchos efectos separados en el error de forma informativa. Se tomará una aproximación de programación dinámica. Una vez se tienen las derivadas del error para una capa oculta, se utiliza para calcular la derivada del error para las actividades de la capa anterior. Y una vez se encuentran las derivadas del error para las actividades de las unidades ocultas, es sencillo obtener las derivadas del error para los pesos que conducen a una unidad oculta. Para una mejor explicación se usará la figura 2.21.

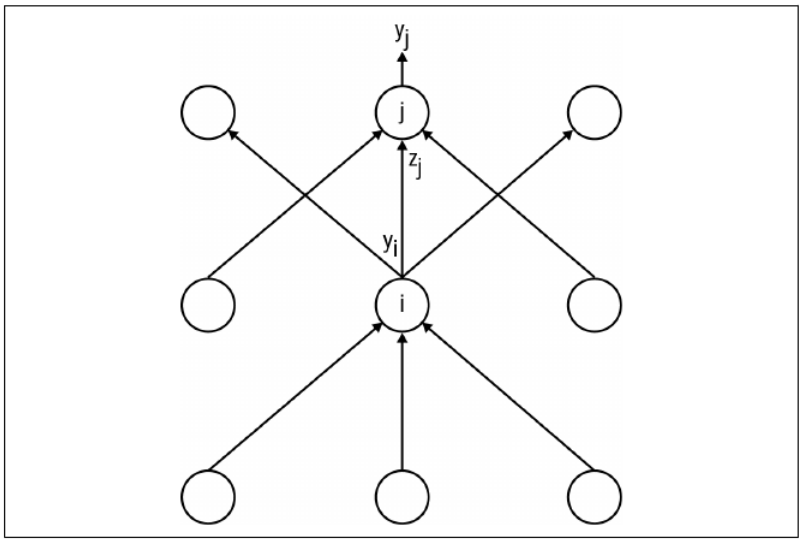


Figura 2.21: Diagrama de referencia para la obtención del algoritmo de retropropagación, imagen tomada de [31].

El subíndice utilizado se refiere a la capa de la neurona. El símbolo y se referirá a la actividad de la neurona.De forma similar, el símbolo z se referirá al logit de la neurona. El logit representa el logaritmo de la razón de probabilidad de un evento que ocurre respecto a un evento que no ocurre. Matemáticamente, se define como:

$$logit(p) = ln\left(\frac{p}{1-p}\right) \tag{2.41}$$

Donde:

- $logit(p)$ es el logit de la probabilidad p.
- ln representa el logaritmo natural.
- p es la probabilidad de que ocurra un evento.

Específicamente se calcula la derivada del error en la capa de salida:

$$E = \frac{1}{2}\sum_{j\in salida}(t_j - y_j)^2 \rightarrow \frac{\partial E}{\partial y_j} = -(t_j - y_j) \tag{2.42}$$

Y ahora se aborda el paso inductivo. Presumiblemente, se tiene las derivadas del error para las capas j. A continuación, se calculan las derivadas del error para las capas anteriores, capa i. Para realizar esto, se debe de acumular información acerca de como la salida de la neurona en la capa i afecta los logits de cada neurona en la capa j. Esto se puede realizar aprovechando el hecho de que la derivada parcial del logit con respecto a los datos de salida de la capa anterior es meramente el peso de la conexión w_{ij}:

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial z_j}\frac{dz_j}{dy_i} = \sum_j w_{ij}\frac{\partial E}{\partial z_j} \tag{2.43}$$

Además, se puede observar que:

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{dy_j}{dz_j} = y_j(1-y_j)\frac{\partial E}{\partial y_j} \tag{2.44}$$

Combinando estas dos expresiones, se puede obtener la expresión para las derivadas del error en la capa i en términos de el las derivadas del error en la capa j:

$$\frac{\partial E}{\partial y_i} = \sum_j w_{ij}y_j(1-y_j)\frac{\partial E}{\partial y_j} \tag{2.45}$$

Una vez se ha llenado apropiadamente la tabla con todas las derivadas parciales (de las funciones de error con respecto a las capas ocultas). se puede determinar como el error cambia con respecto a los pesos. Esto brinda una manera de modificar los pesos después de cada ejemplo de entrenamiento.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial E}{\partial z_j} = y_i y_j (1 - y_j) \frac{\partial E}{\partial y_j} \tag{2.46}$$

Finalmente, para completar el algoritmo, y como antes, simplemente se suman las derivadas parciales sobre todos los ejemplos de entrenamiento en el conjunto de datos. Esto da la siguiente modificación a la fórmula:

$$\Delta w_{ij} = - \sum_{k \in dataset} \in y_i^{(k)} y_j^{(k)} (1 - y_j^{(k)}) \frac{\partial E^{(k)}}{\partial y_i^{(k)}} \tag{2.47}$$

2.5.6. Redes neuronales convolucionales (Convolutional Neural Networks, CNN)

Una red Neuronal Convolucional, también conocidas como CNN's (por sus siglas en inglés) ó ConvNet, es una clase de redes neuronales que se especializan en el procesar datos que tienen *topología de red* [32]. Una imagen digital es una representación binaria de información visual. Contiene una serie de píxeles en arreglos de *red*, que contienen valores de píxeles que denotan que tan brillante y de que color debe ser cada píxel. Las capas de este tipo de redes están configuradas de tal manera que detectan patrones simples (líneas, curvas, etc.) y patrones más complejos (caras, objetos, etc.) más adelante. Usualmente la arquitectura de una CNN simple tiene 3 capas: la capa convolucional, la capa de agrupación *pooling* y una capa completamente conectada (*fully connected layer*).

Capa convolucional

La capa convolucional es el bloque central de una CNN. Esta capa realiza una convolución entre dos matrices, donde una matriz es el conjunto de parámetros aprendibles, también conocido como *kernel*, y la otra matriz es la región restringida del campo receptivo. El kernel es espacialmente menor que la imagen, pero es más profundo. Esto significa, que si la imagen está formada por tres canales (RGB) el kernel será espacialmente más pequeño, pero se extenderá por los tres canales de la imagen. Durante la propagación hacia delante, el kernel se desliza alrededor de el ancho y largo de la imagen, como se ilustrado en la figura 2.22, produciendo una representación del campo receptivo. Esto produce una representación bidimensional de la imagen conocida como mapa de activación (*activation map*) que da la respuesta del kernel a cada posición espacial de la imagen. Y el tamaño del deslizamiento del kernel es llamado *stride*. Si se tiene un tamaño de entrada $H \times W \times C$ y $Cout$ número de filtros con un tamaño espacial de F con unstride de S y un $Padding$ P, entonces el tamaño de la salida puede ser determinado como:

$$W_{salida} = \frac{W - F + 2P}{S} + 1 \tag{2.48}$$

La convolución aprovecha tres ideas importantes que motivan la visión por computadora: interacción dispersa (*sparse interaction*), compartición de parámetros (*para-*

meter sharing) y representación equivariante (*equivariant representation*). Las redes neuronales usan multiplicación de matrices con una matriz de parámetros para describir la interacción entre la entrada y la salida. Esto significa que cada unidad de entrada interactúa con cada unidad de salida. Sin embargo, las redes neuronales tienen *sparse interactions*. Esto se consigue haciendo kernels más pequeños que la entrada. Esto significa que se almacenarán menos parámetros, lo que no solo reduce los requerimientos de memoria del modelo, si no que también mejora la eficiencia estadística del modelo.

En redes neuronales tradicionales, cada peso en la matriz de pesos, es usado solo una vez, a diferencia de en una CNN y gracias a la propiedad de *shared parameters*, para obtener la salida, los pesos aplicados para la entrada son los mismos que los aplicados en cualquier otro lado. Y por último la propiedad de *equivariant representation*, si se cambia la entrada de alguna forma, la salida también cambiará en esa manera.

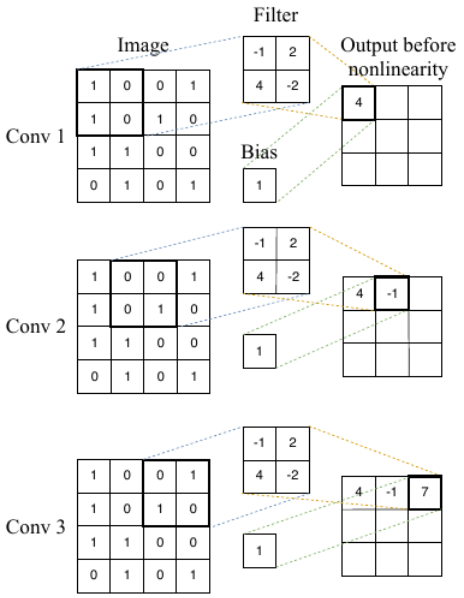


Figura 2.22: Kernel (Filtro) convolucionando a través de la imagen, imagen tomada de [22].

Capa de agrupación (Pooling)

La capa de Pooling reemplaza la salida de la red en ciertas ubicaciones obteniendo una estadística promedio de las salidas cercanas. Esto ayuda a reducir el tamaño espacial de la representación, la cual reduce la cantidad de cálculos y pesos en el modelo. La operación pooling es procesada en cada porción de la representación individualmente. Existen muchas funciones de pooling, tales como el promedio del vecindario rectangular, norma $L2$ del vecindario rectangular o un promedio ponderado basado en la distancia del centro de cada píxel; ver figura 2.23.

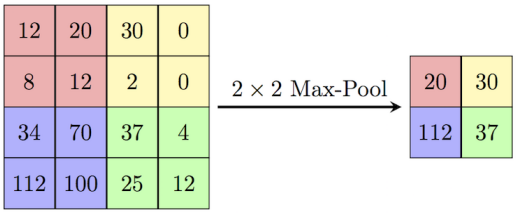


Figura 2.23: Operación de Max Pooling donde se obtiene el máximo del vecindario rectangular, imagen tomada de [33].

S se tiene un mapa de activación de tamaño $H \times W \times C$, un kernel de pooling espacial de tamaño F, un *stride* S, entonces el tamaño de la salida puede determinarse como:

$$W_{salida} = \frac{W - F}{S} + 1 \tag{2.49}$$

Esto lleva a una salida de tamaño $W_{salida} \times W_{salida} \times C$. En todos los casos, el pooling provoca cierta invariancia de traslación, lo que significa que un objeto puede ser reconocible independientemente de donde aparezca en el cuadro.

Capa completamente conectada (Fully Connected)

Las neuronas en esta capa tienen una conectividad completa con todas las neuronas en las capas previas y las capas siguientes, como en una red neuronal convencional. Este tipo de capas ayudan a mapear la representación entre la entrada y la salida.

2.6. Modelos generativos

Un modelo generativo puede ser definido, a grandes rasgos, como un modelo que describe como un conjunto de datos es generado, en términos de un modelo probabilístico [34]. Tomando muestras de este modelo es posible generar nuevos datos.

Para esto primero se requiere un conjunto de datos existente que contenga muchos ejemplos de la entidad que se está tratando de generar. Esto es conocido como datos de entrenamiento, y uno de estos datos se denomina *observación*. Cada *observación* se compone de muchas características, para generación de imágenes, las características suelen ser píxeles individuales. Aquí el objetivo es construir un modelo que pueda generar nuevos conjuntos de características que luzcan como si hubieran sido creados usando las mismas reglas que el conjunto original.

Un modelo generativo debe de ser probabilístico en vez de determinístico, como se muestra en la figura 2.24. Si el modelo es meramente un cálculo fijo, tal como tomar el valor promedio de cada píxel en el conjunto de datos, no sería un modelo generativo porque el modelo produce la misma salida siempre, es por eso que el modelo debe incluir un elemento estocástico (aleatorio) que influencie las muestras individuales generadas por el modelo.

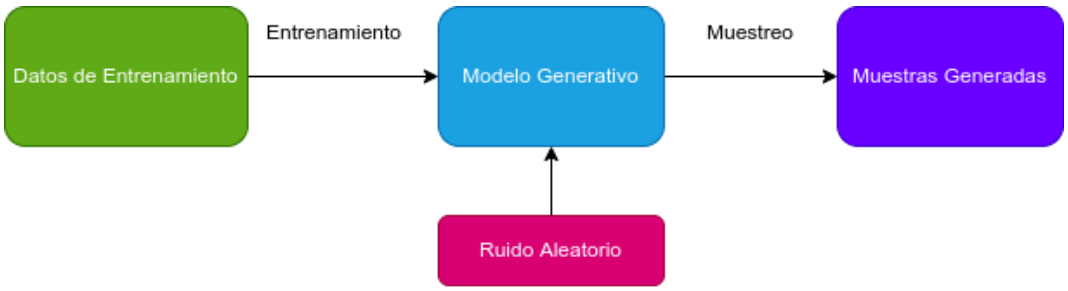


Figura 2.24: Estructura de un modelo generativo.

2.6.1. Autocodificadores (Autoencoders)

El objetivo de este tipo de modelos es crear una arquitectura que tenga un *cuello de botella*, que garantice una representación de menor dimensionalidad de la información original [35]. Estas arquitecturas están basadas en dos componentes separadas una llamada **codificador** y la otra llamada **decodificador**. La tarea del codificador es comprimir los datos en una representación de menor dimensionalidad, también llamado *espacio latente*. En otras palabras, se toma la información como una entrada, y se aprenden formas mediante las cuales se puede comprimir esa información en una representación con menor dimensionalidad. El *espacio latente* es una representación de los datos originales que solo se enfoca en los atributos o características más importantes. Sin embargo, para que el codificador sea eficiente, se tiene que cumplir la condición de que la información tiene que tener dependencias a través de las dimensiones, es decir, si todas las dimensiones de los datos de entrada son independientes, es prácticamente imposible aprender una representación de menor dimensionalidad, ya que no hay manera de capturar las características más importantes sin perder mucha información. A diferencia de métodos de reducción de dimensionalidad como **PCA** que solo puede aprender relaciones lineales, el autoencoder puede aprender relaciones no lineales de los datos de entrada, como es ilustrado en la figura 2.25, y esta capacidad del codificador se puede generalizar a tantas dimensiones como se desee. Esto significa que los autoencoders pueden capturar relaciones de estructuras más complejas. Por otro lado, el decodificador es la contra parte del codificador, ya que descomprime la representación de baja dimensionalidad de vuelta al dominio de la información original.

La forma en que se entrenan los autoencoders es mediante el algoritmo de retro propagación (backpropagation), donde el error que se busca minimizar es el error de reconstrucción (*reconstruction error*), el cual se define como la diferencia entre la señal original y la señal que se obtiene del autoencoder, de modo que la función de pérdida más usada es el error medio cuadrático (MSE). Se busca que un autoencoder sea lo suficientemente sensible para que pueda reconstruir los datos de entrada, pero lo suficientemente insensible para que no se sobre ajuste (memorice) los datos de entrada. Para poder lograr esta meta, es necesaria una función de pérdida más compleja, la cual introduce un término de *regularización*.

Una implementación popular del autoencoder es el llamado Autoencoder Convolucional Profundo (*Deep Convolutional Autoencoder*), el cual es similar a la estructura

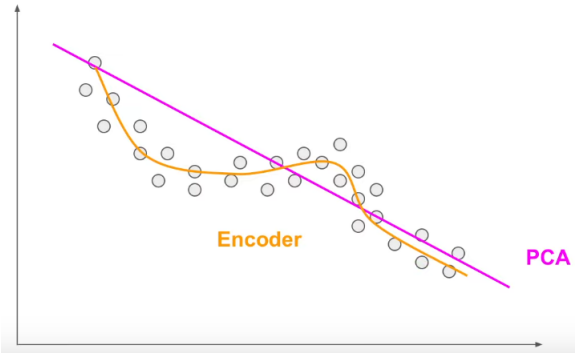


Figura 2.25: Visualización de la capacidad de aprender relaciones no lineales del Codificador (Encoder), a diferencia del PCA que solo aprende relaciones lineales, imagen tomada de [35].

con capas completamente conectadas, pero este cuenta con capas convolucionales en lugar de las neuronas convencionales. Para el codificador (Encoder), cada capa se compondrá de la convolución, una función de activación *Leaky ReLU* y una normalización de lote (*Batch Normalization*). Y para el decodificador (Decoder) cada capa se compondrá de una convolución traspuesta, la función de activación *Leaky ReLU* y una normalización de lote (*Batch Normalization*). Cabe mencionar que, esta no es siempre la configuración que se encuentra en todos los modelos, pero es bastante similar. Hasta el momento podría parecer inútil este tipo de arquitectura, ya que se inicia con una señal, se comprime y después se vuelve a tener la señal reconstruida, lo cual podría llevar al pensamiento de dejar de lado esta arquitectura y usar directamente la señal original. Sin embargo, la razón para hacer todo esto es el *el espacio latente*, este espacio guarda las características más importantes de los datos de entrada, y una vez teniendo esta versión comprimida de la versión original resulta ser útil para realizar tareas de generación, reducción de ruido (*Denoising*), detección de anomalías, entre otras.

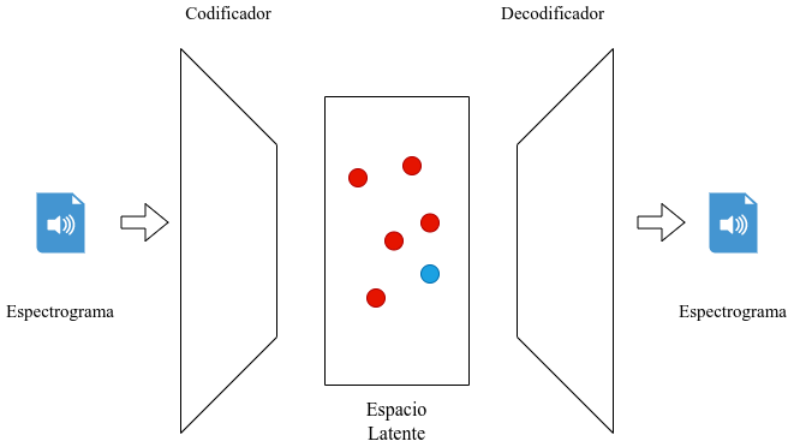


Figura 2.26: Proceso de generación con autoencoder.

Para la tarea de generación, la idea es alimentar información al autoencoder (imágenes, audio, etc.), para este ejemplo tomaremos espectrogramas, de forma que el codificador codificará los espectrogramas originales como un punto dentro del espacio latente, y cuando ese punto se pase al decodificador, este será capaz de reconstruir el espectrograma, que idealmente, será bastante similar al original introducido en el codificador. Los puntos en el espacio latente serán tantos como espectrogramas se introduzcan al codificador, para la etapa de generación solo es necesaria el espacio latente y el decodificador, de forma que la intuición detrás de esto es muestrear un punto dentro del espacio latente que sea diferente de todos los puntos entes aprendidos, y este nuevo punto será utilizado para generar un nuevo espectrograma que sea distinto a cualquier espectrograma involucrado en el proceso de entrenamiento, este proceso puede verse en la figura 2.26.

Sin embargo, este tipo de configuración de autoencoder tiene ciertas limitaciones, por ejemplo, si se grafica el espacio latente generado después del entrenamiento, dicha gráfica no es simétrica alrededor del origen, y el problema con esto es que entonces no se cuenta con ningún punto de referencia que se pueda tomar como guía para nuevas generaciones de datos. Otro inconveniente es que algunas etiquetas son representadas sobre áreas más pequeñas del espacio latente, lo que deriva en falta de diversidad cuando se muestrea un punto aleatoriamente sobre el espacio latente. Un inconveniente más es que existen espacios entre los puntos de cada etiqueta, esto provoca que algunas de las imágenes generadas sean de muy mala calidad, todos estos fenómenos pueden verse en la figura 2.27.

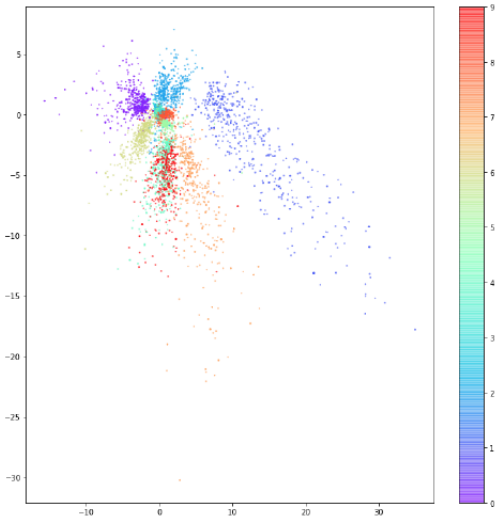


Figura 2.27: Espacio latente generado después de entrenar al autoencoder con el conjunto de datos MNIST. Cada color de punto corresponde a un dígito diferente que varían entreo 0 y 9, imagen tomada de [36]

2.6.2. Autoencoders variacionales (Variational Autoencoder, VAE)

En la sección anterior se discutieron los inconvenientes que se tienen con los autoencoders, como lo son: que el espacio latente no es simétrico al rededor del origen, algunas etiquetas son representadas en áreas más pequeñas que otras y que existen discontinuidades en el espacio latente. Para solucionar estos problemas, se desarrollaron los Autoencoders Variacionales, los cuales tiene dos elementos principales distintos en su composición: el componente del encoder y la función de pérdida. En el autoencoder simple (AE) cuando aprende una característica, se mapea un solo punto en el espacio latente, a diferencia del Autoencoder Variacional (VAE), el cual mapea una distribución normal multivariada en el espacio latente, mostrado en la figura 2.28.

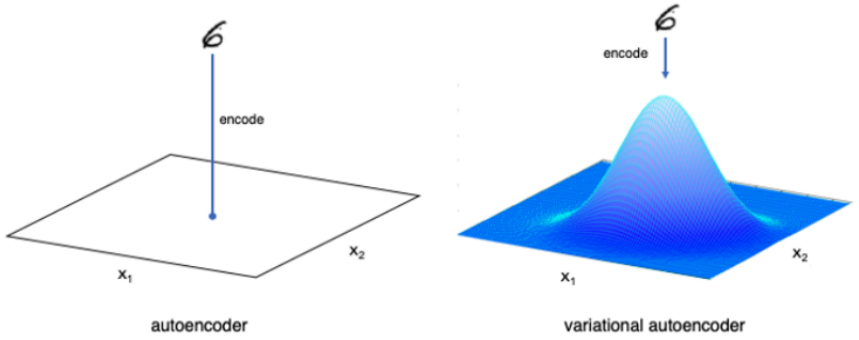


Figura 2.28: Comparación de AE (autoencoder) contra VAE (variational autoencoder). En el AE se mapea un solo punto en el espacio latente, y en el VAE se mapea una distribución normal multivariada, imagen tomada de [34]

Una distribución normal es un sinónimo de distribución Gaussiana, y si se tiene una distribución normal en $1D$, también se le conoce como distribución normal univariada. Una distribución normal es básicamente una distribución de probabilidad con forma de campana y se utilizan para describir variables aleatorias con valores reales (por ejemplo, población, altura, peso, etc.). Para definir una distribución de probabilidad se necesitan dos variables: μ que corresponde con la media y el centro de la distribución y σ que corresponde con la desviación estándar y la variabilidad de la distribución, y se expresa de la siguiente manera:

$$f(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{2.50}$$

De forma que si se quiere muestrear un punto z de la distribución normal, se tiene que:

$$z = \mu + \sigma\epsilon \tag{2.51}$$

Siendo ϵ una muestra de las distribución normal estándar.

