Gestión de Bandas de Frecuencias en Entornos Celulares mediante técnicas Predictivas de Deep Learning

Andrés Parra Guirado
Máster Universitario en Ingeniería de Telecomunicación UOC-URL
Telemática

Dr. José López Vicario
Dr. Xavier Vilajosana Guillén

Enero 2019

A la memoria de mi Madre y mi Padre.

AGRADECIMIENTOS:

Quiero gradecer a Mariola su gran apoyo durante estos dos años.

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Gestión de Bandas de Frecuencias en Entornos Celulares mediante técnicas Predictivas de Deep Learning*
Nombre del autor:	*Andrés Parra Guirado*
Nombre del consultor/a:	*José López Vicario*
Nombre del PRA:	*Xavier Vilajosana Guillén*
Fecha de entrega (mm/aaaa):	01/2019
Titulación::	*Máster Universitario en Ingeniería de Telecomunicación UOC-URL*
Área del Trabajo Final:	*Telemática*
Idioma del trabajo:	*Español*
Palabras clave	*Deep Learning, TensorFlow, LSTM*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

El trabajo aquí expuesto tiene como finalidad la implementación de un software que sea capaz de predecir las llamadas realizadas en redes celulares en la ciudad de Milán. Las fuentes de datos de tráfico son provistas por Italia Telecom en la ciudad de Milán generados por sus usuarios y por usuarios desplazados a la zona.

Esta implementación es posible con diferentes técnicas de Deep Learning y el framework TensorFlow. Para hacerlo con un horizonte de predicción suficiente que cumpla los requerimientos de algunas aplicaciones, una buena solución es hacer la implementación mediante una red LSTM, ya que estas tienen una gran memoria a largo plazo

En las pruebas realizadas, la red LSTM ha sido comparada con un perceptrón multicapa. Resultando como era de esperar, un rendimiento superior a favor de la red LSTM.

Una aplicación que se podría beneficiar del producto desarrollado en este trabajo sería la activación/desactivación de bandas de frecuencia en repetidores de salto de frecuencia cuando estos vayan a cursar un número de llamadas más elevado de lo habitual.

Como resultado, se ha conseguido hacer predicciones con un horizonte predicción a 18 muestras vista que, con el intervalo temporal de los datos usados, equivale a tres horas de predicción. Este es un horizonte suficiente para satisfacer la aplicación propuesta.

Abstract (in English, 250 words or less):

The purpose of this work is to implement a software capable of predicting calls made in cellular networks in the city of Milan. The sources of traffic data are provided by the city's Italia Telecom generated by its users as well as by users displaced to this area.

Implementation is possible with different Deep Learning techniques and the TensorFlow framework and, to achieve this with sufficent prediction horizon, use of an LSTM network is advisable, as these networks have large long-term Memory.

In the tests performed, the LSTM network was compared with a multilayer perceptron. As expected, LSTM network showed the best performance.

Based on the product developed in this work, the activation/deactivation of frequency bands in Frequency Shift Repeaters (FSR) could be improved when these are going to carry a higher number of calls than usual.

As a result, we have been able to make predictions with a prediction horizon of 18 samples, which with the time interval of the data used is equivalent to three hours of prediction. This is a sufficient horizon to satisfy the proposed application.

Índice de contenidos

1. Introducción 12
  1.1 Contexto y motivación 12
  1.2 Objetivos 13
    1.2.1 Objetivos Generales 13
    1.2.2 Objetivos Específicos 13
  1.3 Planificación del trabajo 13
  1.4 Aspectos legales de este trabajo 17
    1.4.1. Propiedad intelectual del autor 17
    1.4.2. Propiedad intelectual de terceros 17
    1.4.3. Elección de tipo de licencia. 17
    1.4.4. Base de datos utilizada. 17
2. Estado del arte 18
  2.1 Predicción de tráfico de red con técnicas de Deep Learning 19
  2.2 Machine Learning 22
    2.2.1 Tipos de aprendizaje en Machine Learning 22
    2.2.2 Algoritmos de clasificación, regresión y clustering 23
      2.2.2.1 KNN 23
      2.2.2.2 K-means 24
      2.2.2.3 Regresión Lineal 24
      2.2.2.4 Regresión Logística 25
  2.3 Deep Learning 25
    2.3.1 Redes neuronales Artificiales 26
      2.3.1.1 Perceptrón multicapa 27
        2.3.1.1.1 Función de activación 27
        2.3.1.1.2 Proceso de aprendizaje 29
        2.3.1.1.3 Sobreajuste (Overfitting) 31
      2.3.1.2 Redes neuronales convolucionales 32
      2.3.1.3 Redes neuronales recurrentes 33
        2.3.1.3.1 Red neuronal LSTM 34
  2.4 Software Utilizado 38
    2.4.1 PyCharm 38
    2.4.2 TensorFlow 39
    2.4.3 Pandas 39
    2.4.4 Scikit-learn 40
    2.4.5 Matplotlib 41
3. Descripción del trabajo 42
  3.1 Aspectos del problema a resolver 43
  3.2 Repetidores de Salto de Frecuencia (FSR) 45
    3.2.1 Gestión de bandas de frecuencias para FSR 46
4. Metodología e implementación 49
  4.1 Tratamiento y formato de datos 49
  4.2 Ventana deslizante 52
  4.3 División de los datos 52
  54
  4.4 Modelo LSTM implementado 54
5. Resultados 55
  5.1 Métricas 55

5.1.1 Error cuadrático medio 55
5.1.2 Error cuadrático medio relativo 56
6. Conclusiones 73
6.1 Conclusiones 73
6.1.1 Discusión sobre el trabajo 74
6.1.2 Lecciones aprendidas 74
6.1.3 Trabajo futuro 75
Bibliografía 76
Anexos 78
Anexo 1. Código implementado para la red LSTM 78
Anexo 2. Código implementado para el Perceptrón multicapa 82

Índice de tablas

Tabla 1. MSE para datos de test 21
Tabla 2. Ventana deslizante 52
Tabla 3. Ventana deslizante con horizonte de predicción a dos muestras vista 52
Tabla 4. Resultados obtenidos para todas las configuraciones de red probadas 57
Tabla 5. Mejores resultados obtenidos para todas las configuraciones de red probadas 58
Tabla 6. Resultados obtenidos para un horizonte de predicción de 1 muestra vista 58
Tabla 7. Resultados obtenidos para un horizonte de predicción de 4 muestras vista 60
Tabla 8. Resultados obtenidos para un horizonte de predicción de 6 muestras vista 62
Tabla 9. Resultados obtenidos para un horizonte de predicción de 12 muestras vista 64
Tabla 10. Resultados obtenidos para un horizonte de predicción de 18 muestras vista 67
Tabla 11. Resultados obtenidos para un horizonte de predicción de 1 muestra vista para toda el área de la ciudad de Milán 69
Tabla 12. Resultados obtenidos para un horizonte de predicción de 18 muestras vista para toda el área de la ciudad de Milán 70
Tabla 13. Configuraciones de red LSTM válidas para la aplicación de gestión de bandas de frecuencia. En verde la configuración escogida para la aplicación. 71
Tabla 14. Conmutación del canal 3 en NMS 71

Índice de figuras

Figura 1. Ventana deslizante [1] 20
Figura 2. MSE para un rango de nodos entre 100 y 700 [1] 20
Figura 3. MSE para mejor estructura de red entre 1 y 6 capas 21
Figura 4. Resultado de la ejecución del algoritmo K-means 24
Figura 5. Ejemplo de Regresión Lineal [17] 25
Figura 6. Deep Learning en el Paradigma de la Inteligencia Artificial [17] 26
Figura 7. Arquitectura básica del perceptrón multicapa [1] 27
Figura 8. Representación gráfica de la función sigmoide [16] 28
Figura 9. Operación de convolución [18] 32
Figura 10. Ejemplo de max-pooling y average-pooling 33
Figura 11. Celda básica de una red neuronal recurrente [12] 33
Figura 12. Desglose de figura 8 [12] 34
Figura 13. Bloque de memoria LSTM con celda de estado resaltada [12] 35
Figura 14. Bloque de memoria LSTM donde se resalta la puerta de olvido [12] 36
Figura 15. Bloque de memoria LSTM con la puerta de entrada y la capa con la función hiperbólica ambas resaltadas [12] 36
Figura 16. Bloque de memoria LSTM donde se resaltan los elementos actualizados por la celda de estado [12] 37
Figura 17. Bloque de memoria LSTM donde se resalta la última etapa del bloque [12] 37
Figura 18. ©PyCharm 38
Figura 19. ©TensorFlow 39
Figura 20. ©Scikit-learn 41
Figura 21. Diagrama de bloques del sistema completo 43
Figura 22. Grid correspondiente a la ciudad de Milán [15] 43
Figura 23. Mapa de cobertura de la ciudad de Milán [15] 44
Figura 24. Repetidor de salto de frecuencia. [Fuente, elaboración propia] 45
Figura 25. Bandas de operación para casos de vano simple y de vano doble. 45
Figura 26. Canales de 5 MHz en FSR, el tercer canal está apagado 46
Figura 27. Pantalla inicial de NSM 46
Figura 28. Interfaz para la activación/desactivación de canales 47
Figura 29. Diagrama de bloques de NMS con el sistema de predicción 48
Figura 30. Grid correspondiente a la ciudad de Milán [15] 49
Figura 31. Archivo de datos con formato texto 50
Figura 32. Filas de datos en el mismo intervalo temporal. 50
Figura 33. Nueva extensión de la celda 1 50
Figura 34. División de datos realizada 53
Figura 35. Tratamiento realizado a los datos 54
Figura 36. Stack LSTM 54
Figura 37. Evolución del MSE para la red LSTM correspondiente a la prueba nº 1 58

Figura 38. Predicción llamadas salientes para la red LSTM correspondiente a la prueba nº 1 59
Figura 39. Evolución del MSE para el Perceptrón correspondiente a la prueba nº 36 59
Figura 40. Predicción llamadas salientes para el Perceptrón correspondiente a la prueba nº 36 60
Figura 41. Evolución del MSE para la red LSTM correspondiente a la prueba nº 3 60
Figura 42. Predicción llamadas salientes para la red LSTM correspondiente a la prueba nº 3 61
Figura 43. Evolución del MSE para el Perceptrón correspondiente a la prueba nº 38 61
Figura 44. Predicción llamadas salientes para el Perceptrón correspondiente a la prueba nº 38 62
Figura 45. Evolución del MSE para la red LSTM correspondiente a la prueba nº 8 63
Figura 46. Predicción llamadas salientes para la red LSTM correspondiente a la prueba nº 8 63
Figura 47. Evolución del MSE para el Perceptrón correspondiente a la prueba nº 40 64
Figura 48. Predicción llamadas salientes para el Perceptrón correspondiente a la prueba nº 40 64
Figura 49. Evolución del MSE para la red LSTM correspondiente a la prueba nº 29 65
Figura 50. Predicción llamadas salientes para la red LSTM correspondiente a la prueba nº 29 65
Figura 51. Predicción llamadas salientes para el Perceptrón correspondiente a la prueba nº 42 66
Figura 52. Predicción llamadas salientes para el Perceptrón correspondiente a la prueba nº 42 66
Figura 53. Evolución del MSE para la red LSTM correspondiente a la prueba nº 34 67
Figura 54. Predicción llamadas salientes para la red LSTM correspondiente .. 67
Figura 55. Predicción llamadas salientes para el Perceptrón correspondiente a la prueba nº 44 68
Figura 56. Predicción llamadas salientes para el Perceptrón correspondiente a la prueba nº 44 68
Figura 57. Evolución del MSE para la red LSTM correspondiente a todas las celdas de la 69
Figura 58. Predicción llamadas salientes para la red LSTM correspondiente a todas las celdas de la ciudad de Milán: 1 muestra vista 70
Figura 59. Evolución del MSE para la red LSTM correspondiente a todas las celdas 70

1. Introducción

1.1 Contexto y motivación

La inteligencia artificial se ha abierto un gran espacio dentro de nuestras vidas. En concreto, la disciplina de Deep Learning ha mejorado en gran medida muchas de las aplicaciones con las que trabajamos habitualmente, como reconocimiento de imágenes, texto predictivo o traductores, entre otras.

Con el paso del tiempo y el avance de las tecnologías, los humanos estamos más conectados entre nosotros, ya sea a través de una comunicación voz, de datos o a través de redes. Es más, ya empezamos a estar conectados con diversos dispositivos para controlar o monitorizar diversas aplicaciones. Con la llegada de la próxima generación de redes celulares 5G, se espera una bajada en la latencia de los datos, y con ello, aparecerán numerosas aplicaciones a las que estaremos conectados.

En este contexto, se denota un incremento de tráfico generado, provocando este una cantidad ingente de tráfico cursada a través de las redes que lo sustentan. Siendo de gran importancia que estas redes de telecomunicaciones cuenten con los recursos suficientes, de forma que no se descarten llamadas de voz o datos y se consiga completar el máximo número posible de comunicaciones con éxito, ya que por ejemplo, una llamada de voz en un momento dado, puede ser de suma importancia para el emisor o bien para el receptor de la misma. En cualquier caso, cada intento de comunicación es importante dentro de su contexto.

De cara a una mejora en la gestión de los recursos en redes celulares para incrementar el número de llamadas cursadas, se pueden emplear técnicas de inteligencia artificial para predecir el tráfico que fluye por una red en cuestión. Con un horizonte de predicción suficiente, un operador de telecomunicaciones puede saber con antelación el comportamiento de una red supervisada, de esta forma puede añadir o quitar recursos a la misma.

Un recurso importante en las redes celulares son las bandas de frecuencias, ya que la capacidad y número de estas son un factor importante en la cantidad de llamadas cursadas con éxito.

Existen zonas rurales de escasa población donde los operadores de telecomunicaciones no obtienen una rentabilidad suficiente si implementan una estación base (en inglés Base Station, BTS), ya que la población no tendrá los usuarios necesarios para usar toda la capacidad de la misma. Una solución económica para dar cobertura a estas regiones es la implementación de un Repetidor de Salto de Frecuencia (FSR).

Estos repetidores tienen activos un número de canales de transmisión acordes con su población, pero en algunas épocas la población crece un

porcentaje importante. No habiendo en este caso, recursos suficientes para cursar las llamadas generadas debido a este incremento, y por tanto, un gran número de llamadas serán descartadas, cada una de ellas con su respectiva importancia dentro de su ámbito.

La motivación de este trabajo es implementar un sistema de predicción con un horizonte de predicción suficiente para gestionar las bandas de frecuencias de forma automática en repetidores de telefonía móvil. De esta forma, se podrá activar un canal adicional de forma automática para cubrir el incremento de llamadas generadas.

1.2 Objetivos

1.2.1 Objetivos Generales

1. Implementar un modelo capaz de predecir el número de llamadas salientes en una región de la ciudad de Milán.

2. Capacidad de elección sobre el tipo de red neural adecuado para la predicción de series temporales.

1.2.2 Objetivos Específicos

1. Capacidad para tratar y extraer información de datos estructurados.

2. Capacidad para resolver un problema de predicción de una serie temporal a través de técnicas de Deep Learning.

3. Capacidad para implementar un sistema con un horizonte de predicción amplio, aplicable a la gestión de bandas de frecuencias en entornos celulares.

4. Capacidad de interpretación sobre los resultados obtenidos detectando si los datos de entrenamiento provocan overfitting.

1.3 Planificación del trabajo

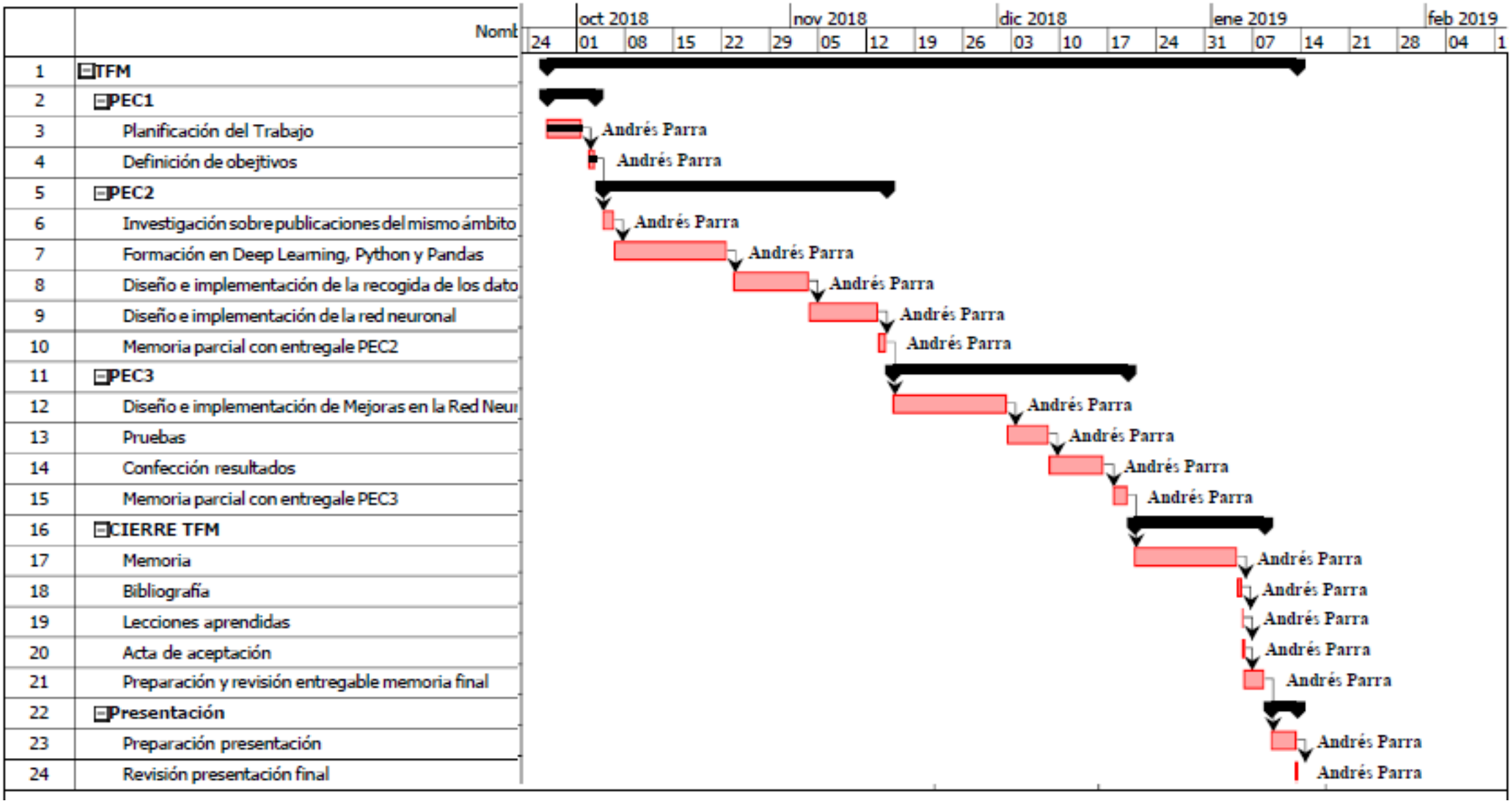
La planificación del trabajo seguida se muestra en las siguientes páginas. Se ha tratado de seguir de la forma más exacta en la medida de lo posible, por lo que no ha habido desviaciones en la misma.

Un resumen de la planificación con el costo en horas del trabajo realizado queda reflejado en la siguiente tabla:

PLANIFICACION PORCENTUAL TFM			
PEC	Costo en Horas	Porcentaje trabajo	Porcentaje PEC2/PEC3 de trabajo técnico
PEC1	12	4,00%	
PEC2	131	43,67%	60,09%
PEC3	87	29,00%	39,91%
Cierre TFM	48	16,00%	
Presentación	22	7,33%	
Total Horas	300		

A continuación, se muestran los hitos del trabajo, después se muestran los hitos junto con el diagrama de Gantt. Se ha usado para ello el software ProjectLibre.

	Nombre	Inicio	Terminado	Trabajo
1	**TFM**	**26/09/18 19:00**	**13/01/19 11:00**	**300 horas**
2	**PEC1**	**26/09/18 19:00**	**3/10/18 20:00**	**12 horas**
3	Planificación del Trabajo	26/09/18 19:00	1/10/18 20:00	10 horas
4	Definición de obejtivos	2/10/18 19:00	3/10/18 20:00	2 horas
5	**PEC2**	**4/10/18 19:00**	**14/11/18 21:00**	**131 horas**
6	Investigación sobre publicaciones del mismo ámbito	4/10/18 19:00	6/10/18 12:00	6 horas
7	Formación en Deep Learning, Python y Pandas	6/10/18 12:00	22/10/18 21:00	43 horas
8	Diseño e implementación de la recogida de los datos de entrenamiento y de test	23/10/18 19:00	3/11/18 17:00	37 horas
9	Diseño e implementación de la red neuronal	3/11/18 17:00	13/11/18 20:00	39 horas
10	Memoria parcial con entregale PEC2	13/11/18 20:00	14/11/18 21:00	3 horas
11	**PEC3**	**15/11/18 19:00**	**19/12/18 21:00**	**87 horas**
12	Diseño e implementación de Mejoras en la Red Neural	15/11/18 19:00	2/12/18 11:00	38 horas
13	Pruebas	2/12/18 11:00	8/12/18 10:00	15 horas
14	Confección resultados	8/12/18 10:00	16/12/18 11:00	25 horas
15	Memoria parcial con entregale PEC3	17/12/18 19:00	19/12/18 21:00	6 horas
16	**CIERRE TFM**	**20/12/18 19:00**	**8/01/19 21:00**	**48 horas**
17	Memoria	20/12/18 19:00	4/01/19 19:00	32 horas
18	Bibliografía	4/01/19 19:00	5/01/19 11:00	4 horas
19	Lecciones aprendidas	5/01/19 11:00	5/01/19 17:00	3 horas
20	Acta de aceptación	5/01/19 17:00	5/01/19 20:00	3 horas
21	Preparación y revisión entregable memoria final	5/01/19 20:00	8/01/19 21:00	3 horas
22	**Presentación**	**9/01/19 19:00**	**13/01/19 11:00**	**22 horas**
23	Preparación presentación	9/01/19 19:00	13/01/19 10:00	6 horas
24	Revisión presentación final	13/01/19 10:00	13/01/19 11:00	1 hora


Nomb
oct 2018
nov 2018
dic 2018
ene 2019
feb 2019
24 01 08 15 22 29 05 12 19 26 03 10 17 24 31 07 14 21 28 04 1
1 TFM
2 PEC1
3 Planificación del Trabajo
4 Definición de obejtivos
5 PEC2
6 Investigación sobre publicaciones del mismo ámbito
7 Formación en Deep Learning, Python y Pandas
8 Diseño e implementación de la recogida de los dato
9 Diseño e implementación de la red neuronal
10 Memoria parcial con entregale PEC2
11 PEC3
12 Diseño e implementación de Mejoras en la Red Neu
13 Pruebas
14 Confección resultados
15 Memoria parcial con entregale PEC3
16 CIERRE TFM
17 Memoria
18 Bibliografía
19 Lecciones aprendidas
20 Acta de aceptación
21 Preparación y revisión entregable memoria final
22 Presentación
23 Preparación presentación
24 Revisión presentación final
Andrés Parra

Como se comentaba anteriormente, se ha seguido con rigor los tiempos marcados para cada tarea e hito en la planificación propuesta al inicio, por lo que no ha habido desvíos.

Después de ejecutar este trabajo y tener más conocimiento sobre el mismo, se ha llegado a la conclusión de que la tarea nº 20 en concepto de Acta de aceptación no aplica para este tipo de trabajo, por lo que no se ha realizado. Lo cual no ha provocado ningún desvío. Por lo tanto, **toda la planificación propuesta se ha llevado a cabo, exceptuando la tarea nº 20.**

1.4 Aspectos legales de este trabajo

En este apartado, como su nombre indica, se pretende mostrar los diferentes aspectos legales a tener en cuenta en este trabajo.

1.4.1. Propiedad intelectual del autor.

La propiedad intelectual pertenece a autor del trabajo, en este caso, el autor del trabajo es el estudiante, siempre y cuando las contribuciones pertenezcan al mismo. En caso de que se incluyan contribuciones de otros autores, la propiedad intelectual seguirá perteneciendo a estos terceros.

1.4.2. Propiedad intelectual de terceros.

Los recursos utilizados de terceros deben ser citados explícitamente, así como respetar el tipo de licencia para su uso y distribución.

1.4.3. Elección de tipo de licencia.

El autor del trabajo debe elegir el tipo de licencia con la que quiere depositar su trabajo en la biblioteca de la universidad y decidir los derechos los derechos que se reserva. El tipo de licencia para este trabajo es GNU.

1.4.4. Base de datos utilizada.

La base de datos para entrenar la red neuronal corresponde a una Licencia Abierta de Bases de Datos (en inglés Open Database License (ODbL)). La cual indica la forma en que debe hacerse referencia a la base de datos. En caso de que se modifique la base de datos, la nueva adaptación también debe ser libre bajo el mismo tipo de licencia.

2. Estado del arte

En las dos últimas décadas, la telefonía celular, internet y sus aplicaciones se han convertido en una herramienta de comunicación, de ocio y de trabajo esencial para un gran número de usuarios. Realizar actividades cotidianas, como llamar a un familiar cercano, llamar al CIO de una empresa o subir archivos a un gestor documentar cuando se está en el trabajo, pueden generar un gran volumen de tráfico en una red celular.

Una Matriz de tráfico proporciona una representación abstracta del volumen de flujo de tráfico existente en una red, desde cualquier nodo de origen posible hasta otro nodo de destino para un intervalo de tiempo definido [1]. Comprender la matriz de tráfico es esencial para proveedores de servicios de red, principalmente por el gran diverso número de elementos que componen la red. Esta proporciona información de gran valor a los operadores para tomar importantes decisiones de cara a la gestión de red, tales como el mantenimiento, optimización, diseño, balanceo de carga, diseño de protocolos, detección de anomalías y predicción del tráfico de la red. Para abordar el problema de predicción de los parámetros de una red, se han empleado diferentes enfoques.

La mayoría de los métodos para obtener una matriz tráfico están basados en el método denominado “Tomogravity method” [1]. Este método es una combinación de dos procesos, son los denominados como “Gravity method” y “Tomographic method” [2]. “Tomogravity method” aporta el conocimiento del número de enlaces y la información de la ruta. Idealmente, los parámetros de predicción de una matriz de tráfico son basados en sus características estadísticas, principalmente porque existe una gran relación entre el orden de la secuencia de los valores. Estas características estadísticas han cambiado mucho debido a la diversidad de arquitectura de la red actual y sus aplicaciones. De ahí que ahora estas características no sean apropiadas para el tráfico de la red actual, ya que estas no son modeladas adecuadamente por los modelos Gaussianos y de Poisson [1].

