Universidad Autónoma Metropolitana

Unidad Iztapalapa

Posgrado en Ciencias y Tecnologías de la Información

División de Ciencias Básicas e Ingeniería

Generación Automatizada de Música

Idónea Comunicación de Resultados para obtener el grado de
Maestro en Ciencias y Tecnologías de la información

Presenta:

Lic. Brayan Armando Merino Alvarado

Matrícula:

2202800371

Asesores:

Dr. Eric Alfredo Rincón García

Dr. Pedro Lara Velázquez

Sinodales:

Presidente: Dr. Pedro Lara Velázquez

Secretario: Dr. Sergio Gerardo de los Cobos Silva

Vocal: Dra. Bibiana Obregón Quintana

Ciudad de México, México, 22 de noviembre de 2022

Resumen

Crear modelos matemáticos para explicar y generar música ha sido un campo desafiante debido a la relación inherente que el humano le da a la música con la creatividad, emociones y sentimientos. En las últimas décadas en el área de la Inteligencia Artificial (IA), han surgido distintas propuestas para la clasificación y generación de música mediante técnicas de aprendizaje profundo abordando distintos enfoques en la manipulación de conceptos musicales y distintas arquitecturas de Aprendizaje Maquinal (ML: *Machine Learning*). En este trabajo es de interés la creación y evaluación de un modelo generativo para la composición de música polifónica mediante técnicas de ML. Se propone un modelo a base de una arquitectura que consta de Redes Neuronales Recurrentes (RNN: *Recurrent Neural Network*). Para generar la melodía de un primer instrumento principal, se explora y evalúa el desempeño de dos arquitecturas: a base de RNN de Memoria a Largo y Corto Plazo (LSTM: *Long Short-Term Memory*) y Unidad Recurrente Cerrada (GRU: *Gated Recurrent Unit*), para esto, se analiza el efecto de modificar algunos hiperparámetros en las RNA (*Redes Neuronales Artificiales*) utilizadas, después se emplean arquitecturas similares que constan de RNN para generar el acompañamiento. También se explora el impacto de utilizar la técnica de muestreo de temperatura en la decodificación de las RNN y se compra con un muestro por búsqueda codicioso. Los resultados muestran el reto que implica evaluar la calidad de las melodías generadas a partir de este tipo de técnicas. También se observa que, bajo ciertas condiciones, las arquitecturas propuestas muestran un rendimiento diferente.

Agradecimientos

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por apoyarme a realizar mis estudios otorgándome una beca como apoyo financiero para la realización de mi maestría.

A la Universidad Autónoma Metropolitana por darme la oportunidad de realizar mis estudios en su gran casa abierta al tiempo.

A mis asesores Dr. Eric Alfredo Rincón García y Dr. Pedro Lara Velázquez por su valioso tiempo, paciencia, consejos y guía en este proyecto.

Por último, a mi familia, por su apoyo en cada momento y por quienes nunca habrá palabras suficientes para agradecer todo.

Contenido

Lista de figuras

Lista de tablas

Introducción

La inteligencia artificial ha mostrado gran desempeño en un amplio campo de aplicaciones, desde tareas que se consideraban propias de una computadora, como pronósticos y clasificaciones, hasta tareas que se consideran propias de los seres humanos, como algunos campos del arte. Por tal motivo, surge el interés de evaluar el desempeño y las ventajas de utilizar distintas técnicas de inteligencia artificial en la creación de melodías de forma automatizada. Para ello, se fusionan dos áreas principales: la inteligencia artificial, en particular el aprendizaje maquinal (ML: *Machine Learning*) y la música. El aprendizaje maquinal cuenta con una subcategoría conocida como aprendizaje supervisado, la cual incluye modelos discriminativos y generativos. Ambos modelos se centran en el análisis y la interpretación de patrones y estructuras de datos, pero con distintos objetivos, mientras los modelos discriminativos identifican un límite de decisión para generar una clasificación, los modelos generativos son capaces de crear nuevas instancias de una clase. Llevando estos modelos al campo de la música, con modelos discriminativos se podría realizar una clasificación de melodías en distintos géneros y con los modelos generativos se podrían crear nuevas melodías a partir de los estilos aprendidos. Siendo de interés para este trabajo los modelos generativos.

En este sentido, el uso de técnicas de inteligencia artificial para la creación de contenido puede ser aún más robusto mediante el uso del Aprendizaje Profundo (DL: *Deep Learning*). Se puede definir el Aprendizaje Profundo como un conjunto de técnicas de ML basadas en Redes Neuronales Artificiales (RNA), en donde se implementan distintas capas que procesan distintos niveles de abstracción de la información [1]. En estas técnicas de ML se pueden encontrar diferentes tipos de redes neuronales, la Figura *1* intenta ilustrar algunas RNA como son: Redes Neuronales Recurrentes (RNN: *Recurrent Neural Networks*) las cuales se caracterizan por lazos de retroalimentación en el procesamiento de series temporales, este tipo de RNA se abordarán en detalle en el Capítulo 2; Redes Neuronales Convolucionales[1] (CNN: *Convolutional Neural Networks*), las cuales se especializan por el procesamiento de imágenes; Autoencoders[2], los cuales se caracterizan por comprimir los datos de entrada y reconstruirlos para tenerlos a la salida. Cada tipo de red neuronal puede mostrar mejor

[1] Este tipo de RNA no son de interés en este trabajo, pero puede encontrar una descripción detallada de estas e: Raschka, S., & Mirjalili, V. (2019). Python Machine Learning (Third edit). Packt Publishing Ltd. Pg 517.

[2] Este tipo de RNA no son de interés en este trabajo, pero puede encontrar una descripcion detallada de estas en: Skansi, S. (2018). Introduction to Deep Learning From Logical Calculus to Artificial Intelligence. Springer.

desempeño en distintas tareas dependiendo de la naturaleza de los datos de entrada y la tarea generativa o de clasificación que se quiera realizar.

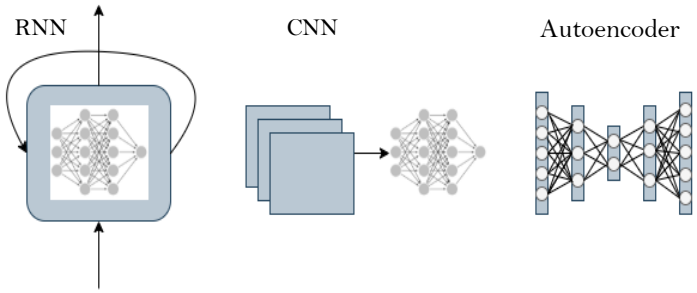


Figura 1. Algunos tipos de RNA.

Por otro lado, por milenios, la música ha sido un arte que ha acompañado a la humanidad, teniendo distintos objetivos en su creación como el entretenimiento, expresión de emociones o meros asuntos comerciales. Como se mencionó, la música es tratada como un arte, que implica el uso de creatividad, imaginación, sentimientos y emociones. Por ello, resulta una tarea desafiante usar técnicas computacionales para crear modelos que capturen el proceso de creación de melodías bien estructuradas.

Un punto de partida importante para aplicar técnicas computacionales de ML para la creación de melodías es tener una concepción de lo que es la música y qué tipo de información la conforma. Para este trabajo, una melodía es vista como una forma de arte organizada en el tiempo, en la cual se presentan distintos tonos en patrones, secuencias y grupos que ofrecen una coherencia auditiva. La Figura 2 muestra un fragmento de una melodía en formato de partitura donde se resaltan algunos elementos fundamentales que conforman la composición de una melodía.

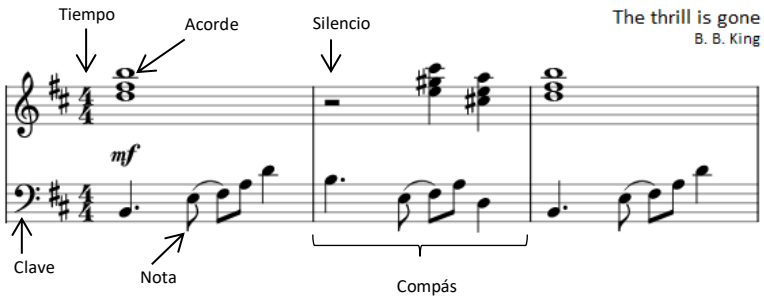


Figura 2. Fragmento de piano de la canción "*The thrill gone*" de B.B King[3].

[3] Parcialmente tomada de: <https://www.sheetmusicdirect.com/es-ES/se/ID_No/74064/Product.aspx>.

Así, con la llegada del aprendizaje profundo, han surgido distintas propuestas de generación de música que aprovechan distintas características de la estructura musical, desde la detección de patrones simples como secuencias rítmicas y repeticiones, hasta un acercamiento en lo más parecido con creación real de melodías condicionando estructuras, acordes y tonos a largo plazo.

En esta idónea comunicación de resultados se presenta un análisis en la comparación de dos modelos distintos en la generación de contenido musical a través de técnicas de aprendizaje profundo. Este documento se encuentra distribuido de la siguiente manera. En el Capítulo 1 se incluye el estado del arte de las técnicas de aprendizaje maquinal para la generación de contenido musical de forma automatizada relevantes para este trabajo. En el Capítulo 2 se presentan algunos conceptos importantes de las redes neuronales recurrentes para luego entrar a más detalle sobre la arquitectura de una LSTM y GRU. En el Capítulo 3 se abordan los modelos propuestos en este trabajo, así como detalles en el preprocesamiento y el posprocesamiento de la información del set de entrenamiento. En el Capítulo 4 se muestra los experimentos realizados, así como los resultados obtenidos. Finalmente, en el 0, se mencionan las conclusiones de este trabajo.

Objetivos

A continuación, se describe el objetivo general y particulares de este trabajo.

Objetivo general

El objetivo general de este trabajo es el siguiente

- Comparar dos algoritmos basados en redes neuronales recurrentes LSTM y GRU para la generación automatizada de música.

Objetivos particulares

Los objetivos particulares que se han definido son los siguientes:

- Estudiar los conceptos más importantes sobre redes neuronales básicas y recurrentes.
- Conseguir y preprocesar un corpus con melodías.
- Desarrollar dos algoritmos basados en enfoques diferentes para la generación de melodías.
- Calibrar los hiperparámetros de los modelos propuestos.
- Comparar el costo de la función de pérdida obtenida por ambos algoritmos.

Metodología

Para lograr los objetivos mencionados, primero se realiza la revisión del estado del arte a partir de hacer un estudio en la literatura de las diferentes propuestas relevantes de los últimos años e identificar algunas características importantes de los artículos seleccionados como la arquitectura propuesta, los conceptos musicales manipulados y la representación de la información.

Posteriormente se plantean y desarrollan las arquitecturas de los modelos propuestos y se consigue un conjunto de melodías en formato midi, a partir de este conjunto se realiza un algoritmo que permita el preprocesamiento de las melodías y se extrae del corpus un subconjunto de melodías que permita realizar experimentos rápidos, para calibrar algunos hiperparámetros de las arquitecturas propuestas. Con los modelos ya calibrados se procederá a la experimentación con el corpus completo. Finalmente, se evalúa el desempeño de los modelos generados a partir del coste de la función de pérdida.

Capítulo 1 Estado del arte

Con el avance en el aprendizaje profundo, han surgido distintas propuestas de modelos discriminativos y generativos de música, los cuales difieren principalmente en los siguientes puntos: en la arquitectura de la RNA usada, los conceptos musicales que manipulan y la representación de la información musical.

Ya que este trabajo se enfoca en la implementación y comparación del desempeño de distintas Redes Neuronales Recurrentes (RNN: *Recurrent Neural Network*) para la generación de melodías, se da una breve descripción de sus características más importantes antes de mencionar los trabajos relevantes, en el Capítulo 2 se abordan de una forma más amplia.

Las RNN se introducen como una solución en la predicción de datos secuenciales, los cuales muestran una dependencia temporal entre sí y es necesario recordar información pasada para predecir uno nuevo. Las RNN reciben datos en una secuencia ordenada, x_1, x_2, hasta x_n, y se desea que predigan el valor del dato x_{n+1}, es decir un valor futuro. Sin embargo, las RNN fueron diseñadas para hacer predicciones de los valores de x_i, $1 < i < n + 1$, usando información de los datos anteriores, x_j, $j < i$, esto significa que una RNN se entrena para predecir el valor de x_2 usando información de x_1, luego predice x_3 usando x_1 y x_2, y así sucesivamente. Para lograrlo tienen una memoria interna que almacena las características más importantes de los valores de cada secuencia. Lamentablemente, esta memoria se degrada cuando la secuencia es muy larga y las predicciones dejan de ser confiables, a este problema se le conoce como desvanecimiento del gradiente. Dos arquitecturas particulares de RNN son las celdas de Memoria a Largo y Corto Plazo (LSTM: *Long Short-Term Memory*) y las celdas de Unidad Recurrente Cerrada (GRU: *Gated Recurrent Unit*).

Las celdas LSTM se desarrollan como una propuesta a la solución de problema del desvanecimiento del gradiente presente en una RNN común y se especializan en problemas de predicción de series temporales, estas serien temporales son una secuencia de datos y cada dato representa un instante de tiempo o un paso de tiempo. Estas celdas se caracterizan por el uso de diferentes compuertas, cuyo objetivo es recordar información importante de pasos de tiempo anteriores durante un largo periodo y por olvidar la información que ya no es necesaria, por ejemplo, si un enunciado empieza hablando de las manzanas, se debe recordar que es un plural, pero si más adelante se habla de su costo, se debe desechar esa información y actualizar a la palabra en singular. Mientras que las celdas GRU son una versión

simplificada de la LSTM, no deben verse como un caso especial de LSTM. Ambos modelos tienen un rendimiento similar que depende de la tarea para la que se les entrene.

Adicionalmente, cabe destacar otro tipo de red diferente, las capas de incrustación[4], las cuales son esencialmente una técnica de clustering cuyo objetivo es agrupar elementos[5] similares y que estos grupos sean lo más heterogéneos posibles entre sí asignándole un vector de valores tipo *float* a cada elemento.

Enfocándose en modelos de ML generativos, algunos trabajos a resaltar son los que se mencionan a continuación.

En [2], se realiza uno de los primeros estudios en la generación de música mediante redes LSTM para aprender una forma de blues y comparan su enfoque con intentos anteriores. En este trabajo se utiliza un enfoque particular en la forma de representación de los datos, comparado con trabajos anteriores. Para representar acordes utilizan una unidad de entrada y una unidad de salida por nota, donde, en cada unidad, un 1 representa encendido y 0 representa apagado. Para representar la duración de cada acorde se utiliza un vector de entrada que representa una porción de tiempo mediante pasos de tiempo, por ejemplo, si la cuantización del tiempo se establece en el nivel de la octava nota, se necesitan ocho pasos de tiempo para representar una nota completa. En este artículo se presentan dos experimentos, el primero se enfoca en utilizar la red solo para aprender los acordes, para asegurarse que la red pueda aprender la estructura de acordes en ausencia de la melodía, el segundo experimento consiste en aprender la melodía completa (acordes y duración) en conjunto, así, cuando se ha concluido el entrenamiento, se introduce una secuencia de notas para generar una nueva melodía. El trabajo propuesto logra demostrar que las redes LSTM pueden aprender con éxito la estructura musical y se pueden utilizar para componer nuevas melodías.

En [3], se busca generar música pop que incluya la melodía y acompañamiento de acordes y batería, esto lo hacen al crear una arquitectura jerárquica a base de RNN donde cada nivel se encarga de crear un aspecto de las melodías. El nivel inferior consta de dos capas LSTM, la primera genera las notas y la segunda, condicionada a la nota generada, produce la duración de la nota. El siguiente nivel consta de dos capas LSTM que se encarga de generar los acordes, creando uno para cada paso de tiempo. El último nivel se encarga de generar la batería a partir de dos capas LSTM, la generación de la batería está condicionada a las notas del primer nivel. Para cada nivel, la información musical se representa mediante una codificación one-hot. Muestran que el modelo propuesto es capaz de crear melodías

[4] O Embedding en inglés

[5] Hablando en términos de series temporales, cada elemento es un paso de tiempo.

agradables al realizar estudios en humanos, las personas prefieren las melodías generadas por su propuesta que la de trabajos previos. Adicionalmente, proponen dos aplicaciones en su propuesta, la primera se basa en generar la música, un dibujo humano a trazos bailando y texto tipo karaoke para que la gente pueda cantar.

En [4], proponen BachBot, un modelo de generación de música del estilo de los corales de Bach basado en redes LSTM. Proponen un nuevo enfoque de codificación secuencial para representar la información musical polifónica. Este nuevo enfoque se basa en utilizar notas individuales en lugar de acordes para reducir significativamente el tamaño del vocabulario, así, la representación se forma al colocar secuencialmente cada una de las notas (sean de acordes o no) comenzando desde la nota superior (visto como una partitura) en cada paso de tiempo[6], cada nota está acompañada de un indicador de si está vinculada con una nota en el mismo tono del paso de tiempo anterior o no, además, en este enfoque se agrega un símbolo que indica la separación de pasos de tiempo y también un símbolo que indica si una fermata está presente. La arquitectura se basa en múltiples capas LSTM ocultas y una capa de incrustación como entrada. El trabajo presenta diversos experimentos en donde exploran distintos hiperparámetros como: la cantidad de capas ocultas, cantidad de unidades por capa, tamaño del vector de incrustación, tamaño de la secuencia de entrada y el porcentaje de las capas dropout. En este trabajo obtienen que su modelo logra reproducir música del estilo Bach, demostrado a partir de un estudio en humanos en el cual el 51% de las personas lograron diferencial a BachBot de Bach.

En [5], presentan DeepBach, un modelo destinado a modelar música polifónica, específicamente piezas tipo himnos al estilo de Bach. En este trabajo representan la información musical con seis listas secuenciales en las que cada una representa distinta información, las primeras cuatro representan listas de cuatro voces (soprano, alto, tenor y bajo) y las últimas dos, representan metadatos, una de ellas hace el trabajo de metrónomo y la otra indican el uso de fermatas. Cada nota en las listas de voz se muestra con su representación midi, además de que se usa un símbolo para indicar la prolongación de duración de nota, la secuencia de metrónomo se representa mediante números enteros del 1 al 4, mientras que el uso de fermatas se indica al colocar un 1 si está presente y de lo contrario un 0. En cuanto a la arquitectura propuesta, está formada por tres entradas y una salida. Dos entradas cuentan con una capa de incrustación seguida de una RNN profunda, una de estas entradas se encarga de procesar información pasada y la otra información futura, la tercera entrada es una RNA simple que tiene como entrada información intermedia. La salida de estas tres entradas es introducida a una RNA simple, donde finalmente se tiene como salida

[6] Se entiende como paso de tiempo (o inglés: *frame*) a cada nota individual, acorde o silencio presente en una partitura.

la información intermedia de la secuencia introducida en una de las capas de entrada. Sus resultados muestran que su modelo es capaz de generar corales al estilo de Bach al realizar una prueba discriminatoria en humanos, donde el 50% votaron que los corales generados fueron hechos por Bach y no por una computadora.

En [6], se presenta un modelo de nombre DAC (*Deep Artificial Composer*) orientado a ser aplicado a diversos estilos musicales para generar melodías monofónicas, pero en este trabajo se enfocan en melodías folk irlandesas y klezmer, además este modelo se basa en RNN. Para representar la información musical utilizan dos secuencias de datos, una representa el tono de las notas y otra la duración de cada nota, a partir de estas, se genera su propio vocabulario para cada secuencia y finalmente se convierten en vectores one-hot. En cuanto a la arquitectura, se tienen dos entradas las cuales son mapeos uno a uno de las unidades de entrada binarias, la entrada a la red de duración es una concatenación de los vectores de duración y tono, mientras que la entrada a la red de tono es una concatenación del tono actual y los próximos vectores de duración y, para cada red (duración y tono), se emplean RNN profundas a base de celdas GRU. Los resultados de este trabajo obtienen que su modelo logra crear melodías que presentan la misma estructura rítmica y estilo que el material empleado para el entrenamiento, esto lo miden mediante una propuesta de evaluación de novedad de las melodías generadas, que se basa en calcular qué fracción de las transiciones de la canción no se encuentran en el corpus musical usado en entrenamiento.

En [7], presentan un modelo llamado XiaoIce Band que genera melodías junto con pistas de acompañamiento interpretadas por distintos instrumentos. Las canciones son generadas en dos fases, la primera se encarga de generar la melodía principal condicionada a una progresión de acordes mediante un modelo al cual nombran Modelo de Generación Cruzada de Melodía y Ritmo basado en Acordes (CRMCG: *Chord based Rhythm and Melody Cross-Generation Model*) y en una segunda fase se generan las pistas de acompañamiento mediante el modelo (MICA: *Multi-Instrument Co-Arrangement Model*). La arquitectura de ambos modelos se basa en autoencoders formados por RNN, en particular celdas GRU, donde para CRMCG se tiene como entrada una progresión de acordes y la red se encarga de generar el ritmo y la melodía y, para MICA se introduce la melodía principal generada y se produce cada uno de los demás instrumentos. Comparan su modelo con otras propuestas mediante una evaluación humana y obtienen que su modelo presenta un mejor rendimiento que otros modelos.

En [8], proponen el modelo MisicVAE que genera melodías monofónicas y tiene como objetivo abordar el problema de los VAE recurrentes de modelar secuencias a largo plazo.

Este problema lo abordan al usar en su arquitectura VAE[7] recurrentes a base de celdas LSTM y, en particular, con el uso de un decodificador jerárquico, que primero genera incrustaciones para las subsecuencias de la entrada y luego usa estas incrustaciones para generar cada subsecuencia de forma independiente, donde las distribuciones de segmentos de melodía más grandes, como los compases, los modelan explícitamente. Para evaluar su modelo, realizan algunos experimentos cuantitativos donde determinan el rendimiento el muestreo, la interpolación y la reconstrucción, así como experimentos cualitativos que se basan en realizar pruebas con humanos y determinar lo agradable de las melodías.

En [9], abordan el tema de generación de música clásica monofónica al estilo de Bach a partir de explorar distintas arquitecturas a base de RNN. La información musical que se enfocan en manipular son las notas, los acordes y los silencios, a partir de esto se generan un diccionario del vocabulario presente en todo el corpus y crean las muestras de secuencias de entrada y de salida. La arquitectura propuesta se basa en una capa de incrustación como entrada seguida de múltiples capas ocultas a base de RNN y a la salida una capa de "*TimeDistributed"*, en esta propuesta exploran el rendimiento de la arquitectura variando diversos hiperparámetros, así como la cantidad y tipo de RNN en las capas ocultas. En las capas ocultas se explora el uso de celdas GRU y LSTM, obteniendo que los GRU superan en rendimiento a los LSTM. Adicionalmente, en este trabajo se muestra una forma empírica de estimar la longitud de entrada o el número de pasos de tiempo de entrada a la red.

En [10], se presenta un modelo llamado PopMNet que se enfoca en la generación de música pop. En este trabajo se busca generar las melodías de forma jerárquica o en dos etapas, primero se enfoca en establecer características de alto nivel como estructuras a base de repeticiones y secuencias entre los componentes de la melodía y a partir de estas estructuras como información condicional se genera la melodía final. Para generar las estructuras de repeticiones y secuencias se emplea una arquitectura que está formada por una Rede Generativa Antagónica (GAN: *Generative Adversarial Networks*)[8] convolucional que se enfoca en generar las matrices de adyacencia del grafico de estructuras melódicas, a su vez, esta GAN está formada de un crítico D que pretende distinguir entre gráficos de estructuras generados y reales y de un generador G que intenta generar gráficos de estructuras de las melodías que puedan engañar a D. La segunda etapa, encargada de generar las melodías, está formada por tres módulos: un extractor de funciones de progresión de acordes a base de GRU, un extractor de funciones de melodía fuente a base de GRU y un generador de melodías a

[7] Para más detalles del funcionamiento y arquitectura los VAE puede consultar Raschka, S., & Mirjalili, V. (2019*). Python Machine Learning*, Third edit. Packt Publishing Ltd. pp. 620.

[8] Estas no son de interés para analizar en este trabajo**,** para más detalles el funcionamiento y arquitectura de las GAN puede consultar Raschka, S., & Mirjalili, V. (2019*). Python Machine Learning*, Third edit. Packt Publishing Ltd. pp. 619.

base de GRU. Para analizar sus resultados realizan dos principales evaluaciones, primero, analizan si las estructuras generadas comparten propiedades similares a las estructuras de las melodías reales a partir de graficar la distribución de distancias entre compases y, segundo, realizan un experimento en humanos para determinar qué tan agradable son comparadas con las generadas con otros modelos. Con sus experimentos concluyen que su modelo es capaz de generar melodías de mayor calidad que los generados por algunos modelos existentes.

En [11], se presentan el modelo Musenet, su objetivo es generar música novedosa y no repetitiva. Su arquitectura se basa en dos módulos principales: uno es el discriminador que, en una primera fase, se encarga de generar el primer acorde de un compás son condicionado a las notas del compás que vinieron anteriormente, y el segundo modulo es el generador, que genera las notas del compás condicionado al acorde base generado por el discriminador. El discriminador está formado por celdas LSTM de cinco capas ocultas, mientras que el generador se forma por LSTM y GPT-2[9]. En este trabajo evalúan sus resultados a partir de los valores de pérdida.