Una vez sabido esto, una distribución normal multivarida es una distribución normal que va a estar en función de dos o más distribuciones normales, como se ve en

la figura 2.29 donde se muestra una distribución normal bi-variada, ya que depende de dos distribuciones normales que se proyectan sobre un plano.

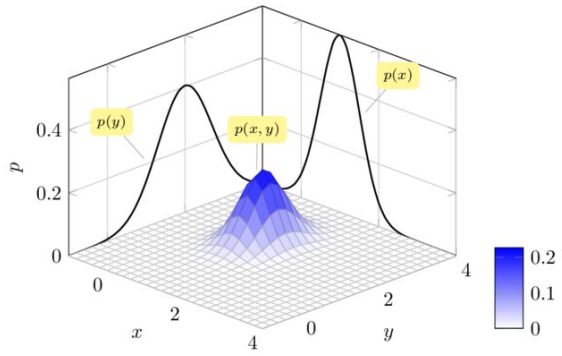


Figura 2.29: Distribución normal bi-variada, imagen tomada de [37]

Y matemáticamente, la distribución de probabilidad multivariada se puede expresar como:

$$f(x_1, ..., x_k) = \frac{e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^T \sum^{-1}(\vec{x}-\vec{\mu})}}{\sqrt{(2\pi)^k |\sum|}} \tag{2.52}$$

Donde $\vec{\mu}$ y $\vec{\sigma}$ tendrán tantos valores como dimensiones se tengan, y Σ será la matriz de covarianza que indica que tanta variabilidad tienen las diferentes distribuciones a lo largo de los ejes, además introduce una variable ρ la cual representa la correlación entre x y y.

Dicha σ para una distribución bi-variada se ve de la siguiente manera:

$$\sum = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix} \tag{2.53}$$

Una vez llegado este punto y habiendo entendido una distribución multivariada, dentro del VAE se asume que todas las dimensiones son independientes ($\rho = 0$), de tal forma que se tiene una matriz Σ diagonal, que contiene la varianza de cada dimension en su diagonal principal. Por lo tanto, el codificador mapea cada entrada en un vector promedio y un vector varianza dentro del espacio latente. Y reescribiendo la expresión vista antes, si se quiere muestrear un punto en el espacio latente, se hace la siguiente forma:

$$z = \vec{\mu} + \Sigma\epsilon \tag{2.54}$$

Donde ϵ es un punto de la distribución normal estándar y a Σ se le agrega un elemento logarítmico para que pueda tomar valores positivos y negativos, por lo que está dada por:

$$\Sigma = e^{\frac{log(\vec{\sigma^2})}{2}} \tag{2.55}$$

De este modo al mapear una distribución normal multivariada en el espacio latente, se eliminan las discontinuidades dentro del mismo. Al muestrear un punto dentro

del espacio latente de un AE, podía caer en un punto nulo, lo que generaba señales de mala calidad, y con un VAE al no tener puntos nulos, se garantiza una mejor inferencia de valores a la salida del modelo, ya que los puntos muestreados cerca de la misma área, producirán resultados similares, reduciendo el error de reconstrucción.

Como se menciono al principio de esta sección, los dos cambios principales de un VAE sobre un AE simple son la forma en que el codificador mapea la entrada en el espacio latente (lo cual ya se describió) y la función de pérdida, la cual se aborda a continuación.

En la sección anterior se observó que la función de pérdida que se utilizaba es MSE, así que para ir de un AE a una VAE se utiliza la función de pérdida RMSE (*Root Mean Squared Error*) agregando un término extra llamado *divergencia de Kullback-Liebler* la cual representa la diferencia entre una distribución normal (vector promedio, vector varianza logarítmico) y una distribución normal estándar, y se ve da la siguiente manera:

$$LOSS = RMSE + KL \tag{2.56}$$

La divergencia de KL no es precisamente una distancia, desde el punto de vista matemático, ya que no es simétrica, es decir, $KL(A, B) \neq KL(B, A)$. Para distribuciones normales puede se puede expresar su forma cerrada como:

$$D_{KL}(N(\mu, \sigma)||N(0, 1)) = \frac{1}{2}\sum(1 + log(\sigma^2) - \mu^2 - \sigma^2) \tag{2.57}$$

Donde $N(\mu, \sigma)$ es la distribución normal y $N(0, 1)$ representa la distribución normal estándar. Se realiza una suma a través de todas las dimensiones del espacio latente. De forma que, en la expresión de la pérdida para el VAE, lo que la divergencia de KL hace es penalizar las observaciones donde los vectores de media y varianza logarítmica difieren significativamente de los parámetros de la distribución normal (media cero y varianza uno). En otras palabras, lo que esta divergencia está tratando de hacer es empujar a la distribución normal a ser una distribución normal estándar. Y la razón para necesitar esto es por que promueve la simetría del espacio latente y disminuye las probabilidades de grandes espacios entre los grupos de puntos, los cuales eran los problemas vistos en el AE de la sección anterior.

El último detalle de la función de pérdida para el VAE es que se agrega un parámetro α que multiplica a la función $RMSE$, y dicha α representa el peso de la pérdida de reconstrucción. Encontrar el valor de α es complicado ya que si se tiene un valor muy grande resultará en imágenes reconstruidas de mala calidad, pero si se cuenta con un α muy grande se tienen los mismos problemas que con el AE, porque el valor de KL será tan pequeño que no figurará en la función de pérdida, por lo tanto α puede ser tratado como un hiperparámetro a ser optimizado durante el periodo de entrenamiento.

2.6.3. Redes generativas adversarias (Generative Adversarial Networks, GAN's)

Una GAN es un modelo generativo que consta de dos partes fundamentales, un generador y un discriminador. El papel del generador es tratar de convertir ruido aleatorio en observaciones que luzcan como si hubieran sido tomadas del conjunto de datos original, y la tarea del discriminador es aprender a distinguir datos reales y los generados por el generador [34]. Para la explicación de este modelo se supondrá que se busca generar imágenes. Usualmente tanto el generador como el discriminador suelen ser redes neuronales distintas. Al principio del proceso, el generador entrega imágenes ruidosas y el discriminador predice aleatoriamente. La clave de las GAN's recae en como se alterna el entrenamiento de ambas redes, de modo que a medida que el generador se vuelve mejor engañando al discriminador, el discriminador debe adaptarse para mantener su habilidad de identificar correctamente cuales observaciones son falsas. Esto lleva al generador a encontrar nuevas formas de engañar a el discriminador y así el ciclo continúa.

El discriminador

El objetivo de el discriminador es predecir si las imágenes son reales o falsas. Esto es un problema de clasificación de imágenes supervisado, así que es posible usar una arquitectura de capas convolucionales apiladas seguidas por una capa densa a la salida. En el artículo original de las GAN, capas densas fueron utilizadas en lugar de las capas convolucionales. Sin embargo, se ha demostrado que las capas convolucionales otorgan un mayor poder predictivo al discriminador. También es común ver capas de normalización de lotes (*batch normalization layers*). Una función sigmoide se utiliza al final de las capas para asegurar que la salida esté escalada entre cero y uno. Esta será la probabilidad predicha de que la imagen sea real.

EL generador

La entrada al generador es un vector, usualmente extraído de una distribución estándar multivariada. La salida es una imagen del mismo tamaño que la original en el conjunto de entrenamiento. De hecho, el generador de una GAN cumple exactamente el mismo propósito que el decodificador en un VAE: convertir un vector en un espacio latente en una imagen. Es común que se introduzca una capa de sobremuestreo (*upsampling layer*), la cual duplica el ancho y el alto del vector de entrada. Esta simplemente repite cada renglón y columna de la entrada para doblar el tamaño. Después es seguida de una capa normal de convolución.

Entrenamiento de la GAN

Se puede entrenar al discriminador creando un conjunto de entrenamiento donde algunas de las imágenes son observaciones reales aleatoriamente seleccionadas del conjunto de entrenamiento y otras son salidas del generador. La respuesta será uno para las imágenes verdaderas y cero para las creadas por el generador . Si se trata a

esto como un problema de aprendizaje supervisado, se puede entrenar al discriminador para aprender a diferenciar entre imágenes reales y las generadas, los valores de salida cercanos a uno serán tratados como imágenes reales y los cercanos a cero como imágenes generadas.

Entrenar al generador es un poco más complicado, ya que no hay un conjunto de entrenamiento que diga la verdadera imagen a la que un punto particular en el espacio latente debería ser mapeado. En su lugar, solo se busca que engañe al discriminador, es decir que cuando se alimente al discriminador, la salida de este sea cercana a uno. Para entrenar al generador, se alimenta al discriminador con la salida del generador de tal manera que la salida de estos modelos combinados sea la probabilidad de que la imagen generada sea real, acorde con el discriminador. Se puede entrenar este modelo combinado creando lotes de entrenamiento, que consisten de vectores latentes aleatoriamente generados de 100 dimensiones como entrada y una respuesta que es fijada en uno, dado que se quiere entrenar al discriminador para producir imágenes que el discriminador crea que son reales, como se puede ver en la figura 2.30.

La función de pérdida es entonces la función de entropía cruzada binaria (*binary cross-entropy*) entre la salida del discriminador y el vector respuesta de uno.

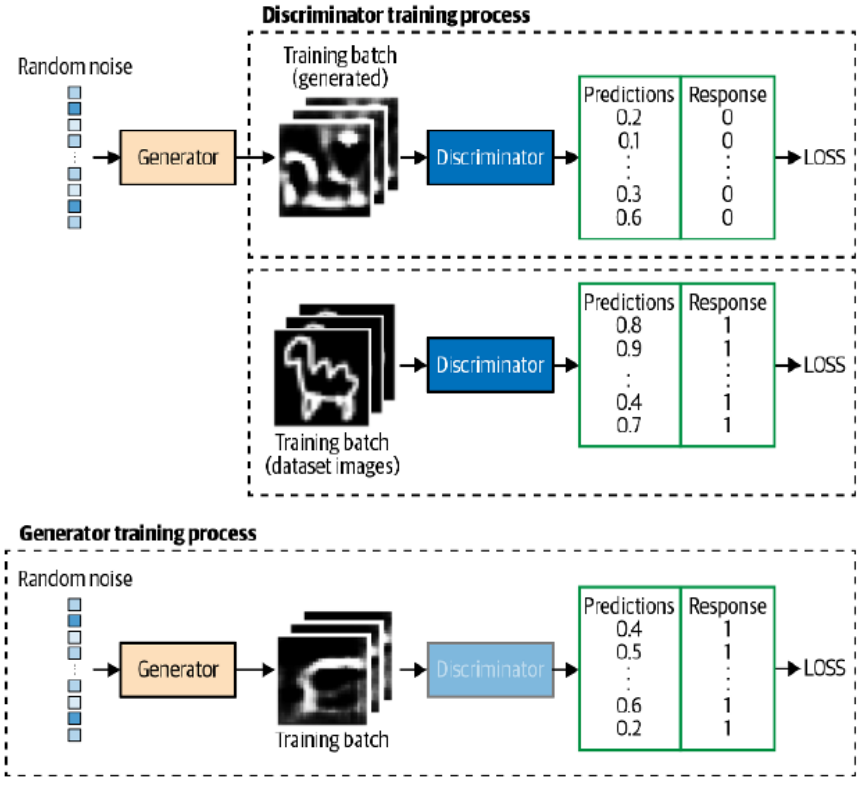


Figura 2.30: Proceso de entrenamiento de una GAN. Arriba el proceso de entrenamiento del discriminador, donde se entrena con dos lotes (/) distintos, uno generado por el generador y otro con imágenes reales. Abajo el proceso de entrenamiento del generador, donde se obtiene el valor de la pérdida (*LOSS*) de un lote imágenes generadas por el generador. Tomada de [34].

Después de una adecuada cantidad de épocas, el discriminador y el generador habrán encontrado un equilibrio que permita que el generador aprenda información importante del discriminador y la calidad de las imágenes empezará a mejorar.

Retos de una GAN

Pese a que las GAN fueron un logro dentro del contexto de los modelos generativos, tienen grandes inconvenientes al momento de entrenar, tales como:

Oscilación de la pérdida

La oscilación de la pérdida, la pérdida del discriminador y del generador puede empezar a oscilar descontroladamente, en lugar de mostrar una estabilidad a largo plazo. Usualmente, suele haber pequeñas oscilaciones de la pérdida entre cada lote de entrenamiento, pero en el largo plazo debería de observarse que la pérdida se estabiliza o gradualmente incremente o disminuya, en lugar de fluctuar de forma errática, esto para asegurar la convergencia de la GAN y su mejora en el tiempo.

Modo colapso

El modo colapso se da cuando el generador encuentra un pequeño número de muestras que engañan al discriminador y por lo tanto no es capaz de producir otras muestras que no sean ese conjunto limitado. Esto significa que el gradiente de la función de costo colapsa cerca de cero. Incluso si se tratará de reentrenar el discriminador para que no sea engañado, el generador simplemente encontrará otra forma de engañar al discriminador, dado que se ha vuelto *iluso* a la entrada y por lo tanto no tiene incentivo para diversificar su salida.

Pérdida desinformativa

Dado que los modelos de aprendizaje profundo son compilados para minimizar la pérdida, sería razonable pensar que mientras más pequeña sea la pérdida del generador, mejores serán las imágenes generadas por este. Sin embargo, dado que el generador es evaluado contra el actual discriminador y el discriminador está constantemente mejorando, no se puede comparar la función de pérdida evaluada en diferentes puntos del proceso de entrenamiento. Esto provoca que en ocasiones el entrenamiento de las GAN sea difícil de supervisar.

Hiperparámetros

Incluso con una GAN simple, el número de parámetros es muy grande. Y este tipo de arquitecturas son muy sensibles a cambios pequeños en todos estos parámetros, por lo que encontrar en conjunto de parámetros que funcionen es usualmente una tarea de intento y error.

2.6.4. Modelos de difusión (Diffusion Models)

Los Modelos de Difusión están inspirados por la termodinámica del no equilibrio. Estos modelos definen una cadena de Markov de pasos de difusión, para lentamente agregar ruido aleatorio a los datos y entonces aprender a revertir el proceso de difusión para construir la muestras de datos deseados a partir del ruido [38]. A diferencia de los VAE o modelos de flujo, los modelos de difusión son aprendidos con un procedimiento fijo y la variable latente tiene alta dimensionalidad (la misma que la entrada). La configuración consiste básicamente en dos procesos [39]:

- Un proceso de difusión fijo hacia delante (*forward*) q de elección del usuario, que gradualmente agregue ruido Gaussiano a una imagen, como se ilustra en la figura 2.31, hasta que se tenga solo ruido.
- Un proceso aprendido de difusión hacia atrás p_θ, donde una red neuronal es entrenada para gradualmente remover el ruido de la imagen empezando con solo ruido, hasta que se termine con una imagen.

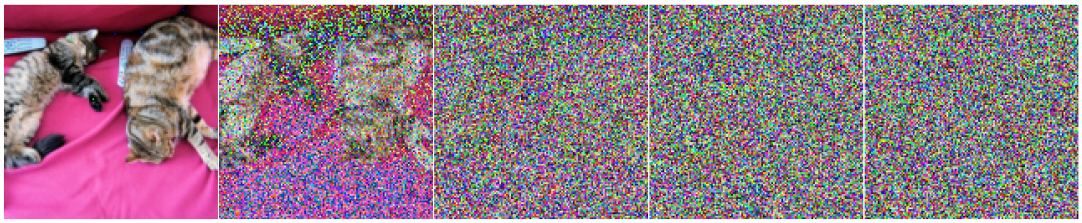


Figura 2.31: Proceso de agregado de ruido para proceso de difusión, imagen tomada de [39]

Ambos procesos indexados por t ocurren por un número finito de pasos de tiempo T. Se empieza con $t = 0$ donde se muestrea una imagen real x_0 que pertenece a la distribución de datos (por ejemplo, la imagen de un gato), entonces en el paso hacia delante muestrea un poco de ruido (definido por el programador de uido) de una distribución Gaussiana en cada paso de tiempo t, el cual se agrega a la imagen del paso de tiempo anterior. Dada una T suficientemente larga y un buen *programa* (*schedule*) para agregar ruido en cada paso de tiempo, se termina con la llamada distribución Gaussiana Isotrópica en $t = T$ a través de un proceso gradual.

De una forma más matemática, sea $q(\mathbf{x}_0)$ una distribución de datos reales de imágenes. Se puede muestrear de ella una imagen, $\mathbf{x}_0 \sim q(\mathbf{x}_0)$. El proceso hacia delante se puede definir como $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ la cual agrega ruido Gaussiano en cada paso de tiempo t, de acuerdo con un programa (*schedule*) de varianza conocido $0 < \beta_1 < \beta_2 < ... < \beta_T < 1$ como:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t I) \tag{2.58}$$

Es preciso recordar que una distribución normal está definida por dos parámetros, la media μ y la varianza $\sigma^2 \geq 0$. Básicamente, cada nueva (ligeramente más ruidosa)

imagen en cada paso de tiempo t, es extraída de una distribución Gaussiana condicional con $\mu_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1}$ y $\sigma^2 = \beta_t$, lo cual se puede hacer muestreando $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ y a continuación estableciendo $\mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\epsilon$.

Notese que β_t no es constante en cada paso de tiempo t (de ahí el sufijo), de hecho la elección del programa está a criterio del diseñador. Muchos modelos generativos basados en en difusión han sido propuestos ideas similares, incluyendo los llamados *Denoising Diffusion Probabilistic Models* (DDPM) [40], del cual se hablará más adelante.

Así que empezando desde $\mathbf{x}_0$, se termina con $\mathbf{x}_1, ..., \mathbf{x}_t, ..., \mathbf{x}_T$, donde $\mathbf{x}_T$ es ruido Gaussiano puro si se fija apropiadamente el programa. Ahora, si se conoce la distribución condicional $p(\mathbf{x}_{t-1}|\mathbf{x}_t)$, entonces se puede efectuar el proceso inverso, esto se logra muestreando algo de ruido Gaussiano $\mathbf{x}_T$ (definido por el programador de ruido), y gradualmente quitar el ruido para que se termine con una muestra de la distribución real $\mathbf{x}_0$. Sin embargo, no se conoce $p(\mathbf{x}_{t-1}|\mathbf{x}_t)$. Es inextricable dado que requiere saber la distribución de todas las imágenes posibles para calcular esta probabilidad condicional. De ahí que, se aprovechará una red neuronal para aproximar (aprender) esta distribución condicional de probabilidad, la cual se denomina $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, con θ siendo los parámetros de la red neuronal, actualizados mediante descenso por gradiente. Dado que se necesita una red neuronal para representar una distribución de probabilidad (condicional) del proceso hacia atrás (*backward process*). Si se asume que este proceso inverso es también Gaussiano, entonces estará en función de una media parametrizada μ_θ y una varianza parametrizada Σ_θ, de tal forma que el proceso parametrizado estará dado por:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)) \tag{2.59}$$

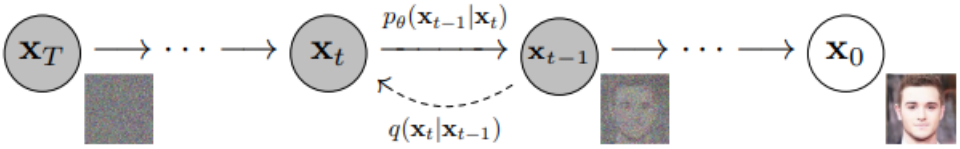


Figura 2.32: Proceso de agregado y quitado de ruido para proceso de difusión, imagen tomada de [39]

Donde la media y la varianza también son condicionadas al nivel de ruido t. De ahí que, la red neuronal necesite aprender (representar) la media y varianza. Sin embargo, los autores del DDPM decidieron dejar fija la varianza, y dejar a la red neuronal que aprendiera la media μ_θ de la distribución condicional de probabilidad. Del artículo original:

"*Primero, definimos* $\Sigma_\theta(\boldsymbol{x}_t, t) = \sigma_t^2 \boldsymbol{I}$ *para desentrenar las constantes dependientes del tiempo, ambas* $\sigma_t^2 = \beta_t$ *y* $\sigma_t^2 = \tilde{\beta}_t$*, tienen resultados similares*"

Definiendo una función objetivo

Para obtener una función objetivo para aprender la media del proceso hacia atrás, el autor observó que la combinación de q y p_θ pueden ser vistas como un VAE. Por lo tanto, el límite inferior variacional (*variational lower bound*), también llamado ELBO, pueda ser usado para minimizar la verosimilitud logarítmica con respecto a la muestra de datos real $\mathbf{x}_0$. Resulta que el ELBO para este proceso es una suma de pérdidas en cada paso de tiempo t, $L = L_0 + L_1 + ... + L_T$. Por construcción del proceso hacia delante q y el proceso hacia atrás, cada término (excepto L_0) de la pérdida es de hecho la divergencia KL entre dos distribuciones Gaussianas, que pueden ser escritas explícitamente como una pérdida $L2$ con respecto de las medias.