Existen varios modelos para la predicción del tráfico que circula por la red. Estos pueden ser clasificados en función de su linealidad o no linealidad. Dentro de los modelos lineales más utilizados se encuentran tales como ARMA/ARIMA y el algoritmo de HoltWinters.

Estos modelos han sido experimentalmente comparados, y sus resultados denotan un uso más apropiado de los modelos no lineales para este ámbito.

Para los modelos no lineales, lo más extendido es el uso de redes neuronales [1]. De forma general, el enfoque para la predicción de tráfico está basado en la consideración de varios factores, tales como la reducción del error cuadrático medio y el coste computacional. Las FNN con predictor de aprendizaje con multiresolución son consideradas como

la mejor técnica para predecir tráfico de redes teniendo en cuenta la precisión y complejidad del modelo [3].

En la actualidad, las Redes Neuronales Recurrentes (RNN) son un modelo mejorado de las tradicionales FFNs. Las RNNs tienen un lazo recurrente que facilitan el traspaso de información de un "time step" a otro. Esta característica de las RNNs encaja más con datos formados por series temporales. Aunque estas tienen problemas cuando se trata de almacenar información a largo plazo.

Las redes LSTM son un tipo de red neuronal recurrente que fueron propuestas para evitar el problema de desvanecimiento y explosión del gradiente que acontece a las RNNs. Con el propósito de reducir el coste computacional provocado por las RNNs, se introdujo la Gate Recurrent Units (GRU). Aunque estas no consiguen el mismo rendimiento que la red neuronal recurrente LSTM [1].

2.1 Predicción de tráfico de red con técnicas de Deep Learning

El artículo "Applying Deep Learning Approaches for Network Traffic Prediction" [1], publicado en el año 2017, está basado en la predicción de tráfico en redes celulares mediante técnicas de Deep Learning. Para llevar a cabo este experimento se hace uso de redes neuronales recurrentes RNN y LSTM entrenadas haciendo uso de la retropropagación a través del tiempo (BPTT) y de una unidad de procesamiento de gráficos (GPU) activada en TensorFlow.

En este experimento, como dataset se han utilizado los datos públicos de la red backbone GÉANT, esto tienen intervalos de 15 minutos. De 10772 matrices de tráfico disponibles, se usan arbitrariamente 1200. Cada una de estas matrices de tráfico TM es transformada en un vector 23x23 = 529 TMV. Estos vectores son concatenados formando una nueva matriz de tráfico TM con un tamaño 1200x529. La matriz es aleatoriamente dividida en dos matrices, 900x529 y 300x329, para entrenamiento y test respectivamente.

Existen varias formas para predecir tráfico. Una de ellas se trata de alimentar los vectores TMV en TM y este, a los algoritmos de predicción para predecir un valor de TMV en el momento. Este método no fue correcto debido al hecho de que el origen y destino de la matriz de tráfico depende de los orígenes y destinos. De ahí que optara por el enfoque de escoger tráfico previo, ya que este puede mejorar el rendimiento de predicción sobre la matriz de tráfico.

Para crear los datos de entrenamiento se utilizó una ventana deslizante, tal y como se muestra en la figura 1. Donde el conjunto de datos en color rojo son los datos usados para predecir el siguiente dato, según la figura, el dato X_{sw+1}, X_{sw+2} ... etc.

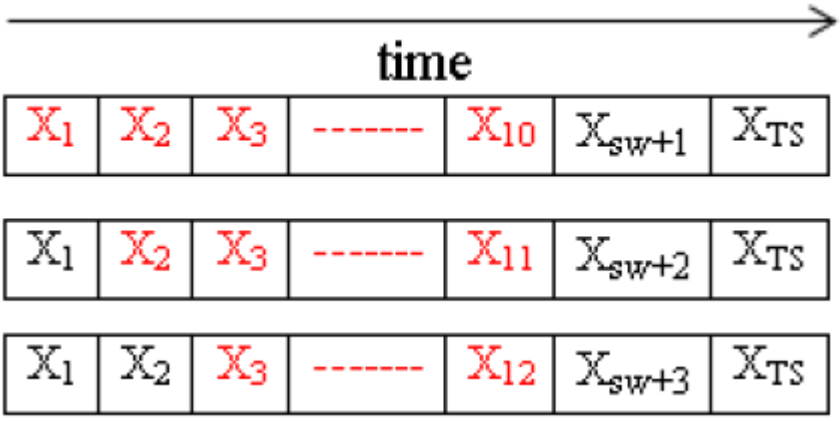


Figura 1. Ventana deslizante [1]

Los datos de entrenamiento son divididos dos partes, una con tamaño 900x529 para entrenamiento y otra de 200x529 para validación. En este experimento se hicieron todas las pruebas para redes neuronales, tales como FFN, RNN, LSTM, GRU e IRNN con un número de nodos, de capas ocultas y learning rate. Los experimentos se hicieron con 100, 200, 300, 400, 500, 600 y 700 nodos para una sola capa. Además, todos los experimentos se hicieron con 200 épocas. En la figura 2 se puede ver la evaluación de los modelos con los datos de evaluación. El experimento realizado con 600 nodos tiene un buen rendimiento en FFN, RNN, LSTM, GRU e IRNN. Como resultado se manifiesta que las redes recurrentes tienen un mejor rendimiento que una red FNN. Además, la red LSTM ha tenido un mejor rendimiento que la red RNN. Se puede ver que el rendimiento de GRU e IRNN es comparable al rendimiento de las redes LSTM.

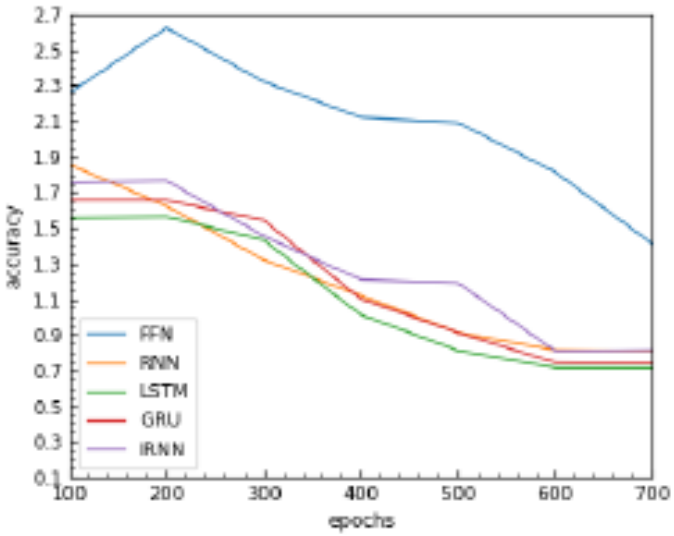


Figura 2. MSE para un rango de nodos entre 100 y 700 [1]

De igual forma se hicieron experimentos variando el learning rate en un rango de valores desde 0.01 hasta 5 para los mismos tipos de redes en cuestión. En la mayoría de los casos las redes LSTM han funcionado bien en comparación con las RNN. Al igual que ocurría al comparar el número de nodos, el rendimiento de GRU e IRNN es comparable al rendimiento de las redes LSTM. Se hizo un nuevo experimento para para todos los tipos de redes en cuestión para estructura una de red de 1 a 6 capas, con 500 nodos, un learning rate de 0.1 y 100 épocas. En la figura 3 se muestra el mejor rendimiento de cada topología de red. Como en los anteriores experimentos, las redes recurrentes han tenido un mejor rendimiento que las FNN. De la misma manera, las redes LSTM han tenido mejor funcionamiento que las redes RNN.

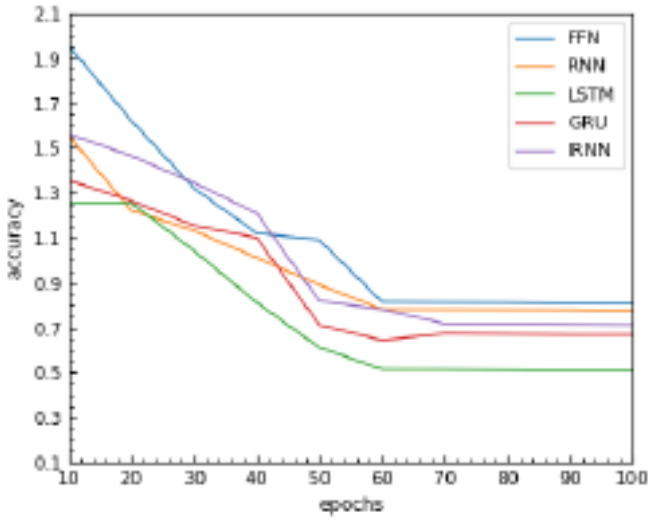


Figura 3. MSE para mejor estructura de red entre 1 y 6 capas

Finalmente, una vez validadas las diferentes redes, se decidió usar como hyperparámetros, 5 capas con 500 nodos y un learning rate de 0.1 para todas las redes, pero ahora usando los datos de test para hacer el experimento. En la tabla 1 se pueden ver los resultados, obteniendo un mejor rendimiento la red LSTM ante las demás.

Red	MSE
LSTM	0.042
GRU	0.051
IRNN	0.059
RNN	0.067
FNN	0.091

Tabla 1. MSE para datos de test

De este experimento podemos observar que los resultados en cuanto a rendimiento son buenos, aunque solamente se predice el siguiente intervalo temporal, correspondiente a solamente a 15 minutos de predicción, quedando limitada su aplicación, por lo que sería deseable y se echa de menos predecir varios intervalos temporales vista.

De ahí que una de las motivaciones de este trabajo sea la implementación de un sistema de predicción con un horizonte de predicción más amplio, lo cual se considera una contribución relevante en esta disciplina.

2.2 Machine Learning

La disciplina de Machine Learning (o aprendizaje automático) ha mostrado una gran expansión gracias al desarrollo conjunto de tecnologías, estamos hablando de la informática, big data e Internet. La mayoría de los dispositivos electrónicos que las personas usamos de forma rutinaria están, directa o indirectamente, influenciadas por el Machine Learning. Hay muchos ejemplos que usan esta disciplina, como el reconocimiento facial, segmentación de la audiencia, reconocimiento de la voz, la clasificación de imágenes en nuestros teléfonos, así como la detección de correo basura en nuestro buzón de e-mail han permitido desarrollos de apps que, muy pocas décadas atrás, solo veríamos en filmografías de ciencia ficción. El uso del aprendizaje en modelos bursátiles, investigación biomédica, o bolsas de trabajo ha impactado a la sociedad de forma masiva. A todos estos ejemplos hay que añadirles los primeros coches sin conductor, drones o robots de todo tipo que ya empiezan a impactar a la sociedad por su capacidad de realizar labores que nosotros mismos llevamos a cabo de forma habitual [16].

Con la anterior introducción el lector ya puede haber discernido ligeramente que es el Machine Learning. Una definición más concisa sería; Machine Learning es el subcampo de la Inteligencia Artificial que busca resolver cómo implementar código informático que sea capaz de mejorar automáticamente adquiriendo experiencia. Es decir, Machine Learning usa herramientas informáticas que utilizan la experiencia pasada para tomar decisiones futuras.

2.2.1 Tipos de aprendizaje en Machine Learning

Hay tres tipos de aprendizaje en Machine Learning. Aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

Los problemas de aprendizaje supervisado utilizan datos etiquetados con un valor correcto, el algoritmo de Machine Learning se entrena a partir de estos datos. Un mayor número de datos favorecerá como el algoritmo mejora su aprendizaje sobre un problema en cuestión. Una vez finalizado el entrenamiento, se usan nuevos datos, estos ya sin las etiquetas de los valores correctos, de esta forma es fácil evaluar si el algoritmo es capaz de generar el valor adecuado con datos para los cuales no ha sido

entrenado. El algoritmo ha utilizado la experiencia pasada adquirida durante el entrenamiento para predecir un resultado [17].

En los problemas de aprendizaje no supervisado, como el lector podrá prever, el algoritmo no se entrena con datos etiquetados. Se trata de que el algoritmo sea capaz de encontrar por si solo patrones que ayuden a comprender al conjunto de datos.

El último tipo son problemas de aprendizaje por refuerzo. En este caso el algoritmo aprende observando su horizonte visible. Para hacerlo, este usa como información de entrada el feedback que obtiene el mundo exterior en respuesta a sus acciones.

2.2.2 Algoritmos de clasificación, regresión y clustering

En este subapartado vamos a ver brevemente los algoritmos más utilizados en Machine Learning, estos pueden agrupase según el tipo de salida del problema a resolver. Hay tres tipos: clasificación, regresión y clusterización.

Se utilizan algoritmos de clasificación cuando el resultado del problema está dentro de un conjunto finito de todos los resultados posibles. Dentro los algoritmos de clasificación de aprendizaje supervisado los algoritmos más empleados son: KNN, Arboles de Decisión, Random Forest y SVM.

Cuando la solución del problema a resolver es un valor que puede determinarse de forma flexible y en función de los inputs del modelo, esto es, predecir valores continuos, se usan algoritmos de regresión. Los algoritmos de regresión utilizan aprendizaje supervisado. Dentro de los algoritmos de regresión más utilizados se encuentran: Regresión Lineal, Regresión Logística y Mínimos cuadrados (OLS).

Para resolver problemas de clasificación de aprendizaje no supervisado se usan algoritmos de clusterización, el algoritmo más empleado es K-means.

Por otra parte, hay otro tipo de técnicas para resolver problemas de clasificación y regresión. Estos problemas también pueden ser resueltos con redes neuronales artificiales, que de alguna forma tratan de copiar la forma de aprender que tenemos los humanos, lo que ha despertado un gran interés en la comunidad científica. Este tipo de redes forma parte de un subcampo del Machine Learning, denominado Deep Learning. Hablaremos con más detalle de este tipo de redes en los siguientes apartados, pero primero vamos a comentar algunos de los algoritmos de Machine Learning nombrados anteriormente.

2.2.2.1 KNN

KNN es un algoritmo de clasificación supervisado, este estima la probabilidad una vez que se tiene el conocimiento de que un elemento

pertenezca a una clase a partir de la información proporcionada por un conjunto de prototipos. El algoritmo proporciona un promedio del punto K más cercano de un punto en cuestión [17].

2.2.2.2 K-means

Hay ocasiones en las que no disponemos de etiquetas para analizar un conjunto de datos. En este caso hay que recurrir a un algoritmo de aprendizaje no supervisado, siendo una buena opción un algoritmo de clusterización. Dentro de estos métodos, el más popular es el algoritmo K-means.

El objetivo de este algoritmo es la división de un conjunto observaciones en K clúster, perteneciendo cada observación al punto central (centroide) de cada grupo que este más cercano a la misma, es decir, todos los puntos pertenecientes a un mismo clúster deben estar más cerca de su centroide que al resto de centroides. También se obtiene un etiquetado del conjunto de puntos, que los asigna a solamente uno de los K clúster [16]. El la figura 4 se muestra un ejemplo del resultado al aplicar este algoritmo a un conjunto de datos dado.

Figura 4. Resultado de la ejecución del algoritmo K-means

2.2.2.3 Regresión Lineal

La regresión lineal forma parte de los algoritmos de aprendizaje supervisado. Es una técnica estadística muy popular que es utilizada para medir la relación de dependencia entre variables. Su popularidad radica

en que usa un modelo para su implementación con una baja complejidad [17]. El modelo puede ser expresado como expresa la ecuación 1:

$$Y_t = \beta_0 + \beta_1 X + \beta_2 X + \cdots + \beta_n X_n + \varepsilon \qquad (1)$$

donde Y_t es la variable dependiente, X_n es la variable independiente, $\beta_n$es un parámetro que mide la influencia que tiene la variable independiente sobre la variable dependiente. La figura 5 muestra el resultado de una Regresión Lineal con variable independiente X y variable dependiente Y.

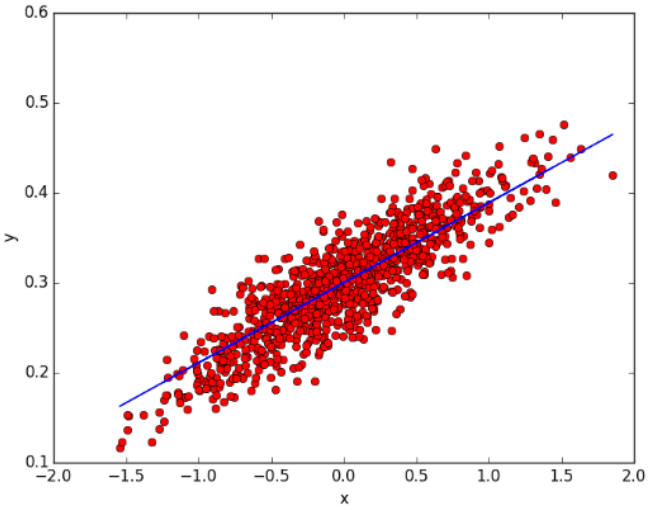


Figura 5. Ejemplo de Regresión Lineal [17]

2.2.2.4 Regresión Logística

Para problemas de clasificación, también pueden ser usados modelos lineales, esto es, se ajusta el modelo lineal a la probabilidad de que una cierta clase ocurra, hecho esto, se necesita crear un umbral para especificar el resultado de una de estas clases. Este umbral se crea a través de una función logística mostrada en la ecuación 2 [17]. Para detectar el mejor modelo en una Regresión Logística, se usa la comparación de varios modelos haciendo uso del cociente de verosimilitud, este indica a partir de los datos que modelo es el más probable.

.

$$f(x) = \frac{1}{1+e^{-1}} \qquad (2)$$

2.3 Deep Learning

Como se ha comentado anteriormente, Deep Learning es un subcampo dentro de Machine Learning y es sin duda el campo de la inteligencia artificial que está teniendo más auge en los últimos años, desde que los desarrolladores de grandes compañías tecnológicas contaran ya con la tecnología y recursos necesarios para llevarla a cabo, ya que esta área

era conocida décadas atrás, pero los ordenadores no disponían de los suficientes recursos para operar con ella. Y fue en 2006 cuando un gran número de las startups en Silicon Valley se especializan en ello, gigantes tecnológicos como Google y Facebook cuentan con equipos de I+D para el avance de esta área [16]. En la figura 6 podemos ver la situación del Deep Learning en el contexto de la inteligencia artificial.

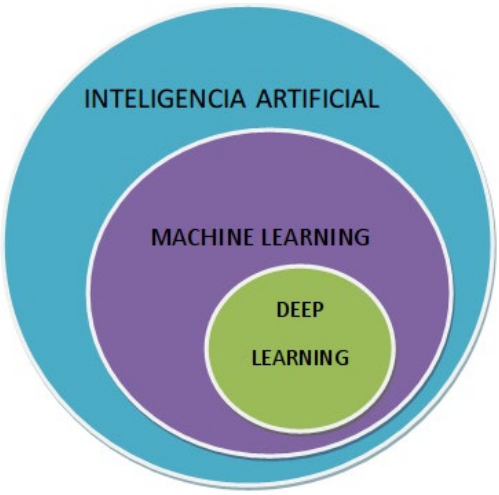


Figura 6. Deep Learning en el Paradigma de la Inteligencia Artificial [17]

Deep Learning intenta hacer una aproximación a la percepción humana utilizando estructuras de redes neuronales artificiales. Con estas estructuras se consigue el aprendizaje de capas apiladas, cada vez con más detalle de algunas características de los datos de entrada. Con cada nivel de capa que se añade a la red neuronal, se consigue más profundidad en el aprendizaje, de ahí el nombre de Deep Learning [17].

2.3.1 Redes neuronales Artificiales

Las redes neuronales artificiales (Artificial neural networks, ANN) son modelos simples que emulan el funcionamiento del sistema nervioso humano en cuanto al procesamiento de información se refiere. Estas están formadas por unidades simples llamadas neuronas. Por norma general estas neuronas se agrupan en capas, estando la red neuronal más básica formada por una sola capa.

A continuación, hablaremos de los aspectos teóricos en los que se basan las redes neuronales artificiales. En primer lugar, se mostrará el modelo del perceptrón multicapa, este modelo es la base de las arquitecturas más habituales: las redes neuronales convolucionales y las neuronales recurrentes.

2.3.1.1 Perceptrón multicapa

El perceptrón multicapa (Multilayer Perceptron, MLP), es el modelo de aprendizaje profundo más básico. Dicho modelo tiene como objetivo la mejor aproximación del vector de pesos de una función cualquiera según unas entradas dadas. Por ejemplo, en un clasificador, una aproximación habitual es a la línea que discrimine dos grupos. En este tipo de arquitectura la información fluye desde la entrada hasta la salida, de ahí que esta arquitectura también reciba el nombre de red feedfordward.

Este modelo dispone de una arquitectura donde las unidades de procesamiento se organizan en tres capas totalmente conectadas: la capa entrada, la capa oculta (o capas ocultas, pueden ser varias) y la capa de salida. En la figura 7 podemos ver la conexión entre las diferentes capas.

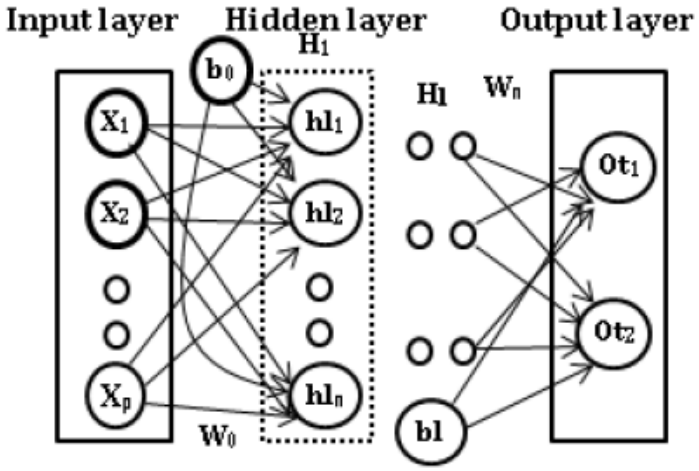


Figura 7. Arquitectura básica del perceptrón multicapa [1]

- Capa de entrada: Esta capa es en realidad un vector formado por los datos de entrada, donde cada elemento de este vector se corresponde con una neurona de entrada. En esta capa no se produce procesamiento, solo se encarga de propagar todos los datos de entrada a la siguiente capa.

- Capa oculta: En esta capa es donde se produce el procesamiento no lineal de los patrones de datos de la capa de entrada. Lo habitual es que se apilen varias capas ocultas, cuanto mayor es el número de las mismas, mayor profundidad se alcanza en el aprendizaje.

- Capa de salida: Esta capa tiene como objetivo proporcionar al exterior la respuesta de la red para cada uno de los patrones de entrada.

2.3.1.1.1 Función de activación

En la figura 7 podemos apreciar las operaciones que tienen lugar entre las capas del modelo. Una capa de entrada con varias neuronas $x_1, x_2 \ldots$, una

capa oculta con varias neuronas h_1, h_2 … y una capa de salida con dos neuronas Ot_1, Ot_2. Se pueden simplificar las neuronas de la capa oculta por una matriz W y un sesgo b. Así de esta forma, se puede definir el mapeo de las neuronas de entrada con las neuronas de la capa oculta como

$$z(x) = b + \sum_i Wx \qquad (1)$$

donde W es una matriz de pesos y b es el vector bias.

Con los parámetros W y b se calcula la suma ponderada, entonces es necesario una función que aplique una transformación para que la salida del resultado almacenado en z se convierta en 0 o 1. Esta función es la **función de activación** y se usa para propagar hacia delante la salida de una neurona, esta salida la reciben las neuronas de la siguiente capa a las que está conectada esta neurona. Hay varias funciones de activación, cada tipo de función se ajusta más en función del tipo de problema abordado. En este trabajo solo vamos a comentar una que goza de gran popularidad, se trata de la función sigmoide. Esta retorna un valor real de salida entre 0 y 1. Dicha función se expresa en la ecuación (2).

$$y(z) = \frac{1}{1+e^{-z}} \qquad (2)$$

Si analizamos la fórmula, podemos ver que siempre tiende a dar valores cercanos a 0 o al 1. Si la entrada z tiene un valor considerablemente grande y positivo, la función exponencial “e” elevada a la menos z retorna un valor igual cero, por tanto, la salida de la función sigmoide es 1. En cambio, si la entrada z es grande y negativa, la función exponencial “e” elevada a un número positivo grande y el denominador resultará ser un número grande y por lo tanto la salida será 0 [16]. En la figura 8 podemos ver gráficamente la salida de la función sigmoide.

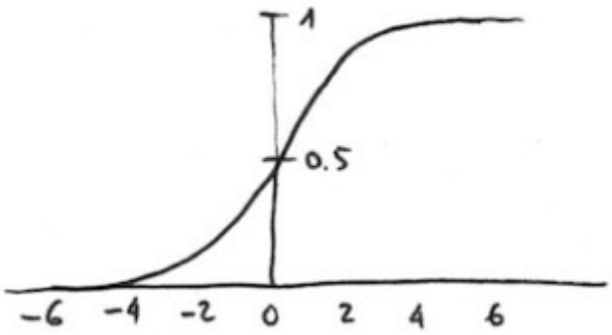


Figura 8. Representación gráfica de la función sigmoide [16]

2.3.1.1.2 Proceso de aprendizaje

El aprendizaje de un Perceptrón podría definirse como la búsqueda del mínimo de una función de error en el espacio abarcado por los pesos de las neuronas. Para cada combinación de pesos se obtiene una salida que difiere (error) de la salida que se desea obtener (aprendizaje supervisado). El aprendizaje se define como la búsqueda de la combinación que minimiza ese error. Unos de los algoritmos más empleados para completar este proceso de aprendizaje es el algoritmo de retropropagación (backpropagation).

Algoritmo de retropropagación (backpropagation)

Este algoritmo es un método de aprendizaje supervisado de gradiente descendente. Este está formado por dos fases:

- Fase de propagación: En esta fase se introduce un patrón de ejemplo a la entrada de la red, y este se va propagando por las diferentes capas hasta generar una salida.

- Fase de aprendizaje: La salida generada en la fase de propagación es comparada con la salida deseada, esta va a generar un error que se irá propagando desde la salida hacia la entrada, de ahí el nombre del algoritmo.

El error que se propaga desde la salida hasta la entrada se va a internar reducir con cada iteración. La forma habitual de calcular este error es haciendo uso del error cuadrático medio indicado en la ecuación (3).

$$e(n) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i(n) - o_i(n)\right)^2 \qquad (3)$$

donde n son cada una de los valores de entrada, N es el número de neuronas de la capa de salida, o_i es la salida deseada y y_i es la salida obtenida correspondiente a cada neurona i de la capa de salida.

El objetivo es minimizar el error cometido $e(n)$, para reducir este se puede usar el método del gradiente descendente. Así cada peso w de la matriz W se actualiza para cada valor de entrada n según se muestra en la ecuación (4).

$$w(n) = w(n-1) - \alpha\frac{de(n)}{dw} \qquad (4)$$

donde α es la tasa de aprendizaje o learning rate.