En [12], presentan Jukebox, el cual es un modelo que genera canciones con canto en el dominio del audio digital. Establecen diferentes estrategias de generación de las canciones: reinterpretaciones, que genera canciones a partir de la combinación de artistas y letras que ya existen, terminaciones, que completa canciones a partir de dar como semilla 12 segundos de canciones existentes, letra aleatoria, que genera música a partir de proporcionarle la letra y estilo libre, donde se le proporciona en género y el artista y genera una canción. La arquitectura del modelo se basa en VQ-VAE[10] que se encarga de comprimir el audio y descomprimir, junto con una función de pérdida diseñada para tener la máxima cantidad de información musical, para aprender a reconstruir las melodías en el dominio del audio sin en bruto sin procesar, a su vez, se utilizan transformers dispersos autorregresivos entrenados con los espacios comprimidos para recrear la información pérdida en cada nivel de compresión. Demuestran que su modelo es capaz de generar música de diversos géneros y estilos musicales de hasta varios minutos de coherencia

[9] Estas no son de interés para analizar en este trabajo, para más detalles sobre GPT-2 puede consultar el trabajo: Alec Radford, Jeffrey Wu, Dario Amodei, Daniela Amodei, Jack Clark, Miles Brundage, and Ilya Sutskever. *Better language models and their implications*. 2019.

[10] Estas no son de interés para analizar en este trabajo, para más detalles sobre los VQ-VAE puede consultar el trabajo: Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. *Neural discrete representation learning*, 2017.

Capítulo 2 Redes neuronales recurrentes

En este trabajo son de interés las Redes Neuronales Recurrentes para el desarrollo de las arquitecturas de los modelos propuestos, por tal motivo se da una descripción detallada de estas. En el Apéndice B se encuentra la estructura básica de una red perceptrón multicapa, o *feed-forward*, desde los elementos que forman una neurona artificial, hasta el algoritmo de *backpropagation*.

De igual forma, en esta sección se abordan algunas de las arquitecturas de RNN de interés. Cabe destacar que se pueden encontrar más arquitecturas disponibles en la literatura, e incluso puede haber variantes de las mismas. Esta sección está ampliamente basada en [13, 15, 16].

2.1. Datos secuenciales

Antes de hablar a detalle sobre las características de una arquitectura de una RNN, se menciona cierto tipo de información de interés en el uso de estas, los datos secuenciales o secuencias de datos. Como su nombre lo dice, estos datos se encuentran en forma de secuencias, donde cada elemento representa un instante de tiempo o paso de tiempo. Este tipo de datos cuentan con algunas características como: se presentan en un orden determinado, no son independientes entre sí y las muestras de entrenamiento no son mutuamente independientes. Comparados con otros tipos de información, en los cuales los datos se asumen independientes e idénticamente distribuidos y por lo que el orden en el que se dan las muestras de entrenamiento a la red es irrelevante, en los datos secuenciales el orden importa. Como ejemplo de datos secuenciales se pueden mencionar distintos tipos de información como el precio de las acciones en la bolsa de valores, una palabra dentro de una oración, una nota musical contenida en un conjunto de compases, e incluso, secuencias de aminoácidos dentro de proteínas. Se debe observar que en estos ejemplos cada dato depende en gran medida de todos los datos anteriores.

Una manera de representar el modelado de datos secuenciales como una relación de entrada y salida dentro de una red cuyo propósito es la predicción del siguiente paso de tiempo se muestra en la Figura 3, del lado izquierdo está la red comprimida y del lado derecho está la misma desglosada en el tiempo. Aquí, x_t y y_t representan los datos en el paso de tiempo t, así, tanto x_t como y_t son secuencias y el objetivo de la red es predecir el valor de

y_t a partir de x_t, cabe destacar que $y_t = x_{t+1}$ dado que es el siguiente valor en la secuencia. Por ejemplo, la oración "una red neuronal recurrente es", cada palabra puede representar una unidad de tiempo, entonces $x_1 =$ "una", x_2"red", $x_3 =$ "neuronal", $x_4 =$ "recurrente" y $x_5 =$ "es". Por lo tanto, $y_1 =$ "red", $y_2 =$ "neuronal", $y_3 =$ "recurrente " y $y_2 =$ "es".

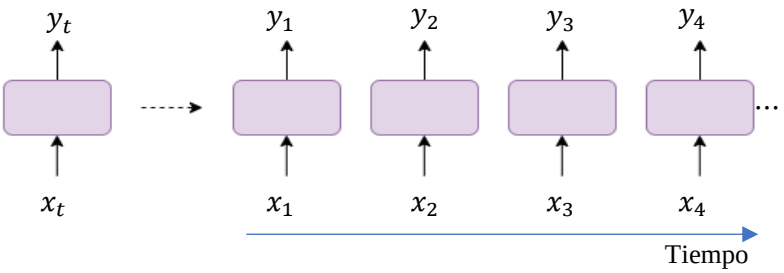


Figura 3. Representación de datos de entrada y salida secuenciales.

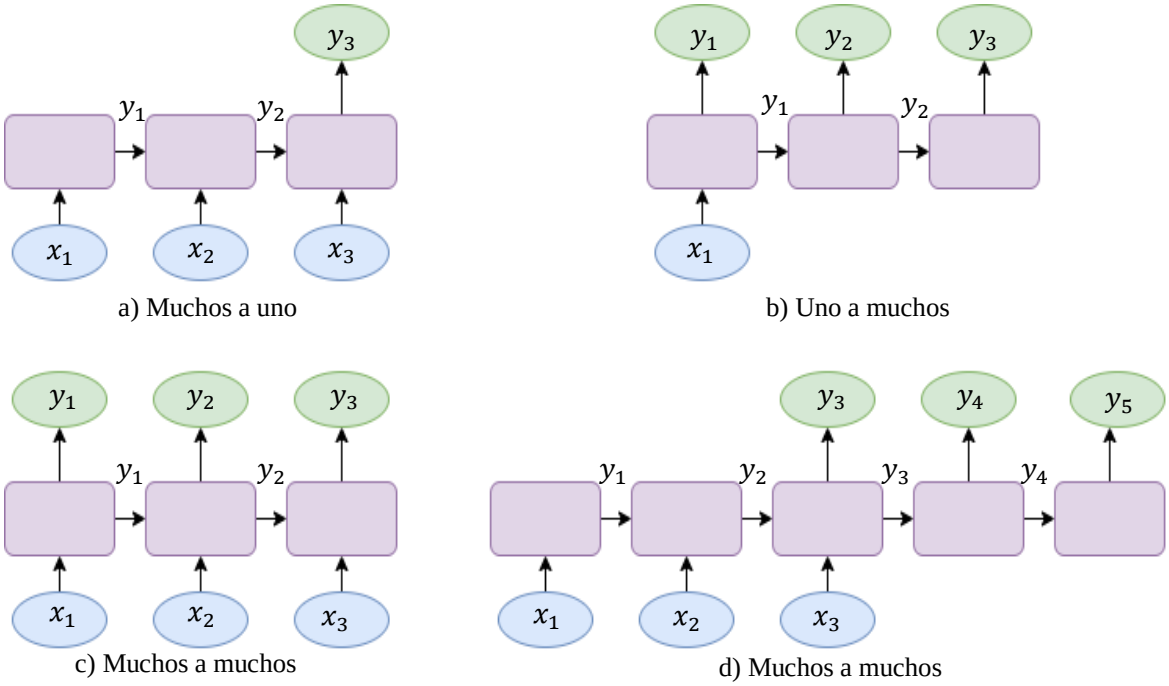


Figura 4. Relación de datos secuenciales de entrada y salida.

La tarea de modelado de secuencias se puede clasificar de distintas maneras dependiendo de la relación de datos de entrada y de salida. La Figura 4 muestra algunas formas comunes de relación de entrada y salida de datos secuenciales desglosadas en pasos de tiempo. Cada una de las configuraciones muestra un nuevo elemento, las flechas que indican que la salida

de un paso de tiempo también se introduce como entrada en el siguiente paso de tiempo, a esto se le conoce como ciclos de retroalimentación que son la principal característica de una RNN, pero se abordará con más detalle en la siguiente sección.

En la Figura 4 se muestran las siguientes relaciones:

- Muchos a uno. Los datos de entrada son una secuencia y la salida es un vector de tamaño fijo o escalar. Como ejemplo se puede encontrar modelos discriminativos como el análisis de sentimientos que muestra la Figura 5, en la cual varias palabras de entrada generan una única salida, en este ejemplo se introducen tres palabras y la red predice el sentimiento que reflejan.

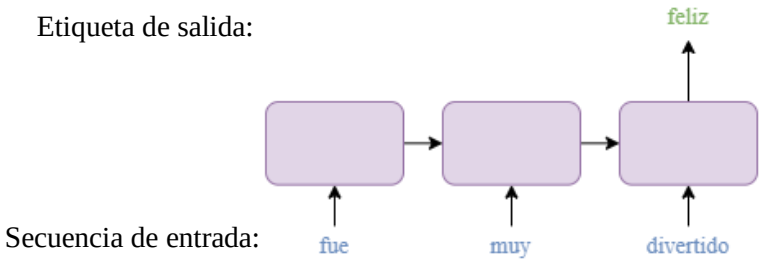


Figura 5. Ejemplo de relación de entrada-salida muchos a uno.

- Uno a muchos. Los datos de entrada son en formato estándar y la salida es una secuencia. Como ejemplo se tiene modelos cuya entrada es una imagen y su salida es la descripción de la imagen con palabras como muestra la Figura 6 donde se tiene una imagen a la entrada y a la salida una secuencia de palabras que la describen.

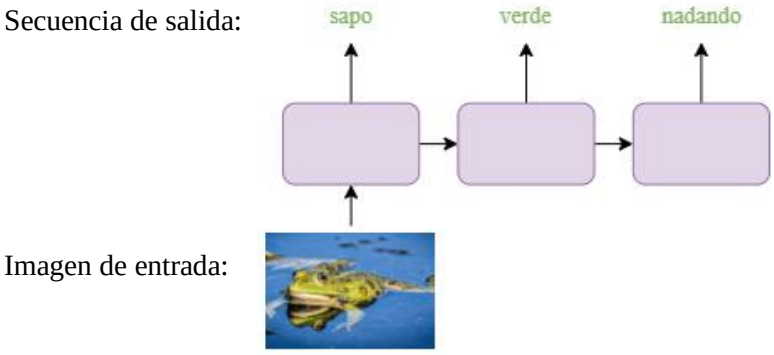


Figura 6. Ejemplo de relación de entrada-salida uno a muchos[11].

[11] Imagen parcialmente tomada de pixabay.com

- Muchos a muchos. Tanto la entrada como la salida son secuencias e incluso pueden dividirse en si están sincronizadas o no como muestran la Figura 4c) y la Figura 4d). Como ejemplo se tiene la traducción de textos entre dos idiomas, la Figura 7 ilustra un ejemplo donde tanto la entrada como la salida son dos secuencias de palabras y la salida es la traducción al inglés de la frase en español en la entrada.

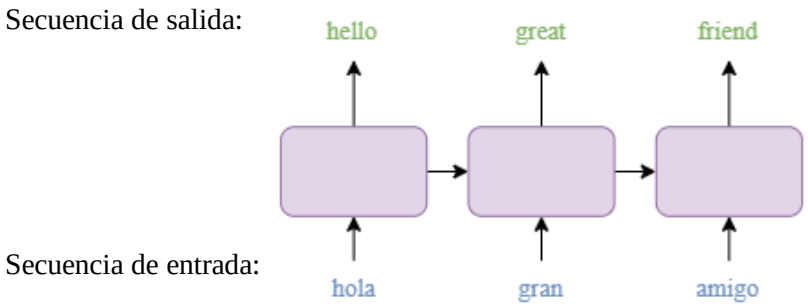


Figura 7. Ejemplo de relación de entrada-salida muchos a muchos.

2.2. Estructura de una RNN

Las RNN se introducen como una gran solución en la predicción de datos secuenciales y fue desarrollada por primera vez en el año 1982 por J. J. Hopfield y nombrada Redes Hopfield [15], mostrando un gran desempeño al tratar de modelar la dependencia de estos datos entre sí. Las RNN están diseñadas para modelar secuencias en las que es necesario recordar información pasada.

Una RNN es parecida a una red feed-forward, excepto que también tiene conexiones de los datos de salida que apuntan hacia atrás. La RNN más simple es la de una sola neurona. Esta, además de recibir los datos de entrada, también recibe como entrada su salida de un paso de tiempo anterior. La Figura 8 muestra la representación de una neurona recurrente desglosada en el tiempo, donde x_t es la entrada en el tiempo t, y_t es la salida en el tiempo t y h_t representa el estado oculto del tiempo t.

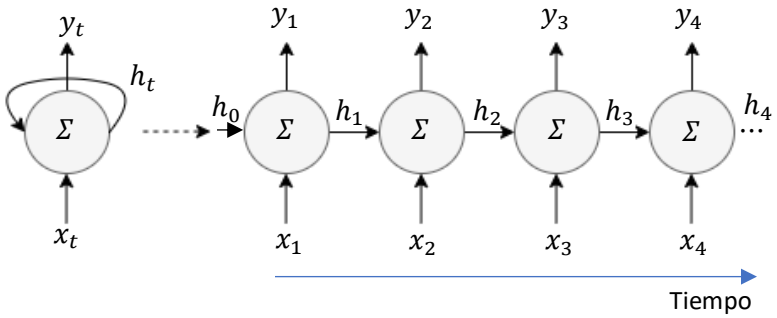


Figura 8. Una neurona recurrente.

A partir de una neurona recurrente, se puede crear una capa de neuronas recurrentes, en la Figura 9 se ilustra una capa recurrente, en cada paso de tiempo las neuronas reciben como entrada el vector x_t y la salida h_{t-1} en el tiempo t. Cabe resaltar que, para una capa de neuronas recurrentes, tanto las entradas como las salidas, ahora representan vectores.

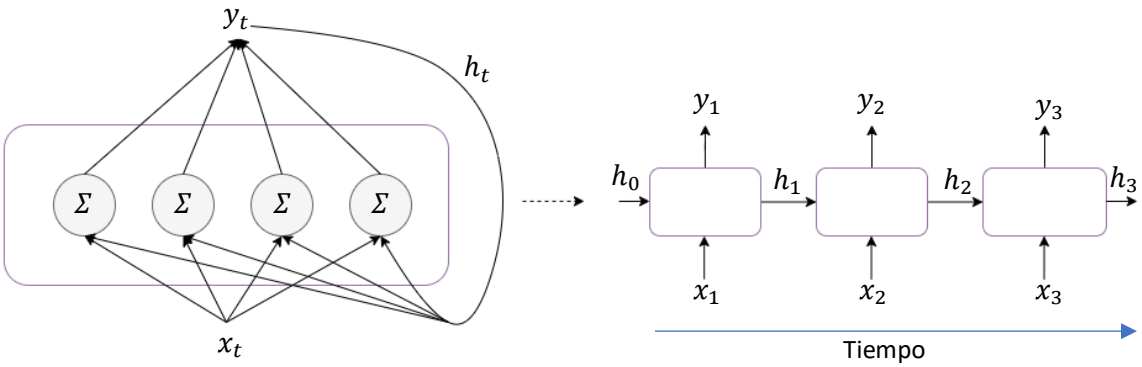


Figura 9. Una capa de neuronas recurrentes.

En una capa de neuronas recurrentes, cada neurona cuenta con tres conjuntos de pesos y sesgos: uno para las entradas x_t, otro para las entradas h_{t-1} y uno para las salidas h_t, estos pesos son w_x, w_h y w_y respectivamente. Considerando la capa completa, se puede representar estos pesos y sesgos en matrices de pesos y sesgos como W_x, W_h y W_y.

La representación de una RNN con estado oculto puede hacerse como muestra la Figura 10, donde ésta representa una celda recurrente, ya sea una sola neurona o una capa.

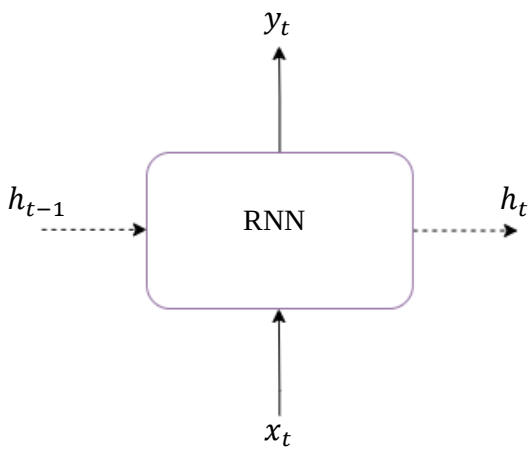


Figura 10. Celda recurrente.

Las salidas y_t y h_t, al igual que una RNA simple, se obtienen al realizar la suma ponderada respectiva y procesar ese resultado mediante una función de activación. Estos valores se calculan de la siguiente manera.

El resultado de la suma ponderada de la salida es:

$$z_t = W_y h_t + b_y \tag{1}$$

El valor de salida y_t:

$$y_t = f_1(z_t) = f_1(W_y h_t + b_y) \tag{2}$$

Donde f_1 es una función de activación.

Para obtener el valor del estado oculto h_t:

$$h_t = f_2(W_h h_{t-1} + W_x x_t + b_h) \tag{3}$$

Donde f_2 es una función de activación.

Dado que la salida y_t de la red depende del estado oculto h_t y, este a su vez, depende del estado oculto anterior h_{t-1}, se crea una relación donde la salida en el tiempo t depende de todos los valores generados anteriormente, produciendo una especie de memoria.

2.3. RNNs profundas

Al igual que una red perceptrón multicapa, una RNN puede formarse con más de una capa, tomando el término de red RNN profunda. En la Figura 11, del lado izquierdo se muestra un ejemplo de una RNN profunda de hasta tres capas cuya entrada es x_t, la salida es y_t y cuyos estados ocultos de cada capa son ${h_t}^{(k)}$ donde k representa el número de capa y t el paso de tiempo. Del lado derecho de la Figura 11 se muestra la misma RNN, pero desglosada en el tiempo hasta el tiempo T. En esta figura se puede observar cómo el estado oculto, además de pasarse entre sí cada capa en cada paso de tiempo, también es pasado de cada capa inferior a la capa superior.

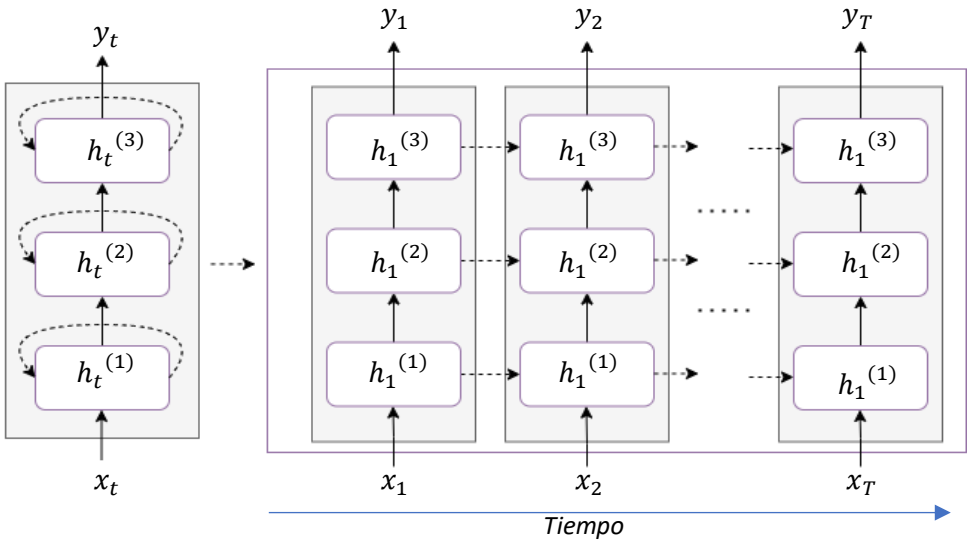


Figura 11. RNN con más de una capa.

2.4. Estructura de una red LSTM

Una celda recurrente, RNN, es una red muy básica, capaz de aprender la relación de la información secuencial de patrones muy cortos. Conforme transcurren más pasos de tiempo, el efecto de un paso dado, comienza a decrecer rápidamente como ilustra la Figura 12 donde un estado oculto $h_1$no tiene la misma influencia conforme avanza el tiempo de t_2 a t_6, a este problema se le conoce como problema de desvanecimiento del gradiente y está presente en las RNN con métodos de aprendizaje basados en gradientes y backpropagation, por ejemplo, para un paso de tiempo $t = 4$, el valor de salida y_4 depende del estado oculto h_4, que a su vez depende de los tres estados ocultos anteriores, así, en este punto, el efecto de h_0 en y_4 se

ha degradado por el procesamiento de cuatro funciones de activación. Otro ejemplo de este problema, pero en un nivel menos abstracto, puede verse en el procesamiento de textos, donde la red en un paso de tiempo presenta la palabra del pronombre "él" y en pasos de tiempo posteriores presenta el pronombre "ella", esto muestra que la red no fue capaz de recordar el pronombre que ha procesado. Por tal motivo se desarrollaron estructuras más complejas capaces de modelar relaciones de secuencias más largas en el tiempo. Tal es el caso de las redes de Memoria a Largo y Corto Plazo (LSTM: *Long Short-Term Memory*), que fueron propuestas originalmente en 1997 por Hochreiter y Schmidhuber [14], para ver más detalles de la historia de las RNA, en el Apéndice A se da un breve recorrido de esta.

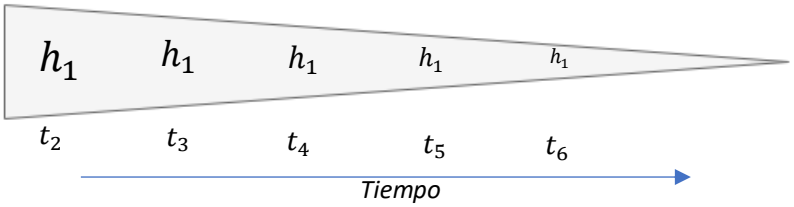


Figura 12. Decremento del efecto de un dato en el tiempo.

El componente básico de una red LSTM es una celda de memoria. Esta celda representa o reemplaza la capa oculta de las RNN estándar. A una red LSTM se le agregan diferentes elementos con respecto a la RNN estándar. A grandes rasgos, un elemento de suma importancia es la celda de estado, denotada a la salida de la red como C_t, la cual sirve como una especie de banda transportadora entre pasos de tiempo en la que se puede agregar o quitar información de la memoria. La Figura 13 muestra la estructura general de una celda LSTM. El funcionamiento de esta celda hace énfasis en las compuertas que la forman.

El valor de C_t se obtiene a partir de modificaciones del valor de C_{t-1} mediante distintas unidades de cálculo llamadas compuertas. Las compuertas $\otimes$ indican un producto por cada elemento y, la compuerta $\oplus$, indica una suma por elemento, x_t indica los valores de entrada y h_t los valores de salida del estado oculto. Además, se incluyen cuatro cacillas en donde se realiza la suma ponderada de los valores de entrada y el estado oculto con sus respectivos pesos, estos valores son procesados por una función de activación, en este caso con tangente hiperbólica, $tanh$, y sigmoide, σ.

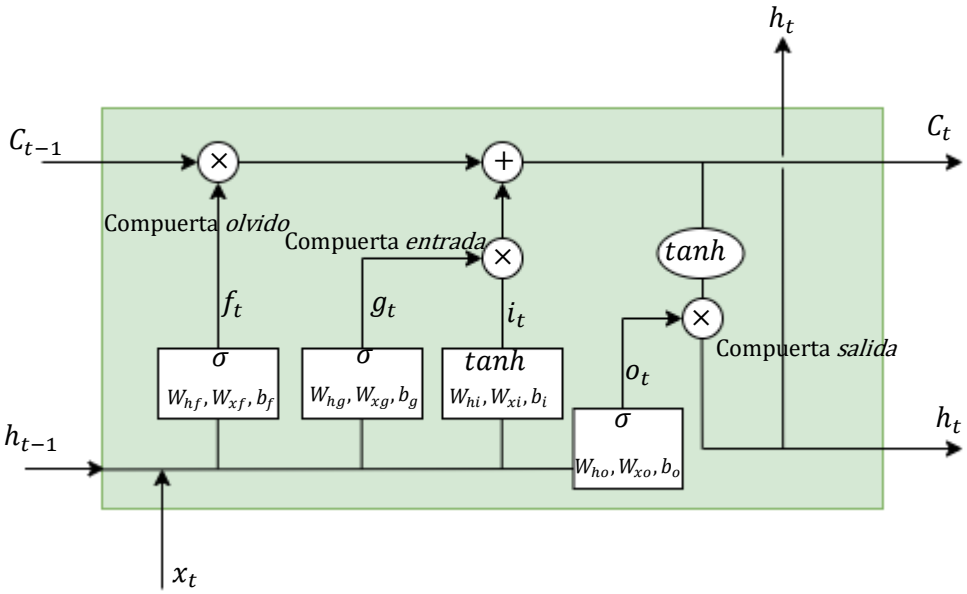


Figura 13. Celda LSTM.

La compuerta del olvido decide qué información del estado a largo plazo deben borrarse, es decir, decide qué información pasa y cuál se suprime. Esto lo logra al permitir que la memoria restablezca el estado de la celda sin crecer indefinidamente, Cuando f_t es cero equivale a cerrar la válvula y olvidar los datos en la memoria. El valor de f_t se calcula de la siguiente manera.

