UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

Departamento de ingeniería informática

HERRAMIENTA DE SOFTWARE PARA PREDECIR EL AGRUPAMIENTO DE BUSES URBANOS

PABLO JAVIER SALINAS CABAÑAS

Profesor Guía: Víctor Parada Daza

Trabajo de titulación presentado en conformidad a los requisitos para obtener el título de Ingeniero Civil en Informática

Santiago – Chile

2017

RESUMEN

Las empresas operadoras de servicio de buses del sistema de transporte público urbano de la ciudad de Santiago de Chile, han debido pagar grandes montos en multas y descuentos a sus ingresos anuales. Esto se debe al incumplimiento de los rangos aceptables de los índices de la regularidad (ICR) y frecuencia (ICF) del servicio, que mensualmente entrega el Centro de Monitoreo de Buses (CMB) del Directorio de Transporte Público Metropolitano (DTPM). Se ha detectado que este problema se debe principalmente al fenómeno de agrupamiento o *bunching* de buses. Este fenómeno se manifiesta cuando un bus transita junto a otro bus del mismo servicio, separados por un *headway* menor a una cierta fracción del *headway* con el que inician. El *headway*, consiste en la diferencia de tiempo entre un par de buses en alcanzar el mismo punto de una ruta fija. Este trabajo tiene como objetivo diseñar, construir y probar un algoritmo para la predicción de *bunching* de buses, basado en redes neuronales recurrentes, utilizando datos históricos de un servicio de buses, con tal de servir de ayuda al momento de tomar acciones preventivas y de mitigación. Se implementan 3 modelos de redes neuronales artificiales, para predecir los tiempos de viaje de un conjunto de buses de un servicio. Se generan combinaciones de pares de viajes en una ventana de tiempo específica, para determinar la ocurrencia de eventos de *bunching* entre ellos. Los resultados muestran que los modelos de redes neuronales recurrentes presentan una precisión mayor al momento de predecir estos eventos de *bunching*, con relación a redes neuronales MLP y modelos basados en promedios históricos. Este proyecto de tesis es de carácter interno a la Universidad de Santiago de Chile.

Palabras clave: redes neuronales artificiales, agrupamiento de buses, redes neuronales recurrentes, predicción de tiempos de viaje, transporte urbano.

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

CAPÍTULO 1. INTRODUCCIÓN

1.1 ANTECEDENTES Y MOTIVACIÓN

En una ciudad existe una necesidad inherente de la población por movilizarse entre diferentes regiones de esta. Los viajes que se realizan dentro de una ciudad están determinados por polos de atracción y generación de viajes. Los polos de atracción suelen constituirse por lugares donde existe una gran afluencia de gente como son las universidades, colegios o centros comerciales mientras que los polos de generación se originan en los domicilios de las personas. Esta necesidad, requiere que exista un sistema de transporte urbano, en donde se puedan realizar estos traslados de forma organizada y eficiente. Este sistema, debe considerar los diferentes modos de transporte existentes, como son los vehículos privados, taxis, buses, metrotrenes, entre otros. Además, estos modos de transporte deben funcionar de manera coordinada e integrada, para conseguir la eficiencia y eficacia en términos operacionales y de costo.

En muchas ciudades de países en vías de desarrollo, los buses urbanos constituyen el principal modo de transporte utilizado por las personas. Por ejemplo, en la ciudad de Santiago de Chile, el 50,9% de los viajes motorizados son realizados por el transporte público colectivo y el 49,1% por automóviles particulares. Además, los buses urbanos concentran la mayor cantidad de viajes realizados en el transporte público con el 78%, que corresponde aproximadamente a 4 millones de viajes diarios. (Ministerio de Transportes y Telecomunicaciones, 2015).

El sistema de transporte colectivo urbano consta principalmente de tres actores: los operadores del servicio, las autoridades públicas y los usuarios. Los operadores, son los encargados de prestar el servicio de transporte, cumpliendo con los requerimientos establecidos por la planificación del sistema, y sopesando entre los costos de operación y la calidad del servicio prestado. Las autoridades públicas, se encargan de la planificación del sistema, la reglamentación y fiscalización de la operación. Los usuarios, buscan satisfacer sus necesidades de desplazamiento a través de la ciudad, además de buscar realizarlo con comodidad, seguridad, puntualidad, rapidez, bajo costo, entre otros aspectos.

En el contexto de Santiago, el Directorio de Transporte Público Metropolitano (en adelante DTPM) es el encargado de velar por la calidad de servicio que realizan empresas operadoras del sistema de transporte de buses, llamado Transantiago. Se establece en los contratos de cada operador, el cumplir ciertos índices relacionados con la frecuencia y regularidad del servicio. Estos índices son evaluados mes a mes, para cada operador, según los datos que arroja el Centro de Monitoreo de Buses (en adelante CMB) (Directorio de Transporte Público Metropolitano, 2018), el cual monitorea en tiempo real el comportamiento de los buses en cada una de las rutas. Según lo valores que arrojen estos índices, se establecen descuentos al ingreso que perciben los

operadores, donde por cada minuto que posean los operadores por desfases, tiempos de espera en exceso e incidentes, se descuentan hasta 0.01 UF del ingreso mensual. Además, se establecen rangos de cumplimiento para cada índice, donde un bajo cumplimiento acarrea multas que van desde 101 UF hasta 2500 UF (véase Anexo A). Las multas que han sido aplicadas a los operadores, por incumplimiento de los índices de regularidad han significado hasta un 5% del ingreso anual de las empresas (Epysa Club, 2015) . El año 2015, las empresas de transportes Express y Subus tuvieron que pagar cerca de $ 6 mil millones y $ 5 mil millones de descuentos, respectivamente. El año 2014, la empresa de transportes Alsacia tuvo que pagar $4.776 millones por bajo cumplimiento de los indicadores de frecuencia y regularidad.

1.2 DESCRIPCIÓN DEL PROBLEMA

Las empresas operadoras tienen la necesidad de disminuir las multas y descuentos debido al incumplimiento de los índices de regularidad del servicio. Sin embargo, para esto es necesario dar cuenta de los factores o fenómenos que afectan a la regularidad del servicio, para luego poder intentar plasmar cambios en el programa de operación y tomar medidas de mitigación en caso de que estos fenómenos se presenten. Desde la propia concepción de los sistemas de transportes urbanos, se ha determinado que el fenómeno de agrupamiento o *bunching* de buses es la principal causa de los problemas de regularidad de los buses urbanos (Turnquist & Bowman, 1980).

El fenómeno de *bunching* de buses urbanos, se define como el intervalo de tiempo en que un bus transita junto a otro bus del mismo servicio, separados por un *headway* menor a una cierta fracción del *headway* programado. El *headway* es la diferencia de tiempo que existe entre 2 buses que recorren la misma ruta, en llegar al mismo punto del recorrido, entendiéndose también como un desfase de sus tiempos de viaje. El *headway* programado, es determinado por el programa operativo de cada empresa operadora de buses, donde se detallan distintos tiempos según el servicio, el sentido (ida o regreso) y el periodo de la semana. La determinación de estos valores depende de la demanda de usuarios estimada para cada intervalo horario.

Desde la perspectiva de los operadores de buses, es de suma importancia disminuir las multas por bajo cumplimiento de los índices de calidad de servicio. Por lo tanto, es necesario poder predecir la ocurrencia del agrupamiento entre los buses de un mismo servicio, para poder así apoyar a los Protocolos de Acción ante Contingencias (PAC). Esto ayudaría a minimizar los tiempos en que la calidad de servicio no sea aceptable, beneficiando a los usuarios. La predicción del agrupamiento de buses, en un intervalo de tiempo futuro, se puede efectuar mediante el procesamiento de datos históricos del comportamiento del servicio.

1.3 SOLUCIÓN PROPUESTA

1.3.1 Características de la solución

La solución consiste en un algoritmo, que permite predecir la ocurrencia de *bunching* entre un par de buses del mismo recorrido, a partir de datos parciales del recorrido de los buses. Es posible determinar entre qué paraderos va a ocurrir cada evento de *bunching*. Para ello se implementan 4 modelos, con distintos enfoques a la solución: una red neuronal *feedforward*, dos redes neuronales recurrentes y un modelo de regresión basado en promedios históricos. Estos modelos permiten predecir los tiempos de viaje de un bus, dada una secuencia de tiempos de viaje parcial. Luego, estas predicciones son aplicadas a un par de buses en tránsito, de manera de calcular una secuencia de *headway* o desfase de sus tiempos de viaje en cada paradero. Sobre esta secuencia, se aplica un algoritmo de detección de *bunching* para cada paradero predicho.

1.3.2 Propósito de la solución

El propósito de la solución es predecir, los eventos de *bunching* de los buses de un servicio específico, con tal de servir de ayuda a una empresa operadora del servicio u organismo regulador, con el fin de tomar acciones correctivas. Éstas, tienen el propósito de disminuir los costos por multas por incumplimiento del intervalo programado, además de mejorar la calidad de servicio de cara a los usuarios.

1.4 OBJETIVOS Y ALCANCES DEL PROYECTO

1.4.1 Objetivo general

Diseñar, construir y probar un algoritmo para la predicción de *bunching* de buses, basado en redes neuronales recurrentes, utilizando datos históricos de un servicio de buses.

1.4.2 Objetivos específicos

Los objetivos específicos son los siguientes:

- Inspección crítica del estado del arte del problema**.**
- Diseñar una red neuronal capaz de predecir el movimiento de un bus en un intervalo de tiempo futuro.
- Implementar y entrenar la red neuronal con datos proveniente del mundo real.
- Evaluar el desempeño de la red con datos del mundo real.
- Diseñar e implementar un algoritmo de detección de agrupamiento, procesando la salida de la red neuronal.

1.4.3 Alcances

Se limita a entrenar y probar los modelos implementados con datos histórico de los tiempos de viaje de un servicio de buses específico, realizando una separación entre los datos utilizados para entrenamiento y los datos utilizados para pruebas. En este sentido, no se pretende realizar pruebas de los sistemas implementados en entornos reales, donde sea posible utilizar datos en tiempo real para realizar predicciones de los tiempos de viajes de los buses de un servicio actualmente en funcionamiento. También, no se pretende abordar las ventajas reales y las formas de uso de los modelos propuestos en un entorno real.

1.5 METODO DE TRABAJO Y HERRAMIENTAS UTILIZADAS

A continuación, se presenta el método de trabajo para la realización de este proyecto:

- En primer lugar, se realiza una investigación crítica del estado del arte, para conocer las mejores soluciones del problema de predicción de tiempos de viaje de buses urbanos, además de los algoritmos y enfoques más idóneos para la detección de agrupamiento de buses.
- En segundo lugar, se realiza el diseño arquitectural de la solución, donde se especifican los distintos componentes de software a desarrollar, principalmente respecto a la arquitectura de la red neuronal a implementar.
- En tercer lugar, se realiza una obtención y transformación de los datos de GPS de un recorrido de buses de la ciudad de Santiago de Chile, de manera de puedan ser utilizados directamente por la red neuronal como datos de entrada.
- En cuarto lugar, se construye la red neuronal, siguiendo las características estipuladas en la etapa de diseño, se entrena y se prueba con los datos correspondientes, siguiendo el procedimiento estipulado en la fase de diseño.
- En quinto, se implementa el algoritmo de detección de agrupamiento de buses, previamente diseñado, de manera que este pueda procesar la salida de la red neuronal.
- Finalmente, se evalúa y prueba la solución completa, según las métricas de evaluación presentadas en la sección de evaluación de la solución.

Las herramientas de software a utilizar para el desarrollo de la Memoria de Título son las siguientes:

- Microsoft Office 365 Pro Plus: herramienta de ofimática utilizada para la escritura del informe de la Memoria de Título, documentación relacionada, planillas de cálculo para

registrar los resultados de la fase de entrenamiento y prueba de las implementaciones, generación de gráficos y carta Gantt del proyecto.

- Theano: paquete para el lenguaje Python utilizado como *backend* para la implementación y manejo de las redes neuronales.
- Keras: paquete para el lenguaje Python utilizado para la construcción, entrenamiento y prueba de las implementaciones de redes neuronales.
- Numpy: paquete para el lenguaje Python, que implementa funciones matemáticas vectoriales y matriciales de alto nivel, utilizado para manejar los conjuntos de datos y realizar operaciones con ellos.
- Sklearn: paquete para el lenguaje Python, que implementa funciones propias del área del *machine learning,* que es utilizado para manejar eficientemente el cálculo de las métricas de rendimiento de los modelos implementados.
- Pandas: paquete para el lenguaje Python, utilizado para importar y filtrar de manera eficiente *datasets* con grandes volúmenes de datos.
- Anaconda: gestor de paquetes y entornos de desarrollo para el lenguaje Python.
- Sublime Text 2: editor de texto utilizado para la construcción y edición del código de generación y manejo de los *dataset*, entrenamiento y prueba de las redes neuronales, detección y predicción del agrupamiento de buses, entre otros.

En cuanto a las herramientas de hardware, se va a utilizar un computador *notebook*, con un sistema operativo Ubuntu 17.04 de 64 bits, procesador Intel® Core(TM) I5-7200U (2.7 GHz), memoria RAM de 12 GB, disco duro (HDD) de 1 TB y una tarjeta de video NVIDIA GeForce 940MX.

1.6 ORGANIZACIÓN DEL DOCUMENTO

El presente documento consta de distintos capítulos que dan cuenta del trabajo realizado. Primero, un capítulo de marco teórico y análisis crítico del Estado del Arte, donde se abordan los aspectos teóricos necesarios para el entendimiento del problema a estudiar y la revisión de la literatura existente. En segundo lugar, un capítulo de diseño de la solución, donde se explica en detalle los modelos implementados y el método de trabajo utilizado. Posteriormente, un capítulo de resultados y discusiones, donde se presentan los resultados de las pruebas realizadas a los distintos modelos implementados además de un análisis de los fenómenos observados. Luego, un capítulo de conclusión donde se evalúa el cumplimiento de los objetivos, además de extraer análisis finales respecto a los resultados obtenidos. Adicionalmente, el documento consta con un

glosario, para un rápido entendimiento de los conceptos básicos que se abordan, una sección donde se detallan las referencias utilizadas a lo largo del documento, y un anexo donde se detalla la estructura de índices de cumplimiento de servicios de buses junto con las multas y descuentos asociados

CAPÍTULO 2. MARCO TEÓRICO

2.1 ASPECTOS TEÓRICOS

2.1.1 *Machine Learning*

Machine learning o aprendizaje de máquinas, es el campo de las ciencias de la computación que estudia la capacidad de los sistemas computacionales de "aprender" a realizar tareas específicas. Estos sistemas permiten resolver 2 tipos de problemas computacionales: problemas de regresión y problemas de clasificación (Samuel, 1959). Los problemas de regresión permiten predecir fenómenos a partir de información del pasado, mientras que los problemas de clasificación asignan categorías o clases a un conjunto de datos. Este proceso de aprendizaje consiste típicamente en la resolución de un problema de optimización que permita minimizar el error en las clasificaciones o predicciones realizadas al utilizar los datos de ejemplo, ajustando distintos parámetros internos de un modelo computacional previamente definido. Algunos de estos modelos son los árboles de decisión, las máquinas de vectores de soporte, las redes neuronales artificiales, entre otros. La arquitectura del modelo computacional y los valores de sus parámetros determinan la capacidad del sistema de desempeñarse la tarea encomendada. Esta característica difiere del desarrollo los algoritmos que estén diseñados explícitamente para resolver un problema en específico.

Los distintos algoritmos de *machine learning* se pueden clasificar principalmente en 3 categorías: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. El aprendizaje supervisado consiste en la retroalimentación de un algoritmo, en su proceso de entrenamiento, respecto a los valores de salida esperados al ser ingresados ciertos datos de entrada. Esto permite al algoritmo minimizar el error de las salidas que entrega, encontrando los patrones en los datos de entrada que permitan generar la salida correcta. El aprendizaje no supervisado solo utiliza los datos de entrada al algoritmo, para reconocer patrones y estructuras en estos. El aprendizaje por refuerzo consiste en la retroalimentación del algoritmo en el proceso de entrenamiento, respecto a respuestas del mundo real de las acciones previamente realizadas por el algoritmo. De esta manera, el algoritmo debe realizar las acciones que maximicen el beneficio que genere en el mundo real, según el contexto del problema estudiado.

2.1.2 Redes neuronales artificiales

Las redes neuronales artificiales son modelos matemáticos y computacionales provenientes del área de *machine learning*, que son utilizados en el contexto de aprendizaje supervisado y no supervisado para resolver problemas de regresión y clasificación. Estas redes están constituidas por unidades llamadas neuronas artificiales y sus interconexiones, que están inspiradas en las neuronas del cerebro humano y las sinapsis que ocurren entre ellas. La red se puede representar

a través de un grafo dirigido *G(V,A)*, donde *V* representa el conjunto de neuronas, y *A* el conjunto de interconexiones de la red. Cada neurona artificial posee una función matemática de activación, que simula la activación de un impulso entre neuronas biológicas. La salida de cada neurona artificial corresponde a la intensidad del impulso a transmitir, donde se aplica una función de activación sobre cada impulso de entrada, ponderando cada uno de manera distinta al momento de constituir el impulso de salida de la neurona (Baier, 2016).

El tipo de redes neuronales artificiales más utilizado en es el perceptrón multicapa (MLP) (Wilamowski, 2009). Es un tipo de red *feedfordward*, donde los impulsos van en una sola dirección a través de las neuronas, sin formar ciclos. Esta arquitectura de red está formada por capas de neuronas, en donde cada neurona está conectada con la salida de cada neurona de la capa anterior. En la Figura 2.1 se puede apreciar que la red se compone de una capa de entrada que recoge las variables de entrada, una capa oculta o intermedia que recoge los valores de las capas de entrada, y una capa de salida, que entrega un único vector de salida de la red neuronal. Es común que pueda existir más de una capa oculta.

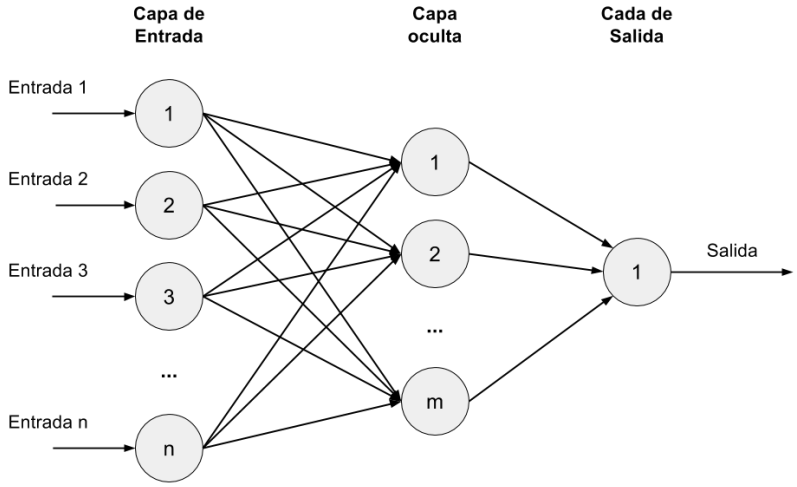


Figura 2.1. Arquitectura de una red neuronal MLP.
Fuente: Elaboración propia, 2017.

Teniendo en cuenta que los impulsos de entrada de una neurona se pueden representar como un vector X, su salida h está dada por la ecuación 2.1, donde se aplica la función de activación f a la sumatoria de la multiplicación de cada la variable de entrada X_i, por el peso interno correspondiente w_i. Este proceso es posible verlo de manera gráfica también en la Figura 2.2. Esta salida, se propaga a través de la red, al constituir parte de la entrada de las neuronas subsecuentes en el grafo de la red neuronal, hasta que cada impulso se convierta en un conjunto de impulsos de salida de la red en su totalidad. Entonces, la salida de una red neuronal está determinada por los valores de sus variables de entrada, y sus "parámetros", es decir, en el patrón

de interconexiones entre las neuronas, los pesos internos de cada neurona y la función de activación que estas posean.

$$h = f(\sum_{i=1}^{i=\#X} X_i * w_i) \tag{2.1}$$

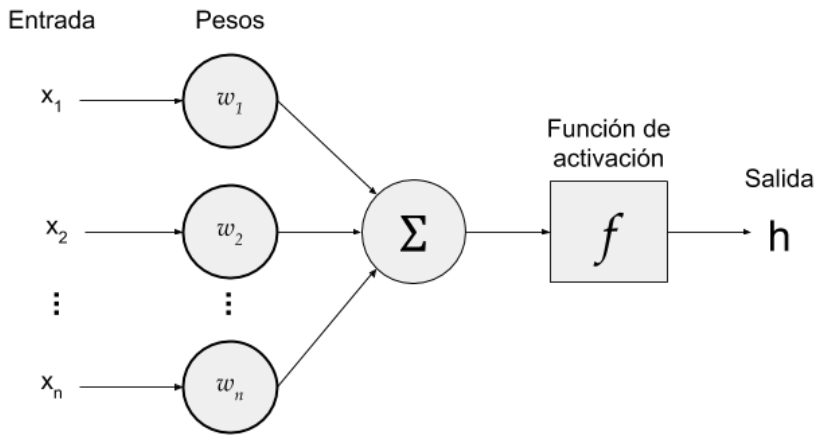


Figura 2.2: Diagrama del funcionamiento de una neurona artificial.
Fuente: Elaboración propia, 2017.

El conjunto de pesos internos de una red neuronal determina su capacidad para resolver un problema de clasificación o regresión específico. Por lo tanto, se debe realizar un proceso de optimización de los valores de estos pesos. Este proceso, consiste en la resolución de un problema de optimización no lineal, que busca minimizar el error producido por la salida de la red, con respecto a la salida esperada (Hopfield, 1985). Para esto se tiene un conjunto de datos de entrenamiento del problema estudiado, donde se conocen las variables de entrada y la salida correspondiente, que debe la red neuronal debe aprender a generar a partir de estas variables de entrada. Se utiliza una función objetivo no lineal para calcular el error que genera la salida de la red neuronal, comúnmente el error cuadrático medio o el error absoluto medio, entre otros.

El método utilizado para resolver el problema de optimización es el gradiente estocástico descendiente (SGD). Este es un método de optimización de primer orden, que utiliza la dirección del opuesto del gradiente de la función objetivo con respecto a cada peso w_{ij} de la red neuronal, para ir disminuyendo paso a paso el valor de la función objetivo. Para cada caso del *dataset* del problema a estudiar, se ingresan de manera ordenada las variables de entrada a la red, recogiendo la salida de la red, y junto con salida esperada, se calcula el valor de la función objetivo. Con este valor se calcula el gradiente con respecto a cada peso de la red, en un proceso llamado *backpropagation* (Rumelhart, Hinton, & Williams, 1986). Para cumplir esta tarea, se debe hacer uso de la regla de la cadena de manera que el cálculo del gradiente se va realizando desde los pesos de la última neurona de la red hasta las neuronas que reciben directamente los datos

de entrada. Luego, un peso *w* cualquiera de la red neuronal se actualiza en el paso *k*, siguiendo la ecuación 2.2, donde la variable α es conocida como la tasa de aprendizaje, que regula la amplitud de los cambios de los pesos en cada paso del entrenamiento. El algoritmo culmina con la determinación de una solución óptima.

$$w^{k+1} = w^k - \alpha \nabla f(w^k) \tag{2.2}$$

La utilización del método SGD para el entrenamiento de las redes neuronales, por sí solo no asegura que la eficiencia ni capacidad de resolver el problema a estudiar sean apropiadas. Esto se debe a que las características del problema a estudiar requieren de un estudio exhaustivo tanto de las variables de entrada y salida, como de los parámetros de la red neuronal. Este fenómeno se hace patente cuando se aplican los mismos parámetros a distintos problemas a estudiar, obteniendo resultados dispares. Por un lado, se debe definir la topología de la red, es decir, el número de capas, el número de neuronas por capa y la forma de interconexión entre cada una de éstas, y por otro las variables de entrada y salida, que representan y caracterizan el fenómeno a estudiar.

A lo largo de los años se han propuestos distintas variantes del modelo SGD, con tal de mejorar la velocidad de convergencia del algoritmo, y evitar converger a mínimos locales. Uno de los métodos más utilizados para entrenamiento en línea es Adaptive Moment Estimation (ADAM), el cual utiliza ponderaciones del promedio del gradiente y el cuadrado del gradiente de cada iteración, con tal de regular de manera adaptativa la magnitud la modificación de los pesos internos de una red neuronal (Kingma & Ba, 2015). Sea *w* un peso cualquiera en la red neuronal, η la tasa de aprendizaje, *L* la función objetivo utilizada, β_1 el factor de olvido del promedio de gradientes y β_2 el factor de olvido del promedio de gradientes al cuadrado, las ecuaciones 2.3 a 2.7 describen el algoritmo de optimización.

La ecuación 2.3 representa una implementación de SGD con *momentum*, que toma en cuenta fuertemente el valor anterior de la propia variable, antes que las oscilaciones del valor de gradiente proveniente de la función objetivo. Se calcula el valor m con respecto a un peso *w* en el tiempo *t+1*, utilizando una ponderación entre el valor de la misma variable en el instante de tiempo anterior y el gradiente de la función objetivo con respecto al peso *w*. Esta ponderación está dada por β_1 que posee un valor entre 0 y 1, que normalmente es cercano a 0,9, para que el aporte del valor del nuevo gradiente calculado no genere oscilaciones muy grandes en los valores.

$$m_w^{t+1} \leftarrow \beta_1 m_w^t + (1 - \beta_1) \nabla_w L^t \tag{2.3}$$

La ecuación 2.4 proviene del algoritmo de optimización Root Mean Square Propagation (RMSProp), que implementa el concepto de *momentum*, pero realizando ponderaciones con respecto al cuadrado del gradiente de la función objetivo. Se calcula el valor *v* con respecto a un peso w de la red neuronal en el tiempo *t+1*, utilizando una ponderación entre el valor de la misma variable en el instante de tiempo anterior y el valor del cuadrado del gradiente del último caso o conjunto de casos de entrenamiento. La ponderación está dada por el valor β_2 que puede tomar valores en el rango entre 0 y 1, aunque generalmente se le asigna un valor mayor a β_2, muy cercano a 1, por ejemplo 0,99.

$$v_w^{t+1} \leftarrow \beta_2 v_w^t + (1-\beta_2)(\nabla_w L^t)^2 \tag{2.4}$$

Las ecuaciones 2.5 y 2.6 normalizan los nuevos valores de m_w y v_w, de manera que se conviertan en vectores unitarios. Cabe destacar que las ecuaciones toman en cuenta a *w* como un peso interno cualquiera en la red neuronal, sin embargo, es aplicable a cualquier topología de red, donde cada variable se convierte en una matriz, con los valores para cada peso de cada nodo de la red neuronal.

$$\hat{m}_w = \frac{m_w^{t+1}}{1-\beta_1} \tag{2.5}$$

$$\hat{v}_w = \frac{v_w^{t+1}}{1-\beta_2} \tag{2.6}$$

La ecuación 2.7, calcula el nuevo valor del peso *w* a partir del valor anterior del mismo peso interno, de las variables m_w, v_w y la tasa de aprendizaje η. Se calcula el valor de $\hat{m}_w$, proveniente del gradiente de la función objetivo, dividido por la raíz cuadrada de la variable $\hat{v}_w$, proveniente del cuadrado del gradiente. Todo lo anterior, multiplicado por η se resta al valor del peso a modificar. Se puede apreciar que, eliminando el valor de $\hat{v}_w$, la ecuación es idéntica a la de SGD. También se puede notar que existe una constante ϵ, que posee un valor pequeño, típicamente 10^{-8}, que previene la división por 0, en la ecuación.

$$w^{t+1} \leftarrow w^t - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w}+\epsilon} \tag{2.7}$$

El establecimiento de los parámetros de la red consiste en un segundo problema de optimización, donde se quiere conseguir la combinación de parámetros y variables que minimice el error de las predicciones, luego de haber realizado el proceso de entrenamiento con estos. Para esta tarea, existen técnicas heurísticas que permiten encontrar una topología adecuada para la red, quitando elementos desde una red sobredimensionada o agregando elementos a una red de una sola neurona, hasta encontrar la arquitectura que entregue un mejor balance entre rendimiento y eficiencia, También, se puede utilizar un algoritmos genético, que permita evolucionar los distintos

elementos de la topología a través de selección, cruzamiento y mutación, con tal de determinar las interacciones y dimensiones adecuadas (Khosravi, Mazloumi, Nahavandi, Creighton, & Van Lint, 2011).

2.1.3 Redes neuronales recurrentes

Una red neuronal recurrente es un tipo de red neuronal artificial, en la cual sus conexiones forman ciclos de retroalimentación dirigidos, que sirven para aprovechar la temporalidad de los datos ingresados (Rumelhart, Hinton, & Williams, 1986). Por lo tanto, estas redes neuronales son idóneas para ser aplicadas a fenómenos cuyos datos generan series de tiempo. Estas redes siguen mayormente los principios de las redes *feedforward*, incluyendo una función de activación y matriz de pesos internos en sus neuronas, siendo compatibles con arquitecturas multicapas. Su particularidad radica en que algunas neuronas se retroalimentan a sí mismas, provocando que su salida dependa no solo de los impulsos de entrada, sino de la salida de la propia neurona en el tiempo anterior. En la Figura 2.3, a la izquierda se aprecia el ciclo de retroalimentación de una neurona n_i de una red neuronal recurrente, y a la derecha, la manera en que este ciclo se desenvuelve a través del tiempo, donde cada entrada x_t es un *timestep* o momento del tiempo distinto y su salida depende de la salida h_{t-1} producida en el timestep anterior.

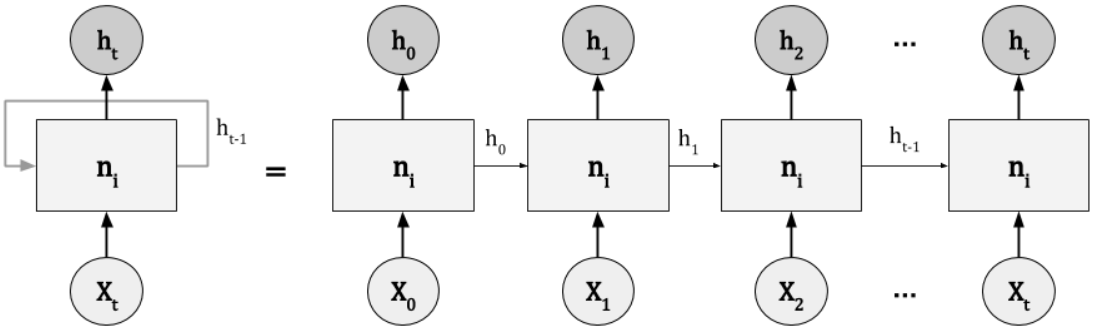


Figura 2.3. Ejemplo de grafo de red neuronal recurrente.
Fuente: Elaboración propia, 2017.

Las redes neuronales recurrentes son las más adecuadas para estudiar fenómenos donde los datos forman series de tiempo. Esto quiere decir que estas redes son adecuadas si las variables del fenómeno poseen una alta autocorrelación a través del tiempo, y por lo tanto el valor de la serie en cierto instante de tiempo, depende fuertemente del valor en el instante de tiempo anterior. Esto se da de manera natural en series donde se muestrean datos a través del tiempo, como los precios de los productos, el tiempo en una cierta región, o los datos de GPS de un vehículo al realizar un viaje. Cabe destacar que todas estas series de tiempo deben ser modeladas adecuadamente, para aprovechar de mejor manera esta dependencia temporal de los valores. Este tipo de redes han sido ampliamente utilizadas para resolver problemas de procesamiento de

lenguaje natural (PLN), como el software de reconocimiento del habla desarrollado por Google, el cual implementa una red neuronal recurrente para reconocer palabras a partir de modelos acústicos, y es utilizado tanto para los servicios de Google Now, tanto en Android, como Google Chrome en PC (Sak, Senior, & Beaufays, 2014).

Sea x_t la entrada a una neurona en el tiempo t, h_{t-1} su salida en el tiempo $t-1$, f su función de activación, W la matriz de pesos internos para el vector de entrada y U la matriz de transición de estados ocultos de la neurona en el tiempo, la salida de la neurona en el tiempo t está definida por (2.8). Esta implementación de red neuronal recurrente fue propuesta por Elman (1990), donde se almacenan los estados ocultos de la red recurrente con el objetivo de encontrar correlaciones entre los eventos que ocurren en el tiempo, incluso separados por más de un *timestep*. Esto es posible debido a que el estado oculto de cierto *timestep* depende de los estados anteriores, configurandose la llamadas dependencias de largo plazo.

$$h_t = f(W x_t + U h_{t-1}) \tag{2.8}$$

Las redes neuronales recurrentes, vistos como sistemas dinámicos, se consideran sistemas no lineales autorregresivos (NAR) (Connor, Atlas, & Martin, 1991). Su connotación de no lineal radica en la utilización de funciones de activación no lineales sobre la salida de las neuronas de la red, representada por la función *f* en (2.8). Este modelo es autorregresivo, debido a que cada neurona utiliza como entrada un conjunto finito de las salidas que registró en *timesteps* anteriores. En el caso de (2.8) se utiliza solo la salida de cada neurona en el *timestep* anterior $\mathrm{h_{t-1}}$, pero es posible generalizar las recurrencias utilizadas según (2.9), donde q es la cantidad de *timestep* que ingresan en un instante de una serie de tiempo discreta, p es la cantidad de recurrencias que utiliza la red neuornal, $\mathrm{W_j}$ es el peso interno correspondiente a la entrada de la serie de tiempo $\mathrm{x_{t-i}}$ y $\mathrm{U_j}$ es el peso interno correspondiente a la recurrencia $\mathrm{h_{t-j}}$.

$$h_t = f(\textstyle\sum_{i=1}^{q} W_i x_{t-i} + \sum_{j=1}^{p} U_j h_{t-j}) \tag{2.9}$$

Según la forma en que se manejan las series de tiempo de entrada y salida en una red neuronal recurrente, es posible definir tres tipos de arquitecturas: uno a uno, uno a muchos, muchos a uno y muchos a muchos (Figura 2.4). Una arquitectura uno a uno, representa una red neuronal convencional, donde para cada entrada se entrega una salida, sin tomar en cuenta el estado oculto de los *timestep* anteriores. En una arquitectura de uno a muchos, se ingresa un solo vector de entrada y la red neuronal genera una secuencia de salida de largo variable, aprovechando la recurrencia de los estados ocultos de sus neuronas en el tiempo. En una arquitectura de muchos a uno, se ingresa una secuencia de timesteps de una serie de tiempo, y la red neuronal va modificando sus estados ocultos para retornar un solo vector de salida. En una arquitectura de

muchos a muchos, existe una secuencia de series de tiempo de entrada, donde para cada uno de sus *timesteps* se genera una salida, formando una secuencia de salida del mismo largo.