Tasa de aprendizaje (Learning Rate)

La tasa de aprendizaje es el parámetro que va a determinar la velocidad de aprendizaje del modelo hasta llegar a la convergencia. Este parámetro α tiene un rango [0,1] donde los valores más cercanos a 0 provocan que

el valor de los pesos varíe más lentamente y los valores más próximos a 1 provocan que el valor de los pesos converja más rápidamente hacia la solución correcta. No obstante, según la configuración de la red neuronal, un learning rate cercano a uno puede provocar oscilaciones cuando se está alcanzando su valor óptimo, estas oscilaciones pueden provocar que el peso alcanzado se aleje del mismo. De ahí que sea importante una configuración óptima de la red, encontrando el valor de learning rate adecuado para cada caso.

Modo de entrenamiento

Distintas estrategias de aprendizaje son requeridas según el tipo de procesamiento. Atendiendo a la forma de actualizar los pesos de una red neuronal artificial, los algoritmos de entrenamiento supervisado pueden dividirse en las siguientes clases:

- Entrenamiento estático o fuera de línea. En este tipo de entrenamiento, el modelo se entrena una sola vez, por lo que es este el modelo que se utiliza durante un tiempo. Hay dos tipos de entrenamiento fuera de línea, por secuencias y por épocas:

    - Por secuencias. La actualización de pesos es realizada después de la presentación de cada secuencia. Cada elemento de la secuencia genera un error instantáneo, todos estos errores serán considerados por la función de error.

    - Por épocas. La actualización de pesos es realizada después de la presentación de todas las secuencias, es decir, cuando se ha visto todo el conjunto de entrenamiento. Por tanto, la función de error es basada sobre todo el conjunto.

- Entrenamiento dinámico o en línea. En este tipo de entrenamiento, los datos se ingresan de forma continua al sistema, y estos se añaden al modelo a través de pequeñas actualizaciones periódicas. Hay dos tipos de entrenamiento en línea, puro y por secuencias:

    - Puro. Es necesario cuando se necesita que la red trabaje en tiempo real. En este caso no se consideran elementos de entrenamiento ni de validación. La función de error es generada con el peso actualizado instantáneo de cada elemento.

    - Por secuencias. Al igual que en el puro, los pesos se actualizan después de cada elemento. Además, se suele reiniciar el estado de la red en algunos momentos durante la fase de training, esto suele ocurrir al final de cada sentencia.

2.3.1.1.3 Sobreajuste (Overfitting)

Cuando la red neuronal artificial ya está entrenada, ésta debe ser capaz de afrontar los retos para los que ha sido entrenada a partir de los datos de entrenamiento. Es decir, ahora la red neuronal artificial va a tener que demostrar que es capaz de abordar su tarea, pero ahora con datos diferentes con los que ha sido entrenada. En este caso, se dice que las redes neuronales artificiales son capaces de generalizar.

El sobreajuste u overfitting se da cuando estas redes han sufrido sobreentranamiento, y cuando esto ocurre, las redes neuronales artificiales pierden su capacidad para generalizar. Es un problema habitual en esta disciplina y hay disponibles varios mecanismos para reducir este efecto.

Una medida frecuente para reducir este efecto en esta disciplina es aislar los datos de entrenamiento a los de validación. Para ello se dispone de un subconjunto de los datos de entrenamiento y se usan solamente para validar el modelo, de manera que la red no sea entrenada con estos datos. De esta forma, se puede ver la evolución del aprendizaje del modelo comparando el error cometido entre los datos de entrenamiento y los datos de validación. Así, cuando se observe que la red no aprende más, se finaliza su entrenamiento, conservando de esta forma su capacidad para generalizar. Este ha sido el mecanismo usado en este trabajo, en el punto 4.3 hablaremos de la división de los datos realizada.

En ocasiones, este mecanismo no funciona para solventar o reducir el overfitting, por lo que hay que recurrir a otras medidas. Una de ellas es reducir el aprendizaje del modelo. Para ello basta con reducir el número de capas o el número de neuronas por capa. Otro mecanismo consiste en poner restricciones a los pesos de las neuronas.

Existe un procedimiento más sofisticado que los anteriores, se trata de la **validación cruzada**. En los mecanismos anteriores solo se puede obtener una estimación sobre si el modelo es capaz de generalizar de forma efectiva. En este caso, se trata de realizar un análisis estadístico de forma que se puedan obtener medidas más concluyentes sobre el rendimiento estimado, estas son la media y la varianza. Con estas medidas se puede comprender como va a variar el rendimiento del modelo según los distintos conjuntos de datos. Con esta variación es posible evaluar la confianza en la estimación del rendimiento. Este mecanismo también es útil cuando se hace uso de un conjunto de datos limitado; ya que en lugar de una trivial división de los datos en una de entrenamiento y otra de evaluación; la validación cruzada es capaz de calcular sus estimaciones sobre todo el conjunto de datos realizando numerosas divisiones e intercambios entre datos de entrenamiento y datos de evaluación [19].

2.3.1.2 Redes neuronales convolucionales

Las redes neuronales convolucionales (Convolutional Neural Networks, CNNs), son muy populares dentro de Deep learning y han tenido un gran impacto en el procesado y reconocimiento de imágenes. Esta arquitectura es muy similar a la del perceptrón multicapa. Difieren principalmente en la inclusión de capas convolucionales en estas últimas, de ahí que en esta arquitectura las neuronas de una capa no tengan total conectividad con las neuronas de la capa anterior [16]. De esta forma cada neurona va a contener características específicas de una región en concreto de la capa anterior.

En cuanto a su arquitectura, se han añadido dos capas más respecto de las que tiene el perceptrón multicapa: las capas convolucionales y las capas de pooling. Las capas de entrada y de salida tienen las mismas funcionalidades que en la arquitectura del perceptrón multicapa presentado anteriormente. Por lo que se hablaremos solo de las dos capas incluidas en esta arquitectura y las capas ocultas, ya que aquí incluyen funcionalidades:

- Capas convolucionales: Imaginemos que la capa de neuronas de entrada forma una matriz de tamaño 50x50, en esta capa se trata de aplicar una ventana de menor tamaño (por ejemplo, de 10x10) que se desliza a lo largo de la capa de neuronas, esta ventana en realidad sería un filtro, ya que realiza la operación de convolución cada vez que se desliza. Por cada iteración de la ventana se ocupa una nueva posición en capa anterior y hay una neurona en la capa oculta que procesa la información producto de esta convolución. Puede haber diversas capas convolucionales que realizan esta operación. La figura 9 muestra el proceso de convolución descrito.

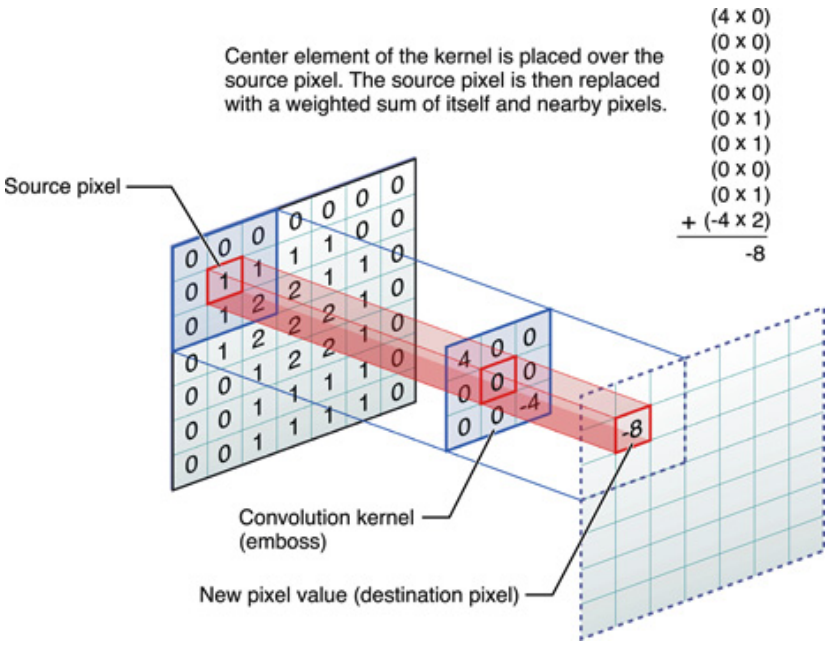


Figura 9. Operación de convolución [18]

- Capa de pooling: En las últimas capas de convolución se añaden unas capas llamadas pooling. Estas capas pooling tienen como misión simplificar la información proveniente la capa convolucional y así formar una versión condensada de las características. De esta forma se consigue reducir notablemente el coste computacional de las siguientes capas convolucionales. Se suelen aplicar dos tipos de pooling: el max-pooling, donde se escoge el máximo valor de una región o el average-poolling, en el que como valor se escoge la media de una región [16]. El la figura 10 se muestra un ejemplo de ambos tipos de pooling.

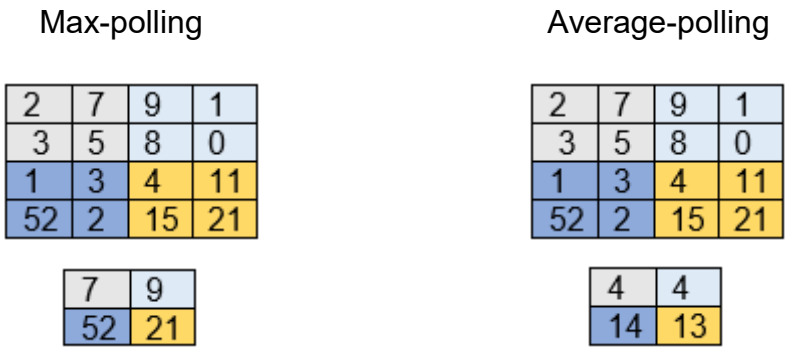


Figura 10. Ejemplo de max-pooling y average-pooling

2.3.1.3 Redes neuronales recurrentes

Cuando nos enfrentamos a problemas y situaciones triviales, por ejemplo, leer un artículo, no tenemos que aprender cada palabra y su significado en ese contexto en concreto, ya que esta información ya es conocida y la tenemos almacenada de forma previa.

Las redes neuronales presentadas anteriormente no tienen la capacidad de operar según experiencias anteriores, por lo que tienen una gran debilidad a la hora solucionar problemas donde se necesite algún tipo de recuerdo previo. Las redes recurrentes trabajan con bucles de retroalimentación permitiendo de esta forma que la información persista una vez pasado cierto tiempo. En la figura 11 vemos una celda de una red neuronal recurrente, donde la celda A ve en su entrada el dato x_t y tiene a su salida un valor h_t. La retroalimentación va a permitir tener disponible el actual valor de la celda en el siguiente time step.

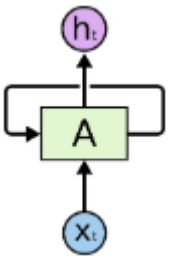


Figura 11. Celda básica de una red neuronal recurrente [12]

Para entender mejor el esquema anterior, en la figura 12 se ve el desglose de la misma. Donde se aprecia explícitamente que el valor de la celda anterior va a estar disponible en la siguiente.

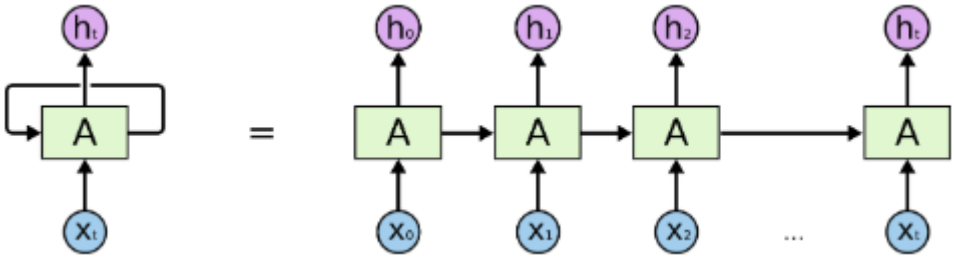


Figura 12. Desglose de figura 8 [12]

De las anteriores figuras se desprende que este tipo de arquitectura está estrechamente relacionada con secuencias. De ahí que este tipo de redes tengan éxito procesando textos y traduciendo los mismos entre otras muchas aplicaciones más.

Sin embargo, este tipo de redes tienen un hándicap. Cuando la brecha que hay entre el momento en que se requiere predecir un dato y el momento donde se encuentra el dato que tiene la información relevante para predecir el actual es demasiado extensa, las redes neuronales recurrentes son incapaces de manejar esta dependencia a largo plazo, no son capaces de aprenderlas. Este problema es conocido como el problema del desvanecimiento del gradiente y es debido a la recurrencia.

Afortunadamente, este problema no ocurre en las redes neuronales recurrentes LSTM, de ahí que sea este tipo de red la elegida para este trabajo.

2.3.1.3.1 Red neuronal LSTM

Las redes LSTM (Long Short-Term Memory) fueron introducidas por Hochreiter y Schmidhuber en 1997 y estas son un tipo especial de red neuronal recurrente que son entrenadas haciendo uso de la retropropagación a través del tiempo [5] y consiguen evitar el problema de desvanecimiento y explosión del gradiente. De ahí que sean capaces de recordar información en largos periodos de tiempo.

El bloque de memoria LSTM puede guardar su valor durante un periodo de tiempo corto o largo como una función de sus entradas, esto permite al bloque de memoria recordar la información que es relevante y no solo el último valor calculado. Este está formado entre otros elementos, por tres puertas que controlan el modo de como la información fluye dentro o fuera de la celda:

- La puerta de entrada: esta controla cuando la información nueva puede entrar en la memoria.

- La puerta del olvido: su función es decidir cuándo se desecha información que no es relevante, lo que permite a la celda tener espacio para recordar nuevos datos.

- La puerta de salida: esta decide si se utiliza en el resultado del bloque de memoria la información que está contenida en esta.

El bloque de memoria LSTM también contiene ponderaciones [11] que controlan a cada puerta. El algoritmo BPTT optimiza esas ponderaciones basándose en el error de salida cometido.

La idea principal que hay detrás de estas redes viene dada por su celda de estado. Una red LSTM es capaz de agregar y quitar información en la celda de estado de forma regulada a través de unas estructuras llamadas puertas [12], podemos ver en la figura 13 la celda de estado, se puede imaginar como una cinta transportadora donde se agrega o quita información. A continuación, vamos a explicar su funcionamiento.

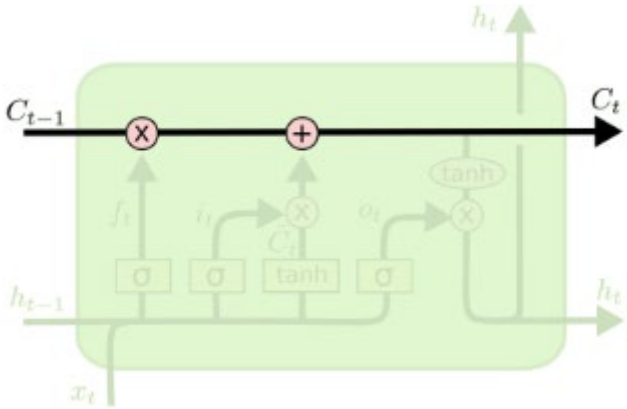


Figura 13. Bloque de memoria LSTM con celda de estado resaltada [12]

La primera decisión que un bloque de memoria LSTM ha de tomar es qué información va a quitar de la celda de estado. Para ello hace uso de la puerta de olvido, dicha puerta contiene una capa sigmoide, esta puerta mira a la salida anterior h_{t-1} y la entrada actual x_t y da una salida entre 0 y 1 para cada número de la celda de estado anterior C_{t-1}. Donde 0 representa que el dato no se mantiene en la celda de estado y, por el contrario 1 representa que el dato se mantiene en dicha celda [12].

En la figura 14 podemos ver un bloque de memoria LSTM con la puerta de olvido resaltada. A continuación, se muestra la ecuación de la puerta de olvido (5). Donde σ representa la función sigmoidea comentada anteriormente, W_f es la matriz de pesos de dicha puerta, así como h_{t-1} y x_t son la salida anterior y el dato actual respectivamente y b_f es el vector bias de la misma puerta [12].

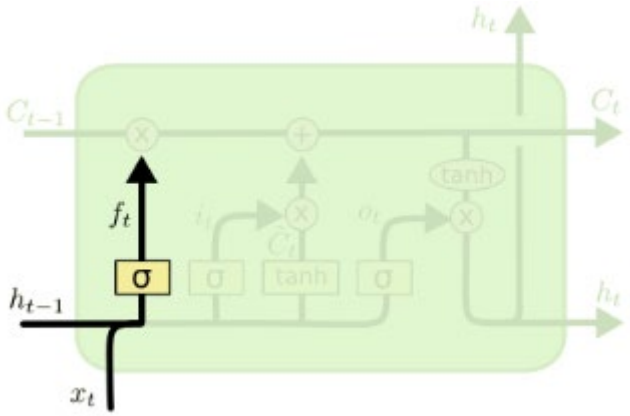


Figura 14. Bloque de memoria LSTM donde se resalta la puerta de olvido [12]

$$f_t = (W_f \cdot [h_{t-1}, x_t] + b_f) \tag{5}$$

Quitada la información más irrelevante, llega el momento de decidir qué información va a ser almacenada en la celda de estado. En este caso intervienen dos elementos, la puerta de entrada, que decide que valores serán actualizados y una capa con la función hiperbólica, que creará un vector con los valores candidatos $\widetilde{C_t}$ a ser almacenados en la celda de estado C_t. En la figura 15 se muestran nuestro bloque de memoria LSTM con la puerta de entrada y la capa con la función hiperbólica ambas resaltadas [12].

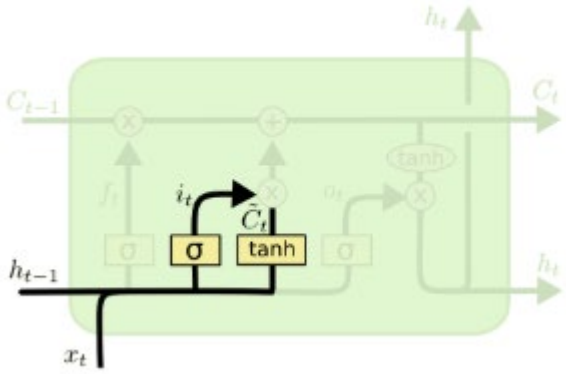


Figura 15. Bloque de memoria LSTM con la puerta de entrada y la capa con la función hiperbólica ambas resaltadas [12]

En las siguientes ecuaciones podemos ver la ecuación de la puerta de entrada (6), donde W_i representa los pesos de la puerta de entrada y b_i el vector bias de dicha puerta, y la ecuación del candidato (7), donde W_C y b_C representan la matriz de pesos del candidato y el vector bias del candidato, estos elementos son función de una tangente hiperbólica, para forzar los valores entre -1 y 1 [12].

$$i_t = (W_i \cdot [h_{t-1}, x_t] + b_i) \tag{6}$$

$$\widetilde{C_t} = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{7}$$

En este punto, se actualiza la celda de estado anterior C_{t-1} a la nueva celda de estado C_t haciendo, para ello se hace uso de las anteriores expresiones en la ecuación (8). En la figura 16 vemos resaltado los elementos que entran en juego para actualizar la celda de estado [12].

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t \tag{8}$$

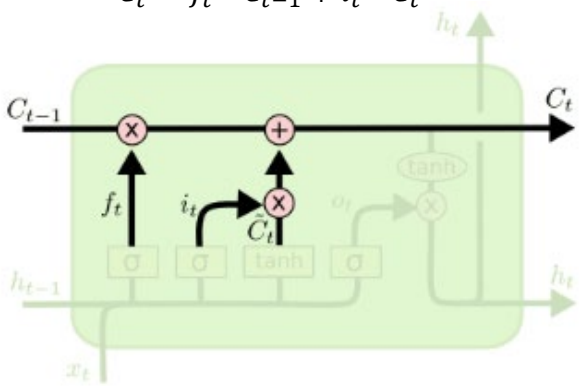


Figura 16. Bloque de memoria LSTM donde se resaltan los elementos actualizados por la celda de estado [12]

Para finalizar, se debe obtener cual va ser la salida del bloque de memoria LSTM. La cual, lógicamente se va a basar en la celda de estado, pero será una versión filtrada [12]. Para ello, se usará una función sigmoidea que decidirá que partes de la celda de estado vamos a sacar, como se puede ver en la ecuación de la puerta de salida (9).

$$o_t = tanh(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{9}$$

A continuación, con la ayuda de la función tangente hiperbólica, al igual que anteriormente, se fuerzan los valores entre -1 y 1 de la celda de estado, dicho resultado es multiplicado por la puerta de salida, así se obtienen solo los elementos de la salida que se han decidido, lo podemos ver en la ecuación (10).

$$h_t = o_t * \tanh(C_t) \tag{10}$$

Igualmente mostramos en la figura 17 los elementos que intervienen en esta etapa final comentada.

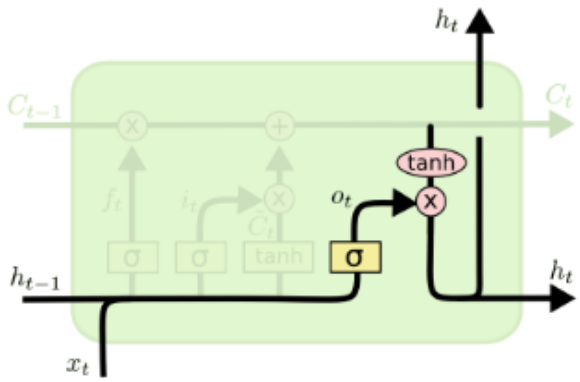


Figura 17. Bloque de memoria LSTM donde se resalta la última etapa del bloque [12]

2.4 Software Utilizado

El desarrollo de software para Deep Learning se puede hacer en diferentes lenguajes de programación. Para este trabajo se tuvo claro desde un inicio que el lenguaje de programación sería Python debido a que este es el más extendido en la disciplina de inteligencia artificial por su robusto y gran número de librerías para tal fin.

2.4.1 PyCharm

Se ha elegido como IDE el software PyCharm community edition [6] debido a que esta es una versión de software libre, además este software dispone de una gran facilidad de uso ya que incorpora una intuitiva interfaz. Lo que hace que sea una tarea trivial poder instalar diferentes librerías sin necesidad de usar el interpreté de comandos. Debemos resaltar también que muestra de forma muy explícita todas las variables que manejadas en el entorno y cuanta con depurador de muy fácil uso, el cual facilitará del desarrollo de este software.

Figura 18. ©PyCharm

Usar esta IDE tiene ventajas similares a las ofrecidas por otras, además, incluye algunas características propias por las cuales goza de gran popularidad. Este posee un editor inteligente, el cual puede completar código en las sentencias mediante predicción de código y atajos de teclado. Otra característica que lo diferencia, le permite navegar a través del código implementado, saltando entre las clases y métodos creados, haciendo de esta forma la implementación mucho más dinámica y fluida.

En todo caso, una de las características que más sobresale de PyCharm es la posibilidad que tiene de refactorizar el código, es decir, este IDE es capaz de hacer cambios en el código sin comprometer la ejecución del mismo.

Por último, señalar que PyCharm posee una gran cantidad de temas y plugins que se pueden usar para trabajar más cómodamente en su entorno.

2.4.2 TensorFlow

En el año 2011, Google Brain desarrollo un sistema propietario para para tareas de aprendizaje automático basado en Deep Learning llamado DistBelief. Su uso creció rápidamente y Google intensificó sus esfuerzos en conseguir una biblioteca más robusta y rápida que sería denominada TensorFlow [7].

TensorFlow es una biblioteca de software liberado por Google el 9 de noviembre del 2015. Es utilizado para calcular operaciones numéricas a través de diagramas de flujo de datos. Las operaciones matemáticas son representadas por los nodos de los diagramas, y las aristas muestran tensores, que en realidad son matrices de datos multidimensionales que están comunicadas entre ellas [7].

Figura 19. ©TensorFlow

Este framework incluye un amplio conjunto de herramientas de visualización que simplifican la comprensión, depuración y optimización de las aplicaciones, por lo que proporciona un nivel de control bajo para maximizar la flexibilidad y el rendimiento. Admite una gran variedad de estilos, desde imágenes y sonido a histogramas y gráficos, así como el entrenamiento de redes neuronales profundas con rapidez y facilidad. También ofrece APIs de alto nivel que facilitan el desarrollo y la implementación de los modelos.

Con TensorFlow se tiene acceso a una extensa documentación y gran número de tutoriales que le facilitarán el desarrollo del usuario en el aprendizaje automático. Además, TensorFlow goza de una comunidad con gran número de usuarios muy activos, que aportan constantes contribuciones de código y resolución de problemas en GitHub.

2.4.3 Pandas

Pandas [8] es una biblioteca de código abierto con licencia BSD que proporciona estructuras de datos de alto rendimiento y operaciones para manipular tablas numéricas y series temporales que son fáciles de usar, así como herramientas de análisis de datos para el lenguaje de programación Python.

Pandas es un proyecto patrocinado por NumFOCUS. De esta manera ayudará a asegurar el éxito del desarrollo de Pandas como un proyecto de código abierto a escala mundial.

Pandas ofrece las siguientes estructuras de datos:

- Series: Son arrays de una sola dimensión con indexación, es decir, arrays con índice o etiquetas. Por lo que un objeto de este tipo tiene dos componentes: un índice y un vector de datos. El índice está formado por valores únicos que normalmente están ordenados.

- DataFrame: Es una estructura de datos tabular similar a las tablas de bases de datos relacionales como por ejemplo SQL. Por norma general, cuando se realiza un estudio estadístico sobre los sujetos o individuos de una muestra, la información se presenta en un dataframe, es decir, una hoja de datos, en los que cada fila corresponde a un sujeto y cada columna a una variable. La estructura de un dataframe es muy similar a la de una matriz, la diferencia es que esta sólo recoge valores numéricos, mientras que un dataframe puede incluir también datos alfanuméricos.

- Paneles: Una dataframe no permite trabajar con más de dos dimensiones. Para hacerlo, Pandas incorpora estructuras capaces de hacerlo, estas son Panel, Panel4D y PanelND.

2.4.4 Scikit-learn

Scikit-learn [9] es una de las mayores librerías para machine learning sobre Python.

Esta librería proporciona entre otros:

- Ejemplos de trabajo
- Herramientas de tratamientos de datos
- Modelos de aprendizaje automático
- Herramientas de evaluación de modelos

En esta están implementadas diversas técnicas como pueden ser entre otras: clustering y arboles de decisión. Además, incluye herramientas para el tratamiento de datos como pueden ser la normalización y el escalado de los mismos. Un inconveniente de esta librería viene dado a que el usuario se va a encontrar con la imposibilidad de cálculo sobre el modelo mediante una GPU, ya que actualmente esta librería no lo soporta.