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{4}$$

Donde f_t representa la compuerta de olvido, σ es una función de activación, W_{xf} son los pesos asociados a x_t, W_{hf} son los pesos asociados a h_{t-1} y b_f los sesgos.

La compuerta de entrada es la responsable de actualizar el valor de la celda, es decir, decide qué información se agrega al estado a largo plazo a partir de los valores de g_t e i_t. Estos valores se calculan de la siguiente manera.

$$g_t = \sigma(W_{xg}x_t + W_{hg}h_{t-1} + b_g) \tag{5}$$

$$i_t = tanh(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{6}$$

Donde g_t representa la compuerta de entrada, i_t el valor candidato, W_{xg} son los pesos asociados a x_t, W_{hg} son los pesos asociados a h_{t-1} y b_g los sesgos. W_{xi} son los pesos asociados a x_t, W_{hi} son los pesos asociados a h_{t-1} y b_i los sesgos.

Así, el valor de la celda de estado, C_t, se calcula de la siguiente forma:

$$C_t = (C_{t-1} \otimes f_t) \oplus (g_t \otimes i_t) \tag{7}$$

La compuerta de salida decide cómo actualizar los valores del estado oculto en cada paso de tiempo a partir del valor de o_t. Este valor se calcula como:

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{8}$$

Donde o_t representa la compuerta de salida, σ es una función de activación, W_{xo} son los pesos asociados a x_t, W_{ho} son los pesos asociados a h_{t-1} y b_o los sesgos.

Así, el valor de salida, que también corresponde al valor del estado oculto, se calcula de la siguiente manera:

$$h_t = o_t \otimes \tanh(C_t) \tag{9}$$

Finalmente, al igual que una RNN estándar, con las celdas LSTM pueden crearse redes profundas en donde cada capa representa una celda LSTM, cada una con la cantidad de neuronas deseadas.

2.5. Estructura de una red GRU

La celda de Unidad Recurrente Cerrada (GRU: *Gated Recurrent Unit*) fue propuesta hace poco tiempo, comparada con la LSTM, en el año 2014 por Kyunghyun Cho [16]. La celda GRU es una versión simplificada de la LSTM, aunque se ha mostrado que pueden funcionar igual de bien [16]. Entre las principales diferencias se encuentran:

- Tanto el estado oculto como la celda de estado se fusionan en uno solo.

- La compuerta de olvido y la compuerta de entrada se sustituye por un controlador de compuerta único. Siempre que se deba almacenar en memoria, primero se borra la ubicación donde se almacenará.

- No hay compuerta de salida, sin embargo, hay una nueva compuerta que controla qué parte del estado anterior se mostrará en la capa principal.

La estructura de una celda GRU se muestra en la Figura 14. La celda GRU cuenta con un solo vector de estado, h_t. Éste es modificado por dos compuertas, una de multiplicación, $\otimes$, y una de suma, $\oplus$. Se incluyen tres cacillas en donde se realiza la suma ponderada de los

valores de entrada y el estado oculto anterior con sus respectivos pesos, los resultados son procesados por una función de activación, en este caso son tangente hiperbólica, $tanh$, y sigmoide, σ. Se cuentan con dos compuertas principales, la compuerta de reinicio y la de actualizar.

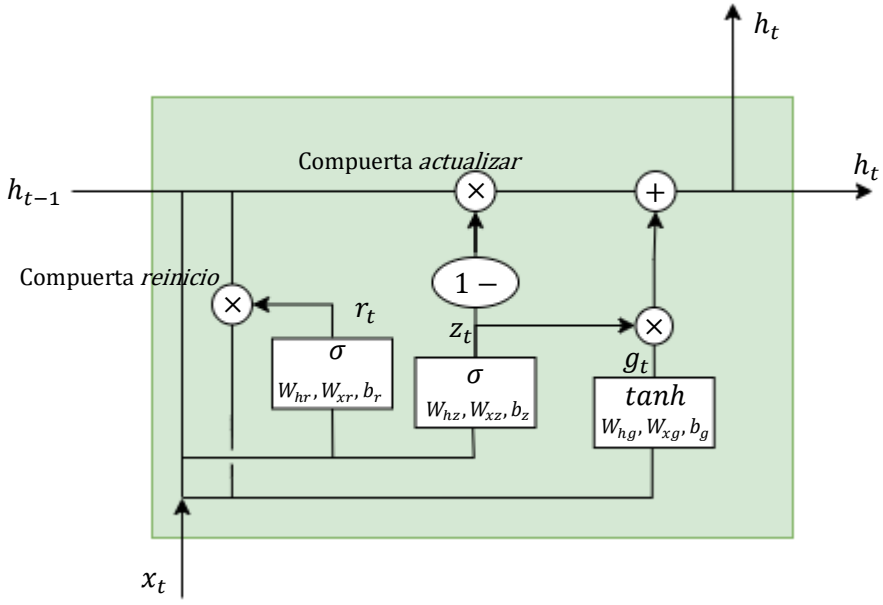


Figura 14. Celda GRU.

La compuerta actualizar hace la función de la compuerta de entrada y de olvido de la LSTM mediante el valor de z_t que se calcula como muestra la Ecuación (10).

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \tag{10}$$

Donde z_t representa la compuerta de actualizar, σ es una función de activación, W_{xz} son los pesos asociados a x_t, W_{hz} son los pesos asociados a h_{t-1} y b_z los sesgos.

La compuerta de reinicio controla qué parte del estado oculto anterior se mostrará en la capa principal, g_t, a partir del valor r_t. Estos valores se calculan como se muestra a continuación:

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \tag{11}$$

$$g_t = tanh(W_{xg}x_t + W_{hg}(r_t \otimes h_{t-1}) + b_g) \tag{12}$$

Donde r_t representa la compuerta de reinicio, σ es una función de activación, W_{xr} son los pesos asociados a x_t, W_{hr} son los pesos asociados a h_{t-1}, b_r los sesgos y $g_t$es la capa principal.

El valor final de estado oculto, h_t, se calcula como muestra la Ecuación (13).

$$h_t = z_t \otimes h_{t-1} + (1 - z_t) \otimes g_t \tag{13}$$

Las celdas GRU no deben verse como un caso especial de LSTM. Ambos modelos tienen un rendimiento aproximado que depende de la tarea en cuestión para la que se les entrene. La LSTM ha sido probada más ampliamente que la GRU debido a que es una estructura más antigua y se ha hecho de gran popularidad [15].

Capítulo 3 Modelos generativos de música

En el presente trabajo se desarrollan dos modelos generativos con el objetivo de comparar su desempeño en la generación de melodías. Ambos modelos se enfocan en la manipulación de los mismos conceptos musicales, los cuales son las notas, los acordes, los silencios y la duración de cada uno y su diferencia radica en la forma en que son manipulados y en la arquitectura de las RNA (Redes Neuronales Artificiales).

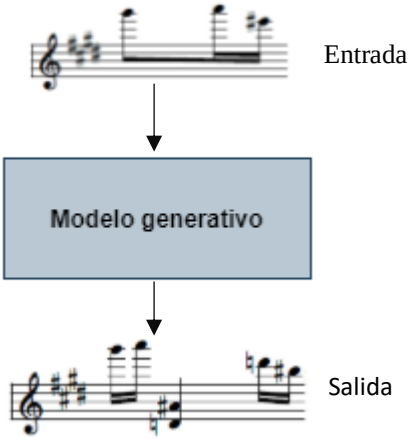


Figura 15. Modelo generativo de música.

Como muestra la Figura 15, ambos modelos generativos tendrán a la entrada datos correspondientes a una melodía y generará una nueva. Se abordará a detalle cada uno de los modelos en las siguientes subsecciones.

En cuanto a la representación de la información, se opta por una representación simbólica a partir del formato estándar midi [17] ya que este tipo de representación (comparada con audio sin procesar) refleja el proceso compositivo parecido a una partitura y es más sencillo realizar un análisis armónico.

El tipo de contenido que se pretende generar es una polifonía de una sola pista[12] en la que se presenta una secuencia de notas, acordes y silencios correspondientes a un solo instrumento y este puede tocar más de uno de estos elementos a la vez.

[12] También conocida como polifonía de una sola voz.

3.1. Modelo 1

Para generar la polifonía, primero se genera la melodía principal y posteriormente el acompañamiento. Para ello se implementa una arquitectura a base de RNN. Se exploran los resultados de generar melodías con los instrumentos por separados, es decir, una RNA se entrena para aprender la melodía principal y una segunda se entrena para aprender los acompañamientos, sin condicionar uno al otro.

3.1.1. Preprocesamiento de los datos

El preprocesamiento de las melodías se basa en traducir la información representada en formato midi a un formato secuencial adecuado para introducirla en la red propuesta. Para el set de melodías de entrenamiento, primero se realiza una depuración para obtener las canciones polifónicas de hasta dos instrumentos, separando en archivos diferentes la melodía principal y el acompañamiento ya que ambos se utilizarán en el mismo modelo, pero en diferentes entrenamientos para generar la melodía polifónica, y después se realiza la codificación de la información musical de interés: notas, acordes, silencios y sus duraciones.

Para obtener las muestras de datos secuenciales de las melodías, se hace uso de la librería music21 [18] y se realizan los siguientes pasos.

1. **Lectura de los archivos**.

Se realiza la lectura de todos los archivos ".mid", donde cada uno de estos archivos corresponde a una melodía y estas son las que se introducen a la red para su entrenamiento. Posteriormente cada archivo se guarda en una cadena de *streams* de *music21*, en la Figura 16 se ilustra esta cadena donde cada *music21.stram* corresponde a un archivo midi.

Figura 16. Archivos MIDI leídos.

2. **Lectura de eventos midi**.

Una vez hecha la lectura de los archivos midi, se realiza la lectura de cada una de las melodías almacenadas para obtener cada uno de los eventos midi y obtener las notas, acordes y silencios junto con su duración.

Cada nota se representa con su número midi, cada acorde se representa como una sucesión de números de clase de tono separado por un punto, cada silencio se representa con el carácter "r" y el tiempo de cada uno de estos elementos se representa con el símbolo "_", el cual intenta modelar el ritmo de las melodías indicando una retención del elemento en cuestión. A su vez, cada nota, acorde y silencio indican una duración de 0.25, y cada "_" indica una retención de 0.25 más. La Figura *17* ilustra un fragmento de la codificación donde, por ejemplo, un acorde de dos notas y con duración de una negra se representa por su vector midi de 9.2 seguido de tres "_" y esto ocurre para cada elemento de la partitura.

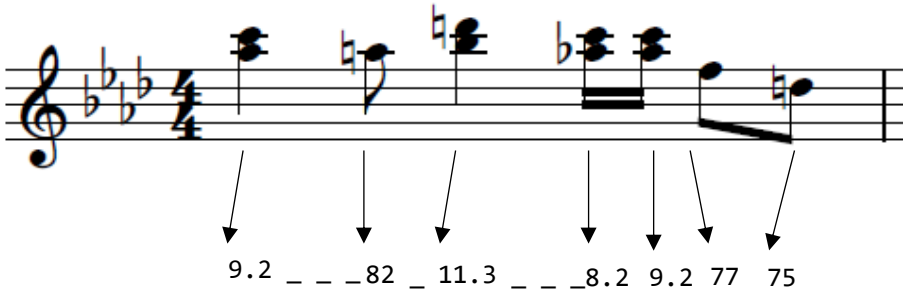


Figura 17. Codificación de los eventos midi.

3. **Creación del vocabulario**.

Cuando se tienen todas las melodías codificadas, se obtienen todos los elementos del vocabulario (nota, acorde, silencio y prolongación de duración) y se tokenizan asignándoles un número entero a cada uno comenzando en 0 y por orden de aparición en la codificación. Por ejemplo, el "48" que representa un C_3 en midi, será convertido en un 0, de esta forma en una cadena de enteros, cada que se vea un 0 se estará haciendo referencia a un C_3. La Figura 18 ilustra un fragmento del diccionario generado almacenado como un archivo JSON.

```
{
    "48": 0,
    "_": 1,
    "r": 2,
    "52": 3,
      ⋮
    "58": 30,
    "49": 31,
    "8.2": 32,
    "67": 33,
    "4.9.7": 34
       ⋮
}
```

Figura 18. Diccionario de símbolos.

Con esto, se traduce toda la secuencia codificada a números enteros por cada elemento en cada canción del set de entrenamiento. La Figura 19 muestra un ejemplo de un fragmento de esta conversión donde una nota midi que corresponde al 71 se le asigna un numero 32 con respecto al diccionario y esto se realiza para cada elemento de las secuencias.

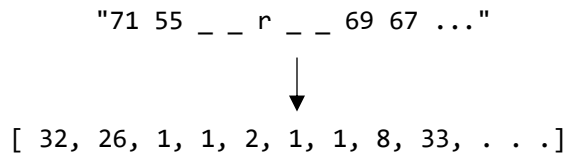


Figura 19. Conversión de símbolo mediante el diccionario.

4. **Muestras de entrenamiento.**

El siguiente paso en el preprocesamiento de los archivos midi es la obtención de las muestras de entrada y de salida.

Cada muestra de entrada es una secuencia de n datos y cada muestra de salida es un solo dato, $n + 1$, cada muestra se obtiene a partir de recorrer una posición a la derecha el índice a partir del cual se comienza la elección de los valores. Así, la RNN debe predecir el dato $n + 1$ a partir de los primeros n. La Figura 20 ilustra la obtención de tres muestras de entrada y de salida, donde las entradas son arreglos de tamaño n y las salidas son arreglos de tamaño 1. El valor de salida será el que deberá predecir la red y este dependerá de los n datos de la secuencia anterior al valor de salida.

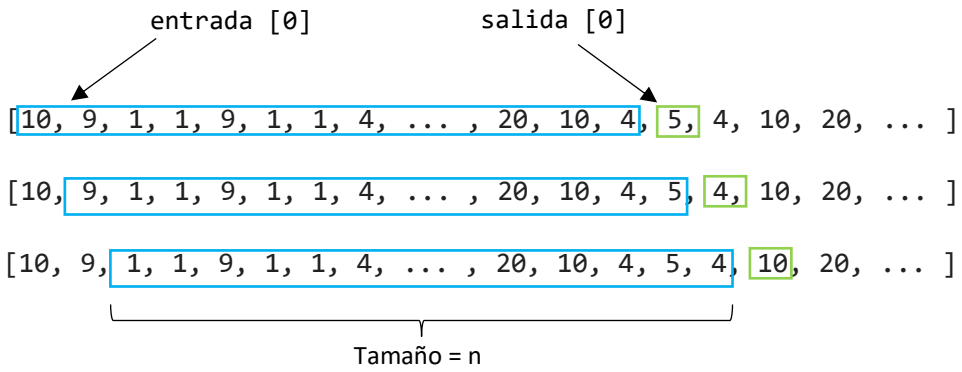


Figura 20. Obtención de tres muestras de entrada y de salida.

5. **Codificación one-hot de las muestras.**

Finalmente, las secuencias de entrada se codifican en vectores one-hot. Estos vectores son una forma de codificación para representar valores categóricos en los que la relación de orden que ofrece una representación mediante números enteros no es conveniente. La dimensión de los vectores one-hot es igual al tamaño del vocabulario y todas sus entradas son cero excepto aquella que representa la clase a la que pertenece. Esta representación vectorial es conveniente en comparación con una representación de valores enteros ya que cada uno de los pasos de tiempos en las melodías es visto como un valor categórico que representa un sonido en particular y no presentan una relación de orden.

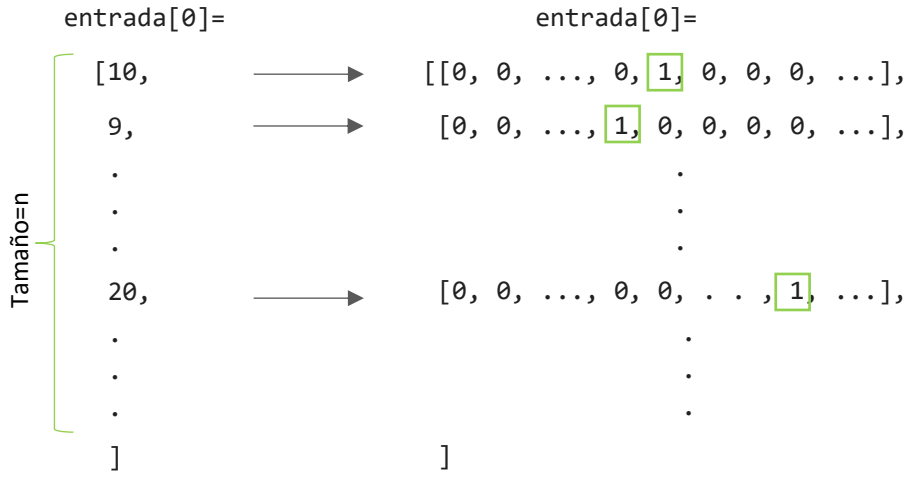


Figura 21. Ejemplo de codificación one-hot.

En la Figura 21 se ilustra un ejemplo de una de las secuencias de entrada codificadas a vectores one-hot, como se observa, cada valor de la secuencia se transforma a un vector tan grande como el tamaño del vocabulario y se coloca un 1 en el la posición del índice establecido por el valor entero.

3.1.2. Arquitectura de la red

Para generar las melodías se aborda la estrategia feedforward iterativo, lo cual consiste en establecer una secuencia inicial de n valores y el modelo creará la melodía principal prediciendo el valor $n + 1$. Posteriormente, para generar el acompañamiento se recibirá de la misma forma la secuencia correspondiente al acompañamiento.

En la Figura 22 se muestra la arquitectura correspondiente a la red del primer modelo. Está formada por una capa de entrada donde se le proporcionan las secuencias de tamaño n y en cada valor de la secuencia se tiene un vector one-hot, seguida de capas ocultas recurrentes con su correspondiente dropout y como salida una capa densa con tantas unidades de salida como el tamaño del vocabulario. Las capas ocultas recurrentes ilustradas en la Figura 22 corresponden a celdas LSTM, aunque también se explora el modelo con celdas GRU.

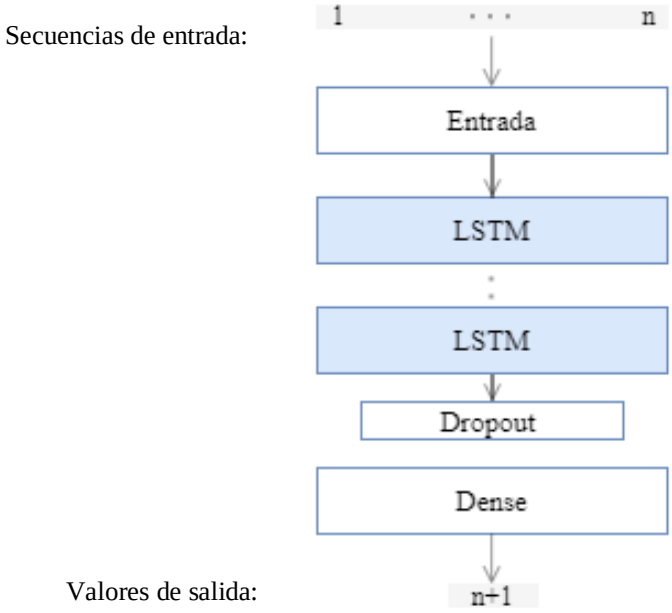


Figura 22. Arquitectura de la red del modelo 1.

Cada muestra de entrada será introducida a la red a través de una capa de entrada cuya longitud es n y cada valor de esta secuencia es un vector one-hot tan grande como el vocabulario generado, como muestra la Figura 23. La capa de entrada cuenta con n nodos y estos conectan su salida, que corresponden a los vectores one-hot y a su vez, cada vector corresponde a un paso de tiempo.

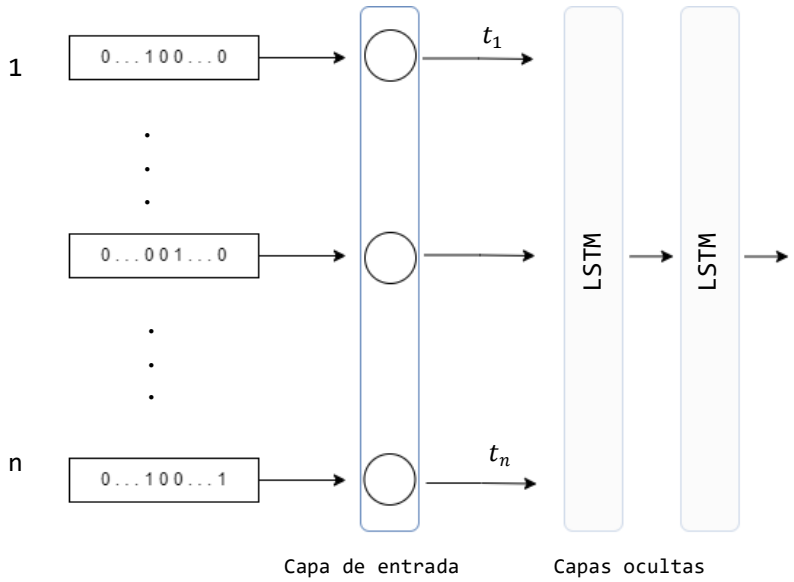


Figura 23. Entrada de datos de la arquitectura del modelo 1.

Adicionalmente, cada capar oculta LSTM, exceptuando la última, su propiedad *return_sequence*[13] se *establece* en true para que la siguiente capa reciba la secuencia de estados ocultos y no solo la última. Esto ya que, en una capa oculta LSTM, normalmente es de interés el ultimo valor de la celda de estado que corresponde al valor predicho en la posición $n + 1$ de la secuencia, por otro lado, en las capas LSTM ocultas que anteceden a la última capa oculta LSTM, son de interés la secuencia completa de los estados ocultos que corresponden a los estados ocultos en cada paso de tiempo ya que estas secuencias son las entradas de la capa siguiente.

Como muestra la Figura *24*, a la salida de la red se encuentra una capa densa con tantas unidades como el tamaño del vocabulario L y con función de activación *Softmax* para así obtener la salida con la probabilidad más alta y determinar el índice correspondiente a la clase del vector one-hot. Por tal motivo se utiliza como función de pérdida la entropía cruzada categórica escasa. Al igual que en el modelo 1, se utiliza la función Softmax a la salida y se utiliza como función de pérdida

[13] Esta propiedad es propia de Keras, la biblioteca utilizada en este trabajo.

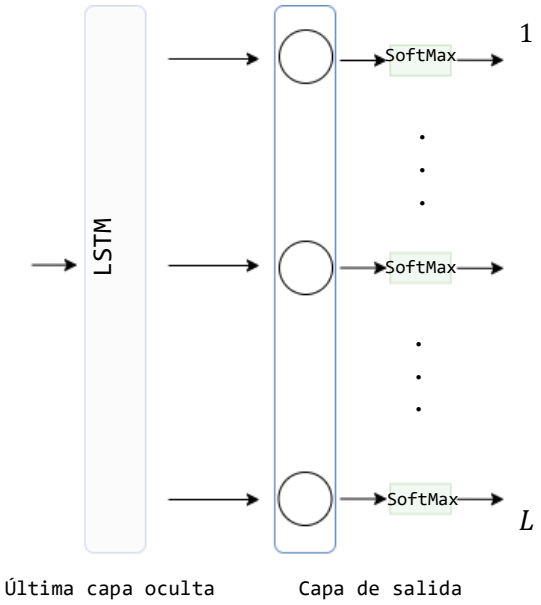


Figura 24. Salida de la red del modelo 1.

En la Figura 25 se muestra el proceso completo de procesamiento de la información en la red desde la partidura (un fragmento) hasta las neuronas en la capa de salida. Como entrada se tiene una partitura y, siguiendo el flujo del primer acorde, este se codifica según el estándar midi como un "9.2" y, ya que su duración es de una negra, en la secuencia se colocan tres "_" que junto con el acorde suman la duración total de este. Posteriormente se transforman en enteros y el acorde es representado con un "72", mientras que el "_" con un "1", que luego se transforman en vectores one-hot. Después de obtener las secuencias de tamaño n, son ingresadas a la red por una capa de entrada que sirve de transporte de los vectores one-hot a la primera capa oculta LSTM. Las capas ocultas LSTM, exceptuando la última, tienen como salida la secuencia de estados ocultos correspondientes a cada uno de los pasos de tiempo en la secuencia de entrada y, cada una de estas secuencias de estados ocultos, tiene tantos valores como la cantidad de unidades que se especifiquen por celda[14]. La última capa LSTM tiene a su salida la celda de estado correspondiente al último valor de la secuencia de entrada y esta es un vector tan grande como la cantidad de unidades que se especifiquen por celda. Esta última salida es introducida a una capa densa que tiene tantas unidades como el tamaño del vocabulario y se asigna una función de activación *softmax* para que cada uno de los nodos de salida de la probabilidad correspondiente al índice de las posiciones de los vectores one-hot y poder tomar el más alto, siguiendo el ejemplo, a la salida se tiene que la posición con la

[14] En el api de Keras, este atributo se llama "units" en la capa LSTM.

mayor probabilidad es el con el índice “2”, esto se puede interpretar diciendo que es la palabra del vocabulario con el número entero “2” asignado.

