CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS
AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

UNIDAD ZACATENCO

DEPARTAMENTO DE CONTROL AUTOMÁTICO

Modelado de sistemas dinámicos usando redes neuronales recurrentes profundas

T E S I S

Que presenta

JESÚS GONZÁLEZ GODOY

Para obtener el grado de

DOCTOR EN CIENCIAS

En la especialidad de

CONTROL AUTOMÁTICO

Director de tesis: **Dr. Wen Yu Liu**

Ciudad de México, Diciembre, 2019

Resumen

Modelado de sistemas dinámicos usando redes neuronales recurrentes profundas

Jesús González Godoy

2019

Un sistema dinámico, se entiende como un conjunto de objetos relacionados gobernados por reglas o normas que regulan un comportamiento característico de esa entidad, el cual evoluciona con el tiempo. Para el análisis de un sistema se requiere un modelo, que es una abstracción del mismo. La forma más precisa de describir un sistema es con un modelo matemático, sin embargo, en muchas ocasiones es en extremo complicado obtener las ecuaciones a partir de los elementos del sistema o simplemente se desconocen. Para esto se han desarrollado múltiples herramientas denominadas herramientas de caja negra, las cuales intentan generar el modelo a partir únicamente de observaciones del comportamiento del sistema y los elementos relacionados con él, es decir, sus entradas y salidas. Una de estas herramientas de caja negra que resultan muy eficientes son las redes neuronales ya que tienen la capacidad de aprender a partir de las observaciones el comportamiento del sistema. Si bien existen muchos avances en cuanto al modelado de sistemas, existen aún grandes retos que afrontar en este rubro. Uno de ellos es el uso de modelos en entornos de simulación, donde se debe predecir la salida del modelo sin una retroalimentación real. Cuando existe una retroalimentación de la salida, realizar la predicción del modelo resulta más sencillo, sin embargo, esto se complica cuando se trata de predecir más allá de un paso en el tiempo con la información ya conocida, o si se tratan de predecir múltiples pasos adelante ya que sería como predecir con las predicciones. Un caso excepcionalmente complicado es el modelado de sistemas que parecen tener un comportamiento aleatorio y solo pueden exhibir un patrón tras un periodo largo de observación. En los últimos años la teoría de las redes neuronales dio un gran salto hacia adelante con la aparición del aprendizaje profundo con el cual se logran resolver tareas antes impensables, como es el etiquetado de imágenes, la descripción de imágenes, traducción, etc. En el presente trabajo se proponen arquitecturas neuronales profundas para el modelado de sistemas dinámicos usando redes neuronales recurrentes, en particular las celdas LSTM y GRU que son especialmente útiles cuando se tratan secuencias largas de información. Los resultados obtenidos con estas arquitecturas son muy favorables y abordan los problemas actuales antes descritos del modelado de sistemas con gran eficiencia, abriendo las puertas a la solución de problemas antes impensables.

Abstract

Dynamic systems modeling using deep recurrent neural networks

Jesús González Godoy

2019

A dynamic system can be understood as a set of related objects governed by rules or norms that regulate its characteristic behavior which evolves with the time. This applies not only to physical objects, a system can exist in economics, social sciences, etc., In general any abstraction whose behavior is dictated by norms or rules is a system. For the analysis of a system, it is necessary a model that is an abstraction of the system; The most accurate way to describe a system is with a mathematical model, however, in many cases it is extremely complicated to obtain the equations from the elements of the system or they are simply unknown. For this, multiple tools called black box tools have been developed, they try to generate the model based only on observations of the system behavior and the related elements, they are the inputs and outputs. One of these black box tools that are very efficient are the neural networks since they can learn from observations the system behavior and thus be able to try to predict its output from a given input with some margin of error. While there are many advances in system modeling, there are also great challenges to face in this area and where even the traditional neural networks show their limits. One of them is the use of models in simulation environments, where the output of the model must be predicted without a real feedback. When there is a feedback of the output, making the prediction of the model is easier, however, this is complicated when it is needed to predict beyond a step in the time with the information already known, or if they try to predict multiple steps forward since it would be like predicting with the predictions. An exceptionally complicated case is the modeling of systems that seems to have a random behavior and can only display a pattern after a long period of observation. In the last years the theory of neural networks gave a great leap forward with the appearance of deep learning which claims to solve tasks previously unthinkable and with a high degree of complexity, such as image tagging, image description, translation, etc. In the present work deep neural architectures for modeling dynamic systems using recurrent neural networks are proposed, particularly LSTM cells and GRUs that are especially useful when dealing with long sequences of information. The results obtained with these architectures are very favorable and address the current problems described above of system modeling with great efficiency, opening the doors to solving problems before unthinkable.

Agradecimientos

Modelado de sistemas dinámicos usando redes neuronales recurrentes profundas

Jesús González Godoy

2019

Estoy sumamente orgulloso de haber cursado mis estudios de doctorado en el CINVESTAV y especialmente en el departamento de Control Automático, un lugar asombroso y hogar de gente brillante. Si bien este fue un duro viaje, tuve la gran fortuna de contar en mi vida con quienes fueron fundamentales para concluir con éxito mi meta y con quien estoy profundamente agradecido, agradezco:

Primeramente, al Dr. Wen Yu, por confiar en mí y aceptarme como su alumno, agradezco su paciencia, su guía y ayuda continua a lo largo de mi trabajo la cual ha sido invaluable, sin duda un gran ejemplo a seguir, alguien a quien admiro, estimo y respeto muchísimo, él es para mí el mejor asesor que pude haber tenido.

A mis amigos, en especial a Paco, con su gran visión y talento, una increíble persona y ser humano que me ofreció su guía y apoyo durante toda esta travesía.

Por su puesto agradezco especialmente a mi compañera de vida, mi hermosa e inteligente esposa quien además de ofrecerme incondicionalmente su guía y apoyo, recorrió a mi lado este duro camino, siempre con una sonrisa, siempre conmigo, sin duda alguna este logro nos pertenece a los dos.

A mis maravillosos padres, para quienes me faltan palabras con que expresar mi inmensa gratitud, ellos son mis pilares de fuerza, mi más grande motivación, mi impulso, mi motor, mi ejemplo, mi guía. Gracias.

Finalmente agradezco al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo brindado durante mi estancia en el programa de Doctorado.

Índice general

Índice de figuras

Capítulo 1

Introducción

Para el modelado de sistemas existe todo un mundo complejo bien desarrollado y fundamentado que soporta esta disciplina, que incluye distintos lenguajes, algoritmos, metodologías elementos matemáticos y demás formalismos [1], de manera general, cualquier abstracción de la vida real puede ser vista como un sistema dinámico, siempre y cuando pueda ser descrita en términos del cambio de su estado respecto a una o más variables de su entorno y este puede ser discreto o continuo. A lo largo del presente trabajo se analizará el caso discreto en el que el tiempo se mide en periodos o lapsos que denominaremos pasos, en particular el caso en el que la estructura del sistema es desconocida a lo que se le llama modelo de caja negra, y se aborda este problema con una representación NARMAX[2], el modelo ideal para abordarlo con redes neuronales.

Dado que este es un tema tan importante existen cantidades de métodos que resuelven el problema con distintos enfoques y resultados como muestran J Sjöberg, Q Zhang, et al[3]. Destacan por su relación con este trabajo los enfoques que hacen uso de redes neuronales [4][5][6][7], en donde existen grandes avances con muy buenos resultados, sin embargo, la mayoría de las veces se realizan pruebas con datos generados artificialmente o con datos obtenidos en el laboratorio, además siguen abiertos y en constante investigación múltiples problemas complejos que son los que se abordan en el presente trabajo, ofreciendo una solución mediante diferentes arquitecturas de redes neuronales recurrentes profundas haciendo uso de celdas LSTM y GRU que en general en este caso ofrecen resultados similares y se pueden usar indistintamente en cualquiera de las arquitecturas propuestas.

También se propone un entrenamiento estable para este tipo de redes, es decir, que siempre se va a obtener un buen resultado, lo que es muy importante en entornos y aplicaciones reales donde se necesita tener completa certeza de como va a reaccionar

el modelo ante situaciones no previstas, ya que de lo contrario puede ser motivo de una complicación, además de tener la ventaja de ser más rápido que la tradicional retropropagación hacia atrás que dada la complejidad de las celdas LSTM y su cantidad de parámetros además de la recursividad, es mucho mas tardado que los tradicionales perceptrones multicapa.

Finalmente, se aplican los modelos propuestos a un tema sumamente complejo debido a la naturaleza aparentemente aleatoria de su comportamiento y de gran interés en la comunidad científica que es, la predicción de temblores, donde además de usar los modelos propuestos en los siguientes capítulos, se lograron buenos resultados que no se habían podido lograr con otros modelos de redes neuronales y que además ponen las bases para futuras investigaciones.

1.1. Motivación

El modelado de sistemas dinámicos es crucial para el entendimiento de los fenómenos naturales, físicos y sociales a los que se enfrenta la humanidad, ya que con su entendimiento que en sí es un gran avance, se puede pensar incluso en controlarlos o manipularlos, esto por supuesto contribuye notablemente al progreso científico y tecnológico, pero también ayuda a dar soluciones a problemas concretos y tan variados como los mismos sistemas. Por lo tanto, el modelado de sistemas dinámicos es entonces un campo de estudio fundamental y de gran importancia que compete no solo al área de matemáticas sino a todas las áreas de la ciencia.

En general los sistemas dinámicos que dan respuesta a problemas reales difícilmente pueden ser representados con un modelo simple incluso si su abstraccion es senclla, haciendo necesaria la existencia de herramientas matemáticas sofisticadas que se van mejorando con el paso del tiempo y con el surgimiento de nuevas tecnologías y métodos, y es precisamente aquí donde el presente trabajo toma lugar.

La complejidad de los sistemas va más allá de su naturaleza intrínseca ya que en muchas ocasiones o bien, no se conocen completamente los factores que se ven implicados en el comportamiento del sistema y por lo tanto las leyes que los gobiernan, o simplemente estos factores no se corresponden con un elemento físico o con una representación conocida, incluso muchas veces las variables involucradas en el sistema son totalmente desconocidas, se conoce únicamente la respuesta del sistema bajo ciertos estímulos y condiciones del entorno sin saber con certeza cuáles de los objetos presentes en ese panorama influye o no directamente en la salida del sistema. Además de otros muchos retos vigentes que se

presentan al momento del modelado.

A pesar de que se ha hecho uso extenso de las redes neuronales para este propósito desde hace ya más de una década estas arquitecturas tienen sus limitantes bien definidas. Por suerte en los últimos años las redes neuronales han evolucionado, en parte gracias al avance de las herramientas computacionales, y se han desarrollado nuevas técnicas y metodologías, entre ellas el aprendizaje profundo y las redes neuronales recurrentes las cuales marcan un hito en la resolución de problemas imponiendo un nuevo paradigma. Las redes neuronales recurrentes surgen de la necesidad de manejar secuencias de datos de manera eficiente y están diseñadas especialmente para eso, en particular las basadas en celdas LSTM y GRU y han demostrado grandes capacidades para la resolución de problemas que de otro modo serian irresolubles.

Es ahora necesario actualizar las herramientas de modelado con estas técnicas de recurrencia, aprendizaje profundo y manejo de secuencias de datos que prometen grandes resultados y así generar una contribución significativa que pueda ser usada, evolucionada y perfeccionada, capaz de resolver problemas actuales en el modelado de sistemas.

1.2. Contribuciones

Se busca abordar con éxito problemas actuales del modelado de sistemas dinámicos con el uso de herramientas propias del aprendizaje automático dentro del marco del control automático, en particular se usaron redes neurales recurrentes profundas con celdas LSTM y GRU que se especializan en información secuencial y resuelven algunos problemas intrínsecos de las redes neuronales tradicionales ademas de tener un enfoque totalmente nuevo y refrescante que obliga a un cambo de paradigma para su entendimiento y aplicación.

Durante todo el trabajo se usó una representación NARMAX para el modelado de los sistemas. Las contribuciones obtenidas se detallan en los capítulos 3, 4 y 5, sin embarg,o estas se resumen puntualmente a continuación.

1. Se propone una nueva arquitectura neuronal (LSTM-NN) que usa tanto celdas LSTM y GRU como el perceptrón tradicional, capaz de resolver el problema de modelado en entornos de simulación con mejores resultados que las redes neuronales tradicionales. En resumen, se resuelve el siguiente problema

$$\hat{y}(k) = f(u(k), \ldots, u(k-n))$$

2. Se propone un entrenamiento estable para la arquitectura LSTM-NN que asegura la obtención de un buen resultado además de acelerar el entrenamiento.

3. Se proponen arquitecturas o modelos recurrentes profundos usando celdas LSTM o GRU que hacen posible la resolución de problemas complejos en el modelado de sistemas dinámicos, en particular el hecho de predecir más de un paso adelante y múltiples pasos adelante. En resumen, se resuelven los siguientes problemas

$$\begin{aligned} \hat{y}(k+m) &= f(y(k-1),\ldots,y(k-n)) \\ [\hat{y}(k+m),\ldots,\hat{y}(k+l)] &= f(y(k-1),\ldots,y(k-n)) \end{aligned}$$

4. Se usaron las estructuras neuronales recurrentes profundas para la prediccion de sismos obteniendo resultados favorables.

1.3. Estructura de la tesis

En el Capítulo 2 se muestran los aspectos teóricos relacionados con la identificación de sistemas, el gradiente descendiente que juega un papel central en el entrenamiento de las redes neuronales, así como algunos métodos de optimización, se presentan también las redes neuronales recurrentes con énfasis en las celdas LSTM y GRU.

En el Capítulo 3 se plantea el problema de modelado tanto de predicción como de simulación y se proponen las celdas LSTM y GRU desde el punto de vista del control automático, después usando estas celdas se crea una arquitectura neuronal que usa tanto celdas LSTM como perceptrones tradicionales (LSTM-NN), se explica su estructura y entrenamiento, además se pone a prueba frente a una MLP tradicional con varios ejemplos y con especial interés en el modelo de simulación que es uno de los problemas a resolver.

En el Capítulo 4 se ofrece una alternativa al entrenamiento de las redes neuronales recurrentes LSTM que a diferencia del tradicional con retropropagación es más rápido y se asegura la estabilidad del método, es decir, que el error siempre converge, además puede ser usado en problemas en línea donde hasta ahora este tipo de redes han mostrado menor eficiencia debido al tiempo que requiere su entrenamiento. En particular se desarrolla el entrenamiento para la arquitectura propuesta en el capítulo 3 y en la parte de simulación y resultados se incursiona en la predicción de temblores.

En el Capítulo 5 se proponen tres arquitecturas más, y esta vez están pensadas en aprovechar al máximo la recurrencia, se explica su estructura y entrenamiento, además en la parte de simulación y resultados se demuestra su superioridad frente a las otras arquitecturas ya que se abordan con éxito problemas actuales del modelado donde las otras arquitecturas tienen sus claras limitantes. Finalmente en la parte de resultados se retoma la predicción de temblores desde un punto de vista más general y conveniente.

Finalmente en el Capítulo 6 se retoma la discusión y comparación de las arquitecturas propuestas además de sugerir caminos futuros y posibles rumbos que pueda tomar la investigación.

1.4. Publicaciones Obtenidas

1.4.1. Congresos internacionales

- **Nonlinear System Modeling Using LSTM Neural Networks**, Second IFAC Conference on Modelling, Identification and Control of Nonlinear Systems, June 20-22, 2018, Guadalajara, Mexico
- **Fast Training of Deep LSTM Networks**, 16th International Symposium on Neural Networks (ISNN 2019), Moscow, Russia, Springer LNCS 11554, 3-10, 2019
- **Earthquakes magnitude prediction using recurrent neural networks,** 2nd International Electronic Conference on Geosciences, June 08-15, 2019

1.4.2. Artículos en revistas

- **Long short-term memory based recurrent neural networks for forecasting the maximum daily earthquake magnitude,** Complexity, 17-10-2019 (En revisión)
- **Fast Training of Deep LSTM Networks with Guaranteed Stability for Nonlinear System Modeling, Neurocomputing**, (Aceptado)

Capítulo 2

Preliminares

2.1. Identificación de sistemas

Actualmente se tiene un amplio conocimiento del universo y las leyes que lo rigen gracias a un largo proceso de análisis y observaciones hechas a través de la historia de la humanidad. El desarrollo de las matemáticas y herramientas como el cálculo junto con el avance en la instrumentación han permitido la introducción de mejores y más precisos modelos de los sistemas dinámicos de la naturaleza como la transferencia de calor, el electromagnetismo, la gravedad etc., ya que ahora pueden ser descritos en términos de ecuaciones diferenciales.

Los modelos matemáticos son esenciales en todos los campos de la ciencia e ingeniería, ya que con ellos se tiene un mejor entendimiento del sistema con el cual se está trabajando y de esta forma poder realizar un seguimiento, aplicar un control o predecir su comportamiento. Realizar un modelo en ocasiones es bastante sencillo, pero otras muchas veces puede ser bastante complicado dependiendo del conocimiento previo que se tenga del problema, el cual comúnmente divide los modelos en tres categorías.

Modelos de caja blanca

En este caso el modelo es conocido a la perfección, es decir, que se puede reconstruir completamente de las condiciones físicas y el conocimiento previo.

Modelos de caja gris

En este caso se conocen algunas de las condiciones físicas, pero varios de los parámetros deben ser determinados por medio de la observación. Se puede hacer la distinción de dos subclases.

1. Modelado físico

   Se puede construir una estructura con base en leyes físicas la cual tiene cierto número de parámetros que deben ser estimados con los datos observados

2. Modelado semi físico

   Las leyes físicas y el conocimiento del problema sugieren cierta estructura para las observaciones realizadas cuyos parámetros deben ser estimados.

Modelos de caja negra

Aquí no se conocen las condiciones físicas ni se tiene ningún conocimiento previo, sin embargo, se puede seleccionar una estructura que se sabe es flexible y ha dado buenos resultados en el pasado.

2.1.1. Proceso de identificación

En identificación de sistemas a menudo es desconocida la estructura del proceso dinámico bajo estudio, así que lo más común es usar un modelo de caja negra. Encontrar un modelo matemático bajo estos supuestos involucra diseñar una secuencia de entrada o estímulo al sistema capaz de generar una respuesta o salida, que represente lo mejor posible su comportamiento y así al aplicar dicho estímulo experimentalmente se registren las respuestas obtenidas para conseguir el par entrada salida (X, Y) que representa el conocimiento previo del sistema como se muestra en la figura 2.1.

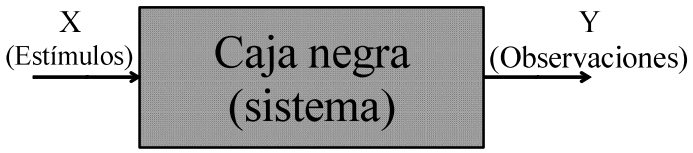


Figura 2.1: Modelo de caja negra

Donde $X = (x_0, x_i, \ldots, x_N)$, $Y = (y_0, y_i, \ldots, y_N)$ y N es el tamaño de la secuencia.

De esta forma el gran problema que debe resolver la identificación de sistemas es encontrar una estructura $f(x_i) = y_i$ (modelo y parámetros) adecuada para que el modelo propuesto del sistema se considere lo bastante bueno, es decir, que bajo una entrada x cualquiera el modelo resulte en $\hat{y} \sim y$ donde y es la respuesta del sistema aplicando x experimentalmente.

Dado que se realiza un muestreo de la señal de respuesta tras aplicar el estimulo apropiado a intervalos regulares, entonces la mayoría de los modelos usados en identificación de sistemas son en tiempo discreto.

Muchas veces es conveniente realizar una aproximación lineal del sistema y de esta forma simplificar no sólo el proceso de identificación, también el control y el entendimiento del mismo, sin embargo, además de ofrecer en muchos casos un comportamiento incompleto, existen condiciones como la histéresis, bifurcaciones o el caos que son imposibles de representar de manera lineal así que hacen inevitable el trabajo con modelos no lineales. Una poderosa herramienta es la representación NARMAX (*Nonlinear Autoregressive Moving Average with eXogenous inputs*), la cuál es una generalización de la representación para sistemas lineales ARMAX (*Autoregressive Moving Average with eXogenous inputs*) que es básicamente una ecuación diferencial que relaciona la salida del sistema en un instante dado con los valores de las entradas, salidas y el ruido en instantes previos. La representación NARMAX da solución al problema de encontrar la ecuación $f(x)$ para un modelo de caja negra.

2.1.2. Representación NARMAX

Un sistema dinámico es aquel que cambia su estado a través del tiempo y se caracteriza por un conjunto de variables cuyo valor determina el estado del sistema en el instante de tiempo t, denominadas variables de estado, además de un conjunto determinista de reglas de transición que dictan el estado siguiente del sistema a partir del estado actual o estados anteriores, comúnmente estas reglas se establecen mediante un sistema de ecuaciones diferenciales de las variables de estado.

En ocasiones es difícil conocer el valor de las variables de estado siendo la salida el único valor medible del sistema, forzando el uso de un modelo de caja negra con una estructura que solo tome en cuenta la relación entrada-salida. La ecuación que relaciona directamente las entradas y salidas del sistema sin tomar en cuenta explícitamente las variables de estado es la llamada ecuación de entrada/salida del sistema.

Como se mencionó anteriormente los pares entrada salida se obtienen tras un proceso de experimentación y muestreo a intervalos regulares, entonces, se puede decir que el problema de identificación consiste en determinar la ecuación discreta entrada/salida del sistema que relacione las muestras de salida u observaciones obtenidas experimentalmente $y(t), y(t-1), \ldots$ con las muestras de la entrada $u(t), u(t-1), \ldots$, el modelo NARMAX[8] se plantea como un conjunto de ecuaciones diferenciales no lineales de la forma

$$y(t) = f(y(t-1), \ldots, y(t-n_y), u(t-1), \ldots, u(t-n_u), e(t-1), \ldots, e(t-n_e)) + e(t)$$

Donde $u(t)$ y $y(t)$ son los vectores de entrada y salida, n_u y n_y son los retrasos máximos o la cantidad de estados anteriores a tomar en cuenta respectivamente, $e(t)$ es el ruido que existe en las muestras pero no puede ser medido directamente y corresponde directamente al error de prediccion, y n su retraso.

La identificación de un modelo NARMAX involucra como se mencionó anteriormente estimar la estructura del sistema (ecuación y parámetros) lo que no es tarea fácil ya que en teoría las posibles implementaciones no lineales de $f(\cdot)$ son infinitas. En la práctica uno de los métodos más populares para encontrar un modelo NARMAX son las redes neuronales como se verá más adelante.

2.1.3. Modelado de sistemas no lineales

Para el modelado de un sistema no lineal del cual se desconoce su estructura se requiere un enfoque de caja negra que puede ser representado usando el modelo NARMAX, donde se caracteriza el sistema con una ecuación diferencial no lineal desconocida $f(\cdot)$ la cual es generalmente muy compleja y el conocimiento previo de su estructura es a menudo nulo, por lo tanto la solución es aproximar $f(\cdot)$ con alguna función conocida simple.

En el presente trabajo, se aproxima el comportamiento de un sistema no lineal usando redes neuronales artificiales con el siguiente modelo

$$y(k) = f(y(k-1), \ldots, y(k-n_y), u(k), \ldots, u(k-n_u))$$

Donde $f(\cdot)$ representa el comportamiento del sistema, $u(k)$y $y(k)$ son escalares medibles que representan la entrada y salida en el instante k, n_y y n_u son los últimos valores de la salida y la entrada respectivamente que son considerados en $f(\cdot)$.

Este sistema discreto puede clasificarse en dos categorías dependiendo el valor que tomen n_y y n_u.

1. Modelo de predicción

   para valores de $n_y > 0$ y $n_u > 0$.

$$\hat{y}(k) = f(y(k-1), \ldots, y(k-n_y), u(k), \ldots, u(k-n_u))$$

Se puede predecir la salida del modelo, es decir, calcular una aproximación de la salida basándose en observaciones pasadas de las entradas y salidas del sistema.

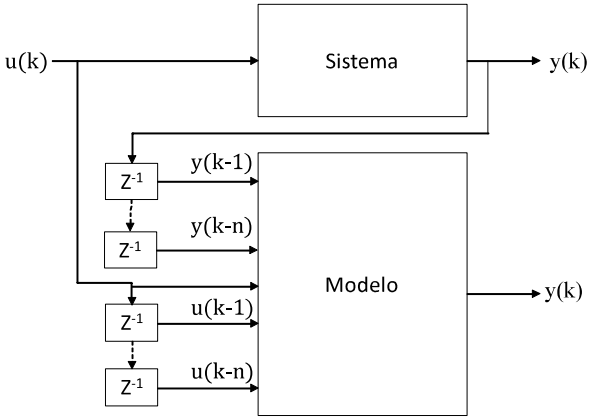


Figura 2.2: Modelo de predicción

2. Modelo de simulación

   para valores de $n_u > 0$ y $n_y = 0$

$$\hat{y}(k) = f(u(k), \ldots, u(k - n_u))$$

   El uso principal de un modelo es simular su salida, es decir, calcular $y(k)$ para valores de entrada dados.

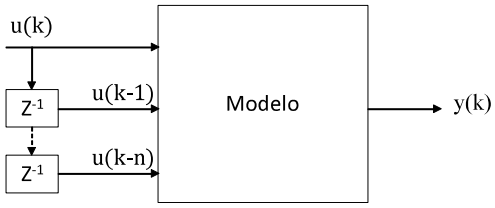


Figura 2.3: Modelo de simulación

2.2. Aprendizaje con gradiente descendente

El gradiente descendente es uno de los algoritmos de optimización más comúnmente usados en el entrenamiento de las redes neuronales y para entender el motivo, se debe de comprender de manera general el propósito de las redes neuronales a través de un pequeño ejemplo, más adelante se retomará nuevamente y con lujo de detalle este tema.

Suponiendo que se tienen N observaciones del comportamiento Y de un evento denotadas por y_i donde $i = 1 \dots N$, tales observaciones están relacionadas de alguna manera con $X_i = (x_{i1}, x_{i2}, \dots, x_{in})$ que son las características que determinan tales comportamientos. En términos simples, el objetivo es proponer una función $f(X_i, \theta)$ tal que $f(X_i, \theta) = y_i$, reduciendo el problema a encontrar $\theta \in \mathbb{R}^d, d \in \mathbb{Z}^+$. Para tal efecto, se necesita un método de evaluación de θ que diga que tan bien se cumple la relación propuesta, así de inmediato se observa que dado θ, si este cumple con la relación propuesta entonces, $\forall X_i \rightarrow y_i - f(X_i, \theta) = 0$, esto quiere decir, que se necesita conocer la diferencia entre el valor estimado y el valor real como medida de evaluación de θ, una de las métricas comúnmente usadas para este fin es el error cuadrático medio dado por

$$E = \frac{1}{N} \sum_i (y_i - f(X_i, \theta))^2$$

En la jerga del aprendizaje automático, a esta métrica también se le conoce como función de pérdida o función de coste.

Sin perder de vista el objetivo, si la diferencia entre el valor estimado y el valor real debería ser siempre cero entonces se debe encontrar un valor de θ que minimice la función de coste, para lo cual se usará el gradiente como se explica a continuación.

2.2.1. Gradiente descendente

Dada una función de coste $J(\theta)$, donde $\theta \in \mathbb{R}^d, d \in \mathbb{Z}^+$ representa los parámetros del modelo de un sistema, el gradiente descendente es un mecanismo que tiene como objetivo minimizar $J(\theta)$ ajustando el valor de los parámetros, este consiste en calcular el gradiente de $J(\theta)$ que se denota como $\nabla J(\theta)$ y se refiere al cálculo de todas las derivadas parciales de esta función, es decir $\nabla J(\theta) = \frac{\partial J(\theta)}{\partial \theta}$.

La interpretación del $\nabla J(\theta)$ resulta bastante conveniente, por ejemplo, para una función $f(x, y)$ su gráfica resultaría en una imagen similar a un terreno con montes, valles y depresiones, además dado un punto (x_0, y_0) arbitrario el gradiente de $f(x, y)$ en ese punto representa la dirección en la que habrá que moverse para llegar a la cima del monte

más cercano, entonces si el punto (x, y) se mueve en la dirección del gradiente siempre subirá a la cima de uno de los montes, dicho en otras palabras si el punto (x, y) se mueve en la dirección del gradiente siempre se incrementa el valor de la función f, entonces, se puede concluir que para minimizar $J(\theta)$ el movimiento debe ser hacia el lado contrario de la dirección del gradiente. En este punto surgen otras interrogantes como ¿Qué tan grandes deben ser los intervalos de movimiento o pasos?, ¿Cómo se puede asegurar que el resultado no es un mínimo local?, ¿Cual debe ser el punto de partida?, etc. Hoy en día aún no existe una solución definitiva a estas cuestiones y existen diversas versiones o algoritmos basados en la idea del gradiente descendente, las cuales difieren principalmente en los datos usados para calcularlo, esto conduce a una dicotomía entre la presición y el tiempo.

Gradiente descendente por lotes

El gradiente descendente por lotes o *Batch gradient descent* en inglés, es también conocido como *Vanilla gradient descent* y utiliza todo el conjunto datos para el cálculo del gradiente usando la siguiente ecuación

$$\theta = \theta - \eta \nabla J(\theta)$$

Donde η, llamado factor de aprendizaje establece la velocidad de descenso.

El algoritmo consiste en los siguientes pasos:

1. Inicializar θ aleatoriamente
2. Iterar desde hasta que $\nabla J(\theta) = 0$ o hasta alcanzar cierto número de iteraciones conocidas como épocas en el aprendizaje automático.