Figura 20. ©Scikit-learn

2.4.5 Matplotlib

Matplotlib [10] es una biblioteca de generación de gráficos 2D para Python a partir de datos contenidos en arrays o en listas. Matplotlib se puede usar en scripts de Python, en shells tanto de Python como de IPython, notebook de Jupyter, en servidores de aplicaciones web y en herramientas de interfaces gráficas de usuario.

3. Descripción del trabajo

Una de las motivaciones de este trabajo es cubrir las necesidades de gestión de las bandas de frecuencias en Repetidores de Salto de Frecuencia (FSR). Para ello, es necesario predecir la cantidad de llamadas que estos van a cursar.

De ahí que este trabajo base sobre la implementación de un sistema de predicción que sea capaz de predecir tráfico en los diferentes puntos de una red celular, en este caso, los diferentes puntos corresponderán con diferentes zonas geográficas de la ciudad de Milán. Para ello, se usarán técnicas de Deep Learning en las que nos facilitará su implementación la ayuda de la biblioteca de código abierto de Google denominada TensorFlow. El lenguaje de programación para su implementación será Python.

Los datos utilizados corresponden a una serie temporal con diferentes observaciones donde estas están dispuestas de forma estructurada. Los datos son descargados de Harvard Dataverse [4]. Este es resultado de la computación en CDRs generados por la red celular de Italia Telecom en la ciudad de Milán. Por lo que estos datos, además de ser de una fuente fiable, tienen una Licencia Abierta de Bases de Datos (en inglés Open Database License (ODbL)), así como el formato correcto para logar el objetivo de este trabajo.

Por otra parte, se ha decidido usar técnicas de Deep Learning debido a que estas disponen de las arquitecturas y algoritmos ideales y con mejor rendimiento para tal fin. Además, se ha optado por el uso de TensorFlow principalmente por el gran número de funciones disponibles y por la granularidad tan explícita que muestra su implementación, ya que esta es una framework de bajo nivel donde operaciones como el producto de matrices son implementadas, de ahí que dará una visión didáctica para principiantes. Otras framework de bajo nivel son PyTorch, Theano y MXNet. Como aplicación de alto nivel se podría usar Keras, con esta framework se pueden conseguir modelos más sencillos, limpios y rápidos, pero no se desprende de forma tan natural los modelos de redes neuronales, por eso se ha optado por TensorFlow.

Una propuesta de la secuencia del sistema de pre-procesado consistiría la descarga de los dados del volumen de tráfico de la ciudad de Milán, tratar estos para conseguir el formato deseado de forma que se pueda operar con los mismos a través de TensorFlow, entrenamiento de la red neuronal, optimización del resultado, comprobación mediante métricas los datos predichos y la visualización de los resultados. En la figura 21 se muestra el diagrama de bloques del sistema completo.

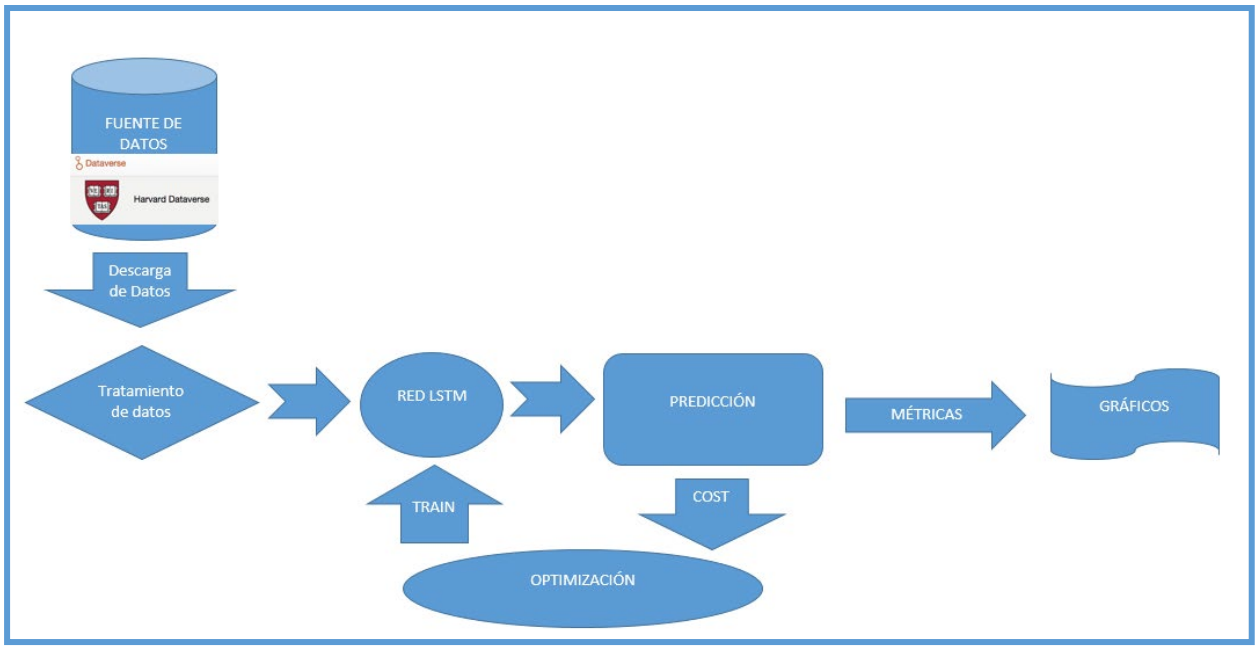


Figura 21. Diagrama de bloques del sistema completo

3.1 Aspectos del problema a resolver

Los datos provistos en este problema a resolver son los descargados correspondientes al volumen de tráfico generado por los usuarios de la ciudad de Milán, este es cursado por la rede celular de dicha ciudad [4]. Dado que los datasets provienen varias compañías, estas han adoptado diferentes estándares, por ejemplo, dividir un área en diferentes regiones, esto es, formar un grid con celdas cuadradas de forma que sea posible hacer comparaciones entre diferentes regiones. De ahí que el área de Milán se haya dividido por un grid de 1000 celdas, cada una de con un área de $235\ m^2$ [15]. En la figura 22 podemos ver el grid de la ciudad de Milán.

9901	9902	···	9999	10000
9801	···	···	9899	9900
···	···	···	···	···
101	102	···	···	200
1	2	3	···	100

Figura 22. Grid correspondiente a la ciudad de Milán [15]

Cada vez que un usuario interactúa con un servicio de telecomunicación, una BTS es asignada por el operador y se encarga enviar la comunicación a través de la red. Es cuando un nuevo registro de llamadas (CDR) es

creado grabando el tiempo de interacción y la BTS que ha intervenido. Es posible obtener la localización geográfica de los usuarios a partir de cada BTS asociada a estos gracias a los mapas de cobertura de cada región que abastece dicha BTS [15]. En la figura 23 se muestra el mapa de cobertura de la ciudad de Milán según la distribución de las diferentes BTSs que cubren la ciudad.

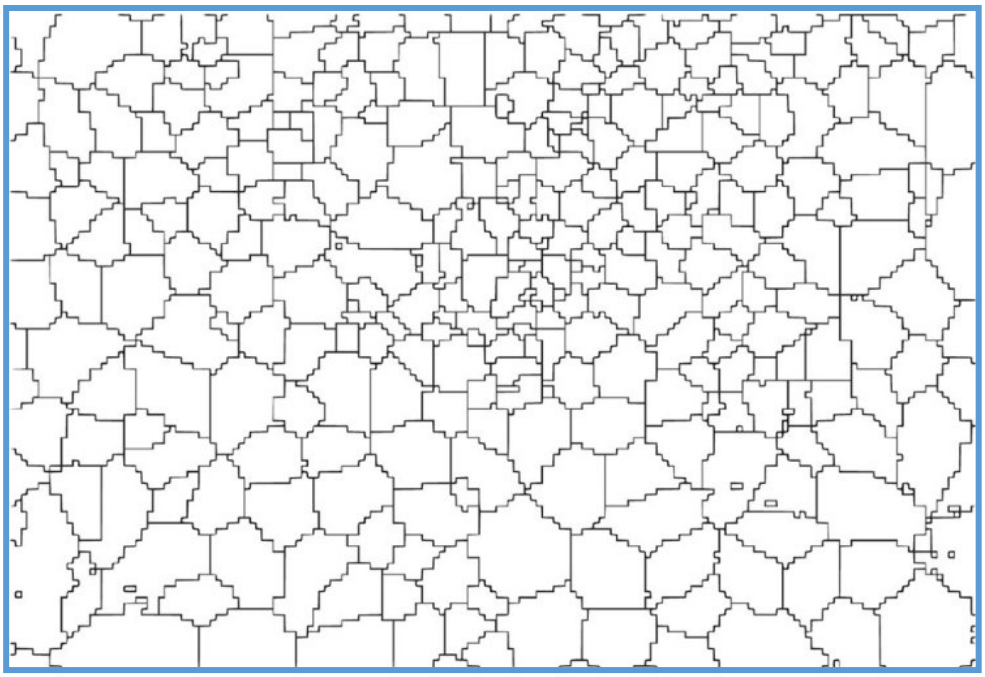


Figura 23. Mapa de cobertura de la ciudad de Milán [15]

En el dataset proporcionado se representa la actividad de telecomunicaciones correspondiente a los usuarios de Telecom Italia de Milán y de otros usuarios que utilizan la red mientras se desplazan. En dicho dataset están presentes los siguientes eventos: llamadas salientes y entrantes, SMS enviados y recibidos, así como las conexiones a internet.

La descarga de datos nos ofrece un archivo con formato texto donde podemos ver que los datos están dispuestos en intervalos de 600.000 milisegundos (10 minutos). Consecuentemente, de lo anterior se deduce que se dispone una serie temporal con un formato de datos estructurado. Con esto vemos que el problema que tenemos que afrontar difícilmente se podrá acometer mediante técnicas de clasificación, propias de redes neuronales convolucionales (CNN). Este tipo de problemas pueden ser resueltos por ejemplo con una Regresión Lineal o mejor aún, a partir de redes neuronales recurrentes (RNN) por su tipo de estructura, ya que estas requieren incluir la variable tiempo para la activación o estado de una neurona [5].

3.2 Repetidores de Salto de Frecuencia (FSR)

La mayoría de las zonas rurales en España no disponen de cobertura 3G, es más, estas zonas apenas pueden disfrutar de un servicio de llamadas 2G. Esta problemática está causada porque los operadores españoles no obtienen rentabilidad cuando invierten en una estación base en estas zonas, ya que estás tienen una población muy baja, normalmente menor de 500 habitantes.

Para solventar este problema, algunos operadores españoles recurren a pequeños repetidores de alta ganancia que consiguen cubrir las necesidades de estas poblaciones en cuanto a cobertura 2G y 3G se refiere. En la figura 24 se muestra como ejemplo un repetidor de un fabricante conocido.

Figura 24. Repetidor de salto de frecuencia.
[Fuente, elaboración propia]

Estos sistemas funcionan para la banda de trabajo en 900 MHz y para la banda de transmisión en 1800 MHz. La figura 25 muestra más detalle de la forma de operación.

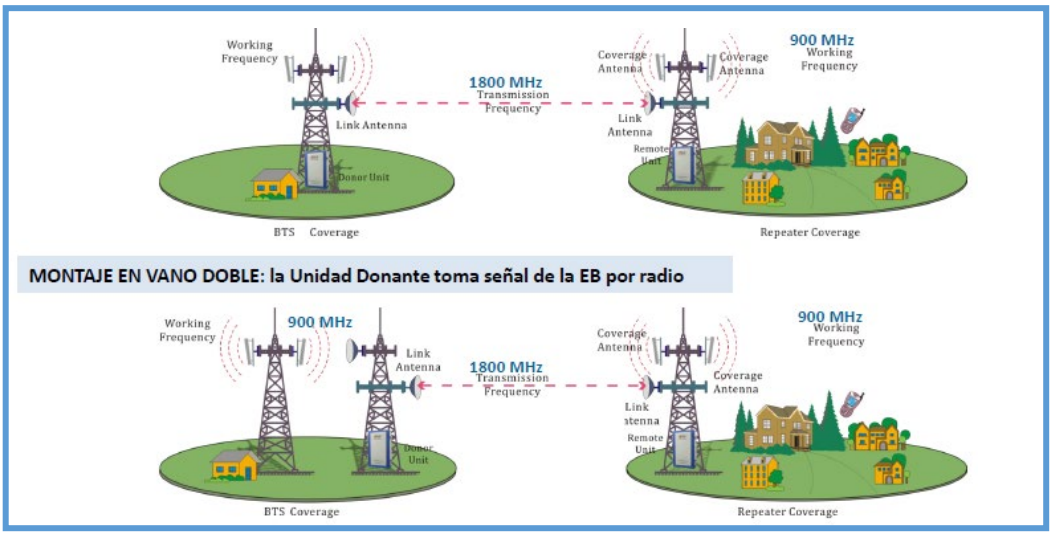


Figura 25. Bandas de operación para casos de vano simple y de vano doble

Estos repetidores operan con tres canales, estando el canal 1 destinado para 2G y los canales 2 y 3 para 3G. Sin embargo, para reducir el consumo y disponer de más potencia en los canales activos, suelen estar de forma general solo dos canales encendidos, estos son el 1 y el 2, ya que con estos dos canales hay suficientes recursos para abastecer a la población rural, activándose solo el canal 3 de forma manual si no se pudiesen cursar todas las llamadas de la población por falta de recursos. Este caso se podría dar si la población creciese en torno a un 30-40%. Este incremento se suele dar algunos fines de semana, puentes y épocas vacacionales. El problema en caso de un incremento de la población en este valor radica en que el canal 3 no se activa de forma automática porque no hay un sistema automatizado para ello, por lo tanto, lo tiene que hacer una persona cuando en una revisión rutinaria se diese cuenta de que hay más llamadas descargadas de lo habitual o si los vecinos de la zona ponen una reclamación al respecto, ya que algunos no podrán realizar llamadas ocasionalmente. Las figuras 26 muestra los canales activos de forma permanente en estos sistemas.

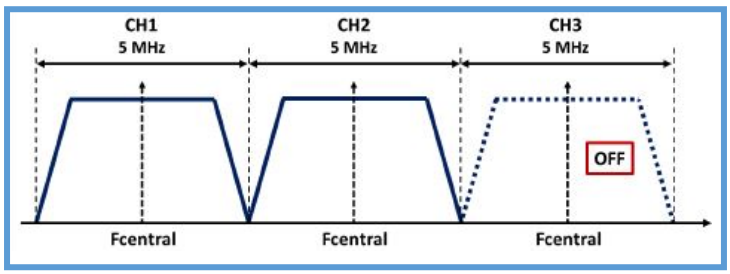


Figura 26. Canales de 5 MHz en FSR, el tercer canal está apagado

3.2.1 Gestión de bandas de frecuencias para FSR

Estos repetidores son monitorizados por un sistema NMS (Network Management System). Este software entre muchas opciones, permite activar y desactivar de forma manual los canales de trabajo de cualquier repetidor en cuestión que este monitorizado por este. La figura 27 muestra el interfaz del NMS en su pantalla principal con todos los repetidores en el territorio español.

Figura 27. Pantalla inicial de NSM

En la figura 28 vemos el interfaz para activar y desactivar los canales. En este caso el canal 3 está desactivado.

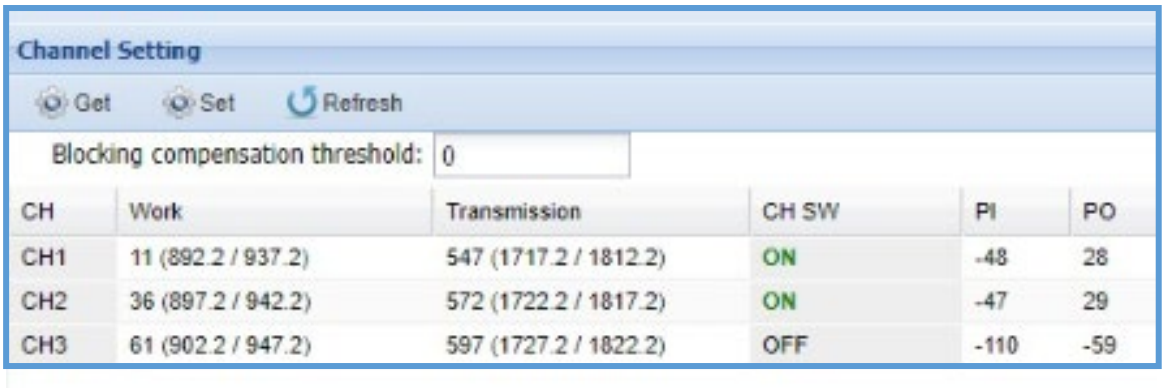


Figura 28. Interfaz para la activación/desactivación de canales

Una solución para que se cursen el máximo número de llamadas cuando la población rural aumenta, sería que se active el canal 3 de forma automática antes de que el número de llamadas se incremente. Po lo que la implementación de un sistema de predicción de llamadas que sea capaz de predecir el número de llamadas salientes alrededor algunas horas solucionaría este hándicap. Esta aplicación puede estar implementada de forma interna al sistema de monitorización NMS o como un módulo externo en la BTS, de forma que cuando en un repetidor concreto el número de llamadas cursadas sea el 85 % de su máximo habitual, se activará el canal 3. De forma similar se desactivará dicho canal si se predice que el número de llamadas está por debajo del 70 % de su máximo habitual**.** Se deja un 15 % de llamadas de diferencia entre la activación/desactivación para que no haya confusión entre los dos estados de conmutación. **El sistema podrá activar/desactivar el canal 3 con una anticipación igual al horizonte de predicción, por lo que se reducirá de forma considerable el número de llamadas no cursadas cuando haya un incremento considerable en la población. Además, se mantendrá la eficiencia del sistema desactivando de nuevo el canal 3 cuando el número de llamadas disminuya.**

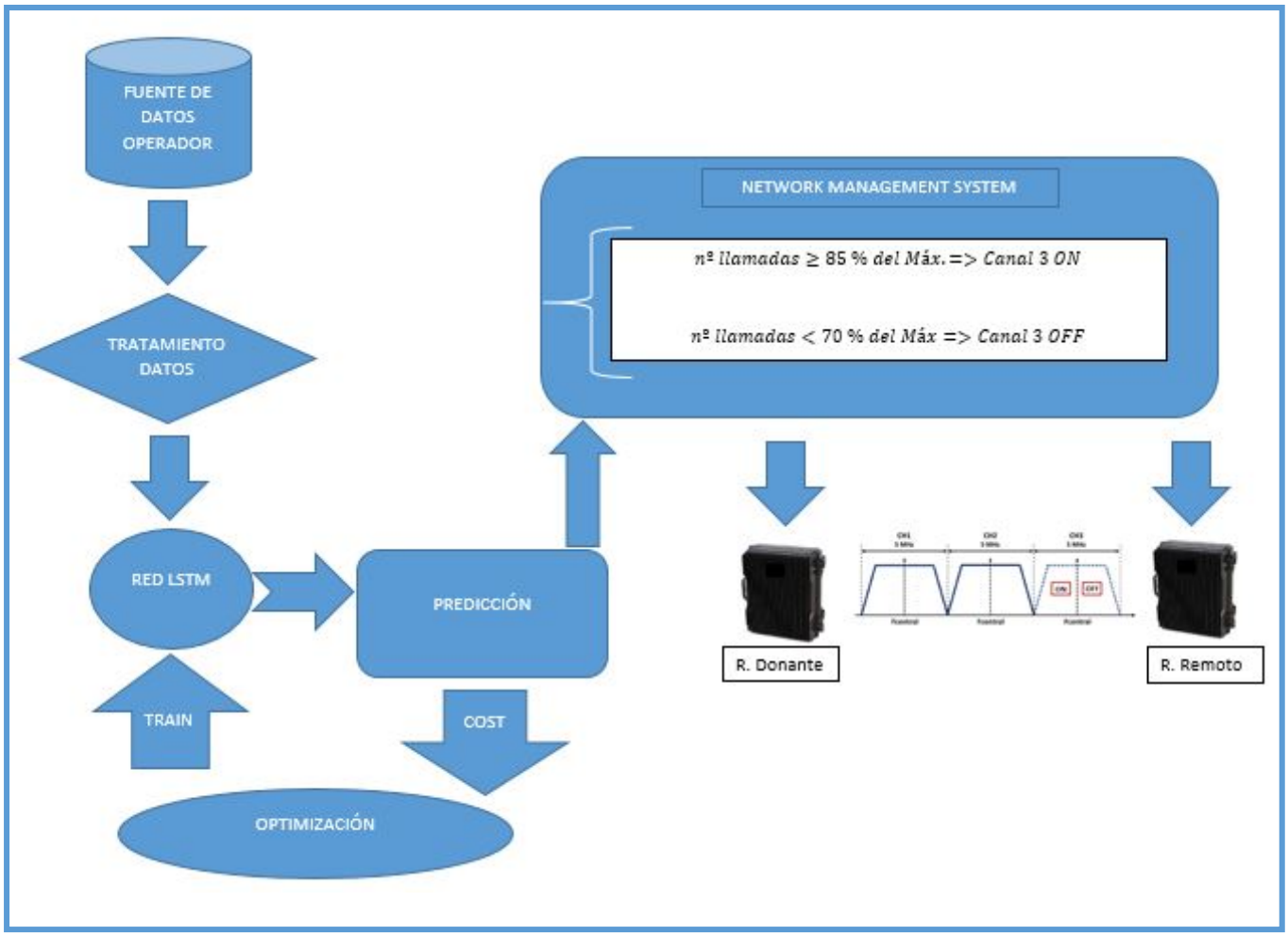


Figura 29. Diagrama de bloques de NMS con el sistema de predicción

En la figura 29 se muestra el diagrama de bloques del sistema completo, incluido la gestión de bandas de frecuencias. El diagrama de bloques indica como los datos recolectados por el operador son tratados para entrenar la red LSTM. Cuando dicha red está entrenada, la salida del sistema de predicción comunica al NSM que este active o desactive el canal 3 de cada repetidor donante y remoto. Una vez el NMS da la orden, los repetidores tienen el tercer canal activo o no, según el número de llamadas salientes.

4. Metodología e implementación

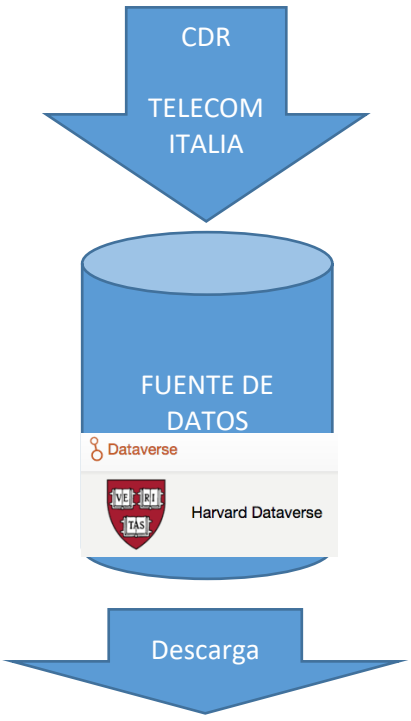
Como hemos comentado en apartados anteriores, estamos ante un problema de predicción. Una forma de intentar resolver este sería recurrir a una arquitectura de RNN. Sin embargo, para predecir valores futuros con un horizonte considerable, vamos a necesitar que la red neuronal aprenda de experiencias pasadas, para ello es necesario que la red neuronal sea capaz recordar valores a largo plazo, pero ya comentamos en el apartado 2.3.1.3 que las RNNs olvidan con el tiempo. Por este motivo se ha decidido llevar a cabo la implementación a partir de una arquitectura de red neuronal recurrente LSTM, donde desaparece este problema.

4.1 Tratamiento y formato de datos

9901	9902	···	9999	10000
9801	···	···	9899	9900
···	···	···	···	···
101	102	···	···	200
1	2	3	···	100

Figura 30. Grid correspondiente a la ciudad de Milán [15]

Datos recolectados del área de Milán, están divididos en 1000 celdas de 235 m^2, lo que facilita la comparación entre diferentes regiones. Se puede ver en la figura 30



Cada vez que un usuario interacciona con una BTS, se crea un CDR en ese instante con la BTS asociada

Dataset con los datos correspondientes a las 1000 celdas asociadas. Se descargan para 42 días. Este está alojado en Harvard Dataverse [4]

Figura 31. Archivo de datos con formato texto

Los archivos datos descargados desde la web Harvard Dataverse [4] tienen formato texto. En la figura 31 podemos el aspecto de los datos, donde cada fila y columna no están identificadas nominalmente.

Los datos son importados con ayuda de la librería Pandas

PANDAS

pd.read_csv

	CellID	countrycode	smsin	smsout	callin	callout	internet
2013-11-01T12:40:00.000000000	1	0	0.43295	nan	nan	0.08190	nan
2013-11-01T12:40:00.000000000	1	20	nan	nan	nan	0.07841	nan
2013-11-01T12:40:00.000000000	1	355	nan	nan	0.02614	nan	nan
2013-11-01T12:40:00.000000000	1	380	nan	nan	nan	0.05460	nan
2013-11-01T12:40:00.000000000	1	39	0.81175	0.70184	0.59962	0.41030	13.93831
2013-11-01T12:40:00.000000000	1	46	nan	nan	nan	nan	0.02614
2013-11-01T12:50:00.000000000	1	0	0.18654	nan	nan	0.02730	nan
2013-11-01T12:50:00.000000000	1	20	nan	nan	nan	0.00179	nan
2013-11-01T12:50:00.000000000	1	370	0.02614	nan	nan	0.02614	nan

Figura 32. Filas de datos en el mismo intervalo temporal.