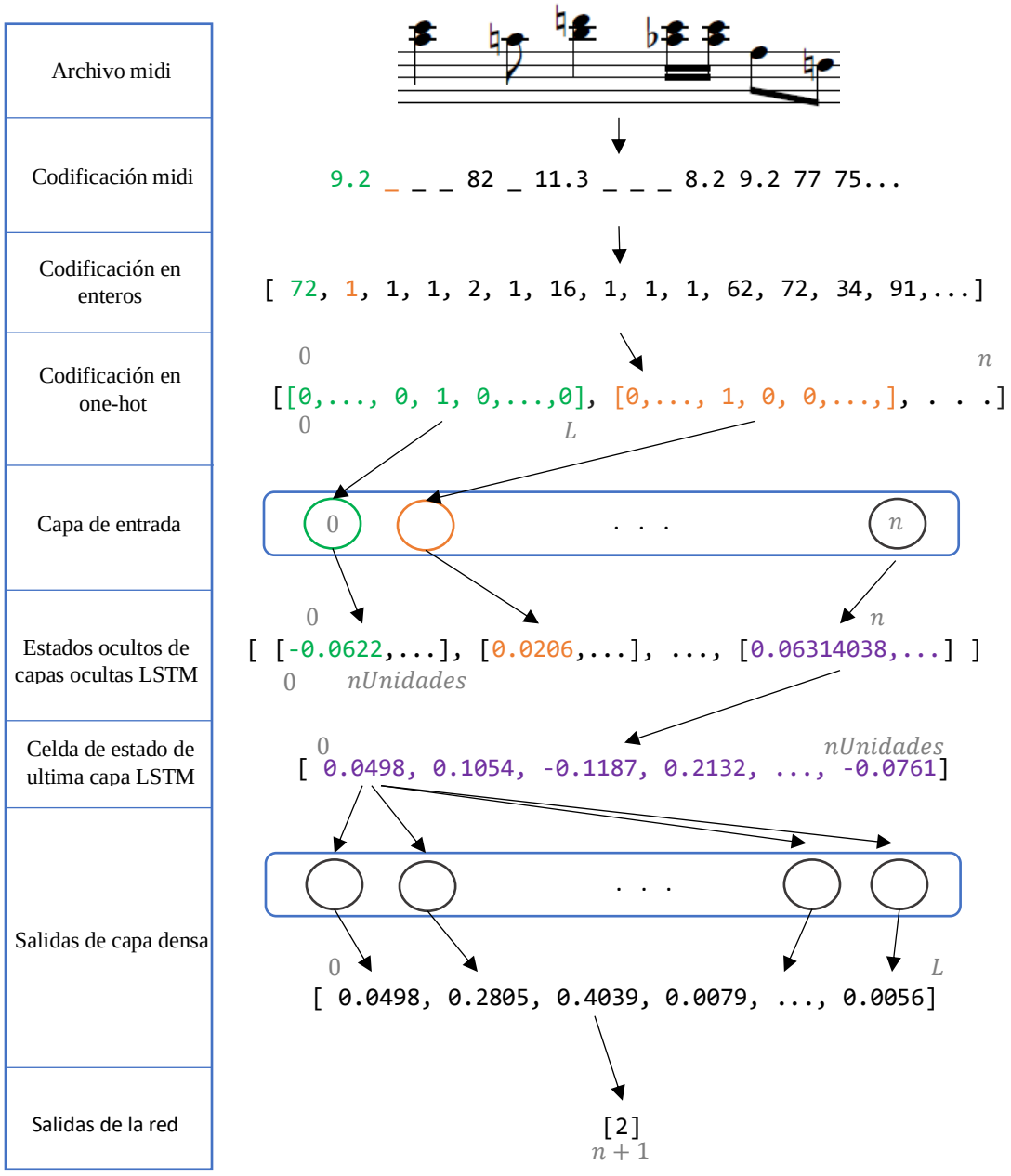


Figura 25. Proceso de generación de melodías en la red del modelo 1.

3.1.3. Posprocesamiento de los datos

El posprocesamiento se enfoca en realizar la traducción de la secuencia de datos predichos a un formato midi. Para ello primero se hace la predicción de un valor a partir de los n valores ingresados a la red. Ese valor predicho se obtiene como un valor entero establecido en el diccionario. Por lo tanto, se utiliza ese diccionario para obtener su correspondiente símbolo y se almacena en un arreglo que contiene los símbolos propuestos como semilla inicial y los símbolos predichos como muestra la Figura 26 donde se ejemplifica la secuencia de elementos correspondientes a la melodía generada y se concatena uno de los nuevos valores al final de esta.

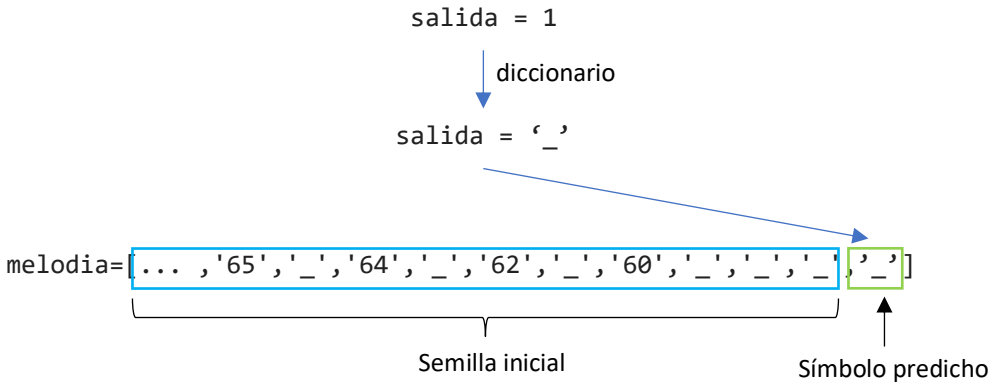


Figura 26. Obtención del símbolo predicho.

Para almacenar la melodía con un formato midi, cada vez que se detecta una nota o silencio, se obtiene su duración y se crea el objeto de evento de music21 y se agrega a un *stream* de *music21*. Al terminar de recorrer la melodía completa, el *stream* se almacena con formato midi. La Figura 27 muestra un ejemplo donde se detectan elementos y su duración en un fragmento de una secuencia de elementos codificados correspondientes a la melodía y se convierten a eventos midi para poder ser anexados al *stream* de music21.

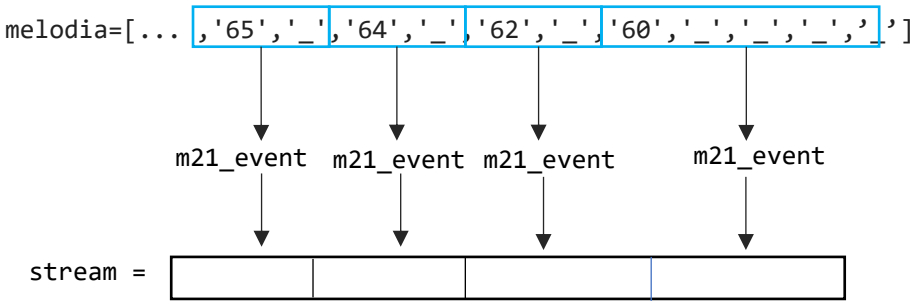


Figura 27. Almacenamiento de melodía en formato midi.

3.2. Modelo 2

Como se mencionó, el segundo modelo propuesto también tiene el propósito de generar una polifonía de una sola pista, salvo con dos diferencias. La primera diferencia radica en la forma en que se representa la duración de cada nota, acorde y silencio. La segunda diferencia se encuentra en la arquitectura de la red. Esto lleva a implementar una codificación de las melodías distinta.

3.2.1. Preprocesamiento de los datos

Para este modelo, el preprocesamiento es diferente ya que se separa la duración de cada elemento en un nuevo arreglo, mientras que las notas, acordes y silencios se representan en un mismo arreglo. Para el set de melodías de entrenamiento se realiza la misma depuración para separar la melodía principal y el acompañamiento.

Para obtener las muestras de datos secuenciales de las melodías, se hace uso de la librería music21 [18] y se realizan los siguientes pasos.

1. **Lectura de los archivos**.

Al igual que el modelo anterior, se realiza la lectura de todos los archivos “.mid” y se guardan en una cadena de streams de music21.

2. **Lectura de eventos midi**.

Una vez hecha la lectura de los archivos midi, se realiza la lectura de cada una de las melodías almacenadas para obtener cada uno de los eventos midi y obtener las notas, acordes

y silencios junto con su duración. Para este segundo caso se crean dos secuencias distintas, como muestra la Figura 28, una de las secuencias corresponde a los tonos, sean notas o acorde y a los silencios, mientras que la segunda secuencia corresponde a las duraciones de cada uno de los elementos en la secuencia de tonos.

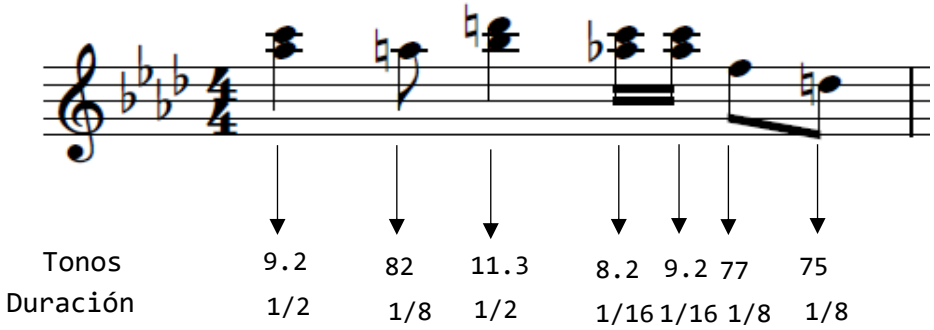


Figura 28. Codificación de los eventos midi, segundo modelo.

3. **Creación del vocabulario**.

Cuando se tienen todas las melodías codificadas, se obtienen todos los elementos del vocabulario para ambas secuencias y se tokenizan asignándoles un número entero a cada uno comenzando en 0 y por orden de aparición en la codificación. La Figura *29*Figura *18* ilustra un ejemplo del diccionario generado para ambas secuencias almacenados en formato JSON.

```
Tonos:                  Duraciones:
{                       {
    "9.10.2": 0,            "6.25": 0,
    "10.2.3.4": 1,          "7.0": 1,
    "11.3": 2,              "17/3": 2,
    "9.0.1.4.5": 3,         "5.25": 3,
          :                       :
          :                       :
    "9.0.2.3": 89,          "11.25": 56,
    "65": 90,               "9.75": 57,
    "4.8": 91,              "25.5": 58,
    "76": 92,               "73.0": 59,
    "78": 93,               "43/12": 60,
          :                       :
          :                       :
}                       }
```

Figura 29. Diccionario de símbolos.

Con esto, se traduce toda la secuencia codificada a números enteros por cada elemento en cada canción del set de entrenamiento como se explicó en la sección 3.1.1. La Figura *30*

Muestra un fragmento de una secuencia de tonos y una secuencia de duración de cada uno de estos tonos a su correspondiente número estero establecido en los diccionarios.

```
"9.2 82 11.3 8.2 9.2 77 75 ..." ⟶  [ 23, 8, 7, 12, 23, 1, 79, . . .]

"1/2 1/8 1/2 1/16 1/16 1/8 1/8 ..." ⟶  [ 0, 7, 0, 8, 8, 7, 7, . . .]
```

Figura 30. Conversión de símbolo mediante el diccionario.

4. **Muestras de entrenamiento.**

Al igual que el modelo anterior, cada muestra de entrada son dos secuencias de n datos y cada muestra de salida son dos datos que ocupan la posición $n + 1$, correspondientes a los tonos y a las duraciones. Cada muestra se obtiene a partir de recorrer una posición a la derecha el índice a partir del cual se comienza la elección de los valores en ambas secuencias.

Para este modelo, las secuencias son ingresadas como una serie de números enteros, no se convierten en vectores one-hot ya que la arquitectura de la red cambia.

3.2.2. Arquitectura de la red

Para este modelo también se sigue la estrategia feedforward iterativo, pero la arquitectura cambia drásticamente. Ya que se realiza una separación de los tonos y sus duraciones, la arquitectura de la red, mostrada en Figura 31, tiene dos entradas, seguidas de una capa de incrustación[15] cada una, la salida de estas capas se concatenan y son ingresadas a las capas ocultas recurrentes, finalmente, la salida de las capas recurrentes son mandadas a una capa densa y posteriormente a otras dos capas densas separadas donde cada una dará a la salida el valor predicho correspondiente al tono y su duración.

Al igual que el modelo 1, las capas recurrentes de esta arquitectura se representan como celdas LSTM, aunque también se implementas GRU para comparar su desempeño.

[15] O Embedding en inglés.

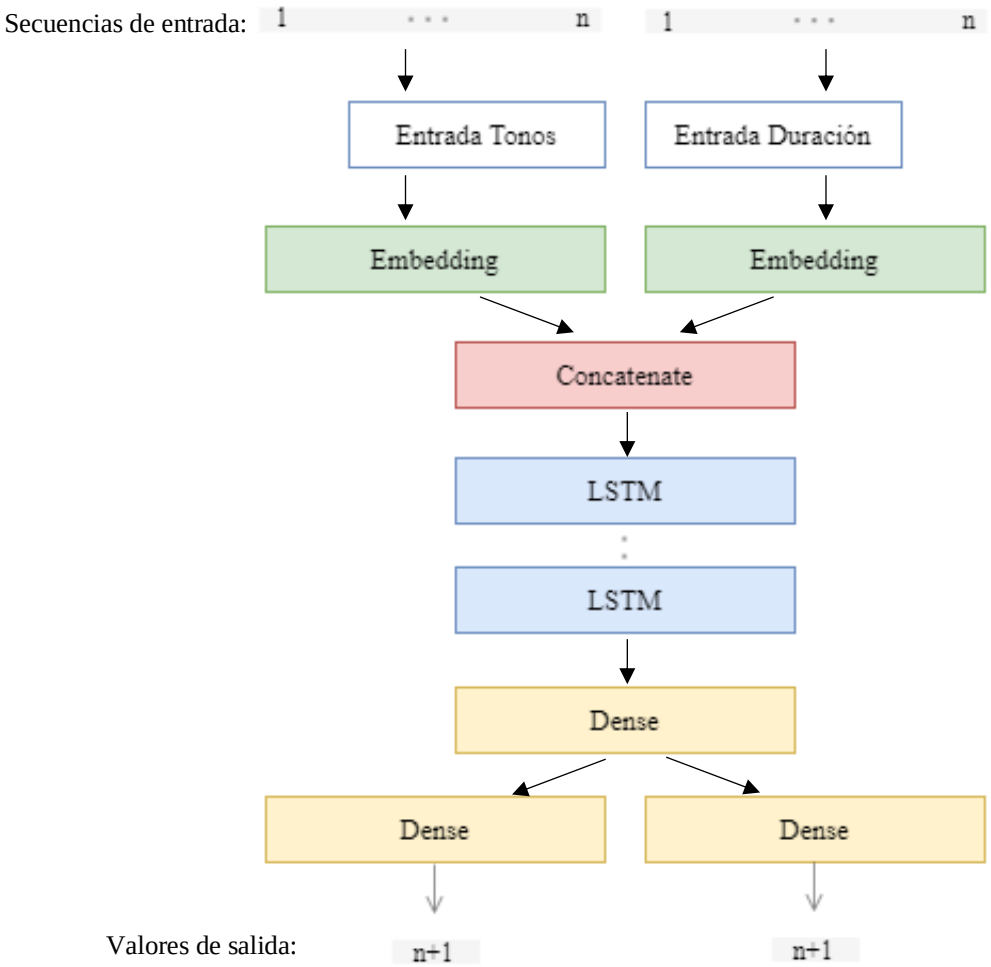


Figura 31. Arquitectura de la red del modelo 2.

La capa de incrustación se encarga de realizar una especie de reducción de dimensionalidad de todo el set de datos, es decir, si el vocabulario cuenta con 550 palabras y los vectores generados por la capa de incrustación son de tamaño 64, todo el conjunto de datos queda representado por un vector de tamaño 64 de tipo de datos *float*, donde cada valor del vector puede representar una característica de las palabras del vocabulario. En la Figura 32 se muestra un ejemplo de la transformación de una secuencia de entrada de tamaño n a vectores de la capa de incrustación de tamaño n_E[16].

[16] Se especifica mediante el atributo output_dim de la capa Embedding de Keras la dimensionalidad del vector generado por cada palabra.

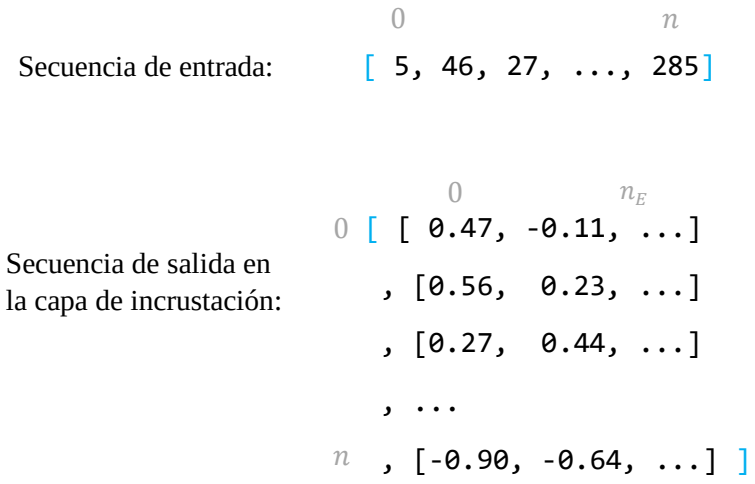


Figura 32. Ejemplo de la capa de incrustación.

En la Figura 33 se observa a más detalle la entrada de la red hasta antes de ser introducidos los valores a las capas ocultas recurrentes. Cada capa de entrada tiene una longitud tan grande como la secuencia de entrada n, estas secuencias de entrada son valores enteros desde el 0 hasta el tamaño del vocabulario, tanto para los tonos como para sus duraciones. Cada capa de entrada es dirigida a una capa de incrustación, la cual transportará cada valor entero a un vector de longitud n_E con valores tipo *float*. Posteriormente la capa de concatenación une las salidas de las capas de incrustación en un solo vector del doble de la secuencia de entrada.

Finalmente, la salida de la capa de concatenación es la entrada de la primera capa recurrente oculta y se puede hacer la analogía con la entrada de la red del modelo 1, con la diferencia que en lugar de ser vectores one-hot los valores de entrada, son vectores creados por las capas de incrustación.

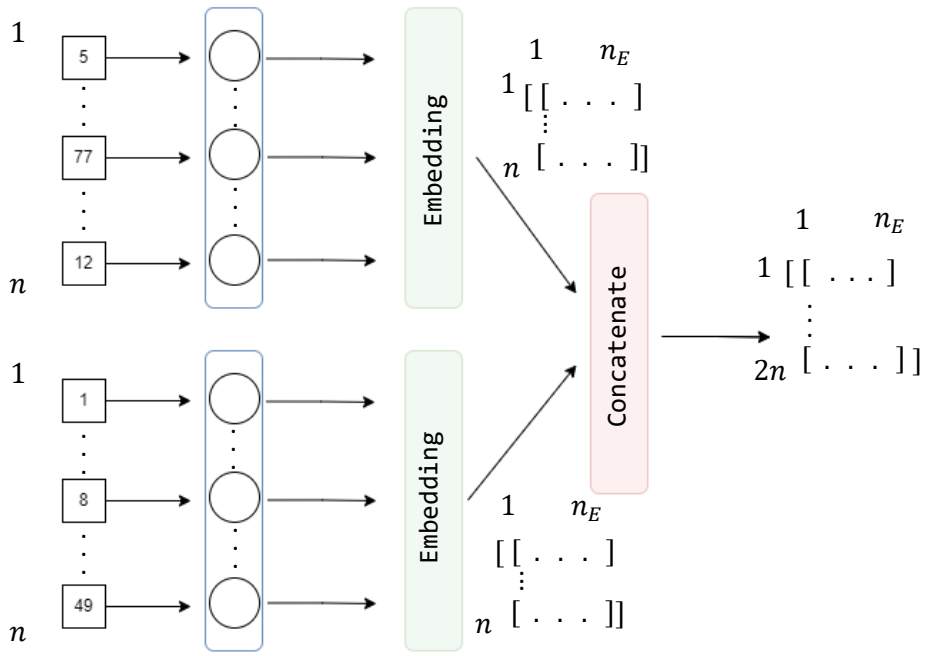


Figura 33. Entrada de datos de la arquitectura del modelo 2.

En cuanto a la salida de la red, la Figura 34 ilustra más detalles, como se observa, seguido de la última capa oculta recurrente, se conecta una capa densa seguida de otras dos capas densas por separado. Estas dos capas densas tienen tantas unidades como el tamaño del vocabulario de los todos, n_T, y de las duraciones, n_D, así, en cada capa se intenta predecir la siguiente nota y su duración por separado. Al igual que en el modelo 1, se utiliza la función *Softmax* a la salida y se utiliza como función de pérdida la entropía cruzada categórica escasa.

Como ejemplo de los valores de salida se ilustra con la estos valores.

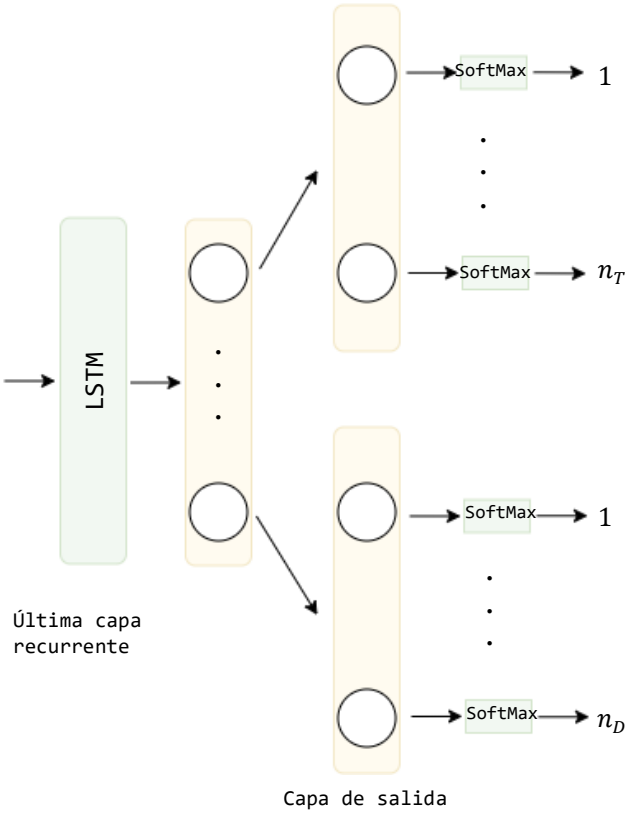


Figura 34. Salida de la red del modelo 2.

Capítulo 4 Experimentos y resultados

En esta sección se presentan las pruebas realizadas a los distintos modelos propuestos, así como los resultados obtenidos, además, se mencionan el conjunto de datos utilizados para el entrenamiento. Finalmente se presentan unas muestras de las melodías generadas.

4.1. Consideraciones de implementación

En este trabajo se utiliza un conjunto de melodías del género blues, y en particular, del subgénero boowie googie. Se realiza un conjunto de experimentos sobre ambos modelos para evaluar su desempeño e intentar calibrar los hiperparámetros. Las redes fueron generadas usando Keras con el backend de Tensorflow y el entrenamiento se realizó en Google colab con una tarjeta de video Nvidia K80.

Como sabemos, el entrenamiento de una RNA se basa en encontrar los pesos de las conexiones entre nodos, estos son conocidos como parámetros, pero, además de estos, también se presentan hiperparámetros en una RNA que establecen la estructura y el control de esta. En este trabajo se ajustan algunos hiperparámetros de estructura como el número de capas, el número de unidades por capa, porcentajes de dropout y, de control como tamaño de lote y épocas. Cabe resaltar que la búsqueda de estos hiperparámetros se realiza de una manera manual explorando solo un subconjunto de las combinaciones de los valores propuestos y no de forma automatizada.

La calibración de los hiperparámetros en implementaciones de generación de contenido artístico suele ser más complicado, comparado con tareas convencionales de predicción o clasificación, ya que en estas la medida de evaluación como los valores de error y precisión es más objetiva [1].

Para los modelos se realizan distintas configuraciones en los hiperparámetros como muestra la Tabla 1.

Tabla 1. Hiperparámetros explorados en el módulo concurrente.

Hiperparámetros	**Valor**
Unidades por capa recurrente	50, 100, 150, 256, 512, 1024
Capas LSTM/GRU	1, 2, 3, 4
Longitud de muestra	32, 64
Tamaño de lote	32, 64, 96
Cantidad de capas dropout	1, 2, 3, 4
Épocas	30, 50, 100

En la implementación con distintas capas recurrentes, se usan celdas LSTM y GRU por separado. Para cada capa recurrente se experimenta colocando dropout después de ellas y sin dropot, este con un valor del 20%.

El set de melodías usado cuenta con 62 canciones, las cuales se separan en melodía principal y acompañamiento. Así mismo, se configura la red para que las muestras se ingresen de forma aleatoria en cada época.