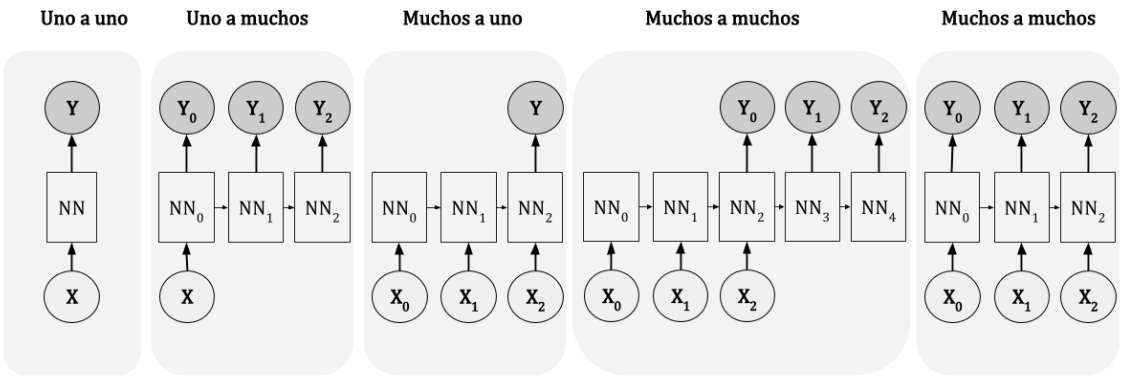


Figura 2.4. Tipos de manejo de secuencias de una red neuronal recurrente.
Fuente: Elaboración propia, 2017.

Para que una red neuronal recurrente pueda ajustar las matrices de pesos de manera que tengan un buen rendimiento, el proceso de entrenamiento debe realizarse de manera coherente a la forma en que va a ser tratada la secuencialidad de los datos, en la salida y entrada de la red neuronal. Esto quiere decir que el proceso de *backpropagation* que comúnmente se realiza en las redes neuronales convencionales, debe adecuarse al contexto de las secuencias de salida y entrada de la red recurrente. Por ejemplo, en una arquitectura de uno a muchos, la actualización de los pesos de la red se debe realizar sobre la función objetivo, evaluada sobre la secuencia completa de salida, la cual proviene de una sola entrada. También, en una arquitectura de muchos a uno, se puede evaluar la función objetivo sobre el único vector de salida, que proviene de una secuencia de valores de entrada. Cabe destacar que también es posible aplicar el concepto de actualización de pesos por lotes de casos en un *dataset*¸ considerando que cada entrada o salida de la red neuronal, puede estar compuesta por secuencias de *timestep*, en vez de un solo vector de salida en caso de las redes convencionales.

En las redes neuronales recurrentes el proceso de *backpropagation* es llamado *backpropagation through time*, debido a que el ajuste de pesos al retro-propagar la función objetivo con respecto a los parámetros de la red, toma en cuenta el estado oculto de cada neurona en el *timestep* anterior (Brownlee, A Gentle Introduction to Backpropagation Through Time, 2017). Al utilizar el gradiente de la función objetivo sobre la función de cada neurona recurrente, no solo se calcula un valor de error por cada peso interno, incluida la matriz U del estado oculto en el *timestep* anterior, sino que, por regla de la cadena, se calcula el error de los pesos de la red neuronal que formaron ese estado oculto. De esta manera, la retro-propagación no solo se realiza desde los nodos de salida hasta la propia entrada de la red neuronal, sino que a través de los distintos

timestep de la serie de tiempo. La Figura 2.5, muestra la manera en que se realiza la retro-propagación de la función objetivo durante el proceso de ajuste de pesos en una red neuronal recurrente, donde x_t es el vector de entrada de la red en el *timestep t*, E_t es la función de error de la salida de la red neuronal en dicho *timestep t* y s_t es el valor del estado oculto una neurona de la red en el *timestep t*.

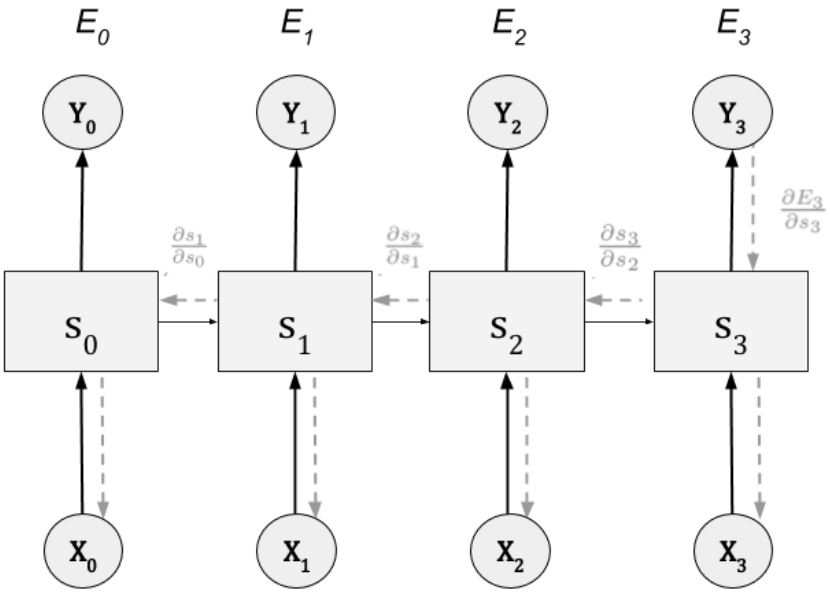


Figura 2.5. Diagrama de Backpropagation Through Time.
Fuente: Elaboración propia, 2017.

Las redes neuronales recurrentes poseen mayores dificultades en el rendimiento del proceso de *backpropagation* debido a que este se desenvuelve a través de los distintos *timestep*, generando largas cadenas de multiplicaciones de valores. Para esto existen implementaciones de algoritmos de entrenamiento que truncan la manera en que retro-propaga el gradiente, dejando un límite de *timestep* por los que realizar los cálculos. Esto mejora el rendimiento en tiempo de ejecución, debido a que disminuye la cantidad de cálculos a realizar, y en la memoria a almacenar, debido a que solo necesita almacenar los estados ocultos de los *timestep* donde se van a realizar cálculos. Sin embargo, este proceso limita la capacidad de la red neuronal de detectar patrones en las series de tiempo a largo plazo.

Las principales dificultades del entrenamiento de redes neuronales recurrentes, recae en la posible ocurrencia de los fenómenos de desvanecimiento o explosión de gradiente. El primero, ocurre cuando al calcular el gradiente de la función objetivo con respecto a algún peso interno, la multiplicación de los valores del gradiente de las neuronas subsecuentes es menor a 1, por lo que su gradiente va a tender a disminuir con respecto a sus neuronas subsecuentes, generando

una tendencia a 0 a medida que se retro-propaga en la red neuronal. El segundo caso es similar, salvo que cuando la multiplicación de los valores de gradiente que conforman un cierto peso interno es mayor a 1, el gradiente de los pesos de las neuronas anteriores tiende a aumentar a medida que se realiza la retro-propagación (Nielsen, 2015). Ambos fenómenos no permiten realizar una convergencia del método de optimización, por lo que se han propuesto distintos mecanismos y modelos para evitar este problema.

2.1.4 Redes recurrentes LSTM

Las redes neuronales recurrentes *Long Short-Term Memory* (LSTM) fueron propuestas por Hochreiter & Schmidhuber (1997), con el objetivo de evitar el problema de explosión y desvanecimiento de gradiente. Su nombre proviene de la idea de manejar memoria de los patrones que se generan en las series de tiempo, a través de la recurrencia de los estados internos, a corto y largo plazo. Este tipo de red neuronal recurrente utiliza dos tipos de recurrencias para lograr su cometido, siendo la salida de la propia neurona en el *timestep* y el estado interno de la celda en el *timestep* anterior.

En una neurona o bloque LSTM, ingresa el estado oculto anterior de la misma celda h_{t-1} y la entrada actual x_t, y a través de compuertas y celdas internas que interactúan entre ellas se obtiene la salida oculta h_t. Cada compuerta posee una matriz de pesos internos los cuales son entrenados como cualquier otro parámetro de la red neuronal. Como es posible ver en la Figura 2.6, estos componentes son la celda de entrada, la compuerta de entrada, la compuerta de olvido, la celda interna y la compuerta de salida. Las compuertas utilizan funciones de activación sigmoidal, con rango de valores entre 0 y 1, que permiten controlar el paso de un dato, mientras que las celdas comúnmente utilizan funciones de activación de tangente hiperbólica, con un rango de valores entre -1 y 1.

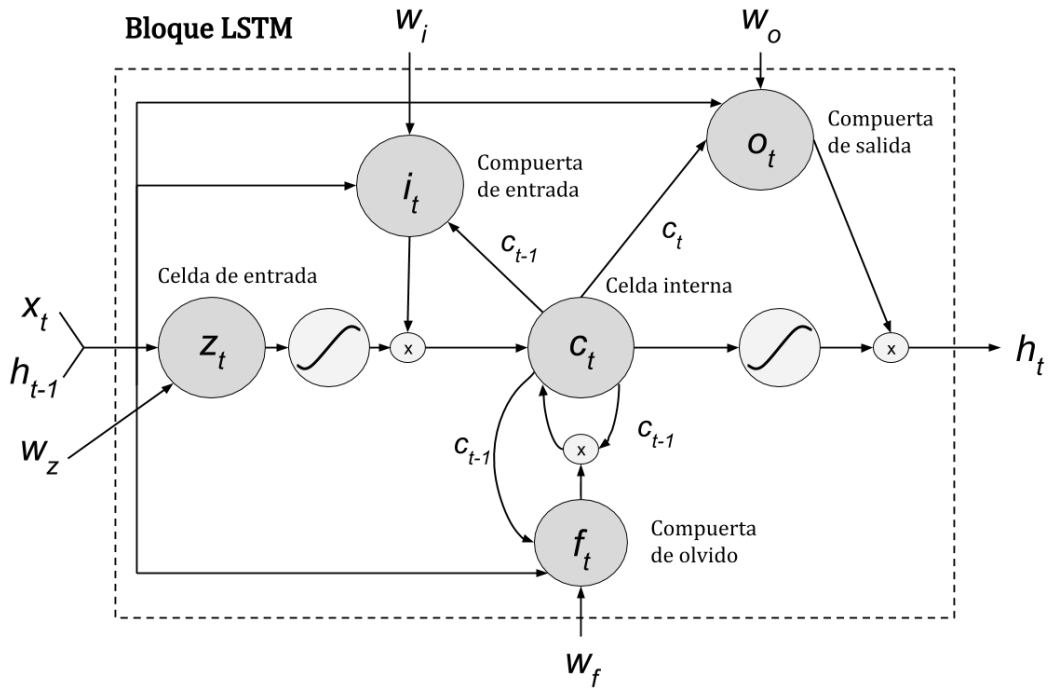


Figura 2.6. Diagrama de un bloque LSTM.
Fuente: Elaboración propia, 2017.

La celda de entrada es el equivalente a una neurona recurrente simple en su formulación. Ésta, recibe el vector de entrada del tiempo actual x_t y el estado oculto de la misma neurona en el tiempo anterior h_{t-1}, las cuales son multiplicadas por sub-matrices de pesos internos W_{zx} y W_{zh}, respectivamente. Estas últimas conforman a la matriz W_z, que posee la totalidad de los pesos internos utilizado por la celda de entrada. Sobre la suma de las matrices resultantes, se aplica una función de activación de tangente hiperbólica para retornar su valor de salida z_t. El comportamiento descrito está definido por la ecuación 2.10.

$$z_t = \tanh(W_{zx}x_t + W_{zh}h_{t-1} + b_z) \quad (2.10)$$

La compuerta de entrada cumple la función de controlar el paso del valor resultante de la celda de entrada en la neurona. Esta recibe el vector de entrada del tiempo actual x_t y el estado oculto de la misma neurona en el tiempo anterior h_{t-1} y el valor de la celda interna en el tiempo anterior c_{t-1}, las cuales son multiplicadas por las sub-matrices de pesos internos W_{ix}, W_{ih} y W_{ic}, respectivamente. Estas sub-matrices provienen de la matriz de pesos internos de la compuerta de entrada W_i. A la suma de las matrices resultantes se le aplica una función de activación sigmoidal, para entregar valores entre 0 y 1, retornando el valor de salida i_t. Esta compuerta está definida por la ecuación 2.11.

$$i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + W_{ic}\,c_{t-1} + b_i) \quad (2.11)$$

La compuerta de olvido cumple la función de controlar la influencia del valor anterior de la celda interior c_{t-1} en el valor actual de la celda interna c_t. Esta recibe el vector de entrada el tiempo actual x_t y el estado oculto de la misma neurona en el tiempo anterior h_{t-1} y el valor de la celda interna en el tiempo anterior c_{t-1}, las cuales son multiplicadas por las sub-matrices de pesos internos W_{fx}, W_{fh} y W_{fc}, respectivamente. Estas sub-matrices conforman la matriz de pesos de la compuerta de olvido W_f. A la matriz resultante se le aplica una función de activación sigmoidal, para entregar valores entre 0 y 1, retornando el valor de salida f_t. Esta compuerta está definida por la ecuación 2.12.

$$f_t = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + W_{fc}c_{t-1} + b_f) \quad (2.12)$$

La celda interna almacena el valor c_t, que representa la representación interna del estado de la neurona. Este valor se calcula cada *timestep*, con la suma de la multiplicación del valor de la celda de entrada z_t y la compuerta de entrada i_t, y la mutilplicación del valor de la celda interna anterior c_{t-1} con el valor de la compuerta de olvido f_t. Esto hace que el valor resultante sea una ponderación entre el valor de una celda recurrente simple, con el valor de la propia celda en el *timestep* anterior. Esto significa que se realiza una ponderación entre una recurrencia externa, representada por el valor del estado oculto anterior h_{t-1}, y una recurrencia interna, representada por el estado interno anterior c_{t-1}. Esta celda está definida por la ecuación 2.13.

$$c_t = z_t \odot i_t + c_{t-1} \odot f_t \quad (2.13)$$

La compuerta de salida cumple la función de controlar el valor de salida proveniente de la celda interna c_t. Ingresa la entrada actual x_t, el estado oculto anterior h_{t-1} y el valor de la celda interna actual c_t, las cuales son multiplicadas por las sub-matrices de pesos internos W_{ox}, W_{oh} y W_{oc}, respectivamente. A la suma de las matrices resultantes se le aplica una función de activación sigmoidal. Esto le permite entregar valores entre 0 y 1, que sean utilizados posteriormente para regular la salida de la neurona. El conjunto de sub-matrices conforma la matriz de pesos internos de la compuerta de salida W_o. Esta compuerta está definida por la ecuación 2.14.

$$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + W_{oc}c_t + b_o) \quad (2.14)$$

La salida de la neurona, en el tiempo actual t, está determinada por el valor la celda interna y la compuerta de salida. Se utiliza el producto de la función de activación, en este caso tangente hiperbólica, del valor actual de la celda interna $\tanh(c_t)$, con el valor de la compuerta de salida o_t. Cabe destacar que este valor será utilizado como estado oculto anterior (h_{t-1}) cuando se ingrese el vector de entrada del siguiente *timestep* (x_t) a la red neuronal. Esta salida está representada por la ecuación 2.15.

$$h_t = tanh(c_t) \odot o_t \quad (2.15)$$

En términos del problema de optimización al entrenar la red neuronal, la manera de entrenar una red LSTM es idéntica a cualquier otra implementación de red neuronal recurrente, donde se debe tener en cuenta la progresión de las variables recurrentes de manera de aprovechar la dinamicidad de la red neuronal en las series de tiempo. En este caso, se tiene como variables recurrentes el valor del estado oculto h_t y el valor de la celda interna c_t, los cuales deben ser ajustadas e inicializadas apropiadamente antes de resolver el problema de clasificación o regresión a estudiar. Una opción para cumplir esto, es alimentar a la red neuronal con secuencias de un largo determinado, inmediatamente anteriores al *timestep* sobre el que se realizan las evaluaciones. Otra opción es inducir analíticamente ciertos estados conocidos, provenientes de datos históricos, donde se cree que el contexto de la serie de tiempo es similar, y por tanto los patrones a reconocer a corto o largo plazo también lo sean.

2.1.5 *Word embedding*

La técnica de *word embedding* es utilizada en el contexto del Procesamiento del Lenguaje Natural (PLN), para mapear palabras de un vocabulario a vectores de números reales. El concepto de *embedding* o encaje en matemáticas, consisten en la utilización de una función inyectiva $f: X \rightarrow Y$ que permita ingresar un conjunto *X* a un conjunto *Y*, preservando su estructura de manera que $X \subset Y$. En este caso se utiliza un conjunto de identificadores únicos para cada palabra dentro de un vocabulario, representados por números enteros, y se mapea a un espacio vectorial real y unitario de *n* dimensiones.

Esta técnica se aplica en redes neuronales, para mapear palabras o metadatos directamente, de manera que cada valor o conjunto de valores, tenga su representación única como vector unitario de valores reales. Este proceso se realiza dentro de la red neuronal, a través de una matriz de pesos internos, que contiene el valor del vector de salida para cada palabra del vocabulario. Esta matriz se entrena como cualquier otra matriz de pesos en una red neuronal, a través del proceso de *backpropagation*. Por consiguiente, los valores de los vectores se van ajustando durante el proceso de entrenamiento, de manera de minimizar el valor de la función objetivo. La similitud o diferencia entre los valores del vector mapeado para cada palabra va a estar determinado directamente por el comportamiento que posean con respecto al problema estudiado, sea tanto de clasificación o regresión. Visto de otra forma, la red neuronal aprende a mapear valores no numéricos como palabras o valores numéricos contextuales como metadatos, generando relaciones no lineales entre ellos, de manera que puedan aportar a la resolución de algún problema a estudiar.

Esta técnica es utilizada en redes neuronales recurrentes para manejar la entrada de metadatos que dan contexto a una serie de tiempo. Este tipo de implementación da paso a 2 tipos de entradas distintas, donde los metadatos son manejados por capas ocultas de *embedding* y capas ocultas no recurrentes, a los datos de la serie de tiempo por capas de neuronas recurrentes.

Ambas subredes se conectan en alguna capa subsecuente común, que une los resultados de ambas partes.

2.1.6 Fenómeno de *bunching* de buses

El agrupamiento o *bunching* de buses, se define como el intervalo de tiempo en que un bus transita al menos junto a otro bus del mismo servicio, separados por un *headway* menor a una cierta fracción del *headway* programado. El *headway* es la diferencia de tiempo que existe entre 2 buses que recorren la misma ruta, en llegar al mismo punto del recorrido, entendiéndose también como un desfase de sus tiempos de viaje. El *headway* programado, es determinado por el Programa de Operación (PO) de cada empresa operadora de buses, donde se detallan distintos tiempos de viaje según el servicio, el sentido (ida o regreso) y el periodo de la semana. La determinación de estos valores depende de la demanda de usuarios estimada, de modo que el intervalo de tiempo es menor en horarios punta, para cubrir una mayor cantidad de usuarios, y es mayor horarios de madrugada o fines de semana, ya que se espera que no exista una gran demanda de usuarios.

Las causas del agrupamiento de buses son diversas, ya que depende tanto de condiciones internas del sistema de transporte, como de condiciones externas. Las condiciones internas son el comportamiento del conductor al ingresar o despachar pasajeros, una programación inapropiada de tiempos de operación, entre otros factores. Las condiciones externas son la congestión del tráfico vehicular, los tiempos de espera en los semáforos, la variación de demanda de pasajeros, entre otros. La gran cantidad de factores que pueden producir este fenómeno, además de la influencia dinámica de estos factores en el tiempo, dificulta la capacidad de poder tomar medidas decisiones basadas en eventos conocidos del pasado.

Dado un conjunto de buses que viajan en una misma ruta, si un bus se retrasa con respecto a los otros buses del mismo servicio, éste tiende a aumentar la cantidad de pasajeros que ingresa, debido a que aumentan los tiempos de espera de los pasajeros en los paraderos, acumulándose una mayor cantidad. Esto genera un aumento de los tiempos de ingreso y despacho de pasajeros, lo que resiente aún más el retraso en los tiempos de viaje del bus. Por otra parte, el bus anterior al bus retrasado ve disminuidos los tiempos de espera en los paraderos, por lo que su demanda de pasajeros tiende a disminuir, desaprovechando su capacidad de operación. Este efecto de “bola de nieve” conlleva a que el bus retrasado y el bus inmediatamente anterior acaben juntándose, generando serios problemas en la calidad de servicio. Esto es posible verlo en la Figura 2.7. Este problema puede extenderse a una mayor cantidad de buses en la ruta, donde un bus retrasado influencia al retraso de los buses que lo suceden en la ruta, formando trenes de buses.

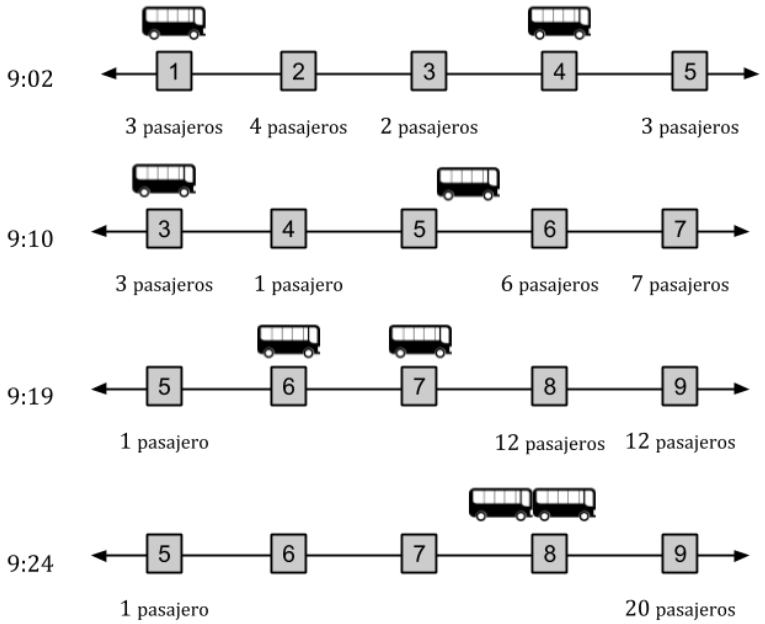


Figura 2.7: Ejemplo de agrupamiento entre dos buses en el mismo recorrido.
Fuente: Elaboración propia, 2017.

Moreira-Matias et al. (2012) propone que, siendo $X = \{x_1, \dots, x_n\}$ una secuencia de *headway* medidos entre un par de buses durante *n* paraderos, se detecta *bunching* entre ellos si existe un x_i que satisfaga la inecuación $|x_i| \leq 0{,}25 * x_1$, con $i \epsilon \{1, \dots, n\}$. Esto quiere decir que durante el recorrido de ambos buses en la ruta, se detecta *bunching* si el *headway* entre ambos es menor al 25% del *headway* con que iniciaron el viaje. Cabe destacar que se utiliza el valor absoluto del *headway* en cada paradero *i* debido a que los buses pueden desfasarse, de tal manera, que el bus que inicia primero su recorrido, sea sobrepasado por el bus siguiente en algún punto del recorrido. De esta forma, si dos buses intercambian su orden en la ruta pero mantienen el *headway* permitido, entonces no es posible considerar que exista *bunching* entre ellos. Se utiliza como *headway* de referencia al *headway* inicial y no al *headway* programado en el programa de operación, debido a que este último es estimativo y la frecuencia de buses de un servicio puede variar dependiendo de la demanda en un momento específico.

Este fenómeno posee distintas acciones de mitigación, en caso de manifestarse en un instante específico, las cuales han sido ampliamente estudiadas en la literatura (Zuñiga, 2011). Estas acciones, son principalmente las siguientes:

- Retención en paradas: se retiene un bus en una parada por un cierto tiempo para evitar entrar en *bunching* con el bus que está adelante en el recorrido, y ajustarse al intervalo de tiempo programado.

- Salto de estación: en caso de que un bus se atrase en sus tiempos de viaje o intervalos de tiempo con respecto al bus siguiente, este puede saltar una parada, de modo de adelantarse y ajustarse al intervalo de tiempo programado.

- Programación preferente de semáforos: consiste en modificar el comportamiento de ciertos semáforos de una ciudad, con tal de producir ciertos efectos deseados en el tráfico de vehículos. En este caso se busca mantener o alcanzar cierto nivel de regularidad de algún servicio de buses. Dentro de las acciones específicas que se pueden realizar, es la extensión del intervalo del semáforo en luz verde, adelantamiento del periodo de la luz verde, entre otras.

- Inyecciones de buses: en caso de que, en los primeros dos tercios del recorrido de un bus, exista un atraso considerable del intervalo de tiempo programado, se pueden "inyectar" buses del mismo servicio, para tomar a los pasajeros del tercio restante del recorrido, de modo de disminuir los tiempos de espera.

2.2 REVISIÓN DE LA LITERATURA

2.2.1 Predicción de tiempos de viaje

El problema de predicción de tiempo de viaje, tanto para buses urbanos, como otro tipo de transporte terrestre, es abordado generalmente en el ámbito de los sistemas inteligentes de transporte (en adelante SIT) (Chowdhury & Leung, 2011). Estos sistemas, utilizan herramientas provenientes del área de las telecomunicaciones y la informática, para controlar el uso de la infraestructura de transporte, además de entregar información oportuna a los usuarios. Para implementar un sistema que permita predecir los tiempos de viaje de un conjunto de buses urbano de un servicio de manera eficiente, es necesario obtener información del desplazamiento de los buses durante su recorrido, del tráfico vehicular aledaño y de la demanda de usuarios. Gracias al desarrollo de los SIT en la actualidad, es posible obtener tal información en tiempo real y tener registro de ella, a través de la coordinación de distintos subsistemas de control.

Uno de los primeros trabajos en utilizar redes neuronales para estudiar el problema de predicción de tiempos de viaje de buses urbanos en líneas de transporte simuladas, es presentado por Chien et al. (2002). Se realizan 380 simulaciones de tiempos de viaje de los buses de un recorrido específico de la ciudad de New Jersey, en Estados Unidos de América, utilizando un *software* especializado en simulación de tráfico vial llamado CORSIM. No se utilizan datos de buses reales, debido a la dificultad técnica que existía en el año 2002, de extraer información a través de dispositivos de geolocalización, como el GPS. A través de estas simulaciones se obtuvo un conjunto de variables que fueron utilizadas para diseñar dos modelos redes neuronales que permitan predecir los tiempos de viaje. El primer modelo, considera variables específicas del recorrido entre cada paradero del bus, como el volumen de tráfico vehicular, las velocidades que

adquiere el bus, entre otras. El segundo modelo considera, además de las variables del primero, los promedios parciales de tales variables, como el volumen promedio del tráfico vehicular, la velocidad promedio del bus, entre otras. Fueron diseñadas 10 arquitecturas de redes distintas, utilizando en todas ellas una única capa intermedia, con un máximo de 7 neuronas. Los bajos errores en las predicciones confirmaron la potencialidad del uso de redes neuronales para resolver este problema.

El trabajo de Jeong & Rillet (2004) muestra que, utilizando redes neuronales es posible obtener mejores predicciones de tiempos de viaje que con modelos de regresión y modelos basados en datos históricos. Se utilizaron 13 modelos de redes neuronales distintas, con 2 funciones de activación distintas, aplicando distintos métodos de aprendizaje para cada uno. Cada modelo utiliza como variables de entrada la hora de llegada de un bus al paradero actual, el tiempo de subida y bajada de pasajeros a tal paradero, el intervalo horario actual y el horario programado de llegada para el bus a ese paradero. La variable a predecir es el tiempo de viaje de un bus entre la parada actual y una parada futura a alcanzar. Se realiza un conjunto de pruebas para determinar la cantidad idónea de neuronas en la capa oculta, determinando que a mayor número de neuronas mayor es la precisión de las predicciones. Los modelos fueron entrenados con un conjunto de viajes provenientes de buses de la ciudad de Houston, Texas.

El trabajo de Gurmu & Fan (2014), propone un modelo simple con una red neuronal con una única capa intermedia, utilizando datos provenientes de muestreos GPS, que permite predecir eficazmente los tiempos de viaje entre paraderos de una línea de buses. Este modelo utiliza como variables de entrada el tiempo de viaje hasta el paradero actual, el intervalo de tiempo del día, un identificador del paradero actual *i* y un código identificador de un paradero posterior en la ruta *j*. La variable de salida es el tiempo de viaje entre el paradero actual y el paradero posterior ingresado. Los datos utilizados provienen de viajes de buses entre el año 2008 y 2009, de una ruta entre Macae y Rio de Janeiro, en Brasil, recorriendo 35 paraderos. Los resultados arrojan un error absoluto porcentual medio (MAPE), de 18,3% en las pruebas realizadas. Además, los resultados de la red neuronal implementada son mejores que los que entrega un modelo de promedios históricos, tanto en los distintos intervalos del día, como en las distintas secciones de la ruta.

El trabajo de de Brébisson et al. (2015) demuestra la potencialidad del uso de redes neuronales recurrentes para realizar predicciones del destino de un taxi usando solo el inicio de su trayectoria, representados en una secuencia de puntos GPS. Esta implementación es la ganadora del primer lugar del concurso ECML/PKDD Discovery Challenge, entre 381 grupos participantes (European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, 2015). Se utiliza un enfoque totalmente automatizado donde se implementan 3 modelos red neuronal: una red de perceptrón multicapa (MLP), una red neuronal recurrente

bidireccional (BRNN) y una red neuronal recurrente LSTM. El *dataset* se compone de 442 series de muestreos GPS, con las trayectorias de distintos taxis de la ciudad de Porto, en Portugal, además de metadatos asociados a cada viaje realizado. La arquitectura ganadora termina siendo el modelo MLP que utiliza una entrada de muestreos GPS fijos, sin embargo, pruebas posteriores realizadas por los propios autores revelan que la red neuronal bidireccional entrega mejores resultados. Cada modelo de red implementada maneja la secuencia de valores GPS de entrada, en capas ocultas distintas a los metadatos, los cuales ingresan a una capa de *embeddings*, que mapean cada combinación de valores como un vector único dentro de la red neuronal.

A pesar de que los casos estudiados de redes neuronales muestran una precisión aceptable para la predicción de tiempos de viaje de buses, no se detalla mayormente el rendimiento que poseen al realizar predicciones a eventos más lejanos en el futuro. Por lo general, se reporta la precisión al predecir al siguiente paradero, o a la distancia recorrida en el próximo muestreo GPS, pero no se detalla respecto a la capacidad de predecir eventos que pueden ocurrir con una mayor lejanía en el tiempo o en distancia en la ruta. Esto último es un elemento crucial, si se pretende controlar la ocurrencia de eventos que empeoren la calidad de servicio, como el fenómeno de *bunching* de buses, debido a que los sistemas de control deben tener el margen tiempo suficiente para tomar las medidas correspondientes para tomar acciones efectivas. En este sentido, el uso de redes neuronales recurrentes no solo facilita la codificación del fenómeno a estudiar, debido a su naturaleza temporal, sino que permite realizar predicciones lejanas en el futuro. Esto se debe a que es posible encadenar las predicciones a realizar para que sean utilizadas con entrada para predicciones de sucesos posteriores, aprovechando la dependencia temporal de los tiempos de viaje en la trayectoria de un bus.

El problema de predicción de tiempos de viaje para una línea de buses es un problema complejo, debido a la gran cantidad de factores de que influyen en una ciudad. En muchas ocasiones, los fenómenos que ocurren en una ciudad no son extrapolables a otras ciudades, debido a las condiciones climáticas, geográficas y viales únicas que presenta cada una. Incluso elementos culturales afectan al problema, debido a que existen ciudades donde se da una alta regularidad del sistema de transporte, debido a que la llegada de los buses a cada paradero está programada con anticipación y los conductores procuran cumplir esos tiempos. Un modelo de regresión aplicado a una ciudad como la anteriormente mencionada conseguiría errores menores en las predicciones, que uno donde no existe una planificación clara y la variación de tiempos de viaje sean mayores.

2.2.2 *Bunching* de buses

El problema de agrupamiento o *bunching* de vehículos, está asociado al problema de predicción de tiempos de viaje, pero su estudio es notoriamente más antiguo, debido a que se gesta en el contexto del propio diseño de sistema de transportes. Por ejemplo, Turnquist et al. (1980) realizan

experimentos de simulación para encontrar los factores que causan el agrupamiento de vehículos. Los resultados arrojan la importancia de controlar la variabilidad de los tiempos de viaje y la planificación de los recorridos, de manera de evitar problemas de regularidad del servicio en el transporte público, como el *bunching* de buses.

El trabajo de Moreira-Matias et al. (2016), implementa un *framework* de control automático, que permite mitigar el fenómeno de *bunching* de buses en tiempo real, utilizando monitoreo del transporte público en tiempo real. En esta herramienta, son utilizados una combinación de distintos enfoques del aprendizaje automático para realizar modelos predictivos respecto a la situación futura del tráfico en una línea de buses. Entre estos enfoques se incluye análisis de regresión, lógica probabilística y redes neuronales. Estos modelos fueron entrenados y probados con datos históricos de 18 rutas distintas de la ciudad de Porto, Portugal. Los resultados arrojaron un posible decremento del 68% de situaciones de *bunching* de buses y un 4,5% de disminución del tiempo de espera de los pasajeros, en caso de que se implemente el sistema en un ambiente real. No se especifica la duración y ni las características de los eventos de *bunching* detectados en las simulaciones, lo cual podría explicar la disparidad entre la disminución del tiempo de espera y los eventos de *bunching* erradicados, donde posiblemente los eventos de bunching detectados no afectan demasiado los tiempos de espera promedio o por el contrario los eventos de *bunching* no erradicados poseen una gran influencia en los tiempos de espera, posiblemente por su duración o rango de valores de *headway*.

En el contexto nacional, existe un sistema de monitoreo en tiempo real de los buses de los distintos servicios y operarios, que está a cargo del Centro de Monitoreo de Buses (CMB) del DTPM (Directorio de Transporte Público Metropolitano, 2018). Este sistema tiene la capacidad de conocer en tiempo real la posición de cada bus que circule en el sistema y compararla con los parámetros establecidos en el programa operacional. De esta manera, automáticamente se calculan los distintos índices de cumplimiento de frecuencia y regularidad de servicio, para cada empresa operaria. Además, el CMB tiene la posibilidad de exigir a la empresa operadora, la ejecución de alguna acción descrita en el Plan de Acción Control, de modo de mantener o mejorar la calidad de servicio del sistema. Sin embargo, este sistema no toma en cuenta el fenómeno de agrupamiento de los buses como causa directa de los problemas de regularidad del servicio.

CAPÍTULO 3. DISEÑO DE LA SOLUCIÓN

3.1 DESCRIPCIÓN DEL CONJUNTO DE DATOS UTILIZADO

El *dataset* utilizado proviene de datos históricos extraídos del sistema de AVL (Automatic Vehicle Location) del Directorio de Transporte Público Metropolitano (DTPM). Se utilizan dos conjuntos de datos: el primero contiene la información del tiempo de llegada de todos los buses de un conjunto de servicios a cada paradero de la ruta, mientras que el segundo, contiene la geolocalización, en forma de *waypoint*, de cada bus de un grupo de servicios, muestreada en intervalos de 30 segundos. Ambos *dataset* contienen un muestreo realizado durante las 24 horas del día. La cantidad de servicios de buses que contienen son 934 y 955, respectivamente, lo cual constituye más del 60% del total de servicios inscritos en el Programa de Operación del DTPM. El corte temporal de estos datos se encuentra entre el 28 de octubre de 2016 y el 27 de febrero de 2017, considerando solo los días laborales, es decir, de lunes a viernes de cada semana. En total existe una extensión de 89 días. La cantidad de filas en cada *dataset* es de 266.668.170 muestreos de llegadas a paraderos y 794.558.225 muestreos GPS.