   2.1 Calcular el gradiente $\nabla J(\theta) = \frac{\partial J(\theta)}{\partial \theta}$

   2.2 Actualizar θ con la regla $\theta = \theta - \eta \nabla J(\theta)$

Para entender completamente el procedimiento se verá un ejemplo. Si se tiene un vector de N muestras(x_i, y_i) que representan el par (observacion, característica) de un sistema de la forma $y = ax + b$ y se desean encontrar los parámetros a y b que determinan el sistema, se puede usar SGD tomando como función de coste el error cuadrático medio MSE por sus siglas en inglés descrito como

$$E = \frac{1}{N}\sum_{i}^{N}(y_i - f(x_i, \theta))^2$$

donde $\theta = (a, b)$. Lo siguiente es entonces calcular las derivadas parciales

$$\begin{aligned}
\frac{\partial E}{\partial a} &= -\frac{2}{N}\sum_{i}^{N}(y_i - f(x_i, \theta))\frac{\partial (y_i - f(x_i, \theta))}{\partial a} = -\frac{2}{N}\sum_{i}^{N}(y_i - f(x_i, \theta))x \\
\frac{\partial E}{\partial b} &= -\frac{2}{N}\sum_{i}^{N}(y_i - f(x_i, \theta))\frac{\partial (y_i - f(x_i, \theta))}{\partial b} = -\frac{2}{N}\sum_{i}^{N}(y_i - f(x_i, \theta))
\end{aligned}$$

y utilizar la regla de actualizacion $\theta = \theta - \eta\nabla J(\theta)$ para cada iteración.

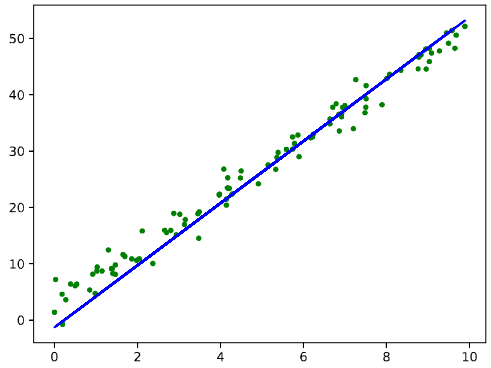


Figura 2.4: Resultado de SGD $y = ax + b$

Dado que para realizar una actualización de los parámetros es necesario recorrer todo el conjunto de datos, entonces, este método puede llegar a ser demasiado lento e intratable en el caso de tener un conjunto de datos muy grande. Con este algoritmo se realizan cálculos redundantes para conjuntos de datos muy grandes debido a que realiza cálculos similares para observaciones muy parecidas del conjunto de entrenamiento.

Gradiente descendente estocástico

A diferencia del gradiente descendente por lotes, con el gradiente descendente estocástico se actualizan los parámetros para cada elemento del conjunto de entrenamiento y consiste en los siguientes pasos:

1. Inicializar θ aleatoriamente

2. Iterar desde hasta que $\nabla J(\theta) = 0$ o hasta alcanzar cierto número de épocas

   2.1 Iterar desde $i = 1$ hasta N donde N es el número de observaciones

   2.1.1 Calcular el gradiente $\nabla J(x_i, \theta) = \frac{\partial J(x_i, \theta)}{\partial \theta}$

   2.1.2 Actualizar θ con la regla $\theta = \theta - \eta \nabla J(x_i, \theta)$

Uno de los inconvenientes de este algoritmo es que dada la superficie del gradiente no se puede asegurar un mínimo global y la optimización de este problema resulta ser NP-completo. Sin embargo, en general el gradiente descendente estocástico es más rápido que el gradiente descendente por lotes. A diferencia del gradiente descendente por lotes, el SGD realiza una actualización para cada observación con una fuerte influencia en la función objetivo.

Gradiente descendente por minilotes

Este método toma lo mejor de los métodos anteriores realizando una actualización cada cierto número de elementos del conjunto de entrenamiento. Normalmente se utilizan lotes que van de los 50 a 256 datos, esto depende del problema a resolver. El gradiente descendente por minilotes es el método más utilizado actualmente y por comodidad de ahora en adelante al igual que en la mayoría de la literatura, este método será llamado simplemente SGD.

Sin embargo, el SGD además de ser un algoritmo que no garantiza la convergencia y que de hecho en la práctica es raro que suceda, su éxito depende de la elección adecuada del factor de aprendizaje η, en general se sabe que si este parámetro es muy pequeño el avance hacia la convergencia será lento y si se escoge demasiado grande entonces incluso puede llevar a una divergencia. Debido a esto se han creado diversos métodos de optimización[9] de entre los cuales algunos se discuten a continuación.

2.2.2. Momentum

Este método es conocido también como SGD con momentum o ímpetu, puede ser aplicado al aprendizaje por lotes o mini lotes y es ampliamente utilizado, probablemente sea el más común en el aprendizaje de redes neuronales.

Como se sabe, el éxito del SGD depende de la elección adecuada del factor de aprendizaje η, por ejemplo, la elección del parámetro η está directamente relacionada

con la velocidad del proceso de minimización de la función, es decir, si se escoge un η muy pequeño el proceso de minimización de la función objetivo será lento y a medida que se seleccione η más grande el proceso se acelerará, sin embargo, si η es demasiado grande entonces, el valor final puede oscilar alrededor del óptimo sin llegar a alcanzarlo jamás o incluso podría alejarse completamente de él.

En las figuras 2.5,2.6 y 2.7 se muestra la gráfica de contorno de la función $\frac{z^2}{c} = \frac{x^2}{a} + \frac{y^2}{b}$ y el camino que siguen los parámetros durante el SGD para minimizar su valor, que en este caso coincide con el centro de la gráfica, en cada gráfica se utiliza un factor de aprendizaje distinto, pequeño, grande y muy grande respectivamente, de esta forma se puede observar claramente la importancia de la correcta elección de η, para el éxito del SGD como se comentó anteriormente.

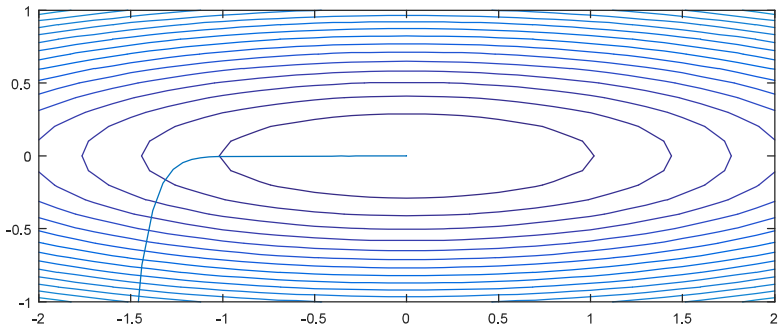


Figura 2.5: SGD con un factor de aprendizaje pequeño y adecuado ya que llega finalmente al valor óptimo

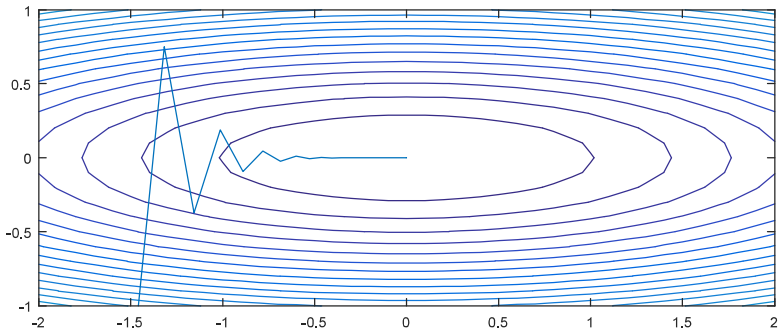


Figura 2.6: SGD con un factor de aprendizaje grande y adecuado ya que llega finalmente al valor óptimo

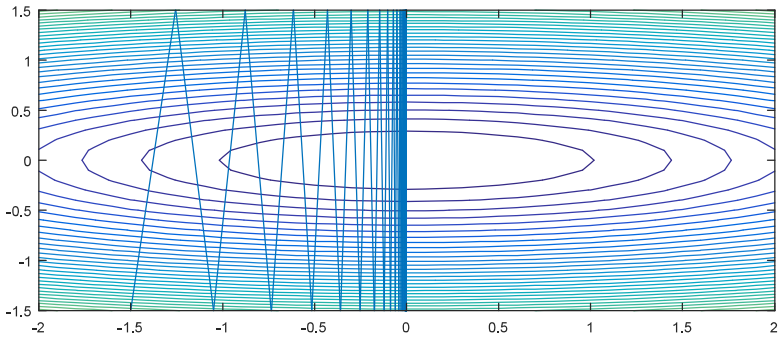


Figura 2.7: SGD con un factor de aprendizaje muy grande, siempre oscila alrededor del óptimo.

Este tipo de escenarios con barrancos, donde la superficie se curva mucho más en una dimensión que en otra son comunes alrededor de un óptimo local y siempre existe el mismo problema para la elección de η en estas situaciones.

Aquí es donde entra el método del momentum, si se observan las figuras anteriores se puede concluir que el gradiente descendente ya sea por lotes o mini lotes conlleva cierto número de pasos oscilantes hasta alcanzar el mínimo, y estas oscilaciones o subidas y bajadas hacen el proceso lento y dependen del factor de aprendizaje para no ser demasiado excesivas y llevar a resultados erróneos, de ahí la importancia de prevenir estas oscilaciones, la forma más simple e inmediata es utilizar forzosamente un factor de aprendizaje muy pequeño pero no siempre es la mejor estrategia ya que la convergencia puede tomar demasiadas iteraciones. Otra forma de abordar este problema es acelerar el SGD en la dirección relevante que en este caso es el eje horizontal y amortiguar las oscilaciones en el eje vertical para evitar pérdidas de tiempo agregando una fracción α de la velocidad anterior a la velocidad actual, a esto se le conoce como el método del momentum o ímpetu. Se puede imaginar por ejemplo que se arroja una pelota en la superficie del error, en un principio esta pelota seguirá la dirección del gradiente ya que por efecto de la gravedad se sabe que la pelota se irá al fondo de la superficie, ahora bien si se igualan las condiciones del SGD, entonces la pelota llevaría siempre la misma velocidad, de manera que si su movimiento es muy lento entonces en algún momento llegará al fondo o a las inmediaciones de este, sin embargo, si va muy rápido la inercia de la pelota hará que aunque llegue al fondo, vuelva a subir entrando en un ciclo de oscilaciones infinito o incluso se podría llegar a salir si su velocidad es demasiado alta, pues bien, en la vida real esto no sucede gracias a la fricción que decrementa la velocidad de la pelota conforme

se acerca al fondo, esto es el equivalente al parámetro α o momento que influye en la velocidad como lo dicta la siguiente ecuación

$$v(t) = \alpha v(t-1) + \eta \nabla J(\theta)(t)$$

Donde $v(t)$ se define de manera recursiva usando el factor de atenuación o momento α cuyo valor es ligeramente menor a 1, más la ya conocida regla de incremento del SGD que involucra el gradiente multiplicado por el factor de aprendizaje η. Para trabajar en términos de los parámetros y su incremento se puede cambiar la ecuación del *momentum* como se muestra a continuación

$$\begin{aligned} \Delta\theta(t) &= v(t) = \alpha v(t-1) + \eta \nabla J(\theta)(t) \\ \Delta\theta(t) &= \alpha \Delta\theta(t-1) + \eta \nabla J(\theta)(t) \end{aligned}$$

Entonces se tiene que la regla de aprendizaje de SGD con *momentum* está dada por

$$\theta = \theta - \Delta\theta(t)$$

Dependiendo la literatura, los signos en las ecuaciones pueden ser intercambiados, pero en resumen el usar el ímpetu o *momentum* se traduce en obtener una rápida convergencia y reducir las oscilaciones.

Sin embargo, el *momentum* presenta un grave problema, ya que cuando el resultado se acerca al objetivo el movimiento hacia él es muy rápido y no existe forma alguna de indicar alguna reducción en la cantidad de movimiento lo que resultaría en la pérdida del mínimo.

2.2.3. Adagrad

Con Adagrad se busca resolver el problema del *momentum*, ya que como su nombre lo dice es un algoritmo adaptativo de gradiente, el cual adapta el factor de aprendizaje de acuerdo con los parámetros, realizando mayores actualizaciones para parámetros poco frecuentes y actualizaciones más pequeñas para parámetros frecuentes, por tal motivo es ideal para lidiar con poca información. Adagrad usa un factor de aprendizaje diferente para cada parámetro en cada tiempo t basado en los gradientes anteriores que fueron calculados para ese parámetro. Dado que el factor de aprendizaje η siempre decrece, en algún punto este es tan pequeño que el aprendizaje termina.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

2.2.4. AdaDelta

AdaDelta nace como evolución del Adagrad y busca detener el decremento de η, así entonces en lugar de sumar la raíz cuadrada de todos los gradientes anteriores se restringen los valores a un valor promedio de los gradientes anteriores. En resumen se calculan factores de aprendizaje individuales para cada parámetro, se calcula el momentum y se previene el decremento excesivo de los factores de aprendizaje.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[g^2] + \epsilon}} \cdot g_{t,i}$$

Actualmente es difícil predecir con seguridad que algoritmo de optimización conviene más a cada herramienta y en cada problema particular sin embargo dado que los algoritmos adaptativos toman lo mejor de cada uno de los algoritmos anteriores, estos parecieran ser la mejor opción para empezar a trabajar.

2.2.5. Adam

Es común ver publicaciones con nuevos algoritmos de optimización, sin embargo, la mayoría de las veces funcionan bien en problemas particulares y bajo ciertas condiciones. El algoritmo Adam fue introducido por primera vez en 2014 por Diederik P. Kingma y Jimmy Lei Ba en el artículo "Adam: A Method for Stochastic Optimization"[10], este algoritmo es simple y eficiente en su implementación, además de consumir pocos recursos y ser adecuado para problemas con una gran cantidad de parámetros, también ha demostrado su efectividad en una gran variedad de arquitecturas de *deep learning*.

Adam busca una estimación adaptativa del momentum por cada parámetro, mejorando los algoritmos de optimización anteriores.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t + \epsilon}} m_t$$

2.3. Redes Neuronales Recurrentes

El objetivo principal de las Redes Neuronales Recurrentes (RNN por sus siglas en inglés) es lidiar con tareas que usan la información de manera secuencial. A diferencia de

las redes neuronales tradicionales también conocidas como "feedforward neural networks.[en] las que se supone que las entradas y salidas son independientes entre sí y por lo tanto sólo permiten el modelado de relaciones estáticas, con las redes neuronales recurrentes al poseer propiedades dinámicas se pueden representar las relaciones a través del tiempo o el espacio.

Anteriormente las redes neuronales recurrentes resultaban complicadas en su entrenamiento debido a la gran cantidad de parámetros que manejan normalmente, sin embargo, hoy en día gracias al avance de las computadoras con su gran capacidad de almacenamiento, el uso de procesadores más potentes y al paralelismo entre otros, el entrenamiento de las RNN se ha vuelto manejable.

2.3.1. Perceptrón

El perceptrón es un modelo matemático simple que trata de emular el comportamiento de una neurona que son los elementos básicos del sistema nervioso y gracias a ellas podemos interactuar con nuestro entorno. Podemos describirlo como un modelo que tiene N entradas y una salida, cada entrada es multiplicada por una variable diferente para cada una de ellas llamada peso y finalmente se aplica una función no lineal a la suma de estos productos para obtener la salida $\hat{y}$. El modelo matemático del perceptrón sería entonces

$$\hat{y} = f((\sum_{i=0}^{N} x_i w_i) + b)$$

donde x_i son las entradas, w_i son los pesos y b es el sesgo de la función f no lineal conocida como función de activación, mientras más grande es b más facil es que el perceptrón se dispare, es decir, que su valor sea 1. La función de activación permite que un pequeño cambio en los pesos o en el sesgo genere un cambio a la salida. Si se establece lo anterior en forma vectorial tenemos

$$\hat{y} = f(XW + b)$$

con $X = x_0, x_1, \ldots, x_N$ y $W = w_0, w_1, \ldots, w_N$. La representación gráfica de este modelo se muestra en la figura 2.8

Las funciones de activación más comunes son la Sigmoide, TanH y ReLU aunque en teoría podría ser cualquier función siempre y cuando se aplique una transformación no lineal a los datos.

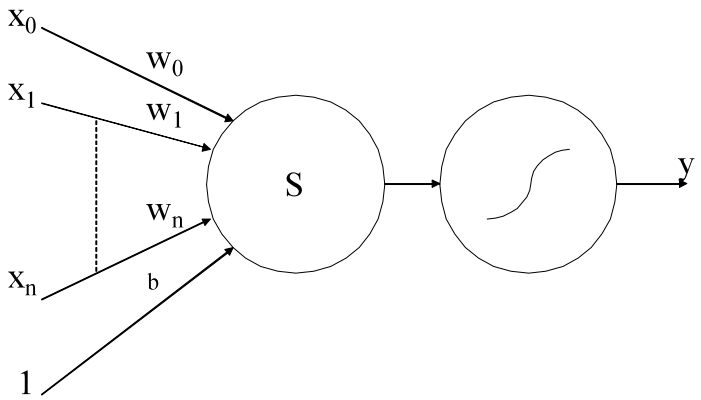


Figura 2.8: Representación gráfica del perceptrón

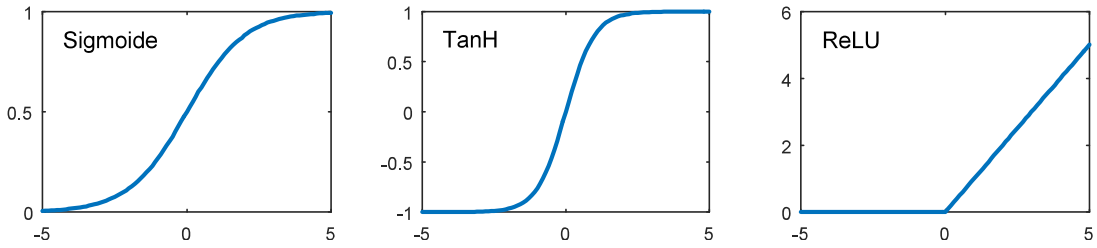


Figura 2.9: Funciones de activación comúnmente usadas en un perceptrón

2.3.2. Red neuronal de perceptrón multicapa (MLP)

Las redes neuronales se pueden ver como un mapa uno a uno, es decir, que a cada elemento X del conjunto de entrada, se le asocia siempre el mismo elemento y del conjunto de salida, las redes neuronales pueden aprender cualquier tipo de relación, es decir, que son capaces de aproximar el comportamiento de cualquier función suave siempre y cuando se tenga un conjunto de datos que muestre las relaciones entrada-salida de esta función y mientras mayor sea el tamaño de este conjunto resulta en una mejor aproximación.

La estructura mínima de una red neuronal de perceptrón multicapa consta de tres capas, la primera es la capa de entrada y cada nodo representa un elemento del vector X del conjunto de entrada, después están la capa oculta y la capa de salida compuestas de al menos un perceptrón cada una que representan los nodos de la capa. Si la salida de los nodos de cada capa son la entrada de cada nodo de la siguiente capa, entonces

la red tomaría el nombre de red neuronal con conexión hacia adelante de perceptrón multicapa completamente conectada *feedforward fully connected neural network* en inglés, sin embargo, dado que es la configuración más usada, en el presente trabajo y en la literatura en general se usa este tipo de arquitectura bajo el nombre de red MLP. En la figura se observa una representación gráfica de este tipo de red.

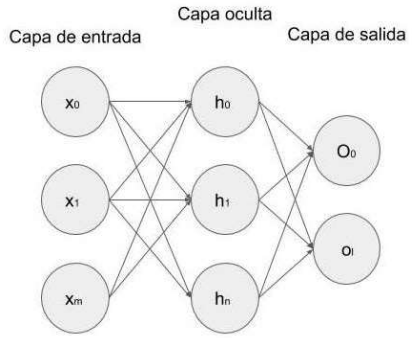


Figura 2.10: Perceptrón multicapa

Entrenamiento de una red MLP. (Retropropagación)

La aproximación de funciones haciendo uso de las redes neuronales depende de la correcta elección de los parámetros de la red que son los pesos W y los parámetros de ajuste b de las funciones de activación, pero entonces ¿Cómo se selecciona el valor de estos parámetros?, pues bien a este proceso de selección se le llama entrenamiento y el algoritmo de entrenamiento de redes neuronales más exitoso es el de retropropagación o *backpropagation* en inglés, el cuál utiliza la regla de la cadena para calcular la derivada de la función de costo o pérdida con respecto a cada uno de los parámetros de la red para así poder minimizarla ajustando estos parámetros.

Para minimizar el error de la red neuronal mostrada en la figura 2.10 usando SGD y como función de activación de cada neurona la función sigmoide, se considera la siguiente notación.

x_j^l : Entrada al nodo j de la capa l.

w_{ij}^l : Peso de la capa previa $l-1$ del nodo i a la capa l nodo j.

$\sigma(x)$: Función de activación sigmoide $\frac{1}{1-e^x}$.

b_j^l : Sesgo del nodo j de la capa l.

O_j^l : Salida del nodo j de la capa l.

y_j : Valor deseado del nodo j capa de salida.

Entonces el error cuadrático medio entre las salidas deseadas y_k y la salidas obtenidas O_k de los nodos de la capa de salida K esta dado por $E = \frac{1}{2}\sum_1^{N_k}(O_k - y_k)^2$, donde N_k es el número de nodos de la capa de salida, se utiliza $\frac{1}{2}$ por conveniencia en el cálculo de las derivadas. Para usar el método del gradiente descendente se necesita calcular $\frac{\partial E}{\partial w_{ij}^l}$, sin embargo, el cálculo para los nodos de salida y los de capas ocultas son diferentes, así que se desarrollan de manera independiente.

Gradiente para un nodo de capa de salida Se tiene que

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}}\frac{1}{2}\sum_1^{N_k}(O_k - y_k)^2 = (O_k - y_k)\frac{\partial}{\partial w_{jk}}O_k$$

Como se sabe, la salida O_k está dada por la función de activación $\sigma(x_k)$, entonces

$$\frac{\partial E}{\partial w_{jk}} = (O_k - y_k)\frac{\partial}{\partial w_{jk}}\sigma(x_k) = (O_k - y_k)\sigma(x_k)(1 - \sigma(x_k))\frac{\partial}{\partial w_{jk}}x_k$$

Se sabe que la entrada x_k está dada por $O_j w_{jk}$ y que $\sigma(x_k) = O_k$, así que

$$\frac{\partial E}{\partial w_{jk}} = (O_k - y_k)O_k(1 - O_k)O_j$$

Con propósitos de simplificar la notación se hace $\delta_k = (O_k - y_k)O_k(1 - O_k)$, entonces

$$\frac{\partial E}{\partial w_{jk}} = O_j\delta_k$$

Gradiente para un nodo de capa oculta Se tiene que

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\frac{1}{2}\sum_1^{N_k}(O_k - y_k)^2$$

Se sabe que w_{ij} está directamente involucrado en todas las salidas O_k ya que $O_k = \sigma(x_k) = \sigma(O_j w_{jk}) = \sigma(\sigma(O_i w_{ij})w_{jk})$, entonces

$$\frac{\partial E}{\partial w_{ij}} = \sum_{1}^{N_k}(O_k - y_k)\frac{\partial}{w_{ij}}O_k = \sum_{1}^{N_k}(O_k - y_k)\frac{\partial}{w_{ij}}\sigma(x_k) = \sum_{1}^{N_k}(O_k - y_k)\sigma(x_k)(1-\sigma(x_k))\frac{\partial}{w_{ij}}x_k$$

Aplicando la regla de la cadena y sustituyendo $O_k = \sigma(x_k)$ y $x_k = O_j w_{jk}$

$$\frac{\partial E}{\partial w_{ij}} = \sum_{1}^{N_k}(O_k - y_k)O_k(1-O_k)\frac{\partial}{O_j}x_k\frac{\partial}{w_{ij}}O_j = \sum_{1}^{N_k}(O_k - y_k)O_k(1-O_k)w_{jk}\frac{\partial}{w_{ij}}O_j$$

Reordenando los términos se tendría

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial}{w_{ij}}O_j\sum_{1}^{N_k}(O_k - y_k)O_k(1-O_k)w_{jk}$$

Como $Oj = \sigma(xj)$

$$\frac{\partial E}{\partial w_{ij}} = O_j(1-O_j)\frac{\partial}{w_{ij}}x_j\sum_{1}^{N_k}(O_k - y_k)O_k(1-O_k)w_{jk}$$

Como $x_j = O_i w_{ij}$

$$\frac{\partial E}{\partial w_{ij}} = O_j(1-O_j)O_i\sum_{1}^{N_k}(O_k - y_k)O_k(1-O_k)w_{jk}$$

Sabiendo que $\delta_k = (O_k - y_k)O_k(1-O_k)$ y haciendo $O_j(1-O_j)\sum_1^{N_k}\delta_k w_{jk}$, entonces se tendría

$$\frac{\partial E}{\partial w w_{ij}} = O_i\delta_j$$

Gradiente para el sesgo de la función de activación Como $\frac{\partial O}{\partial b} = 1$ siempre entonces siguiendo las ecuaciones anteriores se tiene que para cualquier capa l

$$\frac{\partial E}{\partial b} = \delta_l$$

Algoritmo de retropropagación

1. Obtener la salida de la red con los datos de entrada

2. Calcular los δ_k

3. Calcular los δ_j

4. Actualizar pesos y sesgos con la técnica del gradiente descendiente

   $\Delta w = -\eta \delta_l O_{l-1}$

   $\Delta b = -\eta \delta_l$

5. Entonces:

   $w = w + \Delta w$

   $b = b + \Delta b$

En resumen, después de la propagación hacia adelante se produce una salida que se compara con el valor deseado generando un error el cual se propaga de regreso actualizando los parámetros de la red, si la función de activación es distinta entonces el cálculo de los gradientes varia un poco del mostrado anteriormente, pero el algoritmo es exactamente el mismo.

Gradiente de fuga en las redes MLP

El proceso de entrenamiento siempre es un problema de optimización y en este sentido el gradiente descendente además de ser demasiado lento tiene un serio problema conocido como gradiente de fuga o *Vanishing Gradient* en inglés que se presenta cuando alguna de sus derivadas se vuelve cero o resulta en un valor tan pequeño que ya no es computable. Este problema existe especialmente cuando se tienen valores muy altos a la salida de la función de activación. Actualmente existen distintas soluciones a este problema, por ejemplo, usar como función de activación en lugar de una sigmoide que era la más utilizada, una función ReLU. Sin embargo, si una neurona es pobremente inicializada puede no activarse jamás es decir que se pueden tener grandes partes de la red con ReLUs inactivas.

2.3.3. Redes Neuronales Recurrentes

Las redes neuronales recurrentes o RNNs (*Recurrent neural networks*) por sus siglas en inglés son modelos de aprendizaje capaces de almacenar información selectivamente de toda una secuencia de pasos, por esta razón en ocasiones se dice que es un modelo con capacidad de memoria, el cual retiene información del proceso completo de aprendizaje. Las RNNs pueden representar series de tiempo, fragmentos de audio, marcos de video,

cualquier cosa que se presente por medio de secuencias de datos, entendiendo secuencia como un conjunto de datos o valores en el que los valores futuros dependen de los valores previos, por ejemplo, una oración, una función, una onda de sonido, etc. Es decir, aquellas tareas en donde falla la suposicion de dependencia hecha por las redes MLP.

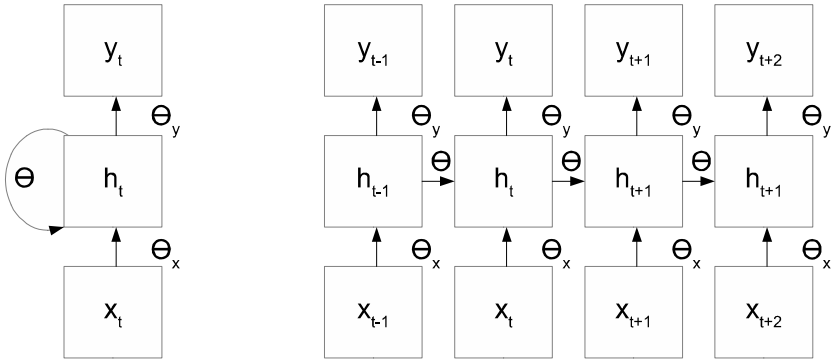


Figura 2.11: Representación gráfica de una RNN de manera a)Compacta y b)Desplegada a través del tiempo

Inicialmente se puede visualizar una red neuronal recurrente como una red MLP en la cual la salida de cada capa retroalimenta nuevamente su entrada. Para facilitar el estudio de la arquitectura de las RNN se puede pensar en la estructura más simple compuesta de una capa de entrada x una capa oculta h y una capa de salida y, esta arquitectura se puede representar gráficamente de dos formas de manera compacta y desplegada a través del tiempo como se muestra en la figura 2.11 donde ambas representaciones son equivalentes.

Cuando se habla de la arquitectura de la red se hace referencia al número de capas, nodos por capa y conexiones entre ellos, en la arquitetura de la RNN expuesta anteriormente se puede observar que la salida y_t esta influenciada obviamente por h_t sin embargo dado que la entrada de h_t es x_t y h_{t-1}, entonces se puede decir que x_{t-1} también influye en la salida y_t e incluso a las salidas posteriores. Para realizar el cálculo de y_t tomando en cuenta el diseño anterior, se deducen las siguientes ecuaciones

$$h_t = W\phi(h_{t-1}) + W_x x_t$$
$$y_t = W_y \phi(h_t)$$

Donde x_t es la entrada, y_t la salida, W, W_x, W_y los parámetros de ajuste y φ representa una función no lineal arbitraria base como una ReLU, $tanh$, una sigmoide, etc.

Un aspecto importante de la arquitectura de la red es que la capa oculta h_t o el así llamado estado de la celda en el argot de las RNN en el tiempo t contiene la información de los estados anteriores lo que resuelve aparentemente el problema de la dependencia a largo plazo.

Cabe mencionar que W, W_x, W_y son los mismos a través del tiempo, lo que reduce el número de parámetros necesarios y permite procesar entradas de un tamaño temporal arbitrario.

Retropropagación a través del tiempo (BPTT)

En las RNN al igual que con las redes MLP el entrenamiento se puede realizar usando *backpropagation,* pero dado que este toma en cuenta no solo el estado presente sino también todos los estados anteriores, entonces el algoritmo es llamado Retropropagación a través del tiempo o "Backpropagation Through Time (BPTT)" en inglés, con el BPTT se debe tomar en cuenta el error o pérdida en cada instante de tiempo y por lo tanto el error o pérdida total será igual a la suma de todos lo errores a través del tiempo.

El error total esta dado por la siguiente ecuación

$$\text{Error total } = J(\Theta) = \sum_t J_t(\Theta)$$

Donde Θ representa los parámetros del modelo.