No se realiza una exploración en todas las combinaciones de los hiperparámetros mostrados, más bien, se utilizan los valores más representativos conforme se realizan las pruebas. Dentro de los hiperparámetros explorados, algunas de las corridas con mejores resultados y representativas son las siguientes:

- **Corrida 1:** Una capa recurrente LSTM, 512 unidades, longitud de muestra igual a 32, 50 épocas.
- **Corrida 2:** Tres capas recurrentes LSTM, 512 unidades, longitud de muestra igual a 32, 50 épocas.
- **Corrida 3:** Tres capas recurrentes LSTM, 1024 unidades, longitud de muestra igual a 32, 50 épocas.
- **Corrida 4:** Tres capas recurrentes GRU, 512 unidades, longitud de muestra igual a 32, 50 épocas.

4.2. Decodificación

Como se sabe, el entrenamiento de las redes neuronales artificiales se basa en obtener la probabilidad que se asigna a cada una de las clases a la salida, se ha demostrado que algunos enfoques basados en la maximización, es decir, obtener la clase con la mayor probabilidad, han conducido a grandes resultados para algunas tareas en específico, sin embargo, para algunas tareas de generación de secuencias temporales pueden conducir a generar contenido degenerado en los cuales se detectan incoherencias o bucles muy repetitivos [19]. Por tal motivo, en este trabajo se ha optado por una decodificación basada en un muestreo[17] de temperatura para generar las melodías, para saber más detalles sobre este tipo de decodificación y conocer otros más puede dirigirse al apéndice C.

Para aplicar el muestro de temperatura se hace uso de la ecuación (14).

$$token_p = \left(softmax\left(\frac{\overrightarrow{y_t}}{t}\right)\right) \tag{14}$$

Donde t representa la temperatura, $\overrightarrow{y_t}$ es el vector de valores que resultan de aplicar el logaritmo natural a las probabilidades dadas por *softmax* a la salida de la RNN y después dividido por la temperatura t y $token_p$ representa la nueva distribución de probabilidades resultante. Posteriormente se realiza una elección aleatoria dada la nueva distribución de probabilidad.

4.3. Resultados

En esta sección se comparan los valores en la precisión y la pérdida obtenidos durante el entrenamiento de los modelos. En la Tabla 2 se muestran los valores correspondientes a la pérdida y precisión obtenidas en el entrenamiento de la red del modelo 1 para las cuatro corridas. Claramente la implementación de celdas LSTM logra un mejor desempeño que las celdas GRU, así como el aumento en las unidades por cada capa oculta y el número de capas.

Tabla 2. Valores de pérdida y precisión en el entrenamiento de la red del modelo 1.

Corridas	Pérdida	Precisión
1	1.076	0.710
2	0.201	0.957
3	0.173	0.989
4	2.046	0.498

[17] Por muestreo se refiere a elegir el siguiente token en una secuencia aleatoriamente a partir de una distribución de probabilidad dada.

También se observó en los experimentos que la manipulación en la longitud de secuencia y en el tamaño del lote no tiene gran efecto en los resultados del entrenamiento en esta implementación.

La Figura 35, Figura *36* y Figura *37* muestran el comportamiento de los valores de precisión y pérdida para los experimentos donde se implementan celdas LSTM en las capas recurrentes. Estas son las que logran un mejor desempeño alcanzando valores de precisión más altos y valores de pérdida más bajos.

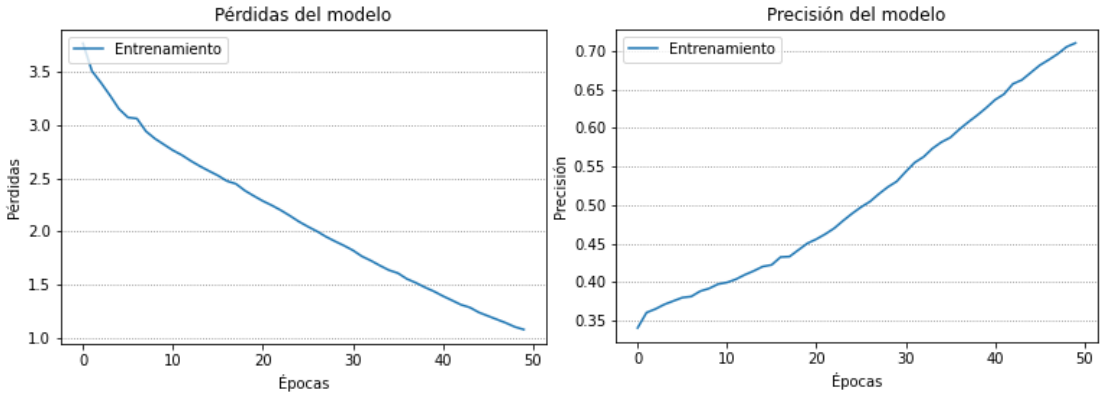


Figura 35. Pérdida y precisión de la corrida 1 de la red 1 con 1 capa oculta LSTM.

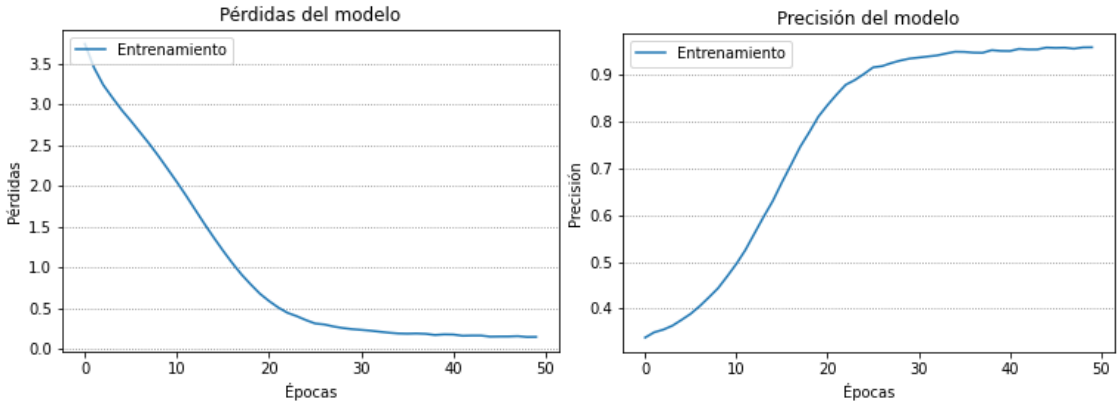
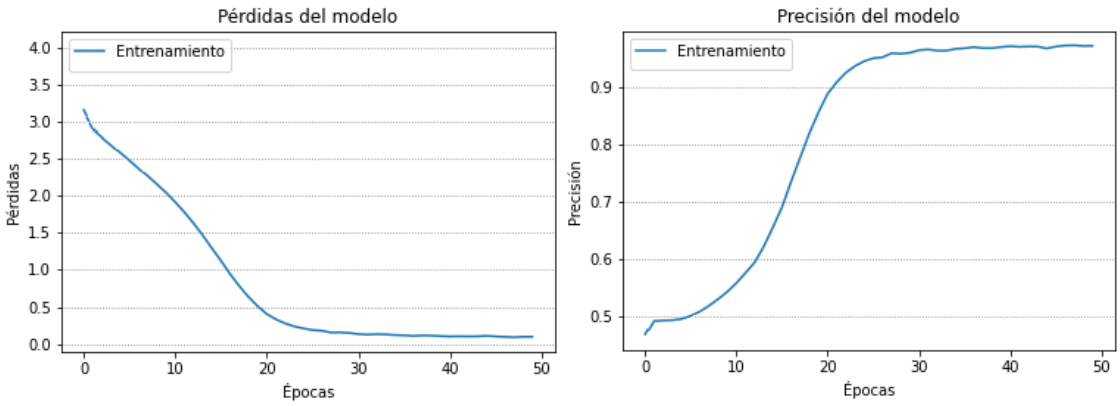


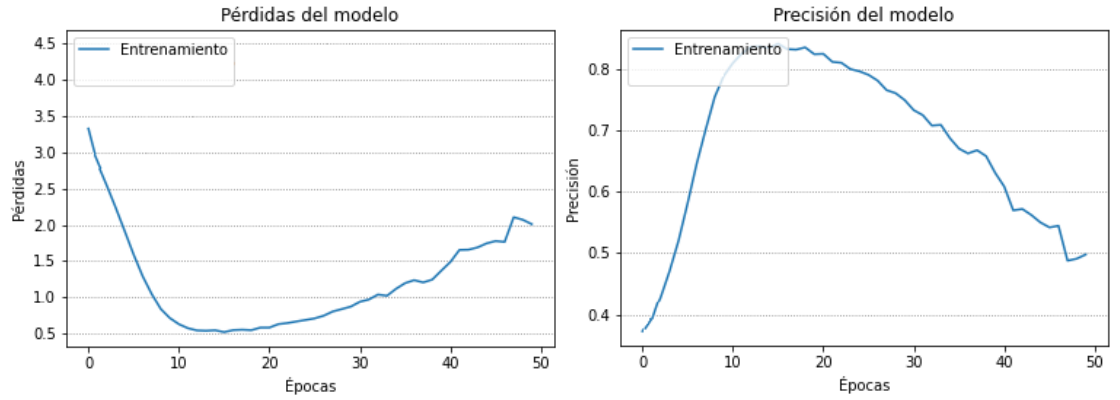
Figura 36. Pérdida y precisión de la corrida 2 de la red 1 con 3 capas ocultas LSTM.



a) b)

Figura 37. Pérdida y precisión de la corrida 3 de la red 1 con 3 capas ocultas LSTM.

En cambio, la Figura 38, correspondiente a la implementación de celdas GRU, muestra un menor desempeño al lograr valores de pérdida mayores y de precisión menores, además de que estos valores lo logran en épocas tempranas y después comienzan a incrementar y decrementar respectivamente. En este trabajo solo se muestra una corrida correspondiente a la implementación con GRU ya que los demás experimentos muestran un comportamiento similar con la diferencia de obtener valores de pérdida mayores y de precisión menores.



a) b)

Figura 38. Pérdida y precisión de la corrida 4 de la red 1 con 3 capas ocultas GRU.

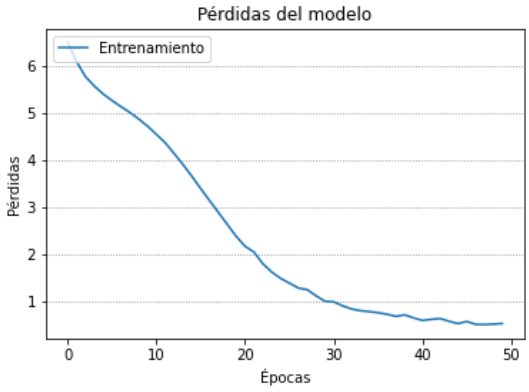
Para la red del modelo 2, los resultados correspondientes a las corridas con las mimas características se muestran en la Tabla 3. A diferencia de la red 1, en este caso se muestra la precisión obtenida en el entrenamiento para ambas salidas, tonos y sus duraciones.

Tabla 3. Valores de pérdida y precisión en el entrenamiento de la red del modelo 2.

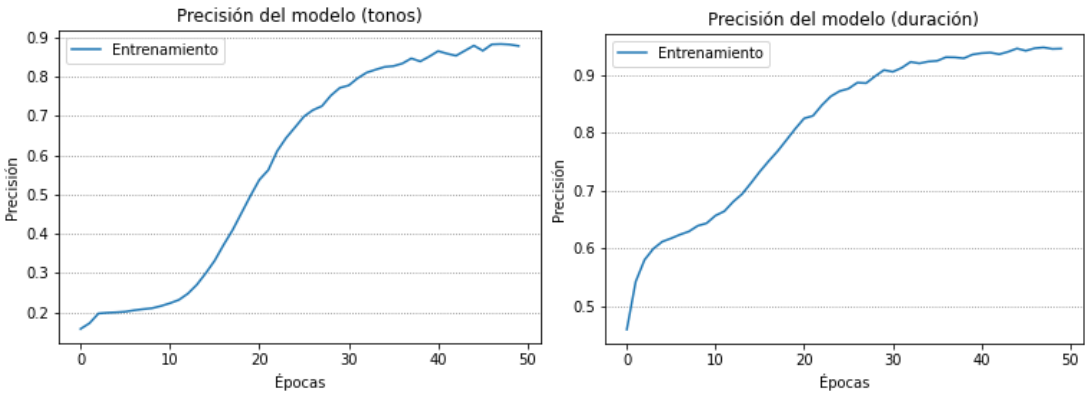
Corridas	**Pérdida**	**Precisión Tonos**	**Precisión Duraciones**
1	0.760	0.827	0.923
2	0.548	0.847	0.945
3	0.780	0.825	0.933
4	1.383	0.722	0.828

Los resultados obtenidos muestran que la red con celdas ocultas LSTM logra un mejor desempeño en el entrenamiento, logrando pérdidas más bajas y precisiones más altas que las celdas GRU.

Las Figura 39, Figura 40 y la Figura 41 muestran el comportamiento del entrenamiento de la red con celdas LSTM, en estas pruebas se alcanzan valores de precisión altos pero un poco menores que los obtenidos con la red del modelo 1, así como pérdidas menores que esta red en las mismas épocas de entrenamiento.

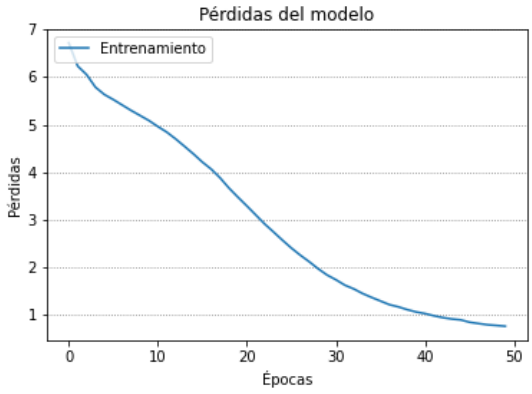


a)

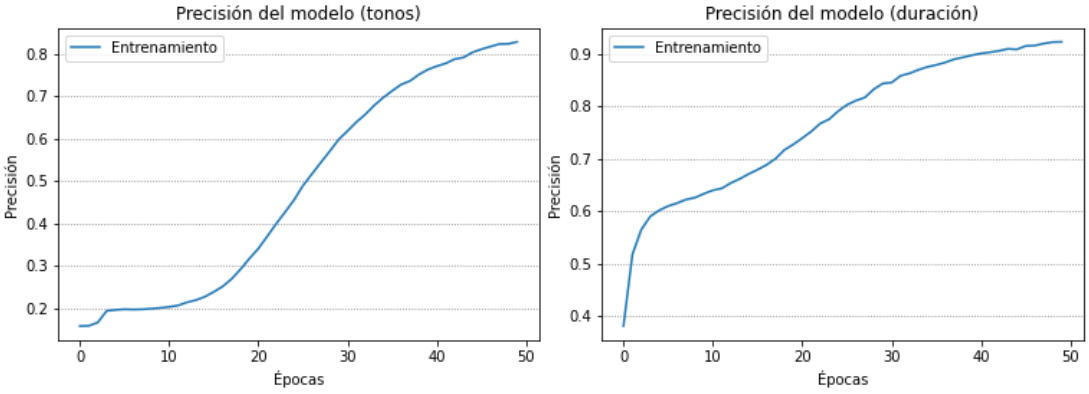


b) c)

Figura 39. Pérdida y precisión de la corrida 1 de la red 2 con 1 capas ocultas LSTM.

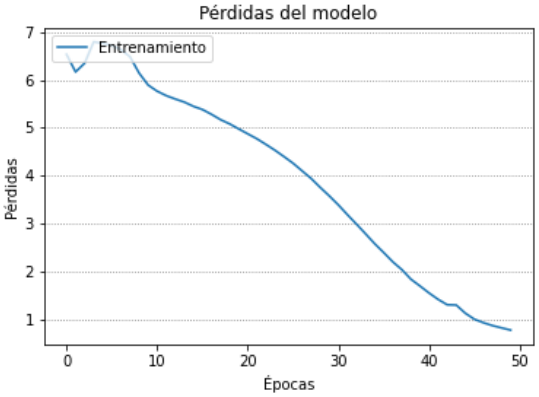


a)

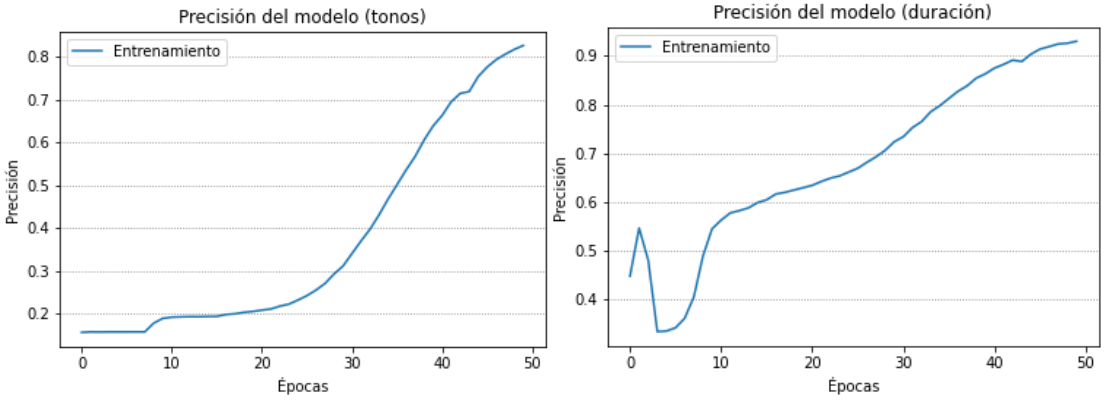


b) c)

Figura 40. Pérdida y precisión de la corrida 2 de la red 2 con 3 capas ocultas LSTM.



a)



b) c)

Figura 41. Pérdida y precisión de la corrida 3 de la red 2 con 3 capas ocultas LSTM.

En cuando a la red implementada con celdas ocultas GRU, cuyo entrenamiento se muestra en la Figura 42, no logra los valores obtenidos que con las celdas LSTM en las 50 épocas de entrenamiento. Para esta red, las gráficas muestran un entrenamiento más estable en comparación que en la red del modelo 1 con celdas GRU ya que los valores muestran una tendencia incremental conforme avanzan las épocas del entrenamiento, mientas que la red del modelo 1 muestra un decremento en las primeras épocas y continúan disminuyendo.

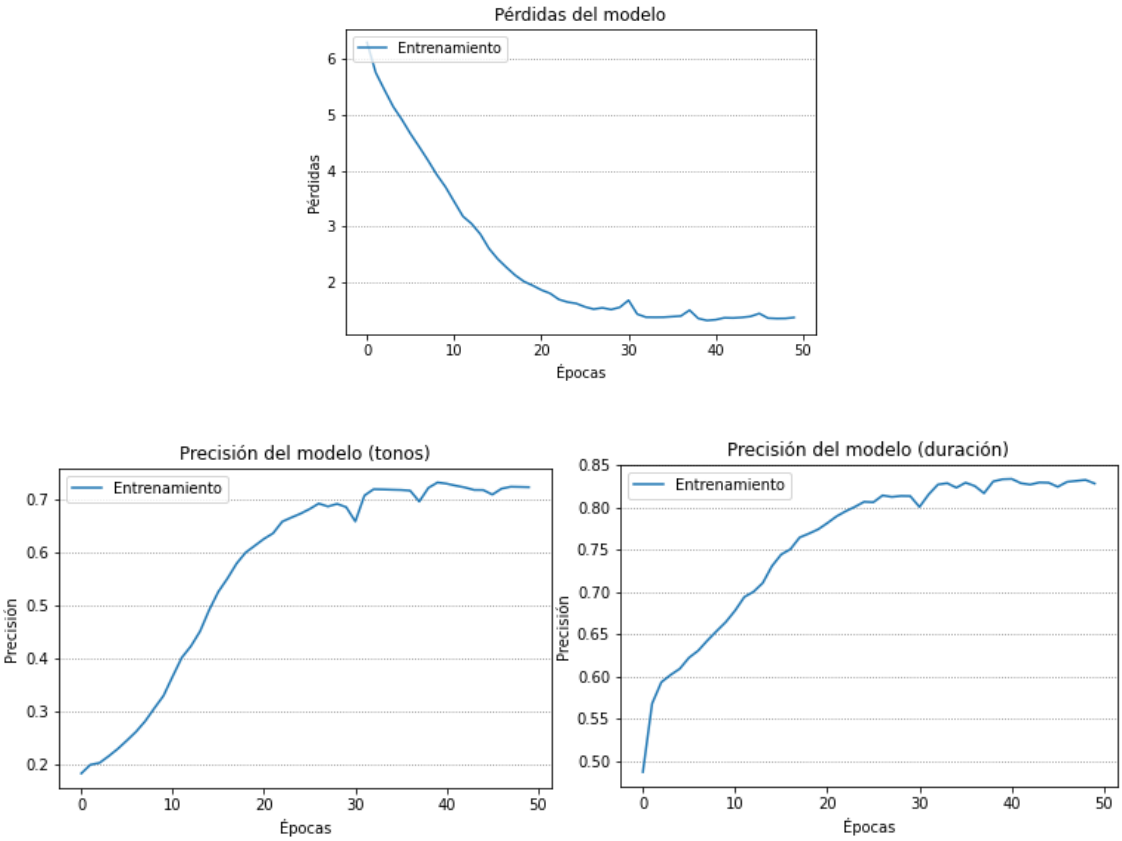


Figura 42. Pérdida y precisión de la corrida 4 de la red 2 con 3 capas ocultas GRU.

4.4. Discusión de melodías generadas

Los distintos experimentos realizados muestran que en el modelo 1 con celdas LSTM en su estructura logran un mejor desempeño que con GRU, teniendo valores de pérdida y precisión mayores que los logrados con el uso de celdas GRU. Uno de los hiperparámetros más sensibles en el modelo 1 es la cantidad de neuronas en cada capa oculta, mostrando un peor desempeño con el menor número explorado.

Los modelos propuestos han sido entrenados con el conjunto de melodías mencionado y ha logrado aprender una estructura de notas, silencios y acordes del género blues. Debido a esto, en las melodías generadas se puede encontrar una tendencia a mezclar fragmentos del set de melodías, pero logrando una consonancia del estilo aprendido. Dentro de las melodías generadas pueden encontrarse fragmentos disonantes, aunque no afectan a la sonoridad de

cada nota o acorde ya que a cada uno de estos elementos se le da sentido auditivo por el contexto en el que se encuentran, logrando momentos de tensión en la armonización de la melodía, el cual es un elemento musical muy usado.

Al unir las melodías generadas, principal y acompañamiento, se logra crear una armonía durante varios compases y, en ocasiones, el acompañamiento logra establecer el rimo de la melodía, creando un paseo más armónico en la melodía. Dicho esto, las melodías generadas no se pueden considerar como una un producto terminado, aunque tampoco parecen una simple elección aleatoria del vocabulario aprendido. Si bien los resultados muestran que una melodía generada no puede ser tomada como una nueva canción, podría ser una herramienta muy útil en el proceso de composición de profesionales siendo una fuente parcial de inspiración en los fragmentos de mayor armonía. La Figura 43 muestra un ejemplo de la melodía generada, tanto el acompañamiento como la melodía principal.

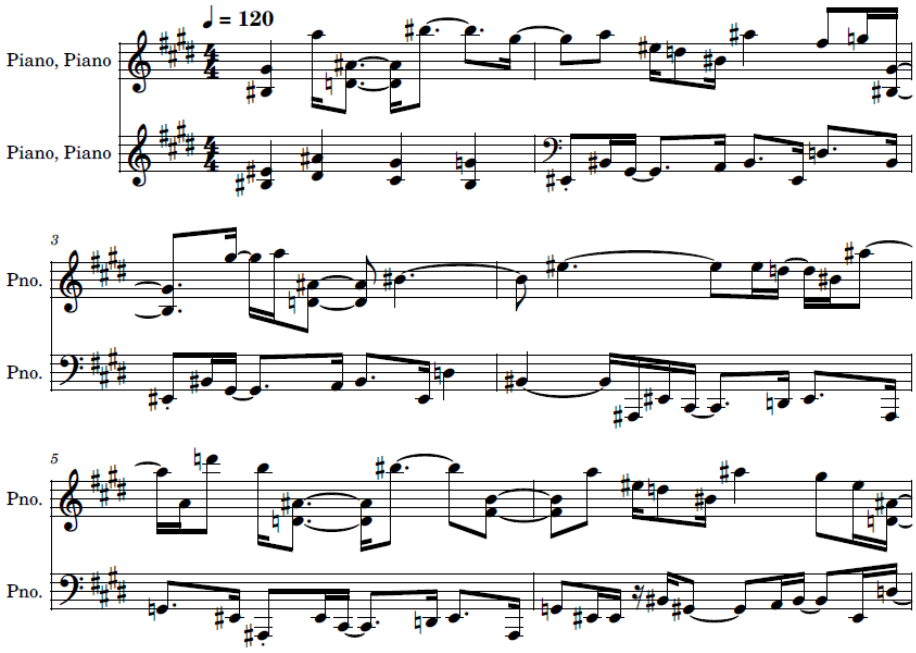


Figura 43. Fragmento de la melodía generada por el modelo 1.

En cuanto a las melodías generadas por la segunda red, la Figura 44 muestra un fragmento. En este caso, las melodías generadas no logran una replicación del ritmo del set entrenado ya que, en su mayoría, se observa que la duración de cada elemento de la partitura es el mismo, una negra. De igual forma, estas melodías sueles generar bucles muy respetivos.