El valor nominal relativo a cada columna es [15]:

- Square id
- SMS-in activity
- SMS-in out
- Call-in activity
- Call-out activity
- Internet traffic activity
- Code country

Las filas están indexadas cada 10 minutos (figura 32). Puede haber filas en mismos intervalos temporales

PANDAS

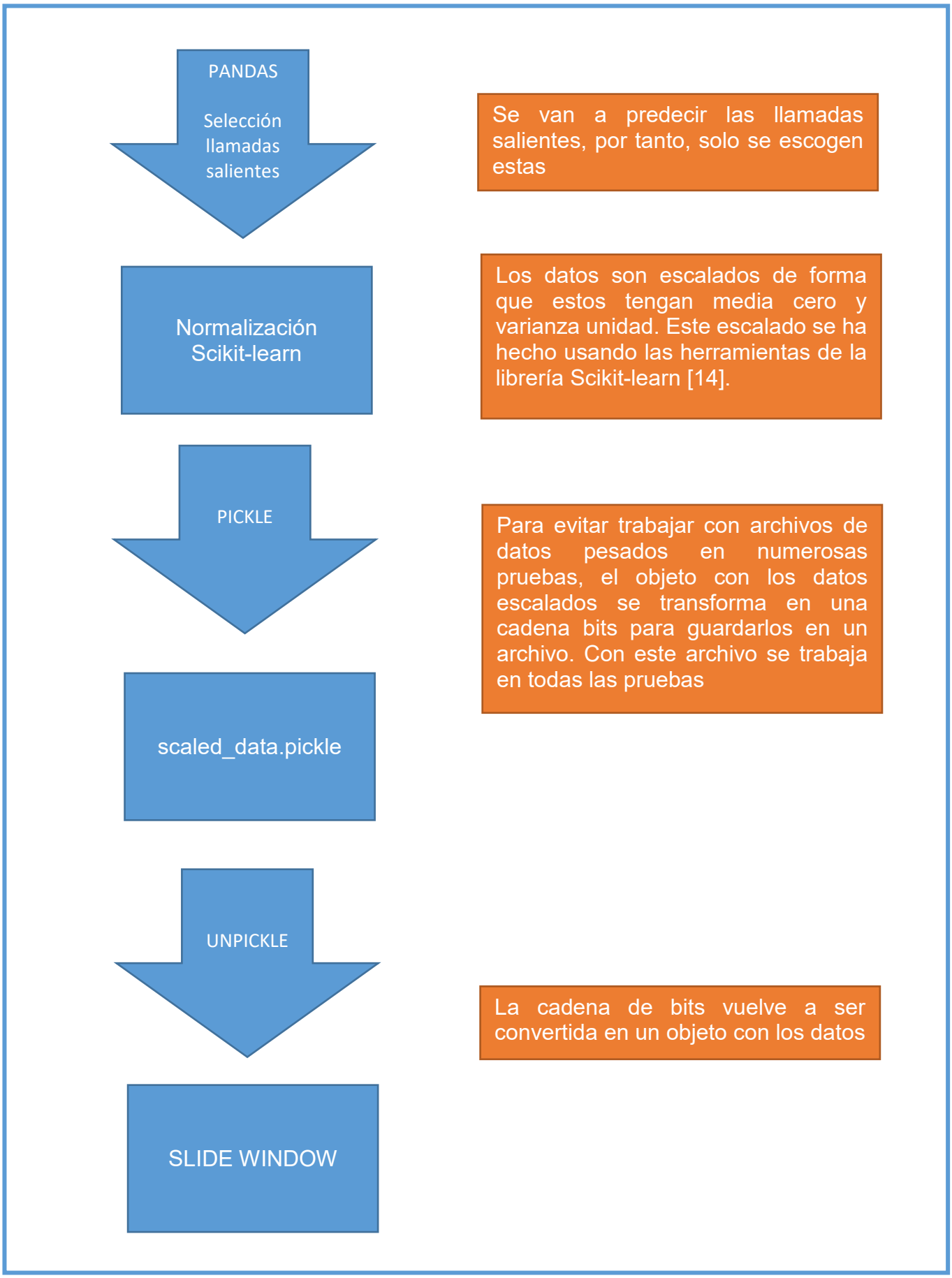
Filtrado y agrupación

Filtrado y agrupación de las 4 primeras celdas contiguas. Se consigue un área de 470 m^2 por celda, que es más acorde con el área de cobertura de una RBS. Datos más decorrelados, también desaparecen las celdas sin datos

9901	9902	...	9999	10000
9801	...	...	9899	9900
...	...	...	...	...
101	102	...	...	200
1	2	3	...	100

Figura 33. Nueva extensión de la celda 1

La figura 33 muestra la nueva extensión de área de la celda 1 en la ciudad de Milán


PANDAS
Selección llamadas salientes
Se van a predecir las llamadas salientes, por tanto, solo se escogen estas
Normalización Scikit-learn
Los datos son escalados de forma que estos tengan media cero y varianza unidad. Este escalado se ha hecho usando las herramientas de la librería Scikit-learn [14].
PICKLE
Para evitar trabajar con archivos de datos pesados en numerosas pruebas, el objeto con los datos escalados se transforma en una cadena bits para guardarlos en un archivo. Con este archivo se trabaja en todas las pruebas
scaled_data.pickle
UNPICKLE
La cadena de bits vuelve a ser convertida en un objeto con los datos
SLIDE WINDOW

4.2 Ventana deslizante

Una red LSTM predice el dato o datos que sigue a un conjunto de datos de entrada, en realidad todos estos datos de entrada forman un input. De ahí que se haya decidido usar una ventana deslizante para alimentar nuestra red neuronal [1].

Esta ventana deslizante va a generar periodos de datos de entrada que van desde x_n hasta x_{n+SW-1}, el dato predicho se prevé que coincida o se asemeje al dato que cae en la muestra $x_{n+SW+N-1}$, siendo *n* el primer dato actual, *SW* la longitud de la ventana escogido y N el horizonte de predicción. Con lo cual, dado que predecimos el siguiente dato y entre estos hay 10 minutos, esté será el tiempo a predecir por número de muestras vista a predecir. En la tabla 2 podemos ver una venta deslizante con longitud para tres muestras que se ha deslizado cuatro veces, donde SW sería 3. Se pretende que los valores de la columna Dato Predicho se asemejen lo máximo posible a las etiquetas Y_data.

Muestra 1	Muestra 2	Muestra 3	Dato Predicho
x_1	x_2	x_3	$x_{1+SW+N-1}$
x_2	x_3	x_4	$x_{2+SW+N-1}$
x_3	x_4	x_5	$x_{3+SW+N-1}$
x_4	x_5	x_6	$x_{4+SW+N-1}$

Tabla 2. Ventana deslizante

A modo de ejemplo en la tabla 3 se muestran los valores para X_data de una ventana deslizante de 3 muestras y sus correspondientes valores de Y_data cuando con un horizonte de predicción de 2 muestras. Para un vector original $X = [1, 2, 3, 4, 5, 6, 7, 8]$ los valores de X_data e Y_data son:

X_data 1	X_data 2	X_data 3	Y_data
1	2	3	5
2	3	4	6
3	4	5	7
4	5	6	8

Tabla 3. Ventana deslizante con horizonte de predicción a dos muestras vista

4.3 División de los datos

Con motivo de comprobar la capacidad de predicción del modelo con datos de entrada a partir de los cuales la red neuronal no ha sido entrenada, y por lo tanto no conoce, además de evitar overfitting, los datos tratados son divididos. De ahí la importancia de dividir los datos tratados en tres partes, una para el entrenamiento del modelo (**Training**), otra para validar (**Validation**) que el modelo ha sido entrenado correctamente y una última división para testear la red entrenada con datos que no ha visto la

red. El porcentaje para escogido para cada tarea ha sido 60%, 20% y 20% para entrenamiento, evaluación y test respectivamente [13]. En la figura 34 se muestra un diagrama de bloques con la división de datos comentada. En la que vemos que la forma del dataset varía en función del tamaño de la ventana utilizada (SLIDE WINDOW, SW).

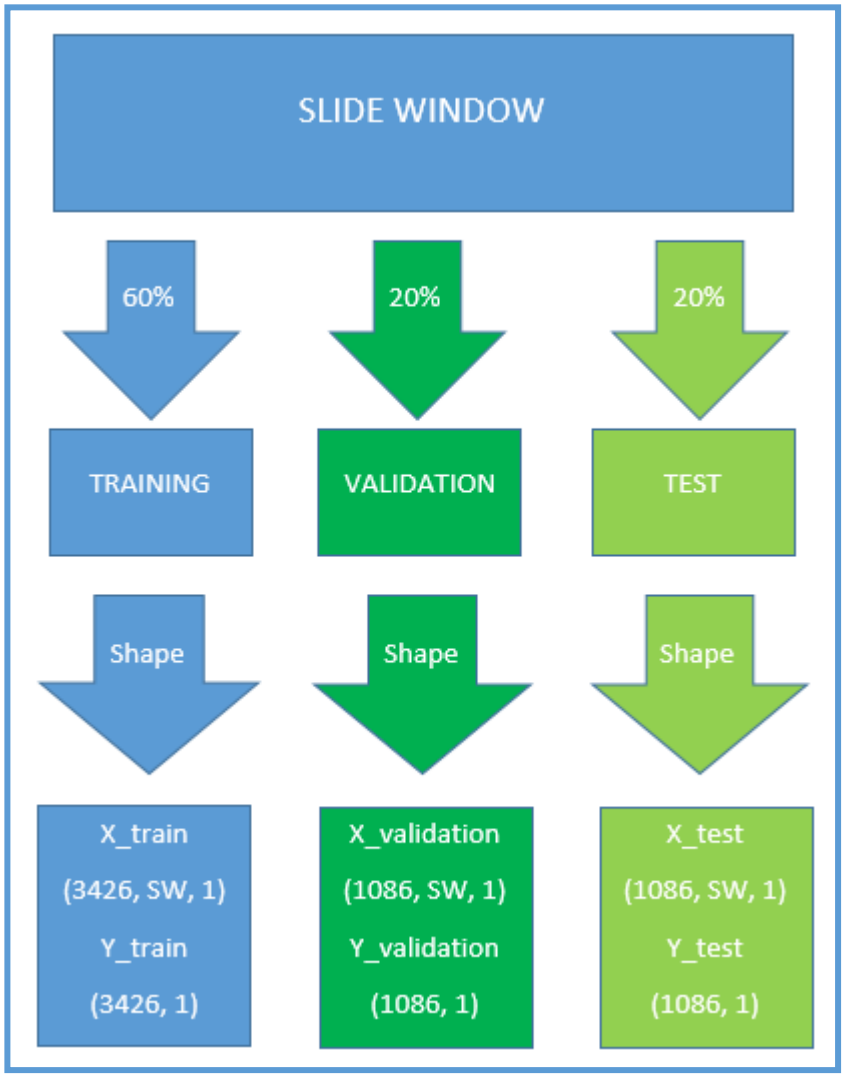


Figura 34. División de datos realizada

Finalmente, en la figura 35 se muestra un resumen del tratamiento realizado a los datos anteriormente presentados.

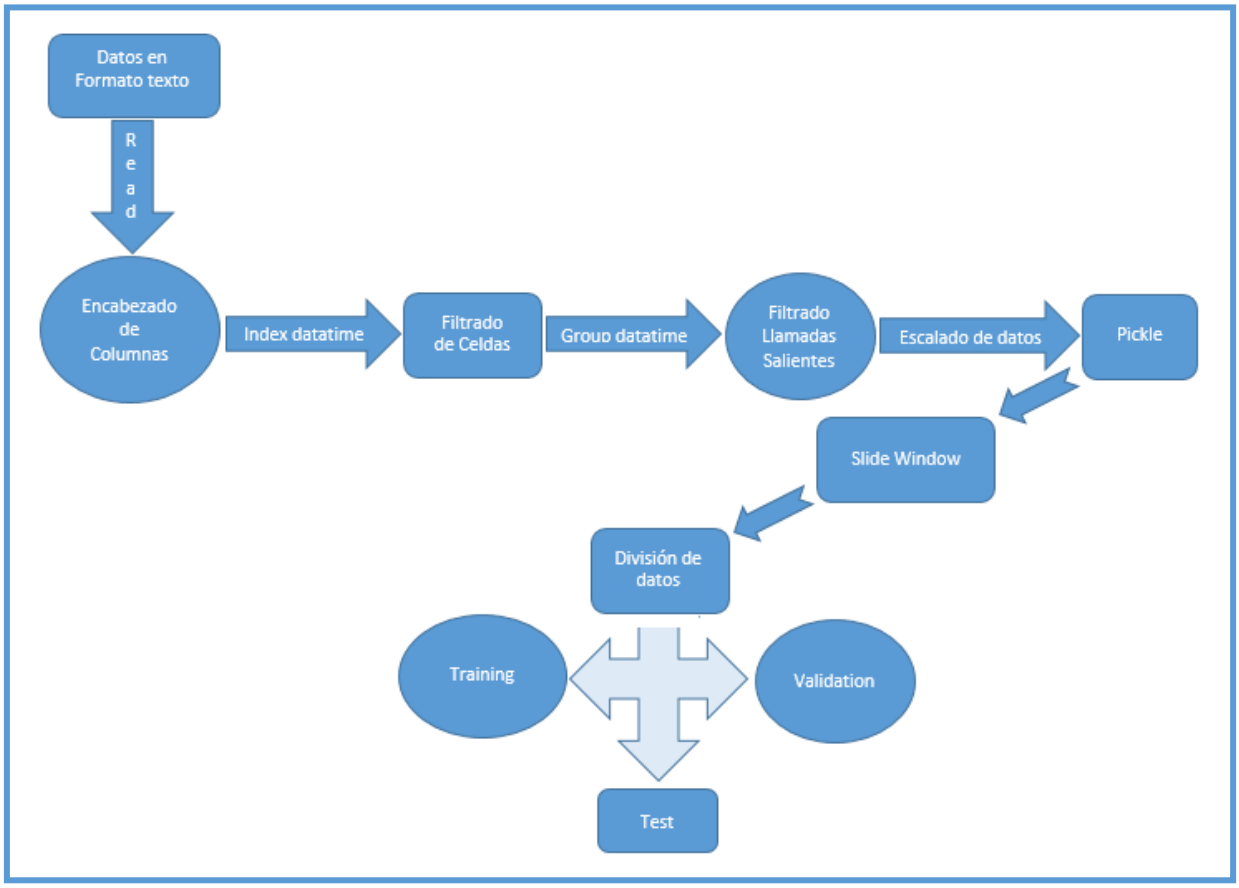


Figura 35. Tratamiento realizado a los datos

4.4 Modelo LSTM implementado

Los modelos LSMT que se han implementado constan de 1 a 8 capas con un número bloques LSTM igual al de la ventana deslizante, se han hecho pruebas con ventanas deslizantes con un número de muestras entre 48 y 192. Cada bloque LSTM cuenta con un número de nodos que van desde 8 a 64, según la prueba realizada en cada caso. En la figura 36 se muestra el modelo con capas LSTM apiladas.

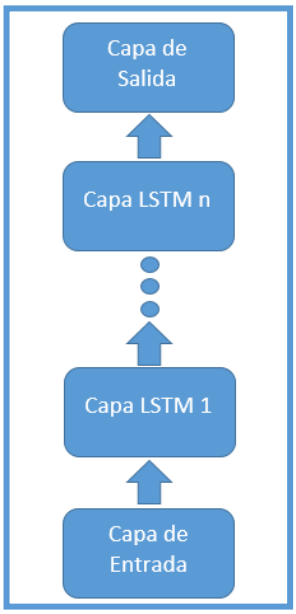


Figura 36. Stack LSTM

5. Resultados

Como hemos comentado anteriormente, los datos han sido divididos en tres partes. La última parte es la correspondiente a los test, donde se podrá medir realmente el rendimiento de la implementación llevada a cabo.

El modelo LSTM implementado va ser comparado con un Perceptrón multicapa, así podremos tener más claro el rendimiento de la red LSTM. Para optimizar la implementación, se han realizado numerosas pruebas variando los hyperparámetros, de forma que fuera posible conseguir unos valores óptimos haciendo uso de los datos de validación. Una vez conseguida una configuración óptima enfrentando la evolución del aprendizaje entre los datos de training y los datos de validación para un modelo de red, se ha hace la predicción con los datos de test, evaluando así, de forma real la red con datos con los cuales la red no ha sido entrenada, conociendo de esta forma el rendimiento real de la misma.

Las pruebas han sido realizadas con modelos de 1 a 8 capas, variado el número de épocas en un rango [15,80], un rango de nodos [8, 300], un rango de learning rate [0.0005, 0.002] y una ventana deslizante con un rango de [20, 192] muestras. El horizonte de predicción ha sido de 1 a 18 muestras vista, que corresponde a 10 y 180 minutos de predicción respectivamente.

Para conformar el dataset se han filtrado los datos generados por las 4 celdas primeras contiguas, esto provoca que los datos tengan una mayor varianza, que añade dificultad a la hora de predecir, resultando de este modo unas métricas menos favorables que sin aplicar este filtro de celdas. No obstante, una vez finalizadas las pruebas con las 4 celdas, se han hecho dos pruebas más adicionales, en este caso incluyendo todas las celdas de la fuente de datos original. Obteniendo así unas métricas mucho más favorables.

5.1 Métricas

En cuanto a las métricas para poder evaluar el modelo, están han sido las propias para un problema de regresión, el error cuadrático medio y el error cuadrático medio relativo.

5.1.1 Error cuadrático medio

El Error cuadrático medio (Mean Square Error, MSE) es un estimador que mide el promedio de los errores al cuadrado. En esta disciplina, el minuendo será el valor a predecir (estimador) y el sustraendo el valor predicho (valor estimado). Su cálculo se indica en la ecuación (11).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(target_i - output_i)^2 \quad (11)$$

5.1.2 Error cuadrático medio relativo

El error cuadrático medio relativo es el error cuadrático normalizado por el estimador. Su cálculo se muestra en la ecuación (12).

$$MSE_r = 100 * \frac{1}{n}\sum_{i=1}^{n}\frac{(target_i - output_i)^2}{target_i} \quad (12)$$

Su resultado se expresa en tanto por ciento, t %. El cual refleja la transcendencia del error cometido respecto al valor real, ya que este error relativo es un porcentaje del valor real.

Implementaciones con un MSE_r máximo del 8% tienen un buen funcionamiento en aplicaciones de predicción similares.

Por ejemplo, si un repetidor cursa un máximo de 1000 llamadas de forma habitual, el sistema de gestión automática puede activar el canal 3 cuando falte un 15% para llegar a este 100%, que son 850 llamadas. Aplicando la ecuación (13) podemos saber con qué número de llamadas reales se activará el canal 3.

$$Llamadas\ reales * 1.08 \geq \%Máx.llamadas\ habitutal \quad (13)$$

Para este ejemplo, el canal 3 se activará cuando el repetidor curse 788 llamadas reales. Este es un error despreciable para este tipo de aplicación que no impediría su correcto funcionamiento, ya que el canal se activaría 62 llamadas antes de las 850.

El canal 3 se desactivará cuando las llamadas estén un 30 % por debajo de su valor máximo habitual. En este ejemplo serían 700 llamadas. A partir de la ecuación (14) obtenemos que el canal 3 se desactivará cuando por el repetidor se curse menos de 648 llamadas.

$$Llamadas\ reales * 1.08 < \%Máx.llamadas\ habitutal \quad (14)$$

5.3 Pruebas realizadas

En la tabla 4 se muestran los resultados obtenidos para 45 configuraciones de red diferentes. Se marcan en verde los mejores resultados para cada horizonte de predicción. Para ello, se ha tenido en cuenta las métricas utilizadas. En el caso de que dos redes obtengan métricas similares, se ha escogido la red que muestra una mejor evolución de aprendizaje.

Prueba	Red	Muestras vista	Capas	Nodos	Ventana	Batch	Epochs	L Rate	MSE	MSE_r(%)
1	LSTM	1	1	32	48	96	15	0,001	0,12	5,77
2	LSTM	4	1	32	96	192	70	0,001	0,13	8,76
3	LSTM	4	2	32	84	168	55	0,001	0,13	5,33
4	LSTM	4	1	16	96	192	70	0,001	0,15	9,74
5	LSTM	4	1	64	192	384	70	0,001	0,14	9,94
6	LSTM	4	3	16	48	96	70	0,002	0,14	6,37
7	LSTM	4	3	32	96	192	70	0,002	0,14	9,41
8	LSTM	6	1	32	84	96	70	0,001	0,14	5,9
9	LSTM	6	1	32	144	288	70	0,002	0,15	6,86
10	LSTM	6	2	8	48	96	70	0,001	0,17	8,92
11	LSTM	6	2	16	96	192	70	0,001	0,16	12,32
12	LSTM	6	2	32	48	96	70	0,001	0,15	6,58
13	LSTM	6	2	64	48	96	70	0,001	0,15	7,83
14	LSTM	6	2	64	48	96	80	0,0009	0,15	6,92
15	LSTM	6	3	64	48	96	80	0,0009	0,16	8,99
16	LSTM	6	4	32	84	168	55	0,001	0,15	7,36
17	LSTM	6	5	32	48	96	40	0,001	0,15	7,83
18	LSTM	12	1	32	48	96	70	0,001	0,17	9,14
19	LSTM	12	2	32	48	96	70	0,001	0,18	9,56
20	LSTM	12	2	16	48	96	70	0,001	0,17	9,12
21	LSTM	12	2	16	72	144	70	0,001	0,16	12,19
22	LSTM	12	2	16	74	148	70	0,001	0,16	9,58
23	LSTM	12	2	32	74	148	70	0,001	0,16	8,39
24	LSTM	12	3	32	74	148	70	0,001	0,16	10,15
25	LSTM	12	1	32	74	148	70	0,001	0,16	8,4
26	LSTM	12	1	32	84	168	70	0,001	0,16	6,92
27	LSTM	12	8	32	74	148	70	0,001	0,17	8,21
28	LSTM	12	2	32	84	168	70	0,001	0,16	6,77
29	LSTM	12	2	64	84	168	70	0,0009	0,16	6,94
30	LSTM	12	3	64	84	168	70	0,001	0,17	7,62
31	LSTM	18	2	32	84	168	70	0,001	0,17	7,78
32	LSTM	18	1	32	84	168	70	0,001	0,19	7,47
33	LSTM	18	3	32	84	168	70	0,001	0,18	8,7
34	LSTM	18	3	64	84	168	70	0,0005	0,17	7,12
35	Perceptrón	1	1	100	50	100	50	0,001	0,14	5,54
36	Perceptrón	1	1	100	50	100	50	0,001	0,14	5,54
37	Perceptrón	1	1	250	50	100	50	0,001	0,26	14,72
38	Perceptrón	4	2	300	50	100	25	0,001	0,2	7,02
39	Perceptrón	4	1	100	50	100	50	0,001	0,22	7,56
40	Perceptrón	6	2	300	50	100	15	0,001	0,27	8,82
41	Perceptrón	6	3	100	50	100	50	0,001	0,31	8,93
42	Perceptrón	12	2	200	50	100	45	0,0007	0,38	14,32
43	Perceptrón	12	2	250	50	100	45	0,0009	0,41	15,02
44	Perceptrón	18	2	300	20	40	50	0,0005	1.7	87.9
45	Perceptrón	18	2	400	20	40	50	0,0004	2.14	84.06

Tabla 4. Resultados obtenidos para todas las configuraciones de red probadas

En la tabla 5 se muestra como resumen las mejores configuraciones de red de la tabla 4. De las cuales se compararán a continuación los modelos de red LSTM con el Perceptrón, mostrando las gráficas con su evolución de aprendizaje y predicción.

Prueba	Red	Muestras vista	Capas	Nodos	Ventana	Batch	Epochs	L Rate	MSE	MSE_r(%)
1	LSTM	1	1	32	48	96	15	0,001	0,12	5,77
3	LSTM	4	2	32	84	168	55	0,001	0,13	5,33
8	LSTM	6	1	32	84	96	70	0,001	0,14	5,9
29	LSTM	12	2	64	84	168	70	0,0009	0,16	6,94
34	LSTM	18	3	64	84	168	70	0,0005	0,17	7,12
36	Perceptrón	1	1	100	50	100	50	0,001	0,14	5,54
38	Perceptrón	4	2	300	50	100	25	0,001	0,2	7,02
40	Perceptrón	6	2	300	50	100	15	0,001	0,27	8,82
42	Perceptrón	12	2	200	50	100	45	0,0007	0,38	14,32
44	Perceptrón	18	2	300	20	40	50	0,0005	1.7	87.9

Tabla 5. Mejores resultados obtenidos para todas las configuraciones de red probadas

Horizonte de predicción a 1 muestra vista

Comparando las redes de la tabla 6, vemos que ambas tienen un buen rendimiento en general, sin embargo, para una sola muestra vista, ya se ve como la predicción de la red LSTM muestra un mejor comportamiento.

Prueba	Red	Muestras vista	Capas	Nodos	Ventana	Batch	Epochs	L Rate	MSE	MSE_r(%)
1	LSTM	1	1	32	48	96	15	0,001	0,12	5,77
36	Perceptrón	1	1	100	50	100	50	0,001	0,14	5,54

Tabla 6. Resultados obtenidos para un horizonte de predicción de 1 muestra vista

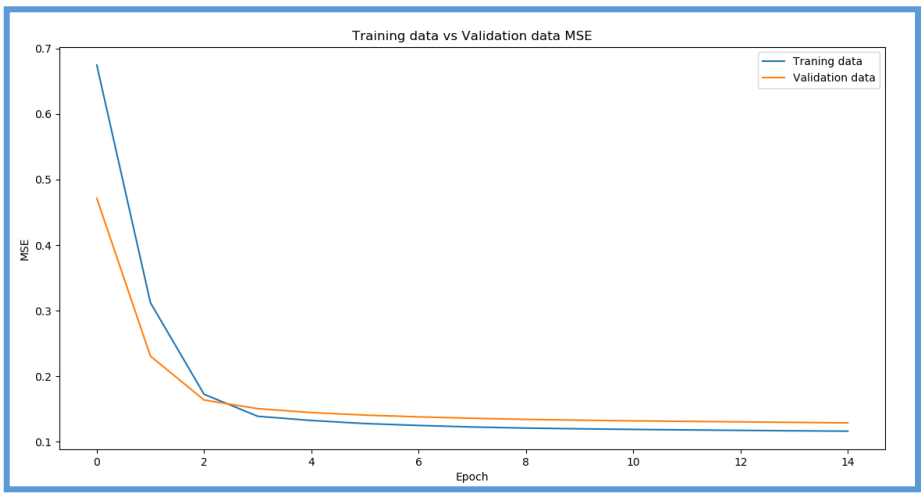


Figura 37. Evolución del MSE para la red LSTM correspondiente a la prueba nº 1

La figura 37 muestra la evolución del MSE de los datos de entrenamiento frente a los datos de validación, dicha evolución no muestra overfitting en el aprendizaje; mientras que la figura 38 ilustra la predicción de la red LSTM, ambas correspondientes a la prueba nº 1.