Una consecuencia directa del proceso hacia delante construido q, es que se puede muestrear $\mathbf{x}_t$ en cualquier nivel de ruido arbitrario condicionado a $\mathbf{x}_0$ (dado que la suma de Gaussianas también es una Gaussiana), por lo que se tiene que:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\overline{\alpha}_t}\mathbf{x}_0, (1-\overline{\alpha}_t)\mathbf{I}) \tag{2.60}$$

Con $\alpha_t := 1 - \beta_t$ y $\overline{\alpha} := \prod_{s=1}^{t} \alpha_s$. Esto significa que se puede muestrear ruido Gaussiano y escalarlo apropiadamente y sumarlo a $\mathbf{x}_0$ para obtener $\mathbf{x}_t$ directamente. Nótese que $\overline{\alpha}_t$ son funciones del conocido programa de varianza β_t y por lo tanto son conocidos y pueden ser pre-calculados. Esto permite, durante el entrenamiento, optimizar términos aleatorios de la función de pérdida L. Otra propiedad es que es posible reparametrizar la media para hacer a la red neuronal aprender (predecir), el ruido Gaussiano sumado (a través de $\epsilon_\theta(\mathbf{x}_t, t)$) por cada nivel de ruido t en los términos KL que constituye la pérdida, esto quiere decir que, la red neuronal pasa a ser un predictor de ruido, en lugar de un predictor de medias. La media puede ser calculada como:

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\overline{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, t)) \tag{2.61}$$

Entonces la función objetivo definitiva L_t estará definida como (para un paso de tiempo aleatorio t dado $\epsilon \sim \mathcal{N}(0, \mathbf{I})$):

$$||\epsilon - \epsilon_\theta(\mathbf{x}_t, t)||^2 = ||\epsilon - \epsilon_\theta(\sqrt{\overline{\alpha}_t\mathbf{x}_0} + \sqrt{(1-\overline{\alpha}_t)}\epsilon, t)||^2 \tag{2.62}$$

Aquí $\mathbf{x}_0$ es la imagen inicial, y se observa el ruido directo t para el proceso hacia delante. ϵ es el ruido Gaussiano puro muestreado en el paso de tiempo t, y $\epsilon_\theta(\mathbf{x}_t, t)$ es la red neuronal. La red neuronal es optimizada usando error medio cuadrático (MSE) entre la imagen real y el ruido Guassiano predicho. Por lo que el algoritmo de entrenamiento seguiría los siguientes pasos:

- Se toma una muestra aleatoria $\mathbf{x}_0$ de la distribución $q(\mathbf{x}_0)$
- Se muestrea un nivel de ruido t uniformemente entre 1 y T (paso de tiempo aleatorio)
- Se muestrea ruido de una distribución Gaussiana y se corrompe la entrada con este ruido en el nivel t

- La red neuronal es entrenada para predecir el ruido basado en la imagen corrupta $\mathbf{x}_t$ (ruido aplicado en la imagen $\mathbf{x}_0$ basado en el programa β_t)

La red neuronal

Esta red neuronal necesita tomar una imagen ruidosa en un paso de tiempo determinado y regresar el ruido predicho. Nótese que el el ruido predicho es un tensor que tiene el mismo tamaño (resolución) que la imagen de entrada. Así que técnicamente, la red toma como entradas y como salidas tensores del mismo tamaño. Así que una rquitectura que sigue este patrón es la arquitectura del Autocodificador. En particular, el autor en [40] optó por una U-Net, introducida por Ronneberger en [41], esta red también consiste en un *cuello de botella* en la mitad, ilustrada en la figura 2.33, que se asegura que la red aprenda solo la información más importante, además, introduce las conexiones residuales entre el codificador y el decodificador, mejorando significativamente el flujo del gradiente.

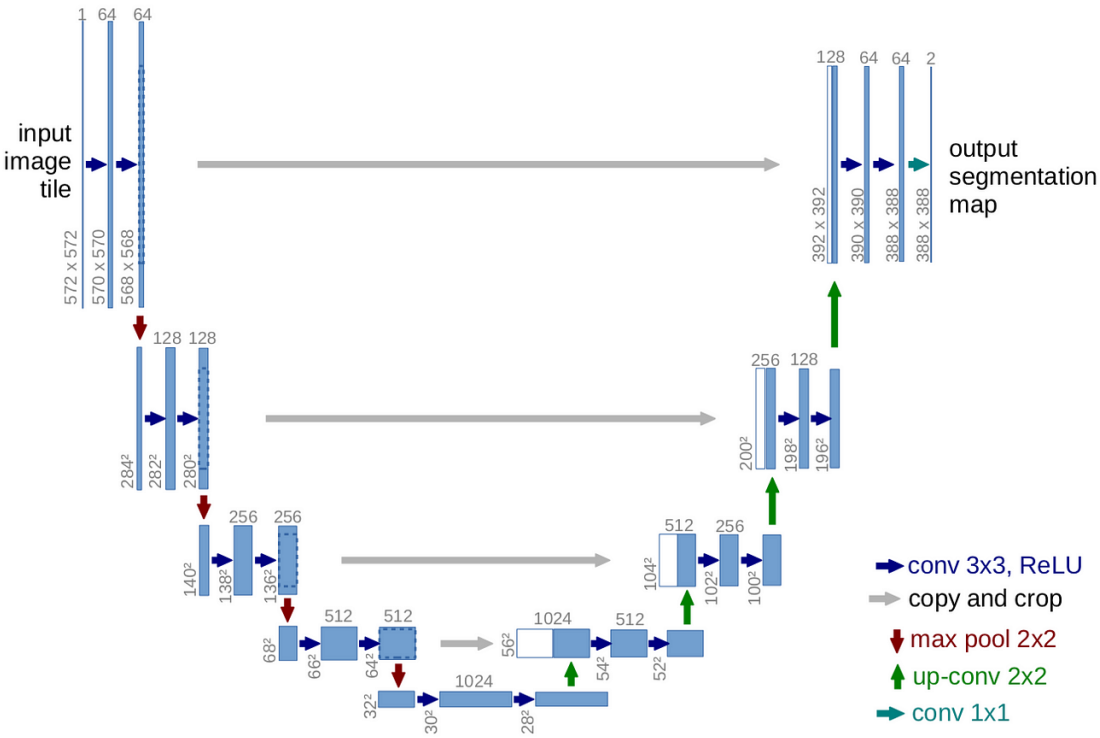


Figura 2.33: Arquitectura U-Net. Como se puede observar, primero se comprime la entrada en términos de resolución espacial y después se vuelve a agrandar, imagen tomada de [39]

2.6.5. Procesamiento de lenguaje natural (Natural Language Processing, NLP)

El procesamiento natural de lenguaje es la disciplina de construir máquinas que puedan manipular el lenguaje humano [42], o datos que se asemejen al lenguaje humano, en la forma en que se escribe, se habla y se organiza. Esta rama evolucionó a partir de la lingüística computacional, la cual usa ciencias de la computación para entender los principios del lenguaje, pero en lugar de desarrollar marcos teóricos, el NLP es una disciplina de la ingeniería que busca construir una tecnología que cumpla una tarea útil. El NLP puede ser dividido en dos subcampos que se traslapan : entendimiento del lenguaje natural (*Natural Language Understanding (NLU)*), que se enfoca en el análisis semántico o determina el significado del texto, y la generación de lenguage natural (*Natural Language Generation, NLG*), que se enfoca en la generación de texto por parte de una máquina. Cabe mencionar que el NLP está separado del reconocimiento de voz, que se encarga de descomponer el lenguaje hablado en palabras, convirtiendo el sonido en texto y viceversa.

Los modelos de NLP funcionan encontrando relaciones entre las partes que constituyen el lenguaje, por ejemplo letras, palabras y oraciones encontradas en un conjunto de datos. Las arquitecturas de NLP usan distintos métodos para el procesamiento de datos, extracción de características y modelado.

Hay una cantidad extensa de literatura que trata sobre NLP. En esta sección se presenta solo el marco teórico sobre el tema de NLP que es relevante al proyecto; fácilmente se podrían escribir varias tesis al respecto y van más allá del enfoque principal. En el capítulo 4 se presentan mayores detalles sobre los datos utilizados y su procesamiento para cumplir los objetivos del proyecto.

Procesamiento de datos

Antes de que un modelo procese el texto para una tarea específica, el texto usualmente necesita ser pre-procesado para mejorar el desempeño del modelo o para convertir palabras y caracteres en un formato que el modelo pueda entender. Algunas de las técnicas usadas en procesamiento de datos son:

Desdoblamiento y Lematización (Stemming and Lematization): El desdoblamiento es un proceso informal donde se convierten palabras a sus formas base usando reglas heurísticas. Por ejemplo, *universidad, universidades , universitario* podrían ser todas mapeadas a la base *univers*. La lematización es una manera más formal de encontrar raíces por medio del análisis la morfología de la palabra usando el vocabulario de un diccionario.

Tokenización: La tokenización separa el texto en palabras individuales y fragmentos de palabras. El resultado generalmente consiste en un índice de palabra y el texto tokenizado en cuyas palabras pueden ser representadas como tokens numéricos para el uso de distintos métodos de aprendizaje profundo.

Extracción de características:

Las técnicas más convencionales del machine learning trabajan sobre las características, generalmente números que describen un documento en relación al corpus que lo contiene.

Modelado:

Después de que los datos son preprocesados, son alimentados a una arquitectura de NLP que modele la información para lograr una variedad de tareas.

2.7. Métricas para audio

La evaluación subjetiva sobre el desempeño de sistemas generativos de audio es una tarea tediosa. Es preciso tener una manera de evaluar los sistemas de forma objetiva, con base en las características físicas y matemáticas de la señal, para medir la diferencia entre las muestras de audio reales y las muestras de audio generadas por un modelo. En este trabajo se usaron métricas derivadas de la Distancia de Fréchet [43] y la divergencia KL [44], las cuales son explicadas teóricamente a continuación y aplicadas en el capítulo 6.1 para evaluar la generación de las distintas técnicas.

2.7.1. Distancia de Fréchet

La distancia de Fréchet [43] es una medida popular entre dos curvas poligonales que toma en cuenta el orden de los puntos a lo largo de la curva. Intuitivamente, la similitud de la forma de ambas curvas es medida. En una forma más rigurosa, la definición del problema se puede definir como: una curva poligonal es una curva $P : [1, n] \to \mathbb{R}^2$ donde n es un entero positivo, tal que la restricción de P al intervalo $[i, i+1]$ es afín, eso es $P(i+\lambda) = (1-\lambda)P(i) + \lambda P(i+1)$, para cada $i \in \{1, 2, ..., n-1\}$. Se le llama a $\{P(i)|i \in [1 : n]\}$ el conjunto de vértices de P. Para medir la disimilitud entre dos curvas poligonales, se utiliza la distancia de Fréchet. Para dos curvas poligonales $P, Q : [1, n] \to \mathbb{R}^2$ y $Q : [1, m] \to \mathbb{R}^2$ su **distancia de Fréchet** está definida como:

$$\delta_F = (P, Q) = \text{ínf máx} \|P(\alpha(t)) - Q(\beta(t))\| \tag{2.63}$$

$$\alpha : [0, 1] \to [1, n] t \in [0, 1] \tag{2.64}$$

$$\beta : [0, 1] \to [1, m] \tag{2.65}$$

Donde α (resp. β) abarca todos los mapeos no decrecientes desde $[0, 1]$ a $[1, n]$ (resp. $[, m]$).

La distancia de Fréchet está típicamente definida mediante *acoplamientos*. Sea $u(P)$ el conjunto vértice de P y $v(Q)$ el conjunto vértice de Q. Un *acoplamiento* L entre P y Q es una secuencia $(u_{a1}, v_{b1}), (u_{a2}, v_{b2}), ..., (u_{aj}, v_{bj})$ de distintos pares de $u(P) \times v(Q)$, tal que $a_1 = 1, b_1 = 1, a_j = n, b_j = m$, y para toda $i \in \{1, 2, ..., j\}$ se

tiene $a_{i+1} = a_i$ ó $a_{i+1} = a_i + 1$ y $b_{i+1} = b_i$ ó $b_{i+1} = b_i + 1$. La longitud de L está definida como el $\max_i d(u_{a_i}, v_{b_i})$, donde d denota la distancia Euclideana entre dos puntos. La distancia discreta de Fréchet entre P y Q es igual a la menor longitud posible entre P y Q.

Usualmente, la distancia de Fréchet es ilustrada por un hombre caminando con su perro. Aquí, la distancia de Fréchet es igual a la menor longitud de una correa que permite y al perro caminar en sus propias curvas de principio a fin.

2.7.2. Distancia de audio de Fréchet

A diferencia de otras métricas de audio existentes, la distancia de audio de Fréchet (*FAD*) [45], no mira a los clips de audio individuales. En su lugar, compara *embeddings* estadísticos generados en todo el conjunto de evaluación con *embeddings* estadísticos generados en un gran conjunto de *música limpia* (e.g. el conjunto de entrenamiento). Esto hace a FAD una medida libre de referencia, que puede ser utilizada para puntuar un conjunto de evaluación, donde no se dispone de la señal de audio de referencia (*ground truth*). La FAD utiliza el modelo **VGGish** [46] con el que obtiene un conjunto de *embeddings de fondo* de la *música limpia* y otro conjunto de *embeddings de evaluación* de la salida del modelo a evaluar. Para esto, se debe de calcular las distribuciones Gaussianas multivariantes en el conjunto de evaluación $\mathcal{N}_e(\mu_e, \Sigma_e)$ y los embeddings de fondo $\mathcal{N}_b(\mu_b, \Sigma_b)$. Dowson et al. [47] mostraron que la distancia de audio de Fréchetentre dos Gaussianas está dada por:

$$\mathbf{F}(\mathcal{N}_b, \mathcal{N}_e) = \|\mu_b - \mu_e\| + tr(\Sigma_b + \Sigma_e - 2\sqrt{\Sigma_b \Sigma_e}) \tag{2.66}$$

Donde tr es la traza de una matriz. Cuando se comparan los modelos, tanto los *embeddings* de fondo como el conjunto de señales ruidosas del conjunto de evaluación que pasan como entrada al modelo son fijos. A menudo se suele referir a la FAD calculada entre los *embeddings del conjunto de evaluación* sin ruido y los *embeddings de fondo* como la puntuación FAD de un modelo.

2.7.3. Modelo de *embeddings* del FAD

El modelo utilizado para obtener los *embeddings* es **VGGish**, el cual está derivado de la arquitectura de reconocimiento de imágenes VGG [48], y fue entrenado en un conjunto de datos de videos de YouTube, similar a YouTube-8M [49], como un clasificador de audio con alrededor de 3000 clases. Las activaciones de una capa de 128 dimensiones previa a la capa de clasificación final son usadas como *embeddings*. La entrada al modelo **VGGish** consiste de 96 *frames* consecutivos de características *log-mel* extraídas del espectrograma de magnitud calculado sobre un segundo de audio. Dada esa condición, el requerimiento de entrada de un segundo es considerablemente menor que una típica evaluación de clips de música.

2.7.4. Divergencia de Kullback-Liebler

Sean un par aleatorio (X, Y) con $Y \in \{0, 1\}$, y $X \in \mathbf{D}^d$ sea un vector con d dimensiones en un espacio discreto $\mathbf{D} = \{1, 2, ..., M\}$. La divergencia de Kullback-Liebler entre dos clases está definida como [44]:

$$\mathbf{I} = \mathbf{E}\left[\eta(\mathbf{x}) \log \frac{\eta(\mathbf{x})}{1 - \eta(\mathbf{x})}\right] \tag{2.67}$$

Donde $\eta(\mathbf{x}) = \mathbf{P}(Y = 1|\mathbf{x})$ es la probabilidad a posteriori de la clase $Y = 1$. Para entender el significado, es preciso notar que la divergencia es cero solo cuando $\eta(\mathbf{x}) = 1/2$, y tiende a infinito cuando $\eta(\mathbf{x})$ tiende a 1 y cuando $\eta(\mathbf{x})$ tiende a cero. Sin embargo, en una situación práctica donde las probabilidades de las clases a priori no pueden ser estimadas y son asumidas como iguales, se utilizan únicamente las probabilidades condicionales de clase $\mathbf{P}(X|Y = 0)$ y $\mathbf{P}(X|Y = 1)$, de modo que la divergencia de Kullback-Liebler se puede calcular de la forma:

$$\mathbf{H}(\lambda) = \sum_{\mathbf{x}} \{\mathbf{P}(\mathbf{x}|Y = 1) - \lambda \mathbf{P}(\mathbf{x}|Y = 0)\} \log \frac{\mathbf{P}(\mathbf{x}|Y = 1)}{\mathbf{P}(\mathbf{x}|Y = 0)} \tag{2.68}$$

Donde λ se fija en cero para la divergencia KL y en 1 para la divergencia de Jeffrey, la cual es su contra parte simétrica. Esta forma de divergencia es una medida no negativa, la diferencia entre estas dos probabilidades condicionales es cero solamente cuando las probabilidades son idénticas, en cuyo caso la probabilidad de error es de $1/2$, y tiende a infinito a medida que las probabilidades difieren entre sí, en cuyo caso la probabilidad de error tiende a cero. Visto de otra forma, mide la capacidad de discriminación subyacente de la naturaleza de las observaciones, descritas por estas funciones de probabilidad condicional.

Capítulo 3

Metodología

Como se estableció en secciones anteriores, el objetivo de este trabajo es desarrollar un algoritmo que permitan tomar una entrada contextual de lenguaje natural (texto) y generar un audio que sea contextual mente acorde con esa entrada. La intuición del modelo es encontrar una representación numérica que logre extraer la información contextual y semántica del texto introducido, para que se pueda introducir a otro modelo, de tal forma que funja como condicionante en la etapa de generación. De forma que, una vez se extraiga esta información semántica y pase por el modelo generativo, se obtenga un espectrograma que posteriormente será convertido en una señal de audio (forma de onda), de características *one-shot*, es decir, que contenga un solo instrumento y sea de unos pocos segundos de duración, el diagrama del proceso se ilustra en la figura 3.1.

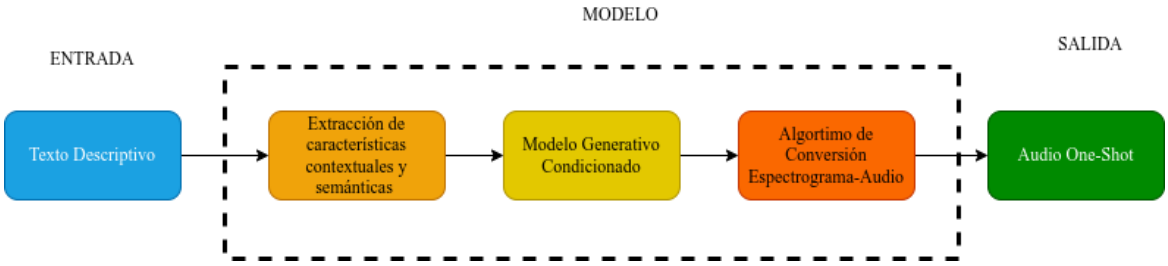


Figura 3.1: Estructura general del modelo objetivo para la generación de audio a partir de lenguaje natural.

Por lo tanto, la estructura general del modelo podría dividirse en tres secciones:

Extracción de características contextuales y semánticas

Esta es la etapa de entrada del modelo, en la cual se toma la entrada de texto y se convierte en una representación numérica, que posteriormente sirve como parámetro de condicionamiento en un modelo generativo.

Modelo Generativo Condicionado

Es la etapa del modelo en la que, con base en la información contextual extraída del texto de entrada, se genera un espectrograma correspondiente con el *prompt* introducido. La razón de porque se genera un espectrograma se explica más adelante en esta capítulo.

Algoritmo de conversión espectrograma-audio

Esta es la etapa final del modelo, aquí se toma el espectrograma generado por el modelo generativo, y se transforma a una forma de onda que es entendible por un humano mediante la audición.

3.1. Metodologías para la solución del problema

Este documento aborda cuatro diferentes técnicas de generación de audio condicionado por lenguaje natural (texto), los cuales están basados en procesamiento de imágenes y representaciones latentes del conjunto de datos con el que se trabajó, además se explican intuiciones que se tuvieron pero que no llevaron a resultados satisfactorios dentro del objetivo establecido. Las técnicas utilizadas fueron:

- ChopAN (*Chopped Audio Network*)
- Audio-Diffusion
- Riffusion
- TANGO (*Text to Audio using iNstruction-Guided diffusiOn*)

Los detalles de cada técnica se explican a continuación en las secciones subsecuentes. Sin embargo, previo a las vicisitudes de cada técnica es preciso explicar porque se optó por trabajar con espectrogramas y no otro tipo de representación de una señal de audio. Esta decisión tiene injerencia directa en el planteamiento de las cuatro técnicas y es necesario comprender antes de abordar los detalles de cada modelo.

¿Por qué espectrogramas?

El campo de los modelos generativos ha tomado particular importancia en recientes años, gracias a el avance en arquitecturas de aprendizaje y la combinación con otras ya existentes. En el campo del audio, previo a la existencia de los modelos generativos condicionados, existían modelos denominados no condicionados, los cuales podían generar fragmentos de audio, dada una muestra de audio previa, tal como WaveNet [50] el cual era el estado de arte en el año 2016. En el artículo original de WaveNet introducen un método de generación basado en la probabilidad conjunta de de forma de onda $\mathbf{x} = \{x_1, ..., x_T\}$ se factoriza como el producto de probabilidades condicionales de la forma:

$$p(\mathbf{x}) = \sum_{t=1}^{T} p(x_t|x_1, ..., x_{t-1}) \tag{3.1}$$

Donde cada muestra de audio x_t está condicionada en las muestras de audio anteriores. Este modelo está basado en convoluciones dilatadas y algo muy importante de resaltar es que trabaja a nivel de forma de onda, es decir, que el tamaño de la muestra a procesar está en función de su duración, su tasa de muestreo y el número de canales del audio introducido. Esto significa que si su usara una señal de un solo canal (mono), con una tasa de muestreo de $16KHz$ y un segundo de duración, se tendrían que generar $16,000$ muestras solo para generar un segundo de sonido, lo cual incrementaría proporcionalmente por cada segundo de audio deseado, y se duplicaría por cada canal extra que se incorporase. Esto deja en evidencia uno de los mayores problemas de este modelo, es decir, la complejidad computacional que implica trabajar con formas de onda para la generación de señales de audio.

Posteriormente en 2019, se presentó un artículo llamado MelNet [51], el cual también es un modelo probabilístico autoregresivo, el cual factoriza la distribución de probabilidad conjunta de un espectrograma x como un producto de distribuciones condicionales de la forma:

$$p(x) = \sum_{i} \sum_{j} p(x_{ij}|x < ij; \theta_{ij}) \tag{3.2}$$

Donde θ_{ij} parametriza una densidad univariada sobre x_{ij}. De modo que, la idea detrás de MelNet es similar a WaveNet, pero con MelNet se reduce considerablemente la cantidad de datos con las que se busca trabajar, ya que considerando una señal de un solo canal, con tasa de muestreo de $16KHz$ y un segundo de duración, si se trabaja con espectrogramas, este tipo de señal puede representar con una imagen de 64×64, resultando en 4096 muestras que procesar a diferencia de las $16,000$ que se necesitaba con WaveNet, además de que para audios más grandes (en la escala de varios segundos), una resolución de imagen de 512×512 es más que suficiente, por lo que la escalabilidad del modelo resulta mucho mejor que aquellos que trabajan con formas de onda. Por lo tanto, esto se convierte en un problema de generación de audio que se ataca mejor desde el campo del procesamiento de imágenes.

3.2. ChopAN (*Chopped Audio Network*)

Este es un modelo basado en CLIP [52] y representaciones latentes. La idea es tener un conjunto de espectrogramas que sirva como **base de datos de características**, de las cuales CLIP selecciona los **5 espectrogramas con mayor afinidad** al texto provisto. Estos cinco espectrogramas son sumados con la finalidad de obtener una representación (espectrograma) que **combine las características** de las 5 muestras obtenidas con CLIP. Posteriormente, mediante un VAE, se **mapea un espacio latente de características** y se obtiene un espectrograma que contiene las características más relevantes de los 5 espectrogramas, por último es **convertido a una señal de audio** mediante el algoritmo de Griffin-Lim (ver figura 3.2).

