Universidad de La Habana
Facultad de Matemática y Computación

Una propuesta para la codificación de soluciones del Problema de Enrutamiento de Vehículos a un espacio continuo

Autor: **Dalianys Pérez Perera**

Tutor: **MSc. Fernando Rodríguez Flores**

Cotutor: **Lic. Alain Cartaya Salabarría**

Trabajo de Diploma
presentado en opción al título de
Licenciado en Ciencias de la Computación

Noviembre de 2021

A mis padres,
a mi hermano,
a Alain,
a Sinaile,
a mi familia,
a todos los que confiaron en mí cuando yo no creía.

Agradecimientos

Un agradecimiento especial a mis padres, por estar siempre presentes en mi vida, por apoyarme y consentirme en todo momento.

A mi tía Marilín por acogerme como otra hija más durante esta etapa.

A Alain por ser el mejor maestro, ejemplo y compañero de viaje.

A Dayany Alfaro González por brindarme su amistad y las largas horas de estudio.

A Ashley Tejeda Meneses por siempre estar en los buenos y malos momentos.

A la familia cojimera, muchas gracias por abrirme sus puertas y ayudarme siempre.

A la profesora Celia Tamara González González por apostar por mí desde primer año, permitirme trabajar con ella y por apoyarme en todo momento, sobretodo en esta última etapa.

A José Jorge Rodríguez Salgado por la brillante idea que significó el punto de partida de esta tesis.

A mi tutor Fernando Rodríguez Flores por depositar su confianza en mí desde un inicio, por guiarme durante todo este proceso y regalarme sus conocimientos y consejos.

Opinión del tutor

Con este trabajo se explora la posibilidad de convertir el espacio de soluciones discreto de un VRP a un espacio continuo sobre el que se puede aplicar cualquiera de los algoritmos de optimización continua existentes. Para realizar esta transformación se usan técnicas de aprendizaje de máquinas.

Creo que la palabra clave en este trabajo es aprendizaje, y no solo el que realizan los *autoencoders* para convertir elementos de un espacio en elementos de otro, sino el que hemos realizado todos los involucrados en este proyecto.

Para realizar esta tesis Dalianys tuvo que incursionar en (aprender) temas que no están incluidos en su plan de estudios, (aprender a) resolver creativamente problemas que aparecieron como parte del proceso de desarrollo y adquirir habilidades (aprender) relacionadas con la escritura de documentos científicos.

Por otra parte, los demás involucrados también hemos aprendido: de la constancia y dedicación de Dalianys, de su independencia, y de su capacidad de búsqueda, organización, procesamiento y análisis de información científica. Por todo eso, y por la trayectoria que ha tenido como estudiante y como alumna ayudante, considero que estamos en presencia de un excelente trabajo, que resume una excelente trayectoria, de una excelente profesional de la Ciencia de la Computación.

MSc. Fernando Raul Rodriguez Flores
Facultad de Matemática y Computación
Universidad de la Habana
Noviembre, 2021

Resumen

El problema de enrutamiento de vehículos (VRP) es uno de los problemas más estudiados en el área de la optimización combinatoria. Su objetivo es diseñar un sistema de rutas que permita satisfacer las demandas de un conjunto de clientes de manera eficiente. Una solución al problema se puede interpretar como una lista de rutas donde cada una de estas constituye una secuencia de los clientes que se visitarán en la misma. Este tipo de representación discreta de las soluciones, ha sido empleada por algoritmos heurísticos y metaheurísticos para resolver el problema. Sin embargo, en este trabajo, se presenta un modelo auxiliado de técnicas de aprendizaje automático para obtener una representación continua de las soluciones.

Se propone una herramienta para transformar soluciones del VRP a vectores reales basada en la teoría de los modelos de redes neuronales que se conocen como *autoencoders*. En particular, se formularon dos propuestas concretas: el modelo *LinearAEC*, fundamentado en un *autoencoder* clásico, y el modelo *VAE*, basado en los *variational autoencoders*. El comportamiento de ambos modelos fue similar, mostrando una alta capacidad de generación de soluciones válidas al problema de enrutamiento.

Abstract

The vehicle routing problem (VRP) is one of the most studied problems in the area of combinatorial optimization. Main objective is to design a routing system that allows to satisfy the demands of a set of customers in an efficient way. A solution to the problem can be interpreted as a list of routes where each route is a sequence of customers to be visited on that route. This type of discrete representation of the solutions has been used by heuristic and metaheuristic algorithms to solve the problem. However, in this work, a model aided by machine learning techniques is presented to obtain a continuous representation of the solutions.

A tool for transforming VRP solutions to real vectors based on the theory of neural network models known as autoencoders is proposed. In particular, two specific proposals were formulated: the *LinearAEC* model, based on a classical autoencoder, and the *VAE* model, based on variational autoencoders. The behavior of both models was similar, showing a high capacity to generate valid solutions to the routing problem.

Índice general

Índice de figuras

Introducción

El problema de enrutamiento de vehículos (VRP por sus siglas en inglés) es uno de los problemas más estudiados en el área de la optimización combinatoria. Fue introducido por primera vez por Dantzig y Ramser en 1959 [17], quienes propusieron una heurística para enrutar camiones repartidores de gasolina a las estaciones de servicio. Desde ese entonces, gran parte de la comunidad científica y empresarial le ha prestado gran atención, tanto por su complejidad como por sus múltiples aplicaciones [53].

El VRP, en su versión más general, consiste en diseñar un sistema de rutas que permita satisfacer las demandas de un conjunto de clientes de manera eficiente. El mismo puede estar sujeto a diferentes restricciones que tienen en cuenta elementos como la duración máxima de las rutas, horarios de atención, flotas compuestas por vehículos con diferentes características, servicios de recogida y entrega de mercancía, entre otros aspectos. Las combinaciones entre estas restricciones derivan en distintas variantes del problema original [53].

El VRP constituye una generalización del problema del viajante (TSP por sus siglas en inglés), por lo tanto pertenece a la clase NP-Duro, al igual que todos sus derivados [53]. De ahí la necesidad de emplear métodos aproximados para su solución. La literatura reporta soluciones a través de métodos exactos para instancias pequeñas [34, 14], mientras que para problemas de mayores dimensiones se han usado heurísticas y metaheurísticas las cuales guían el proceso de búsqueda hacia rutas competitivas de manera eficiente [53] .

Los problemas complejos del mundo real necesitan algoritmos complejos. En este sentido, los modelos de *machine learning* (aprendizaje automático o ML, por sus siglas en inglés) se sitúan en una de las líneas más populares y de mayor atracción por parte de los investigadores [21, 15]. La optimización es una componente principal en el aprendizaje automático debido a que su esencia es construir un modelo de optimización y aprender los pará-

metros de la función objetivo a partir de los datos de entrada. Por lo tanto, tienen el potencial de ser aplicables en muchas tareas de optimización, sin la necesidad de construir algoritmos que tomen en consideración múltiples restricciones y enormes espacios de solución.

Como parte de las propuestas de solución a este tipo de problemas combinatorios, se han aplicado satisfactoriamente varias técnicas de aprendizaje automático que van desde *Pointer Networks* [57, 6], *graph neural networks* [31, 50], *recurrent neural networks* (RNN) [16, 41], *deep reinforcement learning* (DRL) [40, 55, 30], hasta métodos que incorporan heurísticas de búsqueda como *Monte Carlo Tree search* (MCTS) [50, 59]. En general, estos métodos son capaces de descubrir automáticamente sus propias heurísticas basadas en los datos de entrenamiento.

En la facultad de Matemática y Computación de la Universidad de La Habana, se desarrolla una investigación que tiene como objetivo agilizar el proceso de solución de los problemas VRP para reducir el tiempo y esfuerzo humano dedicado a la búsqueda de soluciones [39, 7, 46, 49]. No obstante a ello, en trabajos anteriores no se había explorado el uso de algoritmos continuos para la solución del VRP. Para ello es necesario convertir el espacio de soluciones discreto del VRP a un espacio de soluciones continuo y esto se puede lograr mediante técnicas de aprendizaje automático, en particular, *autoencoders*.

Las redes neuronales son un conjunto de algoritmos dentro del área de ML cuya esencia es definir una función de transformación f que dada una entrada x se obtenga una salida y. Su objetivo es aproximar lo mejor posible $y = f(x, \theta)$ mediante el ajuste de los parámetros θ. Un ejemplo de estas redes son los denominados *autoencoders*. Estos modelos se entrenan para generar una copia de la entrada en la salida, es decir, codifican la entrada a una representación de tamaño fijo, también conocida como código, representación *latente* o vector de contexto, y luego, retornan como salida una reconstrucción de dicha entrada a través de una función decodificadora. El término *latente* quiere decir oculto, pero en estadísticas se usa para referirse a variables que no se observan directamente sino que son inferidas, es decir, no se conoce con certeza la función que la genera [23].

La concepción de los *autoencoders* ha sido parte del panorama histórico de las redes neuronales durante décadas desde 1980 [35, 26]. Tradicionalmente se usan para múltiples propósitos como reducción de dimensión, extracción de características, o pre-entrenamientos no supervisados [25, 8, 23]. El mecanismo básico de codificación y decodificación se aplica a una amplia variedad de elementos de entrada. Su rendimiento está determinado por la capacidad de capturar características propias y representativas de la

entrada en el vector de contexto, de forma tal que el espacio formado por dichos vectores constituya un espacio de búsqueda continuo.

Con esta tesis se propone transformar el espacio discreto del VRP a un espacio continuo, donde para cada solución del primero, se pueda obtener un vector de $\mathbb{R}^n$ que logre atrapar características y propiedades de interés de la misma. Entiéndase por espacio discreto, a que las soluciones del problema estarán dadas por una lista de rutas y estas, a su vez, por una secuencia de clientes que deben ser visitados en ese orden. Mientras que una solución continua del VRP está constituida por un vector de valores reales que a simple vista no aporta información de la transformación subyacente, pero que a partir de él es posible obtener una solución del espacio discreto usando *autoencoders*.

Esta idea de usar algoritmos continuos para resolver un problema discreto no es nueva. Existen algoritmos genéticos que utilizan vectores numéricos para representar soluciones a problemas de optimización combinatoria como en *Gonçalves et al* [22]. Sin embargo, estos enfoques se basan en esquemas de decodificación que estan cuidadosamente elaborados por expertos en el dominio. Por el contrario, en la propuesta que se ofrece en este trabajo, el proceso de codificación y decodificación relativo al problema en particular se aprende automáticamente luego de una etapa de entrenamiento del modelo. En este caso, las soluciones del VRP se codifican a un espacio continuo que posteriormente puede ser explorado empleando un algoritmo de optimización continua.

La importancia de este espacio radica en la estructura embebida en cada uno de sus elementos, en el sentido de que soluciones parecidas en el espacio discreto, se ubiquen en regiones cercanas en el continuo, es decir que mantengan una estructura o semántica similar en el espacio correspondiente. Esta peculiaridad entre las codificaciones facilitaría que un algoritmo de búsqueda empleado sobre este espacio se pueda dirigir hacia zonas prometedoras con soluciones de buena calidad para el problema original.

El estudio y análisis de las propiedades y utilidades de una representación continua para soluciones del VRP resulta de gran interés a lo largo de este trabajo. A esa motivación se suma la introducción hacia nuevas ramas de investigación en el campo de la Inteligencia Artificial, pues los trabajos anteriores pertenecientes a esta línea de investigación en la facultad, no habían tratado técnicas de ML. Como consecuencia surge la siguiente interrogante: ¿Será viable el uso de una arquitectura *autoencoders* a través de redes neuronales para transformar el espacio discreto de las soluciones del VRP a un espacio continuo? Para lograrlo se propone usar un modelo *autoencoders* capaz de codificar una solución de un problema de enrutamiento

a un vector real, y posteriormente decodificarlo nuevamente a una posible solución válida.

Objetivos

La investigación tiene como **objetivo general** la creación de una herramienta que permita transformar el espacio discreto del VRP en un espacio continuo. Para dar cumplimiento a la idea anterior se trazaron los siguientes objetivos específicos:

Objetivos específicos

1. Consultar literatura especializada sobre el estado del arte de los problemas VRP, específicamente las propuestas existentes para codificar una solución a un espacio continuo.

2. Investigar sobre las diferentes arquitecturas del estilo *encoder-decoder* que se aplican en problemas de optimización combinatoria.

3. Implementar un modelo que solucione el problema planteado empleando técnicas de aprendizaje automático, en particular, los modelos *autoencoders*.

4. Crear un marco experimental que permita evaluar distintos modelos *autoencoders* en la tarea de codificación y reconstrucción de soluciones del problema VRP.

5. Analizar los resultados obtenidos a través de un conjunto de métricas y técnicas de visualización del espacio continuo formado.

Organización de la tesis

El presente documento está organizado en 4 capítulos que recogen las distintas etapas por las que transitó la investigación.

En el capítulo 1 **Elementos de Aprendizaje Automático** se realiza una introducción a los elementos y conceptos de esta área abordados a lo largo del trabajo.

En el capítulo 2 **Preliminares**, se presenta un marco teórico con el objetivo de lograr un mejor entendimiento del problema en cuestión y de los métodos escogidos para la solución, sobretodo la teoría de *Autoencoders*.

También se reseña el estado actual de la aplicación de estos modelos a problemas de optimización combinatoria, principalmente TSP y VRP; y las técnicas más recientes en los temas tratados.

El capítulo 3 **Propuesta de Solución** describe la estructura general de la solución propuesta, desde la etapa de procesamiento y generación de los datos, hasta la especificación y diseño de las componentes que integran la misma. Además se precisan los dos modelos implementados para resolver el problema.

El capítulo 4 **Experimentos y resultados** comprende los experimentos realizados para validar el modelo, así como las métricas destinadas para su evaluación.

Por último se ofrecen las conclusiones a partir de los objetivos propuestos y los resultados alcanzados. Adicionalmente se brindan algunas ideas y recomendaciones para trabajos futuros.

Capítulo 1

Elementos de Aprendizaje Automático

Este capítulo provee una breve introducción a los elementos y conceptos del aprendizaje automático abordados a lo largo del trabajo. En su mayoría, corresponden a conceptos esenciales del área y servirán de base para el modelo seleccionado como propuesta de solución.

1.1. Aprendizaje automático

El aprendizaje automático es el subcampo de las ciencias de la computación y una rama de la inteligencia artificial, cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan. Por moderno que pueda parecer, su origen se remonta al año 1950 cuando Alan Turing creó el "Test de Turing". Para pasar el test, una máquina debía engañar a un humano haciéndole creer que se encontraba delante de un humano en lugar de un ordenador.

El aprendizaje automático abre las puertas a un nuevo paradigma de programación. La programación clásica se entiende como un conjunto de reglas diseñadas por una persona con el fin de obtener unas respuestas que cumplan dichas reglas a partir de unos datos de entrada. El aprendizaje automático, en cambio, se encarga de obtener el conjunto de reglas más efectivas que relacionan los datos de entrada con las respuestas que se esperan obtener a través del aprendizaje. Luego, estas reglas se pueden aplicar a nuevos datos para producir respuestas asociadas a los mismos. Esta comparación se muestra en la figura 1.1.

El objetivo en el aprendizaje automático es seleccionar una función o

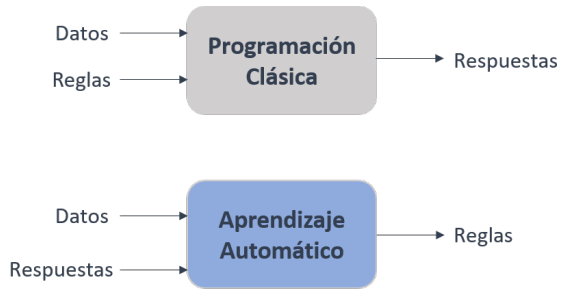


Figura 1.1: Programación clásica frente al aprendizaje automático.

hipótesis que se ajuste de manera óptima a ejemplos futuros o nuevos. Los parámetros de la función se ajustan durante el entrenamiento mediante un algoritmo de optimización conocido como descenso del gradiente. En esencia, transforma los datos de entrada a una salida que tiene una representación significativa, mediante un proceso llamado **aprendizaje**.

Las tareas en este terreno, usualmente se describen en términos de cómo se procesa la entrada al sistema. Por lo general, se expresa como un vector $x \in \mathbb{R}$ donde cada componente del mismo constituye una característica. Por ejemplo, las características de una imagen pueden ser los valores de sus píxeles. Un sistema de aprendizaje automático se basa en un proceso de **entrenamiento** en lugar de ser programado explícitamente. Gracias a ello, permite encontrar una estructura estadística en los datos de entrada que dará lugar a nuevas reglas para automatizar la tarea en cuestión. En la siguiente sección se incluyen algunas tareas que se pueden resolver con técnicas de este dominio y que serán mencionadas a lo largo del documento.

1.1.1. Tareas de ML

Una tarea de aprendizaje automático es el tipo de predicción o inferencia que se realiza, en función del problema o la pregunta que se formula, y los datos disponibles. Las mismas se basan en patrones que pueden aparecer en los datos en lugar de programarse explícitamente. A continuación se muestran algunas de ellas.

Clasificación

Su objetivo es clasificar la entrada del programa en una de las k categorías posibles según el problema. Para resolverlo, el algoritmo de apren-

dizaje produce una función $f : \mathcal{R}^n \rightarrow \{1,...,k\}$. Existen otras variantes de clasificación, por ejemplo, cuando f devuelve una función de distribución de probabilidad sobre las k categorías [23].

Regresión

El algoritmo de aprendizaje reporta una función $f : \mathcal{R}^n \rightarrow \mathcal{R}$ para predecir un valor numérico a partir de la entrada. Un ejemplo de regresión es la predicción de los precios futuros en el mercado [23].

Traducción automática

Del inglés *machine translation*. La entrada consiste en una secuencia de símbolos en un determinado lenguaje, y el algoritmo la convierte a una secuencia de símbolos pero de otro lenguaje [23]. Comúnmente se aplica en lenguajes naturales, por ejemplo para traducir del inglés al francés [51, 3].