El primer conjunto de datos posee 4 columnas. La primera columna corresponde al código del paradero al que ha llegado un cierto bus, constituido por 2 letras seguido de 4 dígitos numéricos. La segunda columna contiene la patente del bus monitoreado, representado por 4 letras seguido por 2 dígitos numéricos o 2 letras seguido 4 dígitos, según la antigüedad de la patente. La tercera columna posee el código del servicio que está realizando el bus, el cual se compone del código del recorrido y el código de la dirección en que está realizando el servicio. La cuarta columna contiene la fecha y la hora en que el bus llega al paradero, la cual está codificada en formato *timestamp*. En la Tabla 3.1 se muestra la cantidad de valores distintos que posee cada columna en este *dataset.*

El segundo conjunto de datos posee 5 columnas. La primera columna contiene la patente del bus, mientras de la segunda contiene el código de servicio, siguiendo el mismo formato del primer *dataset* bruto. La tercera columna posee la fecha y hora en que se realiza el muestreo de la geolocalización, codificada en formato *timestamp*. La cuarta y quinta columna contiene las coordenadas de los grados de latitud y longitud geográfica del bus, en el instante en que se realiza el muestreo. En la Tabla 3.2 se muestra la cantidad de valores distintos que posee cada columna en este *dataset*.

Tabla 3.1: Columnas de los dataset bruto 1.

Nombre de columna	***Cantidad de valores distintos***
Código de paradero	11.173
Patente bus	6.330
Código servicio	934
Hora de arribo	5.764.619

Fuente: Elaboración propia, 2017.

Tabla 3.2. Columnas del dataset bruto 2.

Nombre de columna	***Cantidad de valores distintos***
Patente bus	6.696
Código servicio	955
Hora del muestreo GPS	6.493.682
Latitud GPS	390.365.985
Longitud GPS	391.341.781

Fuente: Elaboración propia, 2017.

Luego de realizar una inspección de los datos contenidos en cada *dataset*, se encuentra la existencia de cupos de muestreos faltantes o erróneos con relación al comportamiento esperado en el trayecto de los buses de cada servicio. Se descubre que la mayoría de los viajes realizados en cada servicio, no cuentan con la secuencia completa de paraderos que debe realizar un bus, según la ruta de referencia entregada por la DTPM. También, el intervalo de tiempo entre muestreos, en la geolocalización de cada bus en circulación, no siempre es de 30 segundos, encontrándose tiempos de 29, 60 y 90 segundos. Además, en ambos *dataset* existen secuencias de muestreos de viajes que no están ordenadas temporalmente, dado que no siguen un orden creciente en los valores de sus *timestamp*. Por último, existen coordenadas de geolocalización que se presume erróneas, debido a que escapan de los límites de la ciudad de Santiago de Chile, o generan que los buses hayan tenido que alcanzar velocidades muy por sobre las capacidades de un bus urbano.

Para asegurarse que los viajes de los buses muestreados en cada dataset sigan un comportamiento coherente con la ruta del servicio y con las capacidades de los buses, se realiza

un proceso de reconstrucción de los datos. En el *dataset* de tiempos de llegada a los paraderos de la ruta, se realizó una interpolación de estos tiempos en los paraderos faltantes, dada una ruta modelo entregada por el DTPM. Además, se eliminaron las secuencias incompletas, donde faltaba más de 5 paraderos en la ruta, o no se llegaba al final de ésta. En el *dataset* de muestreos de la geolocalización de cada bus, se realiza un proceso de ordenamiento de las secuencias, donde se asegura que los *timestamp* sigan un comportamiento creciente, luego se eliminan los datos cuya geolocalización no corresponda a la ruta que el bus debe realizar, para finalizar interpolando los valores de geolocalización de manera que cada muestreo posea 30 segundos de diferencia con el muestreo siguiente.

Para modelar y entrenar los modelos de predicción a partir de los *dataset* a disposición, se elige un servicio que posea las características idóneas y el volumen de datos adecuado. Debido a que el problema a estudiar es el *bunching* de buses, se debe encontrar un servicio donde exista una alta probabilidad de que ocurra este fenómeno. Se presume que los servicios de buses troncales, que circulan por las arterias principales de la ciudad, cruzando varias comunas en su ruta, se ven más afectadas por los efectos de la congestión vehicular y la alta demanda de pasajeros. Si bien estas características no aseguran que ocurra el fenómeno de *bunching*, debido a que éste depende de una gran cantidad de factores, podría servir de indicativo al momento de elegir un servicio idóneo. Otro factor a tomar en cuenta es el volumen de datos, debido que se considera que poseer un mayor conjunto de viajes de buses, da posibilidad a las redes neuronales de reconocer de mejor manera los patrones involucrados, y evitar un sobre-entrenamiento a un conjunto de datos específico.

El servicio de buses “I09” es el elegido para generar el *dataset* de cada modelo implementado, considerando solo el sentido de ida. Su código en el servicio de transportes corresponde a “T379 00I” y su empresa operadora es Buses Vule S.A. Se decide utilizar este servicio debido a que su ruta tiene una extensión promedio con respecto al resto de servicios, siendo de aproximadamente 16 km, con 43 paraderos en la ruta y una distancia promedio de 372 metros entre cada paradero consecutivo. A pesar de que este servicio es de alimentación en la comuna de Maipú, se le puede considerar un servicio troncal porque transita solamente por avenidas importantes de la ciudad, como Av. Los Pajaritos o Av. Libertador Bernardo O’Higgins, cubriendo 3 comunas en su recorrido, como son Maipú, Estación Central y Santiago. La cantidad de viajes de buses registrados en ambos *dataset* para este servicio es de 10.865.

Para realizar el proceso de normalización y estudio de los datos del servicio elegido, es necesario conocer en detalle la ruta que describe. En la Figura 3.1 se muestra la ruta del servicio “I09”, según la Dirección de Transporte Público Metropolitano (2017). El depósito de buses se encuentra cerca del punto A, donde cada bus comienza su recorrido por Camino a Rinconada, en dirección al oriente. Luego, cada bus debe virar hacia el norte en la Plaza de Armas de Maipú,

tomando rumbo hacia Estación Central, por Av. Los Pajaritos. En esta última avenida, los buses poseen un corredor exclusivo, que les permite disminuir los efectos de la congestión vehicular. Luego, los buses continúan por la misma vía hasta el término de la avenida, al cruzar por la calle Las Torres, donde deben tomar la calle Gladys Marín Millie. Posteriormente, ingresan a Av. Libertador Bernardo O'Higgins, donde siguen por la misma vía hasta doblar por Unión Latino Americana, donde finaliza su recorrido en el punto B.

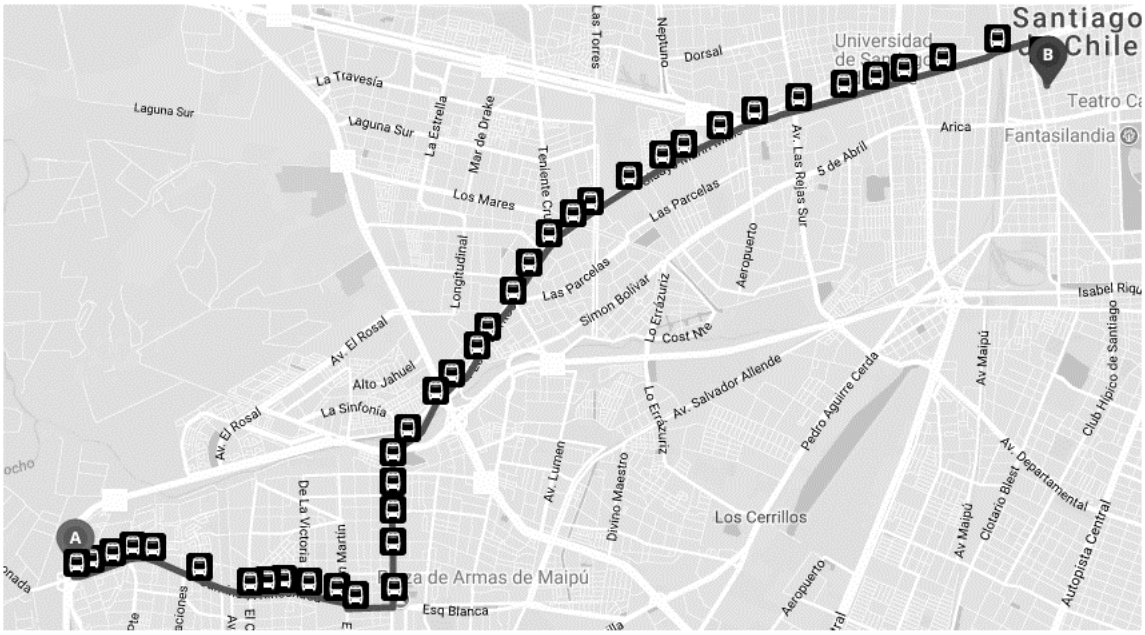


Figura 3.1. Ruta del servicio I 09, en sentido de ida, Santiago de Chile.
Fuente: Elaboración propia, 2017.

Como información adicional a los datos que entregan los *dataset*, es necesario conocer los intervalos del día donde se realiza el servicio elegido. En Tabla 3.3, se muestra una lista de los intervalos con sus respectivos rangos de horario para el servicio "I09", procedentes del Programa de Operación del segundo semestre de 2016. Este servicio se realiza en el horario punta de la mañana, asociado a la hora de llegada de las personas al trabajo o su lugar de estudios, el horario punta del mediodía, asociado principalmente a la salida de estudiantes de los colegios, y el horario punta de la tarde, asociado a la salida de los trabajadores de su lugar de trabajo y la llegada a sus hogares.

Tabla 3.3. Intervalos del día en que se realiza el servicio I09, de lunes a viernes.

Nombre del intervalo	Índice	Rango horario
Punta Mañana	1	6:30 a 8:29
Transición Punta Mañana	2	8:30 a 9:29
Fuera de Punta Mañana	3	9:30 a 12:29
Punta Mediodía	4	12:30 a 13:59
Fuera de Punta Tarde	5	14:30 a 17:29
Punta Tarde	6	17:30 a 20:29
Transición Punta tarde	7	20:30 a 21:29

Fuente: Elaboración propia, 2017.

3.2 ANÁLISIS DE LOS DATOS

Para apoyar al proceso de modelamiento de las redes neuronales es necesario realizar un análisis estadístico del conjunto de datos. Este se lleva a cabo respecto a las variables de distancias de avance de los buses en la ruta y el *headway* entre dos buses.

3.2.1 Análisis de Distancias

Al inspeccionar la distribución de valores de las distancias que recorre un bus, se descubre que estos no siguen una distribución normal. Se encuentra una media (μ) de 0,15224 km y una desviación estándar (σ) de 0,1187 km. Con estos valores se construye una distribución normal, de manera de inspeccionar visualmente la correspondencia de las frecuencias estimadas, respecto al histograma del conjunto de datos, presentado por la Figura 3.2. El histograma de los datos posee un descenso pronunciado de las frecuencias, a medida que va aumentando el valor de distancia, en contraste con una los valores de campana de Gauss alrededor de la media, de la distribución normal generada. De la misma manera, al apreciar el gráfico quantil-quantil, presentado por la Figura 3.3, se muestra que los valores de los datos no se ajustan completamente a los de una distribución normal, sobre todo en los extremos del rango de valores. Además, se realiza un conjunto de test estadísticos para evaluar con mayor exactitud la normalidad de los datos. Estos test son: Shapiro-Wilk (Shapiro & Wilk, 1965), Kolmogorov-Smirnov (Smirnov, 1948) y D'Agostino-Pearson (D'Agostino, 1971). El primer test entrega un valor w de 0,92486 y un valor *p* de 0, el segundo test entrega un valor d de 0,50013 y un valor *p* de 0, mientras que el tercer test da un valor k^2 de 97.931,049 y un valor p de 0. Como es posible apreciar todos los test arrojaron una probabilidad nula, lo cual significa que los datos no tienen correspondencia alguna con una distribución normal.

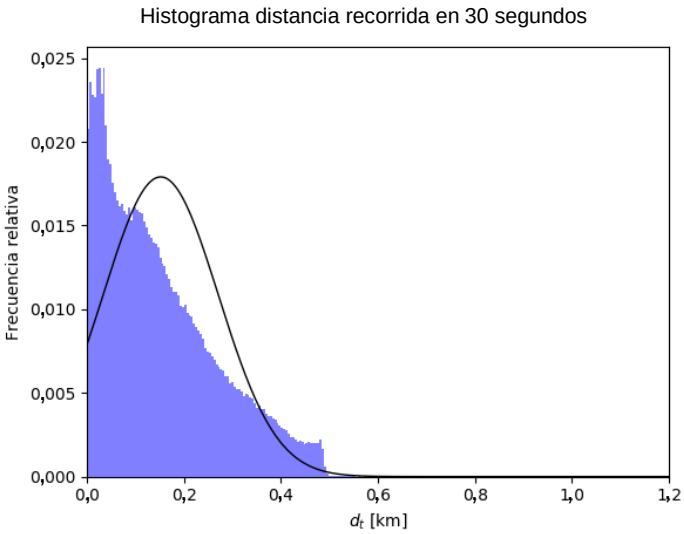


Figura 3.2. Histograma de distancias recorrida en 30 segundos.

Fuente: Elaboración propia, 2017.

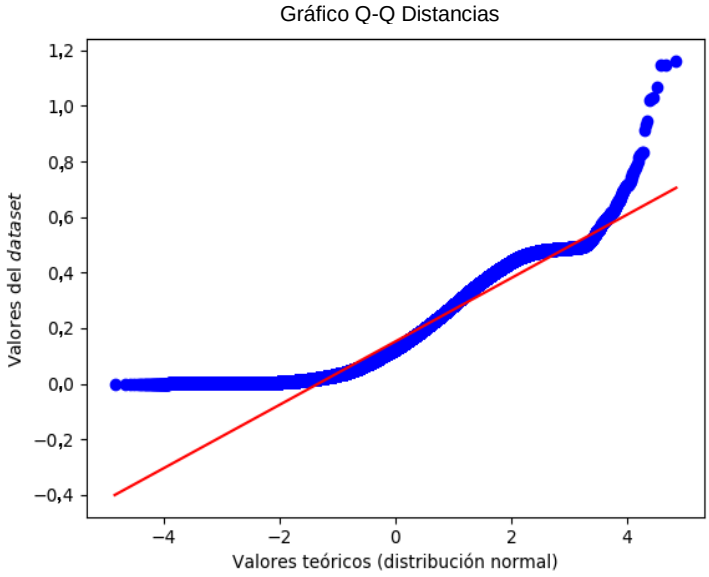


Figura 3.3. Gráfico Cuantil-Cuantil para valores de distancias recorridas en 30 segundos.

Fuente: Elaboración propia, 2017.

Según Gschwender (2016) el proceso de muestreo del posicionamiento los buses, posee un error promedio de 10 metros por cada medición, lo que se aproxima al 1% del rango de valores para las distancias del *dataset*. El valor mínimo registrado es de 0,000236 km y el valor máximo es de 1,162426 km. Este valor mínimo se puede expresar también como 23,6 cm, y por estar muy por debajo del error muestreo GPS se considera que el avance del bus en este lapso es aproximadamente 0. Es importante apreciar que más del 95% de los datos de distancia

muestreados no alcanzan los 500 metros aproximadamente, en lapsos de 30 segundos, y que por tanto la máxima velocidad que alcanza la mayoría de los buses es menor a 60 km/h.

Inspeccionando los datos se conoce la cantidad de muestras por día de la semana e intervalo horario. Por día de la semana, se cuenta con 221.108 muestras para los días lunes, 220.172 para los días martes, 221.312 para los días miércoles, 213.265 para los días jueves, y 223.976 para los días viernes. Como se puede notar, las muestras se distribuyen de manera bastante homogénea a través de los días de la semana, teniendo una diferencia máxima de 4,78%. Por intervalo horario, existen 60.645 para “Punta Mañana”, 200.721 para “Transición Punta Mañana”, 93.249 para “Fuera de Punta Mañana”, 190.780 para “Punta Mediodía”, 110.401 para “Fuera de Punta Tarde”, 198.708 para “Punta Tarde” y 251.329 para “Transición Punta tarde”. Se puede apreciar que el último intervalo horario es el que posee la mayor cantidad de muestras, en demerito de los horarios punta. Los horarios punta de la mañana y de la tarde poseen una cantidad considerable de muestras, como es de esperar, debido a que existe una mayor demanda de pasajeros, y por tanto disminuye el *headway* programado entre cada bus la ruta, aumentando la flota de buses y la cantidad viajes que realizan.

Al realizar un análisis de la correlación lineal entre *timesteps* consecutivos de las distancias que recorre un bus en lapsos de 30 segundos, se obtiene un coeficiente de correlación lineal de 0,33662. En la Figura 3.4 se presenta el gráfico de autocorrelación para *timestep* consecutivos. Se puede confirmar que la mayor parte de los valores están por debajo de 0,5 km y visualmente no se aprecia una autocorrelación lineal, debido que los valores están distribuidos de manera homogénea dentro de este rango.

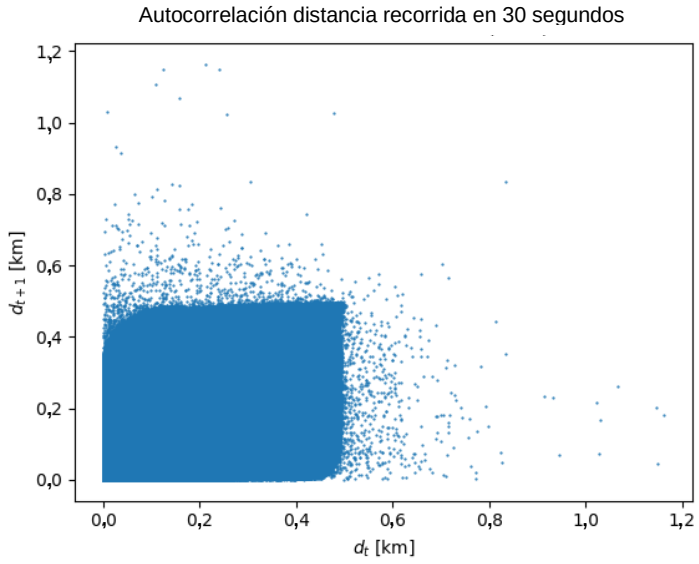


Figura 3.4. Gráfico de autocorrelación en timesteps consecutivos para la distancia que recorre un bus.

Fuente: Elaboración propia, 2017.

3.2.2 Análisis de *Headway*

Al inspeccionar la distribución de valores de las headway entre 2 buses en la ruta, se descubre que estos no siguen una distribución normal. Los datos tienen una media (μ) de 676,73 s y una desviación estándar (σ) de 544,61 s. De manera análoga al análisis de distancias, se construye una distribución normal, con el fin de inspeccionar visualmente la correspondencia de las frecuencias estimadas y el histograma del conjunto de datos, presentado por la Figura 3.5. El histograma de los datos parece ajustarse a la distribución normal generada, si bien su forma es más asimétrica teniendo mayores frecuencias para valores menores a la media y teniendo descenso menos pronunciado de las frecuencias, para valores mayores a la media. Sin embargo, al apreciar el gráfico quantil-quantil, presentado por la Figura 3.6, se muestra que los valores de *headway* no se ajustan a los de una distribución normal en los extremos del rango de valores. Además, se realiza el mismo conjunto de test estadísticos, anteriormente mencionado, para evaluar con exactitud la normalidad de los datos. El test de Shapiro-Wilk entrega un valor w de 0,93553 y un valor *p* de 0, el test de Kolmogorov-Smirnov entrega un valor d de 0,93966 y un valor *p* de 0, mientras el test de D'Agostino Pearson arroja un valor k^2 de 270.521,856 y un valor p de 0. Como es posible apreciar todos los test arrojaron una probabilidad nula, lo cual significa que los datos no tienen correspondencia alguna con una distribución normal, a pesar de las presunciones hechas a partir de los gráficos presentados.

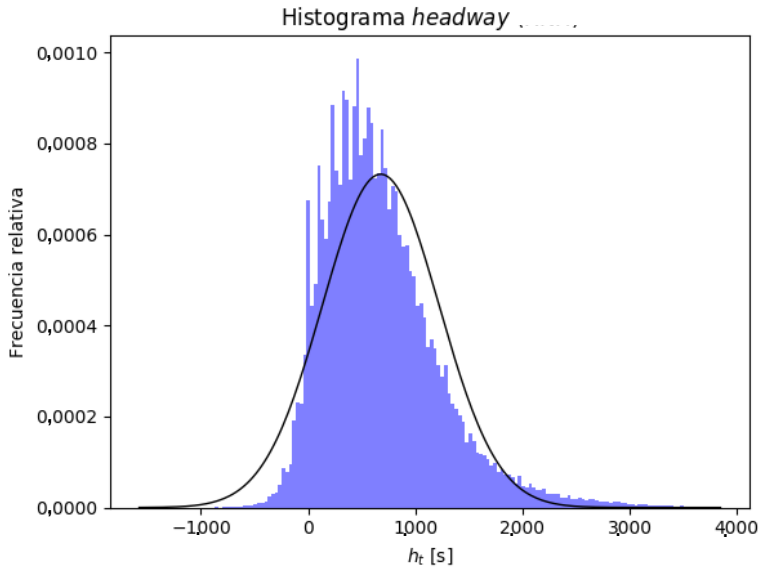


Figura 3.5. Histograma para headway entre 2 buses.
Fuente: Elaboración propia, 2017.

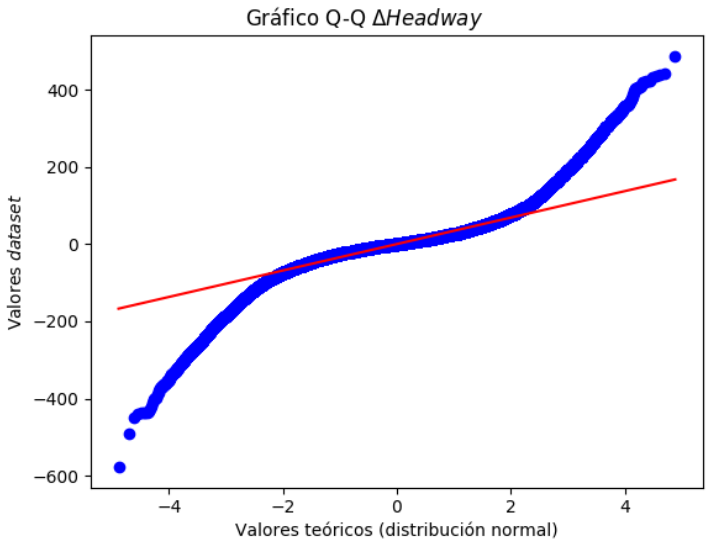


Figura 3.6. Gráfico Cuantil-Cuantil para la diferencia de headway entre 2 buses, en paraderos consecutivos.

Fuente: Elaboración propia, 2017.

Al realizar el mismo proceso de inspección y evaluación de la distribución, para los valores de diferencia de headway entre 2 paraderos consecutivos para cada par de buses, se determina que estos no siguen una distribución normal. Estos datos poseen una media (μ) de 0,04924 s y una desviación estándar (σ) de 35,87587 s. De manera análoga al análisis de distancias, se construye una distribución normal, con el fin de inspeccionar visualmente la correspondencia de las frecuencias estimadas y el histograma del conjunto de datos, presentado por la Figura 3.7. El histograma de los datos parece ajustarse a la distribución normal generada, a pesar de tener una curtosis mayor en su distribución de frecuencias. Al apreciar el gráfico quantil-quantil, presentado por la Figura 3.8, se muestra que los valores de *headway* no se ajustan a los de una distribución normal en los extremos del rango de valores. Al igual que para los datos anteriores, se realiza un conjunto de test estadísticos, para evaluar con exactitud la normalidad de los datos. El test de Shapiro-Wilk entrega un valor w de 0,91247 y un valor *p* de 0, el test de Kolmogorov-Smirnov entrega un valor d de 0,45116 y un valor *p* de 0, mientras el test de D'Agostino Pearson arroja un valor k^2 de 209.008,013 y un valor p de 0. Como es posible apreciar todos los test arrojaron una probabilidad nula, lo cual significa que los datos no tienen correspondencia alguna con una distribución normal.

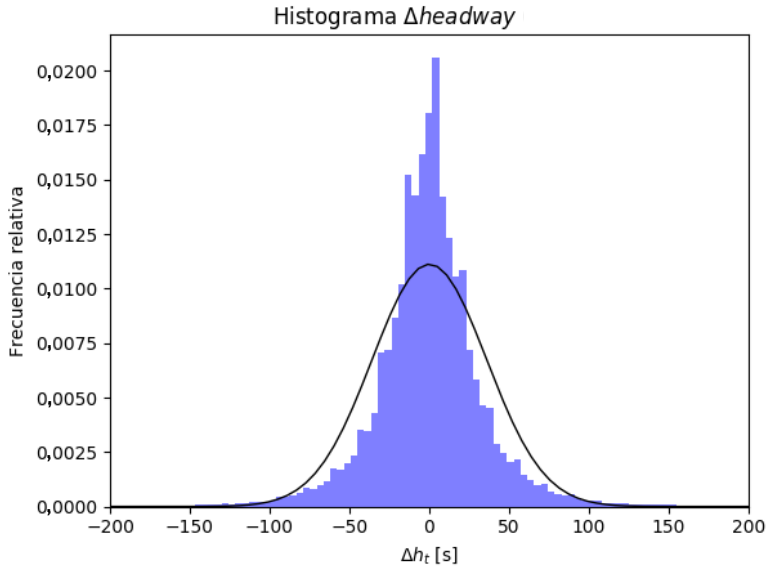


Figura 3.7. Histograma de diferencia de headway entre 2 buses, en paraderos consecutivos.
Fuente: Elaboración propia, 2017.

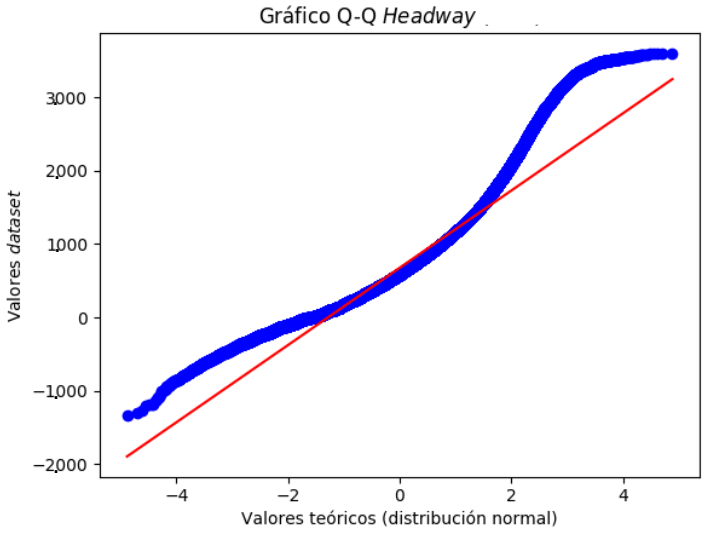


Figura 3.8. Gráfico Cuantil-Cuantil para el headway entre 2 buses.
Fuente: Elaboración propia, 2017.

Extrapolando los datos de Gschwender (2016) sobre el error del sistema posicionamiento los buses, el cual posee un error promedio de 10 metros por cada medición, se considera que error en los valores de *headway* es menor o igual a 1,6 s. Esta información se deduce a partir de la información de que la mayoría de los buses no superan los 60 km/h, y por lo tanto un error de 10 metros a dicha velocidad se traduce en 1,6 s de error en los tiempos de viaje. Los valores de

headway son calculados a partir de los tiempos llegada los buses a los distintos paraderos de la ruta, los cuales utilizan la misma tecnología de posicionamiento que los datos de geolocalización.

Inspeccionando los datos se conoce la cantidad de muestras por día de la semana e intervalo horario. Por día de la semana, se cuenta con 271.404 muestras para los días lunes, 266.700 para los días martes, 265.650 para los días miércoles, 266.868 para los días jueves, y 257.250 para los días viernes. Como se puede notar, las muestras se distribuyen de manera bastante homogénea a través de los días de la semana, teniendo una diferencia máxima de 5,22%. Por intervalo horario, existen 89.334 para “Punta Mañana”, 236.922 para “Transición Punta Mañana”, 112.476 para “Fuera de Punta Mañana”, 219.702 para “Punta Mediodía”, 129.906 para “Fuera de Punta Tarde”, 241.542 para “Punta Tarde” y 297.990 para “Transición Punta tarde”. Se puede apreciar que el último intervalo horario es el que posee la mayor cantidad de muestras, en demerito de los horarios punta, ocurriendo el mismo fenómeno que con los datos de distancias. Esto puede deberse a que ambos conjuntos de datos poseen los mismos viajes del mismo servicio de buses, en el mismo corte temporal, porque es de esperar que la cantidad de muestras sean relativamente proporcionales.

Al realizar un análisis de la correlación lineal entre *timesteps* consecutivos del *headway* entre 2 buses, se obtiene un coeficiente de correlación lineal de 0,99820. En la Figura 3.9 se presenta el gráfico de autocorrelación de los valores de *headway* para *timestep* consecutivos. Se puede apreciar que existe una fuerte correlación lineal entre los valores, al describir un gráfico similar a una recta. Esto significa que existe una fuerte dependencia temporal entre los valores de *headway* tomando en cuenta 1 timestep de diferencia. Por otra parte, al analizar la autocorrelación de la diferencia de headway entre un par de buses, en paraderos consecutivos, se consigue un coeficiente de correlación de 0,05317. Esto último, junto con el gráfico de autocorrelación de la Figura 3.10, confirma que la correlación lineal es muy baja y que por lo tanto no existe una dependencia temporal fuerte en *timestep* consecutivos.

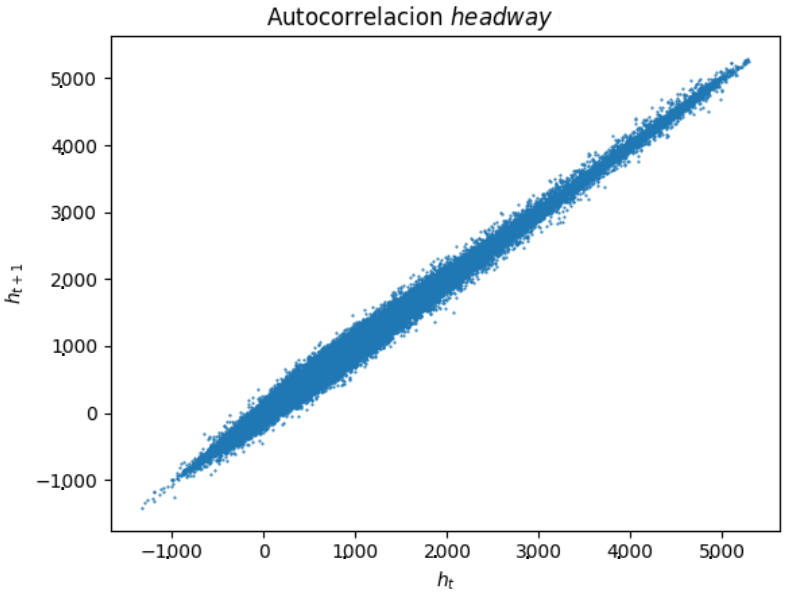


Figura 3.9. Gráfico de autocorrelación de headway en timesteps consecutivos.

Fuente: Elaboración propia, 2017.

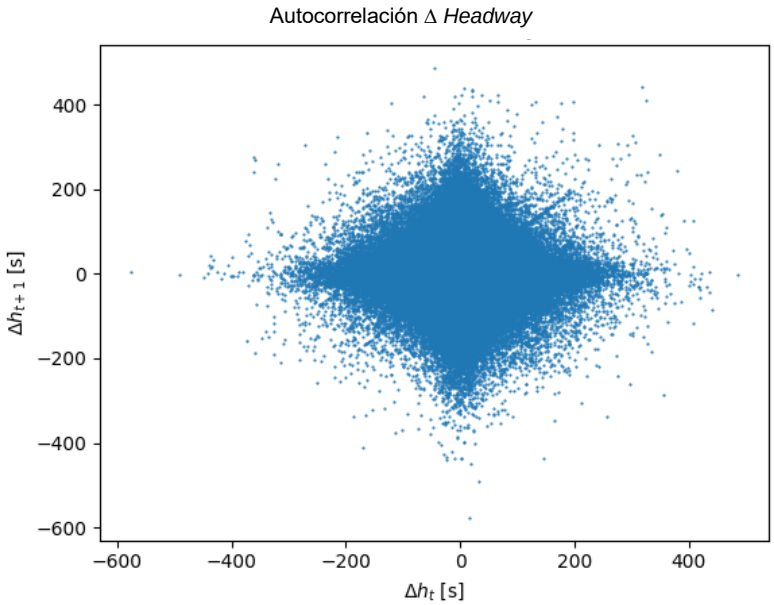


Figura 3.10. Autocorrelación de diferencia de headway en timesteps consecutivos.

Fuente: Elaboración propia, 2017.

3.3 DISEÑO DE LOS MODELOS DE PREDICCIÓN

3.3.1 Red neuronal de paraderos (RNP)

El primer modelo implementado, consiste está basado en el modelo propuesto por Gurmu & Fan (2014) (véase sección 2.2.1). Este modelo implementa una red neuronal *feedfordward* con una capa oculta, para predecir el tiempo de viaje de un bus entre un paradero actual *i* y un paradero *j* futuro, dado el intervalo del día en que se encuentra el bus (*t*), el tiempo de viaje de hasta el último paradero (TT_{1i}), el índice del paradero actual *i* y el paradero futuro *j*. En la Tabla 3.4, se observan en detalle las variables involucradas en este modelo. La ecuación 3.1 presenta el modelo de predicción que relaciona las variables de entradas con la variable de salida a través de la función *f*, que representa el conjunto de operaciones que realizan las neuronas de la red neuronal para convertir los valores de entrada en un valor de salida. En adelante este modelo es nombrado como RNP, para identificarlo con mayor facilidad.

$$y = f(x_1, x_2, x_3, x_4) \tag{3.1}$$

El modelo RNP se implementa con la finalidad de utilizarlo como referencia, para comparar sus resultados con otros modelos implementados, de manera de determinar la idoneidad de sus enfoques de solución. Se considera que RNP ha demostrado alcanzar un nivel de precisión aceptable en las predicciones, utilizando una implementación simple, por lo que se pretende que el resto de modelos de redes neuronales diseñados, logren mejorar o al menos igualar a los resultados obtenidos por la actual implementación.

Tabla 3.4: Descripción de variables de entrada y salida para RNP.

Variable	Tipo de variable	Descripción	Dominio	Rango de valores
x_1	Entrada	Intervalo del día	$\mathbb{N}$	$[1, 7]$
x_2	Entrada	Índice del paradero actual	$\mathbb{N}$	$[1, 42]$
x_3	Entrada	Tiempo de viaje actual en segundos	$\mathbb{N}_0$	$[0, 4.800]$
x_4	Entrada	Índice del paradero a consultar	$\mathbb{N}$	$[2, 43]$
y	Salida	Tiempo de viaje entre el paradero actual y el paradero a consultar, en segundos.	$\mathbb{R}$	$[0, 1.800]$

Fuente: Elaboración propia, 2017.