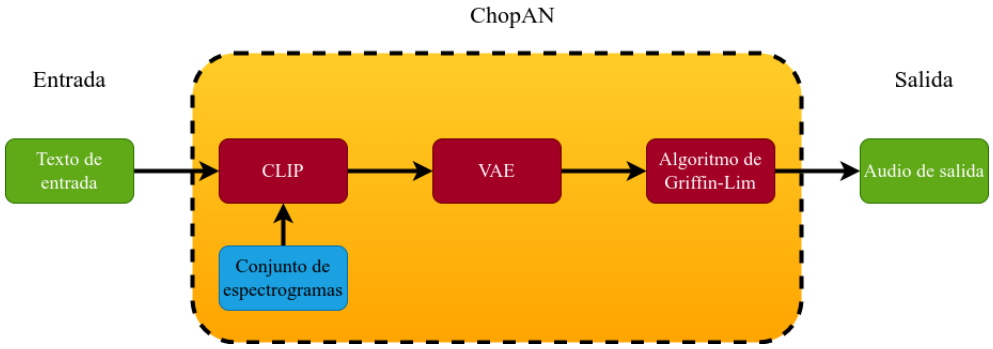


Figura 3.2: Diagrama general de ChopAN. Se observa el flujo que trabajo que tiene el modelo.

3.2.1. Extracción de características contextuales y semánticas

Para la extracción de información del lenguaje natural, se utilizó CLIP [52]. CLIP es un clasificador *zero-shot* desarrollado por openAI, el cual fue entrenado con 400 millones de pares imagen-texto en la tarea de clasificación de imágenes basadas en texto. Dados los pares imagen-texto, se tienen dos codificadores, un codificador de imagen basado en ResNet y *vision transformers* y un codificador de texto basados en transformers y modelos de GPT. De ahí que, una vez pasados los textos por el codificador, se obtienen los *embbedings*, los cuales tienen el tamaño del lote que se utilizó. De igual manera para el codificador de imágenes se obtiene una representación en el espacio de *embeddings* que representa a la señal. Una vez se encuentran representadas las señales en el espacio de embeddings, se obtiene una matriz, en la cual se busca que cada par texto-imagen tenga una similitud coseno de 1 y todas las demás combinaciones tengan similitud de 0, mostrado así en la figura 3.3.

CLIP está basado en el artículo de Yuhao Zhang [54], en la cuál también se busca hacer clasificación de pares imgen-texto, pero para un enfoque médico. Donde se muestrea un mini lote de N pares de entrada $(\mathbf{x}_v, \mathbf{x}_u)$ del conjunto de entrenamiento, y se calculan sus representaciones $(\mathbf{v}, \mathbf{u})$. Se usa $(\mathbf{v}_i, \mathbf{u}_i)$ para denotar el *i-ésimo* par. El objetivo involucra dos funciones de pérdida. La primera es una función de pérdida contrastada imagen-texto par el *i-ésimo* par:

$$\ell_i^{(v\to u)} = -\log \frac{\exp(\langle \mathbf{v}_i, \mathbf{u}_i \rangle / \tau)}{\sum_{k=1}^{N} \exp(\langle \mathbf{v}_i, \mathbf{u}_i \rangle / \tau)} \tag{3.3}$$

Donde $\langle \mathbf{v}_i, \mathbf{u}_i \rangle$ representa la similitud coseno, es decir, $\langle \mathbf{v}, \mathbf{u} \rangle = \mathbf{v}^T\mathbf{u}/||\mathbf{v}||||\mathbf{u}||$; y $\tau \in \mathbb{R}^+$ representa el parámetro de temperatura. Esta pérdida toma la misma forma que la pérdida InfoNCE(*Noise-Contrastive Estimation*) [55], y su minimización conduce a codificadores que preservan al máximo la información mutua entre los verdaderos pares texto-imagen bajo las funciones de representación (embeddings). Nótese que la función de pérdida contrastada es asimétrica para cada modalidad de entrada.

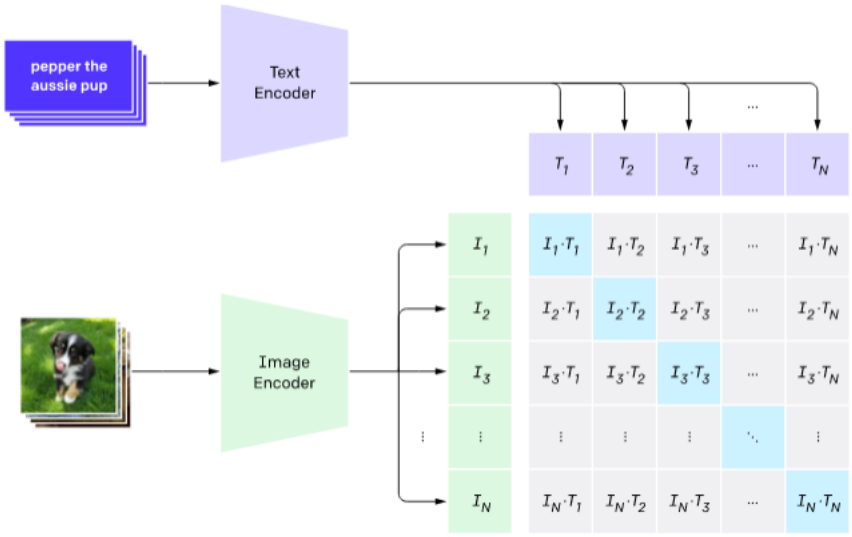


Figura 3.3: Obtención de matriz de similitudes mediante el producto punto de los embeddings provenientes de el codificador de texto (*Text Encoder*) y el codificador de imagen (*Image Encoder*) para un par texto-imagen, imagen tomada de [53]

Por lo tanto se define una función de pérdida similar:

$$\ell_i^{(u \to v)} = -\log \frac{\exp(\langle \mathbf{u}_i, \mathbf{v}_i \rangle / \tau)}{\sum_{k=1}^{N} \exp(\langle \mathbf{u}_i, \mathbf{v}_i \rangle / \tau)} \tag{3.4}$$

De tal forma que la función de pérdida final se calcula como una combinación ponderada de ambas funciones de pérdida promediadas sobre todos los pares texto-imagen verdaderos en cada mini lote:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (\lambda \ell_i^{v \to u} + (1 - \lambda) \ell_i^{v \to u}) \tag{3.5}$$

Donde $\lambda \in [0, 1]$ es un peso escalar. En otras palabras, se obtiene la similitud coseno sobre las filas y las columnas de la matriz de similitudes y la función de pérdida obtiene la similitud ponderada sobre todo el mini lote de combinaciones texto-imagen.

Este procedimiento de función de pérdida en función de la similitud fue utilizado por OpenAI para el entrenamiento del modelo, pero al ser un clasificador *zero-shot*, es decir, realiza tareas de clasificación sin requerir ejemplos de entrenamiento específicos para cada clase, puede realizar inferencias sobre imágenes que no haya visto antes. Lo cual resulta ideal para el problema que busca solucionar este trabajo.

La inferencia se realiza tomando un *prompt* (texto) de entrada, al cual se le hace pasar por el mismo codificador de texto, para encontrar sus embeddings. Lo mismo ocurre con la imagen que se busque clasificar, se obtiene su embedding y se calcula el mismo producto punto, de tal forma que el texto con el que tenga mayor similitud, será la clase a la que pertenece, este proceso se puede observar mejor en la figura 3.4.

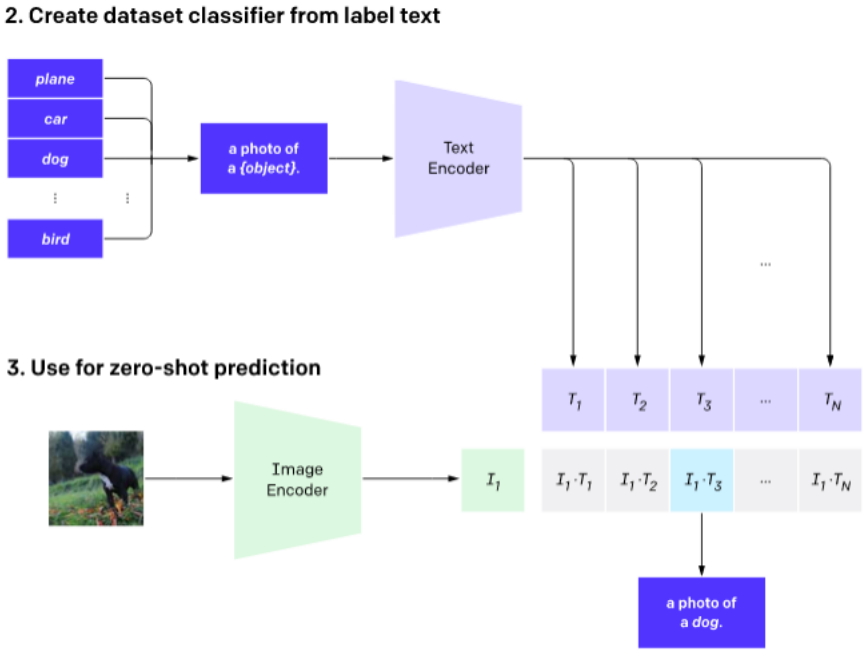


Figura 3.4: Clasificación *zero-shot* para una imagen dado un conjunto de embeddings de diferentes clases, imagen tomada de [53]

Este modelo, de acuerdo con OpenAI, comparte la característica de los modelos GPT de que a partir de un entrenamiento base, es capaz de aprender otras tareas par las cuales no fue explícitamente entrenado para hacer.

3.2.2. Combinación de características

En esta técnica, se tiene un conjunto de espectrogramas, los cuales fueron seleccionados previamente y cuentan con características diferentes (tipos de instrumento, tiempo de liberación, ataque, etc.), y que pertenecen al conjunto de datos. Se comienza introduciendo un texto, el cual mediante CLIP se encuentran los 5 espectrogramas con más similitud. Estos 5 espectrogramas son sumados para obtener un espectrograma final que combine todas las características de las señales:

$$\begin{aligned} Spec(t,\omega) = k_1 \cdot |\tilde{f}_g(t,\omega)|^2 + k_2 \cdot |\tilde{f}_g(t,\omega)|^2 + k_3 \cdot |\tilde{f}_g(t,\omega)|^2 \\ + k_4 \cdot |\tilde{f}_g(t,\omega)|^2 + k_5 \cdot |\tilde{f}_g(t,\omega)|^2 + \mathcal{Z} \end{aligned} \tag{3.6}$$

Donde k_n es un escalar de peso que determina la influencia de cada espectrograma en el espectrograma final. Cada valor de k_n es fijo y está determinado por la similitud que tiene con el *prompt* introducido, es decir, el espectrograma con mayor similitud tendrá un k_n más grande que el espectrograma con menor similitud (dentro de los cinco antes mencionados). Y $\mathcal{Z}$ corresponde a una muestra de ruido Gaussiano de una distribución normal $\mathcal{N}(\mu, \sigma)$, que introduce aleatoriedad al espectrograma.

3.2.3. Representaciones latentes (Modelo Generativo)

Para obtener las representaciones latentes se optó por un Auto codificador Variacional (VAE), debido a que permite tener un espacio latente de características. El VAE toma como entrada un espectrograma que contiene las información espectral de las 5 muestras con mayor afinidad al texto de entrada. Al reducir la dimensionalidad de la señal de entrada, el codificador del VAE extrae la información más importante del espectrograma de entrada, reduciéndolo a un *vector latente* de dimensionalidad menor. Dicho vector es muestreado del espacio latente y posteriormente introducido en el decodificador, el cual pasa de la representación de menor dimensionalidad (*vector latente*) a una representación con información espectral (espectrograma). La elección de un VAE por sobre un Auto Codificador simple se tomó debido a las ventajas que supone sobre este. Las principales virtudes del VAE son que el espacio latente es simétrico al rededor del origen, todas las etiquetas son representadas en un tamaño similar dentro del espacio latente y no existen (casi) discontinuidades en el espacio latente. De modo que, se garantiza que la representación generada no esté sesgada hacia un tipo de característica en específico, y que no importa donde se encuentre localizada la representación latente (dentro del espacio latente), generará muestras de buena calidad.

El VAE se trata específicamente de un Autocodificador Convolucional Profundo (*Deep Convolutional Autoencoder*), es decir, tanto el codificador como el decodificador están compuestos por varias capas convolucionales. Antes de entrar a detalle en la estructura del VAE, es necesario mencionar que el conjunto de espectrogramas que se ven ejemplificados en la figura 3.2, son obtenidos a partir de una base de datos de diferentes archivos de audio, los cuales son pasados por una etapa de pre-procesamiento (la cual se explica más adelante en este documento). Una vez los archivos de audio son pasados por la etapa de pre-procesamiento, el resultado son espectrogramas en escala Mel con 64 frames de la STFT en el eje x y 256 bins de frecuencia en el eje y, a los cuales se les agrega una dimensión extra para que puedan ser tratados por la red como si fueran una imagen en escala de grises, por lo tanto, el espectrograma resultante tiene unas dimensiones de $[256, 64, 1]$. A continuación se aborda a detalle la estructura del Autocodificador Variacional utilizado en este modelo.

Codificador

El Codificador del VAE está compuesto por una capa de entrada, cinco capas convolucionales y dos capas densas de salida que actúan como el cuello de botella en este modelo, ver figura 3.5. La capa de entrada es una capa que tiene las mismas dimensiones que el espectrograma, es decir, $[256, 64, 1]$. Dicha capa toma el espectrograma y lo entrega a las capas convolucionales subsecuentes. Esta etapa del modelo cuenta con cinco capas convolucionales, donde cada capa convolucional está compuesta de una capa de convoluciones $2D$, una función de activación ReLU y una capa de normalización por lotes (*batch normalization*). Cada capa convolucional tiene $512, 256, 128, 64, 32$ *kernels* respectivamente, el tamaño de los *kernels* es de tres y tienen un stride de dos, hay que recordar que en una capa convolucional con un

stride de dos, se está haciendo un *downsampling* a la señal, por lo que se reduce el tamaño de la señal a la mitad. Una vez se pasa por las cinco capas convolucionales, en el cuello de botella, se tiene dos capas densas, las cuales reciben los datos de la última capa convolucional y los aplanan (*Flatten*) con una red densa con un muestreo Gaussiano. De estas dos capas densas, una representa la media $\vec{\mu}$ y otra representa la varianza logarítmica $\log \vec{\sigma}$, de modo que se muestrea un punto en el espacio latente $z = \vec{\mu} + \Sigma\epsilon$. El vector de medias $\vec{\mu}$, es obtenido mediante la capa densa, ϵ es un punto muestreado de una distribución normal estándar $\mathcal{N}(\mu = 0, \sigma = 1)$ y Σ, que es el vector de varianzas logarítmicas, se obtiene con: $\Sigma = e^{\frac{\log(\vec{\sigma^2})}{2}}$. El tamaño de la representación latente z es de $[1, 128]$, es decir, se redujo la señal (espectrograma) de una dimensión de $[256, 64, 1]$ a una representación latente de dimensión $[1, 128]$.

Decodificador

Una vez se consigue tener una representación latente, esta pasa a través del decodificador del VAE. Este toma la representación latente z y la convierte de nuevo a un espectrograma de dimensiones $[256, 64, 1]$, ver figura 3.5. El decodificador está compuesto por una capa densa que tiene las dimensiones de la representación latente, posteriormente pasa a una capa de redimensionamiento, donde toma la forma que tenía en el codificador antes de pasar a las capas densas de $\vec{\mu}$ y $\log \vec{\sigma}$. Una vez adquiere esta forma pasa por cinco capas convolucionales, donde cada capa está compuesta de una capa de convolución transpuesta $2D$, una función de activación ReLU y una capa de normalización por lotes. Las convoluciones transpuestas tienen la misma cantidad de *kernels*, el mismo tamaño de *kernels* y *padding* que sus capas correspondientes en el codificador. Una vez termina de pasar por las capas convolucionales, la señal llega a la capa de salida. La capa de salida toma la señal que proviene de la ultima capa convolucional y le da la forma esperada de $[256, 64, 1]$, por lo que solo cuenta con un *kernel*, que asegura que se tenga un sola dimensión de salida, es decir que tenga un alto, un ancho y un canal de imagen ($[H, W, C]$), recordemos que se trata a la señal como si fuera una imagen en escala de grises. La capa de salida está compuesta por una capa de convolución transpuesta y una función de activación sigmoide, y el tamaño del *kernel* y el *stride* serán fijados como sus respectivos tamaños en la primera capa del codificador, que en este caso serán tres y dos respectivamente.

Autocodificador

Con este tipo de arquitectura, ChopAN es capaz de extraer características relevantes a partir de una señal (espectrograma), mapear esa información dentro de un espacio donde viven estas características (espacio latente) y devolver una señal nueva que contiene los rasgos más importantes de la señal de entrada combinados. La estructura, el número de capas, las dimensiones del espacio latente y capas convolucionales están basados en los trabajos de Valerio Velardo [56], donde con esta configuración, se obtuvieron muestras de audio de personas hablando.

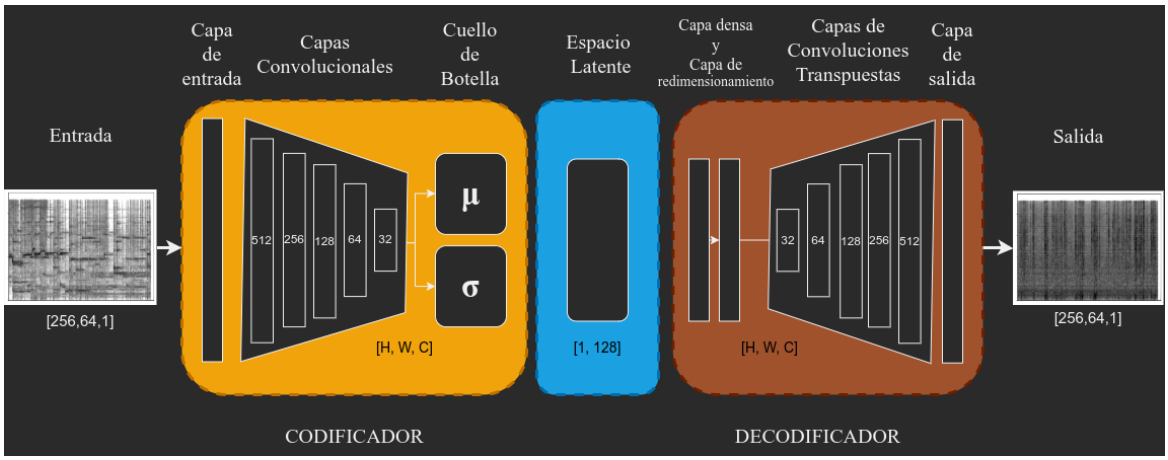


Figura 3.5: Diagrama de el Autocodificador Variacional. El decodificador recibe un espectrograma de entrada, lo comprime a una representación con menor dimensionalidad, posteriormente el decodificador lo transforma de vuelta a una señal de las dimensiones que el espectrograma original.

3.2.4. Algoritmo de conversión espectrograma-audio

En un principio, las señales de audio fueron convertidas a una representación bidimensional mediante el uso de la transformada en tiempo corto de Fourier, por lo que la forma evidente de volver de la representación de espectrograma, a una señal de audio sería mediante la operación inversa, ISTFT, sin embargo, este proceso no es trivial y si se aplican directamente los pasos en forma inversa, deriva en señales de audio muy ruidosas o que no representan la señal original. Por lo que en 1984, D. Griffin y Jae Lim, desarrollaron el algoritmo que a día de hoy se conoce como el algoritmo de Griffin-Lim [57]. Dicho algoritmo espera recuperar un espectrograma de valores complejo, que sea coherente y que mantenga la amplitud A dada, esto se logra mediante:

$$X^{[m+1]} = P_{\mathcal{C}}(P_{\mathcal{A}}(\mathbf{X}^{[m]})) \tag{3.7}$$

Donde $\mathbf{X}$ es el espectrograma de valores complejos actualizado a través de las iteraciones, P_S es la métrica de proyección en un conjunto S, y m es el índice de iteración. Aquí $\mathcal{C}$ es el conjunto de de espectrogramas consistentes, y $\mathcal{A}$ es el conjunto de espectrogramas cuyas amplitudes son las misma que el espectrograma dado. La métrica de proyección sobre los conjuntos $\mathcal{C}$ y $\mathcal{A}$ están dados por:

$$P_{\mathcal{C}}(\mathbf{X}) = \mathcal{G}\mathcal{G}'\mathbf{X} \tag{3.8}$$

$$P_{\mathcal{A}}(\mathbf{X}) = \mathbf{A} \odot \mathbf{X} \oslash |\mathbf{X}| \tag{3.9}$$

Donde $\mathcal{G}$ representa la STFT y $\mathcal{G}'$ es la pseudo inversa de la STFT (ISTFT), $\odot$ y $\oslash$ son funciones de multiplicación y división, respectivamente, que operan elemento por elemento, donde la división por cero es reemplazada por cero. El algoritmo de Griffin-Lim se obtiene como un problema de optimización del siguiente problema:

$$\min_{\mathbf{X}} \|\mathbf{X} - P_{\mathcal{C}}(\mathbf{X})\|_{\text{Fro}}^2 \quad \text{s.t.} \mathbf{X} \in \mathcal{A} \tag{3.10}$$

Donde $\|\cdot\|_{\text{Fro}}$ es la norma de Frobenius. Esta ecuación minimiza la energía de las componentes inconsistentes bajo la condición de amplitud que debe ser igual a la dada. Es usual que que este algoritmo requiera de varias iteraciones para converger en un espectrograma concreto y resulta en reconstrucciones de baja calidad. Esto es debido a que la función de costo solo requiere la consistencia, y las características de la señal objetivo no son tomada en cuenta. Y en este trabajo, la implementación del algoritmo de Griffin-Lim que se utiliza es la del modulo de Python Librosa. De modo que, con este último paso, se obtiene una señal de audio (*waveform*) que selecciona características basado en el *prompt* de entrada.

3.3. Audio Diffusion y Audio Latent Difussion

Esta técnica está basada en modelos de difusión y representaciones latentes. Para llegar a este tipo de implementación se observó la eficiencia de los modelos de difusión para la generación de imágenes, específicamente se tomó el enfoque utilizado en CompVis [58], el cual es un generador de imágenes de alta resolución que utiliza modelos de difusión. A este tipo de arquitectura se les denomina **Modelos de Difusión Latente** (*Latent Diffusion Models*, LDM), debido a que trabajan con un representación de menor dimensionalidad de la imagen de entrada (espectrograma). Sus dos componentes principales son, un VAE que se encarga de obtener estas representaciones de menor dimensionalidad y el modelo de difusión que actúa como el proceso generativo.

La intuición de esta técnica es, tomar una entrada de texto y obtener una representación numérica que sea representativa, contextual y semánticamente, de cada texto de entrada provisto por el usuario. Esto se logra mediante la *tokenización* de dicho texto, esta *tokenización* servirá posteriormente como condicionante para el modelo de difusión. Por otro lado, se cuenta con un VAE, cuyo codificador se encarga de obtener una representación latente (de menor dimensionalidad) de las imágenes de entrada (espectrogramas), esta representación es pasada al modelo de difusión, el cual consiste en una U-Net [41], que recibe la representación numérica del texto de entrada y genera una representación latente condicionada por el prompt de entrada. Posteriormente, la representación obtenida del modelo de difusión pasa a través del decodificador del VAE, de donde se obtiene una imagen (espectrograma) de las mismas dimensiones que la imagen de entrada. Este espectrograma, es convertido a una señal de audio (forma de onda) mediante el algoritmo de Griffin Lim.