Detección de anomalías

Es la identificación de elementos raros, eventos u observaciones que generan sospechas al diferenciarse significativamente de la mayoría de los datos. Normalmente, los datos anómalos se pueden conectar a algún tipo de problema o evento raro como, por ejemplo, fraude bancario, problemas médicos, defectos estructurales, equipo defectuoso, etc [23].

Estimación de densidad(función de probabilidad)

En el problema de estimación de densidades, el algoritmo aprende una función $p_{model} : \mathbb{R}^2 \rightarrow \mathbb{R}$ de donde $p_{model}(x)$ se interpreta como la función de densidad (si x es continuo) o una función de probabilidad (si x es discreto) en el espacio donde se encuentran los elementos de entrada. Esta tarea permite capturar explícitamente una distribución, que además, podrá ser usada en otros fines. Por ejemplo, si se tiene una función de probabilidad ya estimada $p(x)$, la misma se puede usar para obtener un valor faltante x_i a partir del resto de valores ya conocidos x_{-i}, dado por $p(x_i|x_{-i})$.

Reducción de dimensión

Se define como una forma de convertir un conjunto de datos de dimensiones elevadas en un conjunto de datos de dimensiones menores, asegurando que la información proporcionada sea similar en ambos casos. Esta

técnica es especialmente útil como paso intermedio en los modelos predictivos, ya que son conjuntos de datos que contienen un elevado número de características de entrada, y hace más complicada su función. Ejemplo de ello son los modelos *autoencoders* que se abordan en este trabajo [12].

Extracción de características

Consiste en utilizar las representaciones aprendidas por un programa entrenado previamente para extraer características interesantes en nuevas muestras o datos. Básicamente, es un proceso de reducción de dimensión mediante el cual un conjunto inicial de datos se reduce a grupos más manejables para su procesamiento. Se usan para tareas de clasificación de imágenes donde el conjunto de datos es pequeño [12].

Existen además otros tipos de tareas cuya solución es posible en este ámbito. Las listadas anteriormente son una muestra importante que están relacionadas con la investigación realizada en este trabajo.

1.1.2. Ramas dentro del aprendizaje automático

Previamente se presentaron varios tipos de tareas específicas del aprendizaje automático. La definición de las mismas está relacionada con las características y procesamiento de los datos de entrenamiento [12]. En términos generales, los algoritmos de ML se pueden clasificar como no supervisados o supervisados según el tipo de experiencia que se les permite tener durante el proceso de aprendizaje [23]. Clásicamente dichos algoritmos se separan en tres categorías descritas a continuación.

Aprendizaje supervisado

El algoritmo se entrena con variables que incluyen los valores que se desean predecir; a estos valores conocidos se les llama ” etiquetas ”. Su objetivo es crear una función capaz de predecir el valor correspondiente a cualquier objeto de entrada válida después de haber visto una serie de ejemplos, los datos de entrenamiento. Para ello, tiene que generalizar las situaciones no vistas previamente a partir de los datos presentados. La salida de la función puede ser un valor numérico (como en los problemas de regresión) o una etiqueta de clase (como en los de clasificación).

Aprendizaje no supervisado

Consiste en encontrar patrones interesantes en los datos de entrada sin necesidad de observar cuál es la salida. Por lo tanto, el objetivo es extraer información significativa de los datos, los cuales carecen de etiqueta y cuya estructura es desconocida. Existen dos tipos de problemas característicos en el aprendizaje no supervisado: el agrupamiento o *clustering*, y la reducción de dimensión [12]. El agrupamiento se encarga de crear conjuntos de objetos de características similares, mientras que la reducción de dimensión busca información redundante en los datos para reducir el número de variables y así mejorar el rendimiento computacional

Aprendizaje reforzado

El aprendizaje reforzado (*reinforcement learning*, RL) es una forma de aprendizaje automático basado en un sistema de recompensas y castigos en el que un agente busca las decisiones óptimas para obtener la máxima recompensa tanto a corto como a largo plazo. A diferencia del aprendizaje no supervisado, el aprendizaje reforzado trata de maximizar la función de recompensa (*reward function*) en lugar de encontrar ciertos patrones ocultos en un conjunto de datos no etiquetados. El aprendizaje reforzado se hace cargo de una gama cada vez más amplia de aplicaciones del mundo real: vehículos autónomos, robótica, gestión de recursos, educación, etc [12].

Un problema importante en el aprendizaje automático en general, es la tensión entre la optimización y generalización. La **optimización** se refiere al proceso de ajustar un modelo para obtener el mejor rendimiento posible en los datos de entrenamiento (aprendizaje), mientras que la **generalización** se refiere a qué tan bien se desempeña el modelo entrenado en los datos que nunca ha visto antes.

El objetivo es obtener una buena generalización, pero si no es controlada es posible que el modelo se ajuste en función de sus datos de entrenamiento solamente. Los procesos y conceptos involucrados para lograr este equilibrio se argumentan en la siguiente sección.

1.2. Fundamentos del aprendizaje automático

En este apartado se presentan los fundamentos para comprender la forma en la que las máquinas aprenden. Para ello, se explican los conceptos de función de costo y el método de descenso por gradiente. Posteriormente, se

muestran los principales errores o problemas que surgen a la hora de realizar un modelo de ML: la varianza (variance), el sesgo (bias), el sobreajuste (overfitting) y el subajuste (underfitting). Por último, se explican los pasos a seguir para desarrollar e implementar una solución con ML.

1.2.1. Aprendizaje

El aprendizaje en un sistema de ML consiste en el ajuste de los parámetros de un modelo en función de los datos recibidos. Este conjunto de datos contiene variables tanto independientes como dependientes. Las variables independientes (**características**) son aquellas usadas por el algoritmo para generar un modelo que prediga lo mejor posible las variables dependientes. Por otro lado, las variables dependientes (**etiquetas**) son el resultado de una correlación entre las variables independientes, por lo que deben ser predichas por el modelo implementado.

El modelo debe estar lo suficientemente ajustado a los datos de entrada, pero también debe tener la suficiente consistencia como para dar un buen resultado ante la introducción de datos diferentes. Para ello, el conjunto de datos se divide en 3 subconjuntos: los datos de entrenamiento (**training set**), con los que se entrenan los modelos candidatos; los datos de validación (**validation set**), para evaluar los modelos candidatos y seleccionar el mejor; y los datos de prueba (**test set**) para realizar una evaluación final del mejor modelo [47].

Una vez se tienen los datos se establece una hipótesis, es decir, encontrar una ecuación que se aproxime lo mejor posible al comportamiento real del fenómeno que se está modelando. Esta ecuación relaciona los datos de entrada y los parámetros del modelo con la salida y se conoce como **función de pérdida**. Se encarga de recopilar el error entre la variable dependiente que se quiere determinar y la hipótesis, en función de los parámetros del modelo.

La diversidad de modelos y funciones de pérdida en ML obliga a encontrar soluciones para las funciones no-convexas, es decir, para aquellas que tienen más de un mínimo. El método de **Descenso por Gradiente** aprovecha el cálculo de la derivada para encontrar los mínimos locales, ya que la derivada indica el valor de la pendiente en un punto determinado. El gradiente proporciona información sobre cuánto varía la función por cada unidad que varía cada variable en el punto considerado. Un gradiente positivo indica la dirección por la cual avanzar si se quisiera encontrar un máximo (Ascenso del Gradiente). En este caso, se explora el espacio en el sentido contrario al gradiente, ya que el objetivo es encontrar un mínimo

de la función.

1.2.2. El error y los problemas de ajuste

La precisión y la capacidad de generalizar son aspectos clave a la hora de realizar un modelo de ML solo que, lamentablemente, es imposible conseguir que un modelo esté libre de errores por completo. Comprender las principales fuentes de error ayuda a prevenir dos de los problemas más habituales en el ajuste de parámetros: el sobreajuste (**overfitting**) y el subajuste o falta de ajuste (**underfitting**).

Los errores principales en la predicción de un modelo, y que están asociados al algoritmo empleado, son la varianza y el sesgo (bias). El error de varianza está relacionado con el grado en el que la función objetivo cambia según los datos de entrenamiento proporcionados. El error de sesgo es la diferencia entre los valores reales y la predicción que espera el modelo. El error total es una combinación de varianza y sesgo, por lo que para minimizarlo es imprescindible lograr una baja varianza y un bajo sesgo. Sin embargo, la estrecha relación entre la varianza y el sesgo hace que disminuir uno de ellos implique aumentar el otro.

El subajuste (underfitting) se refiere a un modelo con un nivel de complejidad muy bajo que no tiene la precisión suficiente como para alcanzar un ajuste adecuado debido a su alto sesgo. Puede ocurrir cuando el conjunto de datos de entrenamiento no es suficiente, o cuando se utiliza un modelo lineal para ajustar datos no lineales. Por otra parte, el sobreajuste (overfitting) se produce cuando el nivel de complejidad es elevado y, por lo tanto, el modelo no tiene la capacidad de generalizar su comportamiento ante diferentes datos de entrada. Sucede cuando el modelo recoge el ruido de los datos de entrenamiento y, en consecuencia, aumenta mucho su varianza.

1.2.3. Etapas de un proyecto de ML

Una vez presentados los principios básicos del ML, es conveniente conocer el procedimiento para dar solución a problemas reales usando estas técnicas. Un proyecto de ML no se centra únicamente en elegir un modelo y entrenarlo, sino que, como todo proyecto, cuenta con una serie de etapas o pasos a seguir para aumentar sus probabilidades de éxito. A continuación, se describen ocho etapas genéricas 1.2 para llevar a cabo un proyecto de ML.

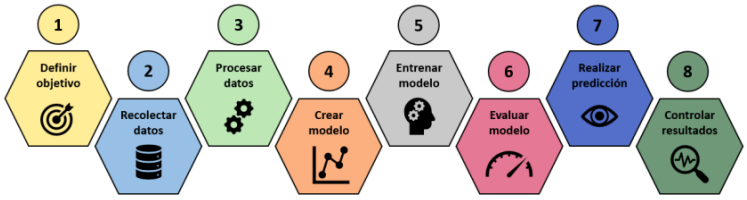


Figura 1.2: Etapas genéricas de un proceso de ML.

La primera etapa consiste en entender el problema que se quiere resolver, ya que gran parte de las decisiones tomadas a lo largo del proyecto dependerán de lo bien que se haya comprendido el contexto. Para ello es necesario definir unos objetivos claros, medibles y alcanzables en un período determinado de tiempo. En este punto se clasifica el problema (supervisado, no supervisado, etc.) e incluso se elige el tipo de modelo que se va a entrenar.

En la segunda etapa se define la cantidad y el tipo de datos necesarios, así como el origen de dichos datos. La calidad de los datos tiene un impacto directo en el funcionamiento del modelo. Por tal motivo, es necesario desarrollar una tercera etapa de tratamiento y procesamiento de los datos, cuyo objetivo es visualizar y analizar cuáles son las variables que mejor representan aquello que se quiere predecir. Además, los datos requieren un formato determinado para poder procesarse de la manera más sencilla posible. Para ello debe se debe tener en cuenta las características de la arquitectura empleada y la representación seleccionada para los datos. En esta tercera etapa se incluyen dos procedimientos que tienen lugar en este trabajo y consisten en lo siguiente:

Vectorización: todas las entradas y etiquetas deben ser vectores con valores reales.

Normalización: una vez vectorizados los datos, deben cumplir además que sus valores se encuentren en el intervalo $[0,1]$.

Una vez hecho esto, se divide el conjunto de datos en los 3 subconjuntos mencionados anteriormente.

Aunque en el primer paso ya se conoce el tipo de problema, es en la etapa cuatro donde se define por completo el modelo que mejor se ajusta al problema: regresión lineal, árboles de decisión, red neuronal, k-vecinos

más cercanos, etc. La solución propuesta en este trabajo contempla dos modelos de redes neuronales conocidos como *autoencoder* y *variational autoencoder*.

La quinta etapa se dedica al entrenamiento del modelo a partir de los datos de entrenamiento. Los parámetros se ajustan automáticamente por el algoritmo seleccionado a medida que se entrena el modelo. En la etapa número seis se verifica la precisión del modelo mediante la introducción de los datos de prueba, que son datos distintos a los del conjunto de entrenamiento. También se evalúan los errores que hacen que el modelo no generalice bien con el fin de elegir la solución más conveniente: adquirir más datos, usar un modelo más simple, usar uno más complejo, comprender mejor el problema, etc. A esta etapa también se la conoce como Parameter Tuning (configuración de parámetros), pues consiste en ajustar los parámetros del modelo para mejorar los resultados obtenidos.

Cuando se alcanza el nivel de error deseado, el modelo queda validado y puede pasarse a la penúltima etapa, que es la unión entre la simulación y el mundo real. Se trata de integrar el modelo en un sistema real con el que pueda comunicarse. Por último, la etapa número ocho pone fin al proceso con la monitorización de los resultados. Es necesario asegurar que el modelo aporta un alto valor predictivo y, lo más importante, que cumple con los objetivos marcados en la primera etapa.

El aprendizaje automático comenzó a florecer en la década de 1990. Según *Chollet et al.* [12], una tendencia impulsada por la disponibilidad de *hardware* más rápido y también conjuntos de datos de mayor dimensión. A diferencia de la estadística, el aprendizaje automático tiende a tratar con conjuntos de datos más grandes y complejos (como un conjunto de millones de imágenes, cada una de ellas formada por decenas de miles de píxeles) cuyo análisis estadístico clásico, como el análisis bayesiano, no sería practicable. Como resultado, el aprendizaje automático, y especialmente el aprendizaje profundo, exhibe poca teoría matemática. Esta disciplina, conocida en inglés como *deep learning*, se introduce en la siguiente sección.

1.3. Aprendizaje profundo

Existe un área específica del aprendizaje automático llamada aprendizaje profundo (*deep learning*), que se basa en algoritmos de aprendizaje en múltiples niveles de representación y de abstracción con el fin de modelar relaciones más complejas entre los datos. Los niveles se corresponden con distintos niveles de conceptos, y a cada uno, le corresponde una capa

dentro de una secuencia formada por capas. Esta idea de representaciones sucesivas es lo que da el nombre de "profundo" a este tipo de aprendizaje, siendo la profundidad del modelo el número de capas que contiene [18].

En el aprendizaje profundo, estas representaciones por capas forman una red neuronal (*neural network*) que constituyen un conjunto estructurado de neuronas 1.3. Estos términos provienen de la neurobiología ya que estas redes se inspiran en el funcionamiento del cerebro. Las capas están formadas por un número determinado de neuronas, donde cada neurona recibe información de la capa anterior por medio de estímulos externos a través de sus conexiones de entrada.

Las neuronas realizan cálculos internamente y sobre ellas se usa una función de activación para propagar la salida de los nodos o neuronas de la capa actual hacia la siguiente capa. Se trata de funciones que producen la activación de la neurona. Este tipo de funciones permiten incorporar la modelación de relaciones no lineales en los datos de entrada a la red.

La función de activación *relu* es la más utilizada en el aprendizaje profundo [12] y se define como $ReLU(z) = max(0,z)$. Otro ejemplo es la función *sigmoid* que se determina mediante la función $\sigma(z) = \frac{1}{1+e^{-z}}$. Ambas funciones se emplearon en los modelos implementados en este trabajo.

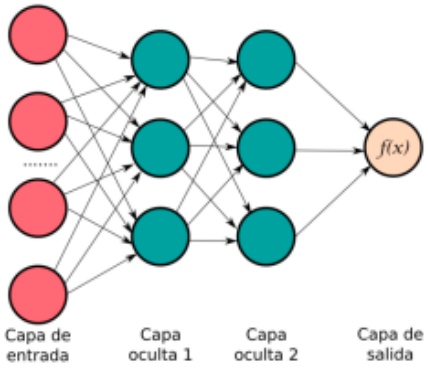


Figura 1.3: Arquitectura básica de una red neuronal.

La especificación de lo que hace una capa con sus datos de entrada se almacena en los **pesos** de la capa, que en esencia, son un conjunto de números. En términos técnicos, se dice que la transformación implementada por una capa se **parametriza** por sus pesos (los pesos también se denominan parámetros de una capa) 1.4. En este contexto, aprender significa encontrar un conjunto de valores para los pesos de todas las capas en una red, de mo-

do que la red transforme correctamente la entrada a su objetivo asociado.

El aprendizaje ocurre al extraer lotes (*batches*) aleatorios de muestras de datos y sus etiquetas, y calcular el gradiente de los parámetros de la red con respecto a la pérdida en el lote. Luego, los parámetros de la red se mueven un poco en el dirección opuesta al gradiente.

Para controlar el valor de la salida de la red, se mide qué tan lejos se encuentra la misma de la respuesta deseada mediante la **función de pérdida**, también llamada función objetivo. La función de pérdida toma las predicciones de la red y el verdadero objetivo (lo que deseaba que produjera la red) y calcula una medida de distancia, capturando qué tan bien ha funcionado la red para cada uno de los ejemplos de entrada.

Todo el proceso de aprendizaje es posible gracias al hecho de que la red neuronal constituyen cadenas de operaciones vectoriales **diferenciables** Chollet et al. [12] y, por lo tanto, es posible aplicar la regla de derivación de la cadena para encontrar la función de gradiente que transforme a los parámetros actuales y al lote actual de datos, a un valor de gradiente.

El truco fundamental en el aprendizaje profundo es utilizar esta medida como una señal de **retroalimentación** para ajustar el valor de los pesos, en una dirección que reducirá la medida de pérdida para el ejemplo actual. Este ajuste es el trabajo del **optimizador**, que implementa lo que se conoce como algoritmo **Backpropagation** [12].

En la literatura existen varios algoritmos de optimización basados en el descenso por gradiente. Entre los más conocidos se encuentran los algoritmos de optimización *Adam* y *RMSProp* [23]. Este último utiliza una tasa de aprendizaje adaptativa en lugar de tratar la tasa de aprendizaje como un hiperparámetro. Esto significa que la tasa de aprendizaje cambia con el tiempo.