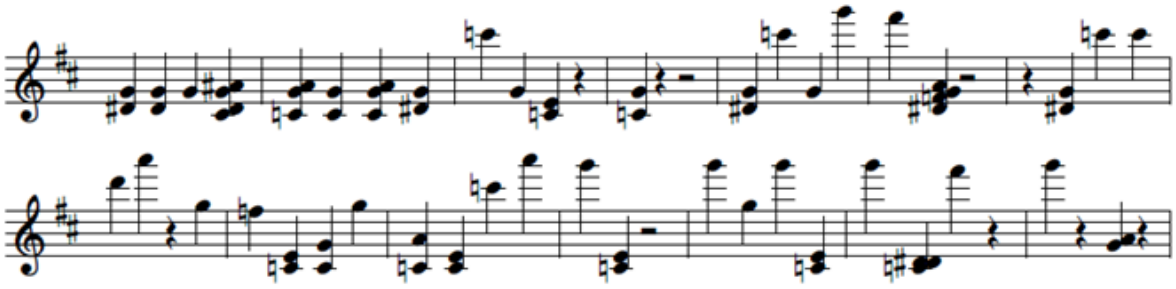
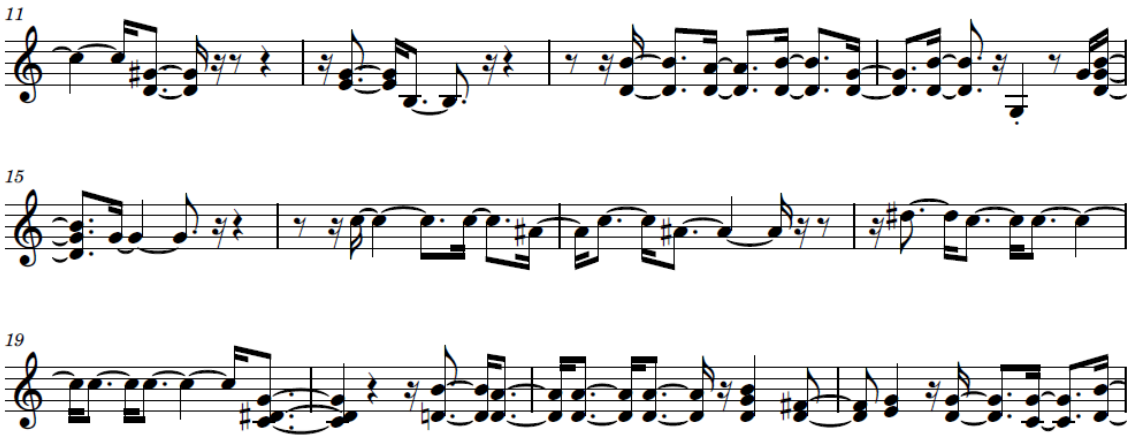
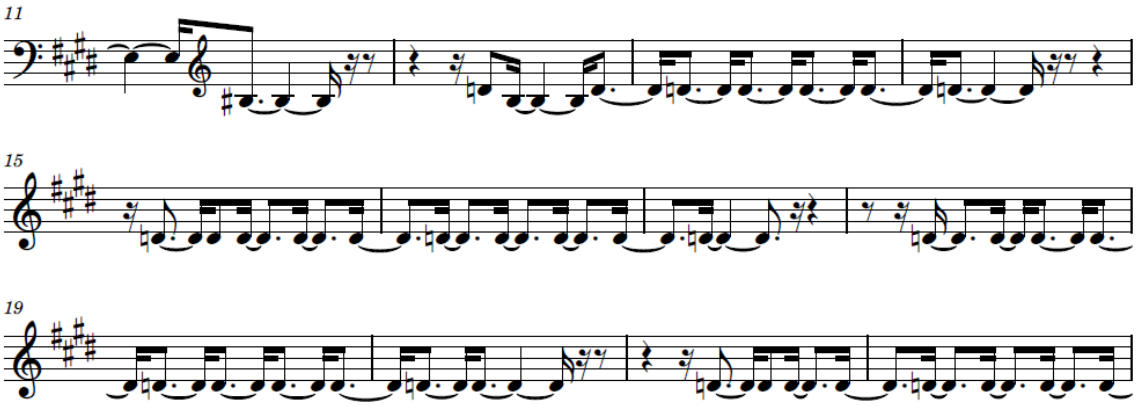


Figura 44. Fragmento de la melodía generada por el modelo 2.

En cuanto a la diferencia que se obtiene al usar una decodificación basada en el muestreo de temperatura comparado con una búsqueda codiciosa, se observa que la reasignación de las probabilidades en cada paso de tiempo mediante el muestreo de temperatura da como resultado melodías más variadas y que mantienen una coherencia., mientras que una técnica como búsqueda codiciosa genera melodías que crean un bucle repetitivo que mantienen por largos periodos de tiempo. La Figura 45 muestra un ejemplo de la diferencia de aplicar ambos tipos de muestreo, la Figura 45a) muestra el resultado de aplicar el muestreo de temperatura, con una temperatura $t = 0.65$, mientras que la Figura 45b) muestra el resultado de aplicar el muestreo de búsqueda codiciosa, se observa que esta segunda técnica crea un bucle a partir del compás 12 del que no logra salir, mientras que la primera técnica logra una mayor diversidad en las notas y acordes generados.



a)



b)

Figura 45. Diferencia de melodía generada entre muestreo de temperatura y búsqueda codiciosa.

Conclusiones

En este trabajo se presentaron dos modelos generativos de contenido musical basado en redes profundas y se comparó su desempeño. Para ello, como primer paso, se realizó una investigación en el estado del arte, con lo que se obtuvo que, con una gran frecuencia, se recurre al uso de una representación simbólica de las melodías y el uso de redes profundas basadas en redes neuronales recurrente debido a que la información musical se puede tratar como un conjunto de elementos organizados en el tiempo y este tipo de información es adecuado para estas redes. Por tal motivo se estableció desarrollar dos modelos generativos de música en los cuales se manipulan los mismos conceptos musicales en tonos y ritmo, pero representados de maneras distintas, de igual forma se proponen dos arquitecturas de RNA distintas para cada caso. Con el fin de evaluar el desempeño para cada red profunda se realizan variaciones en los hiperparámetros de control y de estructura.

Dentro de los hiperparámetros explorados y que más influían en el desempeño de ambas redes fueron: la cantidad de capas recurrentes, la cantidad de unidades por capa oculta y el tipo de capas recurrente, siendo las celdas LSTM y GRU las exploradas. Se observó que las celdas LSTM tuvieron un mejor desempeño, logrando valores de pérdida menores y valores de precisión mayores que las obtenidas con las celdas GRU. Se evaluaron ambos modelos tomando en cuenta su desempeño en el entrenamiento de las redes a partir de la pérdida y precisión, así como la armonía que las melodías generadas lograban y qué tanto lograban replicar el ritmo del estilo musical del set de entrenamiento. En cuando al primer modelo, las melodías generadas lograron una mejor consonancia con el estilo musical, replicando el ritmo y alcanzando una armonía en las pistas polifónicas de una voz. En cuanto a la influencia de la implementación de una técnica de muestreo en la decodificación en las melodías generadas, se observó que técnicas diferentes a la búsqueda codiciosa, como el muestreo de temperatura, logran melodías más diversas y evita crear bucles muy repetitivos de los cuales la estrategia de generación feedforward iterativo no logra salir.

Trabajo a futuro

Como se mencionó en este trabajo, la generación de contenido artístico mediante técnicas computaciones es un área que presenta distintos retos, desde la representación de la información, hasta la evaluación de los resultados. Por tal motivo, como trabajo a futuro se presentan algunas recomendaciones para continuar esta línea de investigación.

Como primera consideración se puede mencionar el proponer una forma de representar algunos elementos importantes de las partituras que se omitieron en este trabajo, como las ligaduras y algunos efectos como los glisando.

Además, se plantea continuar con experimentos adicionales en donde el conjunto de entrenamiento sean otros géneros musicales distintos, siendo algunos de los más explorados como la música clásica y uno de los menos explorados la música pop.

Finalmente, ya que en la música se pueden hacer algunas analogías con el lenguaje natural por su naturaleza de relación en el tiempo, se propone la posibilidad de explorar el rendimiento de utilizar arquitecturas de RNA más complejas como los transformes.

Referencias

[1] J.-P. Briot, G. Hadjeres, and P. François-David, *Deep learning techniques for music generation-a survey*. Springer, 2020.

[2] D. Eck and J. Schmidhuber, "A first look at music composition using lstm recurrent neural networks," *Ist. Dalle Molle Di Stud. Sull Intelligenza Artif.*, vol. 103, 2002.

[3] H. Chu, R. Urtasun, and S. Fidler, "Song from PI: A musically plausible network for pop music generation," *arXiv Prepr. arXiv1611.03477*, 2016.

[4] F. Liang, M. Gotham, M. Johnson, and J. Shotton, "Automatic Stylistic Composition of Bach Chorales with Deep LSTM," *ISMIR*, pp. 449–456, 2017.

[5] G. Hadjeres, F. Pachet, and F. Nielsen, "DeepBach: a Steerable Model for Bach Chorales Generation," *Int. Conf. Mach. Learn. PMLR*, pp. 1362–1371, 2017.

[6] F. Colombo, A. Seeholzer, and W. Gerstner, "Deep artificial composer: A creative neural network model for automated melody generation," *Int. Conf. Evol. Biol. Inspired Music Art*, no. Springer, Cham, pp. 81–96, 2017.

[7] H. Zhu, "Xiaoice band: A melody and arrangement generation framework for pop music," *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 2837–2846, 2018.

[8] A. Roberts, "A hierarchical latent vector model for learning long-term structure in music," *Int. Conf. Mach. Learn.*, no. PMLR, pp. 4364–4373, 2018.

[9] A.-I. Marinescu, "Bach 2.0-generating classical music using recurrent neural networks," *Procedia Comput. Sci.*, vol. 159, pp. 117–124, 2019.

[10] J. Wua, X. Liu, X. Hua, and J. Zhu, "PopMNet: Generating structured pop music melodies using neural networks," *Artif. Intell.*, vol. 286, 2019.

[11] A. Pal, S. Saha, and R. Anita, "Musenet : Music Generation using Abstractive and Generative Methods," *Int. J. Innov. Technol. Explor. Eng.*, vol. 9, 2020.

[12] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A generative model for music," *arXiv Prepr. arXiv2005.00341*, 2020.

[13] S. Raschka and V. Mirjalili, *Python Machine Learning*, Third edit. Birmingham B3 2PB, UK: Packt Publishing Ltd., 2019.

[14] S. Skansi, *Introduction to Deep Learning From Logical Calculus to Artificial Intelligence*. Springer, 2018.

[15] C. C. Aggarwal, *Neural Networks and Deep Learning*. Springer, 2018.

[16] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed. O'Reilly Media, 219AD.

[17] M. Association, "MIDI Specifications." 2021, [Online]. Available: https://www.midi.org/specifications.

[18] "music21: a Toolkit for Computer-Aided Musicology." Michal Scott Cuthbert and cuthbertLab, 2021, [Online]. Available: https://web.mit.edu/music21/.

[19] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, "The curious case of neural text degeneration," *arXiv Prepr. arXiv1904.09751*, 2019.

[20] D. Jorge Matich, "Redes Neuronales: Conceptos Básicos y Aplicaciones." Universidad Tecnológica Nacional – Facultad Regional Rosario Departamento de Ingeniería Química, 2001.

[21] L. B. Arrestegui, "Fundamentos históricos y filosóficos de la inteligencia artificia," *Rev. Investig. y Cult.*, vol. 1, pp. 87–92, 2012.

[22] H. L. Rufiner, "¿Puede la Inteligencia artificial dar una 'mente' a las máquinas," *Inst. Filos. Univ. Austral, Buenos Aires*, 2018, [Online]. Available: https://sinc.unl.edu.ar/sinc-publications/2018/Ruf18/sinc_Ruf18.pdf.

[23] X. Yao, J. Zhou, J. Zhang, and C. R. Boër, "Intelligent Manufacturing to Smart Manufacturing for Industry 4.0 Driven by Next Generation Artificial Intelligence and Further On," *5th Int. Conf. Enterp. Syst. IEEE*, pp. 311–318, 2017.

[24] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[25] E. Gutiérrez Gonzáles and O. Vladimirovna Panteleeva, *Probabilidad y Estadistica. Aplicaciones a la ingenieria y ciencias*, 4th ed. Ciudad de México: Grupo Editorial Patria, 2019.

[26] P. D. McNelis, *Neural Networks in Finance: Gaining Predictive Edge in the Market*. Academic Press, 2005.

[27] E. Serrano, Antonio José, Soria Olivas and J. D. Martín, "Redes neuronales artificiales." Curso 2009-2010, 2010.

[28] F. Izaurieta and C. Saveedra, "Redes neuronales artificiales," *Dep. Física, Univ. Concepción*, 2000.

[29] A. Fan, M. Lewis, and Y. Dauphin, "Hierarchical Neural Story Generation," *arXiv Prepr. arXiv1805.04833*, 2018.

[30] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for Boltzmann machines.," *Cogn. Sci.*, vol. 9, pp. 147–169, 1985.

[31] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving language understanding by generative pre-training,” 2018.

Apéndices

A. Breve historia de las redes neuronales artificiales.

Las computadoras, al igual que los seres humanos, se han adaptado a la solución de diferentes tipos de problemas, desde cálculos matemáticos simples, hasta tareas de predicción y reconocimiento, donde estás ultimas han sido verdaderos retos para algoritmos de aprendizaje automatizado, mientras que para los humanos resulta una tarea común y simple. Las Redes Neuronales Artificiales (RNA) brindan distintas ventajas para tareas computacionales gracias a su gran capacidad de recrear modelos no lineales. Para tener una mejor comprensión del estado actual de las RNA, se presenta un breve recorrido sobre su historia y sus principales avances.

Las RNA fueron inspiradas en la biología del cerebro humano, el cual es una máquina muy potente para procesar grandes cantidades de información y realizar operaciones complejas en instantes, principalmente hubo interés en los elementos que la forman como lo son las neuronas. En el año 1943, Warren McCulloch, un neurofisiólogo, y Walter Pitts, un matemático, trataron de comprender el funcionamiento del cerebro humano y publicaron el primer concepto de una célula cerebral (neurona) simplificada llamada neurona McCulloch-Pitts (MCP). Ellos modelaron una red neuronal simple mediante circuitos electrónicos [13], [20] y la inteligencia artificial tomó una dirección característica de la lógica filosófica "¿se podría modelar el pensamiento como un proceso mental humano con reglas lógicas?" [14] agregando algunas definiciones que hoy se dan por sentadas e introduciendo la idea de la Red Neuronal Artificial. Una neurona es una célula nerviosa interconectada con otras neuronas en el cerebro, en las cuales se transmiten señales de entrada y salida mediante impulsos eléctricos, la Figura 46 ilustra la estructura básica de una neurona biológica.

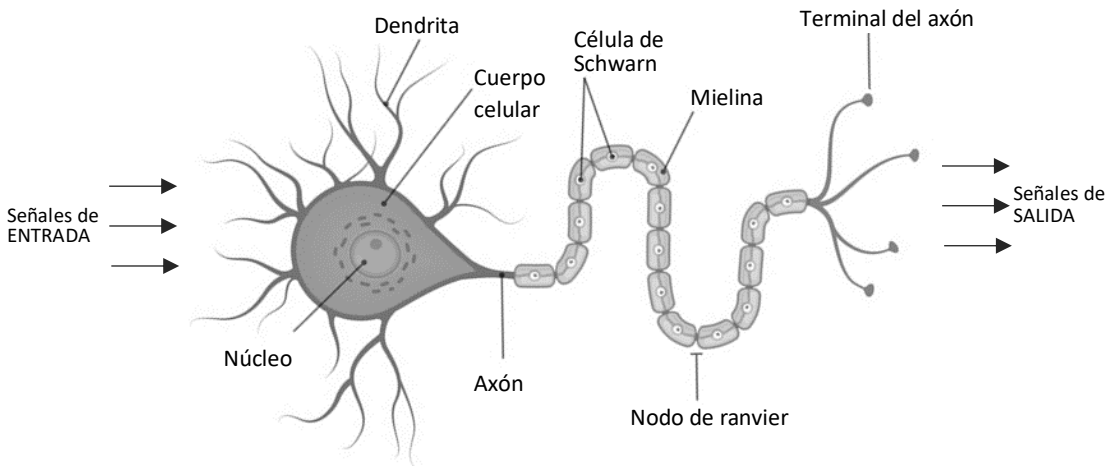


Figura 46. Estructura básica de una neurona biológica[18].

En el año de 1957 Frank Rosenblatt realizó una enorme contribución al introducir el perceptrón y su regla de aprendizaje [14]. Este modelo desarrollado era capaz de realizar generalizaciones reconociendo patrones que no se le mostraron en el entrenamiento [20]. Este modelo presentaba algunas limitaciones importantes que se demostrarían más de una década después. Dos años después, en 1959, Frank Rosenblatt confirmó que el modelo del perceptrón convergía hacia un estado finito bajo ciertas condiciones, es decir, en una tarea de diferenciar dos clases linealmente separables, el perceptrón aprenderá a clasificar en un numero finito de iteraciones de entrenamiento, conocido como Teorema de Convergencia del Perceptrón.

En 1969 ocurre otro acontecimiento importante en el campo de la inteligencia artificial, este acontecimiento no es necesariamente positivo ya que fue un golpe que casi acaba con las RNA. Esta ocasión los protagonistas fueron Marvin Minsky y Seymour Papert en su libro *Perceptrons: An Introduction to Computational Geometry,* en el cual demostraron, mediante la operación lógica XOR, que el modelo del perceptrón no era capaz de resolver problemas no lineales, resultando ser solo un clasificador lineal [14]. El trabajo *Perceptrons* provoca el primer invierno de la IA, que consistió en una falta de interés y de financiación para la investigación de las RNA [21], [22], abarcó el periodo de 1969 a finales de la década de los setenta.

Después del gran golpe en el área de investigación de las RNA, en 1975 sucedió un primer paso para su avance. Hasta este momento, se conocía cómo entrenar una red neuronal artificial de una sola capa y también se sabía que, con aumentar una sola capa oculta a la red,

[18] Parcialmente tomada de: <https://www.cognifit.com/es/neuronas>.

aumentaba enormemente sus capacidades. El gran desafío que nadie había logrado era entrenar una red con más de una capa, así fue como en este año, Paul Werbos, desarrolló la idea básica del algoritmo de Aprendizaje de Propagación Hacia Atrás (*backpropagation*) [20]. Este economista descubrió una forma de propagar el error a través de una capa oculta, pero su trabajo pasó desapercibido durante una década y fue hasta el año de 1985 cuando David Parker publicó formalmente lo descubierto sobre *backpropagation*. Otras personas como Yann LeCun, Rumelhart, Hinton y Williams también descubrieron *backpropagation* [14]. Posterior a estos sucesos, ocurrió el segundo invierno de la inteligencia artificial de 1987 a 1993, debido a que muchas empresas se quedaron en el camino al no cumplir sus promesas extravagantes, como el colapso del mercado de máquinas Lisp en 1987, o el proyecto de quinta generación de Japón que no logró sus objetivos [21], [23].

Después de la formalización de *backpropagation*, las RNA avanzaron sin ningún evento sobresaliente hasta principios de la década de los 90, debido al mayor interés en las Máquinas de Vectores de Soporte (SVM: *Support Vector Machines*), que a diferencia de las RNA, estaban bien fundamentadas matemáticamente [14]. Fue hasta finales de esta década cuando se lograron importantes avances en las RNA. En 1997 las Redes Neuronales Recurrentes (RNN: *Recurrent Neural Network*) supervisadas de Memoria a Largo y Corto Plazo (LSTM: *Supervised Long Short-Term Memory*) fueron creadas por Hochreiter [24], las cuales presentan un gran desempeño en el tipo de información de series temporales, también estas arquitecturas siguen siendo unas de las más usadas y más características del Aprendizaje Profundo (DL: *Deep Learning*). Posteriormente en 1998, LeCun, Bottou, Bengio y Haffner crearon la primera Red Neuronal Convolucional (CNN: *Convolutional Neural Network*) llamada LeNet-5 [14], cuyo propósito es el procesamiento de imágenes y la detección de características en estas.

Como último evento a resaltar en este resumido recorrido de la historia de las RNA, se encuentra el ocurrido en el año 2006 con el artículo de Hinton, Osindero y Teh, quienes introdujeron Redes de Creencias Profundas (DMB: *Deep Belief Networks*), la cual es un modelo gráfico que se basa en redes simples y no supervisadas y su desempeño lo logran al extraer características de la información de entrada y reconstruirlas. La DMB logró mejores resultados en el conjunto de datos MNIST que otras técnicas hasta ese momento [14], además, Geoffrey Hinton acuña el término “Deep Learning” que le ayuda a explicar nuevas arquitecturas con mejor desempeño. Este término hace referencia arquitecturas donde se implementan varias capas donde cada una procesa distintos niveles de abstracción de la información.

Hasta el momento se han mencionado aportaciones importantes en el área de las RNA, pero cabe mencionar que no es el objetivo de esta sección mencionar todos los avances, sino aquellos que marcaron un salto sobresaliente dentro de la historia de las RNA. Otro punto a resaltar en la historia de las RNA son los dos periodos de menor actividad en el área de investigación y aplicación, conocidos como los dos inviernos de la inteligencia artificial, donde cabe destacar que los periodos que abarcan cada invierno pueden variar en distintos autores, sin embargo, la mayoría concuerda en las causas que los provocan.

B. Conceptos básicos de redes neuronales artificiales.

En este apéndice se aborda la descripción general de lo que es una neurona artificial hasta una Red Neuronal Multicapa, mejor conocida como Perceptrón Multicapa. Para ello, primero se abordarán algunos conceptos necesarios para comprender la estructura de una RNA y, en secciones posteriores, se describirán algunas arquitecturas particulares de RNA.

B.1. Conceptos preliminares

Los conceptos abordados en esta subsección están ampliamente basados en [13], [25] y [26].

B.1.1. Regresión lineal

El primer concepto para poder abordar las RNA es la regresión lineal. La regresión lineal es una técnica de modelado estadístico, utilizada para la estimación de la relación lineal de una variable de respuesta continua como una función de una o varias variables predictoras [25]. Haciendo referencia a las variables predictoras, la regresión lineal puede dividirse en dos casos dependiendo de la cantidad de variables: si se cuenta con una sola variable predictora, la técnica se conoce como regresión lineal simple, pero si se cuenta con dos o más variables predictoras, la técnica se conoce como regresión lineal múltiple.

Tomando como ejemplo inicial la regresión lineal simple, la técnica de regresión lineal consiste en que, dados una serie de valores correspondientes a la variable de respuesta, y, y de las variables predictoras, x, que se pueden graficar como un diagrama de dispersión y se debe encontrar la recta que mejor se ajuste o que mejor represente la relación entre y y x. La Figura 47 representa una serie de valores de y y x con distintas rectas candidatas a representar la relación de los datos.

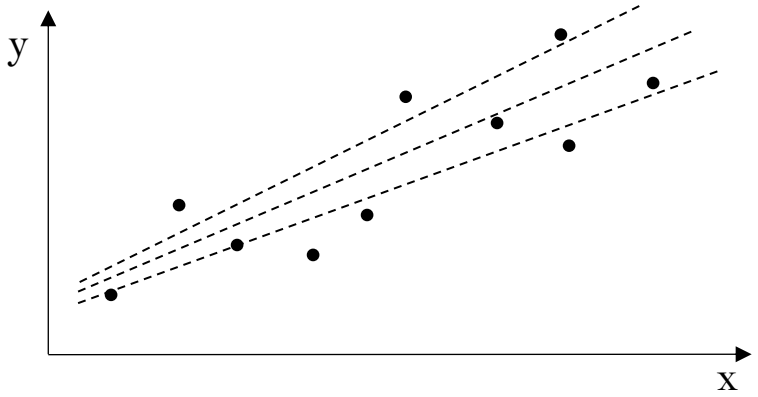


Figura 47. Ejemplo de rectas de regresión lineal simple.

La ecuación del modelo de regresión lineal simple es:

$$y = w_0 + w_1 x_1 \tag{15}$$

Donde w_0 es la ordenada al origen y w_1 la pendiente de la recta. Esta recta, al representar a los datos reales, sirve para realizar predicciones sobre la variable de respuesta a partir de la variable predictora.

Cuando se habla de regresión lineal múltiple, se puede generalizar el modelo a más de una variable x de la siguiente manera:

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + \cdots + w_n x_n \tag{16}$$

En este caso, ya no se habla de rectas que se ajustan a los datos, sino de planos en caso de dos variables o hiperplanos en espacios multidimensionales para más variables.