Con fines de exploración y comparación, se optó por tener dos tipos de configuraciones, la primera en la que solamente se tiene el modelo de difusión (U-Net), ver figura 3.6, y la segunda, que cuenta con el VAE y el modelo de difusión, ver figura 3.7, ambas metodologías se explican a continuación.

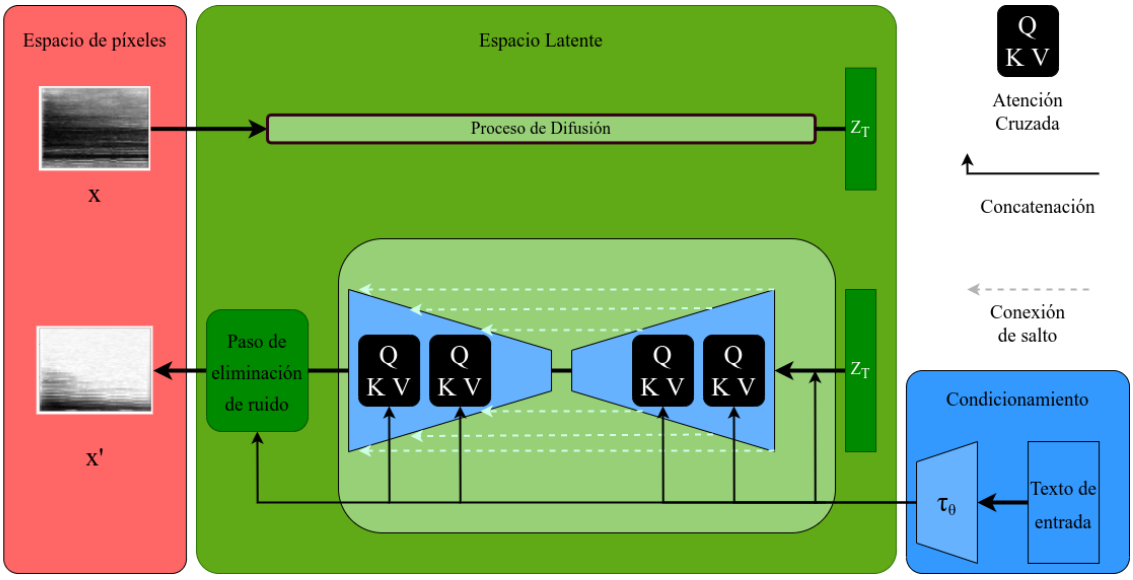


Figura 3.6: Diagrama de técnica Audio Difussion, utilizando únicamente modelos de difusión.

3.3.1. Text-To-Text Transfer Learning (T5)

En esta técnica, T5 actúa como extractor de características semánticas y contextuales. T5 es un modelo codificador-decodificador[59], que convierte problemas de Procesamiento de Lenguaje Natural (NLP), en problemas con formato texto a texto. Fue presentado en 2020 por Google AI, y T5 significa *Text-To-Text Transfer Learning*. T5 introdujo un marco de trabajo (framework) texto a texto, en donde cada tarea de NLP tiene la misma estructura subyacente, en donde el texto es alimentado como entrada y texto es producido como salida. El modelo funciona usando la misma estructura estándar de codificador-decodificador vista en los modelos transformers estándar [60]. El modelo base está diseñado para qeu el codificador y decodificadorsean similares en tamalo y configuración a $BERT_{BASE}$ [61]. Específicamente, el codificador y decodificador consisten de 12 bloques. Cada bloque contiene autoatención, una red feed-forward, y un modulo opcional codificador-decodificador con atención. La red *feed-fordward* en cada bloque consiste de una capa densa con una dimensionalidad de salida de $d_{ff} = 3072$, seguida por una función de activación no lineal ReLU y otra capa densa. Las matrices de *"key"* y *"value"* de todos los mecanismos de atención tienen una dimensionalidad interna de $d_{kv} = 64$ y todos los mecanismos de atención tienen 12 cabezas de atención. Todas las otras sub capas y *embeddings* tienen una dimensionalidad de $d_{model} = 768$. El modelo fue pre-entrenado (por Google) con $2^{19} = 524,288$ pasos en C4 (*Colossal Clean Crawled Corpus*) [62], el cual es un conjunto de datos que se creó utilizando rastreo web de páginas en línea que tiene un tamaño aproximado de $\sim 7TB$, y que se presentó junto con el modelo T5. Se utilizó una longitud máxima de secuencia de 512 y un tamaño de lote de 128 secuencias. Con este tamaño de lote y el número de pasos de entrenamiento, corresponde a entrenar con $2^{35} = 34B$ de tokens, que es considerablemente menor que los $137B$ de

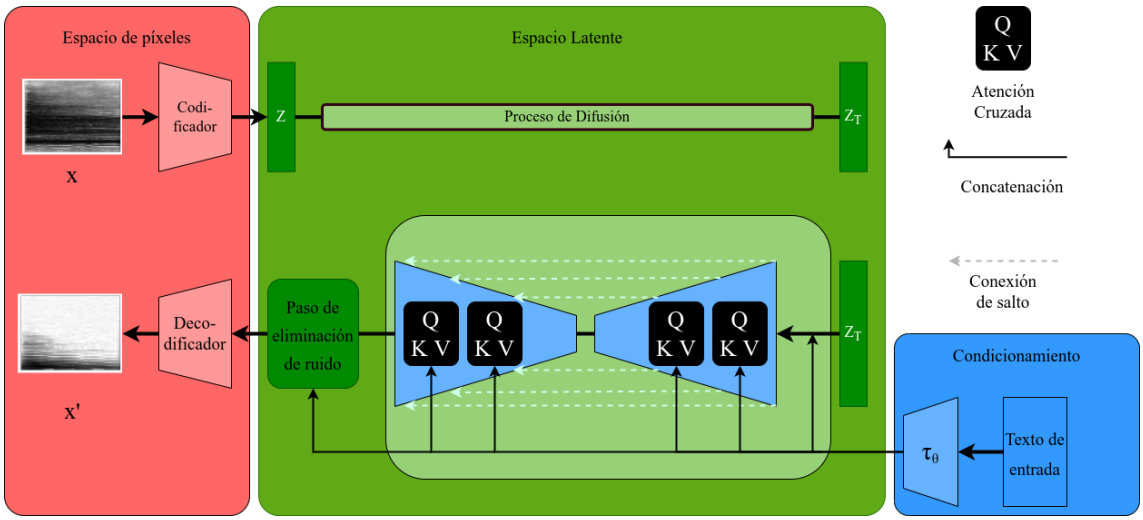


Figura 3.7: Diagrama de técnica Audio Difussion, utilizando representaciones latentes y modelos de difusión.

tokens usados en BERT. Usar s^{35} tokens resulta en un presupuesto computacional razonable, y al mismo tiempo, provee una cantidad suficiente de entrenamiento para un desempeño aceptable.

Sin embargo, aunque es un modelo de lenguaje natural capaz de tareas mucho más complejas, la sección que es relevante para este trabajo, es la capacidad de extraer una representación numérica del texto provisto que capture la información semántica y contextual del mismo, de forma que el texto de entrada será introducido al modelo y se obtendrá dicha representación numérica, ver figura 3.8.

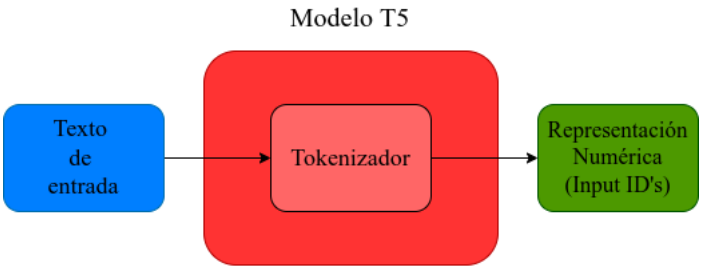


Figura 3.8: Diagrama del flujo para la obtención de una representación numérica del texto de entrada.

El modelo inicia con un proceso de tokenización, el cual se encarga de separar el texto en segmentos significativos. El tokenizador en T5 funciona a nivel de segmentos de frase (*SentencePiece*) en conjunto con *Unigram*, el cual es un algoritmo de tokenización de sub-palabras [63], en donde inicializa su vocabulario base con un grán número de símbolos y los va recortando progresivamente para obtener un vocabulario más reducido. Unigram guarda las probabilidades para cada token en el corpus de entrenamiento, además de guardar el vocabulario para que la probabilidad de cada tokenización posible pueda calcularse después del entrenamiento. Y el algoritmo sim-

plemente toma la tokenización más probable en la práctica, donde las probabilidades son definidas por la pérdida sobre la que el tokenizador fue entrenado. Asumiendo que los datos de entrenamiento consiste en palabras $x_1, ..., x_N$ y que el conjunto de todas las posibles tokenizaciones para una palabra x_i es definido como $S(x_i)$, entonces la pérdida total se define como:

$$\mathcal{L} = -\sum_{i=1}^{N} \log \left(\sum_{x \in S(x_i)} p(x) \right) \tag{3.11}$$

Una vez son obtenidos los tokens, el modelo genera los *input ID's*, estos son las representaciones numéricas de los tokens que constituyen la secuencia de entrada. T5 cuenta con distintos tamaños de modelo, en función de cuantos parámetros tiene cada uno. Debido a que no se utilizará el modelo para tareas de NLP, y solo se utilizará la etapa de tokenización, se optó por usar el modelo *T5-base*, el cual cuenta con 220 millones de parámetros y fue entrenado con el conjunto de datos C4. Dadas estás características de tokenización y entrenamiento, T5 es útil para capturar la información en frases pequeñas, como lo son las comprendidas en este trabajo, además se vuelve un modelo con un desempeño que sobrepasa a BERT, siendo además, más ligero. De ahí que se haya escogido este modelo por sobre BERT o Robustly Optimized BERT Approach [64], los cuales mantenían el estado del arte en tareas de NLP. T5 genera una representación numérica de dimensión 100, donde se entiende que para cada imagen tendrá su texto correspondiente. El tamaño de 100 fue escogido porque se considera que es un tamaño adecuado para una frase pequeña que corresponda con la descripción de un audio, además es el formato requerido para la siguiente parte de esta técnica, el modelo de difusión

3.3.2. Modelo de difusión latente (Modelo Generativo)

Debido a el gran éxito que han tenido los modelos de difusión en procesos de generación de imágenes, se optó por explorar este tipo de arquitecturas, ya que al trabajar con representaciones visuales de una fuente sonora (espectrograma), la tarea de generar imágenes de calidad se vuelve crucial para este trabajo. Los modelos de difusión [65], son modelos probabilísticos diseñados para aprender una distribución de datos $p(x)$, mediante la eliminación gradual de ruido de una variable normalmente distribuida, lo cual corresponde con el proceso inverso de una cadena fija de Markov de longitud T. Para la síntesis de imágenes, los modelos más exitosos se basan en una variante reponderada de la cota inferior variacional, que refleja las puntuaciones de eliminación de ruido [66]. Estos modelos pueden ser interpretados como secuencias de autocodificadores eliminadores de ruido igualmente ponderados $\epsilon_\theta(x_t, t); t = 1...T$, los cuales son entrenados para predecir una variante sin ruido de su entrada x_t, donde x_t es una versión ruidosa de la entrada x [58]. Lo cual se puede simplificar como:

$$L_{DM} = \mathbb{E}_{x,\epsilon\sim\mathcal{N}(0,1),t} \left[\|\epsilon - \epsilon_\theta(x_t, t)\|_2^2 \right] \tag{3.12}$$

Sin embargo, a pesar de que se ha observado que los modelos de difusión son capaces de ignorar detalles perceptualmente irrelevantes, mediante el submuestreo de los términos de pérdida correspondientes, siguen requiriendo costosas evaluaciones en el espacio de píxeles, lo que significa una mayor complejidad en términos de poder computacional y tiempo de ejecución. Por lo tanto, en [58], se propone un método que evita este inconveniente, mediante la separación explícita de la etapa de compresión y la etapa de generación. Para esto se utiliza un modelo de autocodificador, el cual aprende un espacio que es preceptivamente equivalente al espacio de la imagen, pero que ofrece una reducción significativa en términos de complejidad computacional. Dicha aproximación ofrece ventajas tales como:

1. Al dejar el espacio de imagen que tiene una mayor dimensionalidad, se obtienen modelos de difusión mucho más eficientes debido a que el muestreo se lleva a acabo en un espacio de menor dimensionalidad.

2. Se explota el sesgo inductivo de los modelos de difusión, heredados de su arquitectura (U-Net), lo cual los hace particularmente efectivos para datos con estructura espacial, y por tanto alivia la necesidad por niveles agresivos de compresión, que reducen la calidad utilizados en aproximaciones previas.

3. Se obtiene un modelo con compresión de propósito general, cuyo espacio latente puede ser utilizado para entrenar múltiples modelos generativos.

Compresión perceptual de la Imagen

Dada una imagen $x \in \mathbb{R}^{H\times W\times 3}$ en un espacio RGB, el codificador $\mathcal{E}$ codifica x en una representación latente $z = \mathcal{E}(x)$, y el decodificador $\mathcal{D}$ reconstruye la imagen a partir de la representación latente, resultando en $\hat{x} = \mathcal{D}(z) = \mathcal{D}(\mathcal{E}(x))$, donde $z \in \mathbb{R}^{h\times w\times c}$. Dado este modelo de compresión perceptual que consiste en $\mathcal{E}$ y $\mathcal{D}$, se tiene acceso a un espacio latente más eficiente y de menor dimensionalidad, donde los detalles imperceptibles de alta frecuencia se abstraen. En comparación con un espacio de píxeles con mayor dimensionalidad, este espacio es más adecuado para modelos generativos basados en la verosimilitud, de forma que adquieren la capacidad de:

- Concentrarse en los bits importantes de información semántica

- Entrenarse en un espacio de menor dimensionalidad que es mucho más eficiente en términos de complejidad computacional.

Este tipo de modelos toma ventaja de los sesgos inductivos específicos de la imagen, lo cual permite construir la U-Net principalmente a partir de capas convolucionales 2D, y centra aún más el objetivo en los bits más perceptualmente relevantes usando el límite reponderado, el cual ahora se ve como:

$$L_{LDM} := \mathbb{E}_{\mathcal{E}(x),\epsilon\sim\mathcal{N}(0,1),t}\left[\|\epsilon - \epsilon_\theta(z_t, t)\|_2^2\right] \tag{3.13}$$

La columna vertebral $\epsilon_\theta(\circ, t)$ del modelo es plasmado como una U-Net condicional en el tiempo [41]. Dado que el proceso hacia delante es fijo, z_t puede ser obtenido

eficientemente de $\mathcal{E}$ durante el entrenamiento, y las muestras de $p(z)$ pueden ser decodificadas al espacio de imágenes con un solo paso a través de $\mathcal{D}$ [58].

Mecanismos de condicionamineto

Los modelos de difusión son capaces de modelar distribuciones de la forma $p(z|y)$. Esto puee ser implementado con un autocodificador $\epsilon_\theta(z_t, t, y)$, lo cual facilita el proceso de controlar el proceso de síntesis a través de entradas y, tales como texto. Para pre-procesar y se introduce un codificador de dominio especifico τ_θ (T5), que proyecta y a una representación intermedia $\tau_\theta(y) \in \mathbb{R}^{M \times d_\tau}$, la cual es mapeada a las capas intermedias de la U-Net a través de las capas de atención cruzada implementando Atención$(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) \cdot V$, con:

$$Q = W_Q^{(i)} \cdot \varphi(z_t), K = W_K^{(i)} \cdot \tau_\theta(y), V = W_V^{(i)} \cdot \tau_\theta(y) \tag{3.14}$$

Donde, $\varphi(z_t) \in \mathbb{R}^{N \times d_\epsilon^i}$ denota una representación intermedia (aplanada) de la representación de la U-Net implementando ϵ_θ y $W_V^{(i)} \in \mathbb{R}^{d \times d_\epsilon^i}$, $W_Q^{(i)} \in \mathbb{R}^{d \times d_\tau}$ & $W_K^{(i)} \in \mathbb{R}^{d \times d_\tau}$ son matrices de proyección aprendibles, ver figura 3.7. Por lo tanto, basado en los pares texto-imagen, el LDM condicional aprende a través de:

$$L_{LDM} := \mathbb{E}_{\mathcal{E}(x), y, \epsilon \sim \mathcal{N}(0,1), t}\left[\|\epsilon - \epsilon_\theta(z_t, t, \tau_\theta(y))\|_2^2\right] \tag{3.15}$$

Para esta técnica cuando se dice modelo de difusión, se está hablando más específicamente de un modelo probabilístico de difusión para la eliminación de ruido (*Denoising Diffusion Probabilistic Models*), también conocidos como DDPM's, los cuales en palabras simples, y a groso modo, son redes neuronales que aprenden a eliminar gradualmente el ruido de los datos, empezando con ruido puro. Esencialmente, este modelo comprende dos procesos, donde el primero es un proceso fijo de difusión hacia adelante q, que gradualmente agrega ruido a la imagen, hasta que se termine con ruido puro. El segundo es un proceso aprendido de difusión en reversa p_θ, donde una red neuronal es entrenada para gradualmente eliminar el ruido de una imagen que empieza siendo ruido puro, hasta que eventualmente se termina con una imagen.

3.3.3. Algoritmo de conversión espectrograma-audio

Debido a la complejidad de la tarea de conversión para llevar un espectrograma a una forma de onda (señal de audio), existen pocos métodos que generen un resultado de buena calidad, por lo tanto, se utilizó el mismo método que en la técnica ChopAN explicado en la sección 3.2.4 de este documento.

3.4. Riffusion

Para esta tercera técnica, se abordó el problema desde una perspectiva del aprendizaje por transferencia (*transfer Learning*) [67]. El aprendizaje por transferencia es una técnica que utiliza el conocimiento de modelos entrenados previamente para

aprender sobre otro conjunto de entrenamiento. De esta manera, se pretende mejorar el aprendizaje en el dominio objetivo aprovechando el conocimiento del dominio de origen y la tarea de aprendizaje. Esta aproximación es adecuada debido a que se puede trabajar con una menor cantidad de información en la etapa de entrenamiento. Dado que la tarea esencial dentro de este modelo, es generar imágenes de calidad, se optó por Stable Diffusion [68], que al igual que la técnica anterior (Audio Latent Diffusion), es un modelo de difusión latente, lo cual significa que cuenta con un VAE para obtener una representación de menor dimensionalidad de la señal de entrada, pasa por una etapa generativa (modelo de difusión) y se vuelve a obtener una imagen con el decodificador del VAE, para una explicación más detallada consultar la sección 3.3.2 de este documento. Siendo más específicos, se utilizó Riffusion [69], el cual es un modelo que tiene una arquitectura exacta a Stable Difusión (versión 1.4), pero fue sometido a un proceso de *fine-tuning* para generar imágenes de espectrogramas, de tal forma que se aprovechó la capacidad del modelo para generar imágenes y se enfocó a la generación de espectrogramas. Por lo tanto, esta técnica consiste en hacer un proceso de fine-tuning a Riffusion para generar las señales de audio que se buscan.

Dada la información anterior, podría uno preguntarse ¿Para que volver a hacer fine-tuning a este modelo si ya hace lo que este proyecto busca? Y pese a que Riffusion es capaz de generar exitosamente muestras de audio, hay que recordar que el propósito de este trabajo es generar muestras de audio one-shot, es decir, archivos de audio que al reproducirlos contengan información de un solo instrumento (condicionado por el texto de entrada), mientras que Riffusion genera muestras de audio que contienen información de varios instrumentos, por lo que no cumple por completo con el propósito de este proyecto. La estructura de Riffusion también esta basada en CompVis, por lo que si se quiere una explicación más detallada del proceso, se puede consultar la sección 3.3 de este documento. La sección que si es distinta respecto a la técnica de Audio Difusion y Audio Latent Difusion es el códificador de texto, el cual se menciona a continuación.

3.4.1. CLIP ViT-L/14

Para la etapa de codificación del texto de entrada, se utiliza CLIP, específicamente CLIP ViT-L/14, e, el cual es un modelo provisto por OpenAI, y funciona exactamente igual que lo explicado en la sección 3.2.1 de este documento, sin embargo, es un modelo con fine-tuning, que se entrenó con muestras de mayor resolución (336 píxeles) y por una época adicional [52], el cual de acuerdo con OpenAI, obtiene un desempeño 2.6 % mejor que los modelos comparados en sus artículo. Por lo tanto, al igual que en las técnicas vistas anteriormente, un texto es provisto como entrada y mediante este modelo se obtiene una representación numérica (embeddings de texto) que posteriormente son pasadas como condicionante en el proceso de Difusión.

3.4.2. Espectrograma a audio

El proceso de conversión del espectrograma a una señal de audio (forma de onda), es llevado a cabo mediante el algoritmo de Griffin-Lim. Sin embargo, a diferencia de

las dos técnicas anteriores, las cuales utilizaron la implemntación de Librosa, en este modelo se utiliza la implementación de pyTorch del algoritmo de Griffin-Lim [70].

3.5. Tango (*Text to Audio using iNstruction-Guided DiffusiOn*)

Al igual que la sección anterior, esta técnica se basa en el aprendizaje por transferencia, que busca aprovechar las capacidades de un modelo previamente entrenado con otro conjunto de datos, para generar muestras de audio. El modelo es llamado TANGO [71], el cual también aproxima la solución a este problema mediante el uso de difusión latente. La estructura de este modelo es muy similar (prácticamente identica) al utilizado con Riffusion (Stable Diffusion), en el sentido de que ambos usan un VAE para codificar y decodificar espectrogramas y un modelo d difusión como etapa de generación condicionada. Sin embargo, tiene dos diferencias claves, las cuales fueron las razones de el porque se utilizó este modelo, la primer diferencia es que su codificador de texto es FLAN-T5-LARGE [72], el cual es una versión mucho más grande del codificador T5, visto en ChopAN en la sección 3.2 de este documento, por lo que se denomina un gran modelo de lenguaje. Esta etiqueta denota que, evidentemente, es un gran modelo, que cuenta con 540 B de parámetros, con lo cual logra tener un mucho mejor desempeño que los modelos base de T5, aunque estando todavía por debajo de un humano experto en tareas de lenguaje. La segunda diferencia clave de Tango con Riffusion es su método de conversión espectrograma-audio, ya que utiliza Hi-Fi GAN, el cual es un vocoder para generación de voz. Esto provee una alternativa viable a al algoritmo de Griffin-Lim utilizado en las técnicas anteriores.