Así como en el algoritmo de retropropagación, se calcula un gradiente a cada parámetro P y el gradiente total es la suma de los gradientes del parámetro P a través del tiempo.

$$\frac{\partial J}{\partial P} = \sum_t \frac{\partial J_t}{\partial P_t}$$

Si se calcula el gradiente para los pesos W, entonces se tiene

$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J}{\partial W_t}$$

Ahora usando como ejemplo la figura 2.12, al enfocarse en el tiempo $t = 2$ y usando la regla de la cadena se tiene que

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2}\frac{\partial y_2}{\partial h_2}\frac{\partial h_2}{\partial W}$$

Sin embargo, para el término $\frac{\partial h_2}{\partial W}$ se debe resaltar que $h_2 = W\phi(h_1) + W_x x_2$, y dado que h_1 también depende de W entonces no se puede tratar a $\frac{\partial h_2}{\partial W}$ como una constante sino que se debe agregar a ese término todos los gradientes relacionados de los estados anteriores lo que indica que los pesos W contribuyen al error en el tiempo t basados en cómo contribuyen en los tiempos anteriores, es decir, se toman en cuenta las contribuciones al error de W en cada uno de los momentos anteriores.

$$\frac{\partial h_2}{\partial W} + \frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W} + \frac{\partial h_2}{\partial h_0}\frac{\partial h_0}{\partial W}$$

Entonces generalizando se tiene que

$$\frac{\partial J_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial J_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W}$$

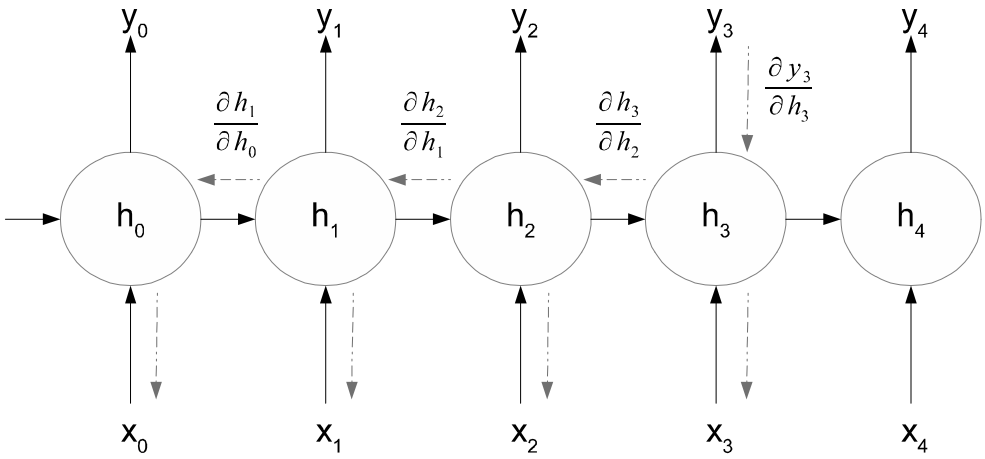


Figura 2.12: Retropropagación del error a través del tiempo

Gradiente de fuga en las RNN

Desafortunadamente cuando se usa BPTT existe exactamente el mismo problema que en el algoritmo de retropropagación mencionado anteriormente, el llamado "Vanishing

Gradient", que viene de la regla de la cadena aplicada en la retropropagación, y este se hace más grave debido a que como la profundidad de las RNN puede ser arbitrariamente larga, si en algún punto el gradiente es demasiado pequeño entonces lleva a todos los demás gradientes a cero y se ignora la mayoría de la secuencia.

Se aprecia perfectamente en la ecuación $\frac{\partial J_t}{\partial W}$, por ejemplo, si se calcula el último instante temporal de una secuencia muy larga entonces se tendría que

$$\frac{\partial h_n}{\partial h_k} = \frac{\partial h_n}{\partial h_{n-1}} \frac{\partial h_{n-1}}{\partial h_{n-2}} \cdots \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial h_0}$$

Se puede observar que mientras la brecha de tiempo se hace más grande entonces esta regla de la cadena será cada vez más grande, como se sabe cada una de estas derivadas es la derivada del estado en h en el tiempo n respecto a los estados anteriores y cada uno de estos estados es calculado por la multiplicación de las entradas por sus pesos para después pasar a través de una función de activación, por lo tanto, en este punto se trabaja con números pequeños casi siempre menores a uno, así que lo que realmente se está haciendo al aplicar la regla de la cadena es multiplicar muchos números menores a uno y mientras más larga sea la secuencia se estará minimizando cada vez más esa cantidad. Entonces, se puede decir que mientras más larga sea la secuencia más pequeños son los gradientes y dado que se ajustan los pesos de acuerdo con estos gradientes entonces solo se estarían tomando en cuenta los últimos estados y los parámetros estarían limitados a capturar únicamente las dependencias a corto plazo.

Una vez dicho esto se observa que el problema es difícil de resolver únicamente escogiendo la función de activación correcta, por tal motivo se desarrollaron las llamadas celdas de compuerta en donde se sustituyen las neuronas del diseño original por unidades más complejas internamente con compuertas que controlan la información que pasa a través de ellas, como por ejemplo las LSTM.

2.3.4. Celda LSTM

Hoy en día, normalmente cuando se habla de redes neuronales recurrentes se hace referencia a modelos LSTM (*Long short-term memory*) por sus siglas en inglés que surgen principalmente como un método para resolver el problema de *Vanishing Gradients,* y lo logra gracias a que se pueden conservar de manera íntegra las entradas importantes, es

decir, que la dependencia a través del tiempo se da por medio de la adición, por lo tanto, no se crean grandes cadenas de productos como en la regla de la cadena.

Todas las redes neuronales recurrentes en general se componen de ciertos bloques o módulos encadenados en el tiempo como se puede apreciar en la figura 2.11, cabe aclarar que esta dependencia no necesariamente represente un espacio temporal, en este caso, dado que el conjunto de sucesos con los que trabajamos es finito o infinito numerable se puede decir que se trabaja con un espacio discreto donde cada uno de los eventos se representa por k con $k \epsilon N$ que es el número de elementos del conjunto de entrada.

Una red LSTM es una red neuronal recurrente que usa como neuronas en sus capas ocultas bloques o celdas especiales más complejas que un perceptrón, como se puede ver en la figura 2.13, donde las flechas gruesas representan vectores, las flechas delgadas valores escalares y las discontinuas los ciclos que hacen de la red una red neuronal recurrente.

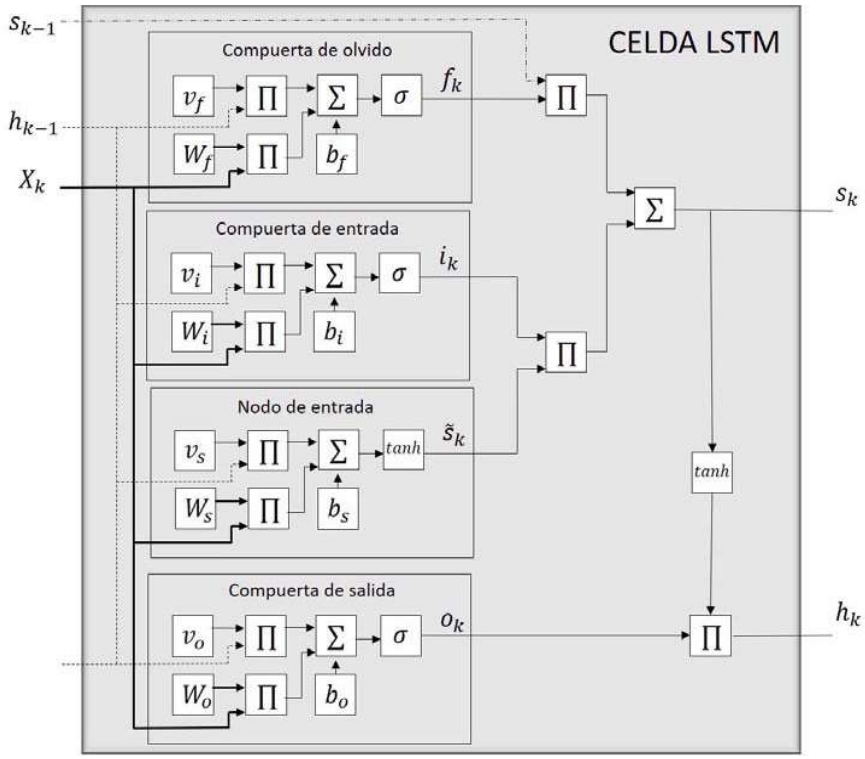


Figura 2.13: Representación gráfica detallada de una celda LSTM

Básicamente se puede describir el funcionamiento de una celda LSTM mencionando

sus tres características principales. La primera es la capacidad de desechar información irrelevante resultante de la etapa previa, la segunda es el poder actualizar selectivamente el estado interno de la celda y por último controlar lo que finalmente será mostrado a la salida. En resumen, se tiene que con este tipo de celdas la red controla tres cosas, qué olvidar, qué recordar y qué salida producir, o dicho de otra manera, se puede seleccionar aquello que la red debe y no debe aprender.

Un concepto importantísimo y diferenciador de las celdas LSTM es el llamado estado de la celda que se puede ver como un carril que atraviesa la celda desde la entrada s_{k-1} que es el estado anterior hasta convertirse en s_k o estado de la celda después de pasar por algunas modificaciones lineales, estas modificaciones dotan a las LSTM de la habilidad de escoger qué información se debe olvidar y cual se debe recordar, gracias a esto las celdas LSTM son llamadas en ocasiones celdas de memoria. Estas modificaciones son reguladas por entidades llamadas compuertas, las LSTM se componen de tres compuertas que son un enfoque distintivo de las LSTM, las compuertas son básicamente neuronas tradicionales cuyo valor es usado para multiplicar el valor de otro nodo y esto las hace parecer compuertas ya que si el valor de la compuerta es uno permite el flujo total de la información y si es cero entonces la compuerta se cierra y la información no pasa. También dentro de una celda LSTM se tiene un nodo de entrada y algunas modificaciones lineales para obtener las salidas, estos elementos mostrados a detalle en la figura 2.13 se analizan a continuación.

Compuerta de olvido

La compuerta de olvido es la encargada de decidir qué información no forma parte del estado de la celda y se realiza con una función sigmoide que toma en cuenta la salida de la celda en un paso anterior h_{k-1} y la entrada X_k multiplicadas por sus respectivos pesos, y como salida f_k se obtiene un número entre 0 que representa olvidar completamente la información y 1 que guarda entonces la información completa.

$$f_k = \sigma(W_f \cdot X_k + v_f \cdot h_{k-1} + b_f)$$

Compuerta y nodo de entrada

El nodo de entrada es una estructura que funciona en la manera tradicional tomando la entrada de X_k y la suma ponderada de este vector pasa por una función de activación qué aunque en el trabajo original es una función sigmoide, típicamente se usa una función

tanh, la cual genera información candidata a ser tomada en cuenta para agregar al estado de la celda y es regulada por la compuerta de entrada que decide qué información se debe conservar. Esta compuerta es la que decide qué información se debe conservar en el estado de la celda una función sigmoide que actualiza la información y una función tanh que crea nuevos vectores que podrían ser añadidos al estado, finalmente se combinan las salidas y este será el valor que actualice el estado.

$$\begin{aligned} i_k &= \sigma(W_i \cdot X_k + v_i \cdot h_{k-1} + b_i) \\ \hat{s}_k &= \tanh(W_s \cdot X_k + v_s \cdot h_{k-1} + b_s) \end{aligned}$$

Compuerta de salida

Como se mencionó anteriormente, las celdas LSTM permiten seleccionar qué información se mostrará a la salida, por lo tanto, primero se filtra la información que tiene el estado interno s_k con esta compuerta.

$$o_k = \sigma(W_o \cdot X_k + v_o \cdot h_{k-1} + b_o)$$

Salidas

Las celdas LSTM tienen dos salidas el estado s_k que permite las relaciones a largo plazo y la que será la entrada a la siguiente capa de la red, la salida h_k que muestra el estado s_k regulado por la compuerta o_k, normalmente el estado se pasa por una función *tanh* para que tenga el mismo rango dinámico que una neurona estándar, sin embargo, en ocasiones se omite ya que así es más sencillo entrenar la red..

$$\begin{aligned} s_k &= \hat{s}_k \cdot i_k + f_k \cdot s_{k-1} \\ h_k &= o_k \cdot \tanh(s_k) \end{aligned}$$

Representacion compacta de una celda LSTM

Una vez analizados a detalle los conceptos que soportan la estructura de una red LSTM es posible darse cuenta que la estructura interna de las compuertas y el nodo de entrada se diferencian únicamente en la función de salida por lo tanto en la figura 2.14 se muestra una representación compacta de la celda que ayudará a una mejor visualización además

de introducir también el concepto de retraso que es ampliamente usado en el argot del control automático generando así un mejor entendimiento para el posterior análisis del entrenamiento de este tipo de red.

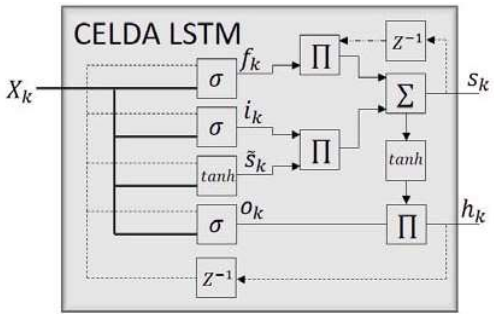


Figura 2.14: Representación compacta de una celda LSTM

En resumen, el algoritmo de propagación hacia adelante de una celda LSTM con compuerta de olvido está dado por las siguientes ecuaciones.

$$
\begin{aligned}
f_k &= \sigma(W_f \cdot X_k + v_f \cdot h_{k-1} + b_f) \\
i_k &= \sigma(W_i \cdot X_k + v_i \cdot h_{k-1} + b_i) \\
\hat{s}_k &= \tanh(W_s \cdot X_k + v_s \cdot h_{k-1} + b_s) \\
o_k &= \sigma(W_o \cdot X_k + v_o \cdot h_{k-1} + b_o) \\
s_k &= \hat{s}_k \cdot i_k + f_k \cdot s_{k-1} \\
h_k &= o_k \cdot \tanh(s_k)
\end{aligned}
$$

Por comodidad se pueden concatenar X_k y h_{k-1} en un mismo vector de entrada como $U_k = \begin{bmatrix} X_k \\ h_{k-1} \end{bmatrix}$ y agregar los valores escalares v a los vectores W, teniendo entonces $W_F = [W_f \; v_f]$, $W_I = [W_i \; v_i]$, $W_S = [W_s \; v_s]$, $W_O = [W_o \; v_o]$ y resultando en las siguientes ecuaciones.

$$
\begin{aligned}
f_k &= \sigma(W_F \cdot U_k + b_f) \\
i_k &= \sigma(W_I \cdot U_k + b_i) \\
\hat{s}_k &= \tanh(W_S \cdot U_k + b_s) \\
o_k &= \sigma(W_O \cdot U_k + b_o) \\
s_k &= \hat{s}_k \cdot i_k + f_k \cdot s_{k-1} \\
h_k &= o_k \cdot \tanh(s_k)
\end{aligned}
$$

BPTT para las LSTM

Como se hizo en la retropropagación de las MLP, se parte de una función de pérdida en base al error generado por un valor meta y_k al que se debe aproximar el resultado lo más cerca posible.

$$E = \sum_{k=1}^{N} (h_k - y_k)^2$$

Donde N es el tamaño de la secuencia.

Entonces el objetivo es minimizar este error a cada paso k para lo cual se usa la técnica del gradiente descendente lo que lleva a calcular el gradiente de E para algún parámetro P.

$$\frac{\partial E}{\partial P}$$

Dado que E solo depende de los valores h_k y de y_k entonces se puede usar la regla de la cadena

$$\frac{\partial E}{\partial P} = \sum_{k=1}^{N} \sum_{j=1}^{M} \frac{\partial E}{\partial h_k^j} \frac{\partial h_k^j}{\partial P}$$

donde h_k^j es la salida de la celda j y M es el número de celdas, ahora se sabe que

$$\frac{\partial E}{\partial h_k^j} = \frac{\partial}{\partial h_k^j} \sum_{k=1}^{N} (h_k - y_k)^2$$

Dado que la red propaga la información hacia adelante en el tiempo, un cambio en h_k^j no afecta la perdida anterior al paso k, por lo tanto, se puede reescribir la ecuación

anterior como

$$\frac{\partial E}{\partial h_k^j} = \frac{\partial}{\partial h_k^j} \sum_{l=k}^{N} (h_l - y_l)^2 = \frac{\partial E_k}{\partial h_k^j}$$

Donde E_k será el error acumulado desde k hacia adelante. De esta manera se puede reescribir la ecuación del gradiente como

$$\frac{\partial E}{\partial P} = \sum_{k=1}^{N} \sum_{j=1}^{M} \frac{\partial E_k}{\partial h_k^j} \frac{\partial h_k^j}{\partial P}$$

Se debe poner especial atención en E_k que es donde se encuentra la propagación a través del tiempo, ya que

$$\frac{\partial E_k}{\partial h_k^j} = \frac{\partial E_k}{\partial h_k^j} + \frac{\partial E_{k+1}}{\partial h_k^j}$$

Entonces para calcular el gradiente de E_k forzosamente ocupo E_{k+1} así que siempre se debe comenzar calculando E_N y de ahí hacia atrás en el tiempo.

2.3.5. Celda GRU

Desde la creación de las celdas LSTM y debido a su complejidad han surgido diversas variantes que tratan de resolver este problema, una de ellas y que además será usada en el presente trabajo es la celda GRU por sus siglas en ingles de *Gated Recurrent Unit*, presentada por primera vez por Cho, et al. [22] cuya representación se muestra en la figura 2.15.

La celda GRU combina las compuertas de olvido y de entrada en una sola celda de actualización además de eliminar el concepto de estado de la celda y combinarlo con su salida o estado oculto, de esta manera las ecuaciones que caracterizan este tipo de estructuras están dadas por:

$$\begin{aligned} z_t &= \sigma(W_z \left[h_{t-1}, x_t\right]) \\ r_t &= \sigma(W_r \left[h_{t-1}, x_t\right]) \\ \widetilde{h}_t &= \tanh(W \left[r_t * h_{t-1}, x_t\right]) \\ h_t &= (1 - z_t) * h_{t-1} + z_t * \widetilde{h}_t \end{aligned}$$

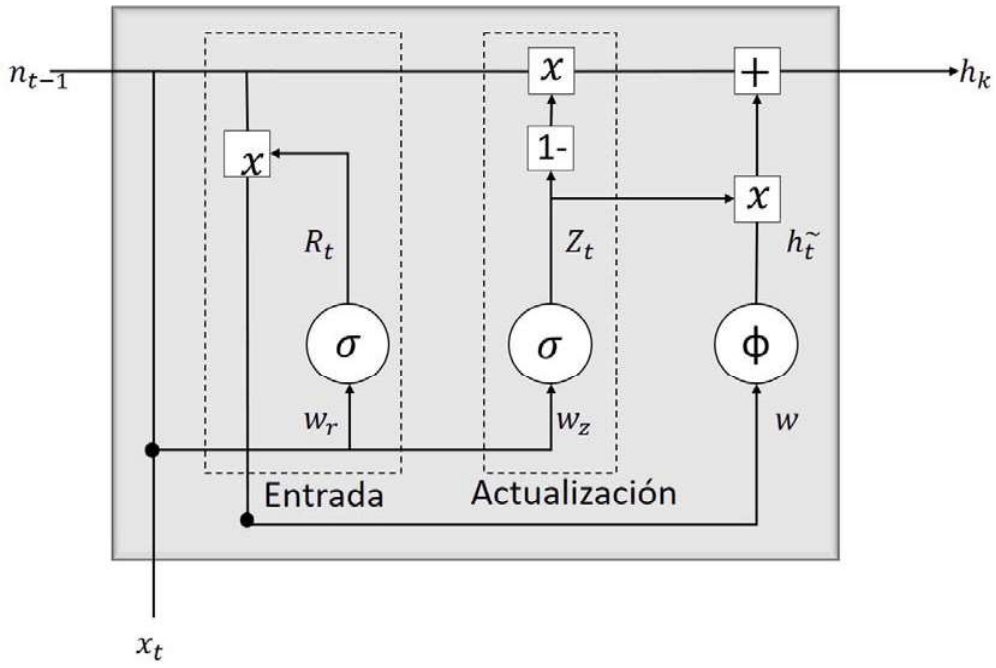


Figura 2.15: Celda GRU

Para esta las redes GRU el entrenamiento es exactamente igual al que se reviso para las LSTM, es decir que se ocupa BPTT.

Capítulo 3

Modelado de sistemas dinámicos con RNN LSTM-GRU y NN profunda

3.1. Red LSTM-NN profunda

Lo primero que se necesita para proponer y entender la nueva arquitectura es definir la celda recurrente basada en las celdas LSTM o GRU que será usada en este capítulo.

A continuación, se definen los dos tipos de celdas que de manera general ofrecen resultados muy similares, por esta razón se pueden usar indistintamente en cualquiera de las arquitecturas planteadas en el presente trabajo. La celda GRU será usada en la realización de los cálculos ya que tiene todas las ventajas de la LSTM además de una arquitectura interna más simple. Las celdas LSTM y GRU se basan en un concepto clave denominado estado de la celda que se podría entender como un carril que transfiere la información relevante a la secuencia, también se le conoce como la memoria de la celda, la información es agregada o removida de la memoria con las compuertas, esta celda se compone de tres tipos de compuertas para proteger y controlar el estado de la celda, la compuerta de olvido, la compuerta de entrada y la compuerta de salida.

1. La compuerta de olvido está dada por:

$$F(k) = \sigma\left(W_F(k)\left[y(k-1), u(k)\right]\right) \tag{3.1}$$

Donde $u(k)$ es la entrada, $y(k-1)$ es la salida, σ es una función sigmoide, si la función $\sigma = 1$ representa "guarda esto" y si $\sigma = 0$ representa "deshazte de esto",$W_F(k)$ es el peso de la compuerta.

2. La compuerta de entrada $I(k)$ la cual decide que cosas agregar al estado de la celda está dada por

$$\begin{aligned} I(k) &= \sigma\left(W_I(k)\left[y(k-1), u(k)\right]\right) \\ \widetilde{x}(k) &= \tanh\left(W_X(k)\left[y(k-1), u(k)\right]\right) \end{aligned} \tag{3.2}$$

Donde $\widetilde{x}(k)$ e $I(k)$ son estados internos. σ decide que valores actualizar y $tanh$ crea un vector de nuevos candidatos a agregar al estado.

Para calcular el estado de la celda se agrega la salida de la compuerta de olvido y la compuerta de entrada

$$x(k) = F(k)\, x(k-1) + I(k)\, \widetilde{x}(k) \tag{3.3}$$

3. La compuerta de salida decide cual será finalmente la salida de la celda

$$\begin{aligned} O(k) &= \sigma\left(W_O(k)\left[y(k-1), u(k)\right]\right) \\ y(k) &= O(k)\tanh\left(x(k)\right) \end{aligned} \tag{3.4}$$

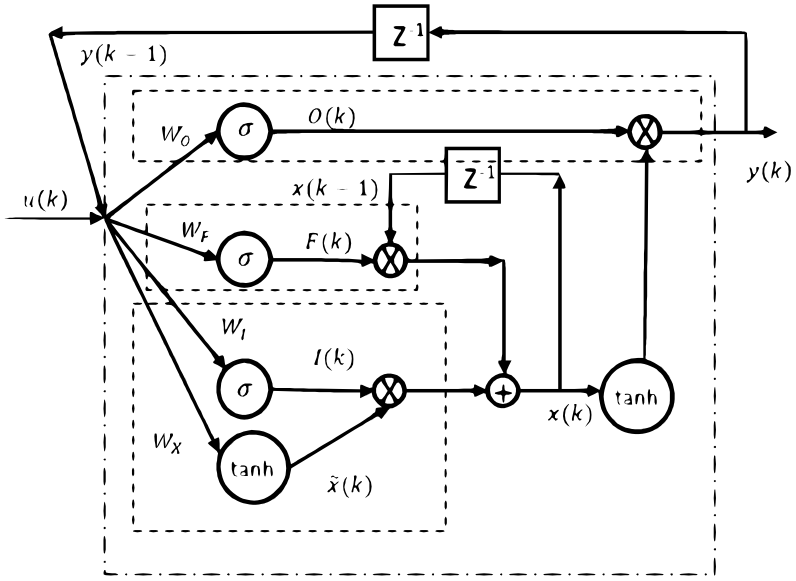


Figura 3.1: Estructura recurrente basada en la celda LSTM

El modelo de la celda de la figura 3.1 es bastante complejo, pero existe una variante de esta celda que consiste en algo llamado conexiones de mirilla o "*peephole connections*"[11]

en inglés, que consiste en crear conexiones en la celda LSTM que permiten a las compuertas depender no solo de su estado previo si no también de su estado actual, de esta forma las ecuaciones de las compuertas están determinadas por

$$F(k) = \sigma\left(W_F(k)\left[x(k-1), y(k-1), u(k)\right]\right) \tag{3.5}$$
$$I(k) = \sigma\left(W_I(k)\left[x(k-1), y(k-1), u(k)\right]\right) \tag{3.6}$$
$$O(k) = \sigma\left(W_O(k)\left[x(k-1), y(k-1), u(k)\right]\right) \tag{3.7}$$

$\widetilde{x}(k)$ permanece sin ningún cambio, y el estado de la celda se compone de la siguiente forma

$$x(k) = F(K)x(k-1) + (1 - F(k))\widetilde{x}(k) \tag{3.8}$$

El objetivo de esta arquitectura es modelar un sistema no lineal, esto actualizando los pesos W_F, W_I, W_X y W_O, tal que la salida de la red converja hacia la salida real del sistema, esto es

$$\underset{[W_F, W_I, W_X, W_O]}{\arg\min} \left[\widehat{y}(k) - y(k)\right]^2 \tag{3.9}$$

Pues bien, la arquitectura definida anteriormente puede ser empleada perfectamente para el modelado de sistemas no lineales, sabiendo que sí

$$U(k) = f(y(k-1), \ldots, y(k-n_y), u(k), \ldots, u(k-n_u)) \tag{3.10}$$

entonces se trata de un modelo de predicción, y si

$$U(k) = f(u(k), \ldots, u(k-n_u)) \tag{3.11}$$

entonces, se trata de un modelo de simulación.

Sin embargo, como de muestra en [12] y [13] los modelos recurrentes normales de LSTM y GRU no pueden identificar adecuadamente un sistema usando el modelo de simulación, además de en ocasiones cortar los picos de y como se muestra en la figura 3.2

Para resolver esta situación, en este capítulo se combina una red LSTM clásica con una MLP, esta arquitectura a la que se denominará LSTM-NN para el modelado de sistemas dinámicos se muestra en la figura 3.3, donde se tienen p capas ocultas, y cada una de las

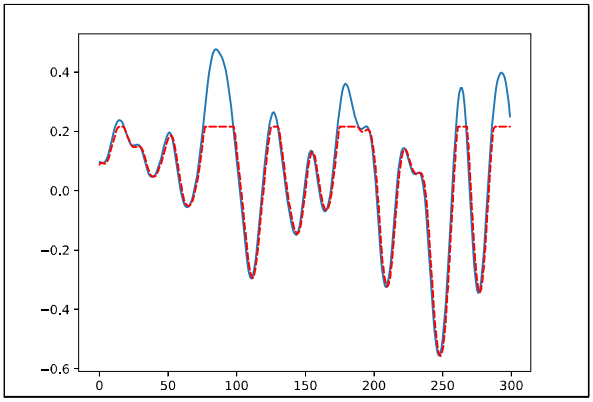


Figura 3.2: Sistema no lineal modelado con RNN LSTM tradicional

capas tiene q estructuras recurrentes LSTM definidas anteriormente y la última capa es un perceptrón multicapa (MLP).

Se diseñó una arquitectura bastante genérica que si bien puede tener las capas y nodos que sean necesarios, en la práctica se demostró que la MLP puede ser sustituida simplemente por un perceptrón, además en el caso del modelo de predicción un bloque LSTM es mas que suficiente para obtener un buen resultado y para el caso de simulación basta con agregar nodos en esa misma capa por lo tanto una capa oculta fue mas que suficiente para realizar el modelado de sistemas no lineales con los que se puso a prueba esta arquitectura lo que resulta muy eficaz debido al tiempo que tarda el entrenamiento de cual se hablará en la siguiente sección.

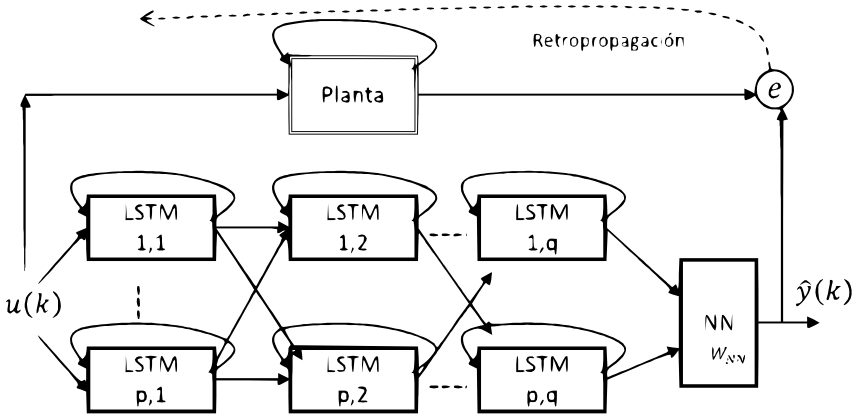


Figura 3.3: Arquitectura LSTM-NN para el modelado de sistemas dinámicos

3.2. Entrenamiento de la red LSTM-NN profunda

Siempre que se habla de redes neuronales se habla de bloques o nodos conectados entre sí a los que se les denomina neurona, la estructura de estas neuronas esta formada por una sola función de activación, comúnmente *tanh*. Las redes LSTM lógicamente muestran esta misma estructura y la arquitectura LSTM-NN no es la excepción, aunque cada uno de los bloques es en sí algo más complejo. De tal forma que el entrenamiento de esta red tambien se realiza usando el concepto de retropropagacion, sin embargo, ya que se toma en cuenta no solo el estado presente sino los estados previos, entonces el entrenamiento se usará la retropropagacion a través del tiempo, comúnmente conocida como (BPTT).