El objetivo es que los resultados en las predicciones de tiempos de viaje mediante RNP sean similares a los conseguidos por Gurmu & Fan (2014), procurando que la implementación a realizar sea lo más fiel posible a la original. Sin embargo, hay elementos propios del trabajo original que no son posibles de replicar, como el *dataset* utilizado para realizar el entrenamiento y prueba de la red neuronal, por lo que es lógico considerar que las posibles diferencias en el volumen y comportamiento de los datos generen diferencias en los resultados. Por otra parte, esta implementación no se limita a utilizar los hiper-parámetros y procedimientos explicados en el trabajo original, sino que se realiza un proceso de optimización propio, para ajustar estos hiper-parámetros a las condiciones del *dataset* utilizado.

Para generar el conjunto de datos necesario para realizar el entrenamiento y prueba de la red neuronal, se utiliza el primer *dataset* bruto, que posee el registro de la hora de llegada de un conjunto de buses del mismo servicio, a cada paradero de la ruta. Se toma en cuenta que el recorrido comienza en el primer paradero y termina en el último paradero de la ruta, por lo tanto, la hora de inicio y fin del viaje realizado por un bus están determinadas por estos valores, respectivamente. Es necesario destacar que a este *dataset* bruto, anteriormente se le ha realizado un proceso de normalización, para obtener las secuencias ordenadas de horas de llegada a cada paradero, para un conjunto de viajes de buses, filtrando los datos que no sean coherentes, interpolando los datos para paraderos donde no exista registro de la hora de llegada, entre otros aspectos (véase sección 3.1).

El algoritmo utilizado para generar el *dataset* a partir de un conjunto de secuencias de tiempos de llegada a cada paradero de la ruta, es el siguiente:

- Primero, se resta la hora de llegada T_i de un cierto bus a cada paradero *i*, con la hora de inicio del viaje T_1, resultando en una secuencia de tiempos de viaje del bus en cada paradero: $TT_{1i} = T_i - T_1,\ 2 \leq i \leq 43$.

- En segundo lugar, se calcula el intervalo del día (*t*) en que se realiza cada viaje, de acuerdo con el rango de horas de cada viaje, clasificándolo según el conjunto de intervalos del servicio seleccionado, del Programa de Operación del DTPM.

- En tercer lugar, para cada paradero *i* de la ruta, exceptuando el último, se calcula la diferencia de los tiempos de viaje con respecto a cada paradero *j* posterior en la ruta (TT_{ij}), formando una combinación de tiempos de viaje entre cada paradero de la ruta. Esto resulta en 903 combinaciones de tiempos de viaje, por cada viaje dentro del *dataset* bruto, generando 9.811.095 de combinaciones de tiempos de viaje en total.

- Finalmente, se exportan los datos generados a un archivo de texto plano, donde cada variable presente en la Tabla 3.4 es una columna del *dataset*, y cada fila es una muestra de los tiempos de viaje entre paraderos de un bus durante su recorrido.

La definición de la arquitectura de la red neuronal depende de los resultados del proceso de ajuste de hiper-parámetros (véase sección 3.4.4). En este proceso se establece el método de aprendizaje a utilizar, la cantidad de capas ocultas, la cantidad de neuronas por cada capa y la función utilizada en la normalización del *dataset*. Por otra parte, hay elementos que no son modificados en el transcurso de las pruebas, como las variables de entrada y salida, la función de activación de las neuronas y la arquitectura general de la red neuronal. La arquitectura utilizada, es de perceptrón multicapa, compuesta de una capa de entrada, que captura las 4 variables de entrada; un conjunto de capas ocultas por determinar, donde cada neurona está completamente conectada con las neuronas de la capa anterior; y una capa de salida, con 1 solo nodo de salida, que retorna la predicción del tiempo de viaje entre un par de paraderos. La función de activación utilizada en las neuronas de las capas ocultas es la tangente hiperbólica, debido a su utilización en la implementación de Gurmu & Fan (2014).

El proceso de normalización de los datos de entrada y salida de la red neuronal fue implementado de dos maneras distintas, una de las cuales sigue las características del modelo original de Gurmu & Fan (2014). Este proceso se aplica tanto para las variables de entrada a la red neuronal, como a los valores de referencia de la salida de la red neuronal ($\hat{y}$), al momento de realizar el entrenamiento de la red neuronal. La primera implementación, presentada en la ecuación 3.2, escala los valores de manera lineal, donde los valores de una cierta variable x_i son divididos por el valor máximo posible de esa variable y escalados a un rango de valores entre -1 a 1. Al realizar este proceso, se modifica el dominio de las variables que se encuentren en los números naturales ($\mathbb{N}$), quedando todas dentro del conjunto de los números reales ($\mathbb{R}$). La segunda implementación, presentada en la ecuación 3.3, se aplica el mismo proceso de escalado que en la primera implementación, pero luego se le aplica una función de tangente hiperbólica. Esto hace que la distribución de los valores de las variables, en el rango de -1 a 1, ya no sea lineal. El proceso de denormalización se realiza utilizando la función opuesta a la de normalización, en ambos casos, presentadas por las ecuaciones 3.4 y 3.5, respectivamente.

$$x_i' = 2 * \frac{x_i}{max(x_i)} - 1 \tag{3.2}$$

$$x_i' = tanh\left(2 * \frac{x_i}{max(x_i)} - 1\right) \tag{3.3}$$

$$x_i = \frac{max(x_i)}{2} * (x_i' + 1) \tag{3.4}$$

$$x_i = \frac{max(x_i)}{2} * (atanh(x_i') + 1) \quad (3.5)$$

3.3.2 Red neuronal de distancias (RND)

El segundo modelo implementado consiste en una red neuronal recurrente, que predice las distancias que recorre el bus en la ruta, en lapsos de tiempo de 30 segundos. Se ingresa una serie de tiempo parcial, que describe la distancia recorrida de un bus en los últimos *timestep* muestreados, y un conjunto metadatos del viaje, para predecir la distancia recorrida en siguiente *timestep*. En este sentido este modelo de predicción es de tipo One-Step-Ahead, debido a que realiza predicciones solo a 1 *timestep* posterior al último ingresado en la serie de tiempo. Se presume que, debido a la naturaleza del recorrido de los buses en una ruta predeterminada, las distancias que recorren en un lapso de tiempo están influidas por las distancias recorridas en los lapsos de tiempo anteriores. También, distintas zonas de la ruta pueden presentar distintos comportamientos, generándose patrones específicos para cada zona, debido principalmente a las condiciones viales estáticas. Se pretende aprovechar la capacidad de las redes neuronales recurrentes de manejar la dependencia temporal de las series de tiempo, para obtener predicciones más precisas que utilizando una red neuronal *feedforward* convencional. También, se pretende aprovechar el contexto temporal en que se desarrollan los viajes de los buses, manejando de manera independiente, aspectos como el intervalo del día y el día de la semana en que ocurren. En adelante este modelo es nombrado como RND, para identificarlo con mayor facilidad.

De manera similar a RNP, la definición de parte de la arquitectura y diseño de la red neuronal de RND depende de los resultados del proceso de ajuste de hiper-parámetros (véase sección 3.4.4). En este proceso se establece el método de aprendizaje a utilizar, la cantidad de capas ocultas, la cantidad de neuronas por cada capa oculta, la tasa de aprendizaje, y la arquitectura de la sección de la red neuronal que maneja los metadatos. Además, se establece el modelo de predicción idóneo, encontrando las variables de entrada y salida que minimizan el error en las predicciones, a partir de un conjunto de modelos candidatos. Por otra parte, hay elementos que no son modificados en el transcurso de las pruebas, como la función de activación de las neuronas y la función de normalización de los datos de entrada y salida. La función de activación utilizada para cada neurona es sigmoidal, con un rango de valores entre 0 y 1, debido a que se trabaja solo con variables de entrada y salida positivas, que luego también son normalizadas a un rango de valores entre 0 y 1.

La arquitectura utilizada consta de 2 bloques de capas independientes, donde cada uno maneja distintos datos de entrada, que luego se concatenan en un solo bloque que posee un nodo de salida común. Este modelo se presenta en la Figura 3.11, donde por un lado se ingresan datos de una serie de tiempos a un bloque de capas recurrentes, tipo LSTM, mientras que, por otro

lado, se ingresa un conjunto de metadatos, los cuales son manejados por otro bloque de capas, de manera independiente. Esta implementación está inspirada en el trabajo de Brébisson et al. (2015), el cual implementa 2 bloques de capas independientes, para manejar, por una parte, series de tiempo de geolocalizaciones de un taxi, y por otra, metadatos del viaje que realiza tal taxi. El bloque que maneja la serie de tiempo fue implementado con capas de neuronas recurrentes, mientras que los metadatos, con una capa que realiza un proceso de *embedding*.

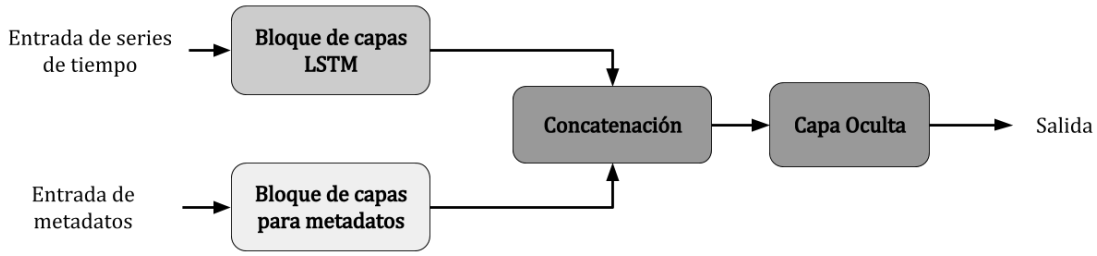


Figura 3.11. Esquema de la arquitectura de la RND.
Fuente: Elaboración propia, 2017.

El primer bloque de la red neuronal posee como entrada una serie de tiempo, con un conjunto de variables con las distancias y tiempos parciales del viaje de un bus. Estos datos son ingresados a un número por determinar de capas de neuronas LSTM. Se pretende que el bloque completo retorne un solo valor por neurona a partir de un conjunto de *timestep* de entrada, lo que comúnmente se denomina arquitectura de muchos a uno. Sin embargo, se realizan pruebas con más de una capa LSTM en el mismo bloque, puestas de manera secuencial, por lo que en ese caso se configura cada capa de manera distinta. Específicamente, las capas LSTM intermedias retornan una salida por cada *timestep* ingresado, en una arquitectura de muchos a muchos, mientras que la última capa del bloque implementa una arquitectura de muchos a uno, de manera que pueda realizar una sola salida. Esta salida puede ser considerada como una predicción parcial interna de la red neuronal, que solo toma en cuenta los patrones internos de la serie de tiempo, pero no toma en cuenta el contexto del viaje representada por los metadatos.

El segundo bloque de la red neuronal maneja la entrada de los datos de contextos o también llamados metadatos del viaje. Estos metadatos son ingresados a una capa especial, implementada por la biblioteca Keras, que mapea el conjunto de valores de los metadatos a un único vector unitario de *n* dimensiones. Este proceso se llama *word embedding,* el cual utiliza una matriz de conversión, cuyo tamaño está determinado por la cantidad de dimensiones del vector y por la cantidad de palabras del vocabulario. En este caso, el vocabulario está definido por los posibles valores que tomen las variables de los metadatos de un viaje. Esta implementación pretende entregar distintos valores a distintos escenarios de los metadatos, donde no existe necesariamente una correlación numérica entre la similitud de los valores de cada variable de metadatos y la similitud de escenarios encontrados. Por ejemplo, el tráfico vehicular y la demanda

de pasajeros los días lunes no necesariamente es similar al que ocurre los días martes, o también, el tiempo que espera de un bus en el paradero 16 de la ruta puede ser muy distinto al encontrado en el paradero 15.

La salida de ambos bloques es concatenada a un solo vector, para ser ingresada a una única capa oculta que pondera cada valor del vector a un único valor de salida. De esta manera, se considera no solo la predicción realizada por el bloque de capas recurrentes, sino también el procesamiento de los metadatos. Esta capa, que utiliza una función de activación sigmoidal, a través de sus pesos internos pondera la "predicción parcial" que realiza el bloque de capas recurrente, a través del procesamiento de la serie de tiempo, con la representación vectorial de los metadatos de la capa de *word embedding*.

Para realizar una predicción, utilizando RND, se debe ingresar una matriz bidimensional que constituye un conjunto de variables de una serie de tiempo parcial de largo fijo, y un vector unidimensional con los valores de metadatos representativos del contexto de tal serie. Se determina a priori que el largo de las series de tiempo parciales a ingresar, es de 10 *timestep*. Considerando que en promedio la cantidad de muestreos GPS por viaje es de alrededor de 100, cada secuencia ingresada constituye aproximadamente un 10% del viaje completo. Se considera que este número, es un balance entre la cantidad de series parciales que se generan a partir de un viaje de un bus, y la capacidad de la red neuronal, en sus capas recurrentes, de aprovechar la dependencia temporal de los distintos *timestep* de la serie de tiempo.

Para determinar las variables de entrada y salida idoneas para RND, se prueban distintos modelos de predicción que utilizan distintas combinaciones de variables de entrada y salida, a partir de un conjuntos de *features* extraidos de los *dataset* brutos. Cada combinación consiste en un conjunto específico de variables de metadatos, serie de tiempo o de salida. Un modelo de predicción candidato, consta de una combinación para cada tipo de variable. Se prueba el rendimiento de todas las combinaciones generadas de cada tipo de variable, entrenando distintas redes neuronales con los mismos hiper-parámetros, y dejando constante el conjunto de variables que no están siendo evaluadas. Al realizar este proceso, se pretende apreciar el efecto individual de cada combinación de variable en el rendimiento de RND, de manera de conseguir el conjunto de variables que minimice los errores en las predicciones. Las variables evaluadas, surgen a partir del cruce de información entre ambos *dataset* brutos, en un proceso de *feature engineering*. Esto es posible debido a que ambos *dataset* poseen el mismo corte temporal, además de columnas en común, como la patente de los buses.

Para los metadatos de entrada a RND, se implementan 2 combinaciones distintas, llamadas md_1 y md_2, las cuales poseen de 3 variables cada una. Éstas, son extraidas de un total de 6 variables candidatas, presentes en la Tabla 3.5. La combinación md_1, representada por la ecuación 3.6, contiene el identificador del último paradero visitado por un bus (*i*), la hora a la que llega a ese

paradero (T_i) y la hora del inicio del viaje (T_1), mientras que la combinación md_2, representada por la ecuación 3.7, utiliza el día de la semana en que se realiza el viaje (dw), el intervalo del día (*t*) y el identificador del último paradero. Los valores de estas variables caracterizan el contexto del recorrido de una serie de tiempo parcial específica, por lo que ciertas variables como *i* o T_i, pueden cambiar durante series parciales del mismo viaje.

Tabla 3.5. Definición de variables de metadatos candidatas para RND.

Nombre de la variable	Dominio	Rango de valores
Día de la semana (dw)	$\mathbb{N}_0$	$[0, 4]$
Intervalo del día del servicio (*t*)	$\mathbb{N}$	$[1, 7]$
Identificador último paradero (*i*)	$\mathbb{N}$	$[1, 43]$
Cuarto de hora del día (*q*)	$\mathbb{N}_0$	$[0, 67]$
Hora de inicio del viaje, en segundos (T_1)	$\mathbb{N}$	$[21.600, 82.600]$
Hora de llegada al último paradero visitado, en segundos (T_i)	$\mathbb{N}$	$[21.600, 82.600]$

Fuente: Elaboración propia, 2017.

$$md_1 = [i, T_1, T_i] \tag{3.6}$$

$$md_2 = [dw, t, i] \tag{3.7}$$

Para las variables de la serie de tiempo de RND, se generan 4 combinaciones distintas, llamadas sd_1, sd_2, sd_3 y sd_4. Éstas, provienen de un total de 3 variables candidatas, como se muestra en la Tabla 3.6. La combinación sd_1 es represententada por la ecuación 3.8, y solo contiene la variable del tiempo de viaje (TT_i) de un bus para la muestra *i*. La combinación sd_2 es represententada por la ecuación 3.9, y solo contiene la distancia que recorre un bus en un lapso de 30 segundos (Δd_i), entre el muestreo *i* y *i-1*. La combinación sd_3 es represententada por la ecuación 3.10, y contiene las variables TT_i y Δd_i. La combinación sd_4 es represententada por la ecuación 3.11, y contiene la variable TT_i y la distancia total recorrida por un bus hasta el muestreo *i* (d_i).

Tabla 3.6. Definición de variables de serie de tiempo candidatas para RND.

Variables de secuencia		
Nombre de la variable	**Dominio**	**Rango de valores**
Tiempo de viaje (TT_i)	$\mathbb{N}$	$[0, 4.800]$
Distancia recorrida últimos 30 segundos (Δd_i)	$\mathbb{N}_0$	$[0, 1{,}2]$
Distancia total recorrida (d_i)	$\mathbb{N}_0$	$[0, 18]$

Fuente: Elaboración propia, 2017.

$$sd_1 = [TT_i] \tag{3.8}$$

$$sd_2 = [\Delta d_i] \tag{3.9}$$

$$sd_3 = [TT_i, \Delta d_i] \tag{3.10}$$

$$sd_4 = [TT_i, d_i] \tag{3.11}$$

RND utiliza los valores de geolocalización del segundo dataset bruto, para calcular las distancias en las que recorre un bus en lapsos de 30 segundos (Δd_i). Para tales fines, se utiliza la fórmula de haversine que considera un modelo esférico de la tierra para calcular las distancias entre 2 puntos geográficos, basados en su latitud y longitud. La ecuación 3.12 muestra el cálculo que a realizar, donde d es la distancia a conocer, r es el radio de la tierra, φ_2 es la latitud del segundo punto geográfico, φ_1 es la latitud del primer punto geográfico, λ_2 es la longitud del segundo punto geográfico y λ_1 es la longitud del primer punto geográfico. Posteriormente, para realizar el cálculo de la distancia total (d_i) en la ruta que posee un bus en cada muestreo, se debe sumar todas las distancias de muestreos anteriores, en el mismo viaje. También, para calcular el tiempo de viaje (TT_i) de un bus en cada muestra de geolocalización, se debe sumar la cantidad de muestras que se han realizado en el viaje actual y multiplicarla por 30.

$$d = 2r * arcsen\left(\sqrt{sen^2\left(\frac{\varphi_2-\varphi_1}{2}\right) + cos(\varphi_1)\, cos(\varphi_2)\, sen^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right) \tag{3.12}$$

Para las variables de salida de la red neuronal, se eligen 2 candidatos, que representan las distancias que va recorriendo un bus en lapsos de tiempo de 30 segundos. La primera variable corresponde a $y_1 = d_{i+1}$, que es la distancia acumulada recorrida por un bus en el muestreo posterior al último *timestep* ingresado, mientras que la segunda variable $y_2 = \Delta d_{i+1}$, es la distancia que recorre un bus en un intervalo de 30 segundos. Ambas variables provienen de las

series de tiempo que fueron calculadas a partir del segundo *dataset* bruto, que posee la geolocalización de un conjunto de buses de un servicio.

Se define una nomenclatura que permita caracterizar de manera simple cada modelo de predicción, la cual es presentada en la ecuación 3.13. La variable *i* describe la combinación de metadatos que la compone, *j* describe la combinación de variables de series de tiempo que la componene, y *k* a la variable de salida. Esto quiere decir, por ejemplo que RND_{122}, se compone de las combinaciones variables de entrada md_1 y sd_2, para realizar predicciones de la variable y_2. Como puede notar, es posible definir 16 modelos de predicción distintos a partir de la combinatoria de las combinaciones de variables de metadatos, series de tiempo y salida.

$$y_k = RND_{ijk}(md_i, sd_j) \tag{3.13}$$

Cabe destacar, que no se pretende probar todas los modelos de predicciones posibles, debido a que compara el rendimiento de cada combinación a la vez. Por ejemplo, para encontrar la variable de salida idonea, se requieren 2 modelos distintos, que posean la misma combinacion de metadatos y variables de serie de tiempo, y distintas variables de salida cada uno, como son RND_{221} y RND_{222}. Extendiendo esta idea, solo es necesario realizar pruebas con 6 modelos distintos para encontrar las combinaciones de variables que minimizan el error en las predicciones. Estos modelos son RND_{222}, RND_{122}, RND_{212}, RND_{232}, RND_{242} y RND_{221}.

El proceso de normalización de los datos de entrada y salida de la red neuronal es implementado aplicando una función lineal, que escala cada variable a un rango entre 0 y 1. Este rango es elegido, debido a que coincide con el rango de valores de la función sigmoidal, y se presume que internamente en la red neuronal se realiza una diferenciación correcta de los valores. Este proceso se aplica tanto para las variables de entrada a la red neuronal, como a los valores de referencia de la salida de la red neuronal ($\hat{y}$), provenientes del *dataset*, al momento de realizar el proceso de entrenamiento. Sin embargo, este proceso no aplica a las variables de metadatos, debido a que son mapeadas a un espacio vectorial unitario en el proceso de *word embedding*. El cálculo por realizar es expresado por la ecuación 3.14, donde la variable normalizada x_i', se calcula a partir de división entre su valor original x_i y el valor máximo que puede conseguir $\max(\mathrm{x_i})$. Por el contrario, el cálculo de la denormalización es expresado por la ecuación 3.15, que multiplica la variable normalizada por el valor máximo posible, para obtener su valor en el rango original. Al realizar este proceso, se modifica el dominio de las variables que se encuentren en los números naturales ($\mathbb{N}$), quedando todas dentro del conjunto de los números reales ($\mathbb{R}$).

$$x_i' = \frac{x_i}{max(x_i)} \tag{3.14}$$

$$x_i = max(x_i) * x_i' \tag{3.15}$$

3.3.3 Red neuronal de *headway* (RNH)

Este modelo implementa una red neuronal recurrente para predecir el *headway* entre un par de buses en el siguiente paradero de la ruta. Este modelo implementa una arquitectura similar a MPD, presentado en la Figura 3.11, pero cuyos hiperparámetros son ajustados específicamente para manejar secuencias de *headway*. En este sentido, este modelo no predice el comportamiento de un bus a lo largo de su viaje, sino que predice el comportamiento relativo entre un par de buses a lo largo de su recorrido. Se ingresa un conjunto de variables de serie de tiempo, con el *headway* de un par de buses en los últimos paraderos muestreados, junto con un conjunto de metadatos de los viajes de los buses, para predecir el *headway* en el siguiente paradero de la ruta. Al igual que RND, este modelo de predicción es de tipo One-Step-Ahead, debido a que realiza predicciones solo a 1 *timestep* posterior al último ingresado en la serie de tiempo. Se pretende sacar provecho de la capacidad de las redes neuronales recurrentes de manejar la dependencia temporal de los datos de entrada y de reconocer patrones en series de tiempo. Este enfoque permite aplicar de manera directa un algoritmo de detección de *bunching*, sobre las predicciones realizadas por la red neuronal. En adelante este modelo es nombrado RNH, para identificarlo con mayor facilidad.

La arquitectura de RNH, en términos generales, es equivalente a la arquitectura que implementa RND. Como se aprecia en la Figura 3.11, la arquitectura consiste en 2 bloques de capas independientes, donde el primero maneja los datos de entrada de series de tiempo y el segundo metadatos dicha serie. Luego, se concatenan las matrices resultantes de ambas partes en un solo bloque de capas para entregar una salida, con la predicción que realiza la red neuronal. Sin embargo, los modelos de predicción a probar con esta arquitectura son distintos que en RND, debido a que se realizan predicciones de secuencias de *headway* entre un par de buses, y, por lo tanto, los metadatos contextualizan el paso de ambos buses en la ruta. De la misma manera, el proceso de ajuste de hiper-parámetros (véase sección 3.4.4) consigue los valores que minimizan los errores en las predicciones para un modelo de predicción específico, por lo que valores para cada hiper-parámetro idóneos para RNH pueden diferir de los encontrados para RND. Durante este proceso, se establece la tasa de aprendizaje, el método de aprendizaje, la cantidad de capas ocultas y neuronas por cada capa oculta, además de la función de activación que utiliza cada neurona.

Se establece que el largo de las series de tiempo parciales a ingresar a RNH es de 4 *timestep*. De manera similar al proceso realizado para RND, debido a que las series de tiempo de *headway* entre un par de buses, poseen un largo fijo, determinado por la cantidad de paraderos de la ruta, es decir 43, se establece que cada secuencia a ingresar a la RNH debe ser de aproximadamente un 10% del largo total. Se estima que este valor es un balance entre la cantidad de series de

tiempo parciales que se general a partir de serie de tiempo, que corresponde a 39, y la capacidad de las capas recurrentes para aprovechar la dependencia temporal de los valores de la serie.

De manera homóloga al proceso que se realiza con RND, se realizan pruebas de rendimiento de distintos modelos de predicción, a partir de distintas combinaciónes de variables de entrada y salida, con el fin de determinar el modelo de predicción que minimiza los errores en las predicciones. Una combinación consiste en un subconjunto de variables de metadatos, serie de tiempo o variables de salida específicos, extraidos de un conjunto mayor de *features* candidatos. Cada *feature*, es extraido a partir del cruce de información a partir del primer dataset de bruto, junto con información del Programa de Operación vigente en el corte temporal utilizado, en un proceso de *feature engineering*. Un modelo de predicción candidato, utiliza una combinación de cada tipo de variable. Se prueba el rendimiento de todas las combinaciones generadas de cada tipo de variable, entrenando distintas redes neuronales con los mismos hiper-parámetros, y dejando constante el conjunto de variables que no están siendo evaluadas. Al realizar este proceso, se pretende apreciar el efecto individual de cada combinación de variable en el rendimiento de RNH, de manera de conseguir el conjunto de variables que minimice los errores en las predicciones.

Para los metadatos de entrada a RNH, se implementan 2 combinaciones de variables distintas, llamadas mh_1 y mh_2, las cuales poseen de 3 variables cada una. Éstas combinaciones son extraidas de un total de 4 variables candidatas, presentes en la Tabla 3.7. La combinación mh_1, representada por la ecuación 3.16, contiene el día de la semana en que se realiza el par de viajes consecutivos (*dw*), el intervalo del día (*t*) y el *headway* inicial h_1 entre ambos viajes, mientras que la combinación mh_2, representada por la ecuación 3.17, reemplaza la última variable por el índice del último paradero visitado (*i*). Las variables dw y t son calculadas a partir del valor la fecha y la hora del inicio de alguno de los 2 viajes, utilizando la fecha para aplicar un algoritmo que calcula el día de la semana, y utilizando la hora para calcular el intervalo del día, a partir de la información del Programa de Operación del servicio “I09”, presente en la Tabla 3.3. Por otro lado, el valor de índice del último paradero visitado pretende dar contexto al porcentaje de avance en el que se encuentra cierta secuencia de valores de *headway*.

Tabla 3.7. Definición de variables de metadatos candidatas para RNH.

Descripción de variable	Dominio	Rango de valores
Dia de la semana (*dw*)	$\mathbb{N}_0$	$[0, 4]$
Intervalo del día (*t*)	$\mathbb{N}$	$[1, 7]$
Índice de último paradero visitado (*i*)	$\mathbb{N}$	$[1, 43]$
Headway inicial (h_1)	$\mathbb{N}_0$	$[0, 3.600]$

Fuente: Elaboración propia, 2017.

$$mh_1 = [dw, t, h_1] \tag{3.16}$$

$$mh_2 = [dw, t, i] \tag{3.17}$$

Para las variables de entrada de series de tiempo, se implementan 3 combinaciones de variables distintas, llamadas sh_1 y sh_2, sh_3. Estas combinaciones, son extraidas de un total de 4 variables candidatas, definidas en la Tabla 3.8. La combinación sh_1, representada en la ecuación 3.18, contiene todas las variables disponibles, como el *headway* entre un par de viajes de buses, i y j, en el último paradero en común k (h_{ijk}), el tiempo de viaje de ambos buses en el último paradero (TT_{ik} y TT_{jk}) y la tasa de avance del viaje (τ). La combinación sh_2, representada en la ecuación 3.19, utiliza h_{ijk} y τ, mientras que la combinación sh_3, representada por la ecuación 3.20, solo se compone de la variable h_{ijk}. La variable h_{ijk} se calcula restando la hora de llegada al paradero (T_{ik}) entre ambos buses, siguiendo la ecuación 3.21. La variable TT_{ik}, se calcula restando la hora de inicio del viaje *i* (T_{i1}) con la hora de llegada al paradero *k* (T_{ik}), siguiendo la ecuación 3.22. La tasa de avance τ, se calcula dividiendo el índice del último paradero común entre ambos viajes (*k*), con el total de paraderos de la ruta, que en este caso es 43.

Tabla 3.8. Definición de variables de secuencia candidatas para RNH.

Descripción de variable	Dominio	Rango de valores
Headway último paradero (h_{ijk})	$\mathbb{Z}$	$[-3.600, 3.600]$
Tiempo de viaje 1 en último paradero (TT_{ik})	$\mathbb{N}_0$	$[0, 4.800]$
Tiempo de viaje 2 en último paradero (TT_{jk})	$\mathbb{N}_0$	$[0, 4.800]$
Tasa de avance del viaje (τ)	$\mathbb{R}$	$[0, 1]$

Fuente: Elaboración propia, 2017.

$$sh_1 = [h_{ijk}, TT_{ik}, TT_{jk}, \tau] \tag{3.18}$$

$$sh_2 = [h_{ijk}, \tau\,] \tag{3.19}$$

$$sh_3 = [h_{ijk}] \tag{3.20}$$

$$h_{ijk} = T_{ik} - T_{jk} \tag{3.21}$$

$$TT_i = T_k - T_1 \tag{3.22}$$

Para las variables de salida del RNH, se eligen 2 candidatos, ambas relacionadas con el valor de *headway* de un par de buses en cada paradero. La primera variable corresponde a $y_1 = h_{ijk+1}$, que es el valor del headway entre un par de viajes de buses, *i* y *j*, en el paradero *k+1*, mientras que la segunda variable corresponde a $y_2 = \Delta h_{ijk+1}$, que es la diferencia en el valor de *headway*

entre el paradero *k*+1 y *k*. Ambas variables provienen de las series de tiempo que fueron calculadas a partir del primer *dataset* bruto. Las diferencias de headway entre cada paradero, para un cierto par de viajes, se calculan a partir de la serie de *headway*, restando los valores de *headway* consecutivos, siguiendo el cálculo de la ecuación 3.23.

$$\Delta h_{ijk+1} = h_{ijk+1} - h_{ijk} \tag{3.23}$$

Se define una nomenclatura que permita caracterizar de manera simple cada modelo de predicción, la cual es presentada en la ecuación 3.24. La variable *i* describe la combinación de metadatos que la compone, *j* describe la combinación de variables de series de tiempo que la componene, y *k* a la variable de salida. Esto quiere decir, por ejemplo que RNH_{122}, se compone de las combinaciones variables de entrada mh_1 y sh_2, para realizar predicciones de la variable y_2. Como puede notar, es posible definir 12 modelos de predicción distintos a partir de la combinatoria de las combinaciones de variables de metadatos, series de tiempo y salida.

$$y_k = RNH_{ijk}(mh_i, sh_j) \tag{3.24}$$

Cabe destacar, que no se pretende probar todas los modelos de predicciones posibles, debido a que compara el rendimiento de cada combinación a la vez. Por ejemplo, para encontrar la variable de salida idonea, se requieren 2 modelos distintos, que posean la misma combinacion de metadatos y variables de serie de tiempo, y distintas variables de salida cada uno, como son RND_{221} y RND_{222}. Extendiendo esta idea, solo es necesario realizar pruebas con 5 modelos distintos para encontrar las combinaciones de variables que minimizan el error en las predicciones. Sin embargo, en este caso se realizan pruebas sobre 7 modelos de predicción distintos, debido a que para encontrar los valores de distintas variables de hiper-parámetros se utilizan distintos modelos de predicción. Estos modelos son RNH_{232}, RNH_{231}, RNH_{132}, RNH_{212}, RNH_{222}, RNH_{131} y RNH_{111}.

El proceso de normalización de los datos de entrada y salida de RNH, fue implementado escalando el rango de valores de cada variable de manera lineal, a un rango entre -1 y 1. Este rango es elegido, debido a que coincide con el rango de valores de la función de activación, y se presume que internamente en la red neuronal se realiza una diferenciación correcta de los valores. Este proceso se aplica tanto para las variables de entrada de series de tiempo (x_i), como a los valores de referencia de la salida de la red neuronal ($\hat{y}$), provenientes del *dataset*, al momento de realizar el proceso de entrenamiento. Este proceso no aplica a las variables de metadatos, al igual ocurre con RND, debido que son mapeadas a un espacio vectorial unitario en el proceso de *word embedding*. El cálculo de la normalización es expresado por la ecuación 3.25, y el cálculo de la denormalización es expresado por la ecuación 3.26.

$$x_i' = 2 * \frac{x_i}{max(x_i)} - 1 \quad (3.25)$$

$$x_i = \frac{max(x_i)}{2} * (x_i' + 1) \quad (3.26)$$

3.3.4 Modelo de promedios históricos (MPH)

Se implementa un modelo de predicción de tiempos de viaje, en base a los promedios históricos de los tiempos de viaje de los buses del mismo servicio. Se utilizan ventanas de tiempo de 30 minutos para capturar datos parciales de los tiempos de viaje de un bus, de manera similar a la implementada por el resto de modelos. Luego, las predicciones se realizan utilizando estos datos parciales capturados y los promedios históricos de los tiempos de viaje.

Las ecuaciones 3.27 y 3.28 describen el cálculo del modelo de promedios históricos. La variable *i*, describe una ventana de tiempo en un viaje específico de un bus, que constituye un conjunto parcial de paraderos, mientras que *u* es el último paradero de tal ventana y *j* es un índice de paradero posterior a *u*. TT_{ij} es el tiempo de viaje predicho en el paradero *j* en el viaje de la ventana de tiempo *i*. *dw* es el día de la semana en que ocurre el viaje del bus de la ventana *i*. *t* es el intervalo del día del viaje de la ventana *i*. $TT_{Promedio}$ es el tiempo de viaje promedio que demora un bus en llegar a un paradero *j*, en un día de la semana *dw* y en un intervalo del día *t*, según los casos presentes en el *dataset* de entrenamiento. *v* es el índice de un paradero dentro de la ventana de tiempo *i*, por lo tanto TT_{iv} es el tiempo de viaje predicho en el paradero *v* de la ventana de tiempo *i*. Finalmente, *k* es la constante de proporcionalidad del tiempo promedio para el viaje de la ventana *i*.

$$k = \frac{\sum_{v=1}^{v=u} \frac{TT_{iv}}{TT_{Promedio}(dw,t,v)}}{u} \quad (3.27)$$

$$TT_{ij} = TT_{Promedio}(dw, t, j) * k\ ,\ j > u \quad (3.28)$$

Primero, se calcula el valor de *k*, el cual consiste en la media de las constantes de proporcionalidad entre los tiempos de viaje de la ventana *i* actual y los tiempos de viaje promedios, al correlacionar los paraderos recorridos, en el mismo día de la semana y en el mismo intervalo horario del día. En términos prácticos, k es un término de ajuste que compara los tiempos de viaje de una cierta ventana de tiempo seleccionada con los tiempos de viaje promedio, para el mismo *dw* y *t*. Luego, este valor *k* se multiplica por el $TT_{Promedio}$ correspondiente, para cada paradero *j* posterior a la ventana de tiempo, para realizar la predicción de los tiempos de viaje del resto del recorrido. En la Figura 3.12 se observa un ejemplo de predicción realizada por MPH, del viaje 34° del día 5 de diciembre de 2016. La línea naranja representa el viaje real presente en el *dataset* y la línea gris representa el viaje promedio para los días lunes en el segundo intervalo horario. Se

utilizan los datos reales hasta el paradero 16, al establecer una ventana de tiempo entre las 8:00 y las 8:30, para calcular el valor de *k* que muestre la constante de proporcionalidad promedio entre los datos reales y los del promedio histórico. Luego, se genera la secuencia de tiempos predichos a partir del paradero 17, en la línea azul, al multiplicar el valor de *k* a los tiempos de viaje promedio.