TANGO, como se ve en la figura 3.9, tiene tres componentes principales, un codificador de texto, un modelo de difusión latente y un VAE para la conversión de espectrograma a audio. Básicamente el codificador codifica el texto descriptivo de entrada, posteriormente, la representación es usada para construir una representación latente del audio o un audio a partir de ruido Gaussiano, utilizando difusión inversa. Una vez hecho esto, el decodificador reconstruye un espectrograma Mel a partir de la representación latente. Este espectrograma es pasado a través del vocoder para generar la muestra de audio final.

Al igual que con Riffusion, podría parecer que TANGO ya resuelve el problema que se plantea en este trabajo, sin embargo, TANGO fue entrenado con AudioCaps [73], el cual es un conjunto de datos de sonidos ambientales, por ejemplo, sonidos de automóviles, perros ladrando, etc. Por lo tanto, tampoco es capaz de generar muestras de audio one-shot, como se busca en este trabajo, y es factible aplicar aprendizaje por transferencia para aprovechar sus capacidades de generación y encauzarlas hacia el propósito que se busca.

3.5.1. Codificador de texto

Como ya se mencionó anteriormente, el modelo utilizado es FLAN-T5-LARGE (780M), el cual obtiene la codificación del texto $\tau \in \mathbb{R}^{L \times d_{texto}}$, donde L y d_{texto} son el

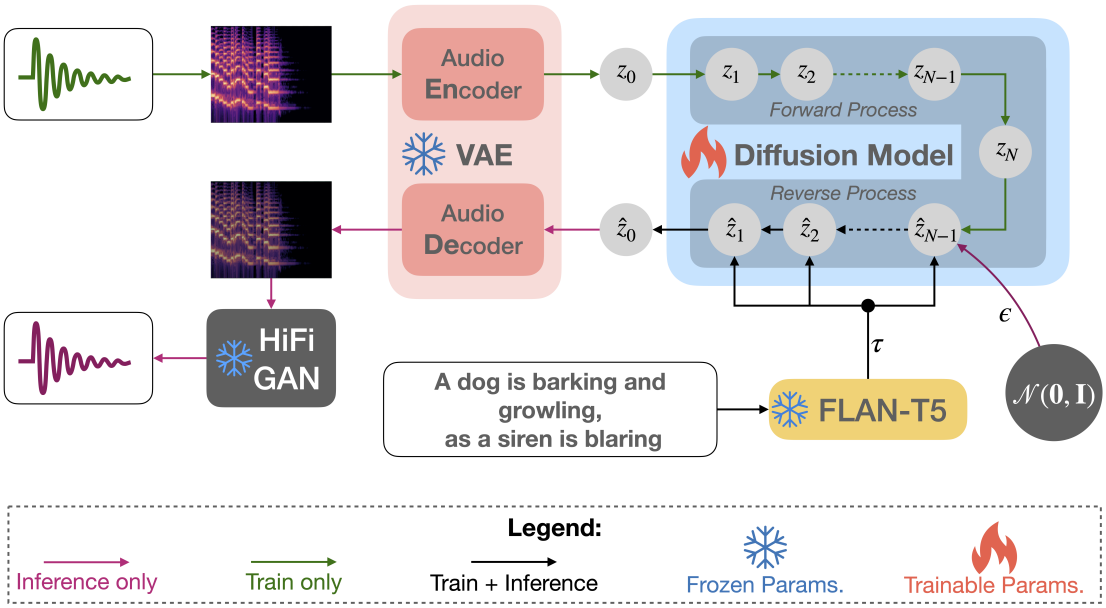


Figura 3.9: Diagrama de flujo para TANGO, imagen tomada de [71]. Las líneas verdes indican los procesos únicamente ejecutados en el proceso de entrenamiento, las líneas moradas las ejecutadas en la etapa de inferencia, las líneas negras las ejecutadas durante entrenamiento e inferencia, el copo de nieve indica los parámetros congelados, es decir, los parámetros que no se modificaran en etapa de entrenamiento y el fuego indica los parámetros que se aprenderán durante la etapa de entrenamiento.

número de tokens y el tamaño del embedding, respectivamente. Este modelo cuenta con una característica llamada *cadena de pensamientos (Chain-of-thought, CoT)* y fue pre-entrenado con un conjunto de datos basado en instrucciones, lo que según los autores, indica que es bastante bueno aprendiendo nuevas tareas mediante el contexto en la información que se le provee, mediante imitando el descenso por gradiente a través de los pesos en los mecanismos de atención. Esto es relevante debido a que modelos más antiguos como RoBERTa [64] o T5 [59] carecen de esta capacidad. Por lo tanto, los pesos del modelo fueron congelados, asumiendo que el proceso de difusión inversa de aprender a mapear la inter modalidad para la construcción de audio. Además, según los autores en TANGO, el proceso de fine tuning a FLAN-T5 puede degradar habilidad para aprender del contexto debido a los gradientes de la modalidad de audio, que están fuera de la distribución del conjunto de datos de entrenamiento.

3.5.2. Difusión latente guiada por texto

El proceso de difusión latente es una adaptación del trabajo de Liu en [74], que tiene la finalidad de construir un audio z_0 con la guía de una codificación de texto τ. Lo que se reduce a aproximar la verdad a priori $q(z_0|\tau)$ con la parametrización de $p_\theta(z_0|\tau)$. El modelo de difusión Latente se puede lograr a través de los procesos de difusión directa e inversa. La difusión directa es una cadena de Markov de una distribución

Gaussiana con parámetros de ruido programados $0 < \beta_1 < \beta_2 < ... < \beta_N < 1$para muestrear una versión más ruidosa de z_0:

$$q(z_n|z_{n-1}) = \mathcal{N}(\sqrt{1-\beta_n})z_{n-1}, \beta_n\mathbf{I} \tag{3.16}$$

$$q(z_n|z_0) = \mathcal{N}(\sqrt{\hat{\alpha}})z_0, (1-\hat{\alpha_n})\mathbf{I} \tag{3.17}$$

Donde N es el número de pasos de difusión directa, $\alpha_n = 1 - \beta_n$, y $\hat{\alpha}_n = \prod_{i=1}^{n} \alpha_n$. Y mediante la reparametrización se puede hacer un muestreo directo de cualquier z_n de z_0 a través de un proceso no Markoviano:

$$z_n = \sqrt{\hat{\alpha}}z_0 + (1-\hat{\alpha_n})\epsilon \tag{3.18}$$

Donde el término de ruido $\epsilon \sim \mathcal{N}(0, \mathbf{I})$. Y el paso final del proceso de difusión directa produce $z_N \sim \mathcal{N}(0, \mathbf{I})$. El proceso de difusión inversa remueve el ruido y reconstruye z_0 a través de una estimación guiada por texto ($\hat{\epsilon_\theta}$) usando la siguiente pérdida:

$$\mathcal{L}_{DM} = \sum_{n=1}^{N} \gamma_n \mathbb{E}_{\epsilon_n \sim \mathcal{N}(0,\mathbf{I}), z_0} \|\epsilon_n - \hat{\epsilon_\theta}^{(n)}(z_n, \tau)\|_2^2 \tag{3.19}$$

Donde z_n es muestreado de la ecuación de z_n usando ruido de una distribución normal ϵ_n, τ es la codificación del texto que se utilizará como guía, y γ_n es el peso del paso inverso n en el proceso de difusión. La estimación de ruido $\hat{\epsilon_\theta}$ es parametrizado con la U-Net mediante componentes de atención cruzada para incluir la guía de texto τ.

3.5.3. Clasificador libre de guía

Para guiar el proceso inverso de difusión y poder reconstruir el audio a priori z_0, se utiliza un clasificador libre de guía del texto de entrada τ [71]. Durante la inferencia, el parámetro de guía w,controla la contribución de la guía del texto en la estimación de ruido $\hat{\epsilon_\theta}$, con respecto a la estimación sin guía, donde un texto vacío es pasado, es decir, no hay condicionamiento:

$$\hat{\epsilon_\theta^{(n)}}(z_n, \tau) = w\epsilon_\theta^{(n)}(z_n, \tau) + (1-w)\epsilon_\theta^{(n)}(z_n) \tag{3.20}$$

3.5.4. VAE de audio y vocoder

Un autocodificador variacional de audio comprime el espectrograma Mel de una muestra de audio, $m \in \mathbb{R}^{T \times F}$, en un audio a priori $z_0 \in \mathbb{R}^{C \times T/r \times F/r}$, donde C, T, F, r, son el número de canales, número de intervalos de tiempo, número de intervalos de frecuencia y nivel de compresión, respectivamente. El modelo de difusión latente reconstruye el audio a priori $\hat{z_0}$ usando como guía el texto de entrada τ. El codificador y decodificador están compuestos de bloques ResUNet [75], que fueron pre-entrenados para maximizar de limite inferior (ELBO) [76] y minimizar la péridida adversaria [77].

Los autores de TANGO adoptaron los *checkpoints* del VAE provistos por Liu en [74]. De forma que, se usó la configuración que reportaba mejores resultados, es decir, configurando C y r como ocho y cuatro, respectivamente. Debido a la capacidad que tiene un vocoder para poder transformar el espectrograma Mel, generado por el VAE, en una muestra de audio, los autores de TANGO decidieron usar HI-FI GAN [75] basado en el trabajo de Liu [74].

Capítulo 4

Conjunto de datos

La decisión de qué conjunto de datos (*dataset*) utilizar es de particular importancia, debido a que no sólo se tiene que alinear con el propósito del proyecto, además, se tiene que verificar la calidad del mismo. Las características que fueron tomadas en cuenta al momento de escoger el conjunto de datos para este trabajo fueron:

- La calidad de las muestras de audio.
- Que fueran señales de audio *one-shot*.
- Que tuvieran más información, además de el audio mismo.
- Que existiera alguna definición textual del archivo de audio.

Dados todos estos requerimientos, no hay una gran variedad de conjuntos de datos disponibles. Por lo que se escogió trabajar con NSynth [78], el cual es un conjunto de datos que surge de la necesidad de crear un *dataset* de alta calidad, a gran escala y de forma que emule otros conjuntos como MNIST, CIFAR o ImageNet. Nsynth fue desarrollado por Jesse Engel et. al., y contiene $305,979$ notas musicales; cada una con un pitch, timbre y envolvente únicas. Se compone por sonidos provenientes de 1006 instrumentos distintos, obtenidos de librerías de muestras de audio comerciales. Los autores generaron fragmentos de audio monofónicos de cuatro segundos, con una frecuencia de muestreo de 16 KHz; dichos fragmentos de audio se denominan como notas, donde los sonidos son sostenidos durante los primeros tres segundos y después pueden o no decaer. El pitch de los instrumentos está en el rango de 21 a 108 en la escala de un piano MIDI.

Algunos instrumentos no son capaces de generar los 87 tonos diferentes del rango mencionado anteriormente (debido a la naturaleza del propio instrumento), resultando en un promedio de 65.4 tonos por instrumento. Además, cada nota contiene tres anotaciones, las cuales son piezas de información, basadas en una combinación de evaluaciones humanas y algoritmos heurísticos:

- Fuente (*Source*): El método en que se produce el sonido para la nota del instrumento. Esto puede ser acústico o electrónico, para instrumentos que fueron grabados de instrumentos acústicos o electrónicos, respectivamente, o sintéticos para instrumentos sintetizados.

- Familia (*Family*): La familia de alto nivel a la cual cada nota es miembro. Cada instrumento es miembro de una familia en específico.
- Cualidades (*Qualities*): Calidades sónicas de las notas. Cada nota tiene cero o más cualidades.

De tal forma que, basado en la información anterior, cada nota contendrá una serie de valores, donde cada característica tiene la siguiente forma:

Característica	Tipo	Descripción
note	int64	Identificador único de cada nota.
note_str	bytes	Un string identificador único en el formato <instrument_str>-<pitch>.
instrument	int64	Identificador secuencial único para la nota del instrumento de la cual fue sintetizada.
instrument_str	bytes	Una string única que sirve de identificador para el instrumento.
pitch	int64	Basado en el rango de tono MIDI en el rango [0, 127].
velocity	int64	Basado en el rango de velocidad MIDI en el rango [0, 127].
sample_rate	int64	Las muestras por segundo para la muestra de audio.
audio*	[float]	Una lista de muestras de audio respresentadas como valores en punto flotante en el rango de [-1, 1].
qualities	[int64]	Un vector binario representando que cualidades sónicas se presentan en esa nota.
qualities_str	[bytes]	Una lista de ID's de cuales cualidades están presentes en esta nota, seleccionadas de la lista de qualidades.
instrument_family	int64	El índice de la familia de instrumentos de la cual esta nota es miembro.
instrument_family_str	bytes	El ID de la familia de isntrumento a la cual este instrumento es miembro.
instrument_source	int64	El índice de la fuente sónica para este instrumento.
instrument_source_str	bytes	El ID de la fuente sónica para este instrumento.

A continuación se muestran tablas que especifican los nombres de las características y los índices usados en el conjunto de datos:

Fuentes de instrumentos:

Índice	ID
0	acoustic
1	electronic
2	synthetic

Familias de instrumentos:

Indice	ID
0	bass
1	brass
2	flute
3	guitar
4	keyboard
5	mallet
6	organ
7	reed
8	string
9	synth_lead
10	vocal

Cualidades de la nota:

Indice	ID	Descripción
0	bright	Contiene una gran cantidad de contenido de altas frecuencias y fuertes armónicos superiores.
1	dark	Una clara falta de contenido espectral, provocando un sonito muteado con tintes de bajos.
2	distortion	Forma de onda que produce un sonido crunchy, además de presencia de muchos armónicos.
3	fast_decay	Envolvente de amplitud de todos los armónicos que decae substancialmente antes del punto de'note-off' a los 3 segundos.
4	long_release	Envolvente de amplitud que decae lentamente después del punto de 'note off'. A veces aún presente al final de los 4 segundos que dura la muestra de audio.
5	multiphonic	Presencia de frecuencias de sobretono (overtone) relacionadas a más de una frecuencia fundamental.
6	nonlinear_env	Modulación del sonido con una envolvente de comportamiento distinto al del decremento monotónico de la nota.
7	percussive	Un sonido fuerte y no armónico en el onset de la nota
8	reverb	Acústica de la habitación que no fue posible remover de la muestra original
9	tempo-synced	Modulación rítmica del sonido en un tempo fijo

4.1. Obtención de par texto-audio

Como se puede observar, este conjunto de datos contiene mucha información relevante acerca de las muestras de audio que contiene. La gran ventaja, y la razón definitiva para usar este *dataset*, es que a partir de la información contenida en las

tres tablas anteriores, se puede obtener un conjunto de características textuales propias de cada audio. Siendo así que se obtuvo un texto descriptivo que contiene información acerca del tipo de instrumento, el tipo de envolvente, la información del contenido espectral, etc. En términos más específicos, de las 13 características con las que cuenta cada audio, se redujo a solamente tres. Estas tres características son *i*) *instrument_source_str*, *ii*) *qualities_string* y *iii*) *instrument_family_string*. Cada característica aporta una descripción clave de la muestra de audio. En el caso de *instrument_source_str*, nos permite saber si la fuente sonora es acústica, electrónica o sintética. En el caso de *qualities_string*, es una descripción de las cualidades sónicas de la muestra, por ejemplo, si es un sonido con distorsión, con un decaimiento rápido, con reverberación, etc. Y en el caso de *instrument_family_string*, se puede saber el tipo de instrumento del que se trata, es decir, un bajo, un teclado, una cuerda, etc. De tal forma que, para un archivo de audio de nombre *keyboard_synthetic_000-027-075.wav* se obtiene una descripción textual de la forma *synthetic distortion fast_decay nonlinear_env keyboard*. Esta misma lógica se aplicó a cada una de las muestras contenidas en el *dataset*, derivando en la obtención de un conjunto de datos que contiene señales de audio, con una descripción textual de cada una de las muestras. Sin embargo, como se vio en al capítulo de la metodología, la aproximación a la solución del problema planteado para este documento no es mediante archivos de audio, si no mediante espectrogramas. Esto hizo que fuera necesario transformar el conjunto de datos de uno audio-texto a uno imagen-texto. Y pese a que las cuatro técnicas vistas en este trabajo manejan espectrogramas muy similares, hay ligeras diferencias las cuales serán explicadas a continuación.

4.2. Preprocesamiento para ChopAN

Para esta técnica se buscó obtener espectrogramas Mel, para que CLIP pudiera buscar y discriminar entre ellos. El tipo de preprocesamiento está basado en la implementación hecha por Valerio Velardo de un VAE de audio en [56]. En dicho trabajo, se obtuvieron resultados satisfactorios al momento de reconstruir muestras de audio a partir de espectrogramas; donde la reconstrucción se llevó a cabo en una etapa posterior a haber pasado por el decodificador del VAE. El flujo de datos (*pipeline*) sigue los siguientes pasos:

- Cargar el archivo
- Realizar un relleno (*padding*) a la señal (en caso de ser necesario)
- Extraer el espectrograma logarítmico de la señal
- Normalizar el espectrograma
- Guardar el espectrograma normalizado

Con este flujo de los datos, se busca homogeneizar el tipo de señales con las que se trabaja. Después de cargar las señales, se realiza un *padding* (en caso de ser necesario),

porque aunque los autores del *dataset* comuniquen que todos los datos tienen la misma duración, es preciso asegurar que todos los datos cuenten exactamente con la misma cantidad de puntos. Posteriormente, se procede a obtener el espectrograma, utilizando la biblioteca de *librosa*, mediante la cual, el primer paso es obtienen la STFT de la señal; se obtiene su valor absoluto, y por último los valores de amplitud se pasan a una escala logarítmica. Para la obtención de la STFT, se utilizó una ventana de Hann con una longitud de 512 (*Frame Size*) y un desplazamiento (*Hop Length*) de 256, resultando en una imagen de 256×256. Después de haber obtenido el espectrograma, pasa por un proceso de normalización, el cual ha demostrado ser útil al momento de trabajar con modelos generativos [58], donde se lleva la señal a un intervalo de valores de $[0, 1]$. Finalmente, se guardan las imágenes de los espectrogramas y sus respectivos valores máximos y mínimos, para poder después, desnormalizar los valores de los espectrogramas.

4.3. Preprocesamiento para Audio Diffusion

Para esta técnica, el flujo de datos está basado en el trabajo de Rombach et. al. en compVis [58], y en el de Valerio Velardo [56], donde el *pipeline* de los datos se ve de la siguiente manera:

- Cargar el archivo
- Realizar un relleno con silencio (*padding*), en caso de ser necesario
- Obtener el espectrograma de la señal
- Normalizar los valores del espectrograma
- Guardar el espectrograma normalizado

Este proceso sigue los mismos pasos que el preprocesamiento de ChopAN, sin embargo, tiene algunas diferencias. El proceso de carga, *padding* y obtención del espectrograma, es prácticamente idéntico al utilizado con ChopAN. La diferencia, es el cambio de parámetros para el proceso de la STFT. Para esta técnica, se optó por utilizar un tamaño de ventana de 2048 (*Frame Size*) y un desplazamiento de 512 (*Hop Length*). Además, en la etapa de normalización, se utilizó un intervalo de $[0, 255]$, a diferencia de ChopAN, donde se tomó un intervalo de valores de $[0, 1]$. Después de este proceso, se generaron dos conjuntos de datos, uno con imágenes de 64×64 y otro de tamaño 256×256, esto con el propósito de poder evaluar la eficiencia del modelo con diferentes tamaños de imagen (eso se explica más adelante en este documento).

4.4. Preprocesamiento para Riffusion

Para esta técnica, los autores [69] decidieron tomar un camino similar a los dos vistos anteriormente, de manera que el *pipeline* de datos tiene la siguiente forma:

- Cargar los datos
- Aplicar *padding* en caso de ser necesario
- Obtener el espectrograma Mel de la señal de audio
- Guardar la imagen obtenida

Como se puede observar, y a diferencia de los *pipelines* de técnicas anteriores, en Riffusion no se aplica un proceso de normalización una vez que se obtiene el espectrograma de la señal de audio. Para la obtención de de la STFT de la señal, se toma una ventana de $100ms$ de duración. En referencia a la obtención del espectrograma, primero se obtiene el espectrograma Mel, mediante la implementación de *pyTorch*, donde se obtienen 512 bandas Mel, esto se ve reflejado en una imagen generada de 512×512.

4.5. Preprocesamiento para TANGO

Para esta técnica, se utiliza el mismo principio que en las técnicas anteriores. Donde el *pipeline* de los datos se ve de la siguiente forma:

- Cargar la muestra de audio
- Re muestrear la señal (en caso de ser necesario)
- Realizar un *padding* a la forma de onda
- Normalizar la forma de onda
- Obtener el espectrograma Mel de la señal

La implementación de este flujo de datos, está basado en el trabajo de Wang con Tacotron [79], esto debido a que posteriormente, se utiliza este modelo para generar audio a partir del espectrograma generado por TANGO. La utilización del Tacotron, como etapa de conversión espectrograma-audio define los requerimientos para el tipo de espectrograma. Es decir, el espectrograma generado tiene que tener dimensiones de $[64, 513]$, siendo 64 el número de bandas Mel y 513 el número de ventanas tomadas de la muestra de audio. Se utilizó la implementación de *Librosa* (en la función $librosa.filters.mel$), el cual obtiene una matriz de transformación lineal, que proyecta los bins de la FFT en bins de frecuencia Mel. Y, dado que se tenían unas dimensiones objetivo, se tomaron 1024 componentes de la FFT, que en combinación con los 64 bandas Mel, genera el espectrograma de las dimensiones deseadas.

Capítulo 5

Implementación y entrenamiento

En este capítulo se describe la implementación y entrenamiento de las instancias de cada una de las técnicas previamente descritas. Específicamente, por cada técnica, se presentan los detalles de la generación de texto-a-audio, la forma y/o técnica para codificar el audio, y los hiperparámetros de cada arquitectura.

5.1. ChopAN

Generación de texto-a-audio. Los experimentos de generación texto a audio se llevaron a cabo con el conjunto de datos NSynth [78]. El conjunto de datos, contiene $289,205$ muestras de audio en el conjunto de entrenamiento. Cada una de las muestras está emparejada con una anotación textual. El conjunto de validación, contiene $12,678$ instancias. Los clips de audio, tienen $\sim$ un segundo de duración, y fueron extraídos de librerías comerciales de muestras de audio. ChopAN fue entrenado usando solamente las instancias de pares *texto-imagen*, provenientes de NSynth. Además, se utilizó el conjunto de prueba de NSynth, como información de evaluación. El conjunto de prueba, contiene pares de la misma forma *texto-imagen*. Para la generación de muestras, se utilizaron frases que contuvieran las palabras utilizadas en el conjunto de datos, o muy similares. Las frases debían tener sentido gramatical, siguiendo la estructura de *adjetivo(s) - objeto*, siendo el objeto el instrumento objetivo. Dicha frase, se usa como mensaje de entrada, y sirve para activar el proceso de generación del modelo.