Se puede analizar primero el caso más sencillo sin pérdida de generalidad, donde k=1 y suponiendo en lugar del bloque LSTM una celda recurrente simple.

$$x\left(k\right)=\phi\left[W\left(k\right)x\left(k-1\right)\right] \tag{3.12}$$

Si se toman m pasos a través del tiempo se tiene que

$$x\left(k-m\right)=\phi\left[W\left(k\right)x\left(k-m-1\right)\right] \tag{3.13}$$

Si $y\left(k\right)$ es la salida deseada, $\hat{y}\left(k\right)=x(k)$, y el error estará dado por

$$J\left(k\right)=\frac{1}{2}\left[\hat{y}\left(k\right)-y\left(k\right)\right]^{2}=e_{o}^{2}\left(k\right) \tag{3.14}$$

Usando la técnica del gradiente, se tiene que

$$W\left(k+1\right)=W\left(k\right)-\eta\frac{\partial J\left(k\right)}{\partial W\left(k\right)} \tag{3.15}$$

donde

$$\begin{array}{c}\frac{\partial J(k)}{\partial W}=\frac{\partial J(k)}{\partial e_{o}(k)}\frac{\partial e_{o}(k)}{\partial y(k)}\frac{\partial y(k)}{\partial x(k)}\frac{\partial x(k)}{\partial p(k)}\frac{\partial p(k)}{\partial W}\\=e_{o}\left(k\right)\phi'x\left(k-1\right)\end{array} \tag{3.16}$$

Si se estiende este resultado a m pasos

$$\begin{array}{c}W\left(k+1\right)=W\left(k\right)-\eta\sum_{i=1}^{m}e_{i}\left(k\right)x\left(k-i\right)\\e_{i}\left(k\right)=e_{i-1}\left(k\right)W\left(k\right)\phi'=e_{o}\left(k\right)\Pi_{i=1}^{m}\left[W\left(k\right)\phi'\right]\end{array} \tag{3.17}$$

Cabe recordar que se está usando una neurona recurrente tradicional, así que se puede

observar que si $\|W(k)\phi'\| > 1$, el gradiente explota. Si $\|W(k)\phi'\| < 1$, el gradiente se desvanece. Las LSTM evitan estos problemas gracias a sus compuertas, de manera que solo cuando $\|W(k)\phi'\| \approx 1$, se activa la celda. Entonces de acuerdo a lo anterior, los pesos de la celda LSTM se actualizarán bajo las siguientes reglas

$$\begin{aligned} W_O(k+1) &= W_O(k) - \eta \sum_{i=1}^{M} eO(k) [\hat{y}(k-i), u(k-i)] \\ W_F(k+1) &= W_F(k) - \eta \sum_{i=1}^{M} e_F(k) [\hat{y}(k-i), u(k-i)] \\ W_I(k+1) &= W_I(k) - \eta \sum_{i=1}^{M} e_I(k) [\hat{y}(k-1), u(k)] \\ W_4(k+1) &= W_X(k) - \eta \sum_{i=1}^{M} e_X(k) [\hat{y}(k-1), u(k)] \end{aligned} \tag{3.18}$$

donde

$$\begin{aligned} e_O(k) &= e_{O-1}(k) W_O(k) \sigma' \phi(x(k)), \\ e_F(k) &= e_{F-1}(k) W_F(k) \sigma' x(k-1) \\ e_I(k) &= e_{I-1}(k) W_I(k) \sigma' \phi(W_X(k) [\hat{y}(k-1), u(k)]) \\ e_X(k) &= e_{X-1}(k) W_X(k) \phi' \sigma(W_I(k) [\hat{y}(k-1), u(k)]) \end{aligned} \tag{3.19}$$

Esta regla de aprendizaje aplica solo para las celdas LSTM, sin embargo, la arquitectura aquí propuesta es una arquitectura jerárquica cuyo objetivo es entrenar los pesos de tal forma que el error entre la salida del modelo neural y la salida de la planta sea minimizado.

$$J = \frac{1}{2} e_0^2 \quad e_o = \widehat{y} - y \tag{3.20}$$

Entonces, para la última capa que es un perceptrón se tiene la siguiente regla de aprendizaje.

$$W_{NN}(k+1) = W_{NN}(k) - \eta \widehat{y}_{i,q}(k) e_o(k) \tag{3.21}$$

donde $\eta > 0$ es la taza de aprendizaje y $\widehat{y}_{i,q}$, $i = 1 \cdots p$, son las salidas de las LSTMs.

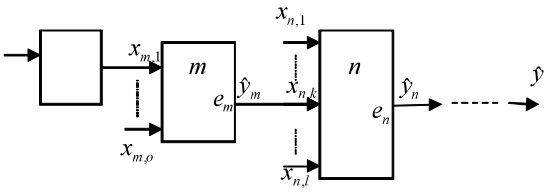


Figura 3.4: Retropropagación del error

Los errores son retropropagados del bloque n al bloque m como se muestra en la figura 3.4. Si conocemos e_n, entonces se puede obtener e_m del modelo jerárquico. Para el bloque

n, se usará la regla de aprendizaje del gradiente descendente

$$w_n\left(k+1\right) = w_n\left(k\right) - \eta x_{n,k} e_n \tag{3.22}$$

Donde $x_{n,k}$ es la entrada al bloque n, la cual es la salida del bloque m, y_m.

Para el bloque m, se usará la regla de aprendizaje del gradiente descendente

$$w_m\left(k+1\right) = w_m\left(k\right) - \eta x_{m,k} e_m \tag{3.23}$$

Ahora para transferir e_n del bloque n al error e_m del bloque m, se tiene

$$e_m = \left[W_O\sigma' + \left(W_I\sigma' + W_X \tanh' + W_F\sigma'\right)\tanh'\right] e_n \tag{3.24}$$

Como la estructura interior de cada bloque LSTM es la misma, se puede usar la misma técnica para el entrenamiento de cada bloque siempre y cuando se conozca la salida del error de cada uno de ellos.

Entonces el algoritmo de entrenamiento es el siguiente:

1. Calcular la salida de cada bloque LSTM. Algunas salidas del modelo deberán ser las entradas del siguiente nivel.

2. Calcular el error de cada bloque comenzando desde el último bloque y después retropropagando los errores de identificación

3. Entrenar los pesos de cada bloque independientemente.

3.3. Simulación y resultados

La arquitectura propuesta LSTM-NN se puso a prueba con tres sistemas no lineales diferentes tanto el modelo de predicción como el modelo de simulación, cabe aclarar que en esta etapa se pensó importante realizar pruebas exhaustivas de comparación contra una red tradicional MLP para saber que aspectos son superiores y en que aspectos se debe mejorar y de acuerdo con esto desarrollar nuevas arquitecturas capaces de superar estas limitantes.

Para las pruebas y resultados se usará el error cuadrático medio o MSE por sus siglas en inglés como punto de comparación, tanto el error resultante en el conjunto entrenamiento MSE_E, como el error resultante al probar la red con los datos de prueba MSE_p, el cual debería ser el más importante.

Durante las simulaciones las variables que se tienen que ajustar para obtener un buen resultado son:

1. Las diferentes combinaciones de $p \in \mathbb{N}$ y $q \in \mathbb{N}$ que conforman la arquitectura de $N(\cdot)$, es decir el número de capas y nodos que conforman la red neuronal.

2. El tamaño de $n_y \in \mathbb{N}$ y $n_u \in \mathbb{N}$, en este caso se consideran $n_y = n_u + 1 = n \in \mathbb{N}$ para reducir el número de combinaciones y simplificar las pruebas.

3. La función de activación de cada neurona que en teoría, y característicamente en el aprendizaje profundo, las funciones podrían ser diferentes en cada nodo, pero eso alargaría más el periodo de pruebas y se deja para etapas posteriores y ajustes finos de la red. Por ahora, se considera a f como la no linealidad de cada uno de los nodos de la red y puede ser de tres tipos, simoide $\sigma(x) = \frac{1}{1+e^{-x}}$, $tanh(x) = \frac{2}{1+e^{-2x}} - 1$, $ReLU(x) = max(0, x)$.

4. Existen variaciones del SGD que han demostrado resolver el entrenamiento mucho más rápido y mejor, en las pruebas se tomarán en cuenta los siguientes algoritmos de optimización: SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax y Nadam.

5. El número de épocas o iteraciones sobre los mismos datos, que determina en gran medida el éxito del entrenamiento.

6. Finalmente aparecen dos variables que se obtienen como resultado de las distintas combinaciones de las variables anteriores, n_θ la cantidad de parámetros ajustables de la red y el tiempo que toma entrenar la red, llamado T_e.

Dado que son muchas variables con las cuales jugar para obtener un buen resultado, la estrategia a seguir será realizar los cambios por etapas, y seleccionar el mejor resultado de cada etapa para utilizarlo en la siguiente. Cabe aclarar que en todos los casos se usa en el algoritmo un batch de 50.

3.3.1. Wiener-Hammerstein

En este ejemplo se usa el conjunto de datos SYSID 2009 Wiener-Hammerstein [14] con 188 000 pares $(u(k), y(k))$, que se dividirán en dos partes, 20 % para entrenamiento y 80 % para prueba. Algo notable en esta división es que solo el 20 % de los datos son suficientes para obtener buenos resultados, el destinar más porcentaje de los datos al entrenamiento aumenta el tiempo de entrenamiento mucho más de lo que reduce el margen de error

Modelo de predicción MLP

Lo primero es encontrar el modelo de predicción, ya que es más sencillo y además sirve de base para el modelo de simulación.

Primero habrá que seleccionar qué función de activación f usarán las neuronas y que algoritmo de optimización es el más conveniente, entonces se proponen $p = 1$,$q = 1$ y $n = 1$ para estas pruebas.

De los resultados obtenidos en las tablas 3.1,3.2 y 3.3 se puede concluir que el algoritmo más ineficiente como era de esperarse es el SGD clásico, esto a pesar que en los tres casos consiguió de hecho el mejor resultado, pero no fue si no después de muchas épocas, demasiadas comparado con los demás algoritmos de optimización.

Como ejemplo de la ineficiencia del SGD, en la tabla 3.1 se llegó a un error de $4x10^{-4}$ después de 500 épocas y tardo mas de 5 minutos a diferencia del RMSprop que alcanzó el mismo error en tan solo 10 épocas y tardo menos de 7 segundos.

$N(\cdot)$	n	f	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
p=1 q=1	1	σ	sgd	5	3	3.30	123	123
p=1 q=1	1	σ	sgd	100	3	59.40	20.15	22.22
p=1 q=1	1	σ	sgd	500	3	314	4.29	4.61
p=1 q=1	1	σ	RMSprop	5	3	3.36	15.91	17.56
p=1 q=1	1	σ	RMSprop	10	3	6.39	4.29	4.60
p=1 q=1	1	σ	Adagrad	5	3	3.36	79	72
p=1 q=1	1	σ	Adagrad	100	3	60	11.10	12.23
p=1 q=1	1	σ	Adadelta	5	3	3.97	68	75
p=1 q=1	1	σ	Adadelta	30	3	19.03	4.25	4.56
p=1 q=1	1	σ	Adam	5	3	3.53	22	24
p=1 q=1	1	σ	Adam	15	3	9.94	4.23	4.55
p=1 q=1	1	σ	Adamax	5	3	3.38	27	30
p=1 q=1	1	σ	Adamax	15	3	9.49	4.24	4.56
p=1 q=1	1	σ	Nadam	5	3	3.8	5.91	6.45
p=1 q=1	1	σ	Nadam	8	3	5.73	4.25	4.58

Cuadro 3.1: Resultados de entrenamiento para el modelo de predicción con 1 capa de 1 neurona con la función sigmoide y diferentes algoritmos de optimización

$N(\cdot)$	n	f	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
p=1 q=1	1	*tanh*	sgd	5	3	3.49	38	42
p=1 q=1	1	*tanh*	sgd	100	3	61.47	7.60	8.16
p=1 q=1	1	*tanh*	RMSprop	5	3	3.31	7.74	8.40
p=1 q=1	1	*tanh*	RMSprop	10	3	6.36	7.60	8.17
p=1 q=1	1	*tanh*	Adagrad	5	3	3.36	17	18
p=1 q=1	1	*tanh*	Adadelta	5	3	3.49	8.32	9.19
p=1 q=1	1	*tanh*	Adadelta	10	3	6.68	7.60	8.17
p=1 q=1	1	*tanh*	Adam	5	3	3.60	7.63	8.25
p=1 q=1	1	*tanh*	Adam	10	3	6.53	7.60	8.19
p=1 q=1	1	*tanh*	Adamax	5	3	3.46	7.61	8.22
p=1 q=1	1	*tanh*	Adamax	10	3	6.61	7.60	8.19
p=1 q=1	1	*tanh*	Nadam	5	3	3.57	7.70	8.35
p=1 q=1	1	*tanh*	Nadam	10	3	6.87	7.62	8.17

Cuadro 3.2: Resultados de entrenamiento para el modelo de predicción con 1 capa de 1 neurona con la función tanh y diferentes algoritmos de optimización

$N(\cdot)$	n	f	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
p=1 q=1	1	*ReLU*	sgd	5	3	3.35	11	12
p=1 q=1	1	*ReLU*	sgd	40	3	61.47	3.14	3.38
p=1 q=1	1	*ReLU*	RMSprop	2	3	1.51	3.15	3.39
p=1 q=1	1	*ReLU*	RMSprop	5	3	3.54	3.36	3.60
p=1 q=1	1	*ReLU*	Adagrad	5	3	3.36	3.43	3.73
p=1 q=1	1	*ReLU*	Adagrad	15	3	9.28	3.14	3.39
p=1 q=1	1	*ReLU*	Adadelta	5	3	3.45	3.18	3.43
p=1 q=1	1	*ReLU*	Adadelta	10	3	6.34	3.15	3.39
p=1 q=1	1	*ReLU*	Adam	5	3	3.52	3.18	3.42
p=1 q=1	1	*ReLU*	Adam	10	3	6.63	3.16	3.41
p=1 q=1	1	*ReLU*	Adamax	5	3	3.46	3.17	3.42
p=1 q=1	1	*ReLU*	Nadam	2	3	1.66	3.15	3.39
p=1 q=1	1	*ReLU*	Nadam	5	3	3.63	3.31	3.55

Cuadro 3.3: Resultados de entrenamiento para el modelo de predicción con 1 capa de 1 neurona con la función ReLU y diferentes algoritmos de optimización

Entre los mejores optimizadores están el Nadam, RMSprop, Adamax y Adam ordenados de acuerdo a sus resultados, de esta forma el Nadam, RMSprop se colocan en este caso casi a la par como los mejores algoritmos de optimización ya que para todas las funciones de activación llegaron al mínimo error siempre en menos épocas que los demás, sin embargo vale la pena considerar los otros dos algoritmos para afinar el resultado de $N(\cdot)$, incluso el SGD que siempre obtiene un error menor comparado con los otros algoritmos.

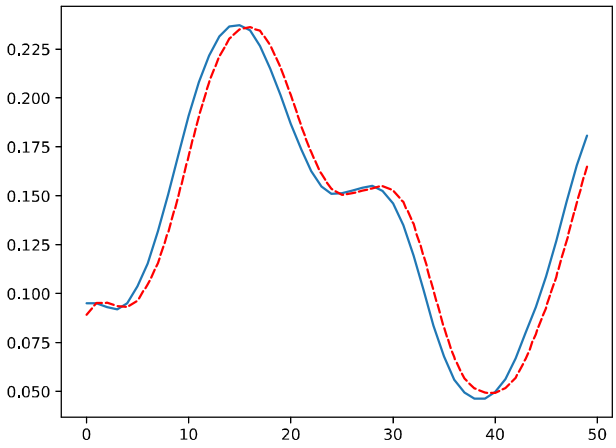


Figura 3.5: Mejor resultado del entrenamiento para MLP con 1 sola neurona usando *ReLU* como función de activación

Respecto a las funciones de activación, se aprecia claramente que la función sigmoide llega a un buen resultado con un error aproximado de $4x10^{-4}$ pero se requieren más épocas que en los otros dos casos como con *tanh* que resultó ser la menos adecuada ya que aunque acelera el proceso de entrenamiento también llega a un resultado menos óptimo a diferencia de la función *ReLU* que como se esperaba dada su naturaleza agiliza el proceso de entrenamiento drásticamente, y además demostró conseguir muy rápidamente el mejor resultado con un error de $3x10^{-4}$ para todos los algoritmos de optimización. Esto da una idea clara de en qué casos o hacia donde se debe enfocar el uso de las distintas funciones de activación.

En la figura 3.5 se muestra el resultado con el conjunto de prueba para $N(\cdot)$ con 1 sola neurona usando *ReLU* como función de activación y Nadam como optimización, después de tan solo dos épocas con un error de $3{,}39x10^{-4}$. Aunque el resultado es ya

$N(\cdot)$	n	f	Optimización	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
p=1 q=2	1	$ReLU$	Nadam	5	5	6.91	3.28	3.52
p=1 q=3	1	$ReLU$	Nadam	5	7	4.2	∞	∞
p=2 q=2	1	$ReLU$	Nadam	2	3	3.98	3.23	3.48
p=2 q=3	1	$ReLU$	Nadam	5	15	4.36	∞	∞
p=5 q=2	1	$ReLU$	Nadam	10	21	14.5	3.14	3.39
p=50 q=2	1	$ReLU$	Nadam	10	201	14.45	3.18	3.42
p=50 q=50	1	$ReLU$	Nadam	10	79311	34.86	24	25
p=1 q=1	2	$ReLU$	Nadam	10	5	6.5	2.76	2.97
p=1 q=1	2	$ReLU$	Nadam	30	5	18.44	0.185	0.201
p=1 q=1	2	$ReLU$	Nadam	100	5	61.71	0.088	0.098
p=1 q=1	3	$ReLU$	Nadam	10	7	6.51	1.28	1.39
p=1 q=1	3	$ReLU$	Nadam	30	7	18.70	0.387	0.412
p=1 q=1	5	$ReLU$	Nadam	10	11	6.34	0.432	0.476
p=1 q=1	5	$ReLU$	Nadam	30	11	18.69	0.318	0.350

Cuadro 3.4: Resultados de entrenamiento para el modelo de predicción con diferentes arquitecturas

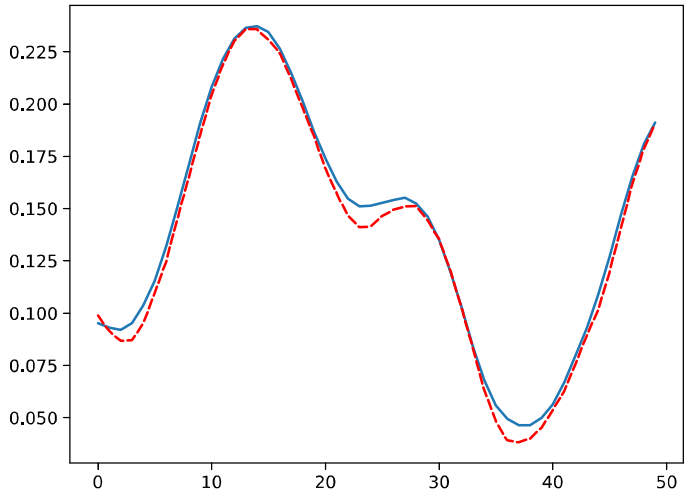


Figura 3.6: Predicción de $N(\cdot)$ para el conjunto de prueba (linea punteada) contra el resultado esperado usando una sola neurona con ReLU algoritmo Nadam de optimización, $n = 2$ y 100 épocas

bastante bueno ahora es momento de trabajar con la estructura de la red para mejorar el resultado. Cabe aclarar que en cada caso $N(\cdot)$ cuenta con q capas y p nodos por capa excepto la capa de salida, cuyo número de nodos esta determinado por la naturaleza del problema.

Los resultados obtenidos en la tabla 3.4 en los que se combinan diferentes valores de p, q y n, es decir se experimenta con la estructura de $N(\cdot)$ son en verdad interesantes ya que el mejor resultado obtenido en las pruebas es con una sola neurona y con $n = 2$, así se mejora muchísimo el error que baja de $4x10^{-4}$ a $9x10^{-6}$, también se puede observar lo sensible que es el modelo a variaciones en la estructura de la red ya que si se cambia n de 1 a 3, el error cambia en el orden de 100, además el agregar capas y nodos de mas a la estructura puede llevar a un error muy grande representado como ∞.

Modelo de predicción LSTM-NN

Si bien las redes neuronales MLP muestran muy buenos resultados para el modelado de sistemas no lineales aún tienen problemas que deben ser resueltos, el más conocido es el llamado "*vanishing gradients*"para el cual se han desarrollado ya múltiples técnicas para superarlo, pero existen otros como el sobre entrenamiento, la correcta definición de sus hiper parámetros como son el número de capas ocultas, los nodos por capa, el tipo de conexión entre capas etc. El crecimiento de las redes neuronales ya sea en el número de capas o nodos por capa hace difícil la exploración de diversas arquitecturas y parámetros para la obtención de un mejor resultado, esto debido al tiempo que toma el entrenamiento y más aún cuando se usan modelos de redes neuronales profundas las cuales por lo general consiguen mejores resultados, sin embargo, son más lentas y difíciles de interpretar, sin mencionar la naturaleza estática de las relaciones representadas por las redes neuronales.

Para el modelado con LSTM se usará el mismo esquema de capas q y nodos por capa p usado anteriormente con la MLP con la diferencia que cuando se hable de cualquiera de los nodos de la red, este hará referencia a una celda LSTM, es decir, que se estará usando una red neuronal recurrente LSTM.

Con esta arquitectura se espera un mejor resultado que con las MLP ya que a diferencia de estas, las arquitecturas recurrentes han demostrado gran eficiencia para representar relaciones dinámicas así como dependencias a largo plazo. Ya que la construcción de las celdas LSTM es diferente que la de un perceptrón se van a realizar nuevamente las pruebas de función de activación y algoritmos de optimización para saber cuáles son los mejores para este tipo de arquitecturas.

Como se observa en la tabla 3.5, la función *ReLU* parece no funcionar para este

tipo de arquitecturas a diferencia de la función sigmoide que en todos los casos arrojó resultados factibles y como era de esperarse los algoritmos RMSprop,Nadam,Adamax y Adam probaron ser los mejores en ese mismo orden de aparición, aunque a diferencia de lo sucedido con las MLP el RMSprop es ligeramente superior. Lo siguiente será probar con la función *tanh* la cual se esperara mejore los resultados anteriores.

$N(\cdot)$	n	f	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
p=1 q=1	1	$ReLU$	sgd	5	16	∞	∞	∞
p=1 q=1	2	$ReLU$	sgd	5	24	∞	∞	∞
p=1 q=1	1	σ	sgd	5	16	10.36	179	193
p=1 q=1	1	σ	RMSprop	5	16	9.79	40.99	45.02
p=1 q=1	1	σ	Adagrad	5	16	10.03	134	146
p=1 q=1	1	σ	Adadelta	5	16	10.13	90.4	98.7
p=1 q=1	1	σ	Adam	5	16	10.05	88	96
p=1 q=1	1	σ	Adamax	5	16	10.14	52	57
p=1 q=1	1	σ	Nadam	5	16	10.06	55	60

Cuadro 3.5: Resultados de entrenamiento para el modelo de predicción con 1 capa de 1 neurona con la función sigmoide y diferentes algoritmos de optimización

$N(\cdot)$	n	f	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
p=1 q=1	1	$tanh$	sgd	5	16	9.80	112	122
p=1 q=1	1	$tanh$	RMSprop	5	16	10.04	16.68	18.71
p=1 q=1	1	$tanh$	Adagrad	5	16	9.87	308	319
p=1 q=1	1	$tanh$	Adadelta	5	16	10.06	39.86	44.13
p=1 q=1	1	$tanh$	Adam	5	16	10.06	68.09	74.40
p=1 q=1	1	$tanh$	Adamax	5	16	10.13	28.84	32.04
p=1 q=1	1	$tanh$	Nadam	5	16	10.08	40.76	44.95

Cuadro 3.6: Resultados de entrenamiento para el modelo de predicción con 1 capa de 1 neurona con la función sigmoide y diferentes algoritmos de optimización

Como se esperaba la función *tanh* mejora los resultados obtenidos, sin embargo, los algoritmos de optimización cambiaron ya que ahora el RMSprop, Adamax, Adadelta

$N(\cdot)$	n	f	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
p=1 q=1	1	*tanh*	RMSprop	10	16	18.78	11.23	12.52
p=1 q=1	2	*tanh*	RMSprop	10	24	18.63	12.78	14.12
p=1 q=2	1	*tanh*	RMSprop	10	28	30.28	9.68	10.77
p=1 q=3	1	*tanh*	RMSprop	10	40	41.26	9.88	10.97
p=2 q=3	1	*tanh*	RMSprop	10	96	41.95	9.58	10.66

Cuadro 3.7: Resultados de entrenamiento para el modelo de predicción con 1 capa de 1 neurona con la función sigmoide y diferentes algoritmos de optimización

y Nadam resultaron ser los mejores en ese orden de aparición, y el RMSprop sigue ligeramente superior.

Una vez analizadas las funciones no lineales y los algoritmos de optimización para una arquitectura de un solo nodo, es turno de mejorar los resultados cambiando la arquitectura de la red para obtener el modelo de predicción. La arquitectura de la red es uno de los aspectos que se buscan mejorar ya que la selección de los muchos parámetros existentes en las MLP complica el tiempo de diseño y según el tamaño de la red el tiempo de ejecución e incluso la implementación.

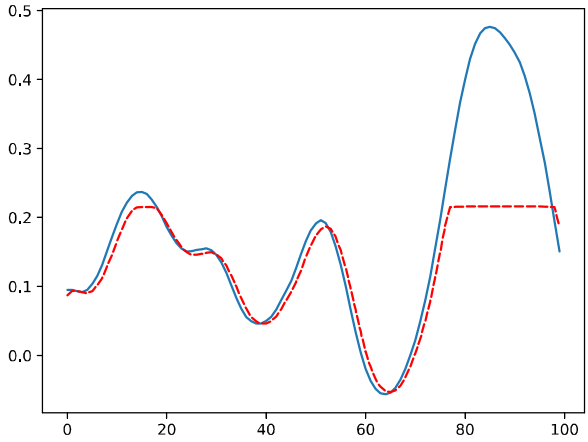


Figura 3.7: LSTM-NN modelo de predicción, sin ajustar

De acuerdo a las pruebas mostradas en la tabla 3.7 se podría concluir que bajo esas condiciones solo se podrá alcanzar un error de $10x10^{-4}$ mínimo y si se analiza la gráfica,

se observa que esta muestra claramente lo que está sucediendo.

En la Figura 3.7 se observa que el modelado en general es bastante bueno, sin embargo, por alguna razón los picos superiores de la función se cortan, es decir que la función $\hat{Y}$ está limitada por un máximo por la función de activación de salida y como se puede ver en la figura 3.8 con un vistazo más amplio de los datos de prueba contra la predicción del modelo, esto sucede para todo k.

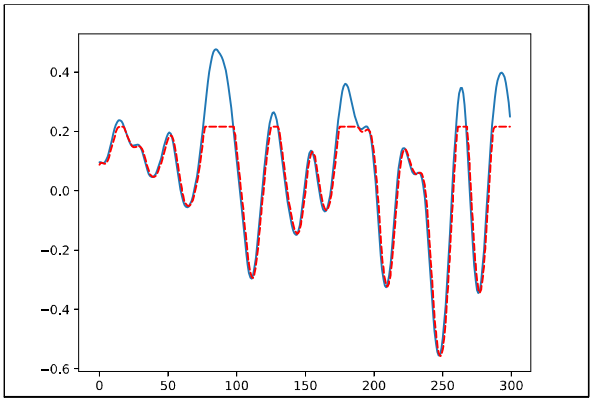


Figura 3.8: LSTM-NN modelo de predicción, sin ajustar, visión amplia

Por lo tanto se tratará, ahora con nuevas configuraciones cambiando únicamente la función de salida, entonces f será la función de activación para todos los nodos o celdas de la red excepto para la celda de salida que tendrá como función de activación f_{sal}.

Una vez encontrada la estructura óptima que como se muestra en la tabla 3.8, se observa que basta con dos celdas LSTM, una en cada capa, la primera con función de activación *tanh* y la segunda con función ReLU y con $n = 4$, cabe recordar que para el modelo de predicción $\hat{y}(k) = f(y(k-1), \ldots, y(k-n_y), u(k), \ldots, u(k-n_u))$, $ny = nu+1 = n \in \mathbb{N}$, además se usa RMSprop que demuestra ser muy eficiente para esta estructura de $N(\cdot)$ encontrada.

Algo que sobresale inmediatamente es que el error alcanzado es de $4{,}5x10^{-6}$ un resultado mucho mejor que el de la MLP aunque muchísimo más tardado en el entrenamiento, ya que tomó casi media hora a diferencia de los 60 segundos de la MLP. Mas adelante se hará un comparativo y se discutirán más a fondo estos resultados, y se verá porque es mejor la arquitectura con LSTM si es que aún no queda claro. En la figura 3.9 se puede ver el resultado de mejor la predicción contra los datos de prueba.

La estructura encontrada para el modelo con la arquitectura LSTM-NN será recurrente de aquí en adelante, así que se puede definir como estructura base $N(\cdot)^*$.