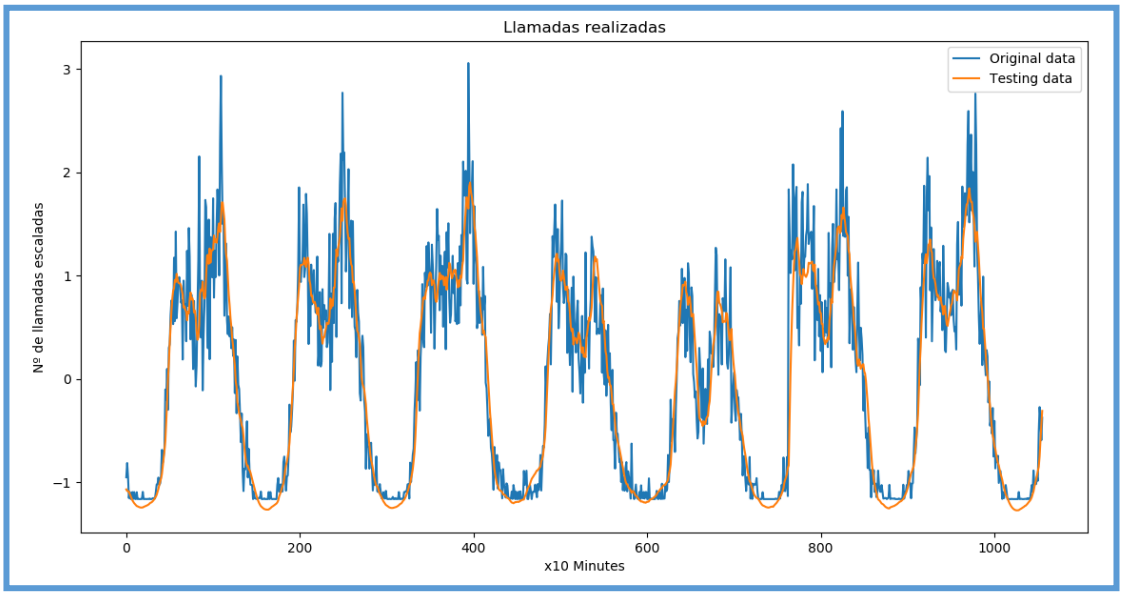


Figura 38. Predicción llamadas salientes para la red LSTM correspondiente a la prueba nº 1

Por otra parte, las figuras 39 y 40 muestran las correspondientes pruebas al igual que las anteriores figuras, pero en este caso para la red implementada con el Perceptrón de la prueba nº 36.

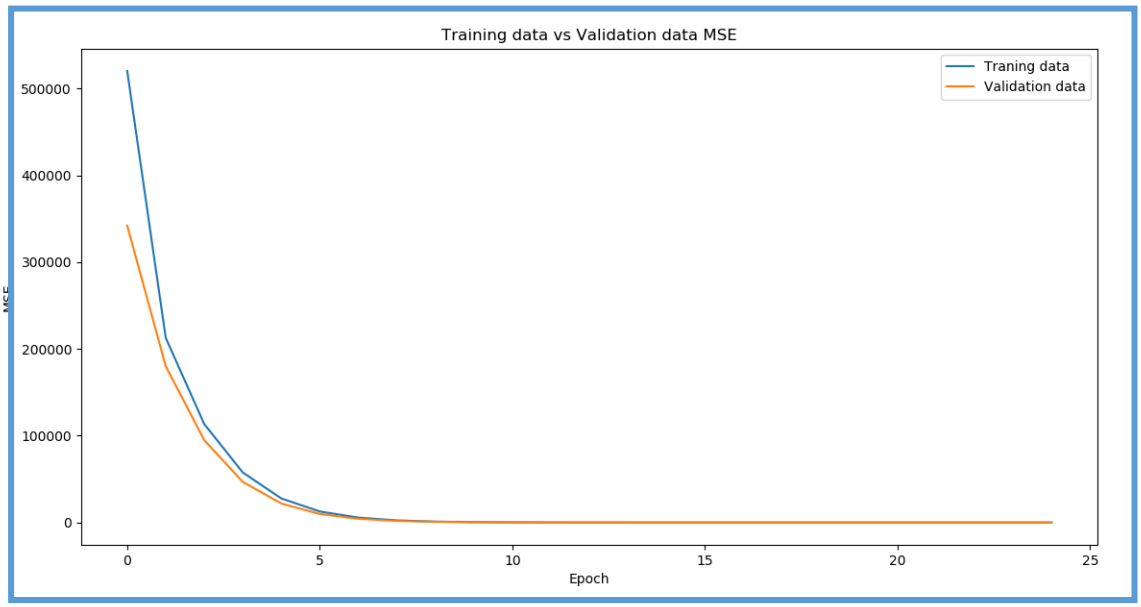


Figura 39. Evolución del MSE para el Perceptrón correspondiente a la prueba nº 36

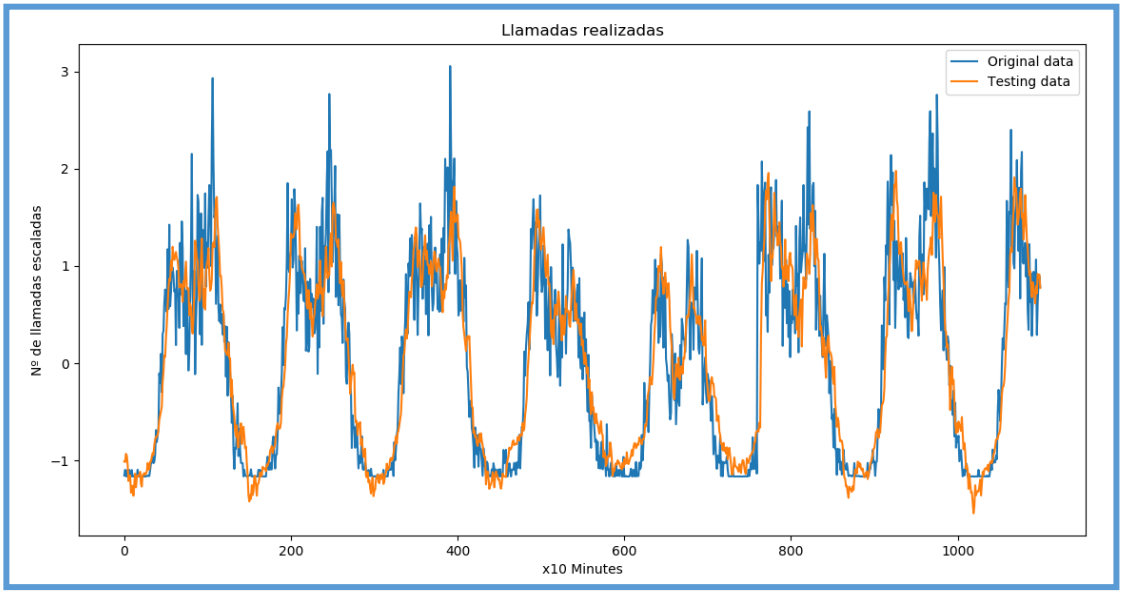


Figura 40. Predicción llamadas salientes para el Perceptrón correspondiente a la prueba nº 36

En la figura 40 se aprecia como la predicción de Perceptrón no es suave como la de la red LSTM.

Horizonte de predicción a 4 muestras vista

En la tabla 7 vemos que para este horizonte de predicción sigue obteniendo un buen rendimiento. Sin embargo, el MSE correspondiente al Perceptrón empieza ser más elevado de lo deseado.

Prueba	Red	Muestras vista	Capas	Nodos	Ventana	Batch	Epochs	L Rate	MSE	$MSE_r(\%)$
3	LSTM	4	2	32	84	168	55	0,001	0,13	5,33
38	Perceptrón	4	2	300	50	100	25	0,001	0,2	7,02

Tabla 7. Resultados obtenidos para un horizonte de predicción de 4 muestras vista

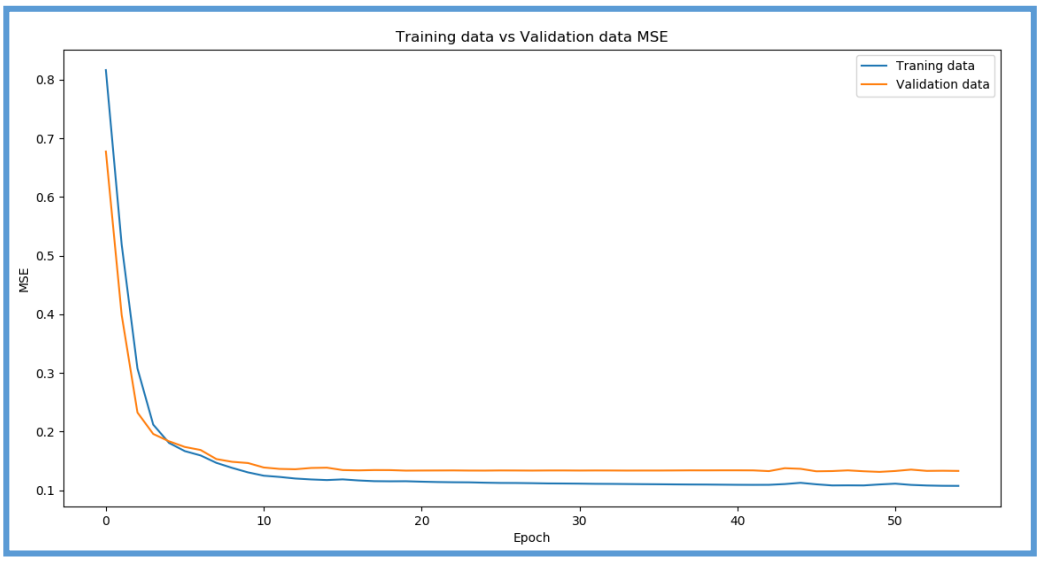


Figura 41. Evolución del MSE para la red LSTM correspondiente a la prueba nº 3

La evolución del MSE correspondiente a los datos de entrenamiento frente a los datos de validación es mostrada en la figura 41, donde no se ven muestras de overfitting; así como la figura 42 ilustra la predicción de la red LSTM, ambas figuras correspondientes a la prueba nº 3.

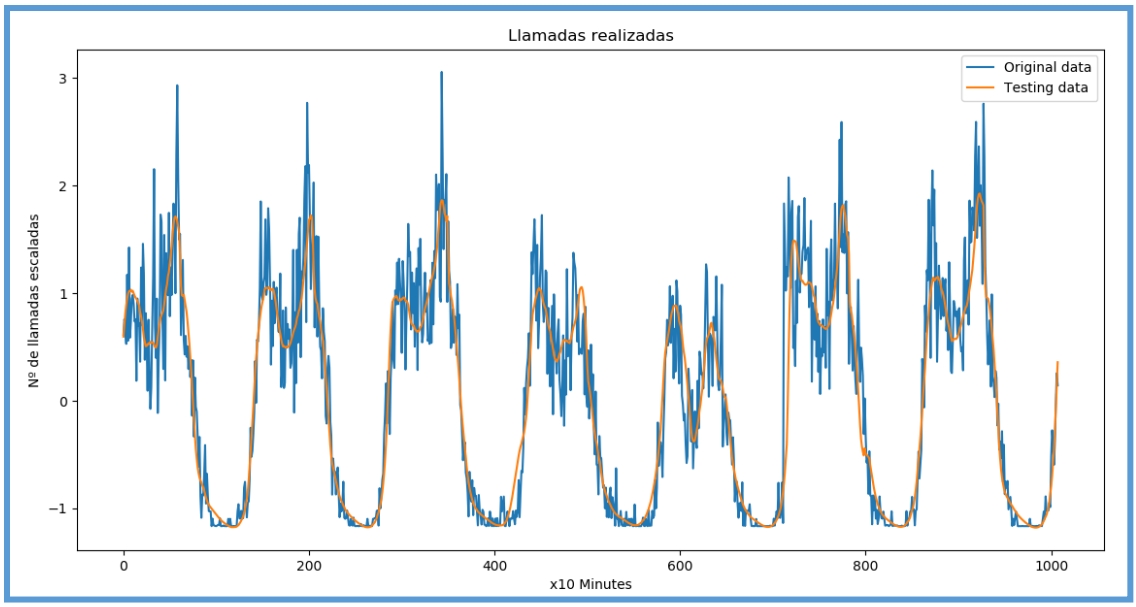


Figura 42. Predicción llamadas salientes para la red LSTM correspondiente a la prueba nº 3

De la misma manera, las figuras 43 y 44 muestran la evolución del MSE y la predicción respectivamente para el Perceptrón de la prueba nº38.

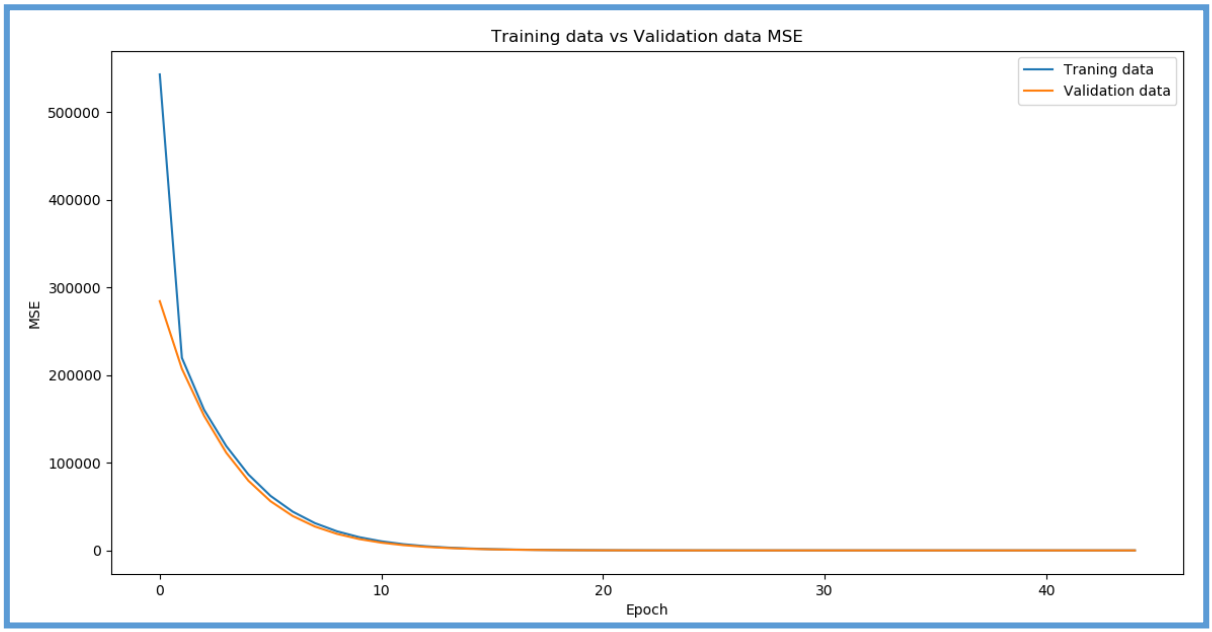


Figura 43. Evolución del MSE para el Perceptrón correspondiente a la prueba nº 38

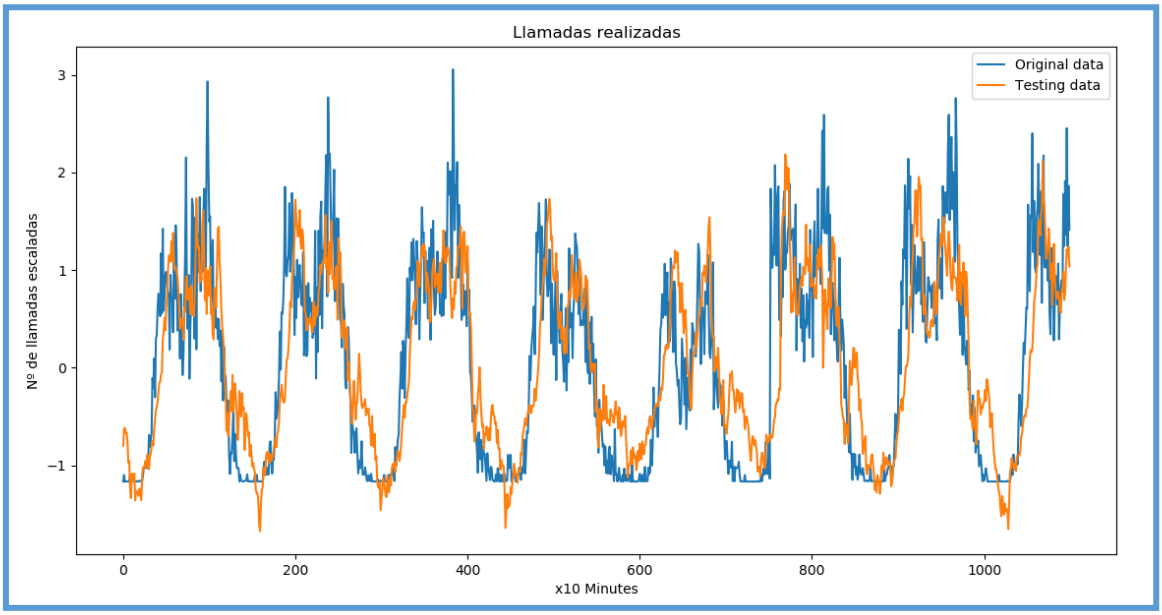


Figura 44. Predicción llamadas salientes para el Perceptrón correspondiente a la prueba nº 38

Vemos que las gráficas correspondientes a la red LSTM siguen teniendo un buen rendimiento, la figura 43 ilustra como la predicción sigue con suavidad a los datos originales, algo propio de una red recurrente. No es el caso de la figura 44, donde vemos como el rendimiento de predicción del Perceptrón ha caído para 4 muestras vista, sin embargo, el aprendizaje del Perceptrón no presenta overfitting.

Horizonte de predicción a 6 muestras vista

Comparando las redes de la tabla 8, vemos que el rendimiento del Perceptrón empieza a caer. No es el caso de la red LSTM, esta sigue teniendo un rendimiento considerablemente bueno.

Prueba	Red	Muestras vista	Capas	Nodos	Ventana	Batch	Epochs	L Rate	MSE	$MSE_r(\%)$
8	LSTM	6	1	32	84	96	70	0,001	0,14	5,9
40	Perceptrón	6	2	300	50	100	15	0,001	0,27	8,82

Tabla 8. Resultados obtenidos para un horizonte de predicción de 6 muestras vista

Los resultados visuales correspondientes a la prueba nº 8 se pueden ver en las figuras 45 y 46, donde vemos que la red LSTM es capaz de predecir 1 hora. La evolución del aprendizaje no presenta overfiting ni oscilaciones, así como la gráfica de la predicción sigue perfectamente a los datos originales.

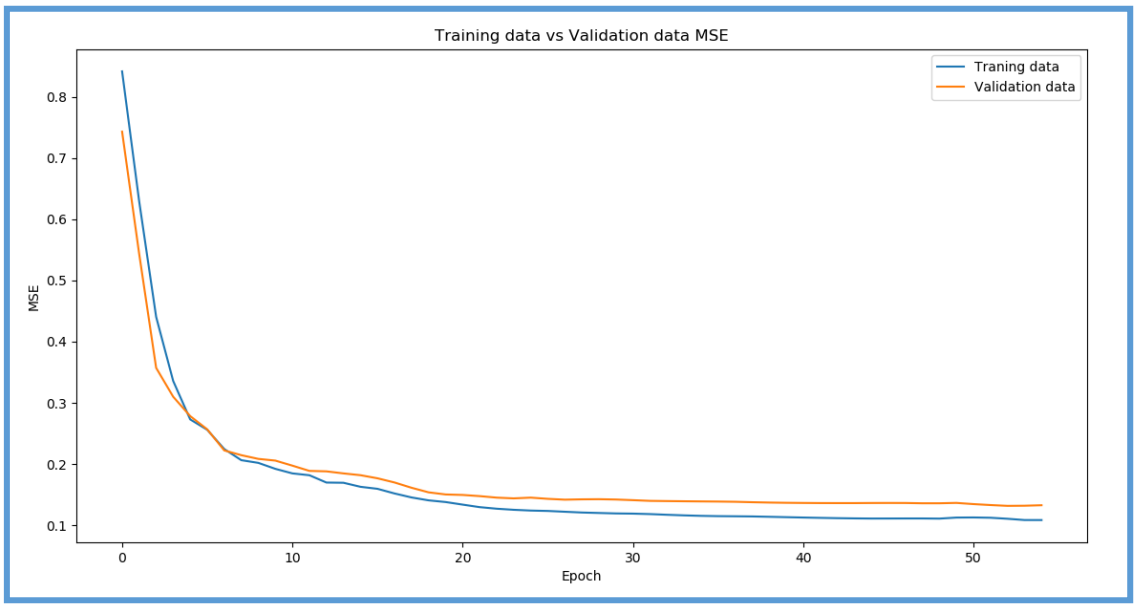


Figura 45. Evolución del MSE para la red LSTM correspondiente a la prueba nº 8

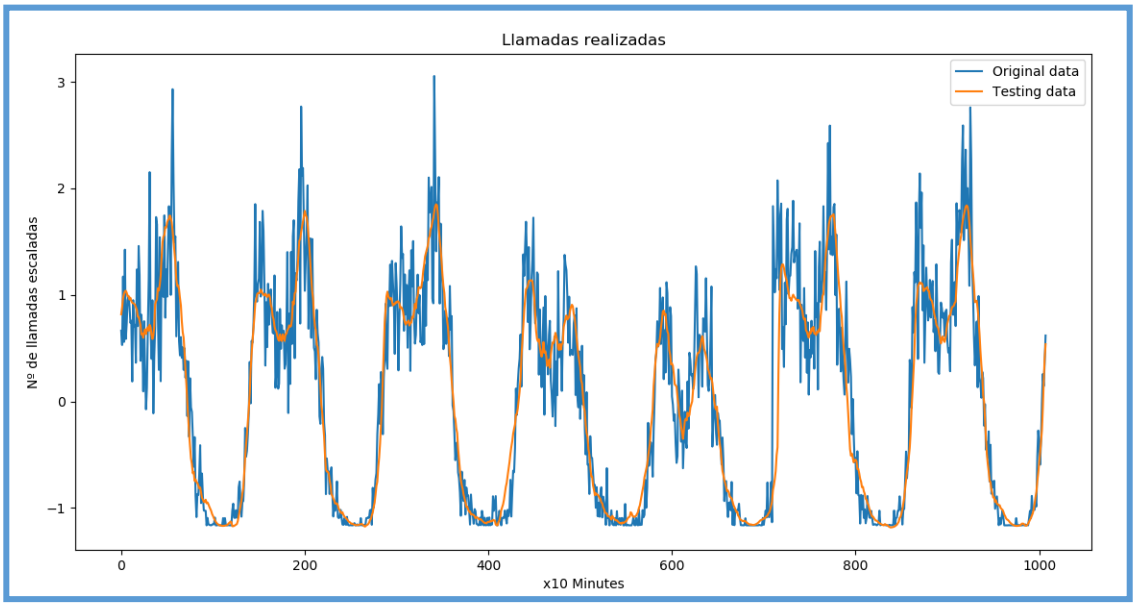


Figura 46. Predicción llamadas salientes para la red LSTM correspondiente a la prueba nº 8

Por su parte, la red implementada con el Perceptrón sigue perdiendo rendimiento según se incrementa el horizonte de predicción. Lo podemos observar en la figura 47 con un MSE de 0.27.

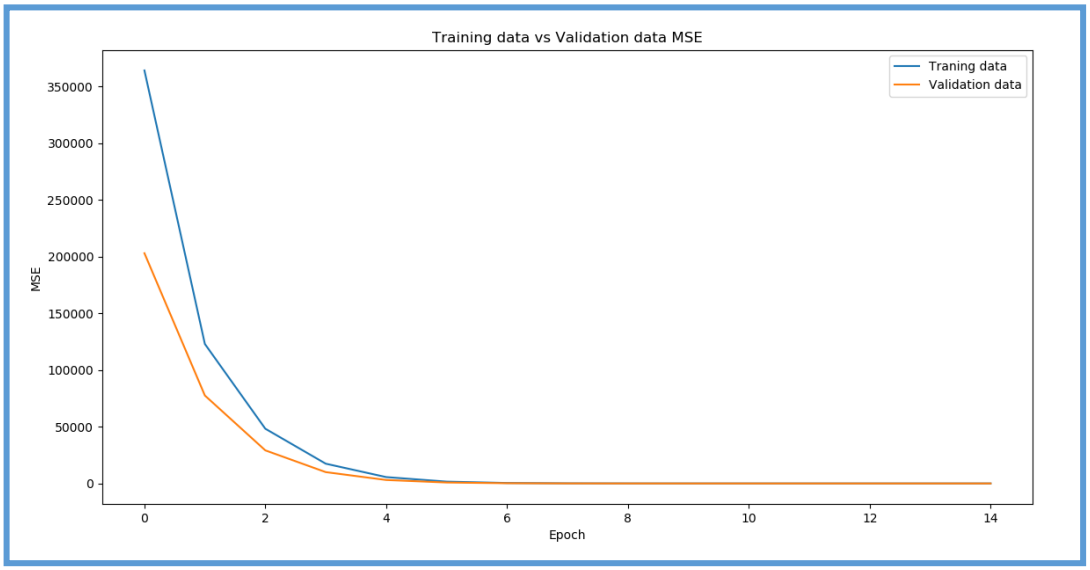


Figura 47. Evolución del MSE para el Perceptrón correspondiente a la prueba nº 40

Como podemos observar en la figura 48, la predicción realizada se parece cada vez menos a las llamadas salientes realizadas originales.

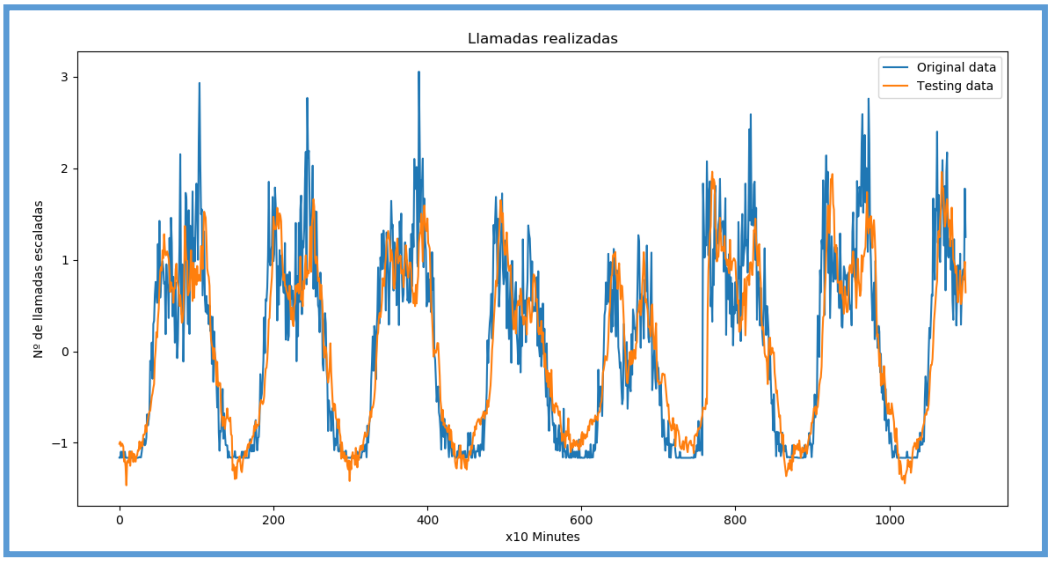


Figura 48. Predicción llamadas salientes para el Perceptrón correspondiente a la prueba nº 40

Horizonte de predicción a 12 muestras vista

En este caso se ha aumentado el horizonte de predicción a dos horas. En la tabla 9 vemos que la red LSTM sigue prediciendo con unas métricas acordes a las de una red recurrente. No es el caso de Perceptrón, donde se resalta que ya ha perdido su capacidad de memoria.