Inicialmente, los pesos de la red toman valores aleatorios mediante un proceso conocido como inicialización aleatoria (*random initialization*). Naturalmente, su salida está lejos de lo que debería ser, y en consecuencia, la medida de pérdida es muy alta. Pero con cada ejemplo que procesa la red, los pesos se ajustan un poco en la dirección correcta, y la pérdida disminuye. Este es el **ciclo de entrenamiento**, que, repetido las veces suficientes, produce valores de pesos que minimizan la función de pérdida. Las iteraciones comprendidas en el ciclo se conocen como épocas (**epochs**) del proceso de entrenamiento. Una red con una pérdida mínima es aquella donde la salida es lo más parecido posible al objetivo.

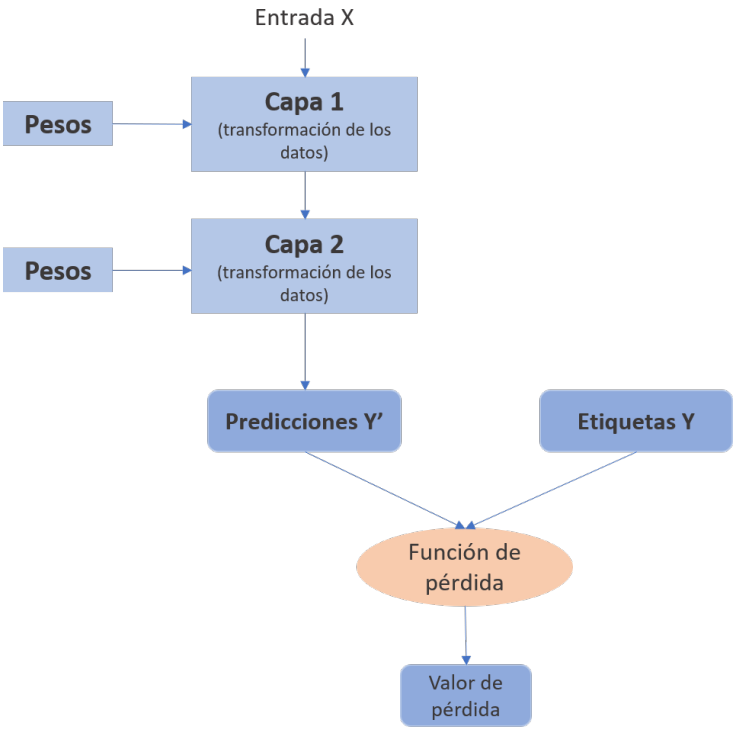


Figura 1.4: Red neuronal parametrizada por sus pesos y con función de pérdida para evaluar la calidad de la salida.

1.3.1. Tipos de redes neuronales

Una característica importante de algunas redes neuronales, como las **redes neuronales densamente conectadas**, es que no tienen memoria. Cada entrada a la misma se procesa de forma independiente, sin que se mantenga ningún estado de una entrada a otra. Para procesar una secuencia o una serie temporal de puntos en los datos, es necesario introducir de una vez la secuencia completa a la red. Tales redes se llaman redes unidireccionales (del inglés *feedforward*).

La predicción de datos secuenciales supone un problema en ML, ya que usualmente se analizan casos aislados como una imagen o un carácter que se desea clasificar. Sin embargo, lo que una persona entiende al ver una película o al mantener una conversación, se basa en conocimientos anteriores que van adquiriendo sentido con cada instante que transcurre. Las redes neuronales recurrentes (recurrent neural networks, RNN) son un ti-

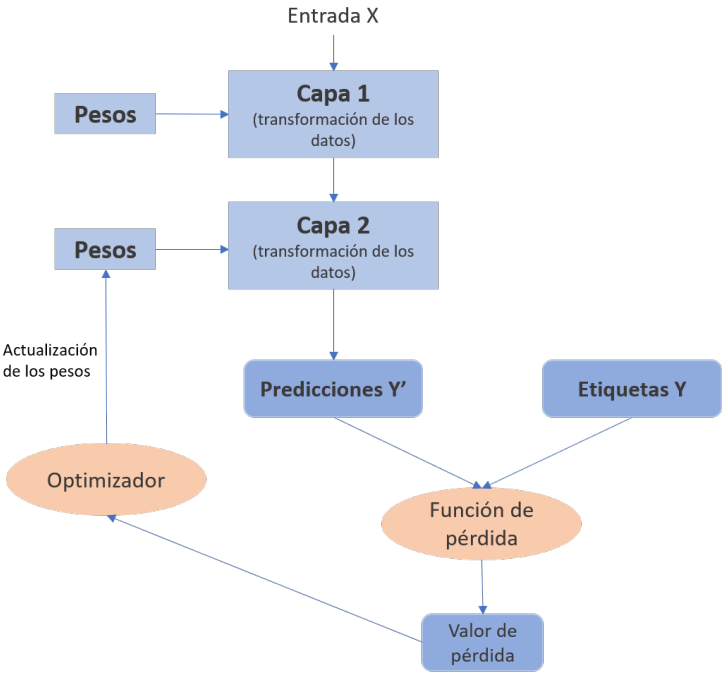


Figura 1.5: La pérdida es usada como señal de retroalimentación para el ajuste de los parámetros de la red.

po de algoritmo de aprendizaje profundo que se encarga de resolver este problema mediante el procesamiento de datos secuenciales [23].

Las RNN en lugar de tener una estructura por capas como las redes neuronales convencionales, poseen una estructura cíclica donde la salida de un estado pasa a ser la entrada del estado siguiente 1.6 . Esto permite detectar dependencias temporales, es decir, dotar a la red de memoria (estado oculto o hidden state) para que retenga la información sobre lo calculado en la etapa anterior. De esta forma, utiliza los mismos parámetros para cada entrada, reduciendo así la complejidad de la red [60].

Como parte del amplio estudio realizado en esta tesis, se revisaron trabajos que incluyen la utilización de este tipo de redes [51, 57]. Además, como se había mencionado, la solución propuesta en esta investigación contempla dos modelos de redes neuronales conocidos como *autoencoder* y *variational autoencoder*. Este último forma parte de los modelos generativos dentro del aprendizaje automático y se abordará en la siguiente sección.

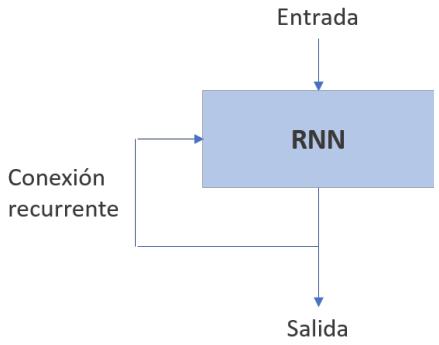


Figura 1.6: Esquema básico de una RNN

1.4. Modelos generativos

Los modelos generativos constituyen un área del aprendizaje automático que se dedica a encontrar la distribución $P(X)$, definida sobre los puntos X de un espacio $\mathcal{X}$ de gran dimensión. Por ejemplo, las imágenes son un tipo de datos muy popular empleadas en modelos generativos [19], donde cada una puede tener miles o millones de píxeles. En este caso la tarea del modelo generativo es capturar de alguna manera las dependencias entre píxeles, por ejemplo, que los píxeles con colores similares se encuentren cercanos entre sí.

Un tipo sencillo de modelo generativo permite calcular numéricamente $P(X)$. En el caso de las imágenes, los valores X que parecen imágenes reales deberían tener una alta probabilidad, mientras que las imágenes que posean determinado ruido aleatorio deberían tener una probabilidad baja.

Su idea principal se basa en generar puntos nuevos, parecidos a los del conjunto de datos inicial, pero no exactamente los mismos. Formalmente, el objetivo es aprender un modelo P para muestrear puntos X que satisfacen una distribución no conocida $P^*(X)$, de forma tal que P sea lo más parecido posible a P^*.

Los *variational autoencoders* constituyen un ejemplo clásico dentro de la familia de modelos generativos. Sus buenas propiedades de aproximación, y el pequeño error introducido con ella, aportan gran capacidad de generación al modelo. Además, son capaces de comprimir los datos de entrada en una representación con menos información, conocida como representación latente. Es por ello que su estudio resulta de gran interés en este trabajo.

En el capítulo 2 se profundizará en la teoría detrás del VAE y posterior-

mente, en el capítulo 3 se explicará en qué consiste su uso para codificar soluciones del VRP a un vector real o vector latente. Antes de llegar a esos puntos, primeramente, se analizarán los modelos de variable latente en la siguiente sección.

1.4.1. Modelos de Variable Latente

Los modelos matemáticos que tratan de explicar las variables observadas en términos de variables latentes se llaman modelos de variables latentes. Para comprender mejor en qué consisten, se presenta un ejemplo mediante el problema de la generación de imágenes referentes a dígitos escritos a mano.

Durante el proceso, el modelo primeramente decide qué dígito va a generar antes de asignar valores a cualquiera de los píxeles. Este tipo de decisiones se conocen formalmente como variable latente. Es decir, el modelo muestrea aleatoriamente un dígito z del conjunto $[0,...,9]$ y luego se asegura de que todos los trazos coincidan con el mismo. En este caso z es la variable latente. A continuación se define formalmente.

Si z es un vector de variable latente en un espacio de dimensión elevada $\mathcal{Z}$, z puede ser muestreado de acuerdo a la función de densidad probabilística $P(z)$ definida sobre $\mathcal{Z}$. También se tiene a $f(z;\theta)$ como una familia de funciones deterministas parametrizadas por un vector θ que pertenece a un espacio Θ, donde $f : \mathcal{Z} \times \Theta \rightarrow X$.

Como f es determinista y θ no lo es, entonces $f(z;\theta)$ es una variable aleatoria en el espacio $\mathcal{X}$. La idea es optimizar los parámetros θ de forma tal que se pueda muestrear z a través de $P(z)$, teniendo en cuenta que $f(z;\theta)$ serán como los elementos X del conjunto de datos.

Matemáticamente el objetivo es maximizar la probabilidad de cada punto X en el conjunto de entrenamiento a lo largo del proceso generativo, de acuerdo a:

$$P(X) = \int P(X|z;\theta)P(z)dz \tag{1.1}$$

Aquí, $f(z;\theta)$ se reemplazó por una distribución $P(X|z;\theta)$, lo que permite hacer explícita la dependencia de X con z utilizando la ley de probabilidad total. En los modelos VAEs la función de distribución empleada generalmente es Normal, es decir $P(X|z;\theta) = \mathcal{N}(X|f(z;\theta),\sigma^2 * I)$. La media se define a partir de $f(z,\theta)$ y la covarianza como la matriz identidad I multiplicada por un escalar σ (que es un hiperparámetro de la red neuronal). Estas sustituciones son necesarias para formalizar la intuición de que

las variables z necesitan ser parecidas a algún X [19].

En general, y particularmente al principio del entrenamiento, el modelo no producirá resultados que sean idénticos a cualquier X en particular. Al tener una distribución gaussiana, se usa descenso por gradiente para aumentar la probabilidad $P(X)$, haciendo que $f(z;\theta)$ se aproxime a X para algún z, es decir, permitiendo gradualmente que los datos de entrenamiento sean más probables bajo el modelo generativo.

Con este primer capítulo se presentó una introducción al aprendizaje automático. En esencia, se ofrecieron los principios y conceptos teóricos más importantes del área que fundamentan el trabajo realizado en esta tesis y muchas de las decisiones tomadas. Aprender a resolver automáticamente problemas complejos como el VRP, podría implicar el próximo salto en el desarrollo de la optimización combinatoria.

La solución propuesta en este trabajo para codificar soluciones del VRP a vectores de un espacio continuo, tiene lugar en este terreno. Específicamente mediante el uso de los modelos *autoencoders* y *variational autoencoders*. En el próximo capítulo se profundiza en la definición teórica de estos modelos de redes neuronales, así como también las aplicaciones de los mismos en problemas de codificación similares. También se exponen una serie de técnicas de ML que resultan de gran interés para esta investigación.

Capítulo 2

Preliminares

Con este trabajo se persigue diseñar un mecanismo para convertir el espacio discreto de soluciones de un VRP en un espacio de soluciones continuo. Para ello, se presentan en este capítulo los elementos fundamentales de la investigación realizada: los problemas de enrutamiento de vehículos, y las técnicas de ML que se pueden usar para obtener un espacio de soluciones continuo, a partir de soluciones discretas.

Primeramente se comienza con una visión general del problema de enrutamiento de vehículos (VRP) y algunas de sus variantes con la sección 2.1. En este contexto, se analizan varias formas de representar vectorialmente una solución a una instancia del VRP.

Luego, en las secciones 2.2 y 2.3 se expone la teoría y principios detrás de los modelos *autoencoder* (AEC) y *variational autoencoder* (VAE) respectivamente. Ambos modelos son implementados como parte de la solución. También se señalan algunas aplicaciones de los VAEs a problemas similares a este trabajo.

Las secciones 2.4 y 2.5 ofrecen varias propuestas existentes en el estado del arte y que, hasta el momento, resultan atractivas en el área del aprendizaje automático para la codificación de secuencias y solución del VRP con heurísticas aprendidas gracias a este tipo de algoritmos. La primera de ellas, aborda las arquitecturas nombradas como *sequence-to-sequence*, muy útiles para tareas donde se requiera transformar de una secuencia a otra. El acercamiento hacia estas arquitecturas parte de la interpretación que se le puede dar a una solución del VRP como un conjunto de secuencias de clientes. Mientras que la segunda reseña el empleo de *reinforcement learning* 1.1.2 en problemas de optimización de tal complejidad, principalmente en el aprendizaje automático de heurísticas para resolver problemas de enru-

tamiento.

2.1. Problema de Enrutamiento de Vehículos

Una solución del VRP requiere la determinación de un conjunto de rutas donde cada una se realiza por un solo vehículo. Las rutas comienzan y terminan en el depósito y están conformadas por una secuencia de clientes que deben ser visitados en ese el orden. La disposición de los clientes dentro del conjunto de rutas debe cumplir con las limitaciones de la flota y restricciones de los clientes, y además, minimizar el costo de transportación u otro criterio [53].

En un gran número de problemas derivados del VRP se asume que la mercancía a distribuir y la flota de vehículos se encuentran inicialmente en los depósitos. La cantidad de depósitos así como su capacidad para atender a todos o solo a un subconjunto de los clientes determinan las estrategias a seguir en la planificación de las rutas.

Los clientes contienen toda la información relativa al servicio que solicitan. Pueden estar caracterizados por la cantidad y tipo de mercancía que requieren, horarios de atención y tipos de servicios.

Por lo general, se denota por $V = \{1,2,...,n\}$ el conjunto de clientes, $V^+ = \{0\} \cup V$ el conjunto formado por los clientes y el depósito 0, d_i la demanda asociada a cada cliente de V y la flota de vehículos $K = \{1,...,m\}$, cada uno con capacidad Q . Luego para cada par de elementos $i, j \in V^+$ se define el arco (i,j) con el costo del viaje asociado c_{ij} , siendo C la matriz de costos tal que $C[i,j] = c_ij$. Las características de cada problema determinan el concepto de costo y factibilidad de las rutas.

En formulaciones más simples, el costo de una ruta se calcula a partir de la suma de los costos de los arcos involucrados en la misma; también, se asume que la flota la componen vehículos idénticos de capacidad Q y en algunos casos se supone que su número es ilimitado [53]. En problemas más complejos, la flota puede estar compuesta por vehículos diferentes en cuanto a su capacidad, costo fijo y/o variable, longitud máxima de los recorridos que pueden realizar, disponibilidad, etc... [53].

Atendiendo a las restricciones que se consideran, los problemas de enrutamiento de vehículos se agrupan en:

VRP con restricciones de capacidad (CVRP) es la variante más estudiada hasta la actualidad según Toth y Vigo en [53]. Todos los vehículos de la flota son idénticos y con capacidad limitada, por lo que durante el recorrido no se debe exceder la capacidad de ninguno de ellos.

VRP con flota heterogénea (HVRP) la flota de vehículos no es homogénea, pues pueden presentar diferentes características.

VRP con múltiples depósitos (MDVRP) existen varios depósitos desde los cuales pueden comenzar y terminar las rutas. La determinación de los depósitos de partida y fin del recorrido son también parte de la solución del problema.

VRP con ventanas de tiempo (VRPTW) es una extensión del CVRP [53] donde además de la restricción de capacidad, los clientes establecen un horario conocido como ventana de tiempo en el cual se realizará el servicio.

VRP con recogida y entrega (VRPPD) los clientes solicitan el servicio de recogida y/o entrega de mercancía.

El **objetivo de un VRP** es minimizar o maximizar determinada función. Según las características del problema las funciones objetivo más frecuentes son:

- Minimizar el costo de transportación a partir de la distancia recorrida.
- Minimizar la cantidad de vehículos.
- Minimizar los costos totales, incluidos los costos fijos y los costos variables.

Las funciones objetivo se evalúan a partir de las soluciones candidatas del problema de enrutamiento. La determinación de una representación vectorial adecuada para las soluciones, resulta una decisión importante en este tipo de tareas. Es por ello que en las próximas secciones se exponen tres formas diferentes de lograrlo

2.1.1. Representación de soluciones del VRP

Una **solución para el VRP** (en su versión más simple) es un conjunto de listas de clientes, donde cada lista representa una ruta. Por ejemplo:

$$s = [(2,5,1),(6,4),(7,8,3)], \tag{2.1}$$

representaría una solución con tres rutas para una instancia del problema con 8 clientes. En la primera se va desde el depósito hasta el cliente 2, de ahí al 5, después al 1, y por último se regresa al depósito. La segunda ruta

parte del depósito, se visitan los clientes 6 y 4, en ese orden, y se regresa al depósito. Para la tercera es similar, el recorrido inicia en el depósito y luego son visitados los clientes 7, 8 y 3, respetando el orden, y finalmente se retorna al depósito. Luego, las soluciones a problemas de n clientes estarían dadas por listas de este estilo con dígitos entre 1 y n.