$N(\cdot)$	n	f	f_{sal}	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
p=1 q=2	1	*tanh*	*ReLU*	RMSprop	10	28	29.91	3.23	3.47
p=1 q=2	1	*tanh*	σ	RMSprop	10	28	30.31	15.4	17.3
p=1 q=2	2	*tanh*	*ReLU*	RMSprop	10	36	29.70	2.89	3.08
p=1 q=2	3	*tanh*	*ReLU*	RMSprop	10	44	30.55	1.45	1.55
p=1 q=2	4	*tanh*	*ReLU*	RMSprop	10	52	29.94	0.643	0.703
p=1 q=2	5	*tanh*	*ReLU*	RMSprop	10	60	30.11	0.822	0.905
p=1 q=2	4	*tanh*	*ReLU*	RMSprop	50	52	137	0.388	0.427
p=1 q=2	4	*tanh*	*ReLU*	RMSprop	100	52	271	0.315	0.347
p=1 q=2	4	*tanh*	*ReLU*	RMSprop	200	52	543	0.272	0.294
p=1 q=2	4	*tanh*	*ReLU*	RMSprop	800	52	2167	0.043	0.045

Cuadro 3.8: Resultados de entrenamiento para el modelo de predicción con 1 capa de 1 neurona con la función sigmoide y diferentes algoritmos de optimización

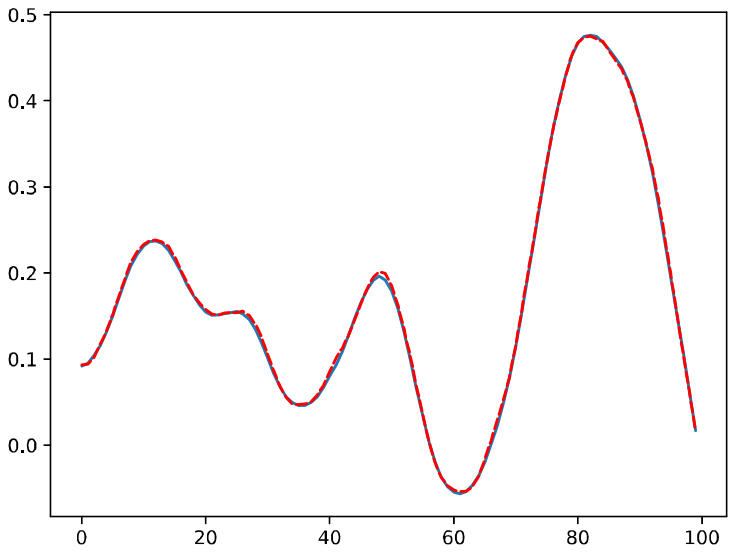


Figura 3.9: Mejor predicción contra los datos de prueba para la LSTM-NN

Modelo de simulación MLP

Una vez obtenido con éxito el modelo de predicción entonces es momento de intentar obtener un modelo simulación que como ya se sabe, será más difícil de entrenar, sin embargo, ya existe un punto de partida con los resultados anteriores, por lo tanto, se empezará probando la estructura que obtuvo el mejor resultado anteriormente y cambiando únicamente los valores de n. Se debe recordar en este apartado que el modelo de simulación es de la forma $\hat{y}(k) = N(u(k), \ldots, u(k-n))$.

$N(\cdot)$	n	f	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
p=1 q=1	1	$ReLU$	Nadam	10	2	∞	171	187
p=1 q=1	2	$ReLU$	Nadam	10	3	∞	171	187
p=1 q=1	5	$ReLU$	Nadam	10	6	∞	171	187
p=1 q=1	30	$ReLU$	Nadam	10	31	6.55	7.72	8.09
p=1 q=1	35	$ReLU$	Nadam	10	36	6.59	6.87	7.24
p=1 q=1	39	$ReLU$	Nadam	10	40	6.56	∞	∞
p=1 q=1	35	$tanh$	Nadam	15	36	10.21	5.23	5.98

Cuadro 3.9: Resultados de entrenamiento para el modelo de predicción con diferentes arquitecturas

En la tabla 3.9 se observa que para valores pequeños de θ, es decir, cuando se utilizan muy pocos valores de la entrada le es imposible a la red realizar un entrenamiento correcto así que se necesitan tomar en cuenta más valores previos de la entrada.

Si bien se podría establecer un modelo de simulación con un solo perceptrón, a diferencia del modelo de predicción este no es lo suficientemente bueno ya que el error es de alrededor de $6x10^{-4}$. Por este motivo, ahora toca variar el número de capas y nodos a ver si existe una mejora. En la tabla 3.10, se observa que el mejor resultado se obtuvo con 6 capas de 25 nodos cada uno más la capa de salida con un nodo y $n = 35$, con un error de $6{,}35x10^{-5}$ que ya es confiable para usar como modelo de simulación. También se puede notar que la eficiencia del algoritmo Nadam es impresionante ya que en tan solo 30 epocas logra un resultado similar el logrado con RMSprop después de 1000 épocas. La figura 3.10 muestra el resultado

$N(\cdot)$	n	f	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
p=2 q=6	35	*tanh*	Nadam	30	105	29.53	4.28	4.64
p=2 q=10	35	*tanh*	Nadam	30	135	29.53	4.89	5.32
p=2 q=30	35	*tanh*	Nadam	30	255	70.07	5.27	5.91
p=5 q=6	35	*tanh*	Nadam	30	336	70.07	1.56	1.72
p=5 q=20	35	*tanh*	Nadam	30	786	54.06	1.33	1.51
p=10 q=6	35	*tanh*	Nadam	30	921	29.42	0.978	1.086
p=20 q=6	35	*tanh*	Nadam	30	2841	30.60	0.786	0.898
p=20 q=6	35	*tanh*	Nadam	100	2841	106.97	0.628	0.711
p=25 q=6	35	*tanh*	Nadam	100	4176	102.41	0.537	0.635
p=30 q=6	35	*tanh*	Nadam	30	5761	31.89	1.2	1.3
p=20 q=6	35	*tanh*	RSMprop	1000	2841	952.3	0.534	0.700

Cuadro 3.10: Resultados de entrenamiento para el modelo de predicción con diferentes arquitecturas

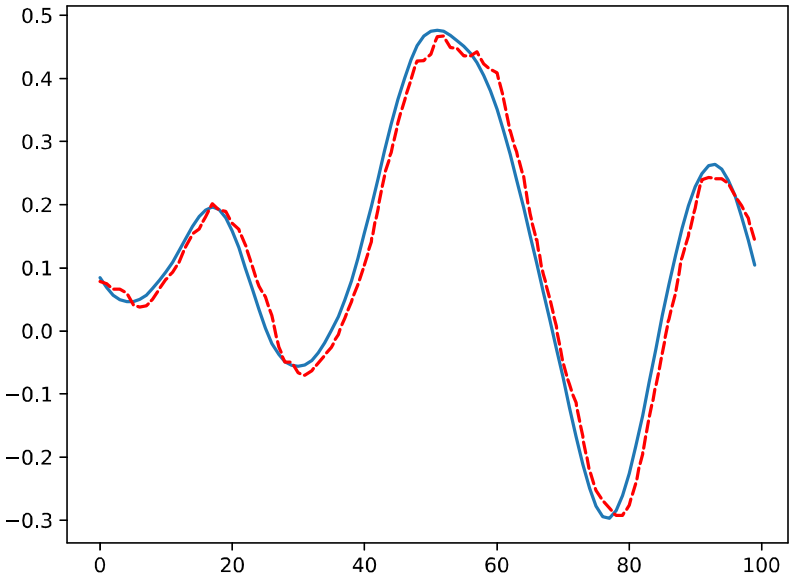


Figura 3.10: Resultado para los datos de prueba del modelo de simulación obtenido con $p = 25$, $q = 6$, $n = 35$, *tanh* y $Nadam$, con un error de $6{,}35x10^{-5}$

Modelo de simulación LSTM-NN

Ahora usando la estructura de la figura $N(\cdot)^*$, se encontrará el modelo de simulación $\hat{y}(k) = N(u(k), \ldots, u(k-n))$ para este sistema. Aquí cuando se usa $N(\cdot)^*$, se hace referencia al número de celdas LSTM en la primera capa, como se muestra en la tabla 3.11.

$N(\cdot)^*$	n	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
1	5	RMSprop	10	40	30.61	170	186
1	10	RMSprop	10	60	31.13	147	161
1	20	RMSprop	10	100	30.65	14.2	15.86
1	30	RMSprop	10	140	30.48	4.70	5.12
1	35	RMSprop	10	160	30.30	6.15	6.21
1	35	RMSprop	50	160	150	2.23	2.44
1	35	RMSprop	500	160	1471	1.74	1.96
2	35	RMSprop	50	320	155	1,78	1.90
3	35	RMSprop	50	488	150	1,74	1.82
4	35	RMSprop	50	664	152	1,99	2.10
3	35	RMSprop	100	488	299	1,53	1.62
3	35	RMSprop	1000	488	2814	0,691	0.761
3	35	RMSprop	2000	488	1468	0,852	0.922
3	35	RMSprop	5000	488	13251	0,685	0.755

Cuadro 3.11: Resultados de entrenamiento para el modelo de predicción con 1 capa de 1 neurona con la función sigmoide y diferentes algoritmos de optimización

Después de una serie de pruebas mostradas en la tabla 3.11 se puede concluir que el error mínimo de 7x10-5 aunque se podría tomar de manera general como un buen resultado, representa un error mayor comparado con el obtenido anteriormente con la MLP para el modelo de simulación de 6x10-5, sin embargo, para obtener con la LSTM un resultado que pueda ser comparado con la MLP se realizó un entrenamiento de 5000 épocas, el cual tuvo una duración de casi cuatro horas, un tiempo enorme sí es que se compara con el par de minutos que tardo la MLP en alcanzarlo, además si se observa la tendencia de crecimiento del tiempo conforme se aumentan las épocas es probable que si se entrena durante 10,000 épocas podría tardar 10 horas o más como se muestra en la figura 3.11 y esto no garantiza obtener un error competitivo ya que si se analiza la figura 3.12, tal parece que el error llegó ya a un margen de oscilación.

En este punto se podría pensar que a diferencia de los nodos de la MLP, el aumentar una celda LSTM más al modelo incrementará drásticamente el tiempo de entrenamiento dada su complejidad por lo tanto la cantidad de celdas deberá mantenerse al mínimo; sin embargo, del análisis de la tabla se puede concluir que esto no es así, pues agregar algunas celdas en la misma capa es irrelevante en cuanto al tiempo de entrenamiento, aunque si genera un gran efecto en el resultado y en el número de parámetros.

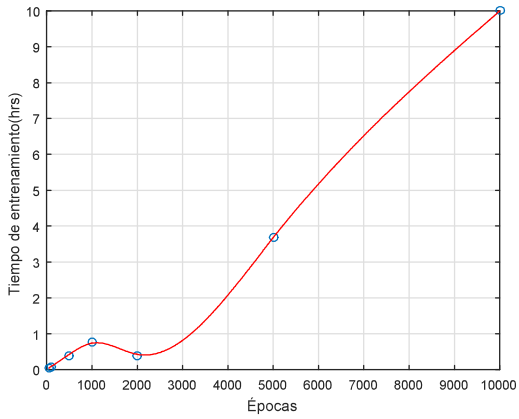


Figura 3.11: Épocas vs tiempo

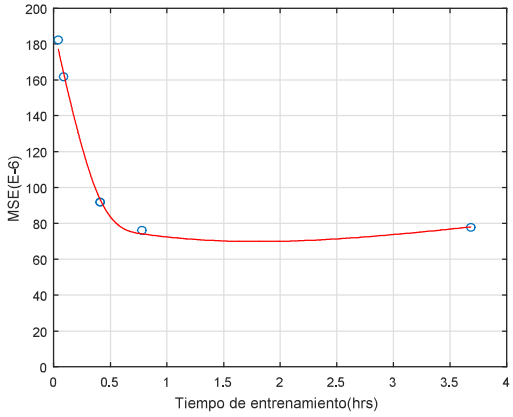


Figura 3.12: MSE vs tiempo

Así que para mejorar este tiempo de entrenamiento se cambia la capa de salida por un perceptrón con función de activación $ReLU$.

$N(\cdot)$	n	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
1	35	Nadam	10	150	19	4.84	5.28
2	35	Nadam	10	307	19	2.02	2.24
3	35	Nadam	10	472	20	1.90	2.05
4	35	Nadam	10	645	20	2.18	2.37
3	35	Nadam	50	472	109	1.68	1.78
3	35	Nadam	100	472	221	0.627	0.699

Cuadro 3.12: Resultados de entrenamiento para el modelo de predicción con 1 capa de 1 neurona con la función sigmoide y diferentes algoritmos de optimización

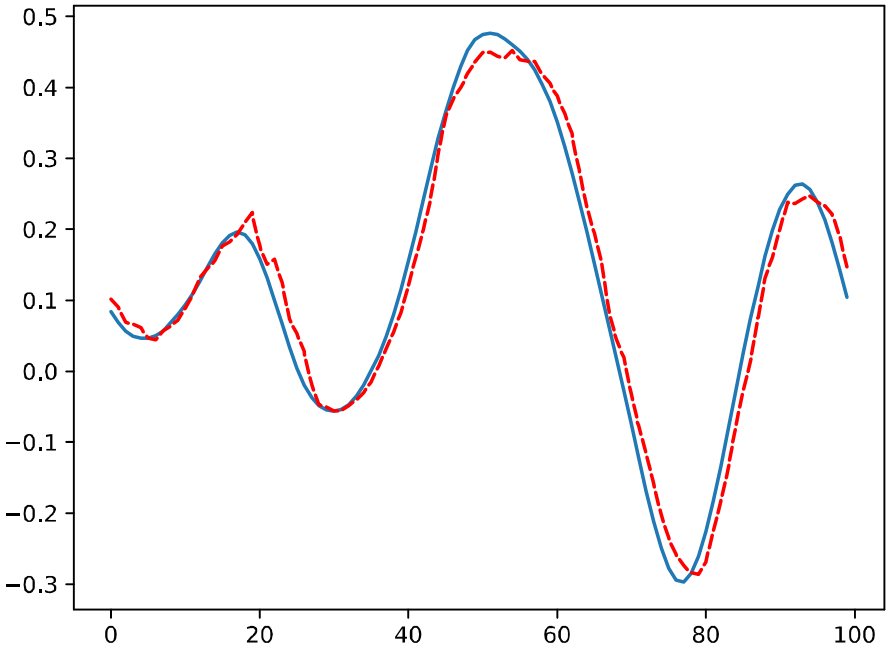


Figura 3.13: Mejor resultado para simulación con la arquitectura LSTM-NN

3.3.2. Horno de gas

El conjunto de datos del horno de gas de Box, Jenkins, y Reinsel, 1994 [15], se usa a menudo para el análisis de series de tiempo. El horno es alimentado con gas metano el cual representa las entradas $u(k)$, y una vez adentro se combina con el aire para formar una mezcla de gases con dióxido de carbono CO2 cuya concentración es la salida $y(k)$.

En este conjunto de datos se tienen 296 observaciones $(x(k), y(k))$ que fueron tomadas con intervalos regulares de 9 segundos.

Modelo de predicción MLP

Tomando como base el conocimiento adquirido anteriormente con las funciones no lineales usadas en los nodos de la red y probados los algoritmos de optimización, podemos partir de ciertas bases para diseñar el modelo de predicción. Dado que son muy pocos los datos que se tienen se ocuparán un 60 % para el subconjunto de entrenamiento y un 40 % de los pares para el subconjunto de prueba.

$N(\cdot)$	n	f	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
p=1 q=1	1	*ReLU*	Nadam	10	3	0.40	301	341
p=1 q=1	1	*ReLU*	Nadam	100	3	0.98	35.69	39.42
p=1 q=1	1	*ReLU*	Nadam	200	3	1.63	17.48	26.59
p=1 q=1	2	*ReLU*	Nadam	100	5	0.98	23.37	44.28
p=1 q=1	3	*ReLU*	Nadam	100	7	0.99	80	121
p=2 q=2	2	*tanh*	Nadam	1000	19	0.98	6.84	24.5
p=2 q=6	2	*tanh*	Nadam	1000	43	8.24	6.01	16.9
p=2 q=10	2	*tanh*	Nadam	1000	73	9.92	10.1	23.4
p=5 q=2	2	*tanh*	Nadam	1000	61	7.07	3.45	14.08
p=5 q=6	2	*tanh*	Nadam	1000	181	8.34	2.97	8.97

Cuadro 3.13: Resultados de entrenameinto para el modelo de predicción con diferentes arquitecturas

Se observa que a diferencia del modelo de Wiener-Hammerstein, el modelo del horno de gas no es posible obtenerlo usando un solo perceptrón, y después de probar varias configuraciones se obtiene el modelo con el mínimo error que es la última fila de la tabla 3.13 y la grafica 3.14 muestra su resultado.

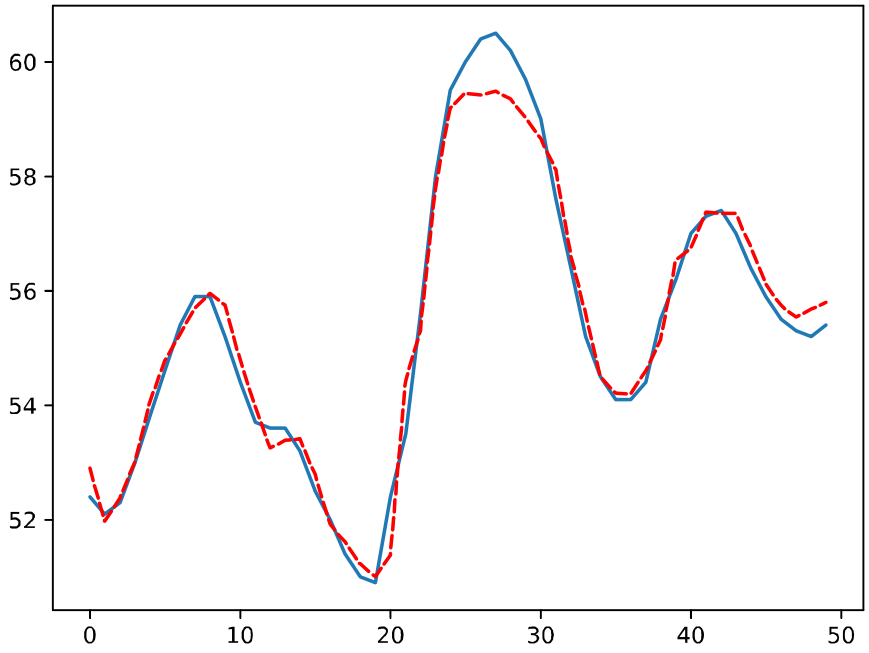


Figura 3.14: Resultado del modelo de predicción $N(\cdot)$ para el conjunto de prueba (linea punteada) contra el resultado esperado, usando 6 capas de 5 neuronas, el algoritmo Nadam de optimización, $n = 2$ y 1000 épocas

Modelo de predicción LSTM-NN

Es momento de probar $N(\cdot)^*$ con el ejemplo del horno de gas, dada la simplicidad de la arquitectura de la red solo habrá que ocuparse de seleccionar acertadamente n ya sea para el modelo de predicción o el de simulación, además del número k de celdas LSTM en la primera capa oculta. También se podría jugar con el algoritmo de optimización, pero de aquí en adelante se usara RMSprop cuando $k = 1$ y Nadam cuando $k > 1$, aunque en teoría se podrían usar indistintamente, en los ejemplos anteriores se comprobó experimentalmente que funcionan mejor de esta forma.

Primero es turno del modelo de predicción.Se puede ver en la tabla 3.14 que con $N(\cdot)^*$ se logro un error de $7x10^{-4}$ que es mejor que el encontrado con la MLP, la figura 3.15 muestra el resultado.

k	n	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
1	1	RMSprop	50	18	2.5	390	510
1	2	RMSprop	50	26	2.5	88	153
1	3	RMSprop	50	34	2.5	138	229
2	2	Nadam	50	59	2.5	309	470
1	2	RMSprop	100	26	3	26,44	49,84
1	2	RMSprop	500	26	7	6,6	18,16
1	2	RMSprop	1000	26	12.5	2,3	8,0
1	2	RMSprop	1500	26	17	1,8	7,7

Cuadro 3.14: Resultados de entrenamiento para el horno de gas, modelo de predicción con diferentes algoritmos de optimización

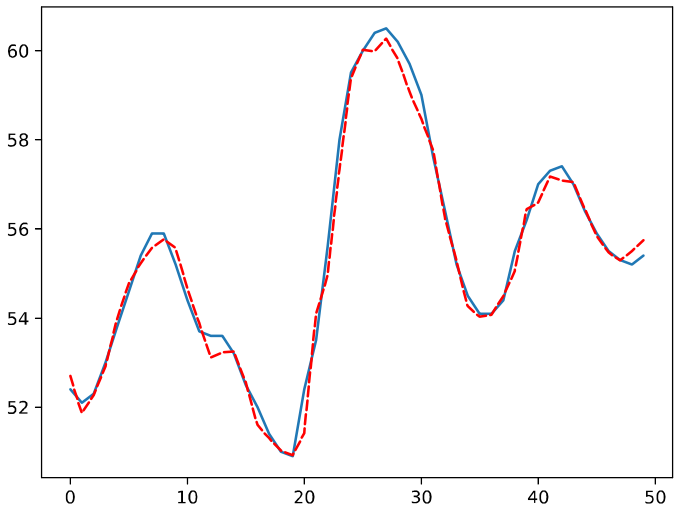


Figura 3.15: Mejor resultado obtenido del modelo de predicción de la LSTM-NN para el horno de gas.

Modelo de simulación MLP

Para el modelo de simulación se tomará en cuenta la estructura ya encontrada para el modelo de predicción y se probará primero para diferentes valores de n, para después ajustar si es necesario la estructura de la red.

$N(\cdot)$	n	f	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
p=5 q=6	3	*tanh*	Nadam	1000	176	9.37	116	218
p=5 q=6	5	*tanh*	Nadam	1000	186	9.74	27.21	72.71
p=5 q=6	7	*tanh*	Nadam	1000	196	9.31	9.44	51.90
p=5 q=6	7	*tanh*	Nadam	2000	196	17.9	8.04	39.78
p=5 q=6	10	*tanh*	Nadam	1000	211	9.31	8.87	41.94
p=5 q=6	15	*tanh*	Nadam	1000	236	9.04	5.07	50.63

Cuadro 3.15: Resultados de entrenameinto para el modelo de simulación con diferentes arquitecturas

Se observa en la tabla 3.15 que el mejor resultado es el obtenido con $n = 7$ y despues de 2 mil épocas. El resultado se muestra en la figura 3.16.

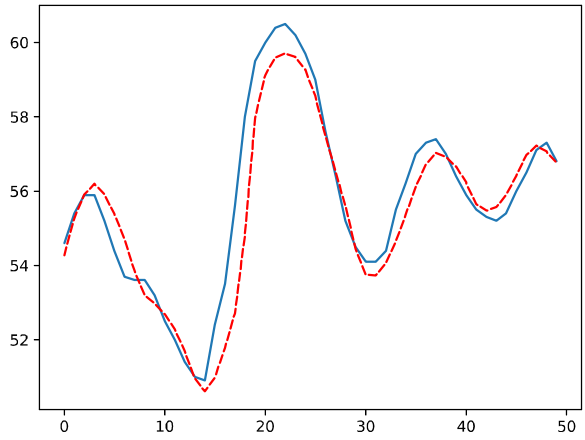


Figura 3.16: Resultado del modelo de simulación para el conjunto de prueba (linea punteada) contra el resultado esperado, usando 6 capas de 5 neuronas, el algoritmo Nadam de optimización, $n = 2$ y 2000 épocas

Modelo de simulación LSTM-NN

Ahora se diseñará el modelo de simulación únicamente variando n para $N(\cdot)^*$ encontrado anteriormente.

k	n	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
1	2	RMSprop	500	18	8	299	377
1	5	RMSprop	500	30	8	57,34	123
1	7	RMSprop	500	38	8	10,98	44,87
1	9	RMSprop	500	46	8	7,62	41,74
1	10	RMSprop	500	50	8	7,49	43,93

Cuadro 3.16: Resultados de entrenamiento para el horno de gas, modelo de simulación con diferentes algoritmos de optimización

En la tabla 3.16 se observa que con realmente muy poco esfuerzo y usando $N(\cdot)^*$ se logra un resultado muy similar al de la red MLP con un error de 4x10-3, la gráfica se muestra en la figura 3.17.

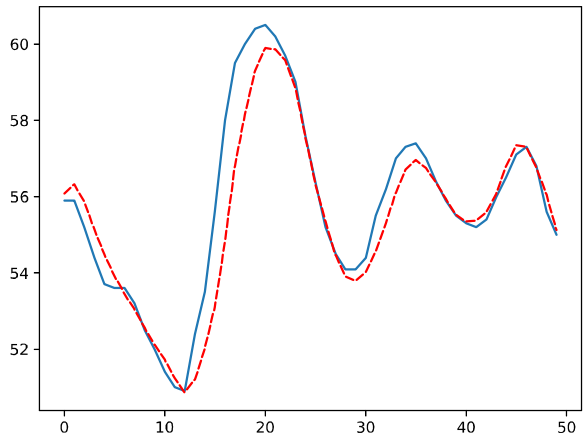


Figura 3.17: Mejor resultado obtenido del modelo de simulación de $N(\cdot)^*$ para el horno de gas.

3.3.3. Sistema no lineal

El sistema a modelar en este ejemplo es el siguiente

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u^3(k)$$

con

$$u(k) = A\sin(\frac{\pi k}{50}) + B\sin(\frac{\pi k}{20})$$

Para modelar el sistema se generó un conjunto de datos de prueba de 80,000 ejemplos donde $A = B = 1$ y para probar el modelo se usó un conjunto de datos con entrada $u(k)$ donde $A = 0{,}9$ y $B = 1{,}1$.

Modelo de predicción MLP

En las pruebas se usa nuevamente el proceso propuesto anteriormente donde se toma como base el algoritmo de optimización Nadam y se usa de la función $ReLU$, para después cambiar a la función $tanh$.

$N(\cdot)$	n	f	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
p=1 q=1	1	$ReLU$	Nadam	10	3	14.09	5.02	5.43
p=1 q=1	2	$ReLU$	Nadam	10	5	14.15	0.28	0.34
p=1 q=1	3	$ReLU$	Nadam	10	7	14.22	0.59	0.72
p=2 q=2	2	$tanh$	Nadam	10	19	16.70	1.64	1.73
p=2 q=10	2	$tanh$	Nadam	10	73	27.71	1.86	1.94
p=5 q=2	2	$tanh$	Nadam	20	61	33.13	0.186	0.225
p=5 q=10	2	$tanh$	Nadam	30	331	81.25	0.074	0.090
p=5 q=10	2	$tanh$	Nadam	50	331	150	0.068	0.076

Cuadro 3.17: Resultados de entrenamiento para el modelo de simulación con diferentes arquitecturas

Se encontró al final un modelo muy bueno con un error de tan solo $3{,}5x10^{-6}$ con solo 50 épocas, posiblemente exista una estructura mejor, pero esta es lo suficientemente buena para detener la búsqueda.

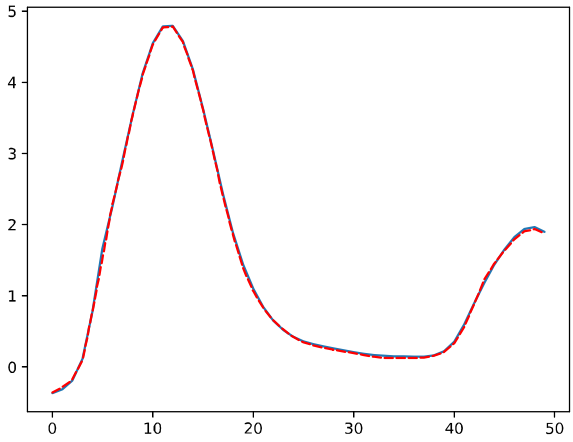


Figura 3.18: Mejor resultado para el modelo de predicción usando MLP

Modelo de predicción LSTM-NN

Usaremos $N(\cdot)^*$ para encontrar el modelo de predicción.

k	n	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
1	1	RMSprop	10	18	39	5,47	5,89
1	2	RMSprop	10	26	40	0,673	0,748
1	3	RMSprop	10	34	40	1,08	1,28
2	2	RMSprop	10	59	41	0,471	0,524
2	2	Nadam	10	59	42	0,290	0,351
3	2	Nadam	50	100	204	0,100	0,114
3	2	Nadam	300	100	1155	0,079	0,093
3	2	Nadam	500	100	1991	0,059	0,073

Cuadro 3.18: Resultados de entrenamiento para el SNL, modelo de predicción con diferentes algoritmos de optimización

Después de 500 épocas se logró un error de $7x10^{-6}$ al igual que con la MLP, el resultado se muestra en la figura 3.18

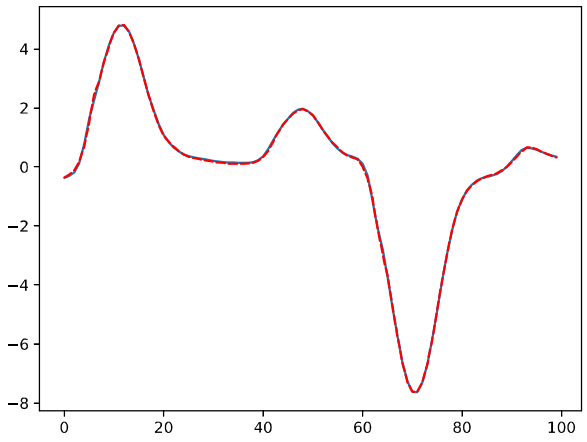


Figura 3.19: Mejor resultado obtenido del modelo de predicción $N(\cdot)^*$ para el SNL

Modelo de simulación MLP

A continuación se encontrará un modelo de simulación adecuado, Siguiendo la misma metodología se sabe que se obtiene un buen resultado haciendo de n un valor grande y tomando como base el modelo de predicción.