Hasta ahora se ha mencionado solo una variable de respuesta, pero se puede generalizar aún más incluyendo más de una variable de respuesta, en tal caso, se puede tener un sistema de ecuaciones como las siguientes.

$$\begin{aligned}
y_1 &= w_0 + w_1 x_{11} + w_2 x_{12} + w_3 x_{13} + \cdots + w_n x_{1n} \\
y_2 &= w_0 + w_1 x_{21} + w_2 x_{22} + w_3 x_{23} + \cdots + w_n x_{2n} \\
y_3 &= w_0 + w_1 x_{31} + w_2 x_{32} + w_3 x_{33} + \cdots + w_n x_{3n} \\
y_4 &= w_0 + w_1 x_{41} + w_2 x_{42} + w_3 x_{43} + \cdots + w_n x_{4n} \\
&\cdots \\
y_m &= w_0 + w_1 x_{m1} + w_2 x_{m2} + w_3 x_{m3} + \cdots + w_n x_{mn}
\end{aligned} \tag{17}$$

Donde los valores w se comparten para cada ecuación. Una forma más práctica de representar estos datos es en forma vectorial.

$$X = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m1} & \cdots & x_{mn} \end{bmatrix} \tag{18}$$

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \tag{19}$$

$$W = [w_0 \quad w_1 \quad \cdots \quad w_n] \tag{20}$$

Donde X representan las variables predictoras y cada columna representa una característica de los datos de entrada y cada fila representa una medición de estos datos. Y representa las variables respuesta que se van a modelar. Finalmente, W representa los parámetros o pesos de cada variable x. De las Ecuaciones (18), (19) y (20) se puede generalizar el modelo de regresión lineal con la siguiente ecuación:

$$Y = WX \tag{21}$$

Como ya se mencionó, esta técnica se basa en encontrar la recta, plano o hiperplano que mejor se ajusta a nuestros datos. Existen diferentes estrategias para hacerlo, y se basan en encontrar los valores de la matriz W. En las subsecciones siguientes se presentan dos métodos diferentes.

B.1.2. Mínimos cuadrados ordinarios

Mínimos cuadrados ordinarios es un método de optimización que permite obtener los valores de los coeficientes o parámetros W, y por lo tanto las curvas que más se ajustan a los datos [25]. Se basa en encontrar el error mínimo entre la recta y los datos. En el caso de regresión lineal simple, el error consiste en la diferencia entre el valor real, y_r, de los datos y el valor predicho, y_p, por el modelo. La Figura 48 muestra el error entre la recta y cada uno de los valores reales.

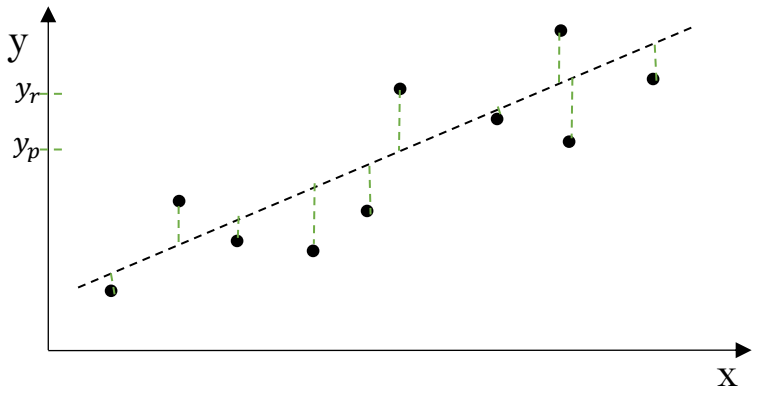


Figura 48. Diferencia entre valores reales y predichos.

En particular, el método de mínimos cuadrados ordinarios trabaja sobre el error cuadrático medio que se define de la siguiente manera:

$$ECM = \frac{1}{n}\sum_{i=1}^{n}(y_{r\,i} - y_{p\,i})^2 \tag{22}$$

La Ecuación (22) representa el error para un caso de regresión lineal simple. Para un caso de regresión lineal múltiple, el error vectorial está dado por la siguiente expresión.

$$ECMV = (Y - WX)^T(Y - WX) \tag{23}$$

Desarrollando la ecuación (23) para encontrar el mínimo error, se obtiene que el mínimo error cuadrático medio está dado por la siguiente ecuación.

$$W = (X^T X)^{-1}(X^T Y) \tag{24}$$

De esta forma, con la ecuación (24), se puede obtener la curva, plano o hiperplano que mejor se ajuste a los datos.

B.1.3. Descenso del gradiente

El segundo método que se puede aplicar a una regresión lineal para encontrar la mejor recta que se ajuste a los datos es el conocido como descenso del gradiente. Este método no solo se aplica a una regresión lineal, sino que es uno de los métodos de optimización más usados en la estimación no lineal [26], y una de las técnicas claves en los algoritmos de aprendizaje automatizado, como ocurre en el funcionamiento de las RNA [13].

Primero se debe ingresar un nuevo concepto, la función de coste, la cual estima el error entre el valor real y el valor estimado por el modelo. En la subsección anterior, la función de coste fue, aunque no se mencionara explícitamente, el error cuadrático medio. Una función de coste, para un caso simple de una sola variable, puede representarse mediante una curva en un plano. Esta curva puede contener puntos mínimos locales y puntos mininos globales, los cuales son de interés para minimizar el error de las predicciones.

Para encontrar el mínimo de una función se puede emplear la derivada de una función. Al derivar la función de coste se obtiene la pendiente. La Figura 49 ilustra una función de coste y su derivada en un punto.

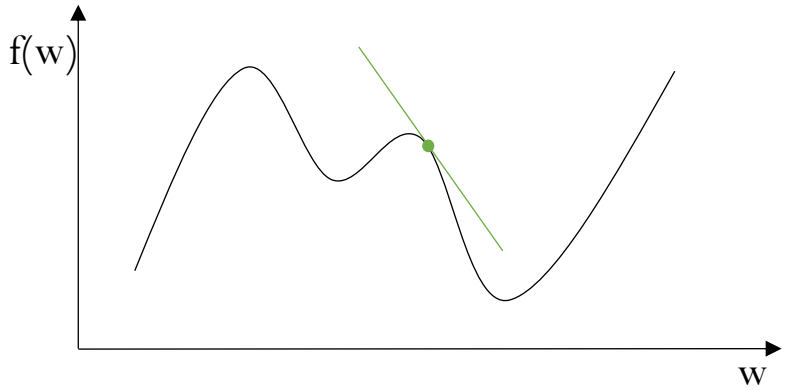


Figura 49. Función de coste y su derivada.

Donde $f(W)$ es la función de coste que depende de los pesos w_ia ajustar para minimizar el error. Por lo tanto, para encontrar un mínimo de la función de coste se puede igualar la derivada de la función a cero, $f'(W) = 0$. Esto podría tener distintas complicaciones, ya que al igualar la derivada a cero se podría estar encontrando, además de mininos globales, mininos locales, máximos globales, máximos locales o puntos de inflexión. En el caso de la función de coste a base del error cuadrático medio, se tiene una función convexa y por lo tanto tiene solo un valor mínimo.

Si la función objetivo cuenta con más de una variable, ya no se habla de la derivada de una función, sino de las derivadas parciales de la función. Al obtener las derivadas parciales de la función de coste, se obtiene un vector que indica la dirección en la que la pendiente crece con mayor rapidez, mejor conocido como el gradiente de la función. La ecuación (25) muestra el gradiente de la función de coste $f(W)$.

$$\nabla f(W) = \begin{bmatrix} \frac{\partial f(W)}{\partial w_1} \\ \vdots \\ \frac{\partial f(W)}{\partial w_n} \end{bmatrix} \tag{25}$$

Ya que el gradiente da la dirección de crecimiento de la función, basta con multiplicarlo por menos uno para obtener la dirección de decremento. Con el gradiente, ahora se pueden actualizar los pesos dando un paso en la dirección de decremento, la cuestión es cuál es la longitud del paso para actualizar los valores w_i. La actualización de los pesos está dada por la siguiente ecuación [26].

$$W \coloneqq W - \alpha \nabla f(W) \tag{26}$$

Donde α es la tasa de aprendizaje de descenso del gradiente. Entonces, como se mencionó, el descenso del gradiente es un método iterativo que puede expresarse de la siguiente manera.

```
repetir hasta alcanzar la convergencia {
      W:=W-α ∇f(W)
}
```
(27)

B.2. ¿Qué es una red neuronal artificial?

Una red neuronal artificial, al igual que el órgano biológico por el cual fueron inspiradas e igual a muchos otros sistemas complejos, están formadas por unidades más pequeñas que, en este caso, son las neuronas artificiales. Para comprender mejor el funcionamiento de una RNA multicapa, primero se define una neurona artificial, los distintos elementos que la forman y su funcionamiento.

B.2.1. Neurona artificial

La unidad fundamental de una RNA es la neurona artificial, que, al igual que la neurona biológica mostrada en la Figura 46, consta de señales de entrada y de salida. La RNA más simple es la formada por una única neurona artificial conocida como perceptrón [15] y que también, históricamente, surgió primero. El perceptrón fue utilizado para problemas de clasificación [27] y tiene la estructura mostrada en la Figura 50, en esta el perceptrón consta de señales de entrada formadas por $X = [1, \ x_1, \ \ \dots, \ \ x_n]$ con n características, con las cuales se realiza una suma ponderada con los pesos $W = [w_0, \ \ w_1, \ \ \dots, \ \ w_n]$ donde el

valor w_0 se conoce como sesgo (bias en inglés), al resultado de esta suma ponderada, $z = W \cdot X$, se le aplica una función de activación y finalmente se obtiene el valor de salida y.

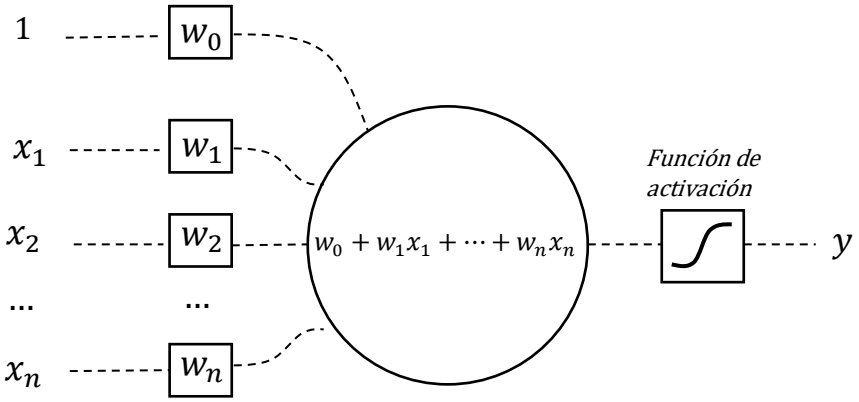


Figura 50. Arquitectura básica del perceptrón.

Como en un principio el objetivo del perceptrón fue la clasificación, se introduce la función de activación signo representada como:

$$sign(z) = \begin{cases} +1 \; si \; z \geq 0 \\ -1 \; si \; z < 0 \end{cases} \tag{28}$$

Utilizado la función signo como función de activación, $y \in \{-1, +1\}$. El valor de salida dependerá del tipo de función de activación que se utilice. Como se puede observar, exceptuando la función objetivo, el perceptrón internamente realiza un modelo de regresión lineal.

En términos generales, el valor de salida del perceptrón está dado por la siguiente expresión.

$$y = sign\{W \cdot X\} = sign\left\{\sum_{i=1}^{n} w_i x_i\right\} \tag{29}$$

Como un adelanto a las RNA multicapa, las funciones de activación se incorporan en la salida de una neurona con el objetivo de no tener modelos lineales, ya que, sin la función de activación, unir más de una neurona nos da como resultado un modelo lineal.

Cuando Rosenblatt introdujo el perceptrón en 1957, también propuso un algoritmo de optimización heurística con circuitos de hardware [15] para minimizar el error en la predicción, se trataba de una estrategia diferente a la propagación hacia atrás empleada

actualmente. Al igual que con la regresión lineal, el objetivo en el entrenamiento de un perceptrón es minimizar la función de coste o función de pérdida. Existen distintas funciones de pérdida que se pueden utilizar para estimar el error, el uso de estas funciones depende del problema que se esté atacando, si es de clasificación, predicción u otro caso.

Siguiendo con el caso de clasificación del perceptrón, se puede definir la función de pérdida como la diferencia entre el valor real y el valor predicho, $E(X) = y - \hat{y}$, donde y es el valor predicho y $\hat{y}$ es el valor real. Habiendo definido la función de pérdida, el ajuste de los pesos W se lleva a cabo mediante la regla de aprendizaje del perceptrón [14]:

1. Se inicializan los pesos W y el sesgo w_0.
2. Se elije una muestra de información de entrada X.
3. Se calcula el valor de salida $y = sign\{W \cdot X + w_0\}$.
4. Se compara el valor real $\hat{y}$ con el valor calculado y.
5. Se calcula el error $E(X)$.
6. Si existe error, se actualizan los pesos W.

La forma en que se actualizan los pesos y sesgo es con la siguiente ecuación:

$$W \Leftarrow W + \alpha E(X)X \tag{30}$$

Donde α corresponde a la tasa de aprendizaje del perceptrón. El algoritmo de entrenamiento del perceptrón puede ser el descenso de gradiente, que minimiza implícitamente el error cuadrático de las muestras X introducidas al azar en el entrenamiento [15].

El perceptrón es bueno para problemas de clasificación de datos separables linealmente. Si se toma el caso de dos variables de entrada $X = [1, \quad x_1, \quad x_2,]$ con pesos y sesgo $W = [w_0, \quad w_1, \quad w_2,]$ y función de activación signo, se puede realizar calcificaciones lineales como muestra la Figura 51.

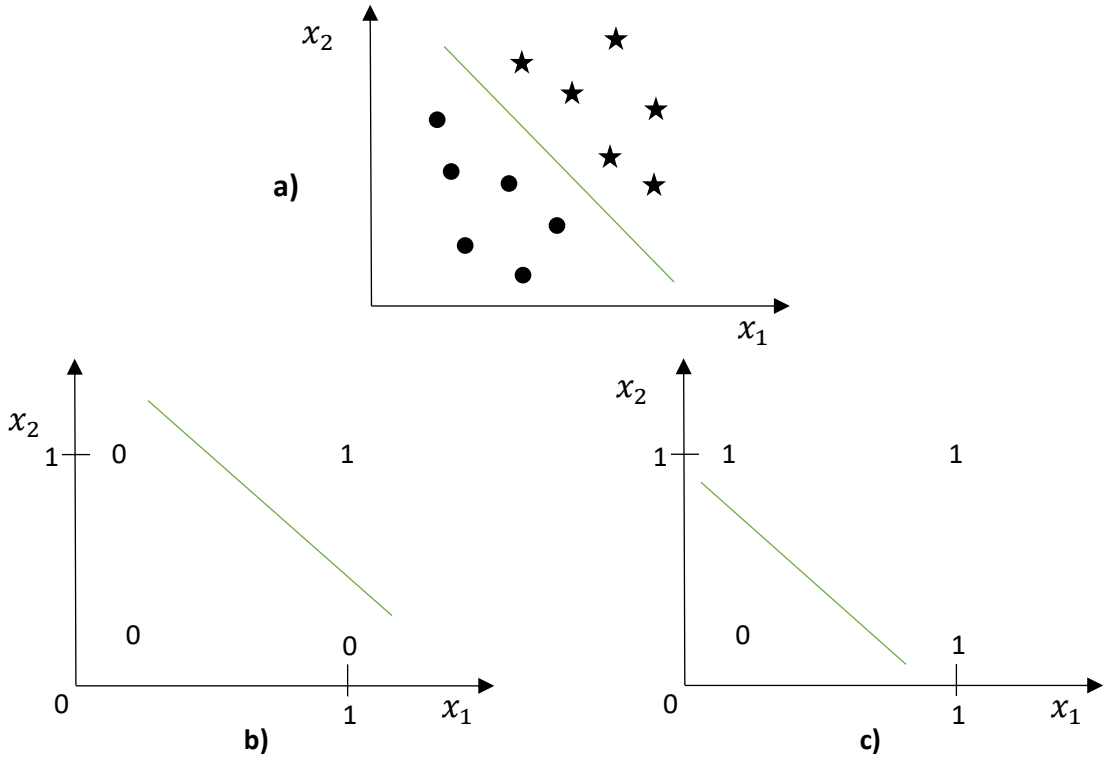


Figura 51. Ejemplos de clasificación lineal.

La Figura 51 muestra ejemplos de calificación lineal de dos variables de entrada en un perceptrón donde la Figura 51a) ejemplifica un caso donde *X* puede tomar valores continuos y en la Figura 51b) y Figura 51c), *X* puede tomar valores discretos entre cero y uno. Para estos últimos dos ejemplos, Figura 51b) y Figura 51c), el perceptrón realiza la separación de los resultados obtenidos a aplicar las operaciones lógicas AND y OR respectivamente.

Se garantiza que el algoritmo del perceptrón siempre convergerá para proporcionar un error cero para los problemas de clasificación separables linealmente [15]. Sin embargo, para problemas donde los datos no se pueden separar linealmente, el perceptrón no garantiza un resultado correcto. Tal es el caso del problema de modelar el comportamiento de la compuerta lógica XOR que surgió en el año 1969 ya mencionado anteriormente, donde el uso de una sola neurona limita la correcta clasificación de los datos como muestra la Figura 52.

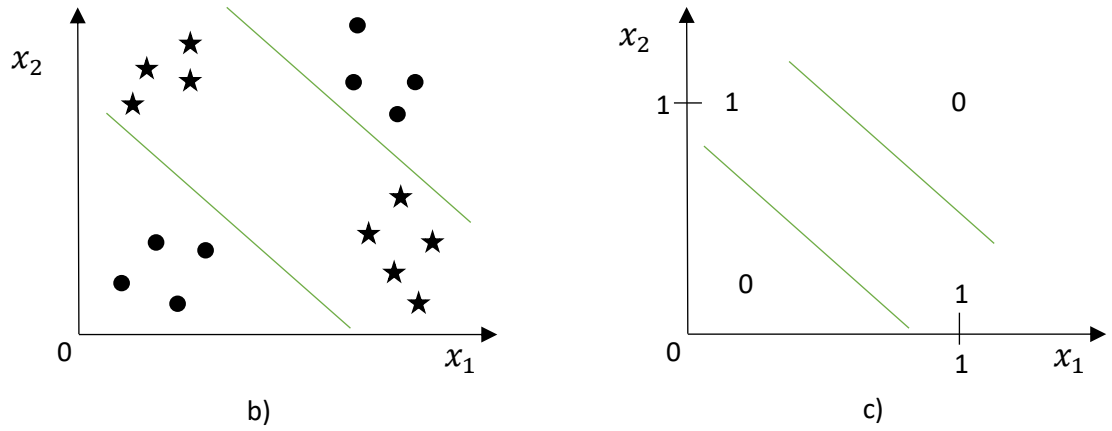


Figura 52. Problema de la XOR.

El problema de la XOR demuestra la limitación inherente de clasificación usando un solo perceptrón, por tal motivo es necesario el uso de RNA de más de una neurona, para crear arquitecturas más complejas que ayuden a crear modelos de clasificación más complejos.

B.2.2. Funciones de activación

Las funciones de activación tienen dos propósitos, el primero es generar modelos no lineales y el segundo es transmitir la idea de "disparar sobre un umbral" [28]. Las funciones de activación clásicas que se utilizaron en los comienzos del desarrollo de las RNA fueron las funciones de signo, sigmoidea y tangente hiperbólica [15], pero actualmente existen otras funciones de activación que se han implementado en las RNA. Algunas de las funciones de activación más importantes son las siguientes:

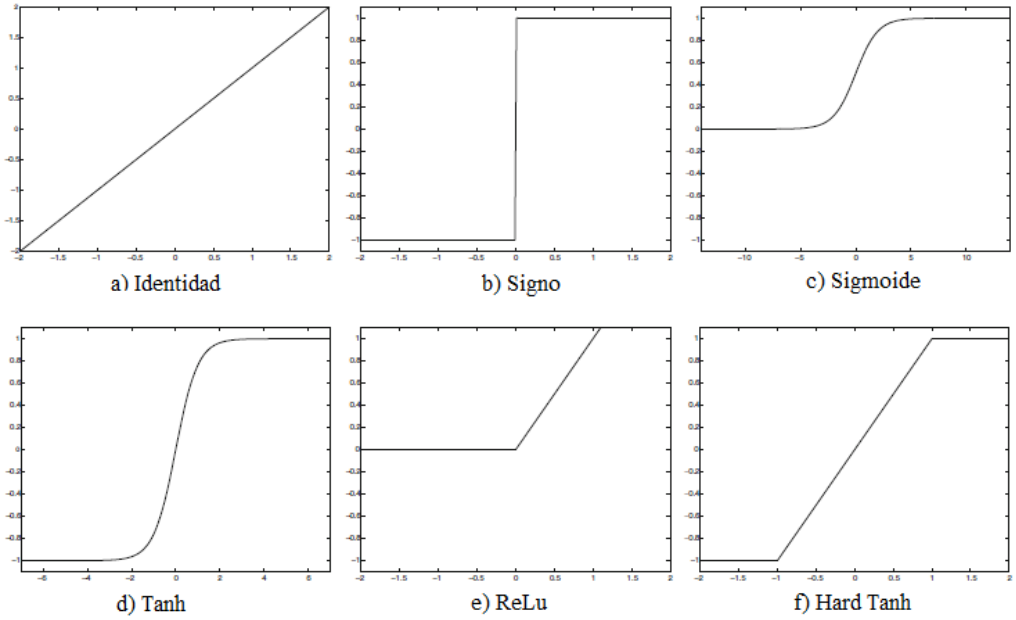


Figura 53. Algunas funciones de activación [15].

La expresión matemática para cada función de activación de la Figura 53 es:

- Función *Signo*

$$f(z) = sign(z) = \begin{cases} +1 \; si \; z \geq 0 \\ -1 \; si \; z < 0 \end{cases} \tag{31}$$

- Función *Identidad*

$$f(z) = z \tag{32}$$

- Función *Sigmoide*

$$f(z) = \frac{1}{1 + e^{-y}} \tag{33}$$

- Función *Tanh*

$$f(z) = \frac{e^{2z} - 1}{e^{2z} + 1} \tag{34}$$

- Función *ReLu*

$$f(z) = \max\{0, z\} \tag{35}$$

- Función *Hard Tanh*

$$f(z) = \max\{\min[z, 1], -1\} \tag{36}$$

B.3. Arquitectura de una red neuronal multicapa (Perceptrón Multicapa)

Para generar Redes Neuronales Artificiales (RNA) más complejas, basta con agregar más neuronas a el modelo, pero ¿Cómo se agregan las neuronas?, ¿Deben tener algún orden específico? En esta sección se abordará la creación de una arquitectura de una Red Neuronal Multicapa, conocida también como Perceptrón Multicapa debido al uso de neuronas como el perceptrón.

Primero, se simplifica la forma de representar una neurona artificial como se muestra en la Figura 54. Donde la neurona incluye los pesos y sesgos correspondientes, así como la función de activación sin escribirlos explícitamente.

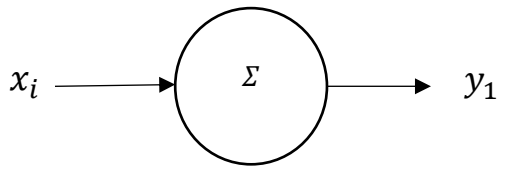


Figura 54. Simplificación de una neurona artificial

Hay dos formas de colocar las neuronas artificiales dentro de una RNA, como muestra la Figura 55, se puede colocar dos neuronas de forma secuencial, Figura 55a), o de forma paralela, Figura 55b).

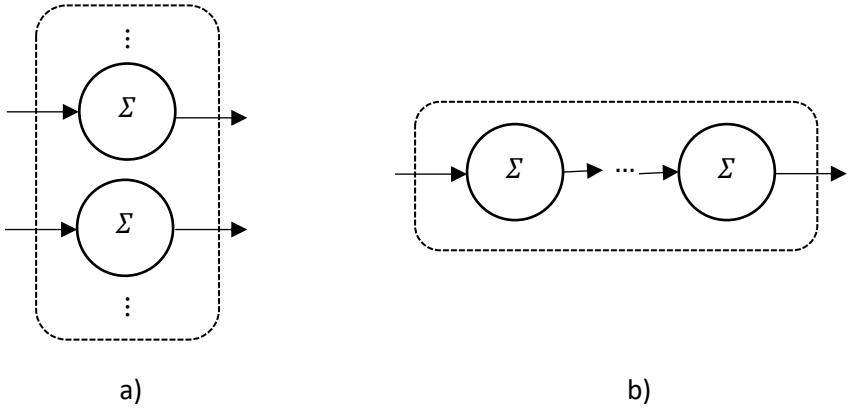


Figura 55. Formas de conectar neuronas artificiales.

La forma de ordenar las neuronas como en la Figura 55a) es mejor conocida como colocar las neuronas en la misma capa, donde cada neurona recibe los mismos valores de entrada. La forma de ordenarlas como la Figura 55b), se conoce como ordenar las neuronas en distintas capas, donde cada capa recibe como valores de entrada los valores de salida de la capa anterior. A partir de estas formas de ordenar las neuronas, se pueden hacer combinaciones de ambas para así crear RNA multicapas con N capas y con M neuronas en cada capa.

B.3.1. Red Neuronal de Una Capa

El perceptrón representa una RNA de una sola capa con una sola neurona y demuestra su limitado alcance cuando se intenta modelar la compuerta lógica XOR. Si se colocan más de una neurona en una misma capa, obteniendo más de un dato de salida, se podrían clasificar más de dos clases de datos. La Figura 56 muestra la estructura general de una RNA de una sola capa.

Al crear una capa con múltiples perceptrones, se crea una red de alimentación directa de una sola capa con múltiples perceptrones. Esta red intenta representar la forma en que el cerebro humano procesa los datos de entrada mediante un trabajo en paralelo [26].