VAE de Audio. Se utilizó el modelo VAE basado en el trabajo de Valerio Velardo [56]. Este autocodificador, fue entrenado con el conjunto de entrenamiento de NSynth. Las muestras fueron recortadas, de muestras de cuatro segundos, a muestras de $\sim$ un segundo de duración. Todas las muestras mantuvieron su frecuencia de muestreo (*sample rate*) de 16 KHz para el proceso de entrenamiento. Se utilizó un nivel de compresión de dos con 11 canales latentes para el VAE.

Modelo, Hiperparámetros, y detalles de entrenamiento. Para el proceso de entrenamiento se congelaron los valores de CLIP, de forma que los únicos paráme-

tros entrenables eran los correspondientes al VAE. El VAE, tiene un total de 2.3 M de parámetros. Se utilizó un optimizador Adam [80], con una tasa de aprendizaje (*learning rate*) de 0.0005 y un programador de tasa de aprendizaje lineal para el proceso de entrenamiento. El modelo fue entrenado por 150 épocas en el conjunto de entrenamiento de NSynth, y los resultados que se informan, son los obtenidos con el *checkpoint* obtenido al final del entrenamiento (época 150). Se utilizó una GPU 1080 Ti para el entrenamiento de ChopAN, donde tomó alrededor de 60 horas para el entrenamiento de 150 épocas. Se utilizó un tamaño de lote de (*batch size*) de 64, sin pasos de acumulación de gradiente.

5.2. Audio Diffusion y Audio Latent Diffusion

Generación de texto-a-audio. Para esta técnica, al igual que con ChopAN, los experimentos se llevaron con NSynth, con las mismas particiones del conjunto de datos que se describe en la sección 5.1. Sin embargo, en este caso, la duración de los audios es de $\sim$ cuatro segundos. Tanto Audio Diffusion como Audio Latent Diffusion fueron entrenados con instancias *texto-imagen*, provenientes de NSynth. Y a diferencia de ChopAN, aquí se generaron dos versiones del mismo conjunto de datos, ambas con las mismas muestras y la misma información: 1) una versión contenía imágenes de 64×64; y 2) la otra versión contenía imágenes de 256×256. La primera versión estaba destinada a Audio Diffusion y la segunda a Audio Latent Diffusion, con la finalidad de observar y evaluar las diferencias de tener, o no, una *etapa latente* dentro del modelo. Para la generación de muestras, se utilizaron frases que tuvieran sentido gramatical, con la estructura *adjetivo(s) - objeto*, siendo el objeto el instrumento que se desea generar. Dicha frase se utiliza como iniciadora del proceso de generación, al igual que en ChopAN.

VAE de audio. Se utilizó el modelo de VAE de audio contenido en el trabajo de Robert Dargavel en [81], el cual a su vez está basado en *Autoencoder KL* de Kingma y Welling [76]. Este VAE fue preentrenado con $200,000$ muestras de espectrogramas mel, cada uno con cinco segundos de duración, obtenidas de muestras de música de *Spotify*. Todos los clips de audio fueron remuestreados a una frecuencia de muestreo de $22,050$ para el preentrenamiento del VAE. El autor utilizó un nivel de compresión de dos y un canal latente.

Modelo, hiperparámetros, y detalles de entrenamiento. Para el entrenamiento del modelo, se congelaron los parámetros del tokenizador **T5-BASE**, de forma que solo se actualicen los parámetros del modelo de difusión. El modelo de Difusión está basado en la arquitectura U-Net, más específicamente, en el modelo *UNet2DConditionalModel* de *HuggingFace* y un programador de ruido DDPM (*Denoising Diffusion Probabilistic Models*) con 100 pasos de ruido. Se utilizaron ocho canales y una dimensión de atención cruzada de 100 en el modelo U-Net. Se utilizó un optimizador AdamW [82] con una tasa de aprendizaje de $1e-4$. Se entrenó al modelo por 100 épocas en el conjunto de datos de NSynth. Se uso una GPU NVIDIA

4090 para el entrenamiento de los modelos, donde tomó un total de ocho horas para entrenar *Audio Diffusion* con el conjunto de imágenes de 64×64 y 13 horas para *Audio Latent Diffusion* con el conjunto de imágenes de 256×256. El tamaño de lote para *Audio Diffusion* fue de ocho con ocho pasos de acumulación de gradiente, lo que deriva en un tamaño efectivo de lote de 8 (instancias) $*$ 8 (acumulación) $= 64$. Por otro lado, el tamaño de lote para *Audio Latent Diffusion* fue de 4 con 12 pasos de acumulación de gradiente, lo que deriva en un tamaño efectivo de lote de 4 (instancias) $*$ 12 (acumulación) $= 48$.

5.3. Riffusion

Generación de texto-a-audio. Los datos con los que fue entrenado *Stable Diffusion* para generar espectrogramas (*Riffusion*) no es especificado por lo autores. Para el proceso de *fine-tuning* realizado durante el desarrollo de este trabajo, los datos utilizados fueron obtenidos del conjunto de prueba del conjunto de datos NSynth, que contiene $4,096$ instancias de pares *texto-audio*. Los clips de audio tienen una duración de cuatro segundos.

VAE de audio. El VAE de audio utilizado es el que se implementa en CompVis [58]. Todos los clips de audio fueron remuestreados a 16 KHz para el proceso de *fine-tuning*. Los autores configuraron la arquitectura para que tuviera un nivel de compresión de 8 y un canal latente.

Modelo, hiperparámetros, y detalles de entrenamiento. Para el proceso de *fine-tuning*, se congelaron los parámetros de el tokenizador de CLIP y solo se entrenaron los parámetros del modelo de difusión latente. Dicho modelo está basado en la arquitectura U-Net de CompVis y tiene un total de 1.45B de parámetros. Cuenta con 320 canales y una dimensión de atención cruzada de dos en la modelo de la U-Net. Se utilizó un optimizador AdamW8bit con una tasa de aprendizaje de $1e-4$ con un programador de tasa de aprendizaje lineal para el entrenamiento, y técnicas de precisión mixta (*mixed precision*). Se entrenó por 30 épocas. Se utilizó una GPU NVIDIA *A*100 de 40 GB de VRAM alojada en *Google Colab*, donde tomó un total de 13 horas para entrenar las 30 épocas. Se utilizó un tamaño de lote de tres con cuatro pasos de acumulación de gradiente. Lo que derivó en un tamaño de lote efectivo de 3(instancias) $*$ 4 (acumulación) $= 12$.

5.4. TANGO

Generación de texto-a-audio . TANGO fue preentrenado con pares *texto-audio* provenientes del conjunto de datos AudioCaps [73], el cual contiene $45,438$ clips de audio emparejados con descripciones textuales escritas por humanos. El conjunto de validación constó de $2,240$ instancias. Los clips de audio cuentan con 10 segundos

de duración y fueron recolectados de videos de *Youtube*. Las muestras de audio originalmente formaban parte de un conjunto de datos más grande llamado AudioSet [83], el cual se usa para tareas de clasificación. De manera consecuente con el resto de técnicas, TANGO fue reentrenado (*fine-tuning*) con instancias *texto-imagen* provenientes de NSynth. Al ser una tarea de ajuste fino, se utilizó un conjunto de datos más pequeño de NSynth como conjunto de prueba. Dicha partición cuenta con $4,096$ instancias del conjunto de datos y tiene la misma estructura detallada en las secciones anteriores de este documento.

VAE de audio y vocoder. Se utilizó el modelo del VAE de audio de Liu et al. [74]. Este VAE fue entrenado con los conjuntos de datos de AudioSet, AudioCaps, FreeSound, y BBC Sound Effect Library (SFX). Los archivos de audio más largos contenidos en BBC SFX fueron truncados a los primeros 30 segundos y luego segmentados en tres partes de 10 segundos cada una. Todos los segmentos de audio fueron remuestreados a 16 KHz para el entrenamiento del VAE. Se utilizó un nivel de compresión de 4 con 8 canales latentes, como indica Ghosal et al. en [71]. Los autores también utilizaron el vocoder de Liu et al. [74] para la generación de formas de onda a partir de espectrogramas mel generados por el decodificador del VAE. El vocoder es una red Hi-Fi GAN [75] entrenada en el conjunto de datos AudioSet. Todos los clips de audio fueron remuestreados a 16 KHz para entrenar la red del vocoder.

Modelo, hiperparámetros, y detalles de entrenamiento. Para el entrenamiento hecho por los autores de TANGO, se congelaron los parámetros de **FLAN-T5-LARGE** (codificador de texto), por lo que solo se entrenaron los parámetros del modelo de difusión latente. El modelo de difusión está basado en la arquitectura U-Net usada en *Stable Diffusion* [58] y tiene un total de 866M de parámetros. Se utilizaron 8 canales y una dimensión de atención cruzada de 1024 en el modelo U-Net. Se utilizó un optimizador AdamW [82], con una tasa de aprendizaje de $3e-5$ y un programador de tasa de aprendizaje lineal para el entrenamiento. Los autores entrenaron el modelo por 40 épocas en el conjunto de datos AudioCaps. Los resultados que se reportan son los obtenidos con el *checkpoint* con la mejor pérdida de validación, que se obtuvieron en la época 39. Los autores usaron cuatro GPU's NVIDIA $A600$ para entrenar a TANGO, lo cual tomó un total de 52 horas para entrenar las 40 épocas, con validación al final de cada época. Se utilizó un tamaño de lote de tres por GPU (2 instancias reales + 1 aumentada), con cuatro pasos de acumulación de gradiente (*gradient accumulation steps*). De tal forma que, el tamaño de lote real para el entrenamiento, está dado como 3 (instancias) $*$ 4 (acumulación) * 4 (GPU) = 48, como lo indica Ghosal et al. en [71]. Para el proceso de *fine-tuning*, se cuenta con los mismos 866M de parámetros, ya que no se modificó la estructura del modelo. Se utilizó el optimizador AdamW y una tasa de aprendizaje de $3e-8$, con un programador de tasa de aprendizaje lineal para el entrenamiento. El modelo se entrenó por 40 épocas sobre el conjunto de $4,096$ instancias obtenidas de NSynth. Los resultados reportados son de el *checkpoint* con mejor pérdida de validación, la cual se obtuvo en la época 39. Se uso una GPU NVIDIA $A-100$ de 40 GB alojada en *Google Colab*, para el

proceso de *fine-tuning*, lo que tomó un total de 12 horas de entrenamiento para las 40 épocas, con validación al final de cada época. Se utilizó un tamaño de lote de dos con cuatro pasos de acumulación de gradiente. De tal forma que el tamaño efectivo del lote está dado como 2 (instancias) * 4 (acumulación) = 8.

Capítulo 6

Evaluación y resultados

6.1. Métricas de evaluación

La evaluación cuantitativa de este tipo de modelos, se vuelve complicada debido a la naturaleza de los mismos, ya que al estar pensados como herramientas creativas, la medición se vuelve equiparable a la cuantificación de la creatividad de cada individuo. Sin embargo, siendo consecuentes con el trabajo realizado en TANGO [71], y en un intento de homologar este tipo de métricas, se puede evaluar la similitud de los resultados obtenidos, con muestras conocidas de un conjunto de datos congruente con la finalidad del proyecto. Dichas técnicas se basan en las distribuciones de probabilidad de las muestras obtenidas y se explican a continuación.

Métricas objetivas. En este trabajo, se utilizaron dos métricas comúnmente utilizadas: La Distancia de Audio Frechet (*Frechet Audio Distance (FAD)*), y la divergencia *Kullback - Leibler* (*KL divergence*). *FAD* [45], es una métrica perceptual que está adaptada de la *Frechet Inception Distance (FID)* para un dominio de audio. A diferencia de las métricas basadas en referencias, ésta mide la distancia entre la distribución del audio generado y la distribución del audio real, sin usar ninguna muestra de audio de referencia. Por otro lado, la divergencia *KL* es una métrica dependiente de referencia, y calcula la divergencia entre las distribuciones de las muestras de audio originales y las generadas, esto basado en las etiquetas generadas por un clasificador pre-entrenado. De tal forma que, mientras *FAD* está más relacionada con la percepción humana, la divergencia *KL* captura las similitudes entre entre las señales de audio originales y las generadas, basado en conceptos presentes en ellas. Adicionalmente del *FAD*, también se utilizó la Distancia de Frechet (*Frechet Distance (FD)*)[74] como métrica objetiva. *FD* es similar a *FAD*, pero reemplaza el clasificador **VGGish** con el clasificador **PANN**. El uso de diferentes clasificadores en *FAD* y *FD* permite evaluar el desempeño del audio generado utilizando diferentes tipos de representaciones de características. Mayor detalle de estas métricas es presentado en la sección 2.7.

Métricas subjetivas. Siguiendo el mismo proceso de evaluación hecho en los trabajos de Liu et al. [74] y Kreuk et al. [84], se seleccionaron cinco personas a las cuales

se les solicitó que evaluaran dos aspectos referentes a las muestras de audio: **calidad de audio en general (*CAG*)** y **relevancia con el texto de entrada (REL)**; ambas métricas son una calificación numérica que va de 0 a 100. A cada persona se le mostró un conjunto de 30 muestras de audio, las cuales contenían archivos generados por cada uno de los modelos, así como también archivos del conjunto de datos original (NSynth). Los evaluadores pese a no ser hablantes nativos del inglés dominaban el idioma en un nivel avanzado, contaban con conocimientos sólidos de teoría musical, interpretación en uno o múltiples instrumentos, experiencia en el ámbito de la producción musical, y fueron bien instruidos para hacer una evaluación justa de las muestras.

6.2. Resultados y análisis

			Métricas Objetivas			Métricas Subjetivas	
Modelo	Conjunto de Datos	Parámetros	FD ↓	KL ↓	FAD ↓	CAG ↑	REL ↑
Datos Originales	-	-	-	-	-	79.15	85.47
ChopAN	NSynth-TS	2.3 M	**55.99**	1.19	**7.66**	41.87	28.59
Audio Diffusion	NSynth-TS	420 M	139.10	1.64	25.88	58.43	49.29
Audio Latent Diffusion	NSynth-TS	420 M	211.05	1.36	19.09	59.38	34.84
Riffusion	LAION-AI	1.45 B	126.03	1.84	24.26	40.93	38.35
Riffusion-FT	NSynth-TeS	1.45 B	84.54	1.34	14.60	52.5	19.06
TANGO	AudioCaps	866 M	225.28	2.93	27.21	46.25	20.62
TANGO-FT	NSynt- TeS	866 M	148.26	**0.82**	13.396	**60**	**69**

Tabla 6.1: Comparación entre todos los diferentes modelos utilizados. *FT* indica si el modelo pasó por un proceso de *fine-tuning*. *NSynth-TS* y *NSynth-TeS* representan al conjunto de entrenamiento y al conjunto de prueba de NSynth respectivamente.

Resultados principales. Los resultados principales se reportan en la tabla 6.1. En dicha tabla se comparan las diferentes técnicas utilizadas, incluyendo la comparación de modelos base como TANGO [71] o Riffusion [69]. En términos de métricas objetivas, ChopAN obtuvo los mejores resultados, seguido de la versión con *fine-tuning* de TANGO. En los modelos en los que se requerían pasos de muestreo para la inferencia de una muestra de audio (todos excepto ChopAN), se tomaron 200 pasos (*sampling steps*), con la finalidad de hacer comparaciones justas. Adicionalmente, se utilizó una escala de *guía libre de clasificador* (*classifier-free guidance*) para los modelos que involucran procesos de difusión.

ChopAN obtuvo los mejores resultados en términos de distancia de Fréchet (FD y FAD), siendo entrenado con todo el conjunto de entrenamiento de NSynth (289, 205 muestras), con puntuaciones de 55.99 FD y 7.66 FAD. Estas puntuaciones son significativamente mejores que el del resto de los modelos, tomando en cuenta modelos base y modelos con *fine-tuning*. Sin embargo, hay que tomar en cuenta que es el único modelo que trabaja con muestras de audio de ~ 1 segundo, lo cual puede influir al momento de la obtención de métricas, debido a que tiene menor rango de variabili-

dad en el tiempo. Asimismo, las distribuciones de las muestras tenderán a ser más parecidas, lo que resulta en mejores puntuaciones en las métricas.

Por otro lado, en el campo de los modelos que involucran difusión, el modelo que supera a todos los demás es TANGO-FT, entrenado mediante un proceso de *fine-tuning* con el conjunto de prueba de NSynth (4096 muestras), con puntuaciones de 148.26 FD, 0.82 KL, 13.39 FAD. Esto es considerablemente mejor que su modelo base, e incluso sobrepasa a Riffusion y Riffusion-FT, los cuales son modelos en los cuales se tenía un pre-entrenamiento y un proceso de *fine-tuning* respectivamente, y que tenían la intención de sus creadores de crear música. Además, es importante mencionar que en términos de parámetros, TANGO y TANGO-FT son más pequeños que Riffusion. El desempeño de TANGO sobre las otras técnicas se le atribuye a dos razones: 1) el uso de un LLM (FLAN-T5) como codificador de texto; y 2) el proceso de obtención señal-audio. Es decir, la forma en que se convierte la señal de salida del modelo de difusión latente en una señal de audio reproducible por un humano, lo cual se logró mediante el uso de HiFi-GAN.

TANGO-FT también mostró un mejor desempeño en los resultados de las métricas subjetivas, con una puntuación de calidad de audio en general de 60 y una puntuación de relevancia de 69, indicando que es significativamente mejor en la tarea de generar sonidos *one-shot* que el resto de las técnicas empleadas en este trabajo.

Efectos de los pasos de inferencia y guía libre de clasificador. Como en cualquier aplicación de los modelos de difusión, el número de pasos de inferencia y la escala de guía libre de clasificador (*classifier-free guidance*), son cruciales para muestrear un espacio de difusión latente [66]. En la tabla 6.2 se reporta el efecto de variar el número de pasos de inferencia, y en la tabla 6.3, la variación de los parámetros en función de la escala de guía para la generación de audio. Con base en los resultados de ambas tablas, se puede notar que, en función del modelo utilizado, la calidad de las muestras obtenidas aumenta a medida que también aumenta el número de pasos de inferencia: para TANGO-FT, un número de 400 pasos de inferencia genera los mejores resultados en las métricas objetivas; de forma homóloga, 100 pasos de inferencia para Riffsuion-FT; 100 pasos de inferencia para ADM; y 100 de inferencia para ALDM. Por otro lado, en los modelos en que es posible variar el parámetro de guía libre de clasificador, una guía de 3 funciona mejor para TANGO-FT y una guía de 2.5 funciona mejor para Rifusion-FT. Cabe mencionar que las métricas objetivas, no logran capturar de manera completamente acertada el objetivo del proyecto, por lo que se tomó como una mejor referencia las métricas subjetivas, lo que derivó en la selección de 200 pasos de inferencia y una guía de 3 para las pruebas comparativas de cada modelo, reflejadas en la tabla 6.1.

Efectividad del pre-entrenamiento de modelos. Aquí se valida si el pre entrenamiento de modelos sobre otros conjuntos de datos es beneficioso, y si puede mejorar el desempeño al momento de la generación de muestras. En la tabla 6.1 se puede observar que, usando modelos pre-entrenados en otros conjuntos de datos de audio, se puede mejorar el desempeño de los mismos en términos de métricas objetivas (FD,

Modelo	Guía	Pasos	FD	KL	FAD
TANGO-FT	3	10	249.71	3.13	20.61
		20	257.16	2.12	21.46
		50	267.032	3.25	21.48
		100	263.62	3.22	21.55
		200	225.28	0.82	**13.4**
		400	**147.87**	**0.24**	14.68
Riffusion-FT	3	10	280.01	1.34	24.59
		20	246.61	**0.72**	246.61
		50	**227.79**	1.12	16.68
		100	252.45	1.18	**14.23**
		200	233.12	0.85	20.33
		400	229.55	0.88	17.5
ADM	-	10	227.34	2.55	**18.31**
		20	172.25	1.07	28.16
		50	215.57	1.96	19.39
		100	**157.61**	**0.67**	19.78
		200	221.67	1.58	25.54
		400	209.7	1.67	39.56
ALDM	-	10	**149.21**	0.79	51.88
		20	176.05	**0.61**	23.95
		50	205.3	1.53	31.36
		100	169.99	0.88	**17.31**
		200	246.25	2.84	22.95
		400	217.36	0.81	20.56

Tabla 6.2: Efecto sobre las métricas objetivas de evaluación de la variación del número de pasos de inferencia.

KL y FAD) y subjetivas (CAG y REL). Específicamente, en el caso de TANGO-FT se logró disminuir de 225.28, 2.93 y 27.21 a 148.26, 0.82 y 13.39 para valores de FD, KL y FAD respectivamente. En el caso de Riffusion-FT, se disminuyeron los valores de FD, KL y FAD de 126.03, 1.84 y 24.26 a 84.54, 1.34 y 14.60 respectivamente. Por lo tanto, es notable que cuando existe un pre-entrenamiento, y se aplican técnicas de aprendizaje por transferencia, un mejor desempeño es obtenido. Cabe mencionar que el número de datos de entrenamiento es considerablemente menor en un proceso de aprendizaje por transferencia (4096), que en un proceso de aprendizaje empezado desde cero ($\sim 280,000$).

Limitaciones. En general, tomando en cuenta todas las técnicas utilizadas en este trabajo, la generación de muestras de audio basado en un control textual sigue careciendo de precisión. Por ejemplo, en modelos como ChopAN, las generaciones de *prompts* como *acoustic reverb string* y *synthetic multiphonic nonlinear env mallet* son muy parecidas entre sí, y distan, según las métricas subjetivas, de representar lo que

Modelo	Pasos	Guía	FD	KL	FAD
TANGO-FT	200	2.5	238.4	1.69	20.09
		3	**161.46**	**0.2**	**14.78**
		5	258.13	2.76	21.03
		10	271.45	3.76	19.21
Riffusion-FT	200	2.5	**209.84**	**0.5282**	**13.21**
		3	231.67	1.1085	15.71
		5	251.03	1.29	18.86
		10	250.81	1.17	19.59

Tabla 6.3: Efecto sobre las métricas objetivas de evaluación de la variación de la escala de la guía de guía libre de clasificador (*classifier-free guidance*).

el *prompt* señala. En modelos como ADM y ALDM, donde la estructura es la misma, y solo se adhiere el elemento *latente* en ALDM, el factor de la falta de calidad en las muestras es evidente. Esto es debido a la cantidad de datos que deben generarse para obtener una señal de audio, ya que se hace un compromiso con la calidad. En relación con los modelos que se entrenaron con técnicas de aprendizaje por transferencia, pese a que mejora la calidad de las muestras, la cantidad de recursos computacionales se sitúa en un nicho de usuarios que cuentan con una tarjeta gráfica de uso comercial de alta gama. Esto representa una gran curva de adaptación que aún tiene que mejorarse para que estos modelos sean asequibles a una mayor cantidad de usuarios. Otra limitación proviene del conjunto de datos, ya que proviene de datos obtenidos a partir de librerías públicas de sonidos y algunas de las muestras no están correctamente etiquetadas o no representan acertadamente lo que la descripción del mismo indica. Esto genera un sesgo *a priori*, lo cual resulta en la generación de muestras poco congruentes con las descripciones brindadas.