$N(\cdot)$	n	f	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
p=5 q=10	1	*tanh*	Nadam	10	316	26.94	9.43	10.46
p=5 q=10	3	*tanh*	Nadam	10	326	26.75	0.809	0.925
p=5 q=10	5	*tanh*	Nadam	10	336	26.95	0.701	0.820
p=5 q=10	7	*tanh*	Nadam	10	346	25.80	0.500	0.660
p=5 q=10	10	*tanh*	Nadam	10	361	26.22	0.448	0.659
p=5 q=10	15	*tanh*	Nadam	10	386	26.86	0.313	0.436
p=5 q=10	20	*tanh*	Nadam	10	411	25.94	0.548	0.780
p=5 q=10	15	*tanh*	Nadam	20	386	59.61	0.089	0.195

Cuadro 3.19: Resultados de entrenameinto para el modelo de simulación con diferentes arquitecturas

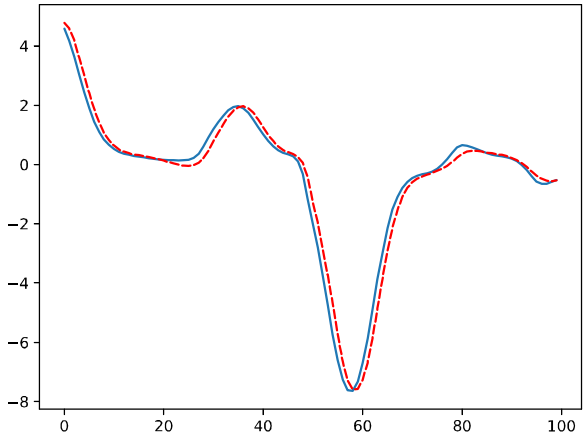


Figura 3.20: Mejor resultado del modelo de simulación para el sistema no lineal usando MLP

Modelo de simulación LSTM-NN

Se seguira con el mismo procedimiento que fue usado anteriormente para encontrar el modelo de simulación.

k	n	Optimizacion	Épocas	θ	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
3	5	Nadam	50	112	196	0,405	0,551
3	10	Nadam	50	232	172	0,458	0,598
3	9	Nadam	200	160	990	0,286	0,413
3	9	Nadam	500	160	2574	0,244	0,387
3	9	Nadam	10000	160	40004	0,091	0,217
3	9	Nadam	20000	160	40004	0,091	0,217
6	9	Nadam	50	391	200	0,143	0,286
6	9	Nadam	400	391	1663	0,081	0,202
6	9	Nadam	600	391	2436	0,067	0,198

Cuadro 3.20: Resultados de entrenameinto para el SNL, modelo de simulación LSTM-NN con diferentes algoritmos de optimización

Como se puede observar en la tabla 3.20 con $k = 6$ y $n = 9$ después de 600 épocas se obtiene un error de $1x10^{-5}$ al igual que con la MLP y el resultado se muestra en la figura 3.21

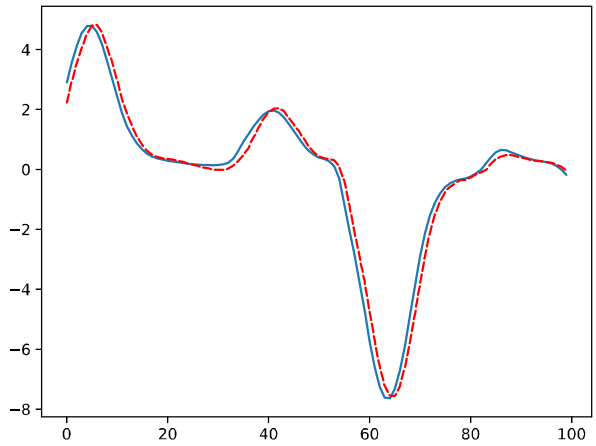


Figura 3.21: Mejor resultado obtenido del modelo de simulación de la MBS-LSTM para el snl

3.3.4. MLP frente a LSTM-NN

A continuación, se muestran tablas comparativas donde se podrán observar claramente los resultados de cada uno de los ejemplos ya sea para el modelo de predicción o de simulación usando el enfoque conocido de MLPs contra la LSTM-NN.

De estas tablas se tienen los siguientes aspectos:

1. Las dos arquitecturas llegan a un resultado muy similar, incluso en términos prácticos se podría decir que es el mismo resultado.

2. El tiempo de entrenamiento en general es mayor para la LSTM-NN.

3. A diferencia de lo que se podría pensar dada la complejidad de las celdas LSTM, el número de parámetros de la MLP crece mucho más junto con la complejidad del problema que en el caso de la LSTM-NN.

4. La arquitectura de la LSTM-NN siempre se mantiene simple y fácil de encontrar.

Por lo tanto, es fácil observar que la LSTM-NN es muy superior que la MLP ya que resuelve el problema de "*Vanishing Gradient*", el número de parámetros necesarios de manera general es menor, encontrar una configuración que resuelva el problema es sumamente sencillo y se obtienen los mejores resultados. El único inconveniente que tiene es el tiempo ya que tarda más en entrenarse que las MLP.

Arquitectura	parametros	p	q	n	épocas	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
MLP	5	1	1	2	100	62	0.088	0.098
LSTM-NN	52	1	1	4	100	70	0.043	0.045

Cuadro 3.21: Resultados MLP contraLSTM-NN para Wiener-Hammerstein, modelo de predicción

Arquitectura	parametros	p	q	n	épocas	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
MLP	4176	25	6	35	100	102	0.537	0.635
LSTM-NN	472	3	1	35	100	221	0.627	0.699

Cuadro 3.22: Resultados MLP contraLSTM-NN para Wiener-Hammerstein, modelo de simulación

Arquitectura	parametros	p	q	n	épocas	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
MLP	181	5	6	2	1000	8.34	2.97	8.97
LSTM-NN	26	1	1	2	1500	17	1.8	7.7

Cuadro 3.23: Resultados MLP contraLSTM-NN para el horno de gas, modelo de predicción

Arquitectura	parametros	p	q	n	épocas	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
MLP	196	5	6	7	2000	9.31	8.04	39.78
LSTM-NN	50	1	1	10	500	8	7,49	43,93

Cuadro 3.24: Resultados MLP contraLSTM-NN para el horno de gas, modelo de simulación

Arquitectura	parametros	p	q	n	épocas	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
MLP	331	5	10	2	50	150	0.078	0.076
LSTM-NN	100	3	1	2	500	1991	0.059	0.073

Cuadro 3.25: Resultados MLP contraLSTM-NN para el snl, modelo de predicción

Arquitectura	parametros	p	q	n	épocas	T_e (s)	MSE_E $(x10^{-4})$	MSE_P $(x10^{-4})$
MLP	386	5	10	15	20	59.61	0.089	0.195
LSTM-NN	391	6	1	9	600	2436	0.067	0.198

Cuadro 3.26: Resultados MLP contraLSTM-NN para el snl, modelo de simulación

Capítulo 4

Entrenamiento estable para la RNN LSTM-NN profunda

Si bien en el capítulo anterior la red LSTM-NN demostró tener múltiples ventajas frente a las redes neuronales tradicionales, una desventaja presente en todo momento fue la complejidad del entrenamiento, pues la retropropagacion a través del tiempo es realmente lenta, además de que el resultado podría llegar a ser incierto, nada garantiza que bajo la selección de parámetros seleccionados el error converge.

Por estas razones, este capítulo se dedica a la obtención de un entrenamiento estable para las redes LSTM y en especial para la arquitectura LSTM-NN antes descrita.

4.1. Celda GRU como estrcutura FNN-RNN

Para simplificar las ecuaciones sin perder generalidad y por su especial similitud como se mencionó anteriormente, se usará como bloque una estructura basada en la celda GRU.

Se compone de dos compuertas únicamente que son

1) Compuerta de olvido:

$$F(k) = \sigma\left(W_F(k)\left[y(k-1), u(k)\right]\right) \tag{4.1}$$

2) Compuerta de entrada:

$$\begin{aligned} I(k) &= \sigma\left(W_I(k)\left[y(k-1), u(k)\right]\right) \\ \tilde{y}(k) &= \tanh\left(W_X(k)\left[I(k)\, y(k-1)\right]\right) \end{aligned} \tag{4.2}$$

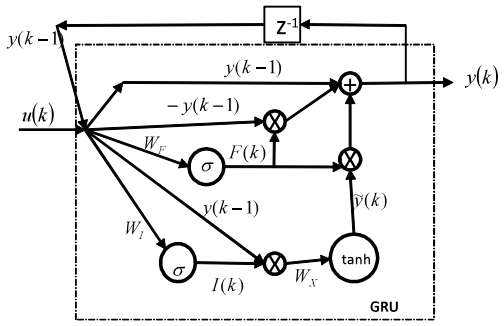


Figura 4.1: La celda GRU

El estado de la celda y el estado oculto se mezclan como

$$\hat{y}(k) = \hat{y}(k-1) + F(k)\left[\tilde{y}(k) - \hat{y}(k-1)\right] \tag{4.3}$$

Para cada bloque se utiliza $x(k)$ como el estado interno, la salida de este bloque es $y(k) = x(k+1)$, la entrada es $u(k)$. De tal forma que la dinámica de la GRU (4.3) es

$$x_{k+1} = x_k - \sigma\left(W_1 x_k + W_u u_k\right) x_k + \sigma\left(W_1 x_k + W_u u_k\right) \phi\left(W_2\left[\sigma\left(W_3 x_k\right) x_k\right]\right)$$

La celda GRU (4.3), mostrada en la figura 4.1 es entonces una simplificación de la celda LSTM.

El bloque GRU en la figura 4.1 se puede transformar en la figura 4.2. De tal forma que la GRU sea una Red Neuronal Recurrente (RNN) con una red *Feedforward* (FNN)

$$\begin{array}{ll} \text{RNN:} & x(k+1) = Ax(k) - \sigma\left[W_1 x(k) + W_u u(k)\right] \tilde{u}(k) \\ \text{FNN:} & \tilde{u}(k) = \phi\left(W_2 \sigma\left[W_3 x(k)\right] x(k)\right) + x(k) \end{array} \tag{4.4}$$

donde $A = I$, $\tilde{u}_k$ es la entrada virtual de FNN a RNN.

Para entrenar la FNN y la RNN en la figura 4.2, el error de entrenamiento tiene que ser retropropagado del bloque RNN a el bloque FNN. El error de entrenamiento a la salida del RNN esta definido como e_R, entonces el error en la salida del bloque FNN se define como e_F, entonces

$$e_F = \frac{\partial \sigma}{\partial t} e_R \tag{4.5}$$

donde σ es la función de activación de la GRU.

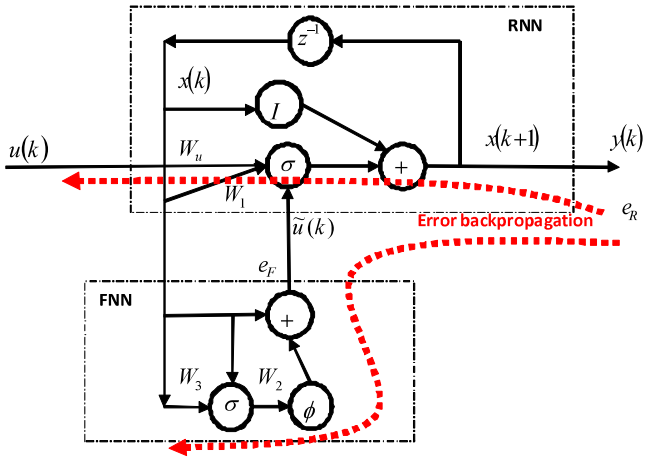


Figura 4.2: Celda GRU como estrcutura FNN-RNN

4.2. Estabilidad para FNN

Primero se diseña un método de entrenamiento estable para la red feedforward (FNN). La entrada de la FNN en (4.4) es $x(k)$,la salida es $\tilde{u}(k)$, el error de entrenamiento es e_R definido en (4.5). Este error de identificacion puede ser representado como

$$e_F(k) = \phi\left(W_{2,k}x(k)\,\sigma\left[W_{3,k}x(k)\right]\right) - \phi\left(W_2^*x(k)\,\sigma\left[W_3^*x(k)\right]\right) + \mu_1(k) \tag{4.6}$$

donde W_2^* y W_3^* son un conjunto de pesos desconocidos, los cuales pueden minimizar el error de modelado $\mu_1(k)$.

Usando series de Tylor alrededor del punto $W_{3,k}x(k)$ y $W_{2,k}$, el error de identificación puede ser representado como

$$\begin{aligned} e_F(k) &= \phi\left(W_{2,k}x(k)\,\sigma\left[W_{3,k}x(k)\right]\right) - \phi\left(W_2^*x(k)\,\sigma\left[W_3^*x(k)\right]\right) + \mu_1(k) \\ &= \phi' x^2(k)\left(W_{2,k} - W_2^*\right)\sigma + \phi'\sigma' W_{2,k}x^2(k)\left(W_{3,k} - W_3^*\right) + \mu_1(k) + \varepsilon_1(k) \\ &= \phi' x^2(k)\,\tilde{W}_{2,k}\sigma + \phi'\sigma' W_{2,k}x^2(k)\,\tilde{W}_{2,k} + \delta_1(k) \end{aligned} \tag{4.7}$$

ϕ' y σ' son las derivadas de la función de activación no lineal $\phi(\cdot)$ y $\sigma(\cdot)$ en los puntos $W_{2,k}x(k)$ y $W_{3,k}x(k)$. $\tilde{W}_{2,k} = \tilde{W}_{2,k} - W_2^*$, $\tilde{W}_{3,k} = \tilde{W}_{3,k} - W_3^*$,

$$\delta_1(k) = \mu_1(k) + \varepsilon_1(k) \tag{4.8}$$

aqui $\varepsilon_1(k)$ es el error de aproximacion de segundo orden de las series de Taylor.

En el presente trabajo se presenta la identificación a lazo abierto, se asume que la planta es BIBO estable, *i.e.*, $y(k)$ y $u(k)$ estan acotadas. Por la cota de la función sigmoide ϕ se puede asumir que $\delta_1(k)$ in (4.8) esta acotada. Entonces el siguiente teorema ofrece un algoritmo de retropropagación estable para FNN en la figura 4.2.

Teorema 1

Para la parte *feedforward* de la LSTM, FNN, el siguiente algoritmo de retropropagación puede hacer al error de identificación $e_F(k)$ acotado

$$\begin{aligned} W_{2,k+1} &= W_{2,k} - \eta_k e_F(k)\,\phi' x^2(k)\,\sigma \\ W_{3,k+1} &= W_{3,k} - \eta_k e_F(k)\,\phi'\sigma' W_{2,k} x^2(k) \end{aligned} \tag{4.9}$$

donde $\eta_k = \dfrac{\eta}{1 + \|\phi' x^2(k)\,\sigma\|^2 + \|\phi'\sigma' W_{2,k} x^2(k)\|^2}$, $0 < \eta \leq 1$. El promedio del error de identificación satisface

$$J = \lim_{T\to\infty} \sup \frac{1}{T} \sum_{k=1}^{T} e_F^2(k) \leq \frac{\eta}{\pi} \bar{\delta}_1 \tag{4.10}$$

donde $\pi = \dfrac{\eta}{1+\kappa}\left[1 - \dfrac{\kappa}{1+\kappa}\right] > 0$, $\kappa = \max_k \left(\|\phi' x^2(k)\,\sigma\|^2 + \|\phi'\sigma' W_{2,k} x^2(k)\|^2\right)$, $\bar{\delta}_1 = \max_k \left[\delta_1^2(k)\right]$

Prueba

Se selecciona una matriz definida positiva L_k como

$$L_k = \left\|\tilde{W}_{2,k}\right\|^2 + \left\|\tilde{W}_{3,k}\right\|^2 \tag{4.11}$$

Despues de la regla de ajuste (4.9), se tiene

$$\tilde{W}_{2,k+1} = \tilde{W}_{2,k} - \eta_k e_F(k)\,\phi' x^2(k)\,\sigma, \quad \tilde{W}_{3,k+1} = \tilde{W}_{3,k} - \eta_k e_F(k)\,\phi'\sigma' W_{2,k} x^2(k)$$

Como ϕ' es una matriz diagonal, y usando (4.7) se tiene que

$$\begin{aligned}
&\Delta L_k = \left\|\tilde{W}_{2,k} - \eta_k e_F(k)\,\phi' x^2(k)\,\sigma\right\|^2 + \left\|\tilde{W}_{3,k} - \eta_k e_F(k)\,\phi'\sigma' W_{2,k} x^2(k)\right\|^2 \\
&- \left\|\widetilde{W}_k\right\|^2 - \left\|\widetilde{V}_k\right\|^2 \\
&= \eta_k^2 e_F(k)\left(\left\|\phi' x^2(k)\,\sigma\right\|^2 + \left\|\phi'\sigma' W_{2,k} x^2(k)\right\|^2\right) \\
&-2\eta_k \left\|e_F(k)\right\| \left\|\phi' x^2(k)\,\sigma\tilde{W}_{2,k} + \phi'\sigma' W_{2,k} x^2(k)\,\tilde{W}_{3,k}\right\| \\
&= \eta_k^2 e_F^2(k)\left(\left\|\phi' x^2(k)\,\sigma\right\|^2 + \left\|\phi'\sigma' W_{2,k} x^2(k)\right\|^2\right) - 2\eta_k \left\|e_F(k)\left[e_F(k) - \delta_1(k)\right]\right\| \\
&\leq -\eta_k e_F^2(k)\left[1 - \eta_k\left(\left\|\phi' x^2(k)\,\sigma\right\|^2 + \left\|\phi'\sigma' W_{2,k} x^2(k)\right\|^2\right)\right] + \eta\delta_1^2(k) \\
&\leq -\pi e_F^2(k) + \eta\delta_1^2(k)
\end{aligned} \tag{4.12}$$

donde π esta definida en (4.10). Porque

$$n\left[\text{mín}\left(\tilde{w}_{2,i}^2\right) + \text{mín}\left(\tilde{w}_{3,i}^2\right)\right] \leq L_k \leq n\left[\text{máx}\left(\tilde{w}_{2,i}^2\right) + \text{máx}\left(\tilde{w}_{3,i}^2\right)\right]$$

donde $n\left[\text{mín}\left(\tilde{w}_{2,i}^2\right) + \text{mín}\left(\tilde{w}_{3,i}^2\right)\right]$ y $n\left[\text{máx}\left(\tilde{w}_{2,i}^2\right) + \text{máx}\left(\tilde{w}_{3,i}^2\right)\right]$ son funciones $\mathcal{K}_\infty$, y $\pi e_F^2(k)$ es una función $\mathcal{K}_\infty$, $\eta\delta_1^2(k)$ es una función $\mathcal{K}$.

Se conoce L_k que es la función de $e_F(k)$ y $\delta_1(k)$, entonces L_k admite una función suave ISS-Lyapunov, la dinámica de identificación del error es entrada-estado estable . porque la entrada $\delta_1(k)$ esta acotada y la dinámica es ISS, el estado $e_F(k)$ es acotado.

Prueba

(4.12) puede ser reescrita como

$$\Delta L_k \leq -\pi e_F^2(k) + \eta\delta^2(k) \leq \pi e_F^2(k) + \eta\bar{\delta}_1 \tag{4.13}$$

Sumando (4.13) de 1 a T, y usando $L_T > 0$ y L_1 una constante, se tiene

$$\begin{aligned}
L_T - L_1 &\leq -\pi\textstyle\sum_{K=1}^{T} e_F^2(k) + T\eta\bar{\delta}_1 \\
\pi\textstyle\sum_{K=1}^{T} e_F^2(k) &\leq L_1 - L_T + T\eta\bar{\delta}_1 \leq L_1 + T\eta\bar{\delta}_1
\end{aligned}$$

(4.10) esta establecida.

Debido a que hay una retroalimentación externa entre las entradas y salidas, los gradientes de los errores en la RNN dependen de sus valores previos. Por lo tanto, el método de entrenamiento para la FNN no puede ser aplicado a la RNN.

4.3. Estabilidad para RNN

La RNN en (4.4) puede ser reescrita como

$$x(k+1) = Ax(k) - \sigma\left[W_1 x(k) + W_u u(k)\right]\tilde{u}(k) \tag{4.14}$$

donde $A = \frac{1}{a}I$, a es una constante positiva $a \geq 1$ la cual es un parámetro de diseño, entonces A es estable.

De manera similar con (4.6), el error de identificación $e_R(k)$ puede ser representado como

$$e_R(k+1) = Ae_R(k) + \sigma\left[W_1 x(k) + W_u u(k)\right]\tilde{u}(k) - \sigma\left[W_1^* x(k) + W_u^* u(k)\right]\tilde{u}(k) + \mu_2(k) \tag{4.15}$$

donde W_1^* y W_u^* son pesos constantes que pueden minimizar el error de modelado $\mu_2(k)$.

Usando series de Tylor al rededor de los puntos $W_1(k)x(k)$ y $W_u u(k)$

$$\sigma\left[W_1 x(k) + W_u u(k)\right]\tilde{u}(k) - \sigma\left[W_1^* x(k) + W_u^* u(k)\right]\tilde{u}(k) = \sigma'\tilde{W}_{1,k}x(k) + \sigma'\tilde{W}_{u,k}u(k) + \varepsilon_2(k) \tag{4.16}$$

donde $\tilde{W}_{1,k} = W_{1,k} - W_1^*$, $\tilde{W}_{u,k} = W_{u,k} - W_u^*$, $\varepsilon_2(k)$ es el error de aproximación de segundo orden. σ' es la derivada de la función de activación $\sigma(\cdot)$ en el punto $W_1 x(k) + W_u u(k)$

Entonces

$$e_R(k+1) = Ae_R(k) + \sigma'\tilde{W}_{1,k}x(k) + \sigma'\tilde{W}_{u,k}u(k) + \delta_2(k) \tag{4.17}$$

donde $\delta_2(k) = \mu_2(k) + \varepsilon_2(k)$.

El siguiente teorema ofrece un algoritmo de aprendizaje estable de la parte RNN.

Teorema 2

Para la parte recurrente de la LSTM, RNN, la siguiente ley de actualización de gradiente puede hacer al error de identificación $e_R(k)$ acotado

$$\begin{aligned} W_{1,k+1} &= W_{1,k} - \eta_k \sigma' x(k)\tilde{u}(k)e_R^T(k) \\ W_{u,k+1} &= W_{u,k} - \eta_k \sigma' u(k)\tilde{u}(k)e_R^T(k) \end{aligned} \tag{4.18}$$

donde $\eta(k)$ satisface

$$\eta_k = \begin{cases} \dfrac{\eta}{1 + \|\sigma' x(k)\,\tilde{u}(k)\|^2 + \|\sigma' u(k)\,\tilde{u}(k)\|^2} & \text{if } a\|e_R(k+1)\| \geq \|e_R(k)\| \\ 0 & \text{if } a\|e_R(k+1)\| < \|e_R(k)\| \end{cases}$$

$0 < \eta \leq 1$, a el eigenvalor de A

Prueba

Se selecciona la función de Lyapunov como

$$V_k = \left\|\tilde{W}_{1,k}\right\|^2 + \left\|\tilde{W}_{u,k}\right\|^2$$

donde $\left\|\tilde{W}_{1,k}\right\|^2 = \sum_{i=1}^{n} \tilde{w}_{1,k}^2 = tr\left\{\tilde{W}_{1,k}^T \tilde{W}_{1,k}\right\}$. De la ley de actualización (4.18)

$$\tilde{W}_{1,k+1} = \tilde{W}_{1,k} - \eta_k \sigma' x(k)\,\tilde{u}(k)\,e_R^T(k)$$

Entonces

$$\begin{aligned} &\Delta V_k = V_{k+1} - V_k \\ &= \left\|\tilde{W}_{1,k} - \eta_k \sigma' x(k)\,\tilde{u}(k)\,e_R^T(k)\right\|^2 - \left\|\tilde{W}_{1,k}\right\|^2 \\ &+ \left\|\tilde{W}_{u,k} - \eta_k \sigma' u(k)\,\tilde{u}(k)\,e_R^T(k)\right\|^2 - \left\|\tilde{W}_{u,k}\right\|^2 \\ &= \eta_k^2 \|e_R(k)\|^2 \|\sigma' x(k)\,\tilde{u}(k)\|^2 - 2\eta_k \left\|\sigma' \tilde{W}_{1,k} x(k)\,\tilde{u}(k)\,e_R^T(k)\right\| \\ &+ \eta_k^2 \|e_R(k)\|^2 \|\sigma' u(k)\,\tilde{u}(k)\|^2 - 2\eta_k \left\|\sigma' u(k)\,\tilde{u}(k)\,\tilde{W}_{u,k} e_R^T(k)\right\| \end{aligned}$$

Existe una constante $a \geq 1$, tal que

Prueba

1)Si $a\|e_R(k+1)\| \geq \|e_R(k)\|$, usando (4.17) y $\eta_k \geq 0$,

$$\begin{aligned} &-2\eta_k \left\|\sigma' \tilde{W}_{1,k} x(k)\,\tilde{u}(k)\,e_R^T(k)\right\| - 2\eta(k) \left\|\sigma' u(k)\,\tilde{u}(k)\,\tilde{W}_{u,k} e_R^T(k)\right\| \\ &\leq -2\eta_k \left\|e_R^T(k)\right\| \|a e_R(k+1) - A e_R(k) - \delta_2(k)\| \\ &= -2\eta_k \left\|e_R^T(k)\,ae(k+1) - e_R^T(k)\,A e_R(k) - e_R^T(k)\,\delta_2(k)\right\| \\ &\leq -2\eta_k \left\|e_R^T(k)\,a e_R(k+1)\right\| + 2\eta_k e_R^T(k)\,A e_R(k) + 2\eta(k) \left\|e_R^T(k)\,\delta_2(k)\right\| \\ &\leq -2\eta_k \|e_R(k)\|^2 + 2\eta_k \lambda_{\text{máx}}(A) \|e_R(k)\|^2 + \eta_k \|e_R(k)\|^2 + \eta(k) \|\delta_2(k)\|^2 \end{aligned}$$

como $0 < \eta \leq 1$, ,

$$\begin{aligned}
&\Delta V_k \leq \eta_k^2 \left\| e_R(k) \right\|^2 \left\| \sigma' x(k) \tilde{u}(k) \right\|^2 + \eta_k^2 \left\| e_R(k) \right\|^2 \left\| \sigma' u(k) \tilde{u}(k) \right\|^2 \\
&-\eta_k \left\| e_R(k) \right\|^2 + 2\eta(k) \lambda_{\text{máx}}(A) \left\| e_R(k) \right\|^2 + \eta_k \left\| \delta_2(k) \right\|^2 \\
&= -\eta_k \left[(1 - 2\lambda_{\text{máx}}(A)) - \eta \frac{\left\| \sigma' x(k) \tilde{u}(k) \right\|^2 + \left\| \sigma' u(k) \tilde{u}(k) \right\|^2}{1 + \left\| \sigma' x(k) \tilde{u}(k) \right\|^2 + \left\| \sigma' u(k) \tilde{u}(k) \right\|^2} \right] e_R^2(k) + \eta_k \delta_2^2(k) \\
&\leq -\pi e_R^2(k) + \eta \delta_2^2(k)
\end{aligned} \tag{4.19}$$

donde $\pi = \dfrac{\eta}{1+\kappa} \left[1 - 2\lambda_{\text{máx}}(A) - \dfrac{\kappa}{1+\kappa} \right]$, $\kappa = \underset{k}{\text{máx}} \left(\left\| \sigma' x(k) \tilde{u}(k) \right\|^2 + \left\| \sigma' u(k) \tilde{u}(k) \right\|^2 \right)$.
como $-1 < \lambda(A) < 0$, $\pi > 0$

$$n \,\text{mín}\left(\widetilde{w}_i^2\right) \leq V_k \leq n \,\text{máx}\left(\widetilde{w}_i^2\right)$$

donde $n \times \text{mín}(\widetilde{w}_i^2)$ y $n \times \text{máx}(\widetilde{w}_i^2)$ son funciones $\mathcal{K}_\infty$, y $\pi e_R^2(k)$ es una función $\mathcal{K}_\infty$, $\eta \delta_2^2(k)$ es una función $\mathcal{K}$, entonces V_k admite la función suave ISS-Lyapunov, la dinámica del error de identificación entrada-salida estable. La "entrada" corresponde al segundo término de la última linea en (4.19), *i.e.*, el error de moedelado $\delta_2(k) = \mu_2(k) + \varepsilon_2(k)$, el "estado" corresponde al primer término de la última línea en (4.19), *i.e.*, el error de identificación $e_R(k)$. Como la "entrada" $\zeta(k)$ es acotada y la dinámica es ISS, el "estado" $e(k)$ es acotado.

2) Si $a \left\| e(k+1) \right\| < \left\| e(k) \right\|$, $\Delta V_k = 0$. V_k es constante, $W_{1,k}$ y $\tilde{W}_{u,k}$ son constantes. Entonces $\left\| e_R(k+1) \right\| < \frac{1}{a} \left\| e_R(k) \right\|$, $\frac{1}{a} < 1$, $e_R(k)$ es acotada.

Nota

La condición $a \left\| e(k+1) \right\| \geq \left\| e(k) \right\|$ es la zona muerta. Si a se selecciona lo suficientemente grande, la zona muerta se vuelve pequeña. Donde $a = 1$, es una LSTM estandar. Los algoritmos estables (4.9) y (4.18) son mucho mas simples que el de retropropagación a través del tiempo (BPTT), el cual es usado en la mayoria de las redes LSTM. Otra ventaja importante de este algoritmo es que puede ser aplicado para modelado en linea, mientras que BPTT es un proceso por lotes, la identificaciín en linea con BPTT es demasiado lenta.

Igual que antes se sabe que el nuevo modelo propuesto LSTM-NN es una estructura jerárquica, cuyo objetivo es entrenar los pesos de tal forma que el error entre la salida del modelo, y la salida de la planta sea minimizado. El rendimiento se define como

$$J = \frac{1}{2} e_0^2 \quad e_o = \hat{y} - y, \quad \hat{y} = \phi \left[W_{N,k} \hat{y}_i \right]$$

El ultimo bloque, el bloque NN, es una MLP clásica, cuyo entrenamiento es el gradiente descendente

$$W_N(k+1) = W_N(k) - \eta \hat{y}_{i,q}(k)\, e_o(k) \tag{4.20}$$

donde $\eta > 0$ es la taza de aprendizaje y $\hat{y}_{i,q}$, $i = 1 \cdots p$, son las salidas de la última capa de LSTMs. El error es retropropagado a las celdas LSTM como $e_n = e_o \phi' W_N$

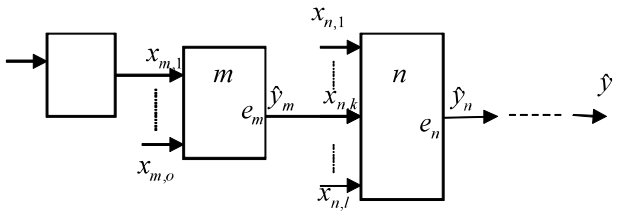


Figura 4.3: Error de retropropagación

Las estructuras interiores de cada LSTM o GRU son las mismas, se puede usar la misma regla de aprendizaje para cada celda RNN. Los errores son retropropagados desde el bloque n hacia el bloque m si se conoce e_n, entonces e_m se puede obtener del modelo jerárquico. Para el bloque n, usamos la regla del gradiente descendente

$$w_n(k+1) = w_n(k) - \eta x_{n,k} e_n$$

donde $x_{n,k}$ es la salida del bloque n, la cual es la salida del bloque m, y_m.

$$e_m = \frac{\partial \sigma}{\partial t} w_u e_n \tag{4.21}$$

Como el interior de las estructuras LSTM son las mismas, se puede usar la misma técnica para entrenar cada bloque si se conoce el error de cada bloque, entonces se puede entrenar. El proceso de entrenamiento es el siguiente.