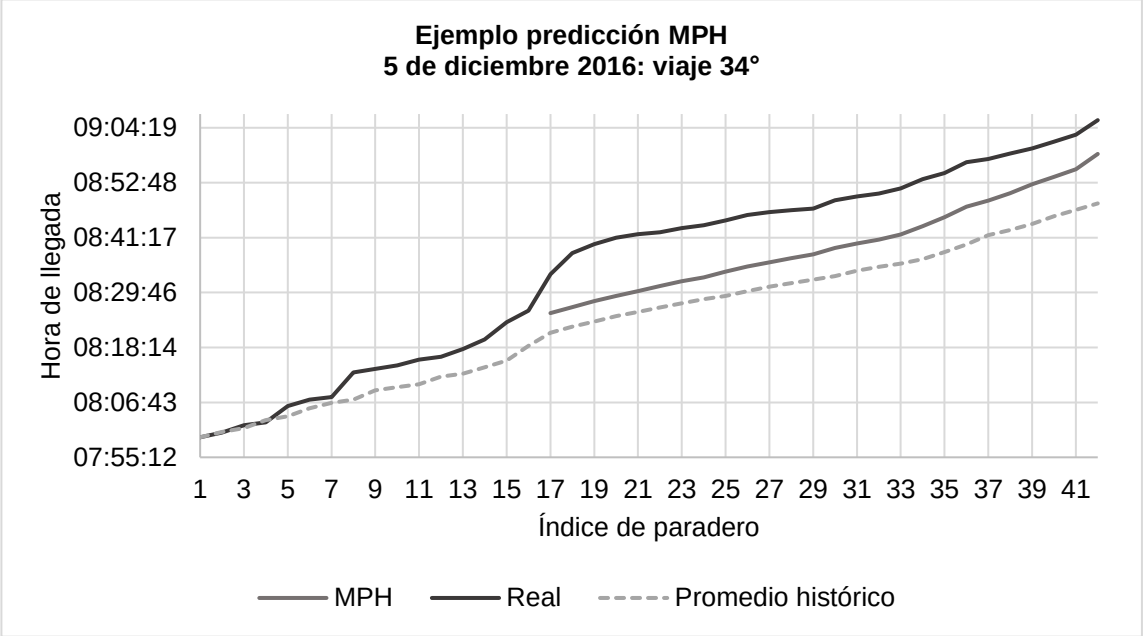


Figura 3.12. Ejemplo de hora de llegada para cada paradero de la ruta, para predicciones de MPH, datos reales y promedio histórico.
Fuente: Elaboración propia, 2017.

Para realizar pruebas de rendimiento de las predicciones en este modelo y poder compararlas con los otros modelos, se realiza partición del *dataset* en datos de entrenamiento y datos de prueba. Los datos de entrenamiento son los utilizados para calcular los tiempos de viaje promedio para cada día de la semana y cada horario del día. De esta manera, se genera un total de 35 secuencias de tiempos de viaje promedio, considerando sólo 5 días de la semana, de lunes a viernes, y 7 horarios distintos. Por otra parte, los datos de prueba son utilizados para realizar las predicciones de los tiempos de viaje en base a los viajes promedio calculados y evaluar su precisión.

Se utilizan los tiempos de viaje de los buses al llegar a cada paradero debido a la facilidad de hacer correlaciones entre los tiempos de viaje en la ruta, a diferencia de los tiempos de viaje dados por el muestreo GPS. La posición de los paraderos en la ruta es fija, por lo que el tiempo de viaje de un bus en llegar a cada paradero, es directamente comparable entre los distintos viajes, facilitando la utilización de promedios históricos. Por el contrario, la geolocalización de los

buses se realiza en intervalos de tiempo fijo, donde la distancia que recorren en estos intervalos es variable, lo que conlleva que la longitud de la serie de distancias de un viaje en el tiempo también sea variable. Cabe destacar que es posible realizar interpolaciones que permitan entregar tiempos de viaje a ciertos intervalos de distancia fijos en la ruta, pero el proceso es menos natural y puede acarrear a márgenes de error mayores.

3.3.5 Modelos de persistencia para series de tiempo (MPST)

Se implementan 2 modelos de persistencia o también llamado predictores ingenuos, utilizando las series de tiempo de RND y RNH, para comparar los resultados obtenidos. Estos modelos establecen como predicción al siguiente *timestep* en una serie de tiempo, como el valor del último *timestep* observado. Esto implica que estos modelos no utilizan información respecto a los valores del pasado de una serie de tiempo, por lo que solo pueden describir un patrón estacionario. Para series de tiempo crecientes, por ejemplo, estos modelos no permiten predecir una relación entre el paso del tiempo y el crecimiento de las variables, por lo que no se considera apropiada su implementación en estos casos. La ecuación 3.29 muestra el comportamiento de un modelo de persistencia, dado una variable *X* en el tiempo *t*.

$$X_{t+1} = X_t \tag{3.29}$$

Para determinar la posibilidad de realizar predicciones sobre fenómenos que poseen una dependencia en el tiempo, es necesario conocer las variables involucradas, que afectan directamente al comportamiento del fenómeno a través del tiempo. Además, debe existir una dependencia temporal entre los valores de una serie, tal que se cumpla la ecuación 3.30 donde $X(t)$ es el vector de variables de una serie de tiempo en el tiempo *t*, que sumado con el vector $\Phi(\tau)$, que describe la variación de los valores de $X(t)$ ocurridas en el lapso de tiempo τ, constituye el vector $X(t+\tau)$ de la serie en el tiempo $t+\tau$.

$$X(t+\tau) = X(t) + \Phi(\tau) \tag{3.30}$$

El modelo de persistencia es utilizado, en este caso, para determinar la predictibilidad de la serie de tiempo utilizada en RND y RNH. Existen series de tiempo donde no es posible conocer las variables involucradas en la función $\Phi(\tau)$, y por tanto las variaciones que se produzcan entre cada lapso de tiempo o *timestep* son completamente aleatorias. Estas series de tiempo se denominan caminos aleatorios (en inglés: *random walk*) (Brownlee, A Gentle Introduction to the Random Walk for Times Series Forecasting with Python, 2017) y es necesario determinar si las series de tiempo utilizadas en los modelos implementados siguen estas características, de modo de evaluar la idoneidad de su modelamiento. Se compara la precisión de la red neuronal y el modelo de persistencia utilizando la misma serie de tiempo, para determinar la existencia de este fenómeno.

3.4 MÉTODO DE ENTRENAMIENTO Y PRUEBA DE MODELOS PROPUESTOS

Para realizar el proceso de entrenamiento y de prueba sobre cada modelo propuesto, se separa el *dataset* en dos secciones, de manera que cada uno tenga un conjunto de datos distinto. Esto se realiza ordenando temporalmente los datos de los viajes de los buses, desde los más antiguos a los más recientes, y se separa en dos partes, dejando el 80% más antiguo para entrenamiento y el 20% más reciente para pruebas. Se desea tener control sobre la sobre-especialización del modelo, al realizar pruebas sobre datos a los que la red neuronal nunca ha sido expuesta, y por lo tanto la capacidad de realizar predicciones está dada por la capacidad de la red neuronal de generalizar patrones sobre los datos de entrenamiento. De esta manera, cuando se evalúe el modelo entrenado sobre nuevos datos, el rendimiento es similar al que se consigue con los datos de prueba.

Es necesario aclarar que no se implementan procedimientos más elaborados que, por ejemplo, utilicen validación cruzada para conocer con mayor detalle el rendimiento de un modelo al utilizar distintas particiones de datos de entrenamiento, validación y prueba. Esto, se debe a que el entrenamiento de cada subconjunto generado multiplica el tiempo de entrenamiento total de los modelos, proporcionalmente a la cantidad de particiones que se realizan. Este aspecto tiene un crucial impacto en el proyecto, dado el considerable conjunto de datos disponibles, las limitadas capacidades de la máquina para realizar las pruebas y el número limitado de horas en los que se debe llevar a cabo el proyecto.

3.4.1 Método de entrenamiento de redes neuronales

Para los modelos de redes neuronales propuestos, se realizan pruebas sobre 2 métodos de entrenamiento, basados en el descenso de gradiente estocástico. Ambos, son implementados en la biblioteca Keras, por lo que solo se requiere determinar los parámetros con los que se utilizan. El primer método consiste en una implementación del método SGD, que posee un único parámetro a determinar, el cual es la tasa de aprendizaje (η). El segundo método consiste en la implementación del método ADAM, cuyos parámetros a determinar son la tasa de aprendizaje (η), la tasa de olvido para el valor de gradiente (β_1), la tasa de olvido para el valor del gradiente al cuadrado (β_2), y un término de regularización que previene la división por 0 (ε). Para ambos métodos, la tasa de aprendizaje será determinada en función de un conjunto de pruebas a realizar. Sin embargo, los valores para el resto de los parámetros del método ADAM son determinados con anterioridad, debido a que no se realizan pruebas sobre estos, al considerarse parámetros estandarizados cuya complejidad para realizar pruebas rebasa los alcances de este trabajo. Por lo tanto, β_1 asume un valor de 0,9, β_2 un valor de 0,99, y ε un valor de 10^{-8}.

Se decide probar solo con estos 2 métodos, debido a que SGD constituye una implementación simple del método de gradiente estocástico, sobre la cual es posible utilizar como base para comparar los resultados de métodos más elaborados. Por otra parte, el método ADAM también se basa en el método de gradiente estocástico, pero implementa mejoras provenientes de otros métodos derivados, como Momentum y RMSProp, que pueden permitir obtener una convergencia más estable en el aprendizaje de las redes neuronales. Del método Momentum, se implementan factores de olvido que permiten hacer combinaciones lineales del valor del gradiente actual, con relación al valor obtenido en la actualización de pesos anterior. Del método RMSProp, se implementa la utilización del cuadrado del gradiente, incorporando factores de olvido para hacer combinaciones lineales de los valores de cuadrado de gradiente actual y de la actualización de pesos anterior.

Se decide no implementar técnicas que permiten enfrentar el problema de la aleatoriedad de los valores de las matrices de pesos de las redes neuronales, al inicio del proceso de entrenamiento. Estos métodos intentan buscar o generar valores iniciales que permitan converger a un mejor rendimiento de una red neuronal luego del proceso de entrenamiento. Sin embargo, algunos de estos procedimientos se basan en la realización de múltiples pruebas utilizando distintas matrices iniciales, lo cual multiplica el tiempo de entrenamiento de los modelos, según la cantidad de pruebas que se realicen. Además, estos algoritmos al aplicarse a problemas de optimización de tipo heurísticos, como es el entrenamiento de redes neuronales, no se puede asegurar que se va a conseguir mejores resultados que realizando 1 solo proceso de entrenamiento. Esto, sumado a las limitaciones del tiempo disponible para la realización del proyecto y la limitada capacidad de procesamiento de la máquina utilizada, impiden la implementación de alguna técnica de esta índole.

3.4.2 Método de prueba

A cada modelo implementado se le aplica un algoritmo de prueba, de manera de conocer la precisión que posee en las predicciones que realiza. Utilizando un cierto modelo, se realizan predicciones de los distintos casos del *dataset*, tanto para el *dataset* entrenamiento, como el de prueba, registrando los valores obtenidos. Luego, se denormalizan los valores obtenidos, de manera de obtener resultados en la misma escala que el *dataset* original. Finalmente, se comparan los resultados obtenidos con respecto a los datos reales, utilizando distintas métricas de precisión. Se realizan predicciones sobre ambas partes del *dataset* y no solo el *dataset* de pruebas, debido que al comparar los resultados obtenidos con los datos que se utilizaron para el entrenamiento del modelo, con respecto a un conjunto de datos a los que dicho modelo no se ha expuesto, es posible detectar el sobre-entrenamiento. Si la precisión de las predicciones realizadas en el *dataset* de prueba es inferior al del *dataset* de entrenamiento, entonces probablemente el modelo esté sobreentrenado. También, obtener resultados de las métricas en

la escala de valores del propio de *dataset*, ya sea en segundos o en metros, permite tener una noción más clara de la precisión del modelo en términos prácticos.

3.4.3 Métricas de precisión

Para evaluar la precisión de las predicciones de cada modelo se utilizan 4 funciones distintas, las cuales corresponden al error cuadrático medio (MSE), raíz del error cuadrático medio (RMSE), error absoluto medio (MAE), y error porcentual absoluto medio (MAPE). Estas métricas son descritas por las ecuaciones 3.31 a 3.34, respectivamente, donde n es la cantidad de casos totales de un *dataset*, y_i es el valor de la predicción, para el caso i, realizada por algún modelo a evaluar, e $\hat{y}_i$ es el valor real de la variable de salida para el mismo caso.

El MSE, calcula la media del cuadrado de la diferencia de entre los valores predichos y los valores reales, para un conjunto de casos. Esta función se utiliza como función objetivo en el proceso de entrenamiento de las redes neuronales, por lo que se considera necesario conocer su valor en diferentes contextos de prueba.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{3.31}$$

El RMSE, calcula la raíz de la media del cuadrado del error entre un conjunto de valores predichos y sus respectivos valores reales. Esta función da una noción del error promedio de las predicciones en la escala de la variable a predecir, pero teniendo la particularidad en los casos en que los errores son mayores a la media reciben una mayor ponderación en el valor final, al elevarse al cuadrado. Esto significa que el valor de la función aumenta a medida que aumenta la dispersión de los errores, en caso de que la media se mantenga constante. Por lo que, en términos prácticos, al compararse el RMSE con el valor de MAE, se tiene una noción de esta variación.

$$\mathrm{RMSE} = \sqrt{\frac{1}{\mathrm{n}}\sum_{\mathrm{i}=1}^{\mathrm{n}}(\hat{\mathrm{y}}_{\mathrm{i}} - \mathrm{y}_{\mathrm{i}})^2} \tag{3.32}$$

El MAE, calcula la media entre el valor absoluto de la diferencia entre un conjunto de valores predichos y sus respectivos valores reales, para una cierta variable. Al utilizar el valor absoluto de los errores, es posible tener una noción de la distancia promedio entre los valores de las predicciones y los valores reales, sin importar si éstos están por debajo o por sobre el valor real. También, esta métrica pondera el error absoluto de cada caso de manera equitativa a diferencia del RMSE, no viéndose afectado por la dispersión de los errores.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i| \tag{3.33}$$

El MAPE, calcula el porcentaje del error absoluto porcentual entre un conjunto de valores predichos y sus respectivos valores reales. Es posible que exista una división por 0, cuando el valor real de una predicción sea 0, por lo que al denominador de la ecuación se le suma una constante ε con valor 10^{-3}, para evitar este problema. Esta función es utilizada para evaluar la implementación realizada por Gurmu & Fan (2014), en la cual se basa RNP, por lo que, debe implementarse para realizar una comparación directa del rendimiento de cada implementación.

$$MAPE = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{\hat{y}_i - y_i}{\hat{y}_i + \varepsilon}\right| \tag{3.34}$$

3.4.4 Ajuste de hiper-parámetros en redes neuronales

Para encontrar los hiper-parámetros y métodos idóneos para maximizar la precisión de los modelos de redes neuronales implementados, se implementa un procedimiento experimental que permite comparar el impacto de los cambios en los valores de cada hiper-parámetro en el rendimiento de las predicciones. Este método no aplica para el modelo basado en promedios históricos y el modelo de persistencia, debido a la ausencia de parámetros variables por determinar. A continuación, se presenta la secuencia de pasos a seguir:

- Se realizan múltiples procesos de entrenamiento a un mismo modelo de predicción, considerando las mismas variables de entrada y salida, modificando en cada caso solo el valor de un hiper-parámetro a la vez, y dejando todos los otros hiper-parámetros constantes. Cabe destacar que, se parte con un conjunto de valores de hiper-parámetros aleatorio. Los hiperparámetros a ajustar son la tasa de aprendizaje, el método de aprendizaje, la cantidad de neuronas por capa oculta, la cantidad de capas ocultas, y en ciertos modelos, la función de normalización y la función de activación. Para cada hiperparámetro se prueba en promedio 3 valores distintos de manera de poder visualizar una tendencia del efecto de los cambios de valores en el rendimiento general del modelo.

- Luego, se comparan los resultados obtenidos para cada valor del mismo hiper-parámetro, con el fin de determinar el más idóneo, tomando en cuenta la precisión obtenida, los tiempos de ejecución, la dispersión de los errores, la velocidad de convergencia, entre otros aspectos.

- Los mejores valores para cada hiper-parámetro, al ser evaluados individualmente, se utilizan para constituir la combinación de valores definitiva para cierto modelo. Cabe destacar, que se parte de la premisa que los efectos que provoca cada hiper-parámetro en el rendimiento de un modelo no afectan el comportamiento de los otros hiper-parámetros, y por lo tanto los efectos de cada uno de estos pueden ser combinados.

- Finalmente, se realiza un último proceso de entrenamiento, con la mejor combinación de valores de los hiper-parámetros para el modelo estudiado, generando los resultados definitivos.

3.5 MÉTODO DE PREDICCIÓN DE *BUNCHING* DE BUSES

El método de predicción de *bunching* de buses se realiza en 5 etapas, las cuales permiten procesar datos parciales de los viajes de un conjunto de buses, hasta obtener detecciones de eventos de headway para cada par de buses, en un conjunto de paraderos de la ruta. En la primera etapa, se capturan datos parciales de los viajes de los buses, basadas en ventanas de tiempo. En la segunda etapa, predicen los tiempos de viaje en base a los datos capturados, utilizando alguno de los modelos propuestos. En la tercera etapa, se generan secuencias de *headway* de un conjunto de pares de viajes, a partir de los viajes capturados anteriormente. En la cuarta etapa, se aplican un algoritmo de detección de *bunching* sobre cada secuencia de *headway*. Finalmente, en la quinta etapa se realizan pruebas de rendimiento sobre las predicciones de *bunching* realizadas, comparándolas con los que se obtienen al detectar *bunching* sobre los datos reales.

3.5.1 Captura de datos en ventanas de tiempo

A través de la predicción del comportamiento de un conjunto de buses, se quiere predecir la ocurrencia del fenómeno de *bunching*. Este fenómeno está intrínsecamente relacionado con el contexto del sistema de transportes en un momento determinado, en cuanto al nivel de demanda de pasajeros, la congestión vehicular, entre otros aspectos. Además, implica la participación de al menos un par de buses de un mismo servicio, que estén realizando su recorrido en un determinado momento. En este sentido, se requiere que un sistema que prediga la ocurrencia de este fenómeno tenga en consideración estos aspectos, ya sea de manera directa al agregarla como dato de entrada, o implícitamente a través del comportamiento de algún otro sistema que lo contenga. En este caso, se tiene un conjunto de modelos que predicen el comportamiento de los tiempos de viaje de uno o más buses, dependiendo del caso, en un contexto determinado.

Para realizar predicciones de eventos de *bunching*, se emula el contexto real en que el fenómeno ocurre. En un cierto instante, entes como el Centro de Monitoreo de Buses, de DTPM, y debe aplicar planes de contingencia antes que ocurran eventos problemáticos en un determinado servicio. Basado en ese contexto, para realizar predicciones se implementan ventanas de tiempo, que abarcan las secuencias de tiempos de viaje del conjunto buses, comprendidos en un rango horario específico, en una fecha específica. Este lapso de tiempo es de 30 minutos, ya que permite capturar una suficiente cantidad de datos y, sin embargo, no captura la serie completa de tiempos de viaje, debido a que el recorrido de un bus dura entre 40 minutos y 1 hora y 20 minutos en realizarse. Esto permite realizar predicciones en cada secuencia parcial de tiempos de viaje

utilizando cada uno de los modelos implementados, hasta completar el viaje o hasta cumplir 30 minutos de predicciones, posteriores al término de una ventana de tiempo.

En la Figura 3.13, se presenta un gráfico con la hora de llegada de un conjunto de buses a cada paradero de la ruta, capturados en una ventana de tiempo de media hora, entre las 8:00 y las 8:30 horas, del día 5 de diciembre de 2016, para el servicio I09. Las líneas verticales representan la hora de inicio y término de la ventana, por lo que entre esos límites se encuentran los datos parciales capturados de los tiempos de viaje de los buses. A la derecha de la línea vertical situada a las 8:30, se encuentran los tiempos de viaje a predecir en base a los datos de la ventana de tiempo. Estos últimos datos, son utilizados posteriormente como referencia en las predicciones realizadas al momento de realizar pruebas de rendimiento.

En total, se utiliza un conjunto de 700 ventanas de 30 minutos, procedentes de *dataset* de prueba de cada modelo, cubriendo 25 días, 5 semanas, desde las 7:00 a las 21:00 horas. Específicamente, los días abarcados corresponde a los días lunes a viernes, desde el 7 de noviembre al 9 de diciembre de 2016. Cabe destacar que cada modelo posee su propio *dataset* de prueba, debido a que poseen distintas variables de entrada y salida, pero todos comparten el conjunto de viajes al ser separados temporalmente, constituyendo el 20% de los viajes más recientes del dataset original. Por este motivo, la implementación del método de captura de datos en ventanas de tiempos y sus respectivas predicciones, es distinta para cada modelo.

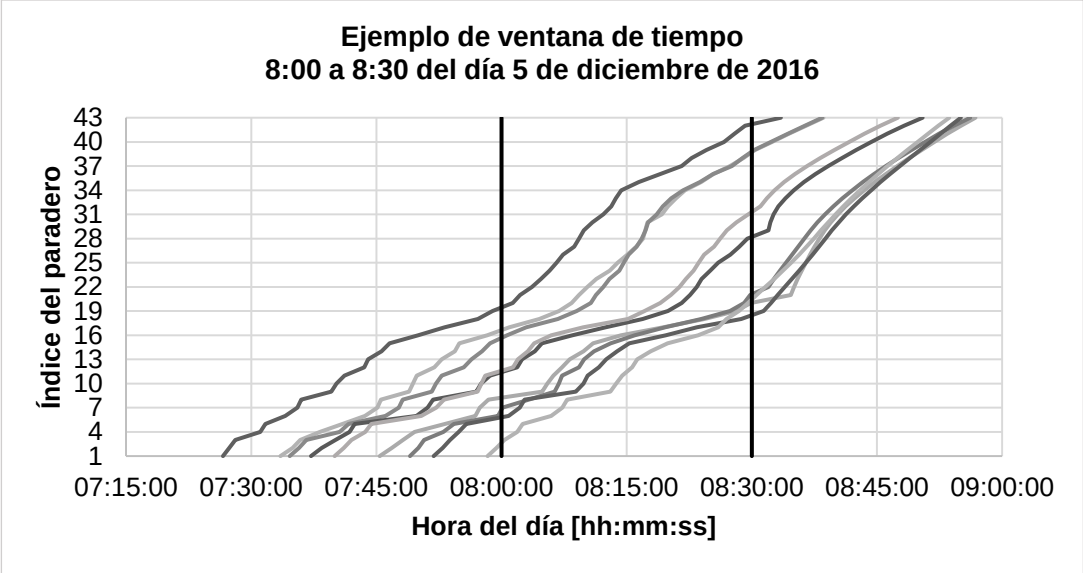


Figura 3.13. Ejemplo de tiempos de viaje capturados en una ventana de tiempo. Fuente: Elaboración propia, 2017.

3.5.2 Realización de predicciones

Para modelos que no predigan valores con relación a la llegada del bus a un cierto paradero, como RND, es necesario realizar una conversión de sus predicciones, para obtener secuencias de tiempos de viaje con las características requeridas. Esto se debe a que para generar una secuencia de *headway*, es necesario tener los tiempos de viaje de un par de buses, para cada paradero en la ruta. Las secuencias de *headway* producto de las predicciones de cada modelo deben poseer las mismas características, independiente de las diferencias que existan en los valores o métodos utilizados. Esto se debe a que posteriormente se deben comparar los resultados obtenidos, por lo que las características de las predicciones de *bunching* deben ser equiparables.

Para la RNP, se utiliza solo los tiempos de viaje del último paradero alcanzado, en una serie de tiempos de viaje parcial, para realizar predicciones de los tiempos de viaje de los paraderos siguientes en la ruta. Se debe tener en cuenta que esta secuencia de tiempos de viaje parciales de un bus está circunscrita en una ventana de tiempo. Se toma en cuenta que el error que se obtiene al hacer predicciones a paraderos lejanos en la ruta es mayor que al realizar predicciones a paraderos cercanos. Por lo tanto, se pretende minimizar la diferencia entre la posición en la ruta del paradero utilizado para ingresar los tiempos de viaje actuales a la red neuronal, y el paradero al que se quiere predecir sus tiempos de viaje. La manera en que se realiza es cambiando el valor de la variable *j,* que representa el código del paradero al que se quiere predecir los tiempos de viaje. Por otro lado, la RNP no necesita de conversiones en sus predicciones para generar secuencias de *headway*.

Para RND, las predicciones a cada *timestep* posterior a una serie de distancias parciales, se realiza utilizando progresivamente los valores predichos por el mismo modelo, en *timestep* anteriores. Esto es posible debido a que en este modelo se ingresa una secuencia de los últimos 10 *timestep* con la distancia que recorre un bus cada 30 segundos, junto con un conjunto de metadatos del viaje, y se obtiene como salida la distancia que va a recorrer el bus en los próximos 30 segundos. Luego, se utilizan los 9 últimos *timestep* provenientes de la ventana de tiempo, y se agrega el dato de la predicción anteriormente realizada, para generar la predicción del siguiente *timestep*. Este proceso de va repitiendo, hasta que se cumpla 30 minutos de predicciones o se alcance la extensión completa de la ruta.

Las predicciones realizadas por RND son convertidas en una serie de tiempos de viaje para cada paradero de la ruta. Primero, se convierten las distancias parciales que recorre un bus en cada lapso de 30 segundos, que constituye la salida de la red neuronal, en una lista de distancias acumuladas en la ruta. Segundo, se importa la distancia exacta donde se encuentra cada paradero en la ruta. Finalmente, se recorre la secuencia de distancias acumuladas, y cada vez que se alcanza o supera la distancia de cada paradero, de manera ordenada, se van interpolando

los tiempos de viaje según la cantidad de predicciones realizadas hasta ese punto. La ecuación 3.35 y 3.36 muestran el proceso de interpolación utilizado, donde TT_i es el tiempo de viaje de la i-ésima muestra de la serie de distancias acumuladas; *p* es el índice de algún paradero en la ruta, d_p es la distancia donde se encuentra el paradero p en la ruta, cuyo valor se encuentra entre los valores de la muestra d_i e d_{i+1} de la serie de distancias acumuladas; y TT_p es el tiempo de viaje interpolado para el paradero *p*.

$$TT_i = 30 * i \tag{3.35}$$

$$TT_p = \frac{TT_{i+1} - TT_i}{d_{i+1} - d_i} * (d_p - d_i) + TT_i \,,\ d_i \leq d_p \leq d_{i+1} \tag{3.36}$$

Para RNH, las predicciones a cada *timestep* posterior a una serie de valores *headway* parciales, se realiza utilizando progresivamente los valores predichos por el mismo modelo en *timestep* anteriores, de manera similar a RND. En este caso, se ingresa una secuencia de los últimos 4 *timesteps* con la diferencia de *headway* que recorre un bus cada 30 segundos, junto con un conjunto de metadatos del viaje, y se obtiene como salida la diferencia entre el último *headway* y el del próximo *timestep*. En primera instancia se ingresa a la red neuronal, los últimos 4 *timestep* de los datos capturados por la ventana de tiempo, y esta entrega los datos para el siguiente *timestep*. Luego, se utilizan los 3 últimos *timestep* provenientes de los datos de la ventana, y se agrega el dato de la predicción anteriormente realizada, para generar la predicción del siguiente *timestep*. Este proceso de va repitiendo, hasta que se alcance el valor de *headway* para el último paradero de la ruta.

Para RNH no se necesita generar secuencias de tiempos de viaje, debido a que a partir de transformaciones a la secuencia de diferencias de *headway*, producto de las predicciones del modelo sobre una ventana de tiempo, es posible construir una secuencia de *headway* para dicho par de buses. Esto proceso consiste en utilizar el valor de *headway* del primer *timestep* de la ventana de tiempo, sumarlo a la diferencia de *headway* del siguiente timestep. Para cada valor de *headway* calculado, es posible generar el valor para el *timestep* siguiente, sumando el valor de su diferencia.

El modelo de promedios históricos está construido para funcionar en el contexto de series de tiempos de viaje parciales, de manera que, al combinar estos datos con los valores promedios históricos, del *dataset* de entrenamiento, se obtengan predicciones para cada paradero de viaje. Según los metadatos de la ventana de tiempo, en términos del día de la semana y el intervalo horario, se elige una serie de tiempos de viaje promedios específica. Por otra parte, se utilizan los datos de tiempos de viaje de la ventana de tiempos, para calcular el factor de proporcionalidad *k* del modelo. Finalmente, se aplica este factor proporcionalidad a la serie de tiempos de viaje promedio para calcular los tiempos de viaje de los paraderos restantes de la ruta. No es necesario

realizar conversiones de los tiempos de viaje que arroja este modelo, para generar las secuencias de *headway*.

Los modelos de persistencia no son utilizados para realizar predicciones de *bunching*, debido a que se implementan por el único motivo de determinar la predictibilidad de las series de tiempo utilizadas en RND y RNH. Además, los modelos de persistencia describen un comportamiento estático en sus predicciones, al reutilizar el último *timestep* de una serie parcial como una predicción, lo cual no permite realizar predicciones efectivas más allá de 1 *timestep*.

3.5.3 Generación de secuencias de *headway*

Con las secuencias de tiempos de viaje de cada bus en una ventana de tiempo, se realiza una combinatoria de pares de viajes, con el fin de generar secuencias de *headway*. El proceso de generación de pares de viajes consiste en recorrer todas las combinaciones de pares de buses de una cierta ventana, y seleccionar solo los que tengan una diferencia en la hora de inicio del recorrido, no mayor a 1 hora. Como puede notar, la diferencia en la hora de inicio del viaje entre 2 buses es en realidad el *headway* inicial que poseen. Esta decisión se basa en el hecho que la oscilación máxima que presenta algún valor de *headway* durante el viaje entre un par de buses es de 40 minutos, para pares cuyo *headway* inicial sea mayor a 1 hora. Un *headway* mínimo de 20 minutos entre un par de buses no es posible considerarlo como *bunching*, al constituir aproximadamente un cuarto del tiempo máximo que demora un bus en realizar el viaje completo, y, por lo tanto, estos pares no aportan al problema estudiado. También, para efectos de eficiencia se limita la distancia entre las posiciones de los viajes, dentro de la ventana, a 3, considerando un conjunto de viajes ordenado temporalmente, al momento de hacer pares de viajes. Esto se debe a que en todos los casos se cumple la condición de que su *headway* inicial es mayor a 1 hora, por lo que no es necesario realizar búsquedas de pares de viajes a distancias mayores.

Una vez que se tienen todos los pares de viaje, de un conjunto de viajes, de una ventana de tiempo determinada, se utiliza la secuencia completa de tiempos de viaje para generar secuencias se *headway*. La secuencia de tiempos de viaje completa consiste en la concatenación de los datos de la ventana de tiempo, y los tiempos de viaje predichos por el modelo utilizado. Además, se almacena el índice del paradero donde se comienzan a realizar las predicciones, para ser utilizado posteriormente. Dada una secuencia de tiempos de viaje TT_i para el viaje i y una secuencia de tiempos de viaje TT_j para el viaje j sucesor de i, el valor de *headway* entre ambos viajes en cada paradero $k = \{1,2,\dots,43\}, k \in \mathbb{N}$, está dado por la ecuación $h_{ijk} = TT_{ik} - TT_{jk}$. Como es posible ver en la Figura 3.14, la diferencia en segundos entre los tiempos de viaje de ambos buses, en cada paradero, constituye la secuencia de *headway*. Luego, en la Figura 3.15 se presenta un ejemplo de secuencia de headway, para cada paradero de la ruta entre un par de buses.

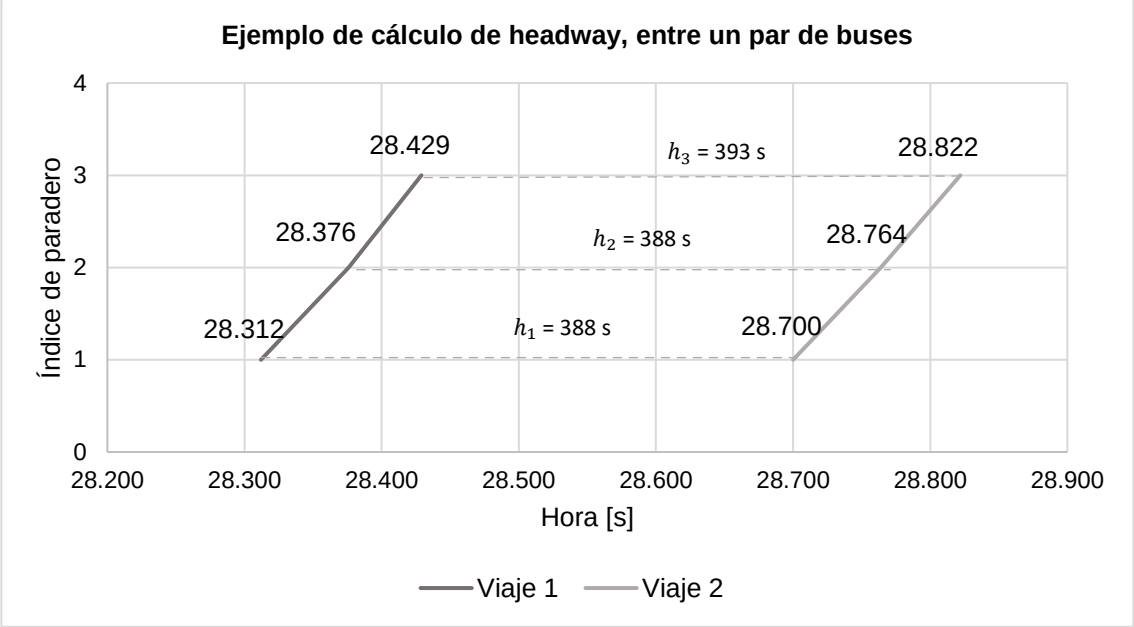


Figura 3.14. Ejemplo gráfico del cálculo de los valores de headway para un par de secuencias de horas de llegada a cada paradero de la ruta.
Fuente: Elaboración propia, 2017.