Sin embargo, si en s se intercambia el orden de las dos últimas rutas como se observa en 2.2:

$$s_1 = [(2,5,1),(7,8,3),(6,4)], \tag{2.2}$$

el resultado constituye la misma solución, pues el orden dentro del conjunto de rutas no afecta el costo y las rutas como tal no cambiaron. Debido a ello, las permutaciones entre las rutas conducirán a la misma solución.

Para evitar estas repeticiones, en este trabajo se asumirá que todas las permutaciones a partir de una solución s son iguales. Para ello se considera ordenar las rutas según el primer elemento o cliente de la ruta. Por ejemplo, la solución vista inicialmente en 2.1 representaría a todas las siguientes:

$$\begin{aligned}
s &= [(2,5,1),(6,4),(7,8,3)],\\
s_1 &= [(2,5,1),(7,8,3),(6,4)],\\
s_2 &= [(7,8,3),(2,5,1),(6,4)],\\
s_2 &= [(7,8,3),(6,4),(2,5,1)],\\
s_3 &= [(6,4),(7,8,3),(2,5,1)],\\
s_4 &= [(6,4),(2,5,1),(7,8,3)].
\end{aligned}$$

Obtener una representación continua de las soluciones anteriores forma parte del objetivo de este trabajo. Para resolverlo, se usó un modelo de redes neuronales cuyo funcionamiento es posible mediante operaciones vectoriales. No obstante, como se mencionó en 1.2.3, los datos deben cumplir ciertas características como la normalización.

Con este fin, se abordan en este trabajo, otros dos tipos de representaciones vectoriales de gran utilidad para el modelo propuesto. La primera de ellas mediante un vector de números enteros simbolizando la adyacencia entre clientes en un grafo dirigido, y la segunda, a través de una matriz binaria de ceros y unos. Estas representaciones se definen en las secciones 2.1.2 y 2.1.3 respectivamente.

2.1.2. Representación a través de grafos

Como se había mencionado en 2.1, un problema de enrutamiento de vehículos se caracteriza por un conjunto V de n clientes, una flota K de m vehículos con capacidad Q y una matriz de costos C. Una solución de este problema, como la mostrada en 2.1.1, puede ser definida formalmente como un grafo dirigido $G(V,E)$ tal que:

1. $V = \{1,2,...,n\}$ es el conjunto de vértices del grafo, en correspondencia con los clientes.
2. $E = \{(i,j)|0 < i,j \leq n, i \neq j\}$ el conjunto de aristas que conforman las rutas. De forma tal que si $(i,j) \in E$ significa que el cliente j se visita después que i en el recorrido.

Al retomar el ejemplo visto en 2.1.1 donde $s = [(2,5,1),(6,4),(7,8,3)]$ el grafo correspondiente de 8 vértices quedaría conformado por el siguiente conjunto de aristas:

$$E = \{(2,5),(5,1),(6,4),(7,8),(8,3)\} \tag{2.3}$$

Gráficamente las componentes del mismo simbolizan a cada una de las rutas como se observa a continuación:

$$\begin{gathered} 2 \to 5 \to 1 \\ 6 \to 4 \\ 7 \to 8 \to 3 \end{gathered}$$

A partir de G se puede obtener una representación vectorial de una solución mediante un vector de longitud n. Dicho vector será denotado como p y constituye un vector de adyacencias entre los clientes en las rutas. De forma tal que $p[j] = i$ si $i+1$ es el padre de $j+1$ en el grafo, es decir, el cliente $i+1$ es el antecesor del cliente $j+1$ en la ruta a la que pertenecen.

Por ejemplo, en el conjunto de aristas 2.3, la arista $(2,5)$ significa que el cliente 5 es visitado después del cliente 2; de la misma forma para la arista $(8,3)$, donde el cliente 3 es quien le sigue al 8 en el recorrido. Por lo tanto, para estos dos casos se tendría que $p[4] = 1$ y $p[2] = 7$ respectivamente.

Entonces p cumple lo siguiente:

$$p[j] = \begin{cases} i & \text{si} \quad \text{el cliente } j+1 \text{ va detrás de } i+1 \text{ en la ruta} \\ j & \text{si} \quad \text{si } j+1 \text{ es comienzo de ruta} \end{cases}$$

Luego para obtener el vector p correspondiente a s es necesario tener en cuenta las relaciones de adyacencia establecidas en E. El procedimiento consiste en tomar cada arista $(i,j) \in E$ interpretada como: j es el cliente que está a continuación de i en su ruta, y fijar $p[j-1] = i-1$. Cada uno de los pasos para el ejemplo 2.3 se enumeran a continuación:

$$\begin{aligned}
(2,5) &\rightarrow p[5-1] = 2-1 \rightarrow p[4] = 1 \\
(5,1) &\rightarrow p[1-1] = 5-1 \rightarrow p[0] = 4 \\
(6,4) &\rightarrow p[4-1] = 6-1 \rightarrow p[3] = 5 \\
(7,8) &\rightarrow p[8-1] = 7-1 \rightarrow p[7] = 6 \\
(8,3) &\rightarrow p[3-1] = 8-1 \rightarrow p[2] = 7
\end{aligned}$$

Además se deben añadir los casos pertinentes a los inicios de rutas: 2, 6, y 7, bajo la condición de que si el cliente $j+1$ es comienzo de una ruta entonces se cumple que $p[j] = j$. Al aplicar esta definición se obtiene:

$$\begin{aligned}
2 &\rightarrow p[2-1] = 2-1 \rightarrow p[1] = 1 \\
6 &\rightarrow p[6-1] = 6-1 \rightarrow p[5] = 5 \\
7 &\rightarrow p[7-1] = 7-1 \rightarrow p[6] = 6
\end{aligned}$$

Finalmente el vector p resulta en $[4,1,7,5,1,5,6,6]$.

Si en s existiera una ruta formada por un solo cliente, por ejemplo un noveno cliente, la nueva solución s' quedaría conformada como:

$$s' = [(2,5,1),(6,4),(7,8,3),(9)] \tag{2.4}$$

Al intentar determinar su vector p' correspondiente, entraría en el último caso de análisis donde el cliente es comienzo de ruta. Por tanto, la interpretación se reduce a $p'[9-1] = 9-1 \rightarrow p'[8] = 8$. Luego, al adicionarlo al vector se obtiene $p' = [4,1,7,5,1,5,6,6,8]$.

En dependencia de la cantidad de clientes del problema particular, y siguiendo la definición dada en 2.1.2, se pueden generar un número considerable de vectores que las cumplan. En este sentido, la generación de soluciones del VRP resulta más fácil, sin tener que estar ordenando las rutas como en la representación a través de listas de rutas vista en 2.1.1. Por tal motivo, el conjunto de datos iniciales se generó según la representación

de soluciones vista en esta sección, o sea, mediante los vectores p asociados a la solución de una instancia del VRP.

No obstante, estos vectores p aún no están normalizados, por tanto, en la siguiente sección se introduce una nueva manera de representar soluciones con la ayuda de matrices binarias. Las mismas juegan un papel fundamental en este trabajo, pues conforman el conjunto de datos conque se entrenó y evaluó la solución ofrecida.

2.1.3. Representación de las soluciones mediante matrices binarias

Esta otra representación vectorial también se basa en la definición del grafo determinado en 2.1.2, pero en esta ocasión, mediante una matriz binaria. Se denota como M la matriz binaria de dimensión nxn tal que $M[i,j] \in \{0,1\} \forall i,j \in \{0,...,n-1\}$. Las soluciones al VRP que se representen mediante estos vectores cumplen con la siguiente definición:

$$M[i,j] = \begin{cases} 1 & \text{si} \quad \text{el cliente } j+1 \text{ va detrás de } i+1 \text{ en la ruta} \\ 1 & \text{si} \quad i = j, \text{ la ruta está formada por un único cliente i + 1} \\ 0 & \text{eoc.} \end{cases}$$

A partir de la definición anterior se cumplen las siguientes propiedades:

1. Si la fila $i-$ésima es completamente nula, entonces el cliente $i+1$ es el último en su ruta.
2. Si la columna $j-$ésima es nula entonces el cliente $j+1$ es el inicio de la ruta a la que pertenece.
3. Si $M[i,i] = 1$ es porque el cliente $i+1$ es el único en su ruta.

Con el fin de visualizar la definición y propiedades anteriores se construirá la matriz correspondiente a la solución del ejemplo 2.4, la cual incluye una ruta con un único cliente.

$$s' = [(2,5,1),(6,4),(7,8,3),(9)].$$

En este caso el conjunto de aristas se conformaba como:

$$E = \{(2,5),(5,1),(6,4),(7,8),(8,3)\}.$$

Al añadir una nueva ruta con un solo cliente no se sumó ninguna otra arista debido a que ese elemento en el grafo no está relacionado con ningún otro. El proceso de construcción es similar al del vector p en 2.1.2, por cada arista del grafo se añade el valor correspondiente según la definición. A continuación se muestran los pasos que se deben desarrollar:

$$\begin{aligned}
(2,5) &\rightarrow M[1,4] = 1 \\
(5,1) &\rightarrow M[4,0] = 1 \\
(6,4) &\rightarrow M[5,3] = 1 \\
(7,8) &\rightarrow M[6,7] = 1 \\
(8,3) &\rightarrow M[7,2] = 1
\end{aligned}$$

En los pasos anteriores, se observa que la posición $(1,4)$ de M tendrá valor 1 porque existe la arista $(2,5)$ en E, lo que se traduce a que el cliente 5 va detrás del 2 en la primera ruta de 2.1.3. De la misma forma ocurre con el resto de las aristas.

Además, con la restricción correspondiente al cliente 9 por ser el único en su ruta, se tiene que $M[8,8] = 1$. En este punto, ya se aplicaron las dos primeras condiciones de la definición, solo faltaría adicionar que el resto de las posiciones de la matriz tomarían valor igual a 0. Dicho todo esto, ya M está completamente definida y quedaría conformada como:

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix} \tag{2.5}$$

En la matriz anterior se cumple que las filas con índices 0,2,3 y 8, correspondientes a los clientes 1, 3, 4 y 9 respectivamente, son completamente nulas de acuerdo a la propiedad número 1. Pues estos clientes, como se observa en 2.1.3, son los finales de la ruta a la cual pertenecen, o sea, no existe ningún otro cliente que le suceda.

También se tiene que las columnas de índices 1,5 y 6 pertenecientes a los clientes 2,6 y 7 tienen todos sus valores iguales a cero, como plantea la segunda propiedad. Como el cliente 9 es el único en su ruta, o sea, es inicio y final simultáneamente, según las propiedades lo que debería pasar es que $M[8,8]$ sea igual a 0 pero según la definición se fijará con el valor 1. Esta decisión se tomó para añadir más valores distintos de cero a la matriz, por tanto $M[8,8] = 1$.

Una vez representadas las soluciones de un VRP de manera vectorizada y normalizada (con posibles valores 0 o 1) con las matrices M, ya se consigue que los datos cumplan con las características requeridas. Por tanto, los conjuntos de datos de entrenamiento, validación y prueba empleados durante los experimentos realizados, se aprovechan de las matrices binarias analizadas previamente.

La conformación del modelo mencionado se abordará en el capítulo 3, pero antes de llegar a ese punto, es necesario comprender los fundamentos de la arquitectura de redes neuronales seleccionada, conocida como *autoencoder*. En la siguiente sección se da inicio a tal contenido.

2.2. Autoencoders

Muchos algoritmos de aprendizaje profundo no supervisado 1.1.2 se basan en la idea de *autoencoders*, según refieren Russell y Norvig en [47] . Estos modelos intentan obtener una representación o código de los datos de entrada de forma tal que la transformación sea provechosa en determinada tarea.

Los *autoencoders* pueden aprender propiedades de los datos sin la necesidad de etiquetas anotadas por humanos, por esta razón es que son clasificados como algoritmos de aprendizaje no supervisado 1.1.2.

La idea de los *autoencoders* ha sido parte del panorama histórico de las redes neuronales desde 1980 [35, 26]. Tradicionalmente han sido usados para reducción de dimensión o extracción de características [4, 58]. Las conexiones teóricas entre los modelos de variable latente y *autoencoders* han ubicado a estos últimos dentro de la familia de modelos generativos. Pueden ser considerados como un caso especial de redes *feedforward* y por lo tanto, ser entrenados con las mismas técnicas de descenso del gradiente y *backpropagation*.

Un *autoencoders* es un modelo que contiene dos partes:

Encoder: función f que transforma la entrada x en un vector latente $\hat{z}$ de menor dimensión, $\hat{z} = f(x)$. Como el vector latente es de baja dimen-

sión, el *encoder* se obliga a aprender solo las características más importantes de los datos de entrada.

Decoder: Trata de reproducir la entrada $\tilde{x}$ a partir del vector latente $\hat{z}$, $\tilde{x} = g(\hat{z})$. Aunque $\hat{z}$ tiene una menor dimensión, posee el tamaño suficiente para permitir al *decoder* recuperar la entrada.

El modelo se entrena de forma tal que $x \approx g(\hat{z})$, o sea, el proceso de codificación sea revertido por el decodificador lo mejor posible.

Según plantea *Goodfellow et al.* [23], para los modelos generativos no resulta conveniente lograr que $g(f(x)) = x$ para todos los valores de la entrada. Esto se debe a que la idea es generar nuevos elementos no conocidos y que pertenezcan a la distribución subyacente en los datos. En cambio, se espera que al entrenar el modelo para que genere la misma entrada, se obtenga una representación capaz de agrupar aspectos y propiedades significativas de los datos [23].

En la figura 2.1 se muestra un esquema general del modelo *autoencoders* que contiene los elementos antes mencionados:

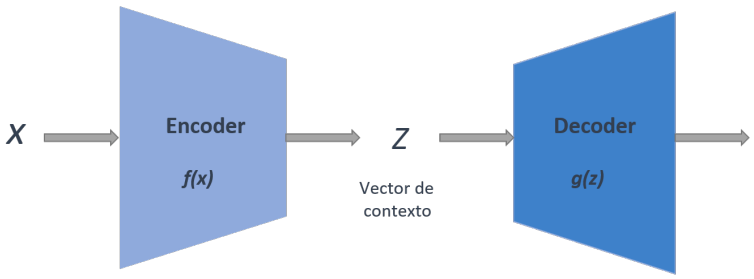


Figura 2.1: Diagrama general de un *autoencoders*

Un ejemplo de *autoencoders* muy simple es el *autoencoders* lineal, donde f y g son lineales y comparten la matriz de pesos W:

$$\begin{aligned} \hat{z} &= f(x) = Wx \\ \tilde{x} &= g(\hat{z}) = W^{\top}\hat{z}. \end{aligned}$$

El proceso de aprendizaje se describe como minimizar una función

$$\mathcal{L}(x, g(f(x))), \tag{2.6}$$

donde $\mathcal{L}$ es una función de pérdida que penaliza a $g(f(x))$ por ser diferente de x. Un ejemplo sería calculando el error cuadrático medio

$$\mathcal{L} = \sum_{j} \parallel x_j - g(f(x_j)) \parallel^2 .$$

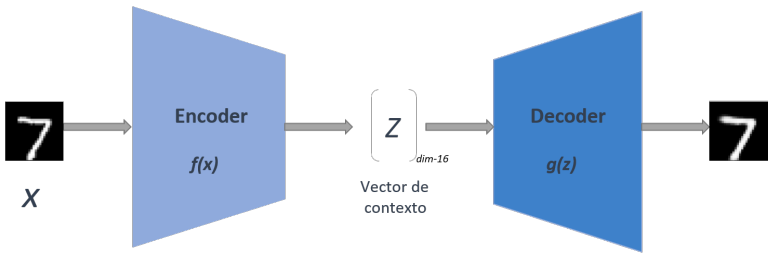


Figura 2.2: Un *autoencoders* con dígitos de MNIST como entrada y salida. El espacio latente es de dimensión 16. MNIST es una base de datos de imágenes con dígitos del 0 al 9 escritos a mano.

Si la entrada se trata como una distribución, se puede interpretar el *encoder* como un codificador de la distribución $p(z|x)$, y el *decoder*, como un decodificador de la distribución $p(x|z)$. La función de pérdida del *autoencoders* se expresa como:

$$\mathcal{L} = -\log p(x|z). \tag{2.7}$$

Si se asume que la distribución de la salida del *decoder* es Normal, entonces la función de pérdida se traduce en:

$$\mathcal{L} = -\log p(x|z) = -\log \prod_{i=1}^{m} \mathcal{N}(x_i, \tilde{x}_i, \sigma^2) = -\sum_{i=1}^{m} \log \mathcal{N}(x_i, \tilde{x}_i, \sigma^2) \alpha \sum_{i=1}^{m} (x_i - \tilde{x}_i)^2. \tag{2.8}$$

En la ecuación 2.8, $\mathcal{N}(x_i, \tilde{x}_i, \sigma^2)$ representa una distribución Normal con media $\tilde{x}_i$ y varianza σ^2, siendo esta última constante. La salida del *decoder* $\tilde{x}_i$ se asume independiente y de dimensión m.

Los *autoencoders* se aplican a muchos problemas, desde el reconocimiento facial [61, 10], detección de características [37, 45], detección de anomalías [63, 48], traducción automática [3, 51] hasta generar aleatoriamente nuevos datos que son similares a los datos de entrada o de entrenamiento.

Algunos ejemplos de estos modelos son los *autoencoders* regularizados que son efectivos en el aprendizaje de representaciones para tareas de clasificación [20] y los *variational autoencoders*, con aplicaciones como modelos generativos [58, 29]. En las siguientes secciones se presentarán estos modelos.

2.2.1. Autoencoders regularizados

Dado que se desea que el modelo descubra atributos latentes dentro de los datos, es importante asegurar que el *autoencoder* no esté aprendiendo

simplemente una forma eficiente de memorizar los datos de entrenamiento. Existen varias técnicas de regularización de la red neuronal con el objetivo de garantizar buenas propiedades de generalización [23, 2].