1. Calcular la salida de cada LSTM. Algunas salidas del modelo serán las entradas del siguiente nivel.

2. Calcular el error de cada bloque, empezando por el último bloque. Entonces retropropagar los errores de identificación

3. Entrenar los pesos para cada bloque independientemente.

4.4. Simulación de un sistema aerodinámico

Para probar este entrenamiento se utilizo el sistema aerodinámico mostrado en la figura 4.4

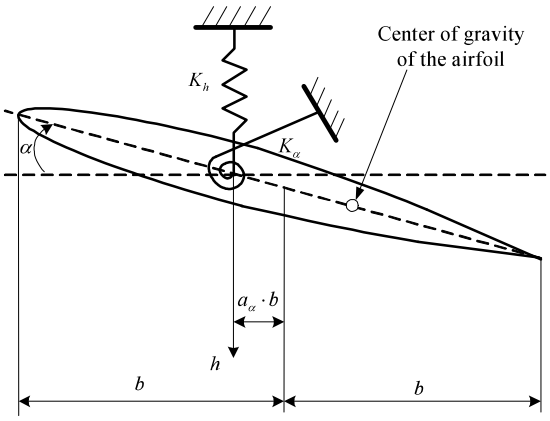


Figura 4.4: Sistema aerodinámico

Este sistema es un modelo de cuatro entradas y dos salidas que pueden ser representadas como

$$\left\{ \begin{array}{c} C_l(t) \\ C_m(t) \end{array} \right\} = \left\{ \begin{array}{c} f_1(M_\infty, \alpha_0, h, \alpha, t) \\ f_2(M_\infty, \alpha_0, h, \alpha, t) \end{array} \right\} \tag{4.22}$$

donde C_l y C_m son el coeficiente de elevación y el coeficiente del momento respectivamente, M_∞ y α_0 son el número de encuentro de flujo libre y el ángulo de ataque medio, respectivamente. h y α son el desplazamiento de hundimiento y cabeceo, respectivamente.

Distintas técnicas pueden ser usadas para reducir el orden del modelo y ponerlo de la siguiente forma

$$\left\{ \begin{array}{c} C_l(t) \\ C_m(t) \end{array} \right\} = \left\{ \begin{array}{c} \widehat{f}_1(h, \alpha, t) \\ \widehat{f}_2(h, \alpha, t) \end{array} \right\} \tag{4.23}$$

El problema es que el modelo aerodinámico de orden reducido es válido alrededor del entorno de la condición de vuelo donde se generan los datos de entrenamiento, y es casi imposible construir un modelo aerodinámico de orden reducido válido para un rango de condiciones de vuelo utilizando los modelos simplificados.

Entonces el reto es entrenar el modelo LSTM-NN para que sea válido sobre el rango de parámetros de vuelo de interés ($M_{\infty\,\text{mín}} < M_\infty < M_{\infty\,\text{máx}}, \alpha_{0\,\text{mín}} < \alpha_0 < \alpha_{0\,\text{máx}}$) que en este caso será el que se muestra en la figura 4.5

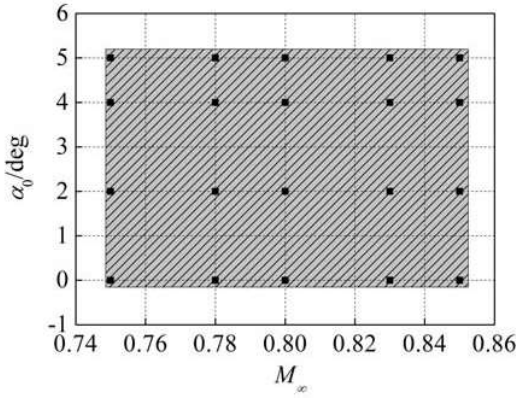


Figura 4.5: Rango de parámetros de interés

Los parámetros seleccionados conforman la matriz X

$$X = \begin{bmatrix} M_{\infty,1} & \alpha_{0,1} \\ M_{\infty,2} & \alpha_{0,2} \\ \vdots & \vdots \\ M_{\infty,M} & \alpha_{0,M} \end{bmatrix} \tag{4.24}$$

Entonces para cada condición en X se realiza una simulación para obtener las correspondientes salidas del sistema que se pueden escribir como

$$Y = \begin{bmatrix} y_{1,1} & y_{2,1} & \cdots & y_{M,1} \\ y_{1,2} & y_{2,2} & \cdots & y_{M,2} \\ \vdots & \vdots & \ddots & \vdots \\ y_{1,N} & y_{2,N} & \cdots & y_{M,N} \end{bmatrix} \tag{4.25}$$

Entonces se entrenará la red LSTM-NN para tratar de aproximar la relación entre las condiciones, usando 8 distintas condiciones para el entrenamiento, es decir, $M = 8$, para una entrada U conocida, y la salida Y obtenida a través de una simulación con ayuda de algún método de reducción válido solo alrededor de ese par de condiciones de X.

Posteriormente se hará la prueba de predicción con 3 condiciones distintas en X y por supuesto usando la misma U . Los resultados se muestran en la figura 4.6.

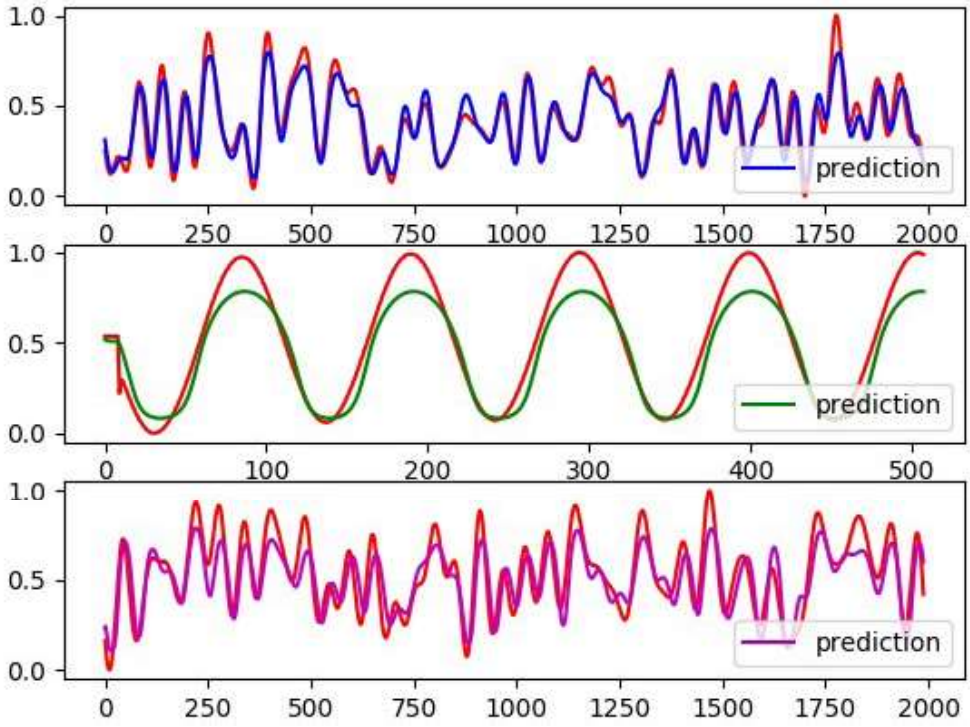


Figura 4.6: Predicción para el sistema aerodinámico con 3 condiciones distintas en X

Además de esto, se quizó realizar una comparación contra una MLP usando este mismo problema pero con uno de los casos mas complejos en el modelado de sistemas dinámicos, que corresponde a

$$\hat{y}(k) = f(\hat{y}(k-1), \ldots, \hat{y}(k-n_y), u(k))$$

es decir, cuando no existe una retroalimentación real y únicamente se pueden usar las entradas y la predicción anterior.

Los resultados de la predicción con la predicción anterior y la entrada de la forma $\hat{y}(k) = f(\hat{y}(k-1), \ldots, \hat{y}(k-n_y), u(k))$, se realizarón tomando en cuenta las mismas 3 condiciones de prueba de la simulacion anterior, los resultados se muestran en las figuras 4.7,4.8 y 4.9 con líneas punteadas.

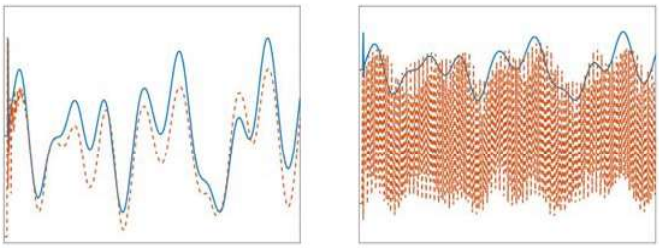


Figura 4.7: Predicción con línea punteada, LSTM-NN (Izquierda), MLP (derecha), condición 1

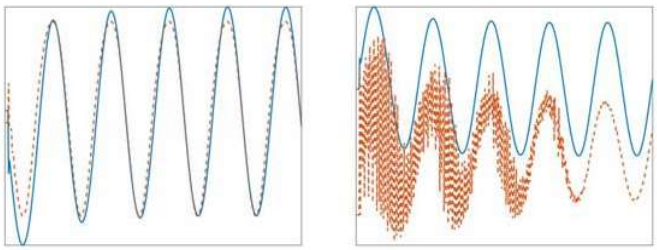


Figura 4.8: Predicción con línea punteada, LSTM-NN (Izquierda), MLP (derecha), condición 2

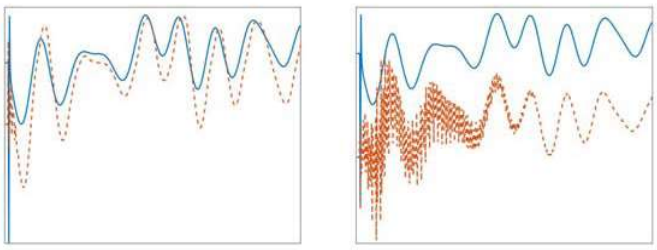


Figura 4.9: Predicción con línea punteada, LSTM-NN (Izquierda), MLP (derecha), condición 3

Los resultados obtenidos para la MLP distan mucho de ser favorables, incluso para la primera condición se pierde totalmente el resultado, a diferencia de lo obtenido con la LSTM-NN qué si bien debe ser mejorado para ser usado en condiciones reales, muestra un resultado favorable.

Capítulo 5

Nuevas RNN LSTM-GRU profundas para el modelado de sistemas dinámicos.

La arquitectura que ha sido usada hasta ahora mostrada en la figura 3.3, demostró varias ventajas en el marco actual para superar algunos de los retos que existen en modelado de sistemas dinámicos, sin embargo, esta arquitectura esta lejos de aprovechar el verdadero poder de las redes neuronales recurrentes y en especial de las LSTM-GRU las cuales cumplen su objetivo cuando se usan de manera recurrente en el tiempo, pero para lograr esto se necesita un cambio de paradigma y pensar en el conjunto de datos como una entidad que varia en el tiempo conforme pasa a traves de la red, es decir que cada pasada hacia adelante será un paso en el tiempo que produce una salida que corresponde a ese momento en el tiempo, este periodo puede representar un día, un mes, un segundo, un fotograma etc., Simplemente una relación secuencial de los datos que se representa de manera conjunta con el modelo. Usando este paradigma como marco de referencia se pueden diseñar distintos modelos capaces de aprovechar las bondades de las redes LSTM - GRU, tales modelos se presentan a continuación.

5.1. Estructura GRU profunda

Las redes neuronales recurrentes son estructuras con las cuales se pueden modelar sistemas dinámicos complejos en especial cuando su comportamiento esta intrínsecamente ligado a su evolución en el tiempo, y esto se logra retroalimentando la salida de la red en

el tiempo t a la entrada del mismo elemento de esa capa en el tiempo $t+1$. Se tiene que el modelo de predicción simple usado en el presente trabajo esta dado por

$$\hat{y}(k) = f(y(k-1), \ldots, y(k-n))$$

Si se despliega la red neuronal recurrente a lo largo del tiempo y a cada paso se proporciona una nueva entrada de la secuencia ademas de la salida previa h, donde la profundidad de la red estara dada por n, es decir el tamaño de la secuencia de retroalimentacion que se usará para realizar la predicción $\hat{y}(k)$, con la cual se hará la retropropagación del error, esta red tiene como diagrama la Figura 5.1 que se muestra a continuación.

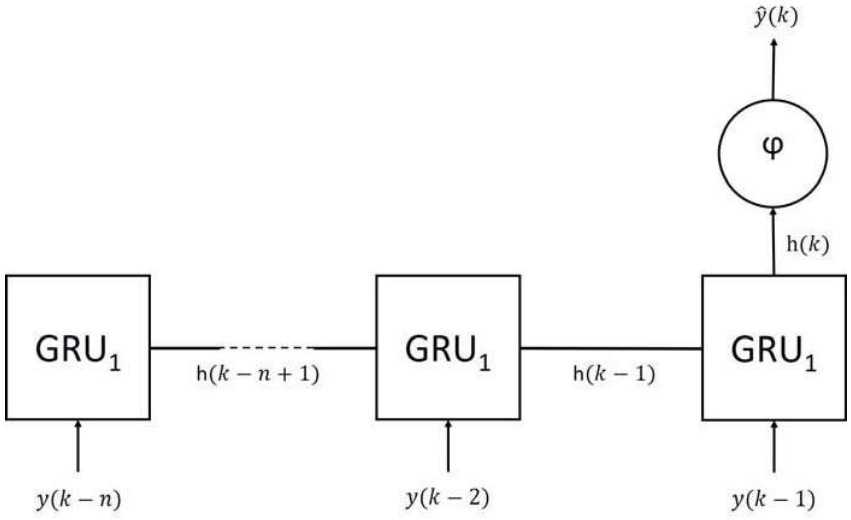


Figura 5.1: Estructura GRU profunda simple

5.1.1. Entrenamiento y generalizaión.

Partiendo del modelo mostrado en la figura 5.1 se puede observar que este se compone de una única celda recurrente, en este caso se analizará el caso de una celda GRU la cual como se sabe, se compone con las siguientes ecuaciones:

$$\begin{aligned} F(k) &= \sigma(W_F(k)[\hat{y}(k-1), x(k)]) \\ I(k) &= \sigma(W_I(k)[\hat{y}(k-1), x(k)]) \\ \tilde{y}\,(k) &= \varphi\,(W_X\,(k)\,[I\,(k) * \hat{y}\,(k-1)\,, x(k)]) \\ \hat{y}\,(k) &= \hat{y}\,(k-1) + F\,(k) * [\tilde{y}\,(k) - \hat{y}\,(k-1)] \end{aligned} \tag{5.1}$$

Donde $\sigma = sigmoide$, $\varphi =$ tanh, y $*$ es un producto de Hadamard.

Para entrenar la celda se usará BPTT considerando el error como

$$e(k) = \hat{y}\,(k) - y\,(k) \tag{5.2}$$

y la función de coste como

$$J(k) = \frac{1}{2}e^2 \tag{5.3}$$

El objetivo entonces es encontrar

$$\underset{P}{\arg\min}\ J(k) \text{ donde } P = [W_F, W_I, W_X] \tag{5.4}$$

Para esto se usa entonces la técnica del gradiente descendente que plantea que

$$P(k+1) = p(k) - \eta\frac{\partial J(k)}{\partial P} \tag{5.5}$$

Dado que en esta estructura la actualización de los pesos se realiza cuando se genera el error, es decir, al finar de la secuencia, entonces solo se considera para la actualización cuando $k = s$ con s igual al tamaño de la secuencia. Entonces tomando como base las ecuaciones generales de una RNN se tiene que

$$\frac{\partial J(s)}{\partial P} = \frac{\partial J(s)}{\partial e(s)}\,\frac{\partial e(s)}{\partial \hat{y}(s)} \sum_{j=0}^{s} \left(\prod_{i=j+1}^{s} \frac{\partial \hat{y}(i)}{\partial \hat{y}(i-1)} \right) \frac{\partial \hat{y}(i)}{\partial P} \tag{5.6}$$

Con motivo de realizar las derivadas parciales se separan los pesos en las ecuaciones de la GRU como se muestra a continuación

$$\begin{aligned} F(k) &= \sigma(W_{FY}(k)\hat{y}(k-1) + W_{FX}x(k)) \\ I(k) &= \sigma(W_{IY}(k)\hat{y}(k-1) + W_{IX}x(k)) \\ \tilde{y}\,(k) &= \varphi\,(W_{XY}\,(k)\,(I\,(k) * \hat{y}\,(k-1)) + W_{XX}x(k)) \\ \hat{y}\,(k) &= \hat{y}\,(k-1) + F\,(k) * [\tilde{y}\,(k) - \hat{y}\,(k-1)] \end{aligned} \tag{5.7}$$

Para entender que succede con las derivadas parciales se trabajará con un ejemplo, cuando $s = 2$, entonces se tiene que:

$$\frac{\partial J(2)}{\partial P} = \frac{\partial J(2)}{\partial e(2)} \frac{\partial e(2)}{\partial \hat{y}(2)} \left[\frac{\partial \hat{y}(2)}{\partial P} + \frac{\partial \hat{y}(2)}{\partial \hat{y}(1)} \frac{\partial \hat{y}(1)}{\partial P} + \frac{\partial \hat{y}(2)}{\partial \hat{y}(1)} \frac{\partial \hat{y}(1)}{\partial \hat{y}(0)} \frac{\partial \hat{y}(0)}{\partial P} \right] \tag{5.8}$$

Se sabe que:

$$\frac{\partial J(2)}{\partial e(2)} = e(2) \text{ y } \frac{\partial e(2)}{\partial \hat{y}(2)} = 1 \tag{5.9}$$

Recordando que $d(uv) = udv + vdu$, entonces si $u = F(k)$, $v_1 = \tilde{y}\,(k)$ y $v_2 = \hat{y}(k-1)$, entonces:

$$\frac{\partial \hat{y}(i)}{\partial \hat{y}(i-1)} = 1 + F(i) * \frac{\partial \tilde{y}(i)}{\partial \hat{y}(i-1)} + \tilde{y}(i) * \frac{\partial F(i)}{\partial \hat{y}(i-1)} - F(i) + \hat{y}(i-1) * \frac{\partial F(i)}{\partial \hat{y}(i-1)} \tag{5.10}$$

Evaluando la expresión anterior por partes se tiene que

$$\frac{\partial F(i)}{\partial \hat{y}(i-1)} = \sigma_F^{'}(i) W_{FY} \tag{5.11}$$

Ahora suponiendo $u = I(k)$ y $v = \hat{y}(k-1)$

$$\frac{\partial \tilde{y}(i)}{\partial \hat{y}(i-1)} = \phi^{'}(i) W_{XY}(i) \left[I(i) + \hat{y}(i-1) * \frac{\partial I(i)}{\partial \hat{y}(i-1)} \right] \tag{5.12}$$

Se obtiene

$$\frac{\partial I(i)}{\partial \hat{y}(i-1)} = \sigma_I^{'}(i) W_{IY} \tag{5.13}$$

Entonces se tiene que

$$\frac{\partial \tilde{y}(i)}{\partial \hat{y}(i-1)} = \phi^{'}(i) W_{XY}(i) \left[I(i) + \hat{y}(i-1) * \sigma_I^{'}(i) W_{IY} \right] \tag{5.14}$$

Sustituyendo en la expresion original se tiene que

$$\begin{aligned} \frac{\partial \hat{y}(i)}{\partial \hat{y}(i-1)} &= 1 + F(i) * \phi^{'}(i) W_{XY}(i) \left[I(i) + \hat{y}(i-1) * \sigma_I^{'}(i) W_{IY} \right] \\ &\quad + \tilde{y}(i) * \sigma_F^{'}(i) W_{FY} - F(i) + \hat{y}(i-1) * \sigma_F^{'}(i) W_{FY} \end{aligned} \tag{5.15}$$

Se hace $\sigma_F^{'} = \sigma_F^{'}(i) W_{FY}$, $\phi^{'} = \phi^{'}(i) W_{XY}(i)$ y $\sigma_I^{'} = \sigma_I^{'}(i) W_{IY}$ se tiene que

$$\begin{aligned} \frac{\partial \hat{y}(i)}{\partial \hat{y}(i-1)} &= 1 + F(i) * \left\{ \phi^{'} \left[I(i) + \hat{y}(i-1) * \sigma_I^{'}(i) W_{IY} \right] - 1 \right\} \\ &\quad + [\tilde{y}(i) + \hat{y}(i-1)] * \sigma_F^{'} \\ &= \delta(i) \end{aligned} \tag{5.16}$$

Ahora falta obtener las derivadas con respecto a los parametros a actualizar, es decir,

$$\frac{\partial \hat{y}(i)}{\partial P} = \frac{\partial}{\partial P} F(i) * (\tilde{y}(i) - \hat{y}(i-1)) \tag{5.17}$$

Donde $P = [W_{FY}, W_{FX}, W_{IY}, W_{IX}, W_{XY}, W_{XX}]$

Entonces se obtienen las derivadas con respecto a cada uno de ellos

$$\begin{aligned}
\frac{\partial \hat{y}(i)}{\partial W_{FY}} &= \sigma_F^{'}(i)\hat{y}(i-1) * [\tilde{y}(i) - \hat{y}(i-1)] \\
\frac{\partial \hat{y}(i)}{\partial W_{FX}} &= \sigma_F^{'}(i)x(i) * [\tilde{y}(i) - \hat{y}(i-1)] \\
\frac{\partial \hat{y}(i)}{\partial W_{IY}} &= \frac{\partial F(i)}{\partial W_{IY}} * \tilde{y}(i) \\
&= F(i)\phi^{'}(i)W_{XY}(i)\frac{\partial I(i)}{\partial W_{IY}} * \hat{y}(i-1) \\
&= F(i)\phi^{'}\sigma_I^{'}(i)\hat{y}(i-1) * \hat{y}(i-1) \\
\frac{\partial \hat{y}(i)}{\partial W_{IX}} &= F(i)\phi^{'}\sigma_I^{'}(i)x(i) * \hat{y}(i-1) \\
\frac{\partial \hat{y}(i)}{\partial W_{XY}} &= F(i)\phi^{'}(i)\left[I(i) * \hat{y}(i-1)\right] \\
\frac{\partial \hat{y}(i)}{\partial W_{XX}} &= F(i)\phi^{'}(i)x(i)
\end{aligned} \tag{5.18}$$

Dados estos gradientes entonces las reglas de actualización de pesos son

$$\begin{aligned}
W_{FY}(s+1) &= W_{FY}(s) - \eta\left[e(s)\sum_{j=0}^{s}\left(\prod_{i=j+1}^{s}\delta(i)\right)\sigma_F^{'}(i)\hat{y}(i-1) * [\tilde{y}(i) - \hat{y}(i-1)]\right] \\
W_{FX}(s+1) &= W_{FX}(s) - \eta\left[e(s)\sum_{j=0}^{s}\left(\prod_{i=j+1}^{s}\delta(i)\right)\sigma_F^{'}(i)x(i) * [\tilde{y}(i) - \hat{y}(i-1)]\right] \\
W_{IY}(s+1) &= W_{IY}(s) - \eta\left[e(s)\sum_{j=0}^{s}\left(\prod_{i=j+1}^{s}\delta(i)\right)F(i)\phi^{'}\sigma_I^{'}(i)\hat{y}(i-1) * \hat{y}(i-1)\right] \\
W_{IX}(s+1) &= W_{IX}(s) - \eta\left[e(s)\sum_{j=0}^{s}\left(\prod_{i=j+1}^{s}\delta(i)\right)F(i)\phi^{'}\sigma_I^{'}(i)x(i) * \hat{y}(i-1)\right] \\
W_{XY}(s+1) &= W_{XY}(s) - \eta\left[e(s)\sum_{j=0}^{s}\left(\prod_{i=j+1}^{s}\delta(i)\right)F(i)\phi^{'}(i)\left[I(i) * \hat{y}(i-1)\right]\right] \\
W_{XX}(s+1) &= W_{XX}(s) - \eta\left[e(s)\sum_{j=0}^{s}\left(\prod_{i=j+1}^{s}\delta(i)\right)F(i)\phi^{'}(i)x(i)\right]
\end{aligned} \tag{5.19}$$

Esta arquitectura de una sola celda es útil y en teoría es suficiente para resolver cualquier tipo de problema, sin embargo, como se ve en las redes multicapa aunque en

teoría es posible resolver cualquier tipo de problema con una sola capa de q neuronas en la practica resulta mucho mas conveniente poner a las redes MLP p capas con múltiples neuronas cada una de ellas. Pues bien, siguiendo este razonamiento con motivo de crear mejores resultados como se comprueba más adelante en las simulaciones, se propone un modelo generalizado que se muestra en la figura 5.2.

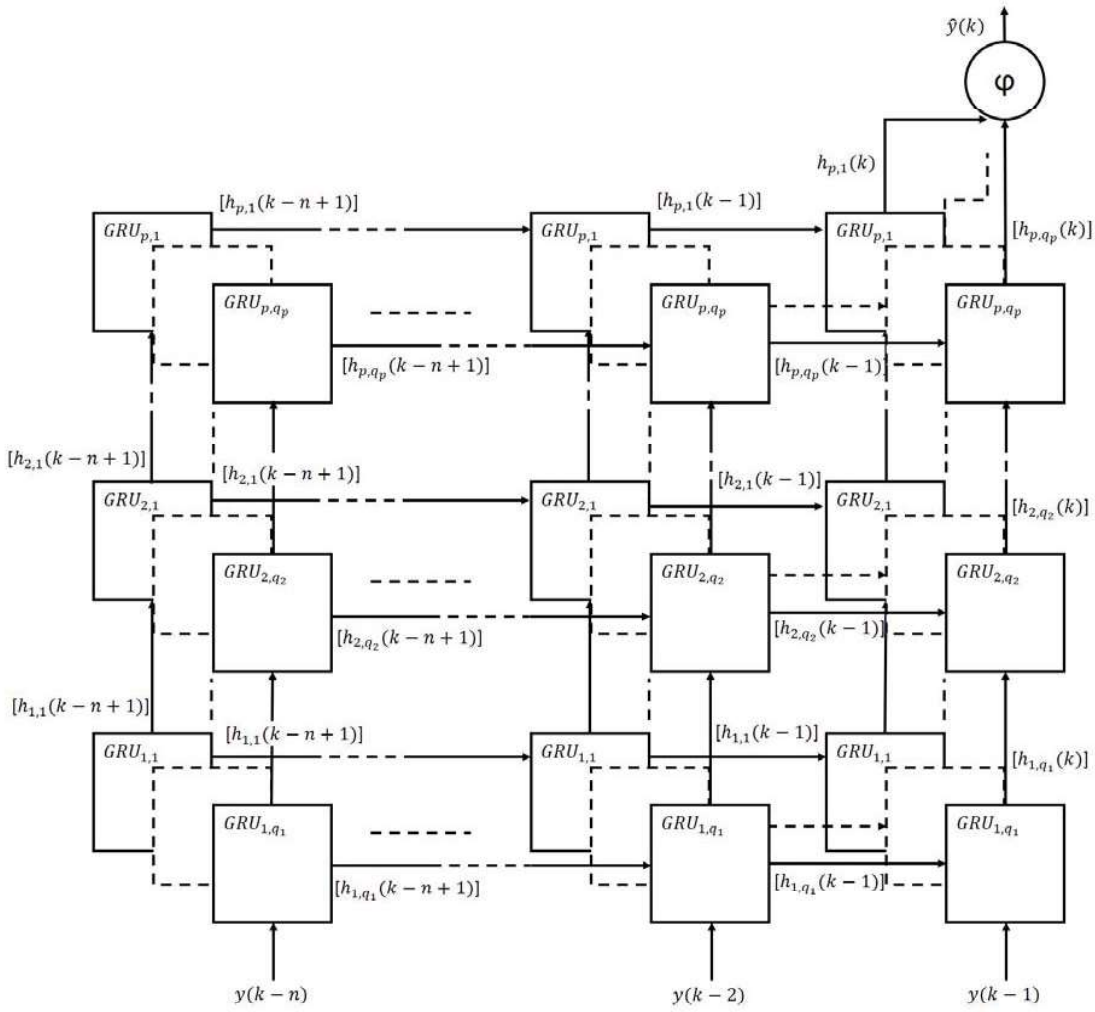


Figura 5.2: Estructura GRU profundo generalizado

Para el entrenamiento de este modelo habrá que hacer algunas modificaciones a las reglas de entrenamiento anteriores. Primero se renombran las variables de entrada y salida de cada celda GRU y se agrega el índice de capa.

$$
\begin{aligned}
F^j(k) &= \sigma(W^j_{FY}(k)x^j(k-1) + W_{FX}x^i(k)) \\
I^j(k) &= \sigma(W^j_{IY}(k)x^j(k-1) + W_{IX}x^i(k)) \\
\tilde{y}^j(k) &= \varphi\left(W^j_{XY}(k)\left(I^j(k) * x^j(k-1)\right) + W^j_{XX}x^i(k)\right) \\
x^j(k) &= x^j(k-1) + F^j(k) * \left[\tilde{y}^j(k) - x^j(k-1)\right]
\end{aligned}
\tag{5.20}
$$

Donde j es el índice de la capa actual, e i representa el índice de la capa anterior.

El objetivo general de la red se mantiene de tal forma que se actualizen los pesos tal que el error entre la salida del modelo y la salida de la planta se minimize, para esto se usa la misma función de costo dada por

$$J = \frac{1}{2}e^2 \tag{5.21}$$

donde $e(k) = \hat{y}(k) - y(k)$

Aquí también se usa la técnica del gradiente descendente que estará dada por

$$W(s+1) = W(s) - \eta\frac{\partial J(s)}{\partial P} \tag{5.22}$$

Para la salida de la última capa se tiene un perceptrón clásico cuya ecuación de salida esta dada por

$$\hat{y}(s) = \sigma(w(s)x^j(s)) \tag{5.23}$$

entonces sus pesos se actualizan de acuerdo a la regla anteriormente descrita donde

$$
\begin{aligned}
\frac{\partial J(s)}{\partial w} &= \frac{\partial J(s)}{\partial e(s)}\frac{\partial e(s)}{\partial \hat{y}(s)}\frac{\partial \hat{y}(s)}{\partial w} \\
&= e(s)\sigma^{'}x^j(s)
\end{aligned}
\tag{5.24}
$$

Ahora se necesita generalizar la regla de actualizacion de las celdas GRU para las capas intermedias de la red, y para visualizar mejor la propagación del error hacia atrás se analizan el caso de la capa j y de la capa i para posteriormente establecer las reglas de actualización de pesos de las GRU.