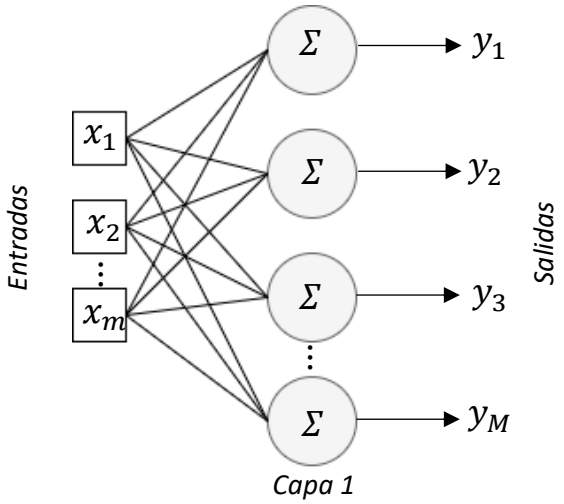


Figura 56. RNA monocapa.

En la Figura 56 se pueden observar los datos de entra, que normalmente se conoce como capa de entrada y no se contabiliza como capa de procesamiento en la RNA. También se cuenta con una capa de salida, que al igual que la capa de entrada, no se contabiliza [28]. Los conectores entre los datos de entrada y los perceptrones, así como las conexiones de los perceptrones con otros perceptrones y la salida se denominan sinapsis [26].

B.3.2. Red Neuronal Multicapa

La gran capacidad de las RNA surge de las múltiples conexiones de las neuronas que se encuentran en más de una capa. Para generar una RNA Multicapa, o Perceptrón Multicapa, se necesita más de una capa de RNA de una capa conectadas secuencialmente. La Figura 57 muestra la arquitectura básica de una RNA Multicapa donde se cuenta con la capa de entrada, N capas ocultas y una capa de salida. Cada capa puede tener una o más neuronas, incluyendo la capa de salida.

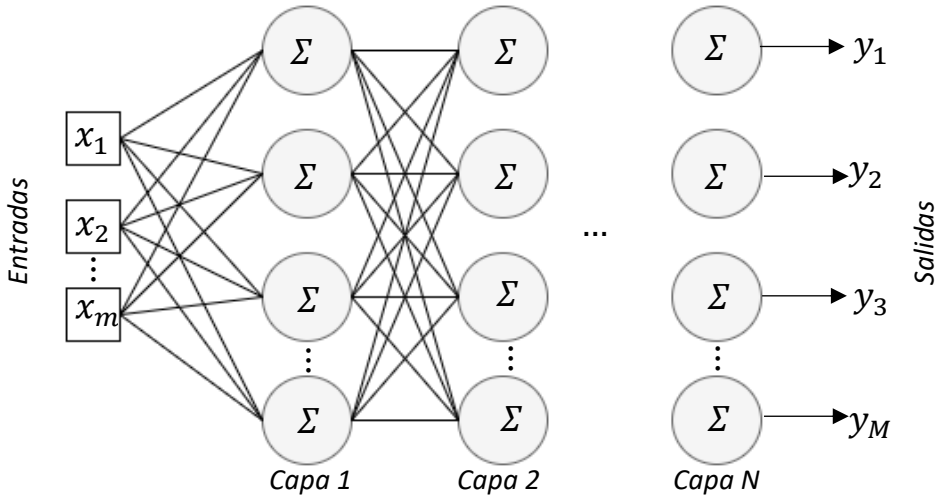


Figura 57. RNA multicapa - Perceptrón Multicapa.

Este tipo de arquitecturas multicapa de las RNA les brinda su gran característica de poder crear modelos no lineales. Para esto, la función de activación juega un papel muy importante ya que si una RNA multicapa usara solo funciones de activación lineales o no incorporara funciones de activación, no proporcionaría mayor capacidad que una RNA lineal de una sola capa [15].

La arquitectura de las RNA multicapa también se conocen como redes feed-forward [15], esto se debe a que cada capa se alimenta de la capa anterior de una manera sucesiva desde la entrada hasta la salida. Además, esta arquitectura asume que las neuronas de una capa están totalmente conectadas con las neuronas de la capa siguiente.

B.3.3. Funciones de coste

Otro punto importante en la arquitectura de una RNA, es la elección de la función de pérdida con la que se ajustarán los valores de los pesos y sesgos de cada neurona. Es fundamental seleccionar una función de pérdida, de tal manera que la salida o salidas de la red sean sensibles al problema que se esté atacando.

Anteriormente se mencionaron funciones de pérdida como el error cuadrático medio o simplemente la diferencia entre valor de real y el valor predicho, pero existen distintas funciones de pérdida aplicadas a las RNA.

En particular, para predicciones, se pueden usar dos tipos de funciones de pérdida, dependiendo de si la predicción es binaria o múltiple [15].

- Objetivos binarios (regresión logística) para casos binarios donde la función de pérdida está dada por la Ecuación (37). En este caso se asume que la función de activación a la salida de la RNA es de tipo signo, donde $y \in (-1, +1)$.

$$L = \log(1 + \exp(y \cdot \hat{y}) \tag{37}$$

- Objetivos categóricos o entropía cruzada (cross-*entropy*) para casos de predicción múltiple. Si $y_1, \ldots, y_k$ son las salidas de las probabilidades de las k clases a predecir, la función de pérdida está dada por la ecuación (38).

$$L = -\log(\ y_k) \tag{38}$$

Estas son solo dos funciones de pérdida, aunque existen más en la literatura. Por último, es importante saber que la función de pérdida, la función de activación y la cantidad de valores de salida dependen mucho del problema que se esté tratando [15].

B.4. Aprendizaje de una RNA mediante backpropagation

Hasta ahora se mencionó el funcionamiento y estructura de una RNA, desde un perceptrón hasta un una RNA multicapa, también se ha mencionado que las RNA pueden ser muy potentes para crear modelos de regresión o clasificación y que, al colocar más neuronas y más capas, la RNA puede crear modelos más complejos. Por último, se mencionó la importancia de elegir una función de pérdida, función de activación y la cantidad de valores de salida correctos para nuestro problema en particular. Pero, como el funcionamiento de la RNA se basa en tener los valores de los pesos y sesgos correctos en cada neurona, es importante establecer una técnica que ayude a que la red los determine de forma automatizada, en otras palabras, se debe establecer una manera de que la red "aprenda" a partir de un "entrenamiento".

Existen diversos algoritmos de aprendizaje que optimizan los pesos y sesgos de una RNA, uno de los más importantes y más utilizados es el algoritmo de *backpropagation* que a surgió en el año 1975.

Antes de comenzar la explicación del funcionamiento y las matemáticas de *backpropagation*, se define un concepto importante: El procedimiento de minimización de pérdida de la red. Este concepto define la cantidad de muestras de entrada con las que se alimentará la red para intentar minimizar la función de pérdida [27]. Hay dos tipos:

- *On-Line*: Para cada muestra de entrada de la red, se calcula el coste y, sobre ese coste, se aplica el algoritmo de minimización para ajustar los pesos y sesgos de la red.

- *Batch*: Se define una cantidad de valores de muestra con las que se alimentará la red y se calcula el valor de la función de pérdida total de todas las muestras y se intenta minimizar la pérdida a partir de ese valor para actualizar los valores de los pesos y sesgos.

Backpropagation es un algoritmo basado en el descenso del gradiente, que retro propaga el error desde la última capa hacia la capa de entrada, optimizando los valores de los pesos y sesgos al minimizar la función de pérdida. Este algoritmo contiene dos fases [15]:

- Fase de avance. En esta fase la red se alimenta con la instancia de muestras de entrada, sea *On-Line* o *Batch*, para generar el o los valores de salida con los valores de los pesos y sesgos actuales, esto genera una cascada de cálculos de capa tras capa de la red para poder calcular los valores de salida, y, por lo tanto, la función de pérdida.

- Fase hacia atrás. En esta fase se aplica el descenso del gradiente a la función de pérdida, calculando la derivada de la pérdida con respecto al valor de salida y después la derivada de la pérdida con respecto a los pesos y sesgos en todas las capas, desde la capa de salida hacia la capa de entrada con la ayuda de la regla de la cadena. Con este gradiente se actualizan los pesos y sesgos intentando minimizar la pérdida.

Ahora que se ha definido la idea detrás de backpropagation, se puede realizar la definición matemática y el procedimiento del algoritmo. Backpropagation se representa como muestra la ecuación (39) [14]. Como se observa, básicamente consiste en aplicar el descenso del gradiente de una forma iterativa.

```
repetir hasta alcanzar la convergencia {
```
$$w_{update} \coloneqq w_{old} - \mu \nabla L(W) \tag{39}$$
```
}
```

Donde w_{old} es el valor de cada peso y sesgo actual al calcular la función de pérdida, w_{update} es el nuevo valor de los pesos y sesgos, μ es la tasa de aprendizaje y $\nabla L(W)$el erro o pérdida.

Cuando se quieren ajustar los valores de los pesos y sesgos para una sola neurona como el perceptrón o una RNA de una capa, el cálculo resulta más sencillo ya que la función de pérdida puede calcularse como una función directa de los pesos y sesgos, por lo tanto, el cálculo del gradiente suele ser sencillo. Cuando se trata una RNA multicapa, el cálculo del

gradiente es más complejo ya que la función de pérdida depende también de los pesos y sesgos de capas intermedias. En este caso el algoritmo de backpropagation aprovecha la regla de la cadena del cálculo diferencial, para calcular el gradiente de la función de pérdida en términos de sumas de productos de gradiente local, en las diversas rutas desde un nodo hasta la salida [15].

Tomando esta última consideración, el gradiente de la función de pérdida se calculará como se muestra a continuación

Como la función de pérdida depende de los pesos y sesgos, se debe obtener la derivada parcial de la función con respecto a los pesos y con respecto a los sesgos. Además, si la RNA se forma por N capas y se usa una función de pérdida L, y como el algoritmo comienza en la última capa, las derivadas en la última capa son:

$$\frac{\partial L}{\partial w^N} \tag{40}$$

$$\frac{\partial L}{\partial {w_0}^N} \tag{41}$$

Para visualizar mejor las dependencias de la función de pérdida, se puede enfocar en dos neuronas, una en la última capa N y otra en la capa anterior $N-1$ como muestra la Figura 58.

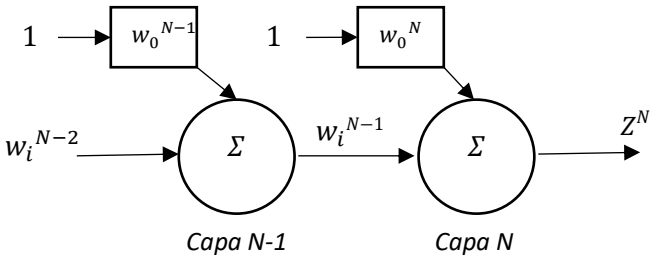


Figura 58. Conexión de dos neuronas de la capa N y N-1.

Como se observa, los valores de entrada de la capa N corresponden a los pesos y sesgos de la capa $N-1$. Como la neurona en la capa N realiza una suma ponderada, el valor de salida Z^N es:

$$z^N = W^N a^{N-1} + {w_0}^N \tag{42}$$

Donde a^{N-1} es el valor de salida de la capa anterior al pasar el valor Z^{N-1} por la función de activación. Además de la suma ponderada, el valor de Z^N es procesado por la función de activación para obtener el valor de salida.

$$a(z^N) = a(W^N a^{N-1} + {w_0}^N) \tag{43}$$

Por último, el valor de a^N es procesado por la función de pérdida para obtener el error, quedando:

$$L\big(a(z^N)\big) = L\big(a(W^N a^{N-1} + {w_0}^N)\big) = error \tag{44}$$

Como se observa en la Ecuación (44), la función de coste está formada por una composición de funciones, entonces para calcular las derivadas parciales para obtener el gradiente, se hace uso de la regla de la cadena. Al aplicar la regla de la cadena a la Ecuación (44), las derivadas parciales de la función de coste con respecto a los pesos y sesgos son:

$$\frac{\partial L}{\partial w^N} = \frac{\partial L}{\partial a^N} \cdot \frac{\partial a^N}{\partial z^N} \cdot \frac{\partial z^N}{\partial w^N} \tag{45}$$

$$\frac{\partial L}{\partial {w_0}^N} = \frac{\partial L}{\partial a^N} \cdot \frac{\partial a^N}{\partial z^N} \cdot \frac{\partial z^N}{\partial {w_0}^N} \tag{46}$$

Ahora se deben calcular cuatro derivadas parciales que se incluyen en las Ecuaciones (45) y (46). Estas derivadas son fáciles de calcular:

- Considerando, como ejemplo, la función de coste, o pérdida, del error cuadrático medio:

$$\frac{\partial L}{\partial a^N} = \frac{\partial}{\partial a^N}\left(\frac{1}{2}(\hat{y} - a^N)^2\right) = a^N - \hat{y} \tag{47}$$

- Considerando, como ejemplo, la función de activación sigmoide:

$$\frac{\partial a^N}{\partial z^N} = \frac{\partial}{\partial z^N}\left(\frac{1}{1 + e^{-z^N}}\right) = a^N(1 - a^N) \tag{48}$$

- Ya que la función z es una suma ponderada, las últimas dos derivadas quedan como:

$$\frac{\partial z^N}{\partial w^N} = \frac{\partial}{\partial w^N}(W^N a^{N-1} + {w_0}^N) = a^{N-1} \tag{49}$$

$$\frac{\partial z^N}{\partial {w_0}^N} = \frac{\partial}{\partial {w_0}^N}(W^N a^{N-1} + {w_0}^N) = 1 \tag{50}$$

Por lo tanto, las ecuaciones de los gradientes quedan de la siguiente forma:

$$\frac{\partial L}{\partial w^N} = (\hat{y} - a^N)(a^N(1 - a^N))(a^{N-1}) \tag{51}$$

$$\frac{\partial L}{\partial {w_0}^N} = (\hat{y} - a^N)(a^N(1 - a^N)) \tag{52}$$

Con las Ecuaciones (51) y (52) ya se tienen las expresiones para calcular los gradientes de cada neurona con respecto a los pesos y sesgos en la última capa. Para una capa anterior, $L - 1$, se aplica el mismo procedimiento. La función de coste en términos de los pesos y sesgos de la capa $L - 1$ es:

$$L\big(a^N(W^N a^{N-1}(z^{N-1}) + {w_0}^N)\big) = L\big(a^N(W^N a^{N-1}(W^{N-1}a^{N-2} + {w_0}^{N-1}) + {w_0}^N)\big) \tag{53}$$

Entonces las derivadas en la capa $L - 1$ son

$$\frac{\partial L}{\partial w^{N-1}} = \frac{\partial L}{\partial a^N} \cdot \frac{\partial a^N}{\partial z^N} \cdot \frac{\partial z^N}{\partial a^{N-1}} \cdot \frac{\partial a^{N-1}}{\partial z^{N-1}} \cdot \frac{\partial z^{N-1}}{\partial w^{N-1}} \tag{54}$$

$$\frac{\partial L}{\partial {w_0}^{N-1}} = \frac{\partial L}{\partial a^N} \cdot \frac{\partial a^N}{\partial z^N} \cdot \frac{\partial z^N}{\partial a^{N-1}} \cdot \frac{\partial a^{N-1}}{\partial z^{N-1}} \cdot \frac{\partial z^{N-1}}{\partial {w_0}^{N-1}} \tag{55}$$

Para las Ecuaciones (54) y (55) solo se desconoce el termino $\frac{\partial z^N}{\partial a^{N-1}}$. Esta derivada es la que se muestra en la Ecuación (56).

$$\frac{\partial z^N}{\partial a^{N-1}} = \frac{\partial}{\partial a^{N-1}}(W^N a^{N-1} + {w_0}^N) = W^N \tag{56}$$

Por lo tanto, las derivadas de la función de pérdida con respecto a la capa $L - 1$ quedan de la siguiente forma.

$$\frac{\partial L}{\partial w^{N-1}} = (\hat{y} - a^N)(a^N(1 - a^N))(W^N)(a^{N-1})(1 - a^{N-1})(a^{N-2}) \tag{57}$$

$$\frac{\partial L}{\partial {w_0}^{N-1}} = (\hat{y} - a^N)(a^N(1 - a^N))(W^N)(a^{N-1})(1 - a^{N-1}) \tag{58}$$

Para calcular las derivadas de la función de coste en términos de los pesos y sesgos de capas anteriores, el proceso se repite y solo basta con cambiar los índices y las derivadas se pueden aplicar.

C. Estrategias de decodificación

En la generación de secuencias temporales mediante técnicas feedforward iterativo surge la decisión de cómo elegir el siguiente token en las secuencias, intuitivamente se podría elegir el token con mayor probabilidad precedido por la RNN, sin embargo, seguir este enfoque a veces conduce a generar contenido degenerado, por ejemplo, en una tarea de generación de texto, podría resultar en frases sin sentido o bucles repetitivos. En esta sección se presentan algunas de las técnicas de muestreo más usadas para la estrategia de decodificación.

C.1. Búsqueda codiciosa (maximización)

Esta técnica se basa en elegir el token o clase con mayor probabilidad en cada paso de tiempo como muestra la ecuación.

$$w_t = maxarg_w P(w|w_{1:t-1}) \tag{59}$$

Esta estrategia ha demostrado grandes resultados en alguna tarea, sin embargo, su problema principal es que algunas palabras de gran probabilidad dentro de un paso de tiempo quedan escondidas detrás de una palabra de baja probabilidad en un paso anterior llevando a crear bucles repetitivos.

C.2. Búsqueda de haz (Beam search)

Esta técnica surge como una solución a la búsqueda codiciosa al introducir la retención de cierta cantidad de pasos anteriores y determinar que ruta logra la mayor probabilidad. En el ejemplo de la Figura 59, con una retención de dos pasos de tiempo, la frase “La joven puede” logra una probabilidad total de $0.10 + 0.82 = 0.92$, que es mayor que la de la frase “La ropa cuenta” que tiene una probabilidad total de $0.25 + 0.40 = 0.65$.

Este tipo de muestreo es bueno en tareas donde la secuencia es más predecible como en tareas de traducción ya que en tareas como la generación de historias o grandes textos, aún sigue presente el problema de repetición den bucles.

La repetición aun presente en este tipo de muestreos se puede solucionar agregando un parámetro donde se forzar la cantidad de no repeticiones de un token, sin embargo, aún con este parámetro resulta difícil encontrar un equilibrio entre los ciclos repetitivos y la cantidad de repeticiones no permitidas.

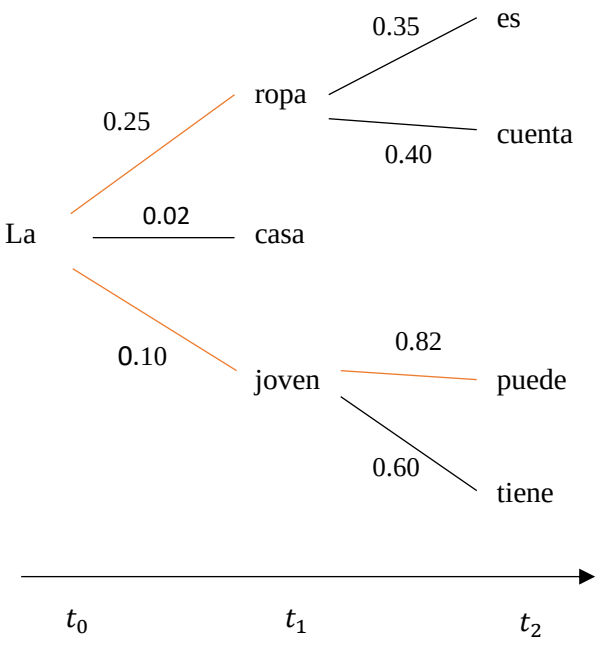


Figura 59. Ejemplo de Búsqueda de haz.

C.3. Muestreo Top-K y Top-P

El muestreo Top-K fue introducido en [29] y se basa en filtrar los K tokens más probables en cada paso de tiempo y realizar una elección aleatoria entre estos. Esta técnica de muestreo muestra ser más efectiva para generar textos largos y no tan repetitivos como la búsqueda por haz. El principal problema de esta técnica es que puede producir distribuciones de probabilidad muy distintas en cada paso de tiempo sobre las que se hace la elección del siguiente token, esto podría introducir token mal ajustados al contexto del contenido que se va generando.

Como solución al principal problema de Top-K, se introduce Top-P en [19], en donde en lugar solo muestra entre los K token más probables en cada paso de tiempo, se establece una probabilidad total y en cada paso de tiempo se seleccionan la menor cantidad de tokens cuya probabilidad acumulada apenas exceda la probabilidad establecida, así, el conjunto de palabras seleccionadas en cada paso de tiempo puede variar y se realiza una elección aleatoria sobre ese nuevo conjunto.

Estas dos técnicas de muestreo pueden usarse en combinación logrando la exclusión de tokens de baja probabilidad sobre un conjunto variable.

C.4. Muestreo de temperatura

Este muestreo está inspirado en la termodinámica estadística [30]. Esta técnica se logra al dividir los *logits* entre la temperatura, $t\epsilon[0,1)$, y después estos valores son introducidos en *softmax* y así se obtiene las nuevas probabilidades de muestreo, la ecuación (60) muestra este procedimiento.

$$P(x = V_l|x_{1:i-1}) = softmax\left(\frac{w_l}{t}\right) \tag{60}$$

Donde w_l representa los *logits*, $w_{1:|V|}$, que se obtienen al aplicar el logaritmo natural a las probabilidades obtenidas por la capa de salida de la RNN con función de activación *softmax*. Así, la elección del siguiente token se hace por medio de una elección aleatoria de las clases dada la nueva distribución de probabilidad.

En esta técnica de muestreo, además de los *logits* como valores de entrada, se tiene el parámetro de temperatura, t, el cual determina el nivel de aleatoriedad. Valores de t mas cercanos a 0 hace que la distribución se parezca más la dada por la RNN, mientras que valores mas cercanos a 1 hacen que la nueva distribución se aleje más de la real.

Esta técnica también se ha usado en [31] en combinación con Top-K donde se aplica primero la ecuación para darle una nueva forma a la distribución de probabilidad antes de hacer la elección aleatoria sobre la K clases más probables.

D. Formato MIDI

Para este trabajo se utilizan conjuntos de melodías en formato midi [17], por tal motivo se da una descripción de los aspectos más importantes de este formato para conocer la forma en que se representan algunos conceptos musicales para su manipulación.

Tener un archivo con extensión ".mid" es equivalente a tener una partitura de alguna melodía, ya que es una forma compacta de almacenar representaciones musicales como notas, acordes, silencios a través de eventos midi. Los eventos midi son una descripción musical de un sonido, mas no el sonido final de cómo se percibe, por lo tanto, no se considera un formato de audio digital, y el sonido final depende del dispositivo que lo reproduzca.

Los archivos MIDI se componen de distintos segmentos como el de la cabecera del archivo, cabecera de la pista y segmentos de pistas que contienen eventos MIDI. En particular, son de interés los eventos MIDI, los cuales contienen las representaciones de las notas, acordes y silencios a manipular. La representación de las notas se realiza a partir de un número entero positivo comprendido entre 0 y 127, cubriendo 11 octavas. Estos valores se muestran en la Tabla 4.

Tabla 4. Valor entero para cada nota en midi.

	Nota											
Octava	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

Los acordes se representan como un vector de distintas longitudes, dependiendo de la cantidad de notas por acorde. Estos vectores pueden formarse con números que representan distintos elementos, pero para este caso se trabaja con valores que representan números de clase de tono: [0, 2, 3, 5].

Las duraciones para cada nota, acorde o silencio se establecen con un valor decimal que representa cada figura y silencio de una partitura. Una duración de 0.25 representa una semicorchea y un 7.25 representa una redonda.

Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA

ACTA DE EXAMEN DE GRADO

No. 00101
Matrícula: 2202800371

Generación Automatizada de Musica.

BRAYAN ARMANDO MERINO ALVARADO
ALUMNO

REVISÓ

MTRA. ROSALÍA SERRANO DE LA PAZ
DIRECTORA DE SISTEMAS ESCOLARES

En la Ciudad de México, se presentaron a las 11:00 horas del día 22 del mes de noviembre del año 2022 en la Unidad Iztapalapa de la Universidad Autónoma Metropolitana, los suscritos miembros del jurado:

DR. PEDRO LARA VELAZQUEZ
DRA. BIBIANA OBREGON QUINTANA
DR. SERGIO GERARDO DE LOS COBOS SILVA

Bajo la Presidencia del primero y con carácter de Secretario el último, se reunieron para proceder al Examen de Grado cuya denominación aparece al margen, para la obtención del grado de:

MAESTRO EN CIENCIAS (CIENCIAS Y TECNOLOGIAS DE LA INFORMACION)

DE: BRAYAN ARMANDO MERINO ALVARADO

y de acuerdo con el artículo 78 fracción III del Reglamento de Estudios Superiores de la Universidad Autónoma Metropolitana, los miembros del jurado resolvieron:

Aprobar

Acto continuo, el presidente del jurado comunicó al interesado el resultado de la evaluación y, en caso aprobatorio, le fue tomada la protesta.

DIRECTOR DE LA DIVISIÓN DE CBI

Roman Linares Romero
DR. ROMAN LINARES ROMERO

PRESIDENTE

DR. PEDRO LARA VELAZQUEZ

VOCAL

Bobregón
DRA. BIBIANA OBREGON QUINTANA

SECRETARIO

DR. SERGIO GERARDO DE LOS COBOS SILVA