Los *autoencoders* regularizados, en lugar de limitar la capacidad del modelo manteniendo una arquitectura poco profunda del *encoder* y *decoder* , así como una reducción de dimensión forzada, utilizan una función de pérdida para alentar al modelo a asumir propiedades que van más allá de la simple capacidad de copiar la entrada en la salida [23].

Se esperaría que para entradas muy similares, la codificación aprendida también fuera muy similar. En otras palabras, que para pequeños cambios en la entrada, se mantenga un estado codificado muy similar. La estrategia de regularización que se conoce como ***contractive autoencoders*** está basada en esta idea. En ellas se penalizan las instancias donde un pequeño cambio en la entrada conduce a un gran cambio en el espacio de codificación.

La nueva función de pérdida, además de contar con la pérdida de reconstrucción intrínseca en el entrenamiento, se le adiciona un término de penalización denotado como $\Omega(\hat{z}, x)$ que se relaciona con la entrada x y el vector de codificación $\hat{z}$.

En los *contractive autoencoders*, este término corresponde a la norma de Frobenius de la matriz jacobiana del vector de codificación $\hat{z}$ con respecto a la entrada. La norma de Frobenius es una generalización de la norma Euclidiana y en este caso el regularizador tiene la forma $\lambda \sum_i \parallel \nabla_x \hat{z}_i \parallel^2$. Luego la función de pérdida resulta en:

$$\mathcal{L}' = \mathcal{L}(x, g(f(x))) + \lambda \sum_i \parallel \nabla_x \hat{z}_i \parallel^2 . \tag{2.9}$$

De esta forma obliga al modelo a aprender una función que no cambia mucho cuando x varía ligeramente. Como la penalidad se aplica sobre los datos de entrenamiento, el modelo está forzado a capturar información sobre la distribución del conjunto de entrenamiento [23].

Casi cualquier modelo generativo con variables latentes y dotado de un procedimiento de inferencia (para calcular representaciones latentes dada una entrada) puede verse como un caso particular de *autoencoders* [23]. Uno de los enfoques de modelos generativos que enfatiza esta conexión con los *autoencoders* son los descendientes de la máquina de Helmholtz *Hinton et al.* [24]. Llamada así por Hermann von Helmholtz, es un tipo de red neuronal artificial que puede describir la estructura oculta de un conjunto de datos, al ser entrenada para crear un modelo generativo de dicho conjunto.

El enfoque antes mencionado se conoce como *variational autoencoders*: un tipo de *autoencoder* que aprende codificaciones de alta calidad de la entrada. Dicha capacidad se adquiere debido a que se entrenan para maximizar la probabilidad de los mismos datos, más allá de copiar la entrada en la salida. Su función de pérdida también se encuentra regularizada mediante un término de penalización.

Como parte de la solución de este trabajo, se proponen dos modelos concretos para lograr codificar un conjunto de soluciones del VRP a un espacio continuo. Uno de los modelos formulados se basa en un *variational autoencoder*. En la siguiente sección se profundizará sobre dicho modelo, su definición, principios y aplicaciones.

2.3. Variational Autoencoders

Los *variational autoencoders* (VAE) fueron descubiertos simultáneamente por Kingma y Welling en diciembre de 2013 y por Rezende, Mohamed y Wierstra en enero de 2014 [12] y pertenecen a la familia de modelos generativos [23, 2, 12]. Son un tipo de *autoencoders* más avanzado que mezcla ideas de aprendizaje profundo con inferencia bayesiana [12].

Como se describió en la sección anterior, un *autoencoders* clásico transforma su entrada a un vector de un espacio a través del encoder, y luego lo decodifica en una salida con las mismas dimensiones que la entrada original. Imponiendo varias restricciones al vector codificado o vector de contexto, el modelo podrá aprender mayores o menores características latentes de la representación de los datos.

Chollet et al. [12] sostiene que los VAEs, superan a los *autoencoders* clásicos con un poco de "magia estadística"que los obliga a aprender ese espacio deseado. Un VAE, en lugar de comprimir la entrada en un código fijo del espacio latente, la convierte en dos parámetros de una distribución estadística: la media y la varianza. Esto significa que se asume que los datos de entrada se generaron por un proceso estadístico, y que la aleatoriedad de este proceso debe tenerse en cuenta durante la codificación y decodificación.

Luego, utiliza ambos parámetros para generar aleatoriamente un elemento de esa distribución, y decodificarlo por medio del *decoder*. En esencia, es un tipo de *autoencoders* cuya distribución de codificaciones se regulariza durante el entrenamiento, o sea, a la función de pérdida se le añade un término de penalización; para asegurar que su espacio latente tenga buenas propiedades y por consiguiente, permita generar nuevos datos. El término

variational proviene de la estrecha relación que existe entre la regularización y el método de inferencia variacional en estadística [12]. La aleatoriedad de este proceso mejora la robustez y obliga al espacio latente a codificar representaciones significativas. Una visión de los fundamentos matemáticos de los VAEs se muestra en la siguiente sección.

2.3.1. Principios del VAE

En presencia de un modelo generativo como el VAE, resulta de gran interés aproximar la verdadera distribución probabilística P_θ de los elementos de entrada x, usando redes neuronales.

$$x \sim P_\theta(x) \tag{2.10}$$

En la ecuación anterior, θ hace referencia a los parámetros determinados durante el entrenamiento. Por ejemplo, en el contexto de la base de datos de rostros de celebridades (CelebA[1]), sería equivalente a encontrar una distribución que pueda dibujar rostros humanos. Similarmente, en el conjunto de datos MNIST[2] que contiene dígitos del 0 al 9, la distribución puede generar dígitos escritos a mano distinguibles por humanos. La figura 2.3 muestra un ejemplo del esquema de codificación básico de un VAE usando imágenes de dígitos.

En el aprendizaje automático, para lograr cierto nivel de inferencia, interesa encontrar la probabilidad conjunta entre la entrada x y la variable latente z, denotada como $P_\theta(x,z)$ [2]. Las variables latentes no son parte del conjunto de entrenamiento pero recopilan ciertas propiedades observables en los datos. Luego $P_\theta(x)$ puede ser calculada como la distribución marginal:

$$P_\theta(x) = \int P_\theta(x,z)dz \tag{2.11}$$

Al tratar de resolver la ecuación 2.11 se presenta el problema de que $P_\theta(x)$ no posee una forma analítica o un estimador eficiente. No puede ser diferenciable con respecto a sus parámetros [2] y, por tanto, el cálculo del gradiente como parte del algoritmo de descenso del gradiente no sería posible. En consecuencia, la optimización o proceso de *backpropagation* en la red neuronal no tiene sentido. [12].

[1] `https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html`

[2] `http://yann.lecun.com/exdb/mnist/`

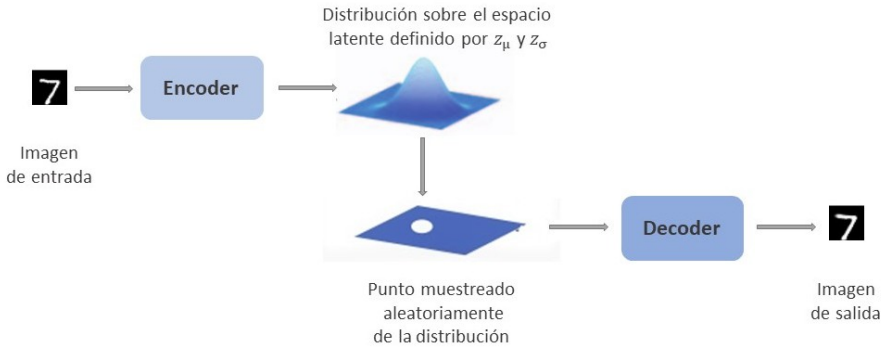


Figura 2.3: Ejemplo de VAE transformando una imagen a 2 vectores, z_μ y z_σ, los cuales definen una distribución de probabilidad sobre el espacio latente y son usados para generar un punto que será decodificado posteriormente.

Usando el teorema de Bayes se tiene una alternativa para la expresión en 2.11:

$$P_\theta(x) = \int P_\theta(x|z)P(z)dz. \tag{2.12}$$

$P(z)$ es una distribución de probabilidad sobre z, no está condicionada a ninguna de las observaciones.

En la práctica, al intentar construir una red neuronal para aproximar $P_{\theta(x|z)}$ sin una función de pérdida adecuada, simplemente ignorará z y llegará a una solución trivial [2] definida como:

$$P_{\theta(x|z)} = P_\theta(x).$$

Por lo tanto la ecuación 2.12 no proporciona una buena estimación de $P_\theta(x)$. En efecto, la ecuación 2.12 puede ser expresada también de la forma:

$$P_\theta(x) = \int P_\theta(z|x)P(x)dz. \tag{2.13}$$

No obstante, sigue siendo intratable calcular $P_\theta(z|x)$. El objetivo de un VAE es encontrar una distribución que aproxime $P_\theta(z|x)$ de la mejor manera posible. Para ello se centra en estimar la media y varianza correspondiente a dicha distribución. A continuación se exponen las ideas principales para lograrlo.

2.3.2. Inferencia en el VAE

Con el propósito de obtener $P_\theta(z|x)$, el VAE introduce un modelo de inferencia variacional(*encoder*):

$$Q_\phi(z|x) \approx P_\theta(z|x). \tag{2.14}$$

$Q_\phi(z|x)$ puede aproximarse por una red neuronal profunda mediante la optimización de los parámetros ϕ. De esta manera, proporciona un buen estimador de $P_\theta(z|x)$; que además de ser parametrizable, es tratable su cálculo en términos computacionales.

A partir del esquema de la figura 2.1, se considera a Q_ϕ para inferir las posibles variables ocultas que se usan en la generación de los vectores latentes. Tal modelo se construye mediante una arquitectura de redes neuronales donde el modelo *encoder* aprende una función de transformación de x a z y de igual forma el *decoder* de z a x como se visualiza en la figura 2.4.

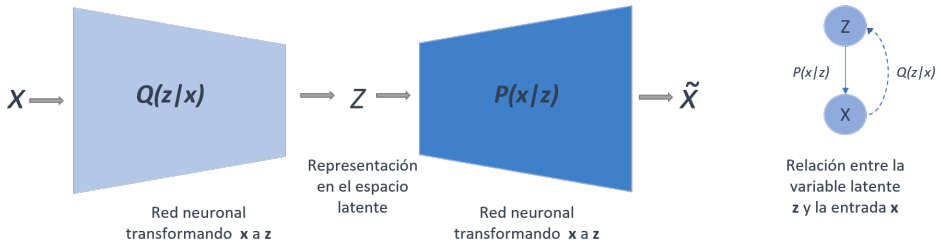


Figura 2.4: Representación del VAE mediante modelos neuronales

$Q_\phi(z|x)$ se distingue por ser una distribución gaussiana multivariada [3].

$$Q_\phi(z|x) = \mathcal{N}(z; \mu(x), \sigma(x) * I) \tag{2.15}$$

Tanto la media $\mu(x)$ como la desviación estandar $\sigma(x)$, se calculan por el modelo neuronal *encoder* usando los datos de entrada. Para construir un verdadero modelo gaussiano multivariado, sería necesario definir una matriz de covarianza que describe la correlación entre las dimensiones. Sin embargo, los VAE evitan describir explícitamente las dependencias entre las dimensiones de z [19]. Adoptan un enfoque para lidiar con este problema: asumen que no hay una interpretación simple de las dimensiones

[3] En probabilidad y estadística, una distribución gaussiana multivariada, es una generalización de la distribución normal unidimensional a dimensiones superiores

de z, y en su lugar afirman que se pueden extraer muestras de z de una distribución simple, es decir, $\mathcal{N}(0, I)$, donde I es la matriz identidad.

El modelo de inferencia $Q_\phi(z|x)$ genera un vector latente z a partir de la entrada x. $Q_\phi(z|x)$ juega el papel del *encoder* en un modelo *autoencoder*. Por otro lado, $P_\theta(z|x)$ reconstruye la entrada a partir de z. $P_\theta(z|x)$ actúa como el *decoder* en *autoencoder*. Para estimar $P_\theta(x)$, se debe identificar su relación con $Q_\phi(z|x)$ y $P_\theta(z|x)$.

Como $Q_\phi(z|x)$ es un estimador de $P_\theta(z|x)$, la distancia entre ambas densidades condicionales se determina mediante la divergencia Kullback-Leibler (KL). Esta métrica es una medida no simétrica de similitud o diferencia entre dos funciones de distribución de probabilidad y en este caso se define como:

$$D_{KL}(Q_\phi(z|x) \parallel P_\theta(z|x)) = \mathbb{E}_{z\sim Q}\left[logQ_\phi(z|x) - P_\theta(z|x)\right] \tag{2.16}$$

La función de pérdida para este modelo consta de dos términos, uno que penaliza el error de reconstrucción (intuitivamente es quien maximiza la probabilidad de reconstrucción) y un segundo término que impulsa a la distribución aprendida $Q_\phi(z|x)$ ser similar a la verdadera distribución $P_\theta(z)$. Esta última se asume que sigue una distribución gaussiana para cada elemento j del espacio latente. Luego, la función de pérdida final resulta en:

$$\mathcal{L}' = \mathcal{L}(x, \tilde{x}) + \sum_j KL(Q_\phi(z|x) \parallel P_\theta(z|x)) \tag{2.17}$$

Técnicamente el funcionamiento de un VAE consiste en:

1. La red *encoder* transforma los datos de entrada x en dos parámetros que se denotan como z_{mean} y z_{log_sigma} correspondientes a los valores μ y la σ de la distribución Normal.

2. Aleatoriamente se muestrea un punto z de la distribución Normal asumida mediante:

   $$z = z_{mean} + exp(z_{log\_sigma}) * \epsilon \tag{2.18}$$

   donde ϵ es un vector aleatorio de valores muy pequeños.

3. La red *decoder* convierte el punto z al vector de entrada original.

Como ϵ es aleatorio, el proceso asegura que cada punto cercano al vector latente z pueda ser decodificado a un vector similar a x. El hecho anterior asegura que el espacio latente sea continuo, así refiere *Chollet et al* [12].

En esencia, la red *encoder* (también conocida como red de inferencia [23]) se usa para describir una distribución de probabilidad de los elementos del espacio latente. Luego, al tomar muestras de este espacio, el *decoder* se usa para formar un modelo generativo capaz de crear nuevos datos similares a los que se observaron durante el entrenamiento. En la siguiente sección, se presentan ejemplos de situaciones donde se aplican los VAEs para obtener una representación latente de cierta información.

2.3.3. Aplicación de los VAEs a problemas similares

Una de las aplicaciones más recientes de los VAEs en general, que comparte ideas con este trabajo es [44]. Rákos et al. aborda el problema de comprimir trayectorias de vehículos del mundo real. Presentan un *variational autoencoder* para codificar la información referente a las trayectorias.

Durante el entrenamiento, el modelo aprende a codificar y decodificar estos datos y produce un pequeño vector de contexto de pocos elementos que puede representar el comportamiento del vehículo de manera probabilística. Muestran que la representación aprendida durante la compresión separa las trayectorias de acuerdo con tres clases de maniobra (mantenimiento de carril, cambio de carril derecho o cambio de carril izquierdo), evidenciando con ello gran capacidad para tareas de clasificación.

La propuesta parte de la hipótesis de predecir el vector de contexto de la trayectoria en lugar de la trayectoria en sí, pero se ve limitada por la carencia de un conjunto de datos de mayor dimensión y que represente mejor la dinámica y condiciones del ambiente de los vehículos. Aunque el sentido de estas trayectorias no comparte relación con las rutas del VRP, la idea detrás de la codificación mediante un *variational autoencoders* sí lo hace.

En [54] se manifiesta el problema de enrutar billones de componentes en dispositivos nanoelectrónicos. El proceso de enrutamiento en este tipo de sistema tiene en cuenta las componentes lógicas y las restricciones entre ellas como por ejemplo, la disposición vertical y horizontal de los cables físicos que las conectan. La solución brindada en este trabajo recibe una imagen como entrada, representada por un vector de píxeles de dimensión nxn para un n grande. En ella están comprendidas las características y restricciones del problema. La salida es un conjunto de posiciones que deben ser incluidas en las rutas contenidas en la imagen de entrada.

Con el fin de maximizar la rutabilidad (cantidad de caminos generados) y minimizar la longitud del cable (cantidad de posiciones que ocupa un camino) en un espacio en *2D* restringido, Utyamishev et al. [54] explota un proceso de entrenamientos progresivos del VAE para aprender una re-

presentación latente con datos altamente dispersos y no balanceados. Para cada ruta, el número de posiciones incluidos es $O(n)$, siendo significativamente menor que el número total de posiciones n^2. Esto último arroja datos dispersos y no equilibrados.

La noción de progresividad está dada porque el VAE fue diseñado para utilizar diferentes tamaños del conjunto de entrenamiento de forma incremental. Dicho método exhibe una rápida convergencia y alto grado de rutabilidad.

El trabajo presentado en *Hottung et al.* [29] en mayo de 2021 aporta interesantes ideas para el desarrollo de esta tesis. Ofrecen un enfoque de optimización basado en aprendizaje que permite guiar la búsqueda dentro de la distribución de soluciones de alta calidad para una instancia del problema. Construyen un *Conditional Variational Autoencoders* para transformar soluciones del problema de enrutamiento a puntos de un espacio continuo que, en conjunto, conforman un espacio de búsqueda. Su idea es que a partir de una solución buena s y una instancia del problema l, codificar esta información a un vector z. Posteriormente la capa de decodificación unido a un algoritmo de optimización, en este caso con una estrategia de evolución diferencial, generan una nueva solución s' a partir de z y l. En esta propuesta se emplean técnicas de aprendizaje automático como los modelos *sequence-to-sequence*, *Pointer Networks*, *attention mechanism* los cuales serán abordados en las próximas secciones, y por otra parte, el uso de métodos de optimización para la búsqueda en el espacio aprendido.