Prueba	Red	Muestras vista	Capas	Nodos	Ventana	Batch	Epochs	L Rate	MSE	$MSE_r(\%)$
29	LSTM	12	2	64	84	168	70	0,0009	0,16	6,94
42	Perceptrón	12	2	200	50	100	45	0,0007	0,38	14,32

Tabla 9. Resultados obtenidos para un horizonte de predicción de 12 muestras vista

En la figura 49 se muestra la evolución de aprendizaje de la red LSTM para 2 horas de predicción. Observamos que comienzan pequeñas oscilaciones, sobre todo en los datos de validación, donde el MSE ya sube hasta 0.16. Para conseguir esta evolución de aprendizaje se tuvieron que probar diferentes configuraciones de red.

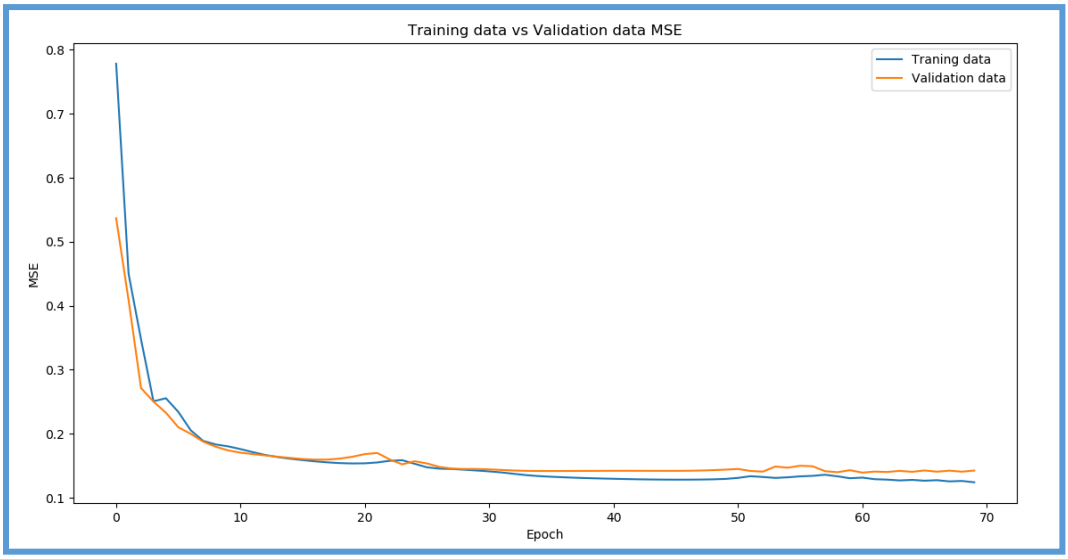


Figura 49. Evolución del MSE para la red LSTM correspondiente a la prueba nº 29

Por otra parte, vemos que el número de llamadas predichas sigue al número de llamadas originales, aunque ya se ven indicios de una pérdida de rendimiento, se puede ver que a la altura de los 720 minutos en la figura 50. No obstante, el modelo LSTM se mantiene con unas métricas aceptables para la aplicación propuesta sobre la gestión de frecuencias en repetidores FSR.

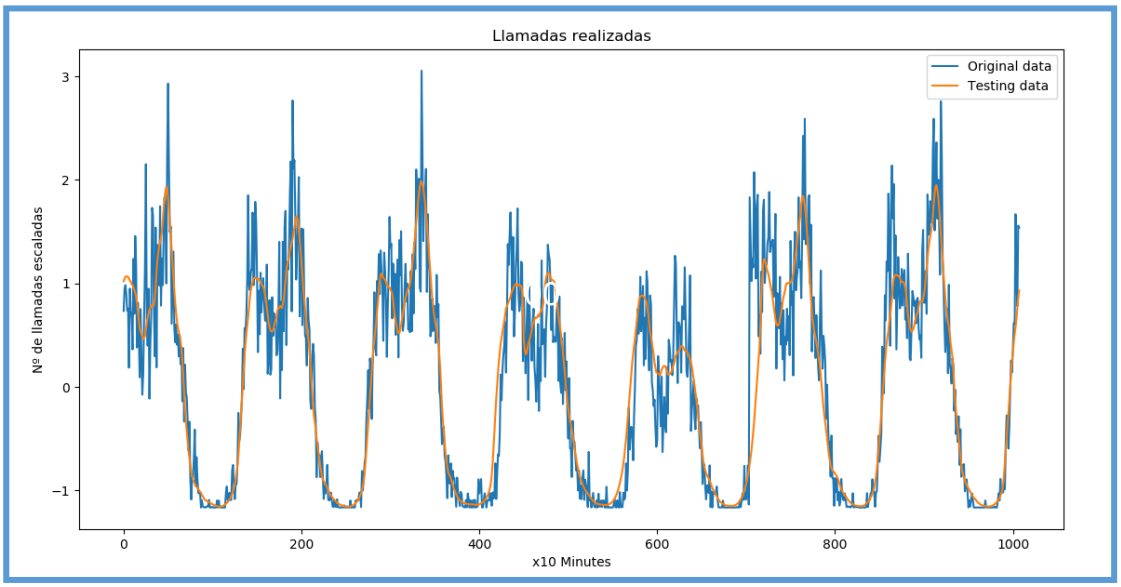


Figura 50. Predicción llamadas salientes para la red LSTM correspondiente a la prueba nº 29

Como era de esperar, las pruebas realizadas para este horizonte de predicción en el Perceptrón, continúan un acentuándose unas pérdidas habituales para este tipo de red en un problema de predicción. Lo podemos ver en la figura 51, donde el MSE alcanza ya un valor de 0.38.

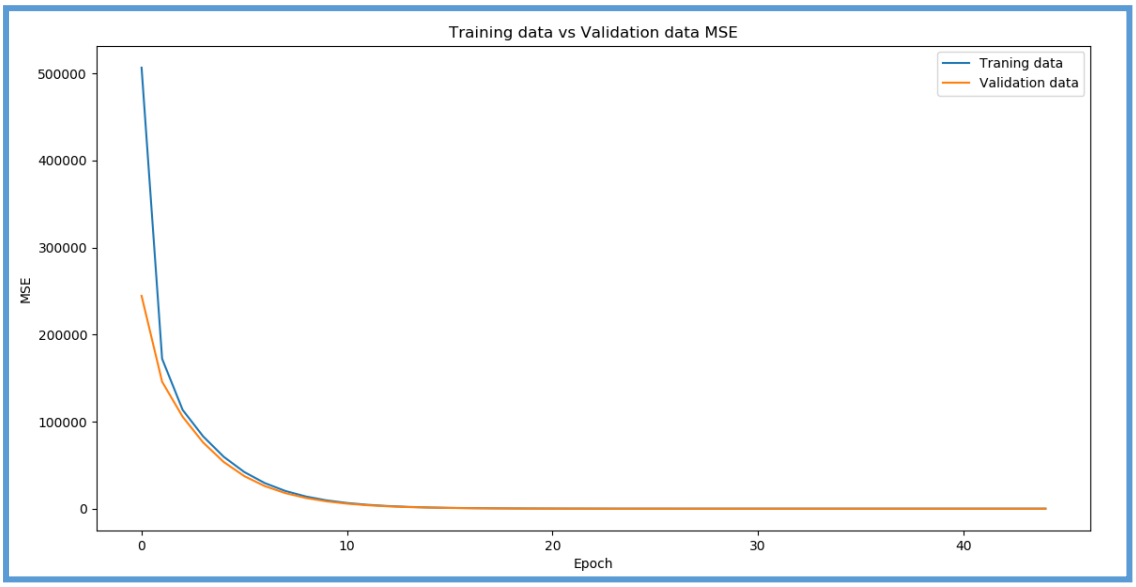


Figura 51. Predicción llamadas salientes para el Perceptrón correspondiente a la prueba nº 42

Dadas las métricas desfavorables de la prueba 42, observamos en la figura 52 que la predicción realizada no es capaz de seguir a las llamadas realizadas originales.

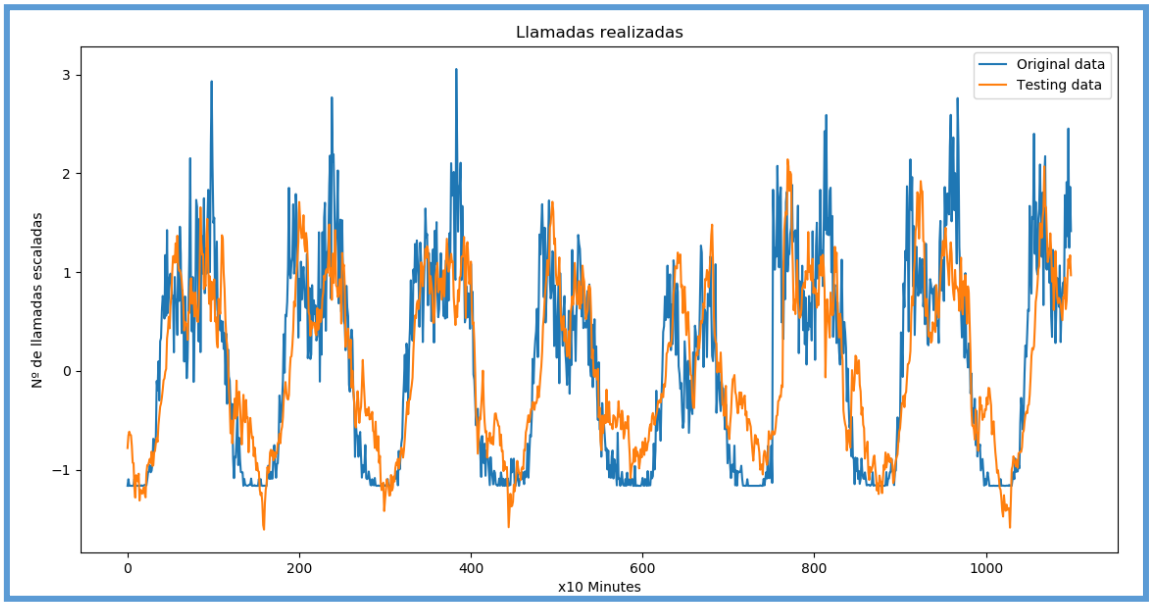


Figura 52. Predicción llamadas salientes para el Perceptrón correspondiente a la prueba nº 42

Horizonte de predicción a 18 muestras vista

En la tabla 10 se presentan las métricas para 3 horas de predicción. Vemos que el MSE de la red LSTM ya sube hasta 0.17, siendo un valor ya a tener en cuenta, su MSE relativo se mantiene por debajo del 10%. No es el caso del Perceptrón donde las métricas son muy elevadas.

Prueba	Red	Muestras vista	Capas	Nodos	Ventana	Batch	Epochs	L Rate	MSE	$MSE_r(\%)$
34	LSTM	18	3	64	84	168	70	0,0005	0,17	7,12
44	Perceptrón	18	2	300	20	40	50	0,0005	1,7	87,9

Tabla 10. Resultados obtenidos para un horizonte de predicción de 18 muestras vista

Mostramos a continuación la figura 53, donde se aprecia una evolución de aprendizaje sin oscilaciones gracias a una bajada del learning rate, aunque el MSE ya ha subido hasta un 0.17.

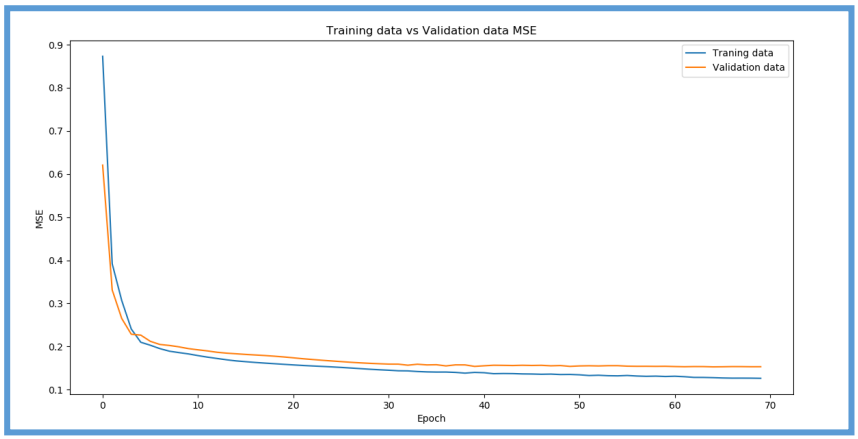


Figura 53. Evolución del MSE para la red LSTM correspondiente a la prueba nº 34

En la figura 54 vemos como ha bajado el rendimiento de la red LSTM, sobresale un pico de las llamadas predichas a la altura del minuto 450, así como se pronuncia más la pérdida de rendimiento en el minuto 720. No obstante, en términos generales, la predicción sigue siendo buena.

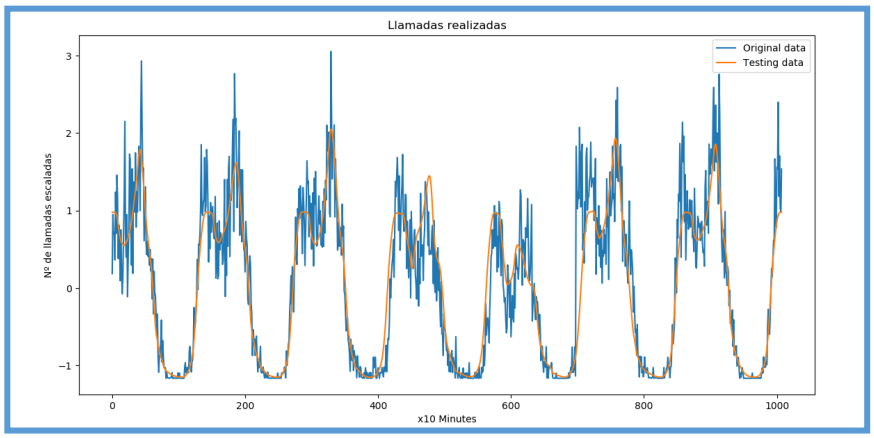


Figura 54. Predicción llamadas salientes para la red LSTM correspondiente a la prueba nº 34

Por su parte, el Perceptrón ha obtenido un MSE de 1.7, lo podemos ver la figura 55.

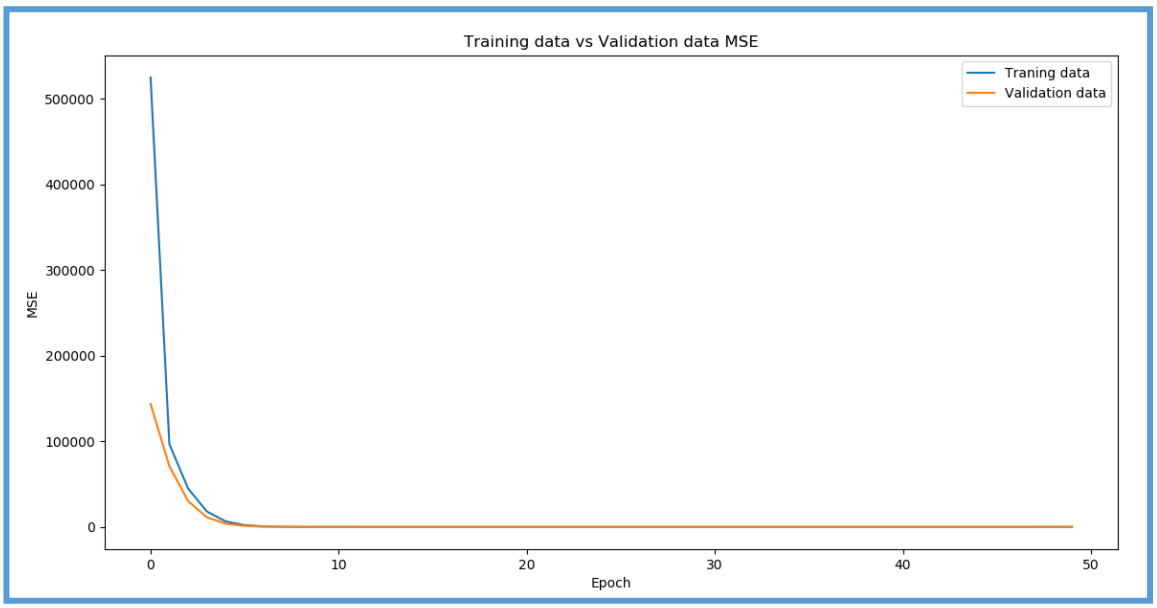


Figura 55. Predicción llamadas salientes para el Perceptrón correspondiente a la prueba nº 44

En la figura 56 vemos como la predicción está totalmente desfasada respecto a los datos originales.

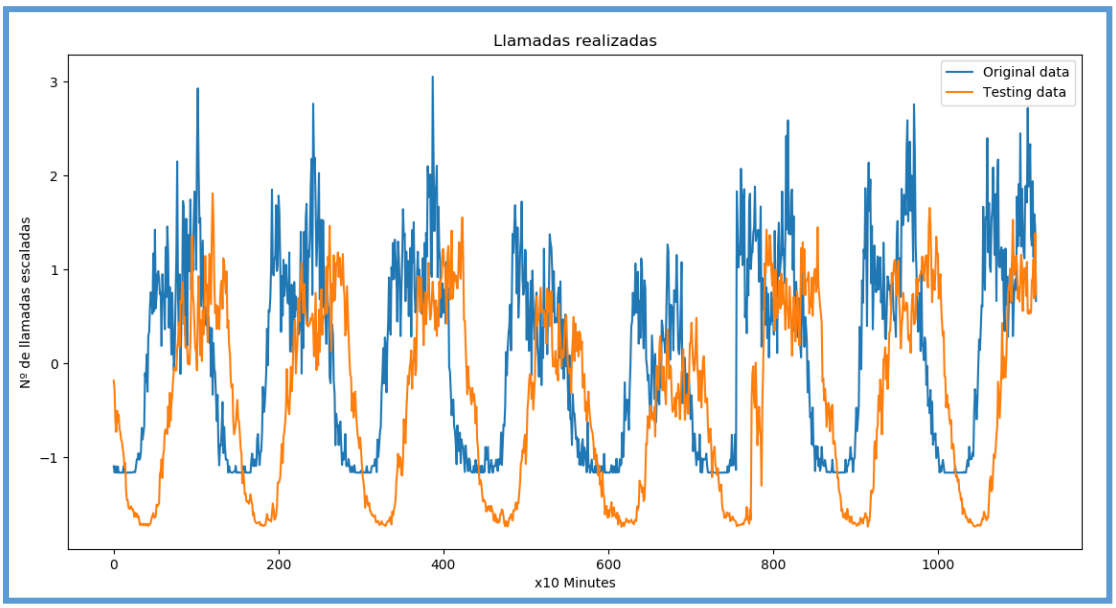


Figura 56. Predicción llamadas salientes para el Perceptrón correspondiente a la prueba nº 44

Horizonte de predicción a 1 muestras vista- - Todas las celdas incluidas

Como ya comentado anteriormente, los datos fueron filtrados agrupando 4 celdas para cubrir un área de 470 m^2, de esta forma se tiene un área aproximada a la cubierta por una estación base.
No obstante, se ha querido hacer una prueba sin filtrar por celdas, de esta forma se verá la capacidad de predicción de la red LSTM para todo el volumen de llamadas salientes de la ciudad de Milán.

En la tabla 11 vemos como las métricas han mejorado considerablemente, en concreto las correspondientes al MSE, las cuales se han reducido en dos órdenes de magnitud.

Red	Muestras vista	Capas	Nodos	Ventana	Batch	Epochs	L Rate	MSE	$MSE_r(\%)$
LSTM	1	2	64	84	168	50	0,001	0,0019	0,1582

Tabla 11. Resultados obtenidos para un horizonte de predicción de 1 muestra vista para toda el área de la ciudad de Milán

En las figuras 57 y 58 se observa que el rendimiento con este formato de dataset está muy por encima del anterior.

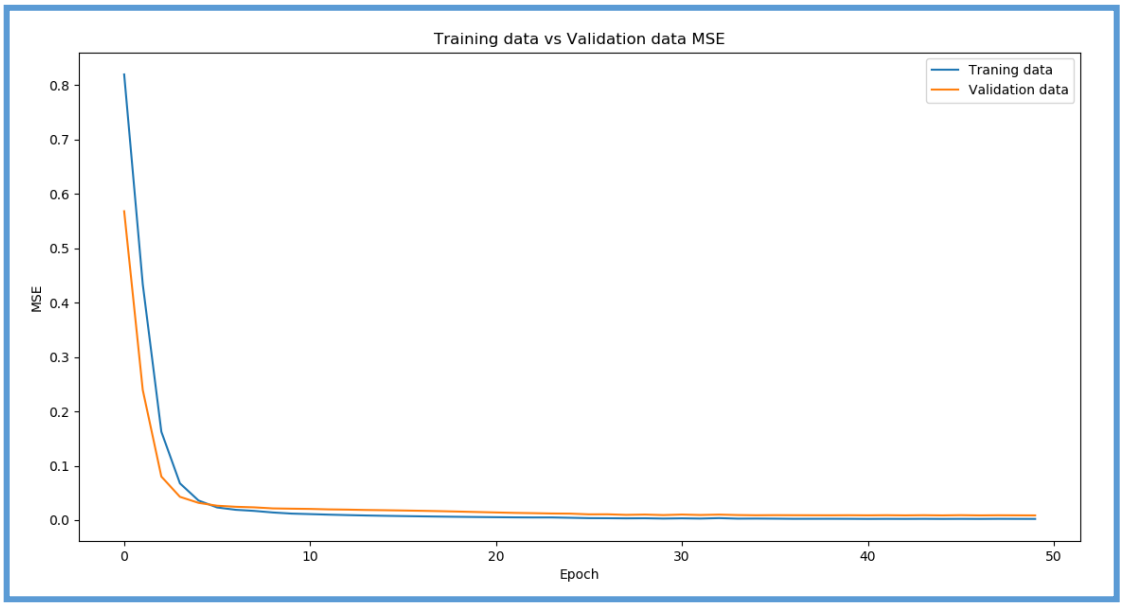


Figura 57. Evolución del MSE para la red LSTM correspondiente a todas las celdas de la ciudad de Milán: 1 muestra vista

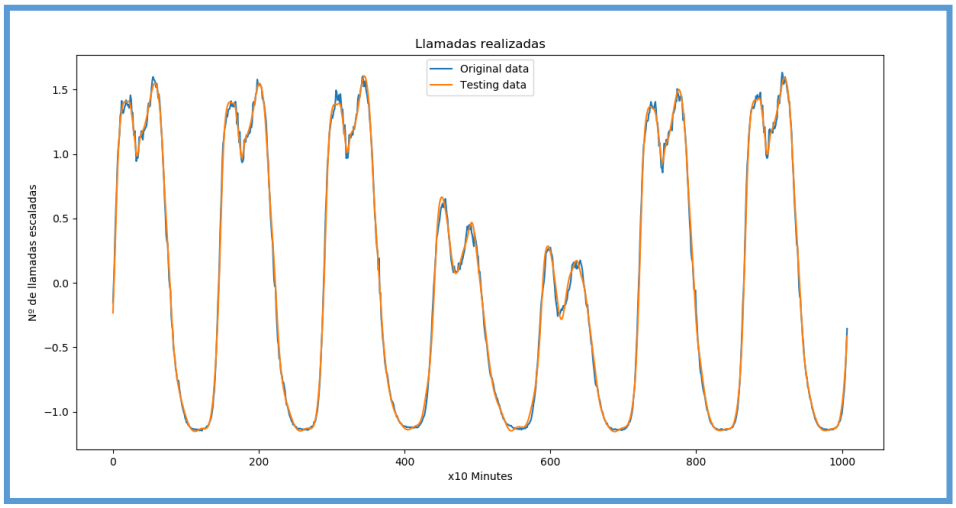


Figura 58. Predicción llamadas salientes para la red LSTM correspondiente a todas las celdas de la ciudad de Milán: 1 muestra vista

La figura 58 muestra los datos originales con todas las celdas no son muy variantes, por lo que la red LSTM es capaz de hacer una predicción muy similar a la original.

Horizonte de predicción a 18 muestras vista - Todas las celdas incluidas

La tabla 12 muestra los resultados obtenidos con este tipo de dataset para el horizonte de predicción máximo probado anteriormente.

Red	Muestras vista	Capas	Nodos	Ventana	Batch	Epochs	L Rate	MSE	$MSE_r(\%)$
LSTM	18	2	64	84	168	70	0,001	0,0290	2,5896

Tabla 12. Resultados obtenidos para un horizonte de predicción de 18 muestras vista para toda el área de la ciudad de Milán

De esta forma, se demuestra que con esta configuración se consigue un mejor rendimiento que la red LSTM estudiada en el estado del arte. Además, se ha conseguido con una predicción de 3 horas.

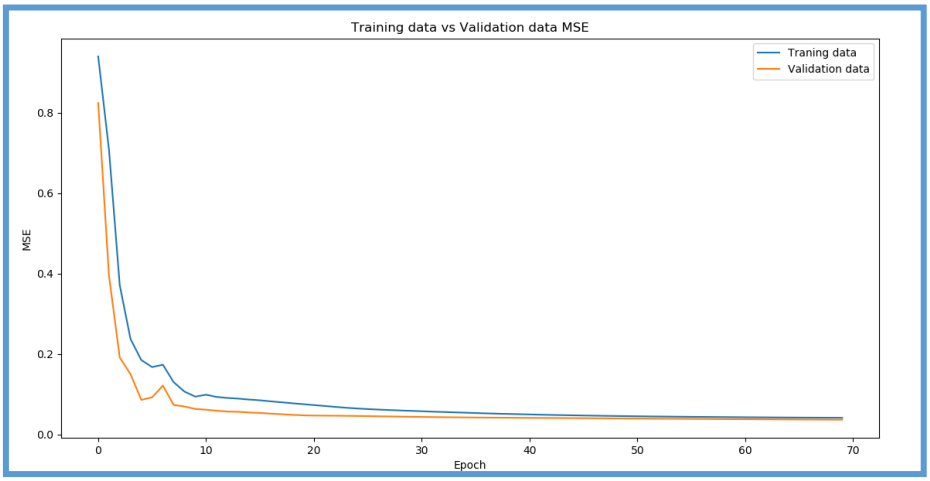


Figura 59. Evolución del MSE para la red LSTM correspondiente a todas las celdas de la ciudad de Milán: 18 muestras vista

En la figura 59 se observa que aún con este horizonte de predicción, se obtiene un MSE muy bajo. Alrededor del minuto 5 hay unas oscilaciones, estas oscilaciones se anularían reduciendo el learning rate y ampliado un poco el número de épocas de entrenamiento. No obstante, la evolución de aprendizaje no muestra overfitting.

En las figuras 58 y 59 se observa que aún con este horizonte de predicción el rendimiento con este formato de dataset sigue mostrando unas métricas excelentes.