Capítulo 7

Discusión y conclusiones

En este trabajo se abordó el problema de generar señales *one-shot* de instrumentos musicales condicionado por lenguaje natural, a través de diversas técnicas basadas en aprendizaje profundo. Se demostró que los modelos de difusión latente son una alternativa viable en la tarea de generación condicionada de señales. Modelos como ChopAN demostraron que al emplear espacios latentes, a través de auto-codificadores, se vuelve viable el uso de imágenes, u otro tipo de representaciones de 2 dimensiones, para tareas de procesamiento de audio. Posteriormente, con la adición de modelos de difusión sobre una base latente (ALDM, Riffusion, TANGO), se logró optimizar el proceso de generación de muestras. Se exploró el impacto de diversos parámetros en este proceso, tales como: el número de pasos de inferencia, el programador de pasos (DDPM ó DDIM), la escala de guía libre de clasificador, entre otros. Adicionalmente, se implementaron técnicas comprobadas para el condicionamiento en la generación de muestras, introducidas en los procesos de difusión mediante el uso de mecanismos de atención, como base comparativa.

En términos de la extracción de *tokens* ó *embeddings* de texto, se demostró una mejoría en modelos como TANGO sobre modelos como ALDM, esto derivado de la elección del *tokenizador* utilizado. Aunque en ambos modelos se utiliza una versión del *tokenizador* T5, el hecho de que FLAN-T5 sea de un tamaño tal que se podría categorizar como un gran modelo de lenguaje (LLM), influye en la calidad de la extracción de *tokens* a partir del texto. Esto, a su vez, se ve reflejado en un mejor condicionamiento durante la etapa de generación. Dichas aseveraciones, provocan a adentrarse más en la aplicación de grandes modelos de lenguaje para tareas de generación de señales (no solo de audio), solo teniendo como mayor desventaja la necesidad de una mayor capacidad de computo para la ejecución de estos modelos.

Otro factor crucial en la tarea de generación de audio, es la conversión espectrograma-audio. En todos los modelos, a excepción de TANGO, la técnica utilizada para esta conversión fue el algoritmo de Griffin-Lim. Sin embargo, se volvió evidente, mediante las muestras generadas y las evaluaciones subjetivas obtenidas, que este algoritmo genera ciertos artefactos en el sonido que merman la calidad del mismo y restaron puntos en las puntuaciones de las métricas subjetivas. Esto es debido a que dan la percepción auditiva de mala calidad y poca relación con el texto dado y, en relación a las métricas objetivas, introdujeron frecuencias que no estaban contenidas en la señal

de referencia, por lo que generaron distribuciones con rangos distintos y derivaron en peores puntuaciones en las evaluaciones. Este último análisis promueve la idea de sustituir los algoritmos tradicionales (Griffin-Lim) con modelos que sean capaces de hacer mejores reconstrucciones de la señal a partir de las diferentes representaciones que pueda tener la señal de audio. Estos problemas, al momento de realizar una reconstrucción, ya que la parte del espectrograma que se utiliza es solo la magnitud, perdiendo así la información de fase. Esto derivó en una pobre reconstrucción de la señal. Sin embargo, modelos como HiFi-GAN (utilizados en TANGO) han demostrado tener un mejor desempeño en tareas de reconstrucción de *speech* y audio en general.

Se ha demostrado que la generación de audio condicionado por lenguaje natural para tareas especificas, como la abordada en este trabajo, son posibles. Pero, otro factor fundamental a tomar en cuenta es el coste computacional de las técnicas que a día de hoy han sido abordadas. En este trabajo se demostró que se pueden desarrollar modelos útiles para nuestro propósito, y que pueden ser ejecutados en tarjetas gráficas de grado comercial como: NVIDIA RTX 4090 de 24 GB de VRAM para entrenamientos, y NVIDIA RTX 3060 de 12 GB de VRAM para la generación de muestras de audio. Sin embargo, dichas especificaciones siguen siendo de acceso para un nicho de personas con recursos medios a altos. Por lo tanto, este trabajo explora y exhorta a trabajar en modelos que utilicen una menor cantidad de recursos computacionales, sin que esto represente un compromiso con la calidad de las muestras generadas. Estas proposiciones van a acompañadas de nociones como la utilización de otro tipo de representaciones que puedan ser más pequeñas en términos de dimensiones de la señal, o que permitan una mejor reconstrucción de las señales de audio obtenidas. Además, este trabajo promueve el desarrollo de modelos específicos para tareas en el campo de *Music Information Retrieval* (MIR) ya que, como se demostró en este documento, la mayoría de las aproximaciones a la solución de problemas de audio son adaptaciones de trabajos en otros campos, hechas por entusiastas del área. Asimismo, también se promueve el desarrollo de modelos y técnicas que contemplen un enfoque específico a problemas de MIR, lo cual definitivamente impulsará y mejorará el avance en temas referentes al audio.

Se propone como trabajo futuro utilizar grandes modelos de lenguaje (LLM), como GPT o LLaMA, para la extracción de *tokens* de una entrada textual, que sirva como agente condicionador en la etapa de generación. Aunado a la extracción de otro tipo de representaciones del audio, como *wavelet* o MFCC, que pueden ser utilizadas al momento de tener representaciones del audio en 2 dimensiones y que sean más pequeñas en términos de tamaño de señal. Por último, se propone dejar de lado algoritmos como Griffin-Lim para las tareas de conversión espectrograma-audio, y explorar técnicas como GAN's o modelos de difusión como alternativas en esta etapa del proceso.

Bibliografía

[1] S. Slade, D. Thompson, S. Massy y S. Webber., *Music Production: What Does a Music Producer Do? – Berklee Online Take Note*. dirección: https://online.berklee.edu/takenote/music-production-what-does-a-music-producer-do/.

[2] *Sound design 101: How to make sounds | Native Instruments Blog*. dirección: https://blog.native-instruments.com/sound-design/.

[3] *Sampling: Its Role In Hip Hop & Its Legacy In Music Production*. dirección: https://abbeyroadinstitute.co.uk/blog/sampling-role-in-hip-hop-and-its-legacy-in-music-production/?cn-reloaded=1.

[4] *Royalty-Free Sounds, FX, Presets & more - Splice*. dirección: https://sounds.splice.com.

[5] *Tracklib - The record store for sampling*. dirección: https://www.tracklib.com/.

[6] J. G. Proakis y D. G. Manolakis, *Digital Signal Processing (3rd Ed.): Principles, Algorithms, and Applications*. USA: Prentice-Hall, Inc., 1996, ISBN: 0133737624.

[7] M. Müller, *Fundamentals of Music Processing*. Springer International Publishing, 2021. DOI: 10.1007/978-3-030-69808-9.

[8] A. Devices, *Definition of Sampling Rate | Analog Devices*. dirección: https://www.analog.com/en/design-center/glossary/sampling-rate.html.

[9] Mathworks, *Sample- and Frame-Based Concepts - MATLAB & Simulink - MathWorks América Latina*. dirección: https://la.mathworks.com/help/dsp/ug/sample-and-frame-based-concepts.html.

[10] G. Ellis, «Chapter 5 - The z-Domain,» en *Control System Design Guide (Fourth Edition)*, G. Ellis, ed., Fourth Edition, Boston: Butterworth-Heinemann, 2012, págs. 73-96, ISBN: 978-0-12-385920-4. DOI: https://doi.org/10.1016/B978-0-12-385920-4.00005-9. dirección: https://www.sciencedirect.com/science/article/pii/B9780123859204000059.

[11] C. Shannon, «Communication in the Presence of Noise,» *Proceedings of the IRE*, vol. 37, n.º 1, págs. 10-21, 1949. DOI: 10.1109/JRPROC.1949.232969.

[12] V. Velardo, *Types of Audio Features for Machine Learning - YouTube*, 2020. dirección: https://www.youtube.com/watch?v=ZZ9u1vUtcIA&list=PL-wATfeyAMNqIee7cH3q1bh4QJFAaeNv0&index=6.

[13] V. Velardo, *Understanding Time Domain Audio Features - YouTube*, 2020. dirección: https://www.youtube.com/watch?v=SRrQ_v-OOSg&list=PL-wATfeyAMNqIee7cH3q1bh4QJFAaeNv0&index=7.

[14] S. Tjoa, *RMS energy*, 2022. dirección: https://musicinformationretrieval.com/energy.html.

[15] *Sinusoides y sonidos compuestos: armónicos y parciales en audio | Hispasonic*. dirección: https://www.hispasonic.com/tutoriales/sinusoides-sonidos-compuestos-armonicos-parciales-audio/43309.

[16] B. B. Hubbard., *The World According to Wavelets The Story of a Mathematical Technique in the Making*. USA: Universities Press (India) Pvt. Limited, 1996, 1996, ISBN: 8173714509.

[17] C. F. Gauss, *Theoria interpolationis methodo nova tractata*, Latin y German. Göttingen, Germany: Königlichen Gesellschaft der Wissenschaften zu Göttingen, 1866, vol. 3, págs. 265-303.

[18] S. Braun, *Encyclopedia of Vibration*. USA: Elsevier, 2001.

[19] *Understanding the Mel Spectrogram | by Leland Roberts | Analytics Vidhya | Medium*. dirección: https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53.

[20] V. Velardo, *Mel Spectrograms Explained Easily - YouTube*, 2020. dirección: https://www.youtube.com/watch?v=9GHCiiDLHQ4&list=PL-%20chwATfeyAMNqIee7cH3q1bh4Q index=17.

[21] *Mel Filter Bank — PyFilterbank devN documentation*. dirección: https://siggigue.github.io/pyfilterbank/melbank.html.

[22] A. Burkov, *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019, ISBN: 9781999579517. dirección: https://books.google.com.mx/books?id=0jbxwQEACAAJ.

[23] A. Müller y S. Guido, *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media, 2016, ISBN: 9781449369897. dirección: https://books.google.com.mx/books?id=vbQlDQAAQBAJ.

[24] A. Jung, *Machine Learning: The Basics* (Machine Learning: Foundations, Methodologies, and Applications). Springer Nature Singapore, 2022, ISBN: 9789811681929. dirección: https://books.google.com.mx/books?id=Sr62zgEACAAJ.

[25] P. Billingsley, *Probability and Measure*, 3rd. New York: Wiley, 1995.

[26] G. Seif, *Understanding the 3 most common loss functions for Machine Learning Regression | by George Seif | Towards Data Science*. dirección: https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3.

[27] *What is Gradient Descent? | IBM*. dirección: https://www.ibm.com/topics/gradient-descent.

[28] Y. Nesterov, *Introductory lectures on convex optimization* (Applied Optimization). Kluwer Academic Publishers, Boston, MA, 2004, ISBN: 9789811681929.

[29] C. C. Aggarwal, *Neural Networks and Deep Learning*. Springer Cham, 2019. DOI: https://doi.org/10.1007/978-3-319-94463-0.

[30] D. E. Rumelhart, G. E. Hinton y R. J. Williams, «Learning representations by back-propagating errors,» *Nature*, vol. 323, págs. 533-536, 1986.

[31] N. Buduma y N. Locascio, *Fundamentals of Deep Learning: Designing Next-generation Machine Intelligence Algorithms*. O'Reilly Media, 2017, ISBN: 9781491925614. dirección: https://books.google.com.mx/books?id=EZFfrgEACAAJ.

[32] M. Mishra, *Convolutional Neural Networks, Explained*. Towards Data Science. dirección: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939.

[33] *Max Pooling Explained | Papers With Code*. dirección: https://paperswithcode.com/method/max-pooling.

[34] D. Foster, *Generative Deep Learning, 2nd Edition*. O'Reilly Media, Inc., ISBN: 9781098134181.

[35] V. Velardo, *Autoencoders Explained Easily - YouTube*, 2020. dirección: https://www.youtube.com/watch?v=xwrzh4e8DLs&list=PL-wATfeyAMNpEyENTc-tVH5tfLGKtSWPp&index=3.

[36] V. Velardo, *Generation with AutoEncoders: Results and Limitations - YouTube*, 2020. dirección: https://www.youtube.com/watch?v=-HqG2s4dxJ0&list=PL-wATfeyAMNpEyENTc-tVH5tfLGKtSWPp&index=8.

[37] R. Selvan, «Bayesian tracking of multiple point targets using Expectation Maximization,» Tesis doct., jul. de 2015.

[38] L. Weng, *What are Diffusion Models? | Lil'Log*, 2021. dirección: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/.

[39] N. Rogge y K. Rasul, *The Annotated Diffusion Model*, 2022. dirección: https://huggingface.co/blog/annotated-diffusion.

[40] J. Ho, A. Jain y P. Abbeel, *Denoising Diffusion Probabilistic Models*, 2020. arXiv: 2006.11239 [cs.LG].

[41] O. Ronneberger, P. Fischer y T. Brox, «U-Net: Convolutional networks for biomedical image segmentation,» en *MICCAI (3)*, ép. Lecture Notes in Computer Science, vol. 9351, Springer, 2015, págs. 234-241.

[42] *Natural Language Processing (NLP) [A Complete Guide]*. dirección: https://www.deeplearning.ai/resources/natural-language-processing/.

[43] M. Buchin y B. Kilgus, «Fréchet distance between two point sets,» *Computational Geometry*, vol. 102, pág. 101 842, 2022, ISSN: 0925-7721. DOI: https://doi.org/10.1016/j.comgeo.2021.101842. dirección: https://www.sciencedirect.com/science/article/pii/S0925772121000985.

[44] W. Zhao y R. Chellappa, «Chapter 19 - Near real-time robust face and facial-feature detection with information-based maximum discrimination,» en *Face Processing*, Burlington: Academic Press, 2006, págs. 619-646, ISBN: 978-0-12-088452-0. DOI: https://doi.org/10.1016/B978-012088452-0/50020-0. dirección: https://www.sciencedirect.com/science/article/pii/B9780120884520500200.

[45] K. Kilgour, M. Zuluaga, D. Roblek y M. Sharifi, «Fréchet audio distance: A reference-free metric for evaluating music enhancement algorithms,» en *INTERSPEECH*, 2019, págs. 2350-2354.

[46] S. Hershey, S. Chaudhuri, D. P. W. Ellis et al., «CNN architectures for large-scale audio classification,» en *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, págs. 131-135. DOI: 10.1109/ICASSP.2017.7952132.

[47] D. Dowson y B. Landau, «The Fréchet distance between multivariate normal distributions,» *Journal of Multivariate Analysis*, vol. 12, n.º 3, págs. 450-455, 1982. DOI: 10.1016/0047-259x(82)90077-x. dirección: https://doi.org/10.1016%2F0047-259x%2882%2990077-x.

[48] K. Simonyan y A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 2015. arXiv: 1409.1556 [cs.CV].

[49] S. Abu-El-Haija, N. Kothari, J. Lee et al., *YouTube-8M: A Large-Scale Video Classification Benchmark*, 2016. arXiv: 1609.08675 [cs.CV].

[50] A. van den Oord, S. Dieleman, H. Zen et al., *WaveNet: A Generative Model for Raw Audio*, 2016. arXiv: 1609.03499 [cs.SD].

[51] S. Vasquez y M. Lewis, *MelNet: A Generative Model for Audio in the Frequency Domain*, 2019. arXiv: 1906.01083 [eess.AS].

[52] A. Radford, J. W. Kim, C. Hallacy et al., *Learning Transferable Visual Models From Natural Language Supervision*, 2021. arXiv: 2103.00020 [cs.CV].

[53] *CLIP: Connecting text and images*. dirección: https://openai.com/research/clip.

[54] Y. Zhang, H. Jiang, Y. Miura, C. D. Manning y C. P. Langlotz, *Contrastive Learning of Medical Visual Representations from Paired Images and Text*, 2022. arXiv: 2010.00747 [cs.CV].

[55] A. van den Oord, Y. Li y O. Vinyals, *Representation Learning with Contrastive Predictive Coding*, 2019. arXiv: 1807.03748 [cs.LG].

[56] V. Velardo, *Generating Sound with Neural Networks - YouTube*, 2020. dirección: https://www.youtube.com/playlist?list=PL-wATfeyAMNpEyENTc-tVH5tfLGKtSWPp.

[57] D. Griffin y J. Lim, «Signal estimation from modified short-time Fourier transform,» *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, n.º 2, págs. 236-243, 1984. DOI: 10.1109/TASSP.1984.1164317.

[58] R. Rombach, A. Blattmann, D. Lorenz, P. Esser y B. Ommer, *High-Resolution Image Synthesis with Latent Diffusion Models*, 2022. arXiv: 2112.10752 [cs.CV].

[59] C. Raffel, N. Shazeer, A. Roberts et al., *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*, 2020. arXiv: 1910.10683 [cs.LG].

[60] A. Vaswani, N. Shazeer, N. Parmar et al., *Attention Is All You Need*, 2023. arXiv: 1706.03762 [cs.CL].

[61] J. Devlin, M.-W. Chang, K. Lee y K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019. arXiv: 1810.04805 [cs.CL].

[62] C. Raffel, N. Shazeer, A. Roberts et al., *c4 | TensorFlow Datasets*, 2020. eprint: 1910.10683. dirección: https://www.tensorflow.org/datasets/catalog/c4.

[63] T. Kudo, *Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates*, 2018. arXiv: 1804.10959 [cs.CL].

[64] Y. Liu, M. Ott, N. Goyal et al., *RoBERTa: A Robustly Optimized BERT Pretraining Approach*, 2019. arXiv: 1907.11692 [cs.CL].

[65] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan y S. Ganguli, «Deep unsupervised learning using nonequilibrium thermodynamics,» *CoRR*, vol. abs/1503.03585, págs. 1, 3, 4, 18, 2015.

[66] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon y B. Poole, «Score-based generative modeling through stochastic differential equations,» *CoRR*, vol. abs/2011.13456, págs. 1, 3, 4, 18, 2020.

[67] J. A. L. Marques, F. N. B. Gois, J. P. do Vale Madeiro, T. Li y S. J. Fong, «Chapter 4 - Artificial neural network-based approaches for computer-aided disease diagnosis and treatment,» en *Cognitive and Soft Computing Techniques for the Analysis of Healthcare Data*, ép. Intelligent Data-Centric Systems, A. K. Bhoi, V. H. C. de Albuquerque, P. N. Srinivasu y G. Marques, eds., Academic Press, 2022, págs. 79-99, ISBN: 978-0-323-85751-2. DOI: https://doi.org/10.1016/B978-0-323-85751-2.00008-6. dirección: https://www.sciencedirect.com/science/article/pii/B9780323857512000086.

[68] R. Rombach, A. Blattmann, D. Lorenz, P. Esser y B. Ommer, *High-Resolution Image Synthesis with Latent Diffusion Models*, 2021. arXiv: 2112.10752 [cs.CV].

[69] S. Forsgren y H. Martiros, «Riffusion - Stable diffusion for real-time music generation,» 2022. dirección: https://riffusion.com/about.

[70] *GriffinLim — Torchaudio 2.1.0.dev20230814 documentation*. dirección: https://pytorch.org/audio/main/generated/torchaudio.transforms.GriffinLim.html.

[71] D. Ghosal, N. Majumder, A. Mehrish y S. Poria, *Text-to-Audio Generation using Instruction-Tuned LLM and Latent Diffusion Model*, 2023. arXiv: 2304.13731 [eess.AS].

[72] H. W. Chung, L. Hou, S. Longpre et al., *Scaling Instruction-Finetuned Language Models*, 2022. arXiv: 2210.11416 [cs.LG].

[73] C. D. Kim, B. Kim, H. Lee y G. Kim, «AudioCaps: Generating Captions for Audios in The Wild,» en *NAACL-HLT*, 2019.

[74] H. Liu, Z. Chen, Y. Yuan et al., *AudioLDM: Text-to-Audio Generation with Latent Diffusion Models*, 2023. arXiv: 2301.12503 [cs.SD].

[75] J. Kong, J. Kim y J. Bae, *HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis*, 2020. arXiv: 2010.05646 [cs.SD].

[76] D. P. Kingma y M. Welling, «Auto-encoding variational bayes,» *CoRR*, vol. abs/1312.6114, 2013.

[77] P. Isola, J.-Y. Zhu, T. Zhou y A. A. Efros, «Image-to-image translation with conditional adversarial networks,» en *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, págs. 5967-5976.

[78] J. Engel, C. Resnick, A. Roberts et al., *Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders*, 2017. eprint: arXiv:1704.01279.

[79] Y. Wang, R. Skerry-Ryan, D. Stanton et al., *Tacotron: Towards End-to-End Speech Synthesis*, 2017. arXiv: 1703.10135 [cs.CL].

[80] *Adam — PyTorch 2.0 documentation.* dirección: https://pytorch.org/docs/stable/generated/torch.optim.Adam.html.

[81] *teticio/audio-diffusion: Apply diffusion models using the new Hugging Face diffusers package to synthesize music instead of images.* dirección: https://github.com/teticio/audio-diffusion.

[82] *AdamW — PyTorch 2.0 documentation.* dirección: https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html.

[83] J. F. Gemmeke, D. P. W. Ellis, D. Freedman et al., «Audio Set: An ontology and human-labeled dataset for audio events,» en *Proc. IEEE ICASSP 2017*, New Orleans, LA, 2017.

[84] F. Kreuk, G. Synnaeve, A. Polyak et al., *AudioGen: Textually Guided Audio Generation*, 2023. arXiv: 2209.15352 [cs.SD].

Siglas

ADC Analog to Digital Converter. 5, 6

AE Auto Encoder. 45

BERT Bidirectional Encoder Representations from Transformers. 70

CLIP Contrastive Language–Image Pre-training. 61

CLIP ViT-L/14, e CLIP Vision Transformer Large. 74

CNN Convolutional Neural Network. 39

DDPM Denoising Diffusion Probabilistic Models. 52

DFT Discrete Fourier Transform. 14

DTFT Discrete-time Fourier Transform. 12

ELBO Evidence Lower Bound. 53

FAD Fréchet Audio Distance. 57

FFT Fast Fourier Transform. 14

FLAN Fine-tuned Language Net. 75

GAN Generative Adversarial Network. 48

GD Gradient Descent. 30

GPT Generative Pre-trained Transformer. 96

ISTFT Short Time Fourier Transform. 67

KL Kullback-Liebler. 47

LDM Latent Diffusion Model. 68

LLaMA Large Language Model Meta AI. 96

LLM Large Language Model. 96

MFCC Mel Frequency Cepstral Coefficients. 96

MIDI Musical Instrument Digital Interface. 5

MIR Music Information Retrieval. 3

ML Machine Learning. 31

MSE Mean Squared Error. 42

NCE Noise Contrastive Estimation. 62

NLG Natural Language Generation. 55

NLP Natural Language Processing. 55

NLU Natural Language Understanding. 55

PCA Principal Component Analisys. 42

ReLU Rectified Linear Unit. 43

RMS Root-Mean-Square Energy. 7, 9

RMSE Root Mean Squared Error. 47

Robustly Optimized BERT Approach Robustly Optimized BERT Approach. 71

VAE Variational Auto Encoder. 45