2.4. Arquitecturas Sequence-to-Sequence

Una solución a un problema de enrutamiento como el TSP y VRP puede ser interpretada, en su versión más simple, como un conjunto de listas de clientes donde cada lista representa una ruta. La disposición de los clientes dentro de la ruta es un factor determinante en el orden de los recorridos que se deben realizar. Por tanto, una solución se compone de un conjunto de rutas y estas a su vez, constituyen **secuencias** de clientes. Como parte de la revisión bibliográfica realizada se decidió indagar en los modelos *sequence-to-sequence*, que pertenecen a la familia de modelos de aprendizaje automático para el procesamiento del lenguaje.

Los modelos *sequence-to-sequence* [51, 57] han resultado útiles en tareas donde se requiere convertir de una secuencia a otra. Han sido ampliamente estudiados en el campo de la traducción automática en los últimos 5 años, con el objetivo de traducir una secuencia en un determinado lenguaje a su

secuencia correspondiente en otro lenguaje. La arquitectura general es muy parecida a un *autoencoder*, pues consiste en dos redes neuronales recurrentes (RNN) nombradas *encoder* y *decoder*. La red *encoder* lee la secuencia de entrada y encapsula la información extraída en un vector de tamaño fijo (o una secuencia de vectores); luego, el *decoder* convierte la información codificada a una secuencia de salida.

2.4.1. RNN Encoder-Decoder

Esta arquitectura que sirve como base y guía para los modelos *encoder-decoders* de secuencias, está compuesto por un *encoder* que lee la secuencia de entrada $X = (x_1, ..., x_T)$ y la convierte a un vector c. Cada elemento x_i de X se conoce también como palabra. La mayoría de las propuestas utilizan una RNN tal que:

$$\begin{aligned} h_t &= f(x_t, h_{t-1}) \\ c &= q(h_1, ..., h_T) \end{aligned}$$

donde h_t es el estado oculto de la RNN en el paso t, y c es un vector de contexto generado a partir de la secuencia de estados ocultos. f y q son funciones no lineales, generalmente un caso de redes neuronales recurrentes nombradas como *LSTM* (Long Short-Term Memory).

El *decoder* se entrena para predecir la próxima palabra y_t dado el vector de contexto c y todas las palabras predichas anteriormente $\{y_1, ..., y_{t'-1}\}$. La unión secuencial de los y_t conforman la predicción final y que se define a través de la probabilidad condicional conjunta:

$$p(y) = \Pi_{t=1}^{T} p(y_t | \{y_1, ..., y_{t-1}\}, c). \tag{2.19}$$

Con una *RNN* cada probabilidad condicional se modela como:

$$p(y_t | \{y_1, ..., y_{t-1}\}, c) = g(y_{t-1}, s_t, c), \tag{2.20}$$

donde g es una función no lineal que estima la probabilidad de y_t, y s_t es el estado oculto de la *RNN*

En la arquitectura elemental Sequence-to-Sequence [51], la secuencia de entrada aparece solo una vez en el *encoder* y la salida del mismo se genera a partir de un único vector (por ejemplo, el último estado oculto del *encoder RNN*). Otras extensiones como *Bahdanau et al.* [3], ilustran que la información de entrada se puede usar de forma más explícita para aumentar la cantidad de información durante los pasos de decodificación. Para ello,

emplean otra red neuronal denominada mecanismo de atención (*attention mechanism*) que "*atiende*" todos los estados del *encoder RNN*.

La idea detrás del mecanismo de atención es permitir que el *decoder* se enfoque o preste atención en zonas importantes de la secuencia de entrada y use la información relevante para producir mejores secuencias de salida. El concepto de *attention* ha sido usado por varios investigadores [57, 41, 43, 56] desde su introducción en 2015 por *Bahdanau et al.* [3].

Bello et al. [5] ofrece una propuesta sustentada en estos modelos, para predecir la distribución sobre las permutaciones de clientes en las soluciones del TSP. El análisis secuencial de las soluciones del VRP podría aportar resultados interesantes para la codificación de las mismas. Varias de las propuestas basadas en esta arquitectura, combinan también técnicas de aprendizaje reforzado. En la siguiente sección se abordan algunos ejemplos de aplicaciones de estos métodos a problemas de enrutamiento.

2.5. Reinforcement learning en problemas de enrutamiento.

Debido a su complejidad y naturaleza combinatoria, el VRP y sus variantes han sido candidatos naturales para problemas de optimización resueltos por métodos de ML. Por sus características secuenciales de decidir en qué orden se deben visitar los clientes, la dinámica del tiempo de demanda, y la falta de una distribución probabilística detrás de las rutas, estos métodos se basan principalmente en técnicas de aprendizaje reforzado (*reinforcement , RL*).

La mayoría de los enfoques de RL para resolver el VRP, lo interpretan como un proceso de decisión de Markov, en el cual la solución óptima puede verse como una secuencia de acciones que deciden qué cliente visitar una vez ha sido atendido el cliente anterior. Estos algoritmos se enfrentan a dos dificultades principales. Primero, el VRP es un problema altamente combinatorio: codificar y decodificar el estado actual y la acción son un paso crucial. En segundo lugar, su estado se ve afectado por cada decisión, de modo que la representación del problema evolucione con el tiempo. Debido a estos inconvenientes, gran parte de los autores que se mencionan en este trabajo se han concentrado en modelos basados en la información del estado previamente observado.

Vinyals et al. [57] fueron los primeros en introducir las *Pointer Networks*, un modelo inspirado en la arquitectura *sequence-to-sequence*. La arquitectu-

ra aprende la probabilidad condicional de una secuencia de salida formada por valores discretos que se corresponden a las posiciones de los elementos en la secuencia de entrada. Los autores entrenan su modelo para resolver instancias del TSP de hasta 50 clientes, usando un mecanismo de atención para obtener una permutación de la secuencia de entrada que constituye una ruta para el TSP. Durante la inferencia, seleccionar la secuencia con mayor probabilidad entre todas los opciones posibles es computacionalmente costoso e impracticable. Por ello se realiza un procedimiento de *beam search* para encontrar la mejor secuencia posible. *Beam search* es un algoritmo de búsqueda heurística que explora un grafo al expandir los k nodos más prometedores dentro de un conjunto limitado de candidatos [47].

Compararon el rendimiento con otros tres algoritmos, mejorando los resultados de [1] en casi todos los casos, excepto para instancias del problema con 40 clientes. En general mostraron buenos resultados para instancias de hasta 30 clientes.

Mientras que en [57] se obtiene una solución del TSP de una forma supervisada, *Bello et al.* [6] lo extiende con un aprendizaje no supervisado que emplea un método *critic actor*. Este método consiste en dos redes donde una actúa como actor y la otra como crítico. El actor decide qué acción se debe tomar y el cŕitico informa al actor qué tan buena fue la ación y cómo debe ejecutarla. Aplican una función de penalidad a partir de la distancia total recorrida en cada estado. Además, adaptaron su trabajo para hacer frente a las limitaciones del VRP, usando un término de alta penalidad sobre las soluciones no factibles.

Nazari et al.[41] amplía aún más este último para resolver el VRP, reemplazando la *Pointer Networks* por un *decoder RNN* basado en una proceso de *attention*. Con ello logran capturar las dependencias de tiempo y los cambios del ambiente en el VRP, principalmente cómo evoluciona la demanda con el tiempo. Su modelo es invariante en el orden de la entrada (el orden de los clientes) por tanto puede manejar eficientemente elementos dinámicos en la entrada (por ejemplo, modificación de la demanda de un cliente una vez que este se visite). Obtienen resultados similares a las propuestas de última generación en problemas de pequeña y mediana dimensión. Múltiples extensiones se han introducido para resolver variantes del VRP, como [55, 11] para el HVRP, y [62] para el VRPTW.

Hottung et al. [28] propone un nuevo enfoque llamado *neural large neigborhood search* (NLNS) que integra aprendizaje de heurísticas a la metaheurística de alto nivel *large neigborhood search* (LNS). El mecanismo de aprendizaje está basado en una red neuronal profunda con un *attention mechanism* cuyo funcionamiento consiste en aprender heurísticas para reparar solucio-

nes incompletas del VRP y emplearlas después como guía en la búsqueda de nuevas soluciones.

Capítulo 3

Propuesta de solución

En este capítulo se ofrece la estructura general que representa la solución propuesta para codificar soluciones del VRP a un espacio continuo. Su diseño abstracto mediante dos componentes principales, *encoder* y *decoder*, permite la rápida adaptación de otra arquitectura de redes neuronales al modelo general.

Como parte de la implementación se formularon dos modelos concretos para resolver el problema de codificación, ***LinearAEC*** y ***VAE***. El primero de ellos está basado en un modelo *autoencoder* clásico y el segundo, en un *variational autoencoder*.

También se describe la etapa de generación y procesamiento de los datos para conformar los conjuntos de entrenamiento, prueba y validación. En este punto se retoman los tipos de representación vectorial de soluciones del VRP expuestos en la sección 2.1.1.

A continuación se explica la idea central del problema, el procesamiento de los datos y el diseño de la solución.

3.1. Definición del problema

La codificación de soluciones del VRP de un espacio discreto a vectores reales, surge por el interés de investigar el espacio continuo que puede obtenerse con el empleo de un modelo *autoencoders* a partir de soluciones discretas del VRP. De ahí que sea fundamental lograr una representación continua capaz de capturar sus características y propiedades.

El VRP es un problema de optimización combinatoria cuyo objetivo es minimizar el valor de una función de costo C sobre el espacio $\mathcal{S}$ de soluciones, es decir $min\ C(s)$, $s \in \mathcal{S}$ teniendo en cuenta las características y res-

tricciones del problema. El espacio $\mathcal{S}$ lo forman soluciones representadas por vectores discretos, bien podrían ser cualquiera de las representaciones vistas en el capítulo 2. En este caso se considera que $\mathcal{S} \subset \mathbb{R}^n$, está formado por vectores con la misma estructura que los vectores p analizados en 2.1.2.

La intención de este trabajo es obtener una representación $z \in \mathcal{Z}$ a partir de s, donde $\mathcal{Z}$ es un espacio continuo $\mathcal{Z} \subset \mathbb{R}^l$, para algún l. Es decir, determinar una función:

$$f : \mathcal{S} \to \mathcal{Z}, \tag{3.1}$$

de forma tal que $f(s) = z$, y además, recuperar la solución codificada en z mediante otra función:

$$g : \mathcal{Z} \to \mathcal{S}, \tag{3.2}$$

cuyo objetivo es conseguir que el resultado de $g(z) = s'$ sea lo más cercano posible a la entrada original s. Con esta idea, las soluciones iniciales s se podrán representar mediante vectores con valores reales cuya dimensión depende de la dimensión del problema, y se conoce como código, vector de contexto o vector latente.

Sea s una solución del VRP, el problema enunciado puede reducirse a aproximar las funciones f y g de forma tal que $f(s) = z$ y $g(z) = s$ con el uso de técnicas de aprendizaje profundo, en particular: modelos *autoencoders*. La funciones f y g se denominan *encoder* y *decoder* respectivamente y están constituidas por redes neuronales. La arquitectura específica del modelo se precisará en la sección 3.3, pero antes se describe el proceso seguido para generar los datos empleados en el proceso de entrenamiento.

3.2. Procesamiento de datos de entrenamiento

Una cuestión importante que se debe abordar antes de introducir el modelo es el procesamiento y preparación de los datos de entrada tal y como se señaló en 1.2.3. Como parte de esta etapa, se pueden aplicar varias técnicas que dependen del tipo de datos con los que se cuenta, por ejemplo si son de tipo texto o imagen.

El procesamiento previo permite una mejor adaptación entre los datos y la red neuronal mediante una serie de técnicas como vectorización de la información y normalización. Considerando que en la representación vectorial expuesta en 2.1.2 con los vectores p, los valores no están normalizados, se decidió representar los datos a través de las matrices binarias M analizadas en 2.1.3.

Como el objetivo principal es la codificación de una solución a un vector real, solo interesan en ellas, los clientes y cantidad m de rutas. La conformación del conjunto de datos o soluciones iniciales para una instancia particular con n clientes se desarrolló generando los vectores p, que como se había explicado, establecen la adyacencia en el grafo formado a partir de las rutas contenidas en la solución. Una vez obtenido ese conjunto de datos, cada solución del mismo se convierte a la matriz M correspondiente, con el fin de obtener una representación normalizada.

Una vez representadas las soluciones del VRP como un vector donde cada componente pertenece al intervalo $[0, 1]$, se puede presentar la estructura general del modelo propuesto en la siguiente sección.

3.3. Estructura general del modelo propuesto

El modelo ofrecido en este trabajo para la codificación de soluciones de un problema VRP, sigue la idea general de los modelos *autoencoders* descritos en 2.2 y consiste en dos componentes principales: *encoder* y *decoder*. En la figura 3.1 se muestra un esquema general del modelo.

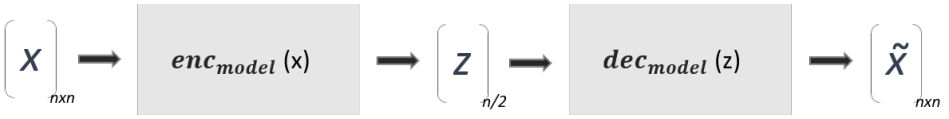


Figura 3.1: Esquema general del modelo propuesto con las componentes *encoder* y *decoder*.

Dichas componentes son las encargadas de aproximar las funciones f y g mencionadas en la definición del problema, de foma tal que f se reemplaza por la componente *encoder* y g se sustituye por la componente *decoder*. Ambas están relacionadas mediante tres vectores que juegan un papel fundamental: el vector de entrada x, vector de contexto z y el vector de salida $\tilde{x}$.

Los **vectores de entrada** constituyen matrices binarias de dimensión $n \times n$ representando a las soluciones del VRP. Además de ser la entrada a la red general, son la entrada de la componente *encoder*. Este devuelve como salida un vector real denotado por z que se conoce como **vector de contexto**. Posteriormente el *decoder* procesa el vector z y retorna un **vector de salida** que establece una reconstrucción de la entrada s. Tanto la entrada inicial al modelo como la salida final presentan las mismas dimensiones como se observa en el esquema 3.1.

La configuración de parámetros en las componentes y la dimensión del vector z dependen de la dimensión del problema, por tanto, la arquitectura del modelo se ajusta al tipo de problema particular. Debido a ello es necesario entrenar el modelo para cada dimensión del problema, o sea, para una cantidad de clientes fija.

3.3.1. Funcionalidades del modelo propuesto

La estructura genérica presentada anteriormente permite el uso de las componentes *encoder* y *decoder* de forma independiente, una vez que se entrene el modelo. En este sentido se distinguen tres funcionalidades que cumple el esquema general de la solución.

Siguiendo las definiciones señaladas en este capítulo, se denotará como s el vector de entrada al modelo que representa una solución del VRP, z el vector de contexto representativo de s en el espacio continuo y s' la reconstrucción de s y también salida del modelo. A partir de este momento, las componentes *encoder* y *decoder* también serán tratadas como enc_{model} y dec_{model}.

Las funcionalidades referidas se exponen a continuación:

- **codificar (encode)**: Mediante esta función es posible obtener el vector de código para una entrada s cualquiera, de forma tal que:

$$encode(s) = enc_{model}(s) = z \tag{3.3}$$

- **decodificar (decode)**: A partir de un vector z del espacio continuo, se puede producir la solución correspondiente en el espacio discreto aplicando:

$$decode(z) = dec_{model}(z) = s' \tag{3.4}$$

- **predecir (predict)**: Esta función da como resultado el vector reconstruido luego de atravesar por las dos etapas básicas de codificación y decodificación. Su empleo tiene sentido con el modelo ya entrenado para comprobar el desempeño del mismo. El funcionamiento consiste en:

$$predict(s) = dec_{model}(enc_{model}(s)) = s' \tag{3.5}$$

En la siguiente sección se profundizará en la descripción de ambos modelos, teniendo en cuenta la arquitectura de redes neuronales seleccionada.

3.4. Modelos propuestos

Como parte de la implementación de la propuesta de solución, se diseñaron dos modelos *autoencoders* para intentar resolver el problema: ***LinearAEC*** y ***VAE***. El primero de ellos conformado sobre la base de un *autoencoders* clásico y el segundo, como su nombre lo indica, ideado a partir de un prototipo de VAE tradicional.

La función de pérdida de reconstrucción empleada en ambos modelos fue MSE (Mean Squared Error), el promedio del cuadrado de las diferencias entre la entrada y la salida. La última capa de cada modelo utiliza la función de activación *sigmoid*, y además, se decidió usar como optimizador *RMSProp*, siguiendo las sugerencias brindadas en *Chollet et al.* [12]. En el resto de las capas de ambos, se utiliza la función de activación *relu*.

En lo adelante se denotarán como E_i y D_i las capas pertenecientes a las redes *encoder* y *decoder* respectivamente. Las características específicas, arquitectura y configuración de ambas propuestas serán abordadas en las siguientes secciones.

3.4.1. Modelo *LinearAEC*

El carácter lineal de esta propuesta está determinado por la arquitectura de capas densas empleado. Tal y como se mencionó en 3.3, se encuentra conformado por dos componentes principales y tres vectores importantes. La configuración de la arquitectura y los parámetros correspondientes será detallada a continuación.