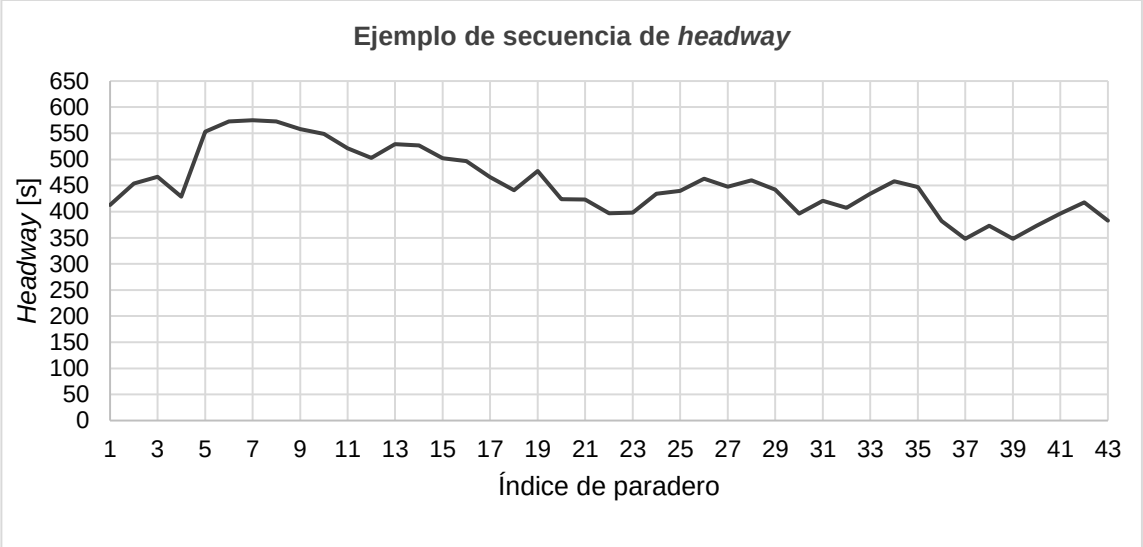


Figura 3.15. Ejemplo de secuencia de headway a través de los distintos paraderos de la ruta.
Fuente: Elaboración propia, 2017.

3.5.4 Algoritmo de detección de bunching

Una vez que se tienen las secuencias de *headway* predichas por cada modelo y las secuencias reales, para cada par de viajes de cada ventana de tiempo, se aplica un algoritmo de detección de *bunching*. El criterio de detección es representado por la ecuación 3.37, que determina de

manera binaria la ocurrencia de *bunching* entre un par de buses, en un cierto paradero de la ruta. Dado un par de viajes i y j, que son recorridos por 2 buses distintos en la misma ventana de tiempo, se detecta *bunching*, retornando el valor 1, si el valor absoluto del *headway* en algún paradero k es menor al 25% del *headway* inicial, y en caso contrario no se detecta *bunching*, retornando el valor 0. Esta definición está basada en el trabajo de Moreira-Matias et al. (2012) (véase sección 2.1.5). Una vez aplicado este algoritmo para cada secuencia de *headway* de cada par de buses de cada ventana de tiempo, se compara la lista binaria de cada modelo, con la que proviene de secuencia de *headway* original, de manera de determinar la precisión de las predicciones.

$$BB_{ijk} = \begin{cases} 1 & si\ |h_{ijk}| < \frac{h_{ijk}}{4} \\ 0 & en\ otro\ caso \end{cases} \tag{3.37}$$

El criterio de detección representa un rango de valores de *headway*, que es posible apreciar de manera más natural con un ejemplo gráfico. En la Figura 3.16, es posible apreciar la forma en que actúa el algoritmo de detección, donde la zona gris representa el rango donde los valores de *headway* entre los buses cumplen el criterio para considerarse en *bunching*. Por ejemplo, entre los paraderos 17 y 36 los buses se encuentran en *bunching*, pero entre los paraderos 37 y 43, dejan de estarlo. Cabe destacar, que un valor de *headway* negativo significa que el primer bus en iniciar su viaje es sobrepasado en algún punto de la ruta por el bus que lo sucede, intercambiando sus posiciones.

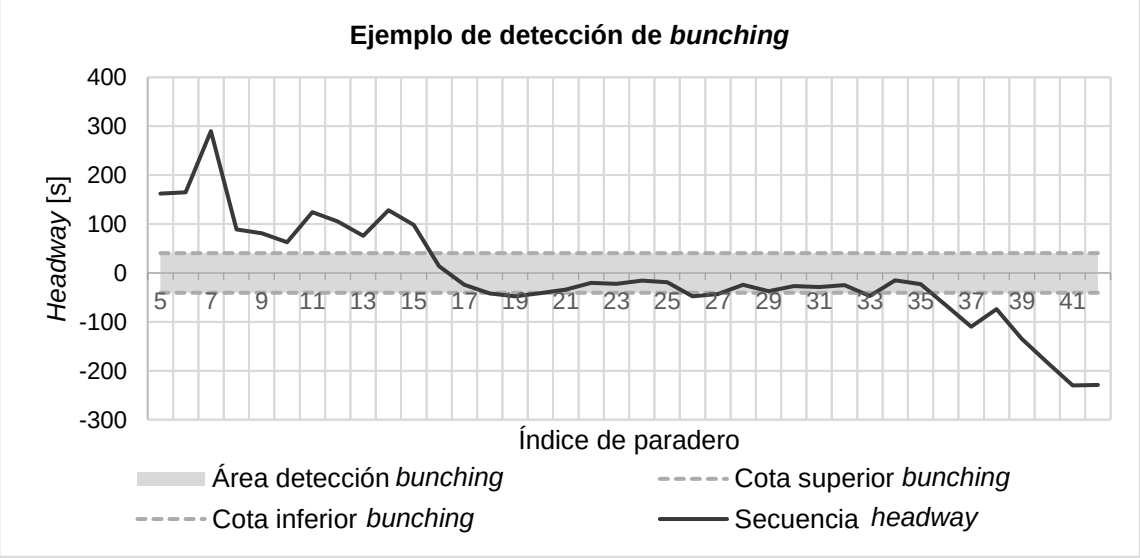


Figura 3.16. Ejemplo gráfico del algoritmo de detección de bunching en una secuencia de headway.

Fuente: Elaboración propia, 2017.

3.5.5 Método de evaluación de predicciones de *bunching*

Se utilizan 4 métricas definidas por las ecuaciones 3.38 a 3.41, para determinar la precisión de sus predicciones de cada modelo. La primera ecuación define la métrica $Accuracy_1$ (ACC_1) que mide la razón entre la cantidad de pares de viajes donde se predice correctamente la ocurrencia de *bunching* en la siguiente media hora (α_i), independiente de la secuencia de paraderos donde ocurra éste, y la cantidad total de pares de viaje evaluados (N_i), para cada ventana de tiempo (i). La segunda métrica es $Accuracy_2(ACC_2)$, que mide la razón entre la cantidad de paraderos donde se predice correctamente la ocurrencia de *bunching* (β_{ij}), y la cantidad de paraderos predichos (P_{ij}), para cada par de viajes (j) de cada ventana de tiempo (i). La tercera es $Sensitivity$ (SES), que mide la razón entre la cantidad de paraderos en donde se predice correctamente la ocurrencia de *bunching* (γ_{ij}), y la cantidad total de paraderos donde ocurre *bunching* (B_{ij}), para cada par de viajes de cada ventana de tiempo. La cuarta es $Specificity$ (SPC), que mide la razón entre la cantidad de paraderos en donde se predice correctamente la no ocurrencia de *bunching* (λ_{ij}), y la cantidad total de paraderos donde no ocurre *bunching* ($P_{ij} - B_{ij}$), para cada par de viajes (j) de cada ventana de tiempo (i). Las últimas 3 métricas mencionadas, se basan en las utilizadas en el trabajo realizado por Yu, Chen, Wu, Ma, & Wang (2016).

$$Accuracy_1(ACC_1) = \frac{100}{V} * \sum_{i=1}^{V} \frac{\alpha_i}{N_i} \tag{3.38}$$

$$Accuracy_2(ACC_2) = \frac{100}{V} * \sum_{i=1}^{V} \frac{\sum_{j=1}^{N} \frac{\beta_{ij}}{P_{ij}}}{N_i} \tag{3.39}$$

$$Sensitivity\ (SES) = \frac{100}{V} * \sum_{i=1}^{V} \frac{\sum_{j=1}^{N} \frac{\gamma_{ij}}{B_{ij}}}{N_i} \tag{3.40}$$

$$Specificity\ (SPC) = \frac{100}{V} * \sum_{i=1}^{V} \frac{\sum_{j=1}^{N} \frac{\lambda_{ij}}{(P_{ij} - B_{ij})}}{N_i} \tag{3.41}$$

CAPÍTULO 4. RESULTADOS Y ANÁLISIS

4.1 RESULTADOS OBTENIDOS CON EL MODELO RNP

Al realizar un análisis del *dataset* utilizado para RNP, se muestra que mientras mayores sean los "saltos" de paraderos (δ), mayores son los tiempos de viaje, debido a que aumenta la distancia que debe recorrer el bus. La variable δ consiste en la diferencia entre el índice del paradero a consultar *j*, y el índice del último paradero visitado *i*, según se presenta en la ecuación 4.1. La Tabla 4.1 muestra que los tiempos de viaje TT_{ij} que recorre un bus, según los datos del *dataset*, están acotados a un rango específico, para distintos valores de δ. Por ejemplo, para δ igual a 1, los tiempos de viaje están acotados entre 6 y 754 segundos, comprendiendo un rango de 748 segundos, mientras que para δ igual a 2, el rango aumenta a 905 segundos. Esto genera que aumenten el MAE en las predicciones, debido a que la fluctuación de los valores de *dataset* es mayor al ir aumentando el rango de valores posibles.

$$\delta = j - i \tag{4.1}$$

En la Figura 4.1 se observa un ejemplo de las predicciones de tiempos de viaje para cada paradero realizadas por RNP, comparadas con los datos reales. Las predicciones se realizan a partir de paradero 17, donde se utiliza los tiempos de viaje de tal paradero para consultar los tiempos de viaje de los paraderos siguientes en la ruta. Se observa que las predicciones describen un comportamiento lineal a partir del último tiempo de viaje ingresado, no pudiendo predecir correctamente los cambios de velocidad que describe el bus en los datos reales. Esto último se aprecia principalmente entre el paradero 17 a 19 donde el bus se ralentiza aumentando la pendiente de la curva, para luego estabilizarse en una pendiente relativamente constante entre el paradero 20 hasta el final de la ruta. Es por este motivo que las predicciones están desfasadas casi paralelamente a los datos reales durante toda la extensión de la ruta.

Se observa que el valor del MAPE va disminuyendo a medida que aumenta el valor de δ, a diferencia del resto de métricas utilizadas. MAPE mide el error absoluto porcentual entre los valores de tiempos de viaje reales y los valores predichos por RNP, por lo que para valores de δ pequeños, el error absoluto considera un porcentaje importante de la escala de valores a predecir, mientras que para valores de δ más grandes, el error es porcentualmente mejor al considerar el rango de valores en que los deben ser predichos los tiempos de viaje. Por ejemplo, para δ igual a 1, $\overline{TT}$ es de 72 segundos y el MAE obtenido es de 37,60 segundos, constituyendo un 52,22% de $\overline{TT}$, mientras que para δ igual a 10, $\overline{TT}$ es de 686 segundos y el MAE obtenido es de 98,63 segundos, constituyendo solo un 14,37% del $\overline{TT}$. Este fenómeno en términos prácticos significa que, si bien la precisión de las predicciones de tiempos de viaje a paraderos cercanos es mejor

que a paraderos lejanos, el efecto de dichos errores es más notorio en paraderos cercanos que en paraderos lejanos, debido que los valores a predecir en paraderos cercanos son más pequeños comparativamente. Por otro lado, MSE, RMSE y MAE aumentan a medida que crece el valor de δ, debido a que para valores mayores de δ aumenta la dispersión de los resultados a entregar, dados los múltiples sucesos que pueden ocurrir en un bus en un rango mayor de paraderos del recorrido.

Luego de realizar un proceso de ajuste de los hiper-parámetros para RNP, se consiguen los valores que permiten minimizar el error en las predicciones, los cuales se presentan en la Tabla 4.2. Posteriormente será desglosado el proceso de prueba realizado para cada hiper-parámetro. La Tabla 4.3, muestra los resultados generales obtenidos con estos hiper-parámetros, donde para el *dataset* de prueba, se consigue un MAPE de 19,07% y un MAE de 95,38 segundos, para cada predicción de tiempos de viaje realizadas, independiente del valor de δ. En términos prácticos, esto significa que si se quiere conocer el tiempo de viaje que demora un bus entre cualquier par de paraderos próximos en la ruta, el error promedio va ser de un 19,07% del valor real, y en términos relativos el MAE representa un 4,07% del tiempo de viaje promedio en completar el viaje para el servicio I09. Los valores de error en cada métrica varían considerablemente dependiendo del valor de δ, por lo que en la Tabla 4.4 se pueden observar los resultados desglosados para los distintos valores de δ. El *dataset* utilizado, es una versión del *dataset* de prueba que solo posee predicciones de tiempos de viajes de hasta 30 minutos y valores de δ de hasta 29 paraderos.

Tabla 4.1. Tiempos de viaje mínimos, máximos, promedio y desviación estándar para distintos saltos de paraderos.

$\boldsymbol{\delta}$	**Tiempos de viaje [s]**			
	$\boldsymbol{TT_{min}}$	$\boldsymbol{TT_{max}}$	$\boldsymbol{\overline{TT}}$	$\boldsymbol{\sigma(TT)}$
1	6	743	72	54
2	13	918	142	82
3	20	1.299	212	105
4	27	1.497	280	125
5	34	1.575	349	142
6	41	1.846	417	156
7	48	2.080	485	170
8	55	2.108	553	183
9	107	2.201	619	194
10	150	2.269	686	204
11-19	172	3.059	1.008	297
20-29	627	3.834	1.634	343
30-43	1.043	4.800	2.351	433

Fuente: Elaboración propia, 2017.

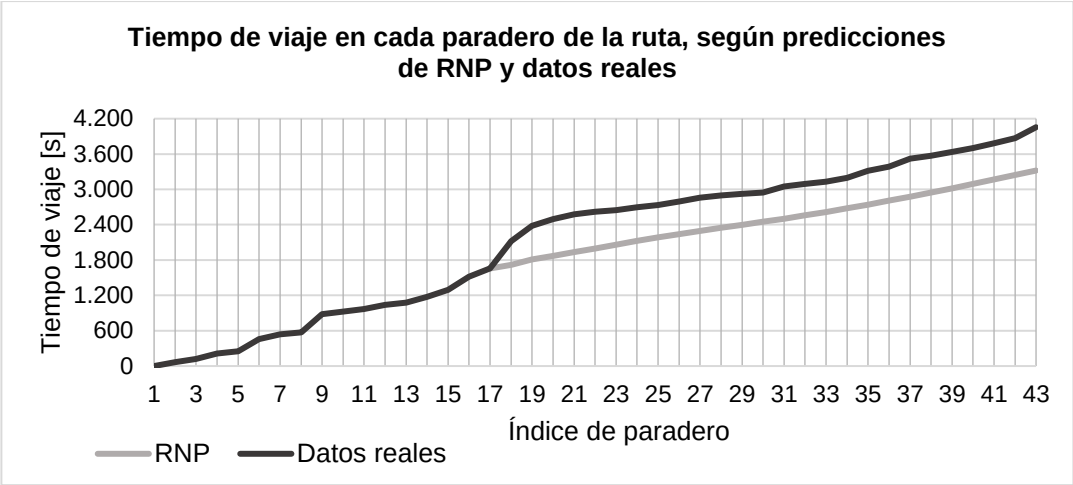


Figura 4.1. Ejemplo de hora de llegada de un bus a cada paradero de la ruta, según predicciones de RNP y datos reales.

Fuente: Elaboración propia, 2017.

Tabla 4.2. Valores de hiper-parámetros finales de RNP.

Nombre del hiper-parámetro	Valor
Épocas de entrenamiento	50
Tasa de aprendizaje	10^{-6}
Método de aprendizaje	ADAM
Cantidad de capas ocultas	2
Cantidad de neuronas por capa oculta	100
Tipo de normalización de variables	$tanh$

Fuente: Elaboración propia, 2017.

Tabla 4.3. Resultados generales para RNP.

Dataset	**MSE [s^2]**	**RMSE [s]**	**MAPE [%]**	**MAE [s]**
Entrenamiento	19.117,52	138,27	19,03	99,92
Prueba	16.688,89	129,19	19,07	95,38

Fuente: Elaboración propia, 2017.

Tabla 4.4. Resultados de RNP para distintos valores de δ.

δ	MSE [s^2]	RMSE [s]	MAPE [%]	MAE [s]
1	2.598,22	50,97	68,32	37,60
2	3.568,38	59,74	37,14	43,64
3	5.072,78	71,22	28,05	51,42
4	6.723,57	82,00	24,02	59,62
5	8.161,05	90,34	21,39	66,59
6	9.671,72	98,34	19,54	72,97
7	11.348,43	106,53	18,03	79,48
8	12.961,23	113,85	16,76	85,27
9	15.054,92	122,70	16,13	92,36
10	16.948,32	130,19	15,50	98,63
11 – 20	25.404,94	159,39	13,10	122,45
21 – 29	60.730,33	246,44	10,95	185,46

Fuente: Elaboración propia, 2017.

En la Tabla 4.5 se observan los valores de los hiper-parámetros restantes utilizados para la realización de pruebas de cada hiper-parámetro, para RNP. En cada fila se encuentran todos los hiper-parámetros cuyos valores son fijados para realizar pruebas sobre uno en específico. En cada columna se encuentra el nombre del hiper-parámetro que es probado, al evaluar los resultados obtenidos al utilizar distintos valores. Por ejemplo, para encontrar la tasa de aprendizaje más idónea para RNP, se entrenan múltiples instancias, cada una con distintos valores de tasa de aprendizaje, pero con los mismos hiper-parámetros restantes y utilizando los mismos datos de entrenamiento. En este caso, se realizan 50 épocas de entrenamiento, con el método de aprendizaje ADAM, con una arquitectura MLP con 2 capas ocultas, cada una con 100 neuronas, normalizando los datos de entrada con la función tangente hiperbólica.

Utilizar una tasa de entrenamiento de 10^{-6}, permite realizar una convergencia más estable y controlada. Estos resultados se obtienen probando el modelo resultante luego de las primeras 10 épocas de entrenamiento, para tasa de aprendizaje de $10^{-4}, 10^{-5}$ y 10^{-6}, realizando predicciones con el *dataset* de prueba y comparando los resultados con los valores reales. Según los resultados de la Figura 4.2, la diferencia promedio de MSE a través de las épocas, para la red con una tasa de aprendizaje de 10^{-6} es de – 874,33, con una tasa de aprendizaje de 10^{-5} es de 1468,09 y con una tasa de aprendizaje de 10^{-4} es de 320,79. También, la desviación estándar de dicha diferencia, para una tasa de 10^{-6} es de 6.030,22, para una tasa de 10^{-5} es de 9.481,77 y para una tasa de 10^{-4} es de 12.134,46. Estos resultados dan cuenta que la tasa de aprendizaje de 10^{-6} es la única que muestra un descenso constante del MSE, además que su oscilación es 36,4% menor que para una tasa de 10^{-5} y 50,3% menor que para una tasa de 10^{-4}. Para obtener

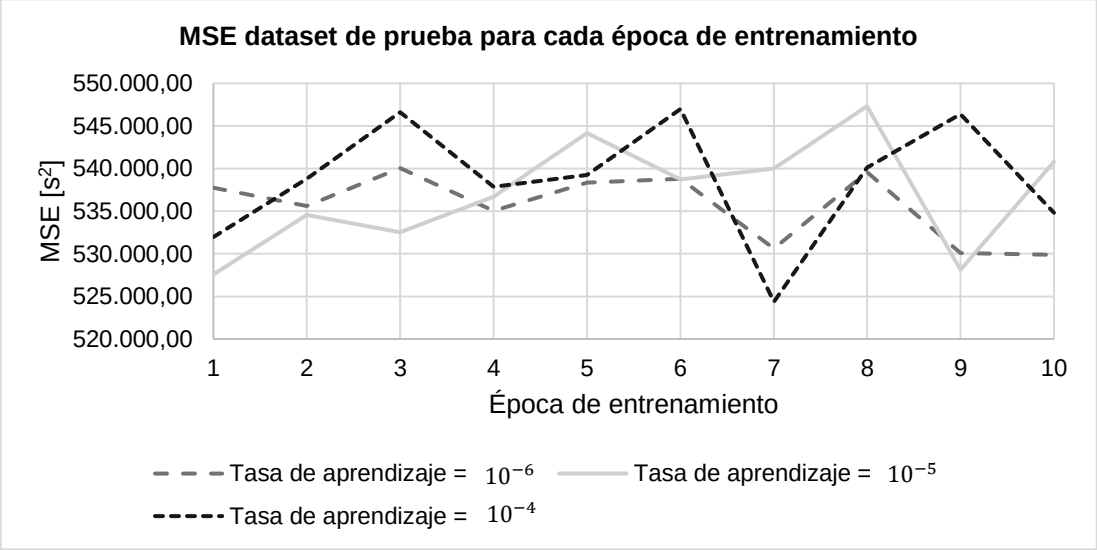
estos resultados, cada red neuronal se entrenó utilizando los hiper-parámetros de la Tabla 4.5, en la columna de tasa de aprendizaje.

Tabla 4.5. Valores de hiper-parámetros restantes utilizados para la realización de pruebas de cada hiper-parámetro, para RNP.

Hiper-parámetros fijos utilizados	**Hiper-parámetro probado**					
	Tasa de aprendizaje	**Método de aprendizaje**	**Cantidad de capas ocultas**	**Cantidad de neuronas por capa**	**Normalización de variables**	**Versiones del *dataset***
Épocas de entrenamiento	50	50	50	50	50	50
Tasa de aprendizaje	-	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-6}
Método de aprendizaje	ADAM	-	ADAM	ADAM	ADAM	ADAM
Cantidad de capas ocultas	2	2	-	1	2	2
Cantidad de neuronas por capa oculta	100	100	-	-	100	100
Normalización de variables del modelo	Tangente hiperbólica	Proporcional	Proporcional	Proporcional	-	Tangente hiperbólica

Fuente: Elaboración propia, 2017.

Figura 4.2. MSE para las primeras 10 épocas de entrenamiento, para distintas tasas de aprendizaje.



Fuente: Elaboración propia, 2017.

El método de aprendizaje ADAM produce mejores resultados que SGD, al ser entrenado bajo las mismas condiciones. En la Tabla 4.6, se puede apreciar que el valor de RMSE al utilizar ADAM, es un 15,7% menor que los resultados obtenidos utilizando SGD, para el *dataset* de entrenamiento y un 19,2%, para el *dataset* de prueba. Esto se puede deber a que ADAM, al utilizar factores de olvido con respecto al promedio del gradiente y el cuadrado del gradiente, permite que la actualización de los pesos de la red sea más consistente a través de las épocas y evite concurrir en ciertos mínimos locales que el método SGD común no logra superar. Para obtener estos resultados, cada red neuronal se entrenó utilizando los hiper-parámetros de la Tabla 4.5, en la columna de método de aprendizaje.

Tabla 4.6. Resultados del modelo RNP entrenada con el método de optimización Adam y SGD.

Dataset evaluado	**ADAM**		**SGD**	
	MSE [s^2]	**RMSE [s]**	**MSE [s^2]**	**RMSE [s]**
Entrenamiento	37.687,15	194,13	53.073,70	230,38
Prueba	29.635,98	172,15	45.396,49	213,06

Fuente: Elaboración propia, 2017.

Se consiguen mejores resultados con 100 neuronas en la capa oculta, para RNP. En la Tabla 4.7, es posible apreciar que el RSME al realizar predicciones con el dataset de prueba con 100 neuronas es 1,69% menor que al utilizar 50 neuronas, y un 11,65% menor que al utilizar 1.000 neuronas. Además, a menor cantidad de neuronas, menor es el tiempo de ejecución del algoritmo de entrenamiento, donde el tiempo de ejecución al utilizar 50 neuronas es 15,56% menor que al utilizar 100 neuronas, que, a su vez, es un 41,56% menor que al utilizar 1.000 neuronas. Esto se puede deber, a que las matrices de pesos de las capas de la red poseen menores dimensiones, y por ende se deben realizar menos cálculos tanto para realizar las predicciones como para el proceso de actualización de los pesos de la red. Para obtener estos resultados, cada red neuronal se entrenó utilizando los hiper-parámetros de la Tabla 4.5, en la columna de cantidad de neuronas por capa.

Tabla 4.7. Resultados del modelo RNP para distintas cantidades de neuronas en la capa oculta.

Cantidad de neuronas	**RSME *dataset* de entrenamiento [s]**	**RSME *dataset* de prueba [s]**	**Tiempo de ejecución [h]**
50	230,24	216,72	38,00
100	230,38	213,06	45,00
1.000	260,70	241,16	77,00

Fuente: Elaboración propia, 2017.

Los resultados al utilizar 2 capas ocultas son mejores que al utilizar 1 capa oculta, incluso si esta última posee un mayor número de neuronas. En la Tabla 4.8, se aprecia que utilizar 2 capas ocultas con 100 neuronas por capa, se consigue un RMSE un 11,63% menor al usar 1 capa con 100 neuronas, y 20,72% menor al usar 1 capa con 1.000 neuronas. El aumento en el tiempo de ejecución al agregar una segunda capa oculta es menor que al agregar más neuronas a la misma capa, siendo este aumento de 6 y 22 horas respectivamente. Esto ocurre a pesar de que la cantidad de pesos internos de RNP con 2 capas ocultas de 100 neuronas es de 10.500, mientras que, para la instancia con 1 capa oculta de 1.000 neuronas, es de solo 5.000. Posiblemente, esto ocurre debido a que las operaciones matriciales de dimensiones menores se realizan de manera más eficiente que las operaciones con matrices de grandes dimensiones, en la plataforma Theano. Para obtener estos resultados, cada red neuronal se entrenó utilizando los hiper-parámetros de la Tabla 4.5, en la columna de cantidad de capas ocultas.

Tabla 4.8. Resultados del modelo RNP con distintas cantidades de neuronas y capas ocultas.

Cantidad de capas ocultas	**Cantidad de neuronas en la(s) capa(s) oculta(s)**	**RMSE *dataset* de entrenamiento [s]**	**RMSE *dataset* de prueba [s]**	**Tiempo de ejecución [h]**
1	100	230,38	213,06	45,00
1	1.000	260,70	241,16	77,00
2	100	206,68	207,75	51,00

Fuente: Elaboración propia, 2017.

Los resultados de RNP entrenada con datos de entrada normalizados con la función de tangente hiperbólica, consigue mejores resultados que utilizar una normalización lineal entre -1 y 1. En la Tabla 4.9, se puede apreciar que el valor del RMSE para la normalización por tangente hiperbólica, para el *dataset* de entrenamiento, disminuye en un 6,1%, y en un 17,3% para el *dataset* de prueba, con respecto a la normalización lineal. Cabe destacar que, para realizar la normalización mediante tangente hiperbólica, primero se escalan los datos de entrada linealmente entre -1 y 1, por lo que la diferencia entre ambas normalizaciones consiste en la aplicación de esa función. Esto muestra que la concordancia de la distribución de los valores de entrada de una red neuronal, con respecto la función de activación utilizada en las capas ocultas, permite favorecer el proceso de aprendizaje de la red. Para obtener estos resultados, cada red neuronal se entrenó utilizando los hiper-parámetros de la Tabla 4.5, en la columna de normalización de variables.

Tabla 4.9. Comparación de resultados para distintas normalizaciones.

Dataset evaluado	**Normalización lineal**		**Normalización tangente hiperbólica**	
	MSE [s^2]	**RMSE [s]**	**MSE [s^2]**	**RMSE [s]**
Entrenamiento	56.776,27	206,68	37.687,15	194,13
Prueba	52.510,35	207,75	29.635,98	172,15

Fuente: Elaboración propia, 2017.

Entrenar RNP para que realice predicciones de tiempos de viaje no mayores a 30 minutos, permite mejorar la precisión en ciertos casos, con respecto a entrenarla para que realice predicciones para el rango completo de valores de los tiempos de viaje presente en el *dataset*. En la Figura 4.3, es posible observar que, para saltos menores a 4 paraderos, RNP con el entrenamiento especializado logra un MAE 14,1% menor que la red entrenada con el dataset completo, para saltos entre 4 a 10 paraderos errores un 0,8% mayor, para saltos entre 11 y 19 paraderos un 5,1% menor, y para saltos entre 20 y 29 paraderos un 4,3% mayor. No se muestran los resultados para saltos mayores a 29 paraderos, debido a que en todos los casos los tiempos de viaje a recorrer (y) son mayores a 30 minutos. Para obtener estos resultados, cada red neuronal se entrenó utilizando los hiper-parámetros de la Tabla 4.5, en la columna de versiones del *dataset*.

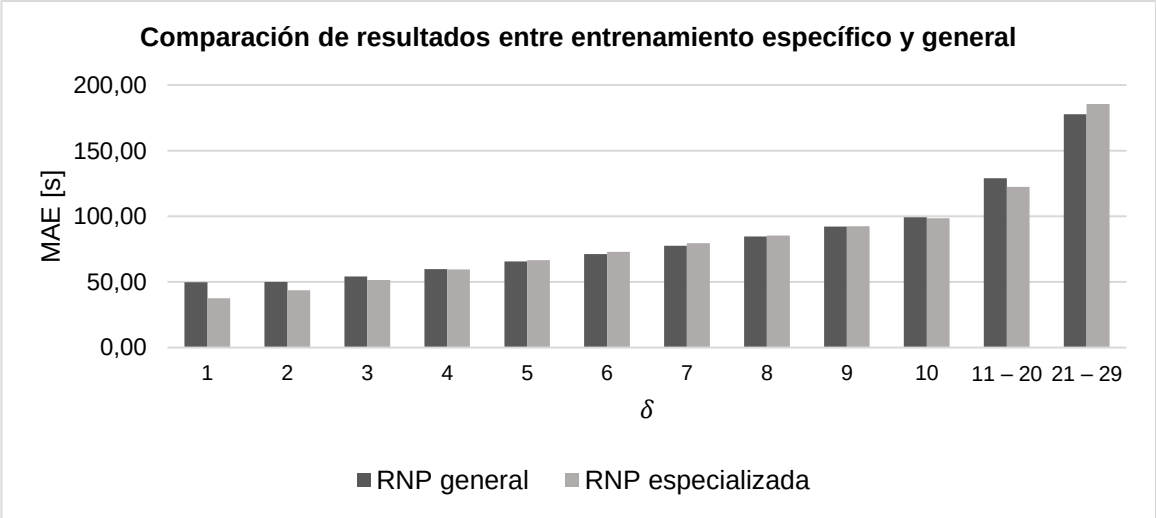


Figura 4.3. Gráfico comparativo del MAE entre RNP con entrenamiento especializado para predicciones de hasta 30 minutos, y con RNP general.
Fuente: Elaboración propia, 2017.

La precisión en las predicciones entre RNP y el modelo presentado por Gurmu & Fan (2014) es similar. Cabe destacar que ante la imposibilidad de conseguir el *dataset* utilizado por Gurmu & Fan (2014), los resultados presentados son los conseguidos al entrenar con datos propios. Los

resultados generales muestran que el MAPE de RNP es de 19,1% y de 18,3% para el modelo de la literatura. Es posible apreciar en la Figura 4.4, que para las predicciones en un rango de tiempo de hasta 50 minutos, el MAPE generado por RNP es menor al modelo de la literatura en promedio un 2%. Luego, para las predicciones realizadas en un rango de tiempo de 50 a 60 minutos, el MAPE de RNP es mayor en un 34,5%. Esto puede deberse a que la red fue entrenada para predecir en tiempos de viaje de hasta media hora, por lo que es lógico pensar que la precisión no sea favorable para tiempos mayores.

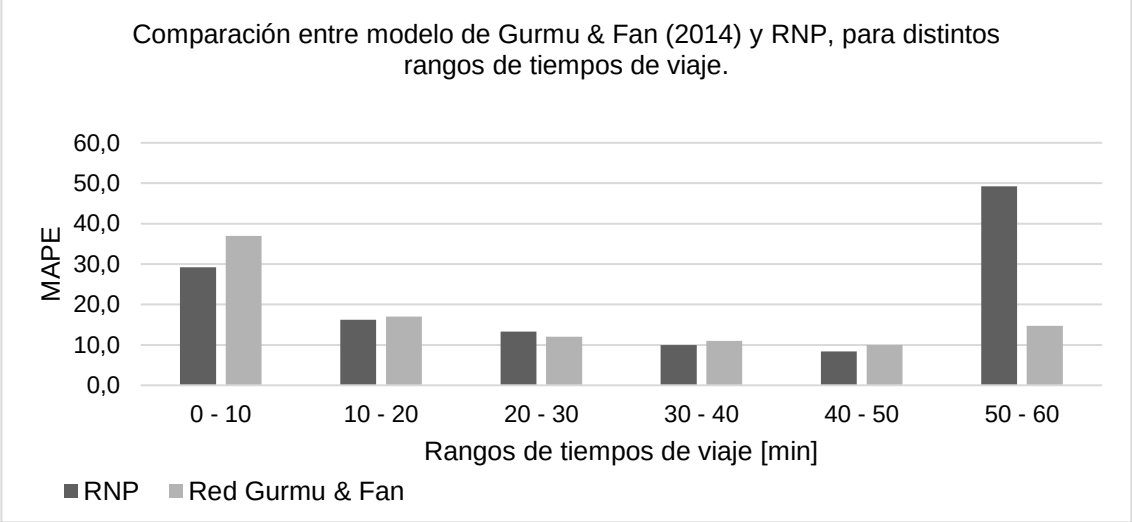


Figura 4.4. Gráfico comparativo entre el MAPE de RNP y el modelo de Gurmu & Fan (2014), para distintos rangos de tiempos de viaje.

Fuente: Elaboración propia, 2017.

4.2 RESULTADOS OBTENIDOS CON EL MODELO RND

Luego de la realizar un proceso de ajuste de hiper-parámetros para RND, se consiguen los valores que permiten minimizar el error en las predicciones, los cuales se presentan en la Tabla 4.10. Posteriormente, se desglosan las pruebas realizadas para cada hiper-parámetro. En la Tabla 4.11, se muestran los resultados de RND entrenada con dichos hiper-parámetros, donde se consigue en las predicciones sobre el *dataset* de prueba, un MSE de 0,0092 km^2, un MAE de 0,0724 km y un RMSE de 0,0952 km. En términos prácticos, esto significa que, dado un bus en tránsito, del cual se conoce la posición en los últimos minutos y el contexto del viaje realizado, se puede predecir la posición del bus en los próximos 30 segundos con un error promedio de 72,4 metros. Considerando que la velocidad mínima registrada por un bus es de 12 km/h y la velocidad máxima es de 55,22 km/h, y que por lo tanto en 30 segundos se recorre entre 100 metros y 460,2 metros, respectivamente, entonces el error porcentual de las predicciones está entre el 15,73% y el 72,4%.

Tabla 4.10. Valores de hiper-parámetros utilizados para obtener los resultados finales de RND.

Nombre del hiper-parámetro	Valor
Épocas de entrenamiento	35
Tasa de aprendizaje	10^{-4}
Método de aprendizaje	ADAM
Cantidad de neuronas por capa oculta	50

Fuente: Elaboración propia, 2017.

Tabla 4.11. Resumen de resultados finales de la RND.

Dataset evaluado	MSE [$\mathbf{km^2}$]	RMSE [km]	MAE [km]
Entrenamiento	0,0144	0,1189	0,0940
Prueba	0,0092	0,0952	0,0724

Fuente: Elaboración propia, 2017.