Elección de la mejor configuración

Las pruebas realizadas con la implementación del Perceptrón multicapa van a ser descartadas dado que no consiguen buenas métricas. Para un horizonte de predicción entre 1 y 4 muestras vista, el rendimiento de este tipo de red si cumple las métricas, pero usar la aplicación para 1 muestra vista no sería una buena opción ya que solo se predicen 10 minutos, por lo que no es una opción viable. Para 4 muestras vista se tiene un MSE de 0.2, que ya es algo elevado junto a que 40 minutos de predicción sigue siendo un horizonte de predicción escaso y podrían perder algunas llamadas.

Para la red LSTM, van a ser válidas para esta aplicación las pruebas realizadas con un horizonte de predicción igual o superior a 6 muestras vista que estén que estén por debajo del 8 % del MRE_r. Des estas configuraciones, cualquiera sería válida para la aplicación motivadora. Por lo tanto, se escogerá la que tenga un mayor horizonte de predicción, que corresponde con la prueba número 34 (sombreada en verde), esta predicción es equivalente a 3 horas (18 x 10 minutos/muestra). Estas configuraciones de red LSTM se muestran en la tabla 13.

Prueba	Red	Muestras vista	Capas	Nodos	Ventana	Batch	Epochs	L Rate	MSE	$MSE_r(\%)$
8	LSTM	6	1	32	84	96	70	0,001	0,14	5,9
29	LSTM	12	2	64	84	168	70	0,0009	0,16	6,94
34	LSTM	18	3	64	84	168	70	0,0005	0,17	7,12

Tabla 13. Configuraciones de red LSTM válidas para la aplicación de gestión de bandas de frecuencia. En verde la configuración escogida para la aplicación.

Siguiendo con el ejemplo del repetidor que curse 1000 llamadas máximo de forma habitual, se muestra en la tabla 14 un resumen de la conmutación del canal 3 para 3 horas de predicción según un MSE_r del 7.12 %.

Acción	Máx. llamadas	Umbral llamadas (%)	Predicción	$MSE_r(\%)$	Llamadas error	Llamadas real
Act. Ch. 3	1000	≥ 85	3 h.	7,12	56	794
Des. Ch. 3	1000	< 70	3 h.	7,12	47	653

Tabla 14. Conmutación del canal 3 en NMS

En la anterior tabla vemos que, si el sistema de predicción indica que hay un número de llamadas mayor o igual al 85 % del máximo de las llamadas cursadas habituales, el canal 3 se activará 3 horas antes. En este caso, el canal 3 conmutará a estado activado cuando el repetidor curse 794 llamadas reales. De la misma forma, el canal 3 se desactivará cuando el sistema de predicción indique que hay un número de llamadas cursadas que sea menor al 70% del máximo de las llamadas cursadas habituales. En este caso, el canal 3 conmutará a estado desactivado cuando el repetidor curse 653 llamadas reales. Para los cálculos se han utilizado las ecuaciones (13) y (14), pero en este caso sustituyendo 1.08 por 1.0712 (8 %, 7.12 %). A continuación se muestra el cálculo para las llamadas reales con las que se activará el canal 3. Para la desactivación se procede de forma similar.

$$Llamadas\ reales \leq \frac{850}{1.0712}$$

6. Conclusiones

6.1 Conclusiones

En la última década, el Deep Learning ha mostrado un gran potencial en numerosas aplicaciones, obteniendo resultados de mayor rendimiento que los aportados hasta ahora con técnicas más tradicionales para aplicaciones similares. Esta disciplina goza de ser una de las ramas con más potencial de la Inteligencia Artificial. Sin embargo, estás técnicas no han llegado a utilizarse aún en la gestión de bandas de frecuencia para el entorno de redes celulares.

Ante esta panorámica y en el marco del presente trabajo, se ha estudiado una posible aplicación de predicción mediante técnicas de Deep Learning para la mejora de la gestión de bandas de frecuencias para repetidores de salto de frecuencia de telefonía móvil.

Para ello, se ha hecho una introducción con los tipos de aprendizaje y técnicas de Machine Learning, incluyendo varios tipos de problemas y algoritmos para su resolución. Donde se señala que actualmente hay técnicas dentro del Machine Learning que aportan un gran potencial en la resolución de problemas de predicción. Se trata de técnicas de Deep Learning, y dentro de estás, las redes recurrentes son una buena opción para problemas de predicción.

De cara a resolver el problema de gestión de bandas de frecuencias de repetidores de salto de frecuencia, se necesita un sistema de predicción con un horizonte de predicción suficiente para que no se descarten llamadas. Un problema de las redes recurrentes es el conocido como desvanecimiento del gradiente, este provoca que estas no sean capaces de manejar ocurrencias a largo plazo, por lo que este tipo de redes no serán adecuadas para acometer el problema planteado.

Las redes LSTM son un tipo de red recurrente que solventa el problema de desvanecimiento del gradiente, por este motivo, ha sido el tipo de red elegida para resolver el problema planteado.

Posteriormente, se han presentado las diferentes herramientas para llevar a cabo la implementación. Entre estas, destacar el framework TensorFlow. Además de su flexibilidad y dinamismo, con TensorFlow se tiene acceso a una extensa documentación y gran número de tutoriales que facilitan el desarrollo del usuario principiante en el aprendizaje automático. Pero la principal característica por la que se ha elegido este framework es porque se trata de un framework de bajo nivel, lo cual permite ver con transparencia las operaciones realizadas.

6.1.1 Discusión sobre el trabajo

Los datos dispuestos de la ciudad de Milán se encuentran divididos en 1000 celdas con un área de 235 m^2 cada una. Cada intervalo temporal contiene datos de varias celdas, por lo que si quiere gestionar recursos de una sola BTS hay que filtrar las mismas. Para tener un área que sea más acorde con la región de cobertura que cubre una BTS se hace una agrupación de 4 celdas contiguas. Esta discriminación de celdas provoca que los datos de una sola celda sean más variables, generando unas métricas más elevadas, aunque en menor medida que si se filtra por una sola celda. Prueba de ello, son las pruebas realizadas sin filtrar las celdas, obteniéndose métricas superiores a las mostradas en los resultados de las redes neuronales analizadas en el estado del arte.

Por otra parte, señalar que **todos los objetivos marcados para este trabajo se han conseguido**. Para ello, se ha seguido la planificación propuesta al inicio sin desvíos, contemplando con rigor los tiempos marcados para cada tarea e hito. Sin embargo, no se lograban los resultados esperados en la implementación de las métricas, lo que llevó a confeccionar un nuevo dataset con mayor número de días una vez comenzada y avanzada la fase de pruebas. No obstante, este problema no ha provocado ningún desvío en la planificación propuesta inicialmente.

En este trabajo se han conseguido tres contribuciones relevantes relacionadas con los objetivos propuestos al inicio del mismo:

1. **Resolver el problema de gestión de canales de frecuencias para repetidores de salto de frecuencia implementados en zonas rurales.**

2. **Mejorar las métricas obtenidas respecto a las redes presentadas en el estado del arte.**

3. **Implementar un sistema de predicción con un horizonte de predicción de 3 horas.**

6.1.2 Lecciones aprendidas

Con motivo de no repetir posibles errores en un nuevo proyecto, una vez finalizado el actual es una buena práctica hacer una reflexión sobre los errores cometidos o factores que podrían haber ocasionado un error o desvío.

En este trabajo, no se han producido errores que hayan ocasionado algún desvío, pero si se han apreciado algunos factores o situaciones que podrían haber ocasionado una pérdida de tiempo importarte. Las lecciones aprendidas en este trabajo se enumeran a continuación:

1. Verificar que el dataset dispuesto es correcto.

2. Es preferible documentarse correctamente para realizar una tarea en lugar gastar tiempo investigando durante su ejecución.

3. Asignar tiempos reales.

4. Evitar una ejecución sin una buena planificación.

5. Es recomendable comenzar la redacción de la memoria al comienzo y en paralelo de cada tarea.

6.1.3 Trabajo futuro

Este trabajo puede ser el enlace para un futuro trabajo de investigación que vaya más allá del aprendizaje supervisado.

En este sentido, comienza a ganar popularidad el aprendizaje por refuerzo. Este tipo de aprendizaje pretende entrenar a las redes neurales a través de respuestas sobre sus propias acciones. Así la red puede medir el éxito de las acciones basándose en diferentes recompensas sobre las mismas.

En el marco de la redes de telecomunicaciones, este modelo de aprendizaje podría mejorar el comportamiento de Redes Definidas por Software (en inglés Software Defined Networking, SDN).

Bibliografía

[1] Vinayakumar R, Soman KP and Prabaharan Poornachandrany, “Applying Deep Learning Approaches for Network Traffic Prediction” 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 16-17 Sept. 2017, pp. 2353-2358.

[2] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg, “Fast accurate computation of large-scale ip traffic matrices from link loads,” in ACM SIGMETRICS Performance Evaluation Review, vol. 31, no. 1. ACM, 2003, pp. 206–217.

[3] M. Barabas, G. Boanea, A. B. Rus, V. Dobrota, and J. Domingo-Pascual, “Evaluation of network traffic prediction based on neural networks with multi-task learning and multiresolution decomposition”, in Intelligent Computer Communication and Processing (ICCP), 2011 IEEE International Conference on. IEEE, 2011, pp. 95–102.

[4]https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/EGZHFV [Consulta: 04/10/2018]

[5] Pedro Isasi Viñuela, Inés M. Galván León, “Redes de Neuronales Artificiales: Un enfoque práctico”, PEARSON EDUCACIÓN S.A. MADRID, 2003

[6] https://www.jetbrains.com/pycharm/download/#section=windows [Consulta: 19/09/2018]

[7] https://www.tensorflow.org/ [Consulta: 19/09/2018]

[8] https://pandas.pydata.org/ [Consulta: 20/09/2018]

[9] https://scikit-learn.org/stable/ [Consulta: 20/09/2018]

[10] https://matplotlib.org/ [Consulta: 28/10/2018]

[11] https://www.ibm.com/developerworks/ssa/library/cc-machine-learning-deep-learning-architectures/index.html [Consulta: 30/10/2018]

[12] http://colah.github.io/posts/2015-08-Understanding-LSTMs/ [Consulta: 7/10/2018]

[13] https://eu.udacity.com/course/deep-learning--ud730 [Consulta: 04/10/2018]

[14] https://scikit-learn.org/stable/modules/preprocessing.html#-preprocessing [Consulta: 08/10/2018]

[15] https://www.nature.com/articles/sdata201555 [Consulta: 30/10/2018]

[16] https://torres.ai/research-teaching/tensorflow/libro-hello-world-en-tensorflow/ [Consulta: 25/09/2018]

[17] https://iaarbook.github.io/ML/#tipos-de-machine-learning [Consulta: 25/09/2018]

[18] http://www.westworld.be/wp-content/uploads/2012/01/kernel_-convolution.jpg [17/11/2018]

[19] https://relopezbriega.github.io/blog/2015/10/10/machine-learning-con-python/ [consulta: 09/12/2018]

Anexos

Como anexos se van a presentar los códigos implementados para las dos redes neuronales comparadas.

Anexo 1. Código implementado para la red LSTM

```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import os
from six.moves import cPickle as pickle
from sklearn.preprocessing import StandardScaler
import datetime

if False or not os.path.exists('scaled_data.pickle'):
    parse = lambda x: datetime.datetime.fromtimestamp(float(x) / 1000)
    #
https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/D
VN/UTLAHU
    sliceSum = pd.DataFrame({})
    for index in range(1,42):
        sliceSum2 = (pd.read_csv('Datos/sms-call-internet-mi-2013-11-
'+str(index).zfill(2) +'.txt', sep='\t',
                                 encoding="utf-8-sig",
names=['CellID', 'datetime', 'countrycode', 'smsin', 'smsout',

'callin', 'callout', 'internet'], parse_dates=['datetime']
                                 , date_parser=parse))
        sliceSum2 = sliceSum2.set_index('datetime')
        sliceSum2 = (sliceSum2[(sliceSum2['CellID'] == 1) |
(sliceSum2['CellID'] == 2) | (sliceSum2['CellID'] == 101)
                               | (sliceSum2['CellID'] == 102)])
        sliceSum2 = sliceSum2.groupby(['datetime'],
as_index=True).sum()
        sliceSum2['hour'] = sliceSum2.index.hour
        sliceSum2['weekday'] = sliceSum2.index.weekday
        sliceSum = sliceSum.append(sliceSum2)

    data =sliceSum['callout'].values
    # Escalado de datos
    scaled_data = StandardScaler().fit_transform(data.reshape(-1, 1))
    pickle_file = open('scaled_data.pickle', 'wb')
    pickle.dump(scaled_data, pickle_file)
    pickle_file.close()

extract_data = open('scaled_data.pickle', 'rb')
scaled_data = pickle.load(extract_data)

# Hyperparameters
slide_window = 84 #84
batch_size = slide_window*2
nodes = 32
learning_rate = 0.001
sample_pred = 18 # número de muestras vista a predecir
epochs = 70
```

```
Layers = 2

# Los datos de training son divisibles por la ventana
Len_train = 0.6*len(scaled_data)
Sub = Len_train % slide_window
if Sub != 0:
    Len_train = Len_train - Sub

Train_data = scaled_data[:math.ceil(Len_train)] #60% de los datos
Rest_data = scaled_data[math.ceil(Len_train):] # 40% de los datos
Validation_data = Rest_data[:math.ceil(len(Rest_data)/2)]# 20% de los
datos
Test_data = Rest_data[math.ceil(len(Rest_data)/2):]# 20% de los datos

# Ventana deslizante
def reformat(data, slide_window):
    X = []
    Y = []

    i = 0
    while (i + slide_window) <= len(data) - 1 - sample_pred:
        X.append(data[i:i + slide_window])
        Y.append(data[i + sample pred + slide window - 1])
        i += 1
    return X, Y

# Dataset creado
X_train, y_train = reformat(Train_data, slide_window)
X_test, y_test = reformat(Test_data, slide_window)
X_validation, Y_validation = reformat(Validation_data, slide_window)

X_train = np.array(X_train)
Y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)
X_validation = np.array(X_validation)
Y_validation = np.array(Y_validation)

# definicion de la red
graph = tf.Graph()
with graph.as_default():

    #placeholders
    inputs = tf.placeholder(tf.float32, [batch_size, slide_window, 1])
    targets = tf.placeholder(tf.float32, [batch_size, 1])
    #pesos de salida y bias
    w_o = tf.Variable(tf.truncated_normal([nodes, 1], stddev=0.05))
    b_ol = tf.Variable(tf.zeros([1]))

    # Modelo LSTM
    #https://www.tensorflow.org/tutorials/sequences/recurrent #Apache
License 2.0
    def lstm_cell():
#
        return tf.contrib.rnn.BasicLSTMCell(nodes)

    stacked_lstm = tf.contrib.rnn.MultiRNNCell(
        [lstm_cell() for _ in range(Layers)])

    initial_state = state = stacked_lstm.zero_state(batch_size,
tf.float32)     #
```

```
    for i in range(slide_window):
#
        output, state = stacked_lstm(inputs[:, i], state)
#

#########################################################################
    outputs = []
    i = 0
    for i in range(batch_size ):
        outputs.append(tf.nn.xw_plus_b(tf.reshape(output[i], (1,
nodes)), w_o, b_ol))

    # MSE
    loss = tf.reduce_mean(
        tf.losses.mean_squared_error(
            targets, tf.concat(outputs, 0)))

    # MSE relativo
    error = tf.losses.mean_squared_error(targets, tf.concat(outputs,
0)) / targets
    mse_r = 100*tf.reduce_mean(error)

    # Optimizado
    gradients = tf.gradients(loss, tf.trainable_variables())
    optimizer = tf.train.AdamOptimizer(learning_rate)
    trained_optimizer = optimizer.apply_gradients(zip(gradients,
tf.trainable_variables()))

    init_g = tf.global_variables_initializer()
    init_l = tf.local_variables_initializer()
with tf.Session(graph=graph) as session:
    session.run(init_g)
    session.run(init_l)

    Lossplot = []
    Lossplotvalid = []
    for j in range(epochs):
        temp = []
        scores = []
        epoch_loss = []
        i = 0
        # Entrenamiento
        while (i + batch_size) <= len(X_train):
            X_batch = X_train[i:i + batch_size]
            y_batch = y_train[i:i + batch_size]
            feed_dict = {inputs: X_batch, targets: y_batch}
            out, perd, _ = session.run([outputs, loss,
trained_optimizer], feed_dict=feed_dict)
            temp.extend(out)
            epoch_loss.append(perd)
            i += batch_size
        epoch_loss = np.mean(epoch_loss)
        Lossplot.append(epoch_loss)

        validation = []
        valid_loss = []
        MRE = []
        i = 0
        # Validación
        while i + batch_size <= len(X_validation):
            feed_dict = {inputs: X_validation[i:i + batch_size],
```

```
targets: Y_validation[i:i + batch_size]}
            out, l, mre = session.run([outputs, loss, mse_r],
feed_dict=feed_dict)
            MRE.append(mre)
            valid_loss.append(l)
            validation.extend(out)
            i += batch_size
        valid_loss = np.mean(valid_loss)
        MRE = np.mean(MRE)
        Lossplotvalid.append(valid_loss)
        print('Epoch {}/{}'.format(j, epochs), ' Training loss:
{}'.format(np.mean(epoch_loss)),
              " Validation MSE: %.2f" % valid_loss, " Validation MSE
Relative: %.2f%%" % abs(MRE))

    MSEr_test = []
    test_loss = []
    test = []
    i = 0
    # Test
    while i + batch_size <= len(X_test):
        feed_dict = {inputs: X_test[i:i + batch_size], targets:
y test[i:i + batch size]}
        out, MSE, MSEr = session.run([outputs, loss, mse_r],
feed_dict=feed_dict)
        MSEr_test.append(MSEr)
        test_loss.append(MSE)
        test.extend(out)
        i += batch_size
    test_loss = np.mean(test_loss)
    MSErtest = np.mean(MSEr_test)
    print('Epoch {}/{}'.format(j, epochs), " Test MSE: %.2f" %
test_loss, " Test MSE Relative: %.2f%%" % abs(MSErtest))

    tests = []
    i = 0
    while i <= len(test) - 1:
        tests.extend(np.reshape(test[i], (len(test[i]))))
        i += 1

    plt.figure(num='Prediction', figsize=(16, 9))
    plt.title('Llamadas realizadas')
    plt.xlabel('x10 Minutes')
    plt.ylabel('N° de llamadas escaladas')

    # Ploteado de datos Originales y datos de test
    plt.plot(y_test[:len(tests)], label='Original data')
    plt.plot(tests, label='Testing data')
    plt.legend()
    plt.show()

    # Ploteado MSE
    plt.figure(num='MSE', figsize=(16, 9))
    plt.title('Training data vs Validation data MSE')
    plt.xlabel('Epoch')
    plt.ylabel('MSE')
    plt.plot(Lossplot, label='Traning data')
    plt.plot(Lossplotvalid, label='Validation data')
    plt.legend()
    plt.show()
```

Anexo 2. Código implementado para el Perceptrón multicapa

```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import os
from six.moves import cPickle as pickle
from sklearn.preprocessing import StandardScaler
import datetime

if False or not os.path.exists('scaled_data.pickle'):
    parse = lambda x: datetime.datetime.fromtimestamp(float(x) / 1000)
    #
https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/D
VN/UTLAHU
    sliceSum = pd.DataFrame({})
    for index in range(1,42):
        sliceSum2 = (pd.read_csv('Datos/sms-call-internet-mi-2013-11-
'+str(index).zfill(2) +'.txt', sep='\t',
                                 encoding="utf-8-sig",
names=['CellID', 'datetime', 'countrycode', 'smsin', 'smsout',

'callin', 'callout', 'internet'], parse_dates=['datetime']
                                 , date_parser=parse))
        sliceSum2 = sliceSum2.set_index('datetime')
        sliceSum2 = (sliceSum2[(sliceSum2['CellID'] == 1) |
(sliceSum2['CellID'] == 2) | (sliceSum2['CellID'] == 101)
                               | (sliceSum2['CellID'] == 102)])
        sliceSum2 = sliceSum2.groupby(['datetime'],
as_index=True).sum()
        sliceSum2['hour'] = sliceSum2.index.hour
        sliceSum2['weekday'] = sliceSum2.index.weekday
        sliceSum = sliceSum.append(sliceSum2)

    data =sliceSum['callout'].values
    # Escalado de datos
    scaled_data = StandardScaler().fit_transform(data.reshape(-1, 1))
    pickle_file = open('scaled_data.pickle', 'wb')
    pickle.dump(scaled_data, pickle_file)
    pickle_file.close()

extract_data = open('scaled_data.pickle', 'rb')
scaled_data = pickle.load(extract_data)

# Hyperparameters
slide_window = 20
batch_size = slide_window*2
nodos = 300
learning_rate = 0.0005
sample_pred = 18 # número de muestras vista a predecir
epochs = 50

# Los datos de training son divisibles por la ventana
Len_train = 0.6*len(scaled_data)
Sub = Len_train % slide_window
if Sub != 0:
```

```
    Len_train = Len_train - Sub

Train_data = scaled_data[:math.ceil(Len_train)] #60% de los datos
Rest_data = scaled_data[math.ceil(Len_train):] # 40% de los datos
Validation_data = Rest_data[:math.ceil(len(Rest_data)/2)]# 20% de los
datos
Test_data = Rest_data[math.ceil(len(Rest_data)/2):]# 20% de los datos

# Ventana deslizante
def reformat(data, slide_window):
    X = []
    Y = []

    i = 0
    while (i + slide_window) <= len(data) - 1 - sample_pred:
        X.append(data[i:i + slide_window])
        Y.append(data[i + sample_pred + slide_window - 1])
        i += 1
    return X, Y

# Dataset creado
X_train, y_train = reformat(Train_data, slide_window)
X test, y test = reformat(Test data, slide window)
X_validation, Y_validation = reformat(Validation_data, slide_window)

X_train = np.array(X_train)
Y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)
X_validation = np.array(X_validation)
Y_validation = np.array(Y_validation)

# define the network
graph = tf.Graph()
with graph.as_default():

    #placeholders
    inputs = tf.placeholder(tf.float32, [batch_size, slide_window, 1])
    targets = tf.placeholder(tf.float32, [batch_size, 1])

    # Pesos
    weights_1 = tf.Variable(tf.random_normal([slide_window, nodos]))
    weights_2 = tf.Variable(tf.random_normal([nodos, nodos]))
    weights_3 = tf.Variable(tf.random_normal([nodos, nodos]))
    weights_out = tf.Variable(tf.random_normal([nodos, 1]))

    # Bias
    bias_1 = tf.Variable(tf.random_normal([nodos]))
    bias_2 = tf.Variable(tf.random_normal([nodos]))
    bias_3 = tf.Variable(tf.random_normal([nodos]))
    bias_out = tf.Variable(tf.random_normal([1]))

    # Perceptrón multicapa
    layer_1 = tf.add(tf.matmul(tf.reshape(inputs, (batch_size,
slide_window)), weights_1), bias_2)
    layer_2 = tf.add(tf.matmul(layer_1, weights_2), bias_2)
    layer_3 = tf.add(tf.matmul(layer_2, weights_3), bias_3)
    outputs = tf.matmul(layer_2, weights_out) + bias_out

    # Define MSE
    loss = tf.reduce_mean(
```

```
        tf.losses.mean_squared_error(
            targets, outputs))
    # MSE Relativo
    error = tf.losses.mean_squared_error(targets, tf.concat(outputs,
0)) / targets
    mse_r = 100 * tf.reduce_mean(error)

    # Optimizado
    gradients = tf.gradients(loss, tf.trainable_variables())
    optimizer = tf.train.AdamOptimizer(learning_rate)
    trained_optimizer = optimizer.apply_gradients(zip(gradients,
tf.trainable_variables()))

    init_g = tf.global_variables_initializer()
    init_l = tf.local_variables_initializer()
with tf.Session(graph=graph) as session:
    session.run(init_g)
    session.run(init_l)

    Lossplot = []
    Lossplotvalid = []
    for j in range(epochs):
        temp = []
        scores = []
        epoch_loss = []
        i = 0
        # Entrenamiento
        while (i + batch_size) <= len(X_train):
            X_batch = X_train[i:i + batch_size]
            y_batch = y_train[i:i + batch_size]
            feed_dict = {inputs: X_batch, targets: y_batch}
            out, perd, _ = session.run([outputs, loss,
trained_optimizer], feed_dict=feed_dict)
            temp.extend(out)
            epoch_loss.append(perd)
            i += batch_size
        epoch_loss = np.mean(epoch_loss)
        Lossplot.append(epoch_loss)

        validation = []
        valid_loss = []
        MRE = []
        i = 0
        # Validación
        while i + batch_size <= len(X_validation):
            feed_dict = {inputs: X_validation[i:i + batch_size],
targets: Y_validation[i:i + batch_size]}
            out, l, mre = session.run([outputs, loss, mse_r],
feed_dict=feed_dict)
            MRE.append(mre)
            valid_loss.append(l)
            validation.extend(out)
            i += batch_size
        valid_loss = np.mean(valid_loss)
        MRE = np.mean(MRE)
        Lossplotvalid.append(valid_loss)
        print('Epoch {}/{}'.format(j, epochs), ' Training loss:
{}'.format(np.mean(epoch_loss)),
              " Validation MSE: %.2f" % valid_loss, " Validation MSE
Relative: %.2f%%" % abs(MRE))
```

```
    MSEr_test = []
    test_loss = []
    test = []
    i = 0
    # Test
    while i + batch_size <= len(X_test):
        feed_dict = {inputs: X_test[i:i + batch_size], targets:
y_test[i:i + batch_size]}
        out, MSE, MSEr = session.run([outputs, loss, mse_r],
feed_dict=feed_dict)
        MSEr_test.append(MSEr)
        test_loss.append(MSE)
        test.extend(out)
        i += batch_size
    test_loss = np.mean(test_loss)
    MSErtest = np.mean(MSEr_test)
    print('Epoch {}/{}'.format(j, epochs), " Test MSE: %.2f" %
test_loss, " Test MSE Relative: %.2f%%" % abs(MSErtest))

    tests = []
    i = 0
    while i <= len(test) - 1:
        tests.extend(np.reshape(test[i], (len(test[i]))))
        i += 1

    plt.figure(num='Prediction', figsize=(16, 9))
    plt.title('Llamadas realizadas')
    plt.xlabel('x10 Minutes')
    plt.ylabel('N° de llamadas escaladas')

    # Ploteado de datos Originales y datos de test
    plt.plot(y_test[:len(tests)], label='Original data')
    plt.plot(tests, label='Testing data')
    plt.legend()
    plt.show()

    # Ploteado MSE
    plt.figure(num='MSE', figsize=(16, 9))
    plt.title('Training data vs Validation data MSE')
    plt.xlabel('Epoch')
    plt.ylabel('MSE')
    plt.plot(Lossplot, label='Traning data')
    plt.plot(Lossplotvalid, label='Validation data')
    plt.legend()
    plt.show()
```