Arquitectura del modelo

Ambas componentes principales *encoder* y *decoder* constituyen redes neuronales formadas por capas densas. En la figura 3.2 se muestra la estructura interna del modelo de inicio a fin:

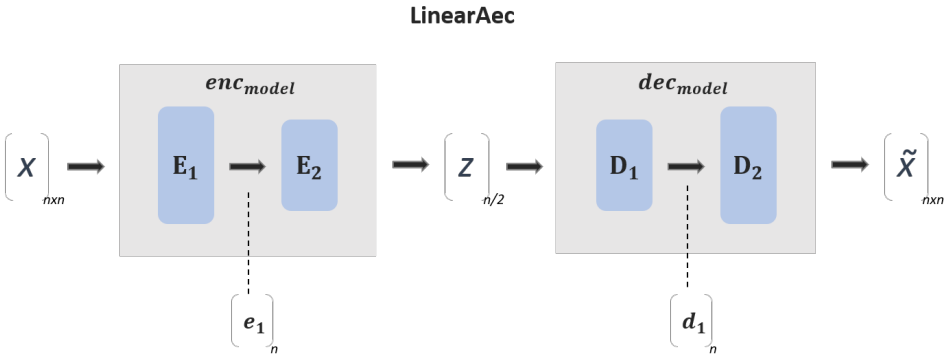


Figura 3.2: Diagrama del modelo LinearAEC

El *encoder* recibe una matriz de entrada x la cual se introduce a la primera capa oculta E_1. La salida de esta se conforma mediante la función de activación *relu* y pasa a ser la entrada de la siguiente capa E_2, encargada de devolver el vector de contexto z. Durante la codificación, la dimensión de los vectores entre una capa y otra va disminuyendo hasta obtener z.

Posteriormente, z se convierte en la entrada de la red *decoder* donde será decodificado con el objetivo de recuperar la entrada x inicial. La primera capa de entrada D_1 en esta componente, recibe el vector latente y devuelve como salida un vector de mayor dimensión con respecto a z, que será la entrada de la última capa D_2. Esta capa es la encargada de reconstruir la solución representada en x. Durante el proceso de decodificación la dimensión de los vectores de salida va incrementando en correspondencia con la disminución que se establece con la codificación. Por tal motivo, se puede decir que ambos procesos sin simétricos.

3.4.2. Modelo *VAE*

Los VAE constituyen una versión más moderna de los *autoencoders* que proporcionan una noción probabilística para describir un punto del espacio latente. En lugar de generar valores directamente para el estado latente como se hace en 3.4.1, el modelo de codificación de un VAE generará parámetros que describen un distribución para cada dimensión en el espacio latente. Dado que se asume una distribución Normal, se generan dos vectores que describen la media y varianza de las distribuciones del espacio. Al generar puntos de tal distribución se obtienen nuevos datos, por eso es considerado un modelo generativo. Finalmente el *decoder* procederá a desa-

rrollar una reconstrucción de la entrada original.

Arquitectura del modelo

Similar al *LinearAEC*, en el *VAE*, las componentes *encoder* y *decoder* constituyen redes neuronales conformadas por capas densas. Lo interesante y diferente en este modelo, con respecto al *LinearAEC*, es el proceso intermedio donde se genera el vector latente z. En la figura 3.3 se muestra un diagrama de la estructura interna del *VAE* implementado, que facilitará la comprensión de sus etapas.

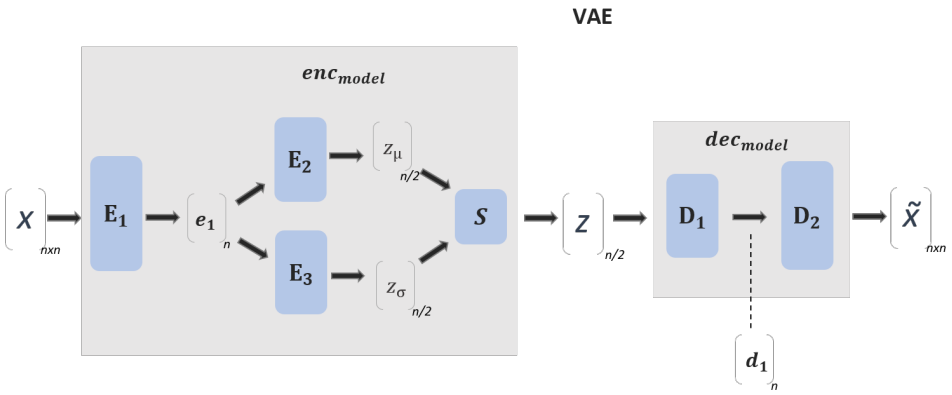


Figura 3.3: Diagrama del modelo VAE

El *encoder* recibe un vector de entrada x de dimensión nxn el cual se procesa por la capa de entrada E_1. A partir de este punto, es donde el proceso de codificación cambia en comparación con el *LinearAEC*. Pues la salida de E_1 se introduce paralelamente a otras dos capas ocultas independientes que se identifican como E_2 y E_3.

El papel de estas nuevas capas es producir los vectores que describen la distribución de las codificaciones z. A través de E_2 se obtiene z_{mean} y con E_3 se origina z_{log_sigma}. Hasta el momento se ha explicado el punto número 1 de 2.3.2, donde se tiene que:

$$z_{mean}, z_{log\_sigma} = enc_{model}(x) \tag{3.6}$$

Como parte de la generación del vector z con los parámetros de la media y varianza de la distribución, se creó una capa personalizada que se deno-

mina *Sampling*. La capa se denota como S en la figura 3.3, recibe los parámetros mencionados y su esencia consiste en $z = z_{mean} + exp(z_{log\_sigma}) * \epsilon$.

Una vez generado el vector z, este se convierte en la entrada del *decoder*. En lo adelante, el comportamiento del *decoder* es igual a la decodificación en el LinearAEC. Constituido por dos capas densas consecutivas D_1 y D_2, donde el vector de salida va aumentando de dimensión de una capa a la otra. Finalmente se obtiene la salida $\tilde{x}$ como reconstrucción de la entrada inicial x.

En el presente capítulo se expusieron dos modelos concretos para obtener una representación continua de las soluciones del VRP. Ambos se sustentan en la teoría analizada en el capítulo 2 e intentan describir una distribución de probabilidad de los vectores de codificación. En el siguiente capítulo se muestran los resultados obtenidos a partir de dos escenarios donde se evalúa el comportamiento de los mismos.

Capítulo 4

Experimentos y resultados

En este capítulo se presentan los experimentos realizados para evaluar el desempeño de los modelos propuestos: ***LinearAEC*** y ***VAE***. Se enuncian las métricas empleadas para evaluarlos, así como también la herramienta utilizada para visualizar el espacio latente conformado por los vectores de codificación. Finalmente se muestran los resultados obtenidos en varios ejemplos y se ofrece un análisis a partir de los mismos.

4.1. Consideraciones de la etapa de experimentación

La generación de un conjunto de soluciones para instancias del VRP con cantidad de clientes considerables, resulta computacionalmente costoso. A ello se le suman los altos requerimientos de cómputo que demanda el proceso de entrenamiento. Por estas razones, se diseñó un marco experimental que fuera factible de ejecutar con recursos limitados.

El equipo de cómputo donde se realizaron los experimentos posee las siguientes propiedades:

Memoria: 16GB - RAM

Procesador: *AMD A10-8700P Radeon R6*

Arquitectura: *64 bits*

La implementación de la solución se desarrolló en el lenguaje de programación *Python* y auxiliado del *framework Keras* [13]. Este último es una biblioteca de código abierto implementada en *Python* y diseñada para posibilitar la experimentación de técnicas de aprendizaje profundo mediante

redes neuronales. Sus principales ventajas consisten en ser amigable para el usuario, modular y extensible, de ahí su selección.

Como parte de esta etapa de experimentación, se realizaron varios intentos progresivos de entrenamiento. Se comenzó por conjuntos de datos pequeños hasta finalmente probar con otros de mayor dimensión y así, observar el desempeño con el aumento de los datos. En cada experimento la cantidad de datos destinados para la evaluación del modelo representa el 20% del total, el 80% restante se conforma de los datos de entrenamiento.

El proceso de aprendizaje pertenece a la categoría de aprendizaje no supervisado debido a que el conjunto de datos está formado por pares de la forma (M, M), donde M constituyen a las matrices binarias de entrada que representan las soluciones del VRP, y a su vez, su respectiva etiqueta o predicción esperada. El objetivo principal evaluado fue la capacidad de generación de soluciones válidas a partir de los vectores de contexto. De esta forma se determina si los vectores codificados son capaces de respresentar de manera continua a las soluciones del problema de enrutamiento.

El entorno de experimentación se desplegó mediante un módulo nombrado **Evaluador**, que se encarga de iniciar y relacionar la ejecución de los pasos comprendidos en este proceso. El evaluador recibe como argumentos la configuración de los parámetros necesarios para el experimento, el modelo a evaluar (LinearAEC, VAE u otro que cumpla la misma estructura definida en 3.3), una función generadora de soluciones del VRP y las métricas empleadas.

Para evaluar los resultados de cada experimento se formularon varias métricas cuantitativas con el fin de comparar la reconstrucción de las soluciones de entrada obtenidas por el *decoder*. Estas se aplican sobre los conjuntos de entrenamiento, validación y prueba que se forman inicialmente con el generador de soluciones. Una vez finalizado el proceso de aprendizaje del modelo particular, se establecen las métricas a partir de una solución de entrada X y su reconstrucción X_p:

Métricas de evaluación

solución válida (valid): su valor es 1 si la solución X_p es válida, en otro caso será 0.

mse: valor del error cuadrático medio entre X y X_p.

cantidad de rutas (eqRoutesNumber): su valor es 1 si la solución X_p posee la misma cantidad de rutas que la solución X, en otro caso será 0.

longitud de rutas (eqRoutesSize): su valor es 1 si el conjunto formado por las longitudes de cada ruta en la solución X es igual al conjunto correspondiente para la solución X_p, en otro caso será 0.

Como se aplican sobre los conjuntos de datos antes mencionados, se calcula el promedio de cada una de ellas en el conjunto completo.

En la siguiente sección se precisa mejor en qué consiste el experimento general elaborado y se ejemplifica el comportamiento de los modelos implicados con varias configuraciones del experimento.

4.1.1. Descripción del marco experimental

El experimento parte de la creación de una instancia del módulo Evaluador a partir de los argumentos que recibe. La configuración inicial consiste en los siguientes parámetros:

n: cantidad de clientes

r: cantidad de rutas de las soluciones

train: tamaño del conjunto de entrenamiento

val: tamaño del conjunto de evaluación

test: tamaño del conjunto de prueba

epochs: cantidad de épocas del entrenamiento

También hay que seleccionar el modelo que se evaluará, en ese caso los posibles son las dos propuestas ofrecidas: ***LinearAEC*** y ***VAE***. Con cada instancia del evaluador se realizará el mismo experimento en ambos modelos para establecer comparaciones con respecto a su rendimiento y capacidad.

La visualización de los datos juega un papel crucial en las aplicaciones de aprendizaje automático, facilitando por lo general, la interpretación y clasificación de los mismos. Por esta razón, se graficó el espacio latente conformado por los vectores reales de codificación mediante una herramienta conocida como *t-SNE*, del inglés *t-distributed stochastic neighbor embedding*. Esta técnica crea una distribución de probabilidad a partir de la distribución gaussiana que define las relaciones entre los puntos en el espacio de alta dimensión, en este caso, el espacio formado por los vectores de contexto denotados por z. Además, es capaz de preservar la estructura local y global de los datos.

Con el propósito de evidenciar las consideraciones expuestas en 4.1 y la descripción de esta sección, se presentan a continuación una serie de experimentos realizados. Para algunos de ellos, se muestran las funciones de pérdida sobre los datos de entrenamiento y validación, los valores de las métricas definidas y el espacio formado por los puntos de codificación.

4.1.2. Ejemplos de experimentos realizados

Escenario 1

La configuración incial del Evaluador se muestra en el cuadro 4.1. Para este escenario el conjunto de datos lo forman soluciones de 25 clientes y 7 rutas

Cuadro 4.1: Configuración del experimento en el escenario 1

n	**r**	**train**	**val**	**test**	**epochs**
25	7	5000	1000	1000	10

Luego de aplicar las métricas en los conjuntos de entrenamiento, validación y prueba de ambos modelos, se obtienen los resultados resumidos en los cuadros 4.2 y 4.3:

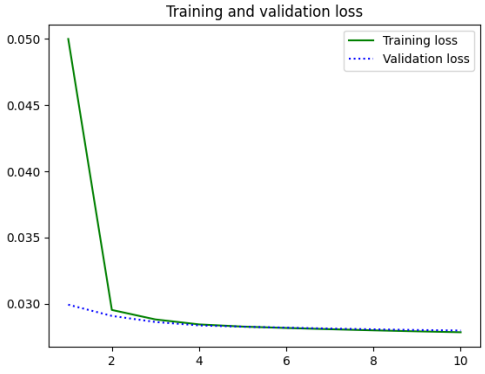
Cuadro 4.2: Resultado de las métricas en el escenario 1 para el modelo LinearAEC

conjunto	**valid**	**mse**	**eqRoutesNumber**	**eqRoutesSize**
train	0.8873	0.0264	0.0007	0.0006
val	0.893	0.0268	0.002	0.0015
test	0.8885	0.0268	0.0005	0.0005

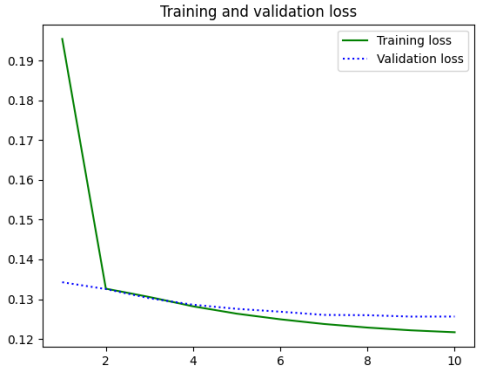
Cuadro 4.3: Resultado de las métricas en el escenario 1 para el modelo VAE

conjunto	**valid**	**mse**	**eqRoutesNumber**	**eqRoutesSize**
train	0.8803	0.0274	0.0023	0.0021
val	0.87	0.0278	0.002	0.0015
test	0.8765	0.0278	0.0035	0.0025

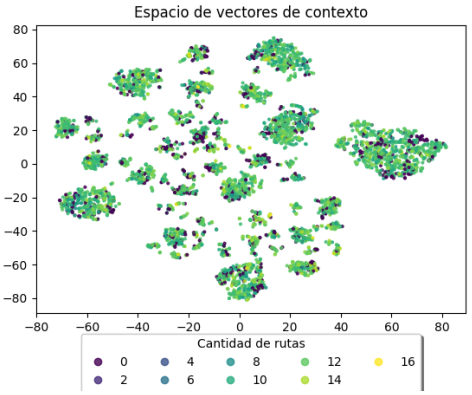
Los gráficos en 4.1.2 muestran el comportamiento de las funciones de pérdida durante el entrenamiento de ambos modelos.



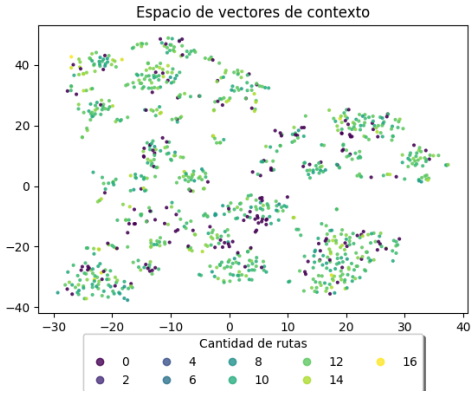
(a) Pérdida en el modelo LinearAECs



(b) Pérdida en el modelo VAE



(c) Espacio latente en el conjunto de entrenamiento del modelo LinearAEC



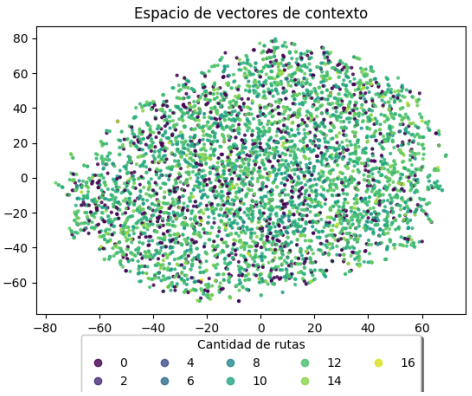
(d) Espacio latente en el conjunto de validación del modelo LinearAEC

Escenario 2

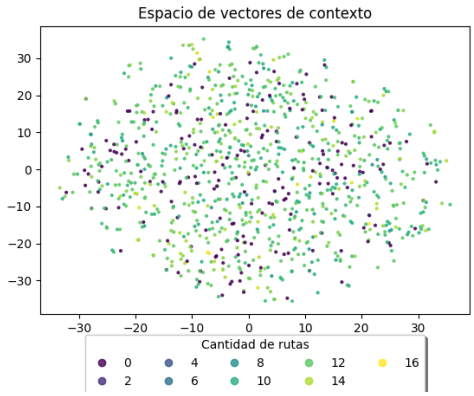
La configuración incial del Evaluador se muestra en el cuadro 4.4. Para este escenario el conjunto de datos lo forman soluciones de 20 clientes y 5 rutas

Cuadro 4.4: Configuración del experimento en el escenario 2

n	**r**	**train**	**val**	**test**	**epochs**
20	5	10000	2000	2000	20



(e) Espacio latente en el conjunto de entrenamiento del modelo VAE



(f) Espacio latente en el conjunto de validación del modelo VAE

Luego de aplicar las métricas en los conjuntos de entrenamiento, validación y prueba de ambos modelos, se obtienen los resultados resumidos en los cuadros 4.5 y 4.6:

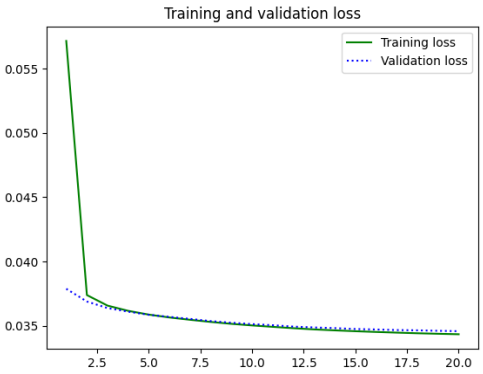
Cuadro 4.5: Resultado de las métricas en el escenario 2 para el modelo LinearAEC

conjunto	**valid**	**mse**	**eqRoutesNumber**	**eqRoutesSize**
train	0.8982	0.0342	0.0013	0.0007
val	0.8745	0.0345	0.001	0.0005
test	0.897	0.0346	0.0	0.0

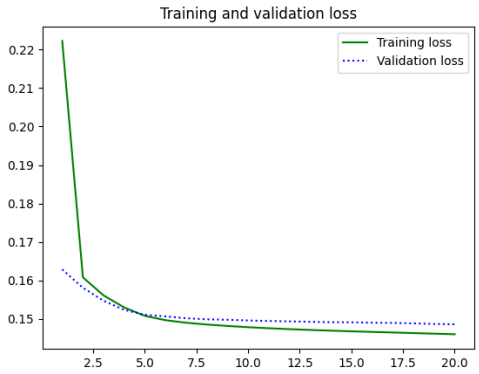
Cuadro 4.6: Resultado de las métricas en el escenario 2 para el modelo VAE

conjunto	**valid**	**mse**	**eqRoutesNumber**	**eqRoutesSize**
train	0.8665	0.0354	0.0319	0.0024
val	0.852	0.0357	0.0245	0.002
test	0.874	0.0357	0.032	0.0015

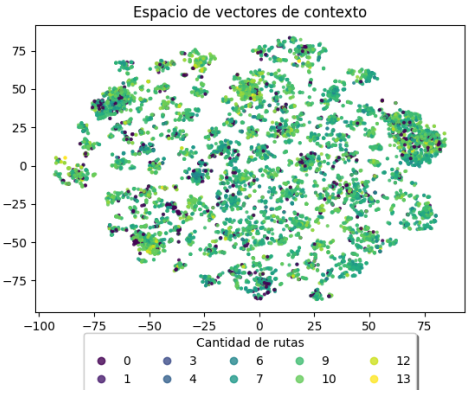
Los gráficos en 4.1.2 muestran el comportamiento de las funciones de pérdida durante el entrenamiento de ambos modelos.