Luego de evaluar los distintos modelos de predicción para para RND, se determina el que consigue minimizar el error en las predicciones, el cual corresponde a RND_{222}. Este modelo contiene la combinación de variables de entrada de metadatos md_2, la combinación de variables de entrada de serie de tiempo sd_2, y la variable de salida y_2, en concordancia con la nomenclatura de modelos de predicción de la ecuación 3.13. Específicamente, se utiliza como entrada de serie de tiempo, una secuencia de 10 *timestep* consecutivos de largo, con valores de la distancia que recorre un bus en lapsos de 30 segundos. Se utiliza como entrada de metadatos, *dw*, *t* y *i*, los cuales coinciden con 3 de las variables de entrada de RNP. Luego, se entrega como salida, una predicción de la distancia que va a recorrer el bus en los siguiente 30 segundos, con respecto al último muestreo de la serie de tiempo ingresada. Posteriormente se desglosan los resultados de las pruebas realizadas para determinar cada combinación de variables.

En la Tabla 4.12 se observan los valores de los hiper-parámetros restantes utilizados para la realización de pruebas de cada hiper-parámetro, para RND. En la primera columna de cada fila se encuentra el nombre de los hiper-parámetros cuyos valores son fijados para realizar pruebas sobre un hiper-parámetro en específico. En cada columna se encuentra el nombre del hiper-parámetro que es probado, al evaluar los resultados obtenidos al utilizar distintos valores. Por ejemplo, para encontrar el método de aprendizaje más idóneo para RND, se entrenan 2 instancias distintas, cada una utilizando métodos de aprendizaje distintos, pero con los mismos valores del resto de hiper-parámetros y utilizando los mismos datos de entrenamiento. En este caso, se realizan 10 épocas de entrenamiento, con una tasa de aprendizaje de 10^{-6}, con 2 capas ocultas LSTM, cada una con 100 neuronas, para el modelo de predicción RND_{222}.

Tabla 4.12. Valores de hiper-parámetros fijos utilizados para la realización pruebas de cada hiper-parámetro y modelo de predicción, para RND.

Hiper-parámetros fijos utilizados	**Hiper-parámetro a probar**					
	Tasa de aprendizaje	**Método de aprendizaje**	**Cantidad de capas ocultas y neuronas por capa**	**Variables de serie de tiempo y metadatos**	**Variables de salida**	**Arquitectura metadatos**
Épocas de entrenamiento	10	10	10	50	10	50
Tasa de aprendizaje	-	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-6}
Método de aprendizaje	ADAM	-	ADAM	SGD	SGD	SGD
Cantidad de capas LSTM	2	2	-	2	2	2
Cantidad de neuronas por capa oculta	100	100	-	100	100	100
Modelo de predicción	RND_{222}	RND_{222}	RND_{222}	-	-	RND_{222}

Fuente: Elaboración propia, 2017.

Entrenar RND con una tasa de aprendizaje de 10^{-4} permite ralentizar los efectos del sobre-entrenamiento durante las primeras épocas de entrenamiento, al mismo tiempo que permite llegar a mejores precisiones, con respecto a tasas de 10^{-5} y 10^{-3}. Los efectos del sobre-entrenamiento se aprecian cuando comienza a aumentar el error en las predicciones sobre el *dataset* de prueba, a medida que se entrena una mayor cantidad de épocas. En la Figura 4.5, es posible ver que con tasa de aprendizaje de 10^{-5} y 10^{-4} no se aprecia este fenómeno, debido a que el RMSE va disminuyendo de manera relativamente consistente a través de las épocas. Sin embargo, con una tasa de aprendizaje de 10^{-3}, se puede apreciar que el mejor rendimiento de la red se obtiene luego de la segunda época de entrenamiento, y luego el error comienza a aumentar, evidenciando el sobre-entrenamiento. También, el menor RMSE presente en la figura es 0,0989, y es logrado en la décima época, con una tasa de aprendizaje de 10^{-4}. Para obtener estos resultados, cada instancia de RND se entrena utilizando los hiper-parámetros de la Tabla 4.12, en la columna de tasa de aprendizaje.

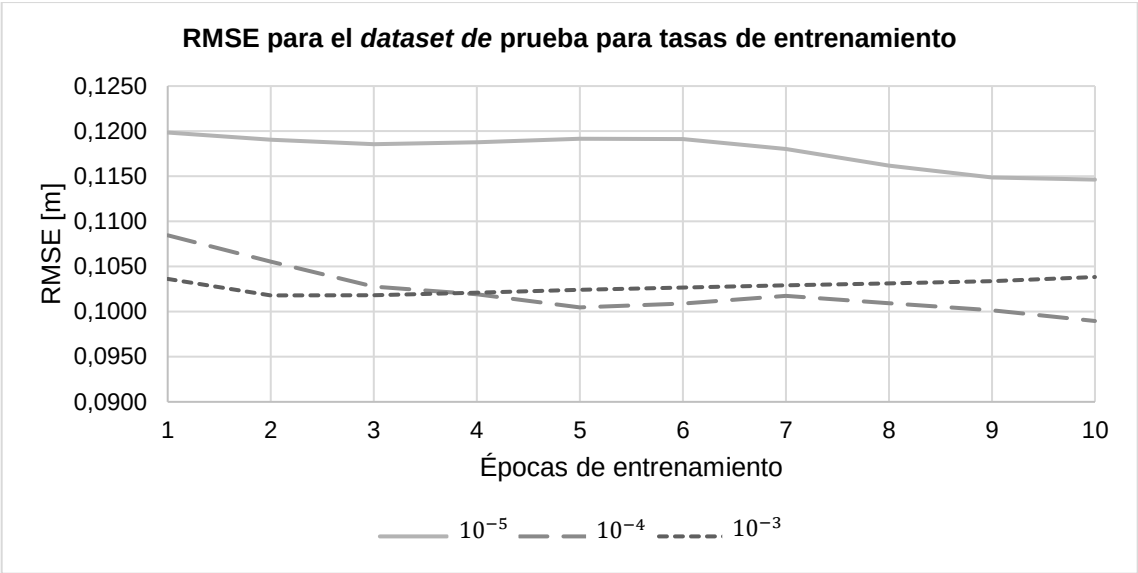


Figura 4.5. RMSE a través de las épocas para distintas tasas de aprendizaje, para RND.
Fuente: Elaboración propia, 2017.

Entrenar RND utilizando el método ADAM, entrega mejores precisiones que utilizando SGD. Según los resultados de la Tabla 4.13, el RMSE conseguido al entrenar con el método ADAM, al realizar predicciones sobre el *dataset* de entrenamiento, es un 14,96% menor que utilizando SGD, y al realizar predicciones sobre el *dataset* de prueba, este es un 10,14% menor. Esto se puede deber a que ADAM realiza un descenso de gradiente estocástico más controlado, tomando en cuenta los valores de actualizaciones de pesos anteriores y al utilizar factores de olvido para el cálculo del gradiente y del gradiente cuadrado de la función objetivo. Para obtener estos resultados, cada instancia de RND se entrena utilizando los hiper-parámetros de la Tabla 4.12, en la columna de método de aprendizaje.

Tabla 4.13. Resultados de RND entrenada con distintos métodos de aprendizaje.

Método de aprendizaje	***Dataset* de Entrenamiento**			***Dataset* de Prueba**		
	MSE [$\mathbf{km^2}$]	**RMSE [km]**	**MAE [km]**	**MSE [$\mathbf{km^2}$]**	**RMSE [km]**	**MAE [km]**
SGD	0,0210	0,1435	0,1147	0,0132	0,1137	0,0887
Adam	0,0151	0,1220	0,0978	0,0105	0,1018	0,0797

Fuente: Elaboración propia, 2017.

Utilizar 2 capas LSTM con 100 neuronas cada una, obtiene el menor RMSE y un tiempo de ejecución moderado para realizar pruebas de los distintos hiper-parámetros. En la Tabla 4.14, se observa que el RMSE de RND con 2 capas LSTM con 100 neuronas cada una, es menor a todas las redes con 1 sola capa, mejorando en un 4,68% a la red con 50 perceptrones, en un 3,06% a la de 100 perceptrones y en un 6,72% a la de 200. Además, su tiempo de ejecución es 9,09%

menor que la red con 200 neuronas, a pesar de tener resultados similares. Para obtener estos resultados, cada instancia de RND se entrena utilizando los hiper-parámetros de la Tabla 4.12, en la columna de cantidad de capas ocultas y neuronas por capa.

Tabla 4.14. Resultados de RND para diferentes cantidades de perceptrones por capa, cantidad de capas LSTM.

Cantidad de capas LSTM	Cantidad de neuronas en la(s) capa(s) oculta(s)	RMSE *dataset* de entrenamiento [s]	RMSE *dataset* de prueba [s]	Tiempo de ejecución [h]
1	50	0,1483	0,1137	18,00
1	100	0,1448	0,1118	19,80
1	1.000	0,1510	0,1162	24,20
2	100	0,1389	0,1084	22,00

Fuente: Elaboración propia, 2017.

La combinación de metadatos de entrada md_2 permite obtener una mejor precisión en las predicciones. Según los resultados de la Tabla 4.15, utilizando la combinación md_2 se consigue un RMSE un 11,79% menor que el de utilizando md_1. En esta combinación, las variables poseen un menor rango de valores posibles, al comparar con md_1, lo que disminuye el largo de la matriz de la capa de *word embedding*¸ la cual posee un valor decimal entre 0 y 1 para cada combinación de valores en el vector de metadatos. Esta matriz, vista desde el punto de vista del aprendizaje de redes neuronales como una matriz de pesos cualquiera (W), al tener menores dimensiones es más simple de entrenar, por tanto, será más fácil encontrar los valores que mejor mapeen la situación dada por ese vector. Sin embargo, esto ocurre a costa que la cantidad de situaciones distintas apreciables para la red neuronal se vea disminuida. Para obtener estos resultados, se entrenan los modelos RND_{122} y RND_{222}, utilizando los hiper-parámetros de la Tabla 4.12, en la columna de variables de secuencia y metadatos.

Tabla 4.15. Resultados de RND para distintas combinaciones de metadatos.

Combinación de metadatos	*Dataset* de Entrenamiento		*Dataset* de Prueba	
	MSE [$\mathbf{km^2}$]	RMSE [km]	MSE [$\mathbf{km^2}$]	RMSE [km]
md_1	0,0210	0,1435	0,0172	0,1289
md_2	0,0171	0,1286	0,0132	0,1137

Fuente: Elaboración propia, 2017.

Utilizar la combinación de variables de serie de tiempo sd_2, que se compone de la variable Δd_i, entrega la mejor precisión en el *dataset* de prueba, con respecto a una serie de combinaciones

de variables probadas. Estas secuencias son de largo 10 y fueron entregadas como entrada para las capas LSTM, las cuales son manejadas por RND de manera separada a los metadatos (ver sección 3.3.2). En la Tabla 4.16, se puede ver que esta variable entrega un RMSE 0.77% menor que al utilizar sd_1, un 8,96% menor con respecto al sd_3, y un 96,51% menor que sd_4. Se interpreta que utilizar la misma variable de salida, como serie de tiempo de entrada a RND para el bloque recurrente, la red puede comprender de mejor manera los patrones y tendencias que se van generando a través del tiempo. Sin embargo, esta mejora es muy pequeña con respecto a una variable como el TT_i, que va aumentando en 30 segundos en cada *timestep* de manera constante, describiendo un comportamiento lineal. Para obtener estos resultados, se comparan los resultados al entrenar los modelos RND_{212}, RND_{222}, RND_{232} y RND_{242}, utilizando los hiper-parámetros de la Tabla 4.12, en la columna de variables de serie de tiempo y metadatos.

Tabla 4.16. Resultados de RND para distintas combinaciones de variables de serie de tiempo.

Combinación de variables de serie de tiempo	***Dataset* de Entrenamiento**		***Dataset* de Prueba**	
	MSE [$\mathbf{km^2}$]	**RMSE [km]**	**MSE [$\mathbf{km^2}$]**	**RMSE [km]**
sd_1	0,0176	0,1303	0,0175	0,1300
sd_2	0,0171	0,1286	0,0172	0,1290
sd_3	0,0196	0,1378	0,0207	0,1417
sd_4	20,3239	4,0504	18,4238	3,6945

Fuente: Elaboración propia, 2017.

Utilizar como variable de salida a $y_2 = \Delta d_i$, entrega una mejor precisión que utilizar la distancia que ha recorrido el bus desde el inicio del recorrido. En la Tabla 4.17 se puede apreciar que el valor del RMSE mejora en un 96,59% para el *dataset* de entrenamiento y un 96,16% para el *dataset* de prueba. La forma de aprendizaje que debe realizar RND es distinta en cada caso, con respecto a la forma en que se comportan los datos. En el primero los valores van oscilando alrededor de cierto rango, dependiendo del segmento del recorrido en el que se encuentra, y las variables de contexto asociadas. Mientras que, en el segundo caso, el valor debe ir aumentando con respecto a la secuencia de valores anteriores, además de captar el contexto provisto por los metadatos. En este sentido se puede apreciar que dejar un rango de valores tan amplio, de 18 km, a una variable que es creciente, por la naturaleza del problema, da mayores complicaciones a la red a entregar respuestas coherentes. Por lo tanto, es posible inferir que la red aprende de mejor manera al dar por hecho este aspecto y solo ocuparse de predecir cuánto va a avanzar al próximo muestreo. Para obtener estos resultados, se entrena los modelos RND_{221} y RND_{222}, utilizando los hiper-parámetros de la Tabla 4.12, en la columna de variables de salida.

Tabla 4.17. Resultados de RND para distintas variables de salida.

Variable de salida	*Dataset* de Entrenamiento		*Dataset* de Prueba	
	MSE [$\mathbf{km^2}$]	RMSE [km]	MSE [$\mathbf{km^2}$]	RMSE [km]
y_1	20,3233	4,0504	18,4238	3,6945
y_2	0,0195	0,1378	0,0206	0,1416

Fuente: Elaboración propia, 2017.

Al utilizar en el bloque de metadatos, una capa que mapeo los metadatos a *word embedding* y luego una capa oculta, se consiguen mejores precisiones que solo utilizar una capa oculta. La función de activación utilizada para las capas ocultas implementadas es sigmoidal, con rango de valores entre 0 y 1. Según los resultados de la Tabla 4.18, al agregar la capa de *word embedding*, el RMSE mejora un 18,05% para el *dataset* de entrenamiento y un 0,70% para el *dataset* de prueba. La capa de *word embedding* es implementada para permitir a la red neuronal mapear diferentes contextos de un viaje de un bus a una representación en un espacio vectorial unitario, de manera que no exista necesariamente una relación lineal entre la cercanía de los valores de los metadatos, y la semejanza de sus implicancias prácticas. Esta condición permite a RND aprender a mapear los valores que disminuyen el error para cada combinación de metadatos, por separado. Para obtener estos resultados, se entrada el modelo RND_{222} utilizando los hiper-parámetros de la Tabla 4.12, en la columna de arquitectura de metadatos.

Tabla 4.18. Resultados de RND para distintas formas de manejar los metadatos.

Manejo de metadatos	*Dataset* de Entrenamiento		*Dataset* de Prueba	
	MSE [$\mathbf{km^2}$]	RMSE [km]	MSE [$\mathbf{km^2}$]	RMSE [km]
Capa de word embedding y capa oculta	0,0171	0,1286	0,0172	0,1289
Capa oculta (Sigmoidal)	0,0254	0,1569	0,0174	0,1299

Fuente: Elaboración propia, 2017.

En la Figura 4.6, se muestra un ejemplo las predicciones que realiza RND, de las distancias que va recorriendo un bus, comparado con los datos reales. Las predicciones se realizaron a partir del *timestep* 38, donde la distancia acumulada de alrededor de 3,86 km. Se utilizaron las propias predicciones realizadas como nuevas entradas a RND de manera iterativa para realizar predicciones a nuevos *timestep*, de manera de predecir secuencias completas de distancias recorridas, hasta el fin de la ruta. Se observa que las predicciones de RND siguen casi linealmente la pendiente de los primeros *timestep*, y por tanto lo logra predecir la desaceleración en el avance del bus entre los 1500 a 2500 segundos de tiempo de viaje del bus. Sin embargo, debido al aumento de la velocidad del bus, en los datos reales, entre los 2500 y 3500 segundos, las

predicciones comienzan a coincidir en gran medida con los datos reales hasta el final del viaje. Finalmente, el MAE entre la secuencia predicha y los datos reales es de 1,211 km, lo cual está muy por encima de los resultados mostrados anteriormente, al predecir 1 solo *timestep* en el futuro.

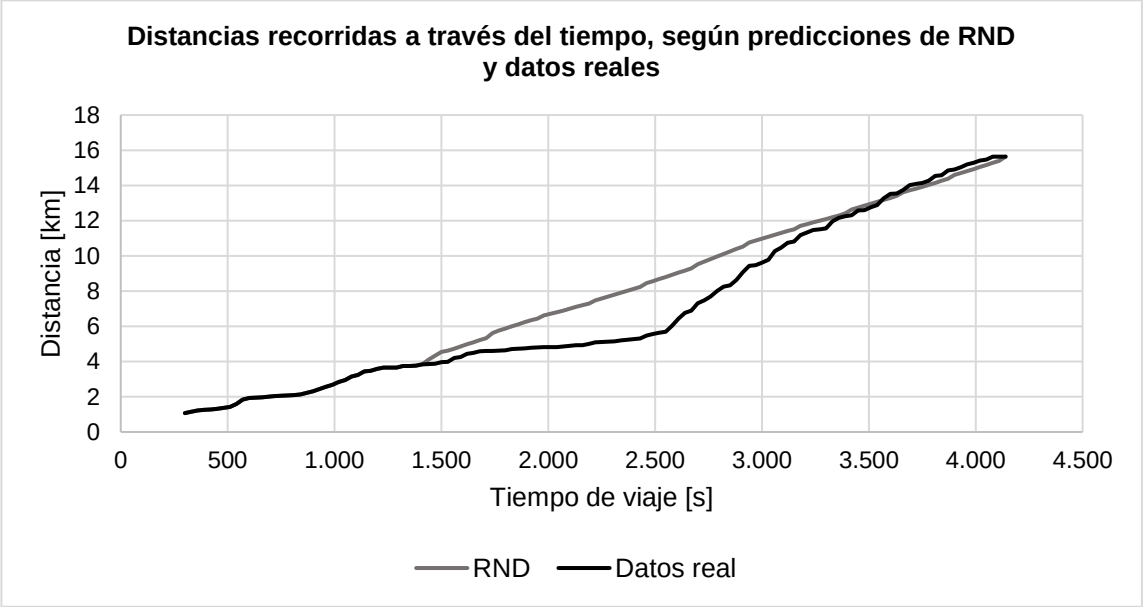


Figura 4.6. Ejemplo de distancias recorridas por un bus a través del tiempo, según predicciones de RND y datos reales.

Fuente: Elaboración propia, 2017.

4.3 RESULTADOS OBTENIDOS CON EL MODELO RNH

Luego de la realizar un proceso de ajuste de los hiper-parámetros para RNH, se consiguen los valores que permiten minimizar el error en las predicciones, los cuales se presentan en la Tabla 4.20. Posteriormente, se desglosan las pruebas realizadas para cada hiper-parámetro por separado. En la Tabla 4.19 se muestran los resultados de RNH entrenada con estos hiper-parámetros, donde se consigue en las predicciones sobre el *dataset* de prueba, un MSE de 1.244,7353, un MAE de 23,8831 segundos y un RMSE de 34,1872 segundos. En términos prácticos, esto significa que, dado un par de buses en tránsito, utilizando el valor de headway entre ellos en los últimos 4 paraderos, y un conjunto de metadatos del contexto de los viajes, es posible predecir la variación del *headway* entre ellos en el próximo paradero de la ruta, con un error del 3.98% con respecto a la variación máxima registrada, que es de 600 segundos.

Tabla 4.19. Resumen de resultados finales de RNH.

Dataset	MSE [s^2]	RMSE [s]	MAE [s]
Dataset de entrenamiento	1.343,6372	35,3509	24,3983
Dataset de prueba	1.244,7353	34,1872	23,8831

Fuente: Elaboración propia, 2017.

Tabla 4.20. Hiper-parámetros utilizados para conseguir los resultados finales de RNH.

Nombre del hiper-parámetro	Valor
Épocas de entrenamiento	10
Tasa de aprendizaje	10^{-5}
Método de aprendizaje	ADAM
Cantidad de neuronas por capa oculta	50

Fuente: Elaboración propia, 2017.

En la Figura 4.7, se muestra un ejemplo de predicción de valores de *headway*, entre un par de buses, realizados por RNH y sus respectivos valores reales. Se utilizaron los datos parciales del par de viajes hasta el paradero 20 para realizar predicciones a los paraderos siguiente. Se puede apreciar que las predicciones de RNH describen una tendencia descendente hasta el paradero 32, coincidiendo con la tendencia de la serie de tiempo de los primeros 20 paraderos. En este sentido, la red no logra emular correctamente los valores reales, que describen una tendencia a permanecer en un *headway* negativo cercano a 0 segundos hasta el paradero 35. Los valores de *headway* que entregan las predicciones se mantienen a una distancia que permite tener un MAE de 98,23 con respecto a los datos reales, lo cual está muy por encima del promedio de errores al realizar pruebas con el *dataset* completo, debido a que se está utilizando las propias predicciones realizadas, como nuevas entradas a RNH para hacer predicciones a más de 1 *timestep* de distancia.

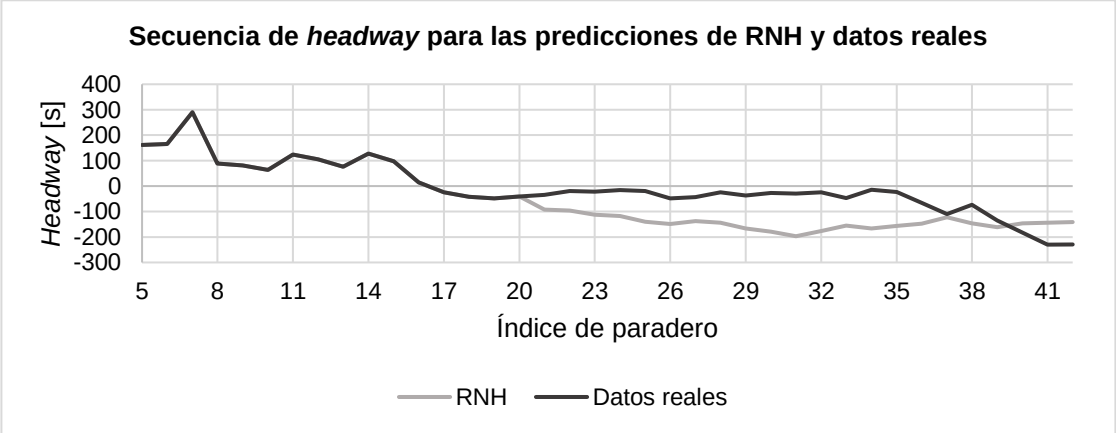


Figura 4.7. Ejemplo de secuencia de headway, para predicciones de RNH y datos reales.

Fuente: Elaboración propia, 2017.

Luego de evaluar los distintos modelos de predicción para para RNH, se determina que RNH_{232} es el que consigue minimizar el error en las predicciones. Este modelo contiene la combinación de variables de entrada de metadatos mh_2, la combinación de variables de entrada de serie de tiempo sh_3, y la variable de salida y_2, en concordancia con la nomenclatura de modelos de predicción de la ecuación 3.24. Específicamente, se utiliza como entrada de serie de tiempo, una secuencia de 4 *timestep* consecutivos, con valores de *headway* entre un par de buses (h_{ijk}), entre el paradero *k-3* a *k*. Se utiliza como entrada de metadatos, *dw*, *t* y *i*, los cuales coinciden con el vector de metadatos encontrado para RND. Luego, se entrega como salida, una predicción de Δh_{ijk+1}, en el paradero siguiente al último muestreo de la serie de tiempo ingresada. Posteriormente se desglosan los resultados de las pruebas realizadas para determinar cada combinación de variables.

En la Tabla 4.21 se observan los valores de los hiper-parámetros restantes utilizados para la realización de pruebas de cada hiper-parámetro, para RNH. En la primera columna de cada fila se encuentra el nombre de los hiper-parámetros cuyos valores son fijados para realizar pruebas sobre un hiper-parámetro en específico. En cada columna se encuentra el nombre del hiper-parámetro que es probado, al evaluar los resultados obtenidos al utilizar distintos valores. Por ejemplo, para encontrar la tasa de aprendizaje más idónea para RNH, se entrenan múltiples instancias distintas, cada una utilizando valores de tasa de aprendizaje distintos, pero con los mismos valores del resto de hiper-parámetros y utilizando los mismos datos de entrenamiento. En este caso, se realizan 10 épocas de entrenamiento, con un método de aprendizaje de ADAM, con 2 capas ocultas LSTM, cada una con 50 neuronas, para el modelo de predicción RNH_{232}.

Al utilizar una tasa de aprendizaje de 10^{-5}, se consigue un aprendizaje de la red constante y significativo a través de las épocas. En la Figura 4.8 es posible apreciar que el valor de MSE para una tasa de aprendizaje de 10^{-5}, va disminuyendo en promedio un 0,63 por época, mientras que con una tasa de aprendizaje de 10^{-6} y de 10^{-4}, solo de un 0,09 y -0,09, respectivamente. Para una tasa de aprendizaje de 10^{-4}, se aprecia una fluctuación descontrolada del error, no apreciándose un descenso claro en la curva de MSE, lo que sugiere que la tasa de aprendizaje es muy grande. Para una tasa de 10^{-6}, el descenso del MSE ocurre de manera constante, pero muy lenta a través de las épocas, con respecto a los resultados con una tasa de 10^{-5}. Para conseguir estos resultados, se entrena el modelo RNH_{232}, utilizando los hiper-parámetros de la Tabla 4.21, en la columna de tasa de aprendizaje.

Tabla 4.21. Valores de hiper-parámetros restantes utilizados para la realización pruebas de cada hiper-parámetro, para RNH.

Hiper-parámetros fijos utilizados	Hiper-parámetro por probar					
	Tasa de aprendizaje	**Método de aprendizaje**	**Cantidad de capas ocultas y neuronas por capa**	**Variables de secuencia y metadatos**	**Variables de salida**	**Función de activación**
Épocas de entrenamiento	10	10	10	10	10	10
Tasa de aprendizaje	-	10^{-6}	10^{-4}	10^{-6}	10^{-6}	10^{-6}
Método de aprendizaje	ADAM	-	ADAM	ADAM	ADAM	ADAM
Cantidad de capas LSTM	2	2	-	2	2	2
Cantidad de neuronas por capa oculta	50	100	-	100	100	100
Función de activación	Tangente hiperbólica	Tangente hiperbólica	Tangente hiperbólica	Tangente hiperbólica	Tangente hiperbólica	-
Modelo de predicción	RNH_{232}	RNH_{231}	RNH_{232}	-	-	RNH_{111}

Fuente: Elaboración propia, 2017.

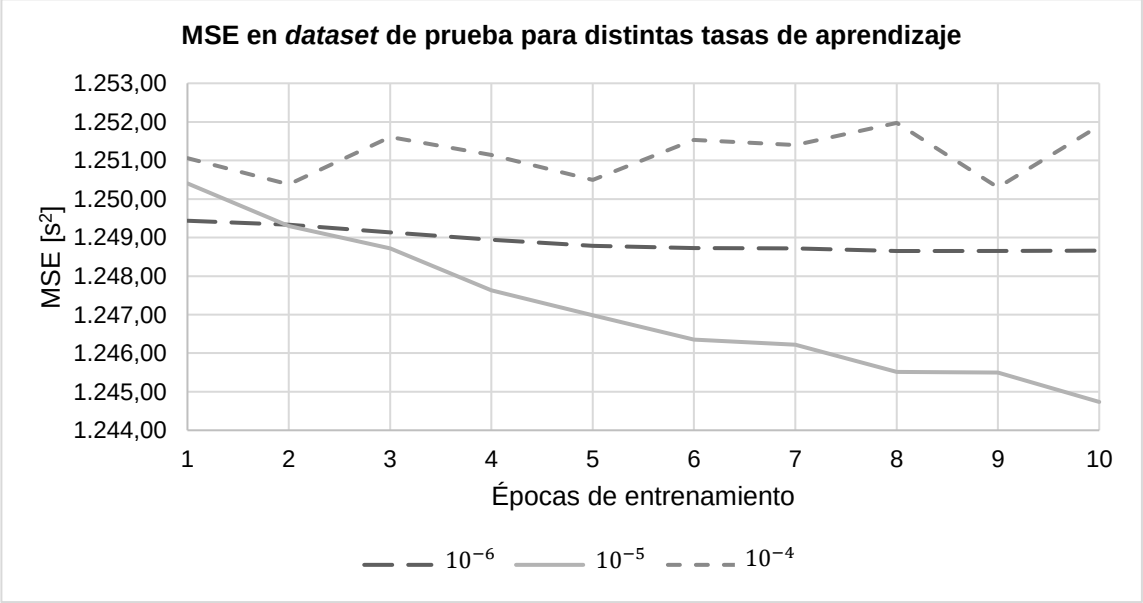


Figura 4.8. MSE en dataset de prueba, para cada época de entrenamiento, usando distintas tasas de aprendizaje.

Fuente: Elaboración propia, 2017.

Los resultados de RNH, entrenada con el método ADAM, son mejores que al entrenar con SGD. En la Tabla 4.22 se aprecia que, al realizar predicciones sobre el *dataset* de entrenamiento, el

valor de RMSE es 1,23% menor al utilizar ADAM, y para el dataset de prueba, un 1,09% menor. Esto confirma, en concordancia con los resultados de los modelos anteriores, que ADAM es un algoritmo de optimización para redes neuronales más robusto que SGD, al incorporar el concepto de *momentum* del valor del gradiente e implementar factor de normalización en base al cuadrado del gradiente de la función objetivo. Para obtener estos resultados, se entrena RNH_{231} utilizando los valores de hiper-parámetros presentes en la Tabla 4.21, en la columna de método de aprendizaje.

Tabla 4.22. Resultados de RNH para distintos métodos de aprendizaje.

Método de optimización	***Dataset* de Entrenamiento**			***Dataset* de Prueba**		
	MSE [s^2]	**RMSE [s]**	**MAE [s]**	**MSE [s^2]**	**RMSE [s]**	**MAE [s]**
SGD	1.379,2335	35,8296	24,8980	1.275,2343	34,6179	24,3489
ADAM	1.346,2984	35,3863	24,4530	1.248,7196	34,2391	23,9636

Fuente: Elaboración propia, 2017.

Al utilizar 2 capas LSTM con 50 neuronas cada una, tanto el RMSE como el MAE son menores que el resto de configuraciones, al momento de realizar evaluar las predicciones realizadas en el *dataset* de prueba. En la Tabla 4.23, es posible ver que el valor de RMSE de la configuración antes mencionada, es un 0,26% menor que al utilizar 1 capa de 50 neuronas, 0,21% menor que utilizar 1 capa de 100 neuronas, y 0,62% que al utilizar una capa de 200 neuronas. La mejora de resultados, al pasar de utilizar 1 capa de 50 neuronas a una 100, se puede explicar como que la una mayor complejidad de la red mejora la posibilidad de ajustar patrones en las series de tiempo. Mientras que la disminución del rendimiento apreciable, al pasar de 1 capa de 100 a 200 neuronas, se puede explicar como que al tener una mayor cantidad de pesos que ajustar, el proceso de aprendizaje es más complejo, por lo que provoca que sea más lento, entregando peores resultados en las primeras épocas de entrenamiento. Para conseguir estos resultados, se entrena el modelo RNH_{232}, utilizando los valores de hiper-parámetros presentes en la Tabla 4.21, en la columna de cantidad de capas ocultas y cantidad de neuronas por capa.

Tabla 4.23. Precisión de RNH, para distintas cantidades de neuronas y capas LSTM.

Cantidad de capas LSTM	**Cantidad de neuronas por capa LSTM**	***Dataset* de entrenamiento**		***Dataset* de prueba**	
		RMSE [s]	**MAE [s]**	**RMSE [s]**	**MAE [s]**
1	50	36,7371	24,4718	35,3745	23,9601
1	100	36,7249	24,4584	35,3564	23,9398
1	200	36,8647	24,6292	35,5019	24,1085
2	50	36,6557	24,3983	35,2808	23,8831

Fuente: Elaboración propia, 2017.

La combinación de metadatos mh_2 consigue minimizar los errores en las predicciones de RNH. En la Tabla 4.24, se aprecia que el valor del RMSE en el *dataset* de entrenamiento que consigue mh_2 es un 2,17% menor que utilizando mh_1, y en el *dataset* de prueba es un 2,04% menor que los resultados conseguidos con mh_1. Esta combinación de metadatos coincide con la combinación de metadatos utilizada para RND, la cual permite disminuir la dimensionalidad de la capa de *word embedding*, haciendo más simple para la red neuronal aprender a diferenciar y caracterizar cada vector de metadatos, de manera de entregar una salida coherente, que disminuye el error de las predicciones. Los valores de la matriz de la capa de *word embedding*, posee la información que permite caracterizan las circunstancias de un cierto par de viajes, entre cada par de paraderos, en cada intervalo del día en que se realiza el servicio y en cada día de la semana. Para obtener estos resultados, se entrenan los modelos RNH_{132} y RNH_{232} utilizando los valores de hiper-parámetros presentes en la Tabla 4.21, en la columna de variables de secuencia y metadatos.

Tabla 4.24. Resultados de RNH para distintas combinaciones de metadatos.

Combinación de metadatos	***Dataset* de Entrenamiento**			***Dataset* de Prueba**		
	MSE [s^2]	**RMSE [s]**	**MAE [s]**	**MSE [s^2]**	**RMSE [s]**	**MAE [s]**
mh_1	1.411,4961	36,3010	25,4155	1.308,2052	35,1082	24,8885
mh_2	1.379,2335	35,8296	24,8980	1.275,2343	34,6179	24,3489

Fuente: Elaboración propia, 2017.

La combinación sh_3 obtiene los menores errores en el *dataset* de prueba. Esta combinación solo contempla una serie de tiempo parcial de 4 *timestep* de largo, con la variable de h_{ij} en los últimos 4 paraderos comunes entre un par de buses en tránsito. Se puede apreciar en la Tabla 4.25, que utilizando sh_3 se consigue un RMSE un 0,82% menor que con sh_2, y un 94,66% menor que utilizando sh_1. Debido a que la variable de *headway* entre 2 viajes está presente en todas las versiones probadas, es posible interpretar que, al utilizar una mayor cantidad de variables, estas solo aportan ruido, dificultando la capacidad de las capas LSTM de encontrar patrones en las series de tiempo utilizadas. También, esto muestra la dependencia temporal que posee el valor de *headway* entre dos buses (h_{ijk}) para paraderos consecutivos. Para obtener estos resultados, se entrenan los modelos RNH_{212}, RNH_{222} y RNH_{232} utilizando los valores de hiper-parámetros presentes en la Tabla 4.21, en la columna de variables de secuencia y metadatos.

Tabla 4.25. Resultados de RNH para distintas combinaciones de variables secuencia.