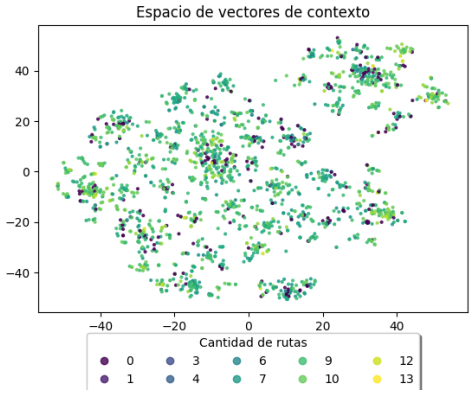
(g) Pérdida en el modelo LinearAECs



(h) Pérdida en el modelo VAE



(i) Espacio latente en el conjunto de entrenamiento del modelo LinearAEC

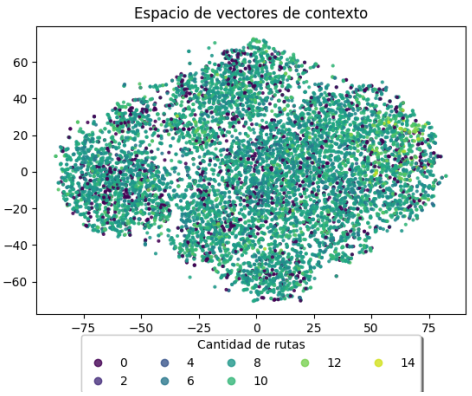


(j) Espacio latente en el conjunto de validación del modelo LinearAEC

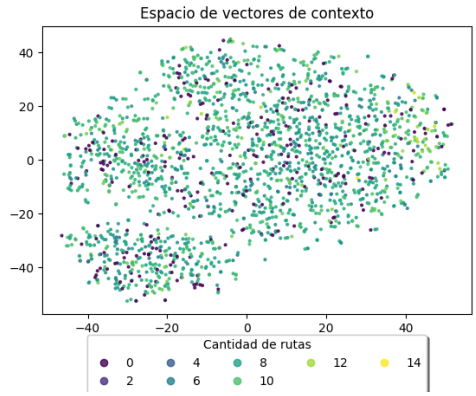
4.1.3. Análisis de resultados

Luego de observar los resultados alcanzados en los escenarios anteriores se puede decir que el comportamiento de ambos modelos fue similar en todos los experimentos. Una de las causas es que poseen una arquitectura de capas y configuración de hiperparámetros semejante.

En los gráficos de las funciones de pérdida se refleja el bajo error característico en los modelos *autoencoders*, con un estilo similar a la pérdida resultante de los trabajos [44, 54]. En el escenario 1, se observa cómo disminuye durante las dos primeras épocas y posteriormente se mantiene moderada.



(k) Espacio latente en el conjunto de entrenamiento del modelo VAE



(l) Espacio latente en el conjunto de validación del modelo VAE

Un valor de error pequeño no significa en este caso que la reconstrucción alcanzada en la salida se corresponda con la entrada. Este planteamieto se corrobora con los valores de las métricas *eqRoutesNumber* y *eqRoutesSize*, que evalúan la similitud entre una solución de entrada y su salida teniendo en cuenta la cantidad de rutas y las longitudes de cada ruta respectivamente. Pues como se observa en los cuadros 4.2, 4.3, 4.5, 4.6, sus valores son insignificantes y no aportan información sobre el parentezco de las soluciones con respecto a las características que examinan.

A pesar de no alcanzar un nivel adecuado de reconstrucción, en todos los escenarios se comprueba un alto grado de generación de soluciones válidas para una cantidad fija de clientes. La métrica *valid* confirma este efecto. Esta medida arroja en todos los escenarios, y para cada conjunto de datos (*train*, *valid*, *test*), un porciento de generación mayor que 85%. Este rendimiento reafirma la capacidad de ambas propuestas como modelos generativos, haciendo posible la generación de nuevas soluciones al explorar el espacio continuo conformado.

Este resultado constituye un logro para este trabajo debido a que se comprueba que las soluciones en el espacio discreto se pueden codificar a un vector real de menor dimensión y que estos, con alta probabilidad se decodifican a una posible solución válida. Se dice posible porque, en este trabajo, no se consideran las restricciones implícitas en el tipo de problema, solo interesa representar una solución a partir de un sistema de rutas.

También se puede apreciar en la visualización de los espacios latentes conformados por las soluciones codificadas, una supremacía de puntos con

cantidad de rutas cercana a la cantidad de rutas r de los vectores con los que se entrenaron los modelos. Además, en un mismo modelo se observa que la distribución de puntos en el espacio latente posee una disposición similar. Por ejemplo, para el caso del modelo *LinearAEC* los puntos se organizan formando pequeños conjuntos. A partir de esta interpretación, quizás resulte conveniente explorar ese espacio para descubrir otro tipo de relaciones. Como por ejemplo, determinar si esas agrupaciones están formadas por soluciones con características similares.

Durante esta etapa de experimentación se prepararon dos escenarios distintos para evaluar el desempeño de ambos modelos. Para ello se calcularon las métricas definidas y se visualizaron los espacios latentes correspondientes. Finalmente se concluye con la obtención de un alto grado de reconstrucción de soluciones válidas.

Conclusiones

Con este trabajo se establecen los primeros pasos en la aplicación de algoritmos de aprendizaje automático al problema de enrutamiento de vehículos en la facultad. A partir del estudio realizado sobre los modelos de redes neuronales *autoencoders* se propone una herramienta que permite obtener un espacio de soluciones continuas a partir de soluciones del problema de enrutamiento en un espacio discreto.

Como parte de la solución se formularon dos modelos: *LinearAEC* y *VAE*, cuya esencia es describir una distribución de probabilidad de los puntos que constituyen la codificación continua de las soluciones de entrada. El comportamiento de ambos modelos fue similar en los experimentos realizados. A pesar de no proporcionar una buena reconstrucción de la solución inicial, muestra una efectividad del 85% en la generación de soluciones válidas en el proceso de decodificación. Con este resultado, se evidencia la capacidad de ambas propuestas como modelos generativos.

También se plantea y analiza una forma de representar las soluciones del VRP mediante matrices binarias con posibles valores en el conjunto $\{0,1\}$. En los espacios latentes graficados, se observa un predominio de soluciones con cantidad de rutas cercanas a la cantidad de rutas presente en las soluciones de entrada.

Recomendaciones

Como recomendaciones y trabajos futuros, se propone considerar los siguientes puntos para mejorar los resultados de este trabajo:

- Mejorar el marco donde se realizaron los experimentos para probar modelos más complejos y con mayor cantidad y diversidad de casos entrenantes.
- Utilizar un algoritmo de optimización continua sobre el espacio latente generado con el obetivo de explorarlo y encontrar nuevas soluciones.
- Añadir técnicas de *reinforcement learning* para mejorar el desempeño del modelo.
- Modelar el problema como una arquitectura *sequence-to-sequence* para examinar el uso de redes neuronales recurrentes en estos modelos.
- Entrenar el modelo con otra representación vectorial de las soluciones del VRP.
- Intentar generalizar los modelos para permitir la entrada de soluciones con distinta cantidad de clientes, o sea, que el proceso de aprendizaje no dependa de n.

Bibliografía

[1] Suboptimal travelling salesman problem (tsp) solver. (Citado en la página 43).

[2] Rowel Atienza. *Advanced Deep Learning with Keras*. Packt Publishing Ltd., 2018. (Citado en las páginas 33, 34, 35 y 36).

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. (Citado en las páginas 8, 32, 41 y 42).

[4] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *arXiv preprint arXiv:2003.05991*, 2020. (Citado en la página 30).

[5] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016. (Citado en la página 42).

[6] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. (Citado en las páginas 2 y 43).

[7] Daniela González Beltrán. Generación automática de gramáticas para la obtención de infinitos criterios de vecindad en el problema de enrutamiento de vehículos. 2019. (Citado en la página 2).

[8] Yoshua Bengio. Learning deep architectures for AI. *Found. Trends Mach. Learn.*, 2(1):1–127, 2009. (Citado en la página 2).

[9] Yoshua Bengio, Eric Laufer, Guillaume Alain, and Jason Yosinski. Deep generative stochastic networks trainable by backprop. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 226–234. JMLR.org, 2014.

[10] Luefeng Chen, Mengtian Zhou, Wanjuan Su, Min Wu, Jinhua She, and Kaoru Hirota. Softmax regression based deep sparse autoencoder network for facial emotion recognition in human-robot interaction. *Information Sciences*, 428:49–61, 2018. (Citado en la página 32).

[11] Xinwei Chen, Marlin W. Ulmer, and Barrett W. Thomas. Deep q-learning for same-day delivery with a heterogeneous fleet of vehicles and drones. *CoRR*, abs/1910.11901, 2019. (Citado en la página 43).

[12] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., 2018. (Citado en las páginas 9, 10, 14, 15, 16, 34, 35, 38 y 49).

[13] Francois Chollet et al. Keras. (Citado en la página 53).

[14] Nicos Christofides, Aristide Mingozzi, and Paolo Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming*, 20:255–282, 1981. (Citado en la página 1).

[15] Przemysław Czuba and Dariusz Pierzchała. Machine learning methods for solving vehicle routing problems. 01 2021. (Citado en la página 1).

[16] Andrew M. Dai and Quoc V. Le. Semi-supervised sequence learning. *CoRR*, abs/1511.01432, 2015. (Citado en la página 2).

[17] PaoloVigo George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959. (Citado en la página 1).

[18] Li Deng and Dong Yu. Deep learning: methods and applications. *Foundations and trends in signal processing*, 7(3–4):197–387, 2014. (Citado en la página 15).

[19] Carl Doersch. Tutorial on variational autoencoders. *CoRR*, abs/1606.05908, 2016. (Citado en las páginas 19, 21 y 37).

[20] Weishan Dong, Ting Yuan, Kai Yang, Changsheng Li, and Shilei Zhang. Autoencoder regularized network for driving style representation learning. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1603–1609. ijcai.org, 2017. (Citado en la página 32).

[21] Claudio Gambella, Bissan Ghaddar, and Joe Naoum-Sawaya. Optimization problems for machine learning: A survey. *European Journal of Operational Research*, 290, 08 2020. (Citado en la página 1).

[22] José Fernando Gonçalves and Mauricio G. C. Resende. A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Comput. Oper. Res.*, 39(2):179–190, 2012. (Citado en la página 3).

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. (Citado en las páginas 2, 8, 9, 16, 18, 31, 33, 34 y 39).

[24] Geoffrey E. Hinton, Peter Dayan, Brendan J. Frey, and R M Neal. The "wake-sleep.ªlgorithm for unsupervised neural networks. *Science*, 1995. (Citado en la página 33).

[25] Geoffrey E. Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504 – 507, 2006. (Citado en la página 2).

[26] Geoffrey E. Hinton and Richard S. Zemel. Autoencoders, minimum description length and helmholtz free energy. In *NIPS*, 1993. (Citado en las páginas 2 y 30).

[27] John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79 8:2554–8, 1982.

[28] André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. *CoRR*, abs/1911.09539, 2019. (Citado en la página 43).

[29] André Hottung, Bhanu Bhandari, and Kevin Tierney. Learning a latent search space for routing problems using variational autoencoders. In

9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021. (Citado en las páginas 32 y 40).

[30] Waldy Joe and Hoong Chuin Lau. Deep reinforcement learning approach to solve dynamic vehicle routing problem with stochastic customers. In *ICAPS*, 2020. (Citado en la página 2).

[31] Elias B. Khalil, Bistra Dilkina, George L. Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 659–666. ijcai.org, 2017. (Citado en la página 2).

[32] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, 2017.

[34] Gilbert Laporte and Yves Nobert. Exact algorithm for the vehicle routing problem. *Ann Discr Math*, 31, 12 1987. (Citado en la página 1).

[35] Yann LeCun. Phd thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models). 1987. (Citado en las páginas 2 y 30).

[36] Henrique Lemos, Marcelo O. R. Prates, Pedro H. C. Avelar, and Luís C. Lamb. Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems. *CoRR*, abs/1903.04598, 2019.

[37] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International conference on artificial neural networks*, pages 52–59. Springer, 2011. (Citado en la página 32).

[38] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level

control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015.

[39] Camila Pérez Mosquera. Primeras aproximaciones a la búsqueda de vecindad infinitamente variable. 2017. (Citado en la página 2).

[40] M. Nazari, Afshin Oroojlooy, Lawrence V. Snyder, and Martin Takác. Deep reinforcement learning for solving the vehicle routing problem. *ArXiv*, abs/1802.04240, 2018. (Citado en la página 2).

[41] MohammadReza Nazari, Afshin Oroojlooy, Lawrence V. Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 9861–9871, 2018. (Citado en las páginas 2, 42 y 43).

[42] Marcelo O. R. Prates, Pedro H. C. Avelar, Henrique Lemos, Luís C. Lamb, and Moshe Y. Vardi. Learning to solve np-complete problems: A graph neural network for decision TSP. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4731–4738. AAAI Press, 2019.

[43] Malte Probst. The set autoencoder: Unsupervised representation learning for sets. 2018. (Citado en la página 42).

[44] Olivér Rákos, Szilárd Aradi, Tamás Bécsi, and Zsolt Szalay. Compression of vehicle trajectories with a variational autoencoder. *Applied Sciences*, 10:6739, 2020. (Citado en las páginas 39 y 59).

[45] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Icml*, 2011. (Citado en la página 32).

[46] Héctor Felipe Massón Rosquete. Exploración de vecindades grandes en el problema de enrutamiento de vehículos usando técnicas estadísticas. 2020. (Citado en la página 2).

[47] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020. (Citado en las páginas 11, 30 y 43).

[48] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*, pages 4–11, 2014. (Citado en la página 32).

[49] José Jorge Rodríguez Salgado. Una propuesta para la evaluación automática de soluciones vecinas en un problema de enrutamiento de vehículos a partir del grafo de evaluación de una solución. 2020. (Citado en la página 2).

[50] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016. (Citado en la página 2).

[51] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014. (Citado en las páginas 8, 18, 32, 40 y 41).

[52] Paolo Toth and Daniele Vigo. An exact algorithm for the vehicle routing problem with backhauls. *Transportation Science*, 31:372–385, 11 1997.

[53] Paolo Toth and Daniele Vigo. *The vehicle routing problem*. SIAM, 2002. (Citado en las páginas 1, 23 y 24).

[54] Dmitry Utyamishev and Inna Partin-Vaisband. Progressive VAE training on highly sparse and imbalanced data. *CoRR*, abs/1912.08283, 2019. (Citado en las páginas 39 y 59).

[55] José Manuel Vera and Andres G. Abad. Deep reinforcement learning for routing a heterogeneous fleet of vehicles. *CoRR*, abs/1912.03341, 2019. (Citado en las páginas 2 y 43).

[56] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *CoRR*, abs/1511.06391, 2016. (Citado en la página 42).

[57] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2692–2700, 2015. (Citado en las páginas 2, 18, 40, 42 y 43).

[58] Ruoqi Wei and Ausif Mahmood. Recent advances in variational autoencoders with representation learning for biomedical informatics: A survey. *IEEE Access*, 9:4939–4956, 2021. (Citado en las páginas 30 y 32).

[59] Zhihao Xing and Shikui Tu. A graph neural network assisted monte carlo tree search approach to traveling salesman problem. *IEEE Access*, 8:108418–108428, 2020. (Citado en la página 2).

[60] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. 09 2014. (Citado en la página 18).

[61] Nianyin Zeng, Hong Zhang, Baoye Song, Weibo Liu, Yurong Li, and Abdullah M Dobaie. Facial expression recognition via learning deep sparse autoencoders. *Neurocomputing*, 273:643–649, 2018. (Citado en la página 32).

[62] Ke Zhang, Meng Li, Zhengchao Zhang, Xi Lin, and Fang He. Multi-vehicle routing problems with soft time windows: A multi-agent reinforcement learning approach. *CoRR*, abs/2002.05513, 2020. (Citado en la página 43).

[63] Chong Zhou and Randy C Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 665–674, 2017. (Citado en la página 32).