Combinación de variables de secuencia	*Dataset* de Entrenamiento		*Dataset* de Prueba	
	MSE [s^2]	RMSE [s]	MSE [s^2]	RMSE [s]
sh_1	80.221.189,6054	8.876,0705	78.673.680,8833	8.784,2311
sh_2	363.722,2341	463,3531	365.883,5815	473,2182
sh_3	368.189,4322	466,0570	358.996,8501	469,3168

Fuente: Elaboración propia, 2017.

Utilizar la variable de salida $y_2 = \Delta h_{ijk+1}$ permite conseguir mejores resultados en las predicciones de RNH. En la Tabla 4.26, es posible apreciar que para *dataset* de entrenamiento, existe una disminución del RMSE es de un 92,22%, y para el dataset de prueba, es de un 92,48%. Al utilizar Δh_{ijk+1} disminuye el rango de valores posibles en que puede oscilar la variable de salida, pasando de 7.200 segundos a 1.200 segundos. Este último valor consiste en el valor máximo que oscila el headway entre 2 buses entre un par de paraderos consecutivos. Al utilizar h_{ijk+1}, la red tiene solo un rango máximo de un sexto del rango total, donde las predicciones que realiza coherentes, lo cual posiblemente dificulta el aprendizaje de la red neuronal. Para obtener estos resultados, se entrenan los modelos RNH_{231} y RNH_{232} utilizando los hiper-parámetros presentes en la Tabla 4.21, en la columna de variables de salida.

Tabla 4.26. Resultados de RNH para distintas variables de salida.

Variable de salida	*Dataset* de Entrenamiento		*Dataset* de Prueba	
	MSE [s^2]	RMSE [s]	MSE [s^2]	RMSE [s]
y_1	369.359,5239	466,9831	369.359,5239	466,9831
y_2	1.411,4961	36,3010	1.308,2052	35,1082

Fuente: Elaboración propia, 2017.

Utilizar una función de activación de tangente hiperbólica en las capas ocultas, consigue mejor precisión en las predicciones de RNH, que utilizar una función sigmoidal. En la Tabla 4.27, se observa que, al realizar predicciones en el *dataset* de prueba, el RMSE al utilizar tangente hiperbólica es un 8.68% menor que al utilizar una función sigmoidal. La variable salida de RNH admite valores negativos y positivos, por lo que utilizar una función como tangente hiperbólica que posee un rango de -1 a 1, permite manejar estos valores internamente de manera natural. Es decir, sin necesidad de realizar normalizaciones que manejen los valores de *headway* negativos en un rango de valores positivos, como es el caso al utilizar una función de activación sigmoidal, con rango entre 0 y 1. Para obtener estos resultados, se entrena el modelo RNH_{111}

utilizando los hiper-parámetros presentes en la Tabla 4.21, en la columna de función de activación.

Tabla 4.27. Resultados de RNH, para distintas funciones de activación.

Función de activación	*Dataset* de Entrenamiento		*Dataset* de Prueba	
	MSE [s^2]	RMSE [s]	MSE [s^2]	RMSE [s]
Sigmoidal	97.746.488,98	9.778,74	93.014.514,61	9.619,49
Tangente hiperbólica	80.221.189,61	8.876,07	78.673.680,88	8.784,23

Fuente: Elaboración propia, 2017.

4.4 RESULTADO OBTENIDOS POR EL MODELO MPH

La precisión de las predicciones de un modelo basado en el promedio de los datos históricos es inferior que el de RNP. En la Tabla 4.28, se aprecia que el valor del RMSE de RNP es un 44,66% menor que MPH, y el MAE de RNP es un 46,11% menor que este modelo. Es posible comparar estos resultados con los de RNP, debido a que ambos predicen el tiempo de viaje que realiza un bus entre un rango de paraderos, en este caso entre el inicio del viaje y un cierto paradero *i* (TT_{1i}). Sin embargo, se debe tomar en cuenta que las variables de entrada y los procedimientos que realiza cada modelo para realizar las predicciones son distintos. Este modelo fue creado como referencia para poder comparar los resultados de los modelos basados en redes neuronales, debido a que se considera a prior que existe una alta linealidad en los tiempos de viaje de los buses.

Tabla 4.28. Resultados generales de MPH comparados con RNP.

Modelo	MSE [s^2]	RMSE [s]	MAE [s]
MPH	112.339,1003	233,4363	176,9877
RNP	16.688,8851	129,1852	95,3805

Fuente: Elaboración propia, 2017.

4.5 COMPARACIÓN DE RESULTADOS EN PREDICCIÓN DE EVENTOS DE *BUNCHING*

En la Figura 3.14, se presenta un ejemplo comparativo de las predicciones realizadas por cada modelo para la secuencia de headway entre los viajes 31° y 32° del día 5 de diciembre de 2016. Las predicciones de realizan a partir del paradero 17, debido a que el proceso de captura de datos se hizo con una ventana de tiempo entre las 8:00 y las 8:30, por lo que solo se capturaron los datos del recorrido en los primeros 16 paraderos. El área de *bunching*, marcada en la figura muestra el rango de valores de *headway* donde se detecta *bunching*¸ correspondiente a cuando

valor absoluto del *headway* en algún paradero es menor a un 25% del *headway* inicial, según la Ecuación 3.25. El proceso de captura de datos se realiza por separado sobre el *dataset* de cada modelo implementado. Luego se realizan las predicciones de los viajes de buses involucrados, de modo de completar su recorrido. Finalmente, se generan secuencias de *headway* entre los pares de viajes, que posean un *headway* inicial menor a 1 hora (véase sección 3.5.3).

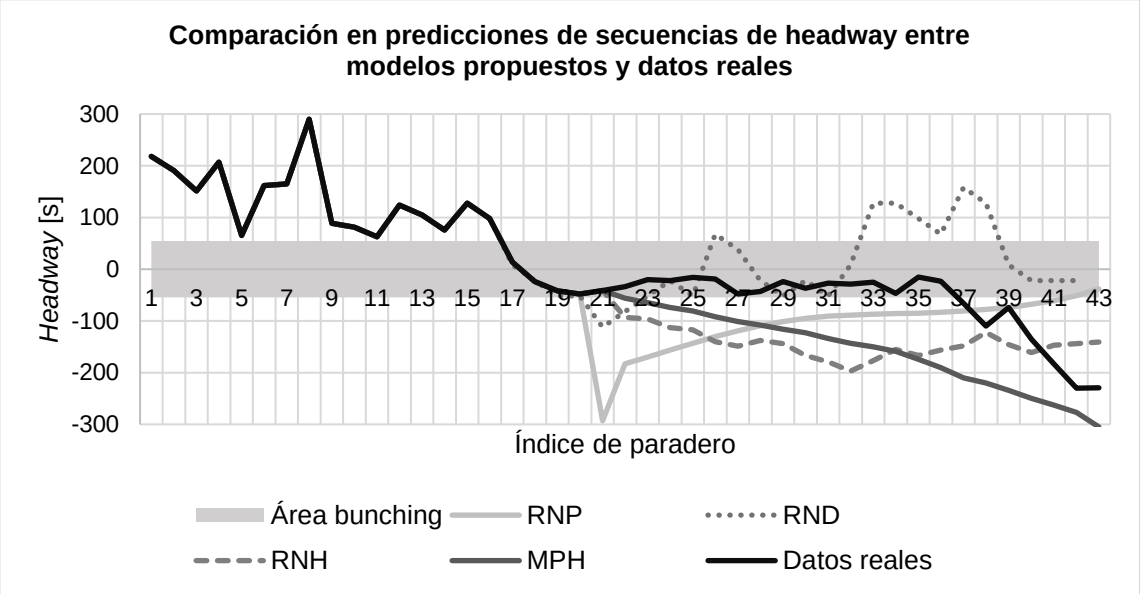


Figura 4.9. Ejemplo de secuencia de headway predicha para cada modelo propuesto, comparado con los datos reales, y aplicación de algoritmo de bunching.
Fuente: Elaboración propia, 2017.

En el ejemplo presentado en la Figura 4.9, el modelo RND logra una mayor precisión en la detección de *bunching* en cada paradero, con un 59,24% de aciertos del total de paraderos predichos. La Tabla 4.29 muestra los resultados de la aplicación del algoritmo de detección de bunching sobre la secuencia de valores de headway predicha por cada modelo. Estos resultados solo sirven para ejemplificar las diferencias en el funcionamiento de cada modelo, en cuanto a la predicción de *bunching*, y no para comparar el rendimiento entre cada uno. La métrica de precisión consiste en la cantidad de aciertos en las predicciones de un cierto modelo, con respecto al total de paraderos predichos, que los cuales son 27 en este caso. La segunda métrica el porcentaje de paraderos correctamente predichos donde existe *bunching*, según los datos reales. La tercera métrica corresponde al porcentaje de paraderos correctamente predichos, del conjunto de paraderos donde no ocurre *bunching* según los datos reales.

Anterior al paradero 17 no se detecta *bunching* entre los buses implicados, como es posible apreciarlo en la Figura 4.9, donde la curva está por sobre la zona de detección de *bunching*. Sin embargo, RNP, RNH y MPH logran predecir correctamente el descenso del *headway* en el paradero 17, detectando correctamente el *bunching* a producirse. También es posible apreciar

que ningún modelo predice correctamente la secuencia completa de paraderos donde ocurre, el cual se presenta entre el paradero 17 y el 36, donde RND logra el mejor rendimiento, con un 65% de la secuencia correctamente predicha. Desde el paradero 37 al 43 los buses dejan de estar en *bunching*, al seguir disminuyendo el valor del *headway*. Un valor de *headway* negativo significa que el bus que comienza posteriormente su recorrido, en algún punto sobrepasa al bus que tenía enfrente, invirtiendo sus posiciones. En esta sección del recorrido RNP, RNH y MPH logran obtener un 100% de precisión en las predicciones de *bunching*¸ mientras que RND logra solo un 42,86% de precisión.

Tabla 4.29. Resumen de resultados de predicción de bunching para cada modelo, en un par de viajes específico.

Métrica	**RNP**	**RND**	**RNH**	**MPH**
Precisión	37,04%	59,26%	44,44%	37,04%
Paraderos en Bunching correctamente predichos	15,00%	65,00%	25,00%	15,00%
Paraderos sin *Bunching* correctamente predichos	100,00%	42,86%	100,00%	100,00%

Fuente: Elaboración propia, 2017.

Aplicando el método antes expuesto al conjunto completo de ventanas de tiempo, con las métricas presentadas en la sub-sección 3.5.5, se obtiene que el modelo RNH posee una mejor precisión en la predicción de ocurrencia de fenómenos de *bunching* entre un par de buses. Esto se observa en la Tabla 4.30, donde RNH posee el valor de ACC_2 más alto con un 91,75%. Esto puede deberse a que este modelo posee la aproximación más directa a la solución, al realizar directamente predicciones de secuencias de *headway*, entre un par de buses. Esto permite predecir eventos de *bunching*, simplemente aplicando el algoritmo de detección a estas secuencias predichas. En contraste, los otros modelos predicen los tiempos de viaje de un conjunto de buses por separado, para luego combinar las predicciones para generar estas secuencias de *headway*. Nótese que la precisión de estas secuencias depende directamente de la precisión de las predicciones de los 2 viajes que la componen. Esto implica que el orden de magnitud del error en las secuencias de *headway*, para estos modelos, va a ser mayor que en los viajes predichos, por separado.

Tabla 4.30. Tabla comparativa de resultados de predicción de bunching para los distintos modelos implementados.

Modelo de predicción	ACC_1 [%]	ACC_2 [%]	SES [%]	SPC [%]
RNH	83,70	91,75	42,63	99,04
RND	84,15	91,07	21,79	96,07
RNP	71,05	86,46	34,33	90,60
MPH	74,04	85,63	23,59	90,57

Fuente: Elaboración propia, 2017.

Los resultados que entrega RND son similares a RNH, a pesar de que el primero tiene una aproximación indirecta en la generación de secuencias de *headway*. Esto es posible verlo, comparando los valores de la Tabla 4.30 entre ambos modelos. En las métricas ACC_1 y ACC_2, RND posee un 0,53% mayor y 0,75% menor que RNH, respectivamente, mientras que para las métricas de SES y SPC, RNH entrega valores 95,64% y 3,09% mayores a RND, respectivamente. A pesar de que RND tenga considerablemente peores resultados que RNH en SES y SPC, consigue un resultado muy cercano a este último en ACC_2. Cabe destacar que el valor de ACC_2 es un promedio entre la métrica SES y SPC, ponderado según la cantidad de paraderos en que ocurre y no ocurre bunching, respectivamente. Esto se puede explicar sobre el hecho de que solo el 7,22% de los paraderos evaluados presentaban eventos de *bunching* entre los buses involucrados, por lo que la diferencia de valores en la métrica SES no afecta demasiado el valor de ACC_2.

La falta de precisión de todos los modelos implementados en la métrica de SES, que representa la tasa de predicciones correctas en los paraderos donde ocurre *bunching*, se puede deber a la falta de casos de prueba en que ocurre *bunching* de buses. Esto dificulta a las redes neuronales de poder predecir este fenómeno, debido a que estadísticamente, ya sea utilizando promedios históricos o utilizando procesos de aprendizaje en redes neuronales, los modelos van a tender a adecuarse a los casos en que no ocurre *bunching*¸ debido a su predominancia en el *dataset*. Cada modelo puede determinar con una mayor precisión el hecho de que ocurra o no algún evento de *bunching* entre un par de buses en alguno de los paraderos de la ruta, representado por el valor de ACC_1, que el conjunto de paraderos específico donde va a ocurrir *bunching*. El valor más bajo lo consigue el modelo de promedios con un 21,79%, y el valor máximo lo consigue RNH con un 42,63%.

Los modelos que implementaron redes neuronales recurrentes LSTM, obtuvieron un mejor rendimiento en la predicción de eventos de *bunching* entre un par de buses de un servicio. Esto concuerda con los resultados obtenidos en la literatura, donde las implementaciones de redes neuronales recurrentes consiguen una mayor precisión al abordar problemas que presentan series de tiempo, y, por lo tanto, sus variables poseen una alta dependencia temporal. Esto es

posible verlo al comparar los valores obtenidos en la métrica ACC_1 donde los modelos RNH y RND, que implementan redes neuronales LSTM, consiguen valores al menos 9,66% mayores que RNP y MPH. Para la métrica ACC_2 los modelos RND y RNH consiguen valores al menos 4,61% mayores que RNP y MPH, mientras que en la métrica SPC, RND y RNH consiguen valores al menos 5,47% mayores que RNP y MPH. Sin embargo, para la métrica SES no se aprecia la misma tendencia.

El modelo RNH posee la mayor flexibilidad para distinguir los valores de *headway* que representan eventos de *bunching*, de los que no, incluso utilizando umbrales de tolerancia distintos a los recomendados por la literatura. En la Figura 4.10, se muestra un gráfico de curva ROC para cada modelo implementado, junto con su respectiva área bajo la curva (AUC). RNH obtiene el mayor AUC con un valor de 0,7041, superando en un 6,2184% a RND, en 8,2289% a RNP y en un 25,3111% a MPH. Cabe destacar que todos los modelos, permiten distinguir la ocurrencia de *bunching* mejor que un clasificador aleatorio, el cual es representado por la línea punteada gris. Los puntos destacados sobre la curva en cada modelo, representa el valor de sensibilidad y 1 – especificidad, presentados anteriormente en la Tabla 4.30. Estos puntos son relevantes porque representan la capacidad de los modelos de realizar clasificaciones sobre las series de valores de *headway*, aplicando la ecuación (3.37), la cual admite un umbral del 25%. El resto de valores fueron obtenidos aplicando distintos porcentajes de umbral, en incrementos de 5%, hasta llegar a 100%.

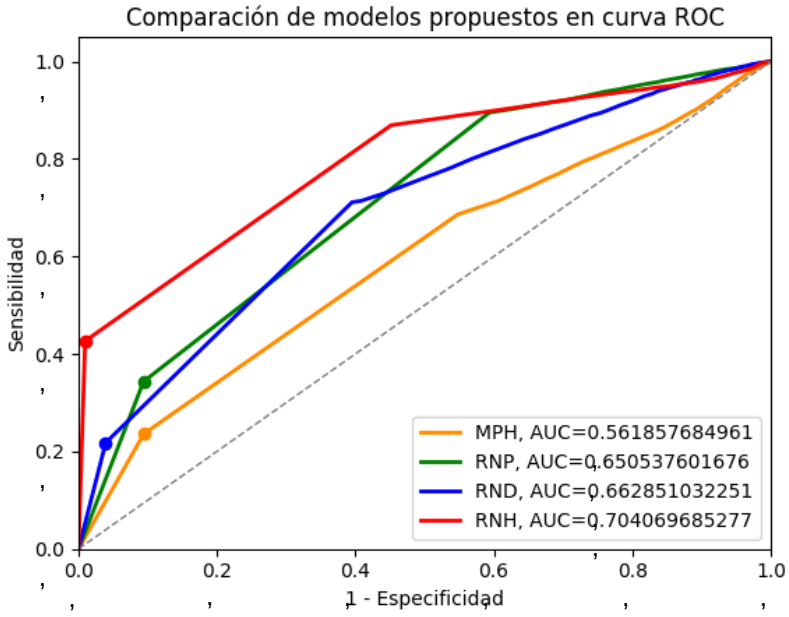


Figura 4.10. Curva ROC comparativa de los modelos propuestos, en la clasificación de bunching.

Fuente: Elaboración propia, 2017.

4.6 EVALUACIÓN DE PREDICTIBILIDAD EN SERIES DE TIEMPO UTILIZADAS

Se implementó un modelo de persistencia para RND y para RNH, donde se utilizó la serie de tiempo de la distancia recorrida en tiempos de 30 segundos por un bus en un viaje específico (Δd) y la diferencia en el valor de *headway* entre 2 buses en cada paradero de la ruta (Δh). Este modelo consiste en utilizar el *timestep* anterior de la serie como predicción para el siguiente *timestep*. Si este modelo presenta mejores resultados que los de la red neuronal respectiva, significa que la serie de tiempo utilizada posee un comportamiento de camino aleatorio. Esto se debe a que no es posible predecir la componente $\Phi(\tau)$ de estas series de tiempo, por su comportamiento aleatorio, por lo que el modelo de persistencia constituiría el mejor predictor posible de construir. En ambos casos, para realizar las pruebas de rendimiento, se realizaron predicciones solo al siguiente *timestep*, con respecto al último ingresado en una serie de tiempo parcial, y no se tomó en cuenta el contexto de ventana de tiempo. Esto se debe a que no se está probando la idoneidad de las soluciones propuestas en un contexto real, sino que la capacidad de estos modelos de efectivamente extraer patrones y tendencias de la serie de tiempo.

La precisión de RND es mejor que la del modelo de persistencia para cada métrica utilizada. En la Tabla 4.31, se puede apreciar que, al evaluar las predicciones realizadas en el *dataset* de prueba, el valor de MAE de RND es un 29,65% menor que el modelo de persistencia, el valor MSE es un 51,47% menor, y el de RMSE un 31,11% menor. Estos resultados revelan que la serie de tiempo utilizada no se comporta como un camino aleatorio, debido a que RND logra predecir el comportamiento de la variable $\Phi(\tau)$, que describe el comportamiento de la serie de tiempo en el lapso τ entre cada muestra, al superar a un modelo que solo replica el valor del *timestep* anterior (ver sección 3.3.5). En este caso, este lapso τ es de 30 segundos, por lo tanto, dado un muestreo *j* de la serie de tiempo de RND, la variable $\Phi_j(\tau)$ representa la diferencia en la distancia recorrida en los últimos 30 segundos, con respecto a la distancia recorrida en muestreo anterior *j-1* ($\Phi_j(\tau) = \Delta d_j - \Delta d_{j-1}$).

Tabla 4.31. Comparación de resultados entre RND, y el modelo de persistencia para serie de tiempo de RND.

Dataset	Modelo de persistencia			RND		
	MAE [s^2]	MSE [s]	RMSE [s]	MAE [s^2]	MSE [s]	RMSE [s]
Entrenamiento	0,1012	0,0187	0,1367	0,0940	0,0144	0,1189
Prueba	0,1029	0,0191	0,1382	0,0724	0,0092	0,0952

Fuente: Elaboración propia, 2017.

Los valores de error de RNH son mayores que los arrojados por el modelo de persistencia para cada métrica utilizada. En la Tabla 4.32, se puede apreciar que realizando predicciones sobre el *dataset* de prueba, el valor de MAE de RNH es un 1,46% mayor que el modelo de persistencia, el valor de MSE un 3,18% mayor, y el de RMSE un 1,58% mayor. Estos resultados muestran que posiblemente la serie de tiempo utilizada para RNH, se comporta como un camino aleatorio, donde la mejor predicción posible de realizar, en este caso, es utilizar el valor de la diferencia entre los valores de *headway* de un par de buses en los últimos 2 paraderos ($\Delta h_{j+1} = h_j - h_{j-1}$). Esto implicaría para la serie de tiempo utilizada no es posible predecir correctamente el componente $\Phi(\tau)$, que aporta los cambios de la variable en el transcurso del tiempo τ entre cada paradero recorrido, al describir un comportamiento aleatorio. Por lo tanto, se presume que el bloque recurrente de RNH no logra captar los patrones internos de la serie de tiempo.

Tabla 4.32. Comparación de resultados entre RNH, y el modelo de persistencia para serie de tiempos de Δh.

Dataset	**Modelo de persistencia**			**RNH**		
	MAE [s^2]	**MSE [s]**	**RMSE [s]**	**MAE [s^2]**	**MSE [s]**	**RMSE [s]**
Entrenamiento	24,1517	1.317,1806	36,2930	24,3983	1.343,6336	36,6555
Prueba	23,5404	1.206,3232	34,7322	23,8831	1.244,7277	35,2807

Fuente: Elaboración propia, 2017.

CAPÍTULO 5. CONCLUSIONES

Tras la finalización del trabajo de tesis se pudo dar cumplimiento al objetivo general del proyecto, el cual consiste en diseñar, construir y probar un algoritmo para la predicción de *bunching* de buses, basado en redes neuronales recurrentes, utilizando datos históricos de un servicio de buses. Además, se cumplen a cabalidad los objetivos específicos en cuanto al diseño de redes neuronales que predigan los tiempos de viaje de un bus, evaluando la precisión de las predicciones con datos del mundo real, e implementando un algoritmo de detección de *bunching*¸ para lograr predecir la ocurrencia de este fenómeno. Para realizar el diseño de las redes neuronales, se realiza una inspección crítica del estado del arte en cuanto a la predicción de tiempos de viaje y la predicción de *bunching* de buses urbanos.

Los modelos que implementan redes neuronales recurrentes y manejo de metadatos (RND y RNH) superan los resultados obtenidos por el resto de modelos, en cuanto a la predicción de eventos de *bunching*. Esto concuerda con los trabajos presentados en la literatura, donde las implementaciones de redes neuronales recurrentes consiguen una mayor precisión al abordar problemas que presentan series de tiempo, reconociendo patrones en variables que poseen una alta dependencia temporal. En este caso, los mejores resultados los presenta RNH, el cual posee el coeficiente de correlación lineal más alto entre *timesteps* consecutivos de la variable utilizada para la serie de tiempo, lo que hace suponer que esta implementación aprovecha de mejor manera los patrones presentes en las series de tiempo para realizar predicciones más precisas.

Se propone un método innovador para realizar predicciones de eventos de *bunching* a largo plazo. Los modelos de redes neuronales propuestos fueron utilizados para realizar predicciones en un contexto similar al que se requiere en un contexto real, utilizando datos parciales de un conjunto de viajes para predecir sucesos de hasta 30 minutos en el futuro. Esto permite comprender de mejor manera la capacidad de estos modelos para servir efectivamente de solución. Esta característica se diferencia de la manera en que se realizan las pruebas en los modelos del estado del arte estudiados, los cuales generalmente realizan pruebas de precisión solo al siguiente *timestep* de una serie de tiempo. Esto impide conocer la capacidad de estos modelos de realizar predicciones en lapsos de tiempos mayores, y su posible aplicación para estos fines.

Si se quiere implementar este modelo con datos en tiempo real es necesario realizar el desarrollo de alguna herramienta que permita manejar de manera más cómoda y eficiente, la entrada de datos y salida. Por un lado, es posible implementar a partir de los modelos propuestos un sistema que apoye a los usuarios en la predicción de los tiempos de viaje de los buses. Por otro lado, es posible implementar un sistema de apoyo a la toma de decisiones tanto para los operadores de los servicios de buses, como del propio sistema de monitoreo de buses (CBM). Las características

de estos sistemas puedes ser múltiples, debido a que las tecnologías utilizadas permiten la integración con plataformas web o móviles.

GLOSARIO

Agrupamiento de buses urbanos (en inglés, *bus bunching*): se define como el intervalo de tiempo en que un bus transita al menos junto a otro bus del mismo servicio, separados por un *headway* menor a una cierta fracción del *headway* programado. El *headway* es la diferencia de tiempo que existe entre 2 buses que recorren la misma ruta, en llegar al mismo punto del recorrido.

Predicción de tiempo de viaje de buses (en inglés, *bus time prediction*): conjunto de técnicas que permiten predecir con cierta precisión, el tiempo de llegada de un bus a una cierta parada (en inglés*, estimated time of arrival* - ETA) y el tiempo de salida de un bus de cierta parada (en inglés, *estimated time of departure* - ETD). Con estos datos se puede estimar, además, cuánto demora un bus en recorrer el trayecto entre dos paradas.

Redes neuronales artificiales: modelos matemáticos y computacionales provenientes del área de machine learning, que son utilizados en el contexto de aprendizaje supervisado para resolver problemas de regresión y clasificación. Están inspirados en la biología del sistema nervioso central de los animales, específicamente el cerebro.

Redes neuronales recurrentes: modelos de redes neuronales artificiales, que presentan la característica de forman ciclos de retroalimentación dirigidos entre las conexiones de sus nodos. Esto le permite manejar dependencias temporales entre las variables de un problema, siendo idóneo para fenómenos de series de tiempo.

Sistemas inteligentes de transporte (en inglés*, Intelligent Transportation Systems*): conjunto de soluciones telemáticas, diseñadas para mejorar la eficiencia, confiabilidad y seguridad del transporte terrestre. Principalmente se encarga del manejo de tráfico en calles y carreteras, y del manejo de información de viaje para los usuarios.

Transporte público urbano: medios por los cuales la gente se transporta dentro de una ciudad o centro urbano. Entre ellos están los autobuses, taxis colectivos, ferrocarriles y trenes, entre otros. Poseen como característica que los pasajeros comparten el medio de transporte y deben adaptarse a los horarios y rutas que ofrezca cada proveedor.

REFERENCIAS BIBLIOGRÁFICAS

Baier, J. A. (2016, Julio 7). *Redes Neuronales.* Retrieved from Ingeniería UC: http://jabaier.sitios.ing.uc.cl/iic2622/rna.pdf

Brownlee, J. (2017, Junio 23). *A Gentle Introduction to Backpropagation Through Time*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/gentle-introduction-backpropagation-time/

Brownlee, J. (2017, Enero 20). *A Gentle Introduction to the Random Walk for Times Series Forecasting with Python*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/gentle-introduction-random-walk-times-series-forecasting-python/

Chien, S. I.-J., Ding, Y., & Wei, C. (2002). Dynamic Bus Arrival Time Prediction with Artificial Neural Networks. *Journal of Transportation Engineering*, 429–438.

Chowdhury, N. K., & Leung, C. K. (2011). Improved Travel Time Prediction Algorithms for Intelligent Transportation Systems. *Knowlege-Based and Intelligent Information and Engineering Systems: 15th International Conference, KES 2011, Kaiserslautern, Germany, September 12-14, 2011, Proceedings, Part II*, 355-365.

Connor, J., Atlas, L. E., & Martin, D. R. (1991). Recurrent Networks and NARMA Modeling. *NIPS'91 Proceedings of the 4th International Conference on Neural Information Processing Systems*, 301-308.

D'Agostino, R. B. (1971). An omnibus test of normality for moderate and large sample size. *Biometrika 58*, 341-348.

de Brébisson, A., Simon, É., Auvolat, A., Vincent, P., & Bengio, Y. (2015). Artificial Neural Networks Applied to Taxi Destination Prediction. *eprint arXiv:1508.00021*. ARXIV.

Dirección de Transporte Público Metropolitano. (2017, Noviembre). *Conoce los recorridos*. Retrieved from Transtiago: https://www.transantiago.cl/mapas-y-recorridos/conoce-los-recorridos

Directorio de Transporte Público Metropolitano. (2018, Enero). *Indicadores de Operación*. Retrieved from DTPM: https://www.dtpm.cl/index.php/indicadores-calidad-de-servicio

Elman, J. L. (1990). Finding structure in time. *Cognitive Science 14*, 179-211.

Epysa Club. (2015). *Epysa Club*. Retrieved from Transantiago: gobierto no revisaría contratos y Subus se sumaría a operadores con problemas: http://www.epysaclub.cl/blog/index.php/noticias/762-transantiago-gobierno-no-revisaria-contratos-y-subus-se-sumaria-a-operadores-con-problemas

European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases. (2015). *DISCOVERY CHALLENGES*. Retrieved from ECMLPKDD: http://www.ecmlpkdd2015.org/discovery-challenges

Gschwender, A. (2016, Noviembre 30). Transformando Datos de GPS y Tarjeta Electrónica de Pago en Información Útil para la Planificación de Transporte Público: la Experiencia de Santiago de Chile. Santiago, Chile.

Gurmu, Z. K., & Fan, W. (2014). Artificial Neural Network Travel Time Prediction Model for Buses Using Only GPS Data. *Journal of Public Transportation*, 45-65.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-term Memory. *Neural computation 9*, 1735-1780.

Hopfield, J. (1985). Neural Computation of Decisions in Optimization Problems. *Biological Cybernetics 52*, 141-152.

Jeong, R., & Rilett, R. (2004). Bus arrival time prediction using artificial neural network model. *Proceedings of the 7th International IEEE Conference on Intelligent Transportation Systems*, 988-993.

Khosravi, A., Mazloumi, E., Nahavandi, S., Creighton, D., & Van Lint, J. (2011). A genetic algorithm-based method for improving quality of travel time prediction intervals. *Transportation Research*, 1364–1376.

Kingma, D. P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. *International Conference for Learning Representations.* San Diego.

Ministerio de Transportes y Telecomunicaciones. (2015, Marzo 19). *Ministerio de transportes y telecomunicaciones.* Retrieved from Presentamos resultados de la Encuesta Origen Destino de Santiago: http://www.mtt.gob.cl/archivos/10194

Moreira-Matias, L., Cats, O., Gama, J., & Mendes-Moreira, J. (2016, Junio 24). An online learning approach to eliminate Bus Bunching in real-time. *Applied Soft Computing 47*, 460–482.

Moreira-Matias, L., Ferreira, C., Gama, J., Mendes-Moreira, J., & Freire de Sousa, J. (2012). Bus Bunching Detection by Mining Sequences of Headway Deviations. *Springer*, 77-91.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning.* Determination Press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature 323*, 533–536.

Sak, H., Senior, A., & Beaufays, F. (2014, Febrero 5). Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. *arXiv:1402.1128*. ARXIV.

Sampedro, J. D. (2012). *Estudio y aplicación de técnicas de aprendizaje automático orientadas al ámbito médico: estimación y explicación de predicciones individuales.* Madrid: Universidad Autónoma de Madrid.

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 210 - 229.

Shapiro, S. S., & Wilk, M. B. (1965). An analysis of variance test for normality: Complete samples. *Biometrika 52*, 591–611.

Smirnov, N. (1948). Table for Estimating the Goodness of Fit of Empirical Distributions. *The Annals of Mathematical Statistics, 19*(2), 279-281.

Turnquist, M. A., & Bowman, L. A. (1980, Marzo). The effects of network structure on reliability of transit service. (sciencedirect, Ed.) *Transportation Research Part B: Methodological*, 79-86.

Wilamowski, B. M. (2009). Neural network architectures and learning algorithms. *IEEE Industrial Electronics Magazine vol. 3, no. 4*, 56-63.

Yu, H., Chen, D., Wu, Z., Ma, X., & Wang, Y. (2016). Headway-based bus bunching prediction using transit smart card data. *Transportation Research Part C 72*, 45-59.

Zuñiga, F. G. (2011). *Estimación Y Predicción De Matrices Dinámicas De Viaje Sobre Un Corredor De Transporte Público, Utilizando Datos Históricos E Información En Tiempo Real.* Santiago de Chile: Pontificia Universidad Católica De Chile.

ANEXO A: ESTRUCTURA DE ÍNDICES DE CUMPLIMIENTO DE SERVICIOS DE BUSES, MULTAS Y DESCUENTOS ASOCIADOS

Se establece en los contratos de cada operador, el cumplimiento ciertos índices relacionados con la frecuencia y regularidad del servicio. Estos índices son evaluados mes a mes, para cada operador, según los datos que arroja el Centro de Monitoreo de Buses (CMB), el cual monitorea en tiempo real el comportamiento de los buses en cada una de las rutas.

Específicamente, se establece una fórmula para el índice de cumplimiento de frecuencia (ICF) y tres fórmulas distintas para el índice de cumplimiento de regularidad (ICR), que miden el retraso producto de los incidentes en las expediciones (ICR-I), el exceso del tiempo de espera de los usuarios con respecto a lo planificado (ICR-E) y la puntualidad de las expediciones según lo planificado en el programa de operación (ICR-P). Específicamente, ICR-I mide el número de intervalos observados sin incidentes dividido por el número total de intervalos en un mes, ICR-E mide la diferencia entre uno y la suma de los tiempos de espera en exceso dividido en la suma de los tiempos programados en un mes, y ICR-P, mide el número de puntos de control con desfases iguales a 0, dividido por el total de punto de control analizado en un mes.

Según los índices anteriormente descritos, se establecen descuentos al ingreso que perciben los operadores. Específicamente, por cada minuto que posean los operadores por desfases, tiempos de espera en exceso e incidentes, se descuentan hasta 0,01 UF del ingreso mensual. Además, se establecen rangos de cumplimiento para cada índice, donde un bajo cumplimiento acarrea multas hacia el operador que la posea. A continuación, se presenta una tabla con los rangos de cumplimiento de los índices de regularidad antes descritos y luego una tabla con las distintas multas asociadas:

Tabla A.1. Rangos de cumplimientos para métricas de regularidad.

Métrica de cumplimiento	Cumplimiento alto	Cumplimiento medio	Cumplimiento bajo
ICR-I	$ICR - I > 90\%$	$80\% \leq ICR - I \leq 90\%$	$ICR - I < 80\%$
ICR-E	$ICR - E > 90\%$	$80\% \leq ICR - E \leq 90\%$	$ICR - E < 80\%$
ICR-P	$ICR - P > 95\%$	$80\% \leq ICR - P \leq 85\%$	$ICR - P < 85\%$

Fuente: Elaboración propia, 2017.

Tabla A.2. Rango de montos asociado a multas por incumplimiento de índices de cumplimiento de regularidad.

Tipo de multa	Monto (UF)
Cumplimiento bajo del indicador ICR-I, durante al menos 3 meses dentro de un periodo de 6 meses.	101 a 350
Cumplimiento bajo del indicador ICR-E, durante al menos 3 meses dentro de un periodo de 6 meses.	101 a 350
Cumplimiento bajo del indicador ICR-P, durante al menos 3 meses dentro de un periodo de 6 meses.	101 a 350
Descuento real mayor o igual a 10% por frecuencia y regularidad, 3 meses en un periodo de un año.	1.501 a 2.500

Fuente: Elaboración propia, 2017.