Suponiendo $s = 2$ y con respecto a la capa j se tiene.

$$\begin{aligned}\frac{\partial J(2)}{\partial P^j} &= \frac{\partial J(2)}{\partial e(2)}\frac{\partial e(2)}{\partial \hat{y}(2)}\frac{\partial \hat{y}(2)}{\partial x^j(2)}\left[\frac{\partial x^j(2)}{\partial P^j}+\frac{\partial x^j(2)}{\partial x^j(1)}\frac{\partial x^j(1)}{\partial P^j}+\frac{\partial x^j(2)}{\partial x^j(1)}\frac{\partial x^j(1)}{\partial x^j(0)}\frac{\partial x^j(0)}{\partial P^j}\right] \\ &= e(s)\sigma^{'}w\left[\frac{\partial x^j(2)}{\partial P^j}+\frac{\partial x^j(2)}{\partial x^j(1)}\frac{\partial x^j(1)}{\partial P^j}+\frac{\partial x^j(2)}{\partial x^j(1)}\frac{\partial x^j(1)}{\partial x^j(0)}\frac{\partial x^j(0)}{\partial P^j}\right] \\ &= e_j(s)\left[\frac{\partial x^j(2)}{\partial P^j}+\frac{\partial x^j(2)}{\partial x^j(1)}\frac{\partial x^j(1)}{\partial P^j}+\frac{\partial x^j(2)}{\partial x^j(1)}\frac{\partial x^j(1)}{\partial x^j(0)}\frac{\partial x^j(0)}{\partial P^j}\right]\end{aligned} \tag{5.25}$$

Donde $e_0(s) = \quad e(s)\sigma^{'}w$

Suponiendo $s = 2$ y con respecto a la capa j se tiene.

$$\frac{\partial J(2)}{\partial P^i} = \frac{\partial J(2)}{\partial e(2)}\frac{\partial e(2)}{\partial \hat{y}(2)}\frac{\partial \hat{y}(2)}{\partial x^j(2)}\frac{\partial x^j(2)}{\partial x^i(2)}\left[\frac{\partial x^i(2)}{\partial P^i}+\frac{\partial x^{i}(2)}{\partial x^i(1)}\frac{\partial x^{i}(1)}{\partial P^i}+\frac{\partial x^{i}(2)}{\partial x^{i}(1)}\frac{\partial x^{i}(1)}{\partial x^i(0)}\frac{\partial x^{i}(0)}{\partial P^{i}}\right] \tag{5.26}$$

Se tiene que

$$\begin{aligned}\frac{\partial x^j(k)}{\partial x^i(k)} &= \frac{\partial}{\partial x^i(k)}\left\{F^j(k) * \left[\tilde{y}^j(k) - x^j(k-1)\right]\right\} \\ \frac{\partial x^j(k)}{\partial x^i(k)} &= F^j(k) * \frac{\partial \tilde{y}^j(k)}{\partial x^i(k)} + \left[\tilde{y}^j(k) - x^j(k-1)\right]\frac{\partial F^j(k)}{\partial x^i(k)}\end{aligned} \tag{5.27}$$

Se obtiene

$$\begin{aligned}\frac{\partial \tilde{y}^j(k)}{\partial x^i(k)} &= \varphi^{'}\left[W^j_{XY}(k)\right]\frac{\partial I^j(k)}{\partial x^i(k)} * x^j(k-1) + W^j_{XX} \\ &= \varphi^{'}\left[W^j_{XY}(k)\right]\sigma^{'}W^j_{IX} * x^j(k-1) + W^j_{XX}\end{aligned} \tag{5.28}$$

Entonces se tiene que

$$\begin{aligned}\frac{\partial x^j(k)}{\partial x^i(k)} &= F^j(k) * \left[\varphi^{'}\left[W^j_{XY}(k)\right]\sigma^{'}W^j_{IX} * x^j(k-1) + W^j_{XX}\right] \\ &\quad + \left[\tilde{y}^j(k) - x^j(k-1)\right] * \sigma^{'}W^j_{FX} \\ &= x_{ji}\end{aligned} \tag{5.29}$$

Por lo tanto se tiene que

$$\begin{aligned}\frac{\partial J(2)}{\partial P^i} &= e(s)\sigma^{'} w x_{ji}\left[\frac{\partial x^i(2)}{\partial P^i}+\frac{\partial x^{i}(2)}{\partial x^i(1)}\frac{\partial x^{i}(1)}{\partial P^i}+\frac{\partial x^{i}(2)}{\partial x^{i}(1)}\frac{\partial x^{i}(1)}{\partial x^i(0)}\frac{\partial x^{i}(0)}{\partial P^{i}}\right] \\ &= e_j(s)x_{ji}\left[\frac{\partial x^i(2)}{\partial P^i}+\frac{\partial x^{i}(2)}{\partial x^i(1)}\frac{\partial x^{i}(1)}{\partial P^i}+\frac{\partial x^{i}(2)}{\partial x^{i}(1)}\frac{\partial x^{i}(1)}{\partial x^i(0)}\frac{\partial x^{i}(0)}{\partial P^{i}}\right] \\ &= e_i(s)\left[\frac{\partial x^i(2)}{\partial P^i}+\frac{\partial x^{i}(2)}{\partial x^i(1)}\frac{\partial x^{i}(1)}{\partial P^i}+\frac{\partial x^{i}(2)}{\partial x^{i}(1)}\frac{\partial x^{i}(1)}{\partial x^i(0)}\frac{\partial x^{i}(0)}{\partial P^{i}}\right]\end{aligned} \tag{5.30}$$

Así de manera general para una capa L oculta cualquiera y tomando en cuenta la contribucion del error de todos los nodos de la capa posterior se tiene que:

$$\frac{\partial J(s)}{\partial P^L}=\sum_{h=0}^{nL+1} e_h^L(s)\sum_{j=0}^{s}\left(\prod_{i=j+1}^{s}\frac{\partial x^L(i)}{\partial x^L(i-1)}\right)\frac{\partial x^L(i)}{\partial P^L} \tag{5.31}$$

Donde $e_h^L(s) = \ e_h^{L+1}(s)x_{ji}$, $e_h^{L+1}(s)$ es el error que se retropropaga de la capa $L+1$ del nodo h de esa capa y $nL+1$ representa el numero de nodos en la capa $L+1$.

Así, las reglas de actualizacion de pesos para cada nodo quedan descritas de la siguiente manera

$$\begin{aligned}W_{FY}^L(s+1) &= W_{FY}^L(s)-\eta\left[\begin{array}{c}\displaystyle\sum_{h=0}^{nL+1} e_h^L(s)\sum_{j=0}^{s}\left(\prod_{i=j+1}^{s}\delta^L(i)\right)\sigma_F^{'}(i)\hat{y}(i-1) \\ *\,[\tilde{y}(i)-\hat{y}(i-1)]\end{array}\right] \\ W_{FX}^L(s+1) &= W_{FX}^L(s)-\eta\left[\sum_{h=0}^{nL+1} e_h^L(s)\sum_{j=0}^{s}\left(\prod_{i=j+1}^{s}\delta^L(i)\right)\sigma_F^{'}(i)x(i)*[\tilde{y}(i)-\hat{y}(i-1)]\right]\end{aligned} \tag{5.32}$$

$$
\begin{aligned}
W_{IY}^{L}(s+1) &= W_{IY}^{L}(s) - \eta \left[\sum_{h=0}^{nL+1} e_h^L(s) \sum_{j=0}^{s} \left(\prod_{i=j+1}^{s} \delta^L(i) \right) F(i) \phi^{'} \sigma_I^{'}(i) \hat{y}(i-1) * \hat{y}(i-1) \right] \\
W_{IX}^{L}(s+1) &= W_{IX}^{L}(s) - \eta \left[\sum_{h=0}^{nL+1} e_h^L(s) \sum_{j=0}^{s} \left(\prod_{i=j+1}^{s} \delta^L(i) \right) F(i) \phi^{'} \sigma_I^{'}(i) x(i) * \hat{y}(i-1) \right] \\
W_{XY}^{L}(s+1) &= W_{XY}^{L}(s) - \eta \left[\sum_{h=0}^{nL+1} e_h^L(s) \sum_{j=0}^{s} \left(\prod_{i=j+1}^{s} \delta^L(i) \right) F(i) \phi^{'}(i) \left[I(i) * \hat{y}(i-1) \right] \right] \\
W_{XX}^{L}(s+1) &= W_{XX}^{L}(s) - \eta \left[\sum_{h=0}^{nL+1} e_h^L(s) \sum_{j=0}^{s} \left(\prod_{i=j+1}^{s} \delta^L(i) \right) F(i) \phi^{'}(i) x(i) \right]
\end{aligned}
$$

5.2. Estructura GRU profunda con reforzamiento

En este caso se propone una mejora en la estructura del modelo respecto al anterior y aunque es un modelo que produce múltiples salidas, en realidad estas salidas forman parte del proceso de aprendizaje ya que cada una de las salidas es en realidad el valor de la entrada al siguiente paso en el tiempo de la red, es decir, que la única salida que se predice en realidad es la última en el tiempo, esto ayuda a mejorar los resultados y permite manipulaciones interesantes.

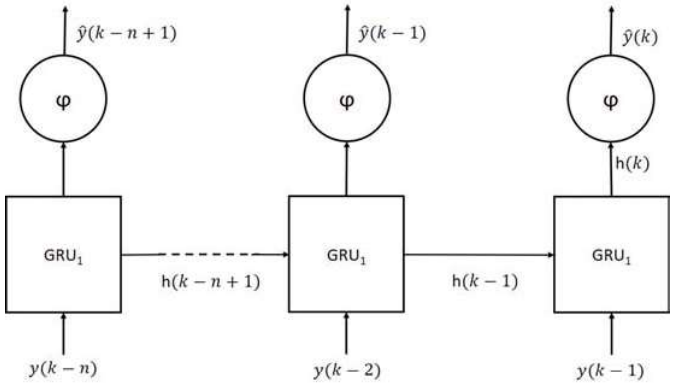


Figura 5.3: Estructura GRU profunda simple con refuerzo

5.2.1. Entrenamiento y generalizaión.

En el caso de esta estructura se puede observar un comportamiento muy similar al caso anterior con la diferencia que en este modelo se da una retroalimentacion con el valor de error en cada paso de tiempo k, es decir, que no solo la actualización de pesos se realiza en cada paso sino que tambien, se toman encuenta los pasos anteriores a cada actualizacion, de esta manera se tiene que el gradiente de la función de costo con respecto a los parámetros estará dado por.

$$\frac{\partial J(k)}{\partial P} = \sum_{l=0}^{k} \frac{\partial J(l)}{\partial e(l)} \frac{\partial e(l)}{\partial \hat{y}(l)} \sum_{j=0}^{l} \left(\prod_{i=j+1}^{l} \frac{\partial \hat{y}(i)}{\partial \hat{y}(i-1)} \right) \frac{\partial \hat{y}(i)}{\partial P} \tag{5.33}$$

Y con este cambio la actualizacion de los pesos estará dada por

$$\begin{aligned}
W_{FY}(k+1) &= W_{FY}(k) - \eta \left[\sum_{l=0}^{k} e(l) \sum_{j=0}^{l} \left(\prod_{i=j+1}^{l} \delta(i) \right) \sigma_F^{'}(i) \hat{y}(i-1) * [\tilde{y}(i) - \hat{y}(i-1)] \right] \\
W_{FX}(k+1) &= W_{FX}(k) - \eta \left[\sum_{l=0}^{k} e(l) \sum_{j=0}^{l} \left(\prod_{i=j+1}^{l} \delta(i) \right) \sigma_F^{'}(i) x(i) * [\tilde{y}(i) - \hat{y}(i-1)] \right] \\
W_{IY}(k+1) &= W_{IY}(k) - \eta \left[\sum_{l=0}^{k} e(l) \sum_{j=0}^{l} \left(\prod_{i=j+1}^{l} \delta(i) \right) F(i) \phi^{'} \sigma_I^{'}(i) \hat{y}(i-1) * \hat{y}(i-1) \right] \\
W_{IX}(k+1) &= W_{IX}(k) - \eta \left[\sum_{l=0}^{k} e(l) \sum_{j=0}^{l} \left(\prod_{i=j+1}^{l} \delta(i) \right) F(i) \phi^{'} \sigma_I^{'}(i) x(i) * \hat{y}(i-1) \right] \\
W_{XY}(k+1) &= W_{XY}(k) - \eta \left[\sum_{l=0}^{k} e(l) \sum_{j=0}^{l} \left(\prod_{i=j+1}^{l} \delta(i) \right) F(i) \phi^{'}(i) \left[I(i) * \hat{y}(i-1) \right] \right] \\
W_{XX}(k+1) &= W_{XX}(k) - \eta \left[\sum_{l=0}^{k} e(l) \sum_{j=0}^{l} \left(\prod_{i=j+1}^{l} \delta(i) \right) F(i) \phi^{'}(i) x(i) \right]
\end{aligned} \tag{5.34}$$

Al igual que en caso de la estructura GRU profunda con reforzamiento es conveniente generalizar la arquitectura de este modelo y así poder obtener mejores resultados, la siguiente figura muestra un diagrama que ayuda a visualizar esta configuración.

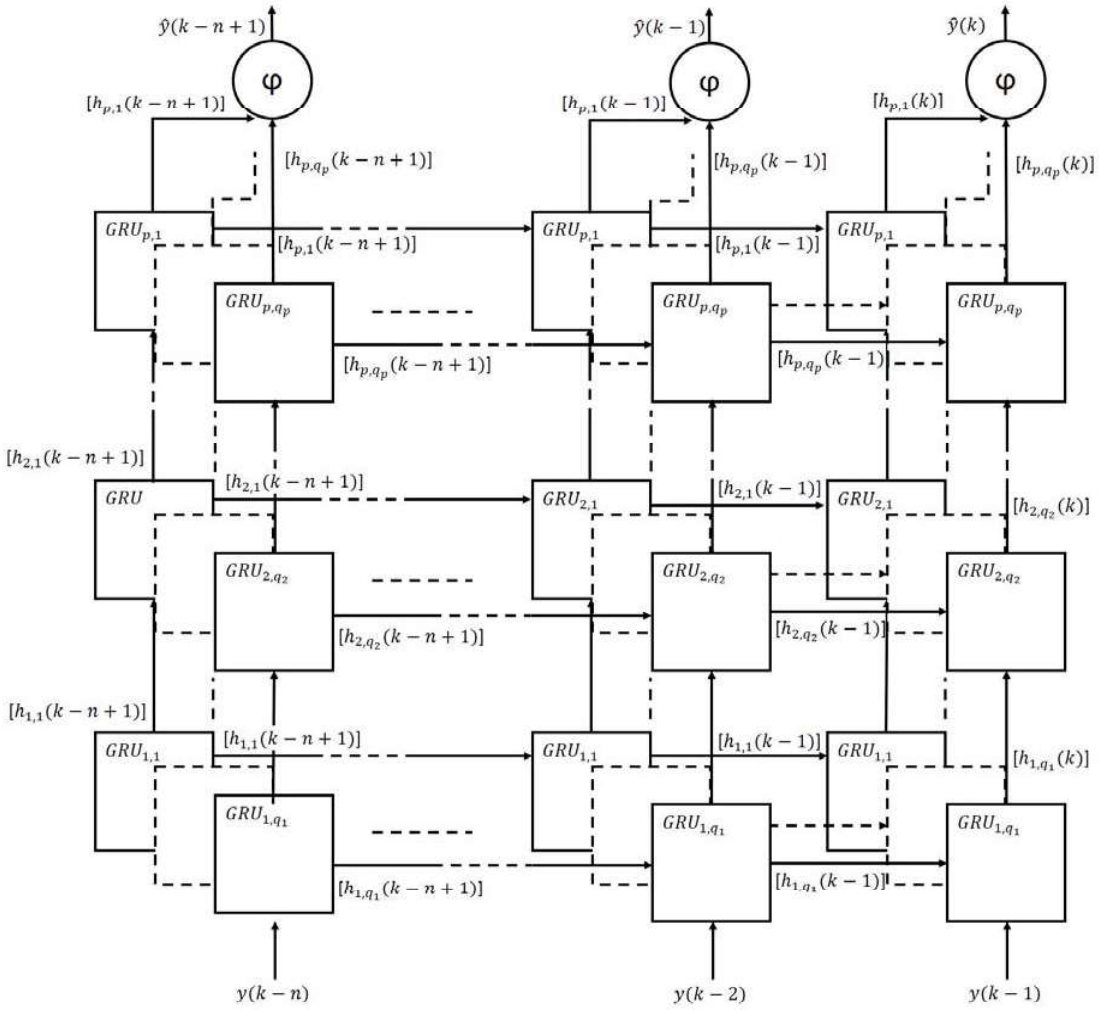


Figura 5.4: Estructura GRU profunda general con refuerzo

En el caso de este modelo la actualizacion de los pesos estara dada por

$$
\begin{aligned}
W^L_{FY}(k+1) &= W^L_{FY}(k) - \eta \left[\sum_{h=0}^{nL+1} \sum_{l=0}^{k} e^L_h(l) \sum_{j=0}^{l} \left(\prod_{i=j+1}^{l} \delta^L(i) \right) \sigma'_F(i) \hat{y}(i-1) * [\tilde{y}(i) - \hat{y}(i-1)] \right] \qquad (5.35) \\
W^L_{FX}(k+1) &= W^L_{FX}(k) - \eta \left[\sum_{h=0}^{nL+1} \sum_{l=0}^{k} e^L_h(l) \sum_{j=0}^{l} \left(\prod_{i=j+1}^{l} \delta^L(i) \right) \sigma'_F(i) x(i) * [\tilde{y}(i) - \hat{y}(i-1)] \right] \\
W^L_{IY}(k+1) &= W^L_{IY}(k) - \eta \left[\sum_{h=0}^{nL+1} \sum_{l=0}^{k} e^L_h(l) \sum_{j=0}^{l} \left(\prod_{i=j+1}^{l} \delta^L(i) \right) F(i) \phi' \sigma'_I(i) \hat{y}(i-1) * \hat{y}(i-1) \right] \\
W^L_{IX}(k+1) &= W^L_{IX}(k) - \eta \left[\sum_{h=0}^{nL+1} \sum_{l=0}^{k} e^L_h(l) \sum_{j=0}^{l} \left(\prod_{i=j+1}^{l} \delta^L(i) \right) F(i) \phi' \sigma'_I(i) x(i) * \hat{y}(i-1) \right] \\
W^L_{XY}(k+1) &= W^L_{XY}(k) - \eta \left[\sum_{h=0}^{nL+1} \sum_{l=0}^{k} e^L_h(l) \sum_{j=0}^{l} \left(\prod_{i=j+1}^{l} \delta^L(i) \right) F(i) \phi'(i) \left[I(i) * \hat{y}(i-1) \right] \right] \\
W^L_{XX}(k+1) &= W^L_{XX}(k) - \eta \left[\sum_{h=0}^{nL+1} \sum_{l=0}^{k} e^L_h(l) \sum_{j=0}^{l} \left(\prod_{i=j+1}^{l} \delta^L(i) \right) F(i) \phi'(i) x(i) \right]
\end{aligned}
$$

5.3. Estructura secuencia a secuencia profunda

Esta estructura resulta de la fusión de las dos estructuras anteriores aprovechando sus ventajas, consta de un bloque sin reforzamiento como bloque receptor o bloque de entrada y un bloque de salida con reforzamiento, en este modelo dependiendo de cómo se den las entradas y salidas puede tener múltiples propósitos siempre sacando el máximo provecho de las recurrencias.

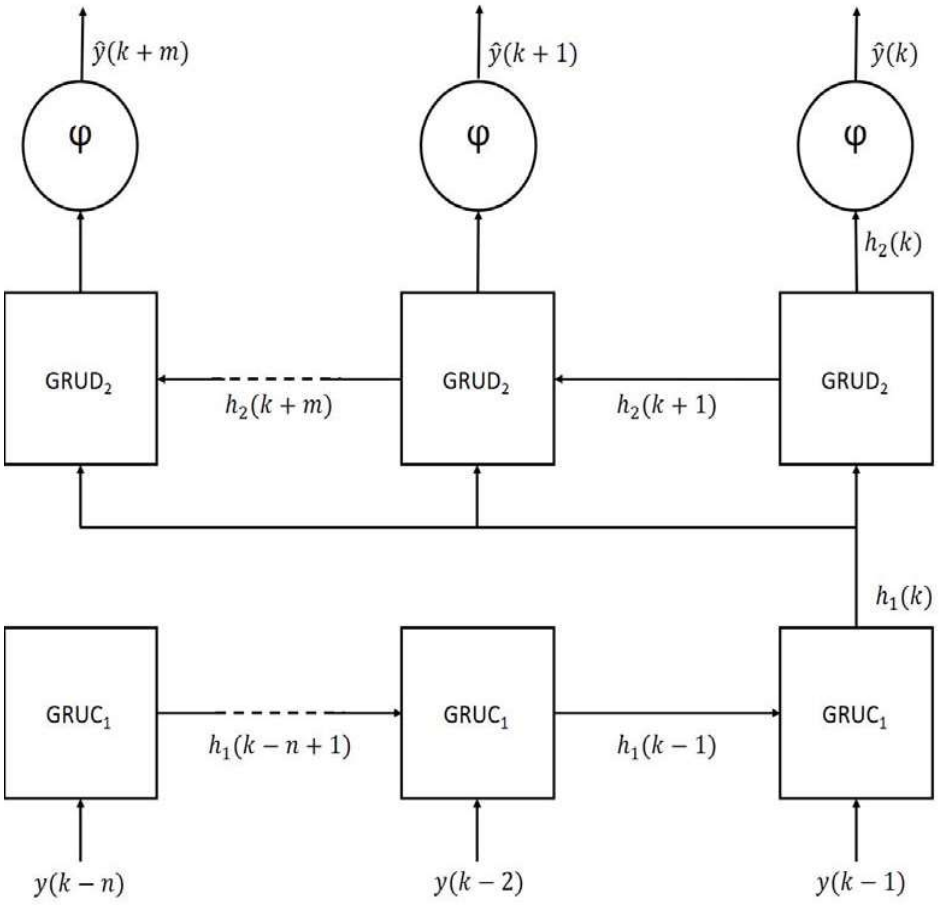


Figura 5.5: Estructura secuencia a secuencia profunda simple

5.3.1. Entrenamiento y generalizaión.

Para el entrenamiento de este modelo se reúsan las ecuaciones de las estructuras anteriores, cada una de ellas en los bloques correspondientes y en los tiempos s y k respectivamente, de esta forma aunque el modelo generalizado resulta algo complejo, una visualizacion de este se muestra en la siguiente figura.

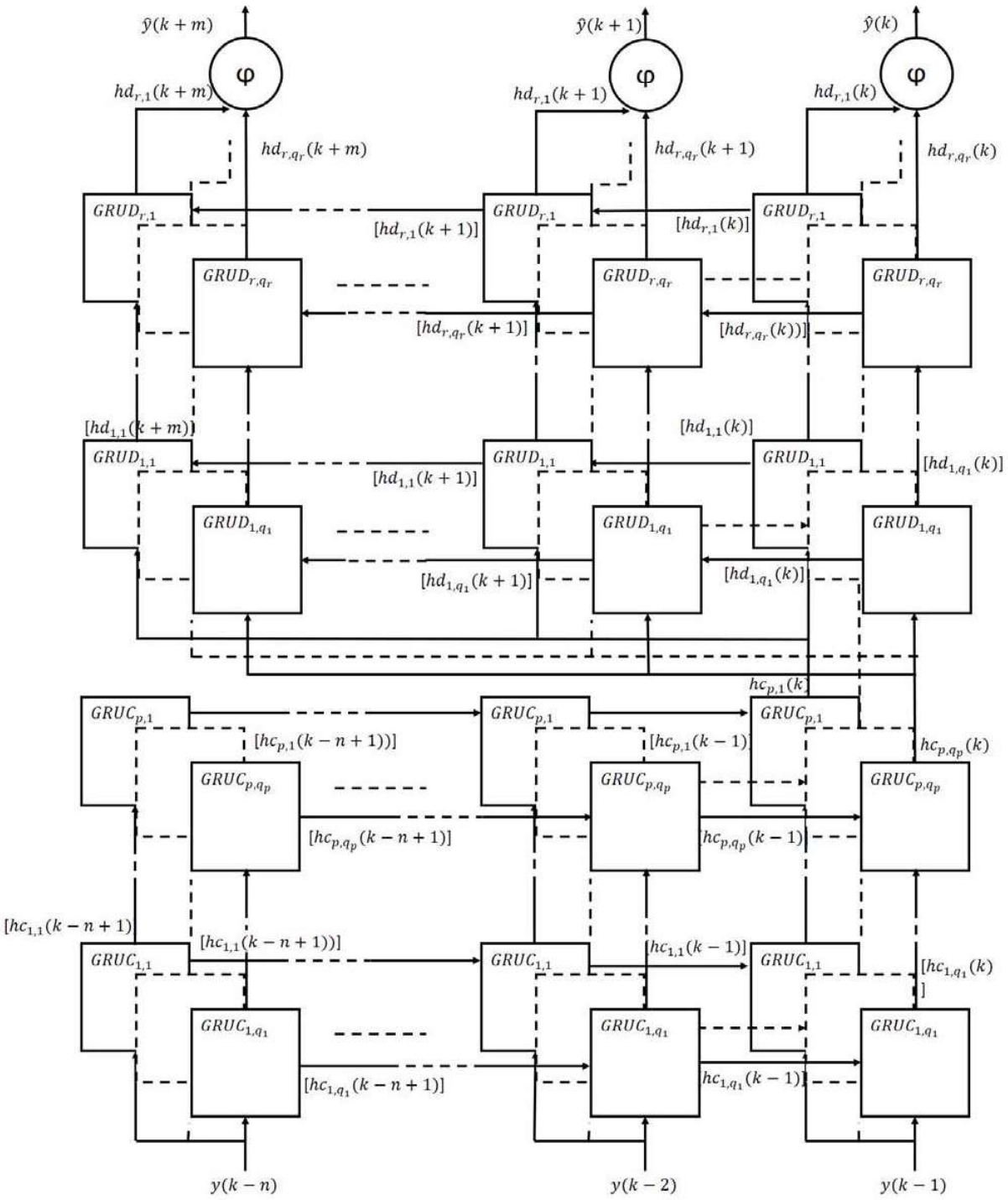


Figura 5.6: Estructura secuencia a secuencia profunda generalizada

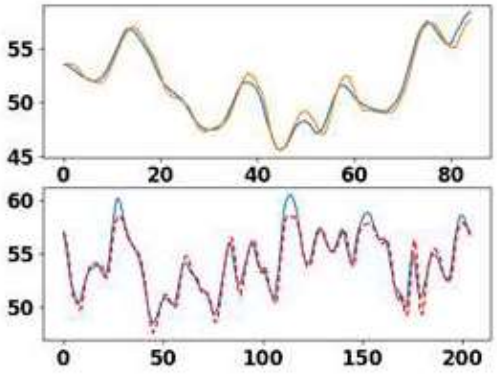
5.4. Simulación para un horno de gas

5.4.1. Estructura GRU profunda

Tomando el conjunto de datos del horno de gas de Box, Jenkins, y Reinsel, 1994 [15], usado a menudo para el análisis de series de tiempo y en el capítulo 3 para demostrar la mejora de las redes neuronales con el uso de las celdas LSTM frente a las redes MLP, ahora se usará para comparar el modelo propuesto en los capítulos anteriores al que se denominó LSTM-NN.

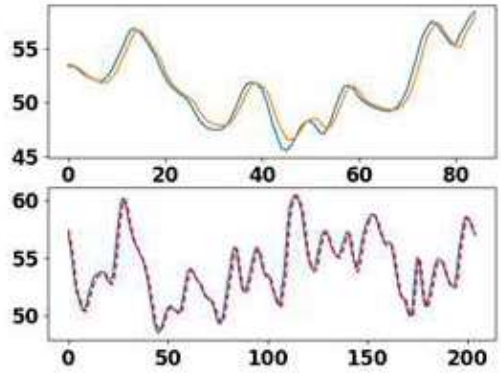
A continuación se muestra el mejor resultado obtenido con 150 epocas y una sola celda LSTM para ambas arquitecturas, usando el 30 % de los datos para el entrenamiento y con un modelo $\hat{y}(k) = f(y(k-1), \ldots, y(k-3)$, es decir, que la red tendrá una profundidad de 3.

LSTM-NN



MSE de entrenamiento:0.001
MSE de prueba:0.002

GRU profunda



MSE de entrenamiento:0.009
MSE de prueba:0.009

Se puede observar en la comparativa anterior que aunque el error promedio de la GRU profunda es un poco mayor que el error de la otra arquitectura, el modelo representa mucho mejor el comportamiento del sistema.

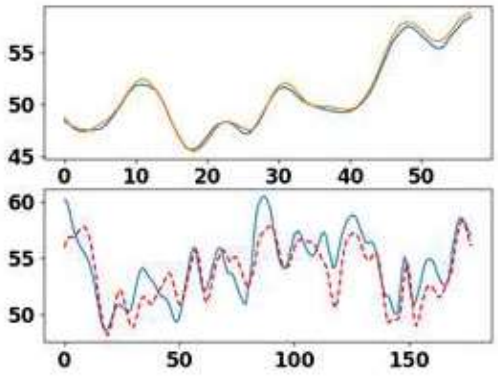
5.4.2. Estructura GRU profunda con reforzamiento

Continuando con los resultados usando los datos del horno de gas, como se menciono anteriormente la estructura GRU profunda ya es en sí superior al modelo LSTM-NN para representar el comportamiento del sistema, sin embargo, el problema con las redes neuronales tradicionales, basadas en el perceptrón multicapa es que cuando la dependencia

entre los datos de la secuencia esta separada por un número significativo de pasos entonces se experimenta el problema del gradiente del cual se hablo anteriormente, que ocurre cuando valores muy pequeños del gradiente se multiplican muchas veces a través de múltiples pasos en el tiempo y hace que tienda asintóticamente a cero, lo que implica que los pesos de esas capas no cambien es decir, que la red deja de aprender, el modelo LSTM-NN consigue mejores resultados en este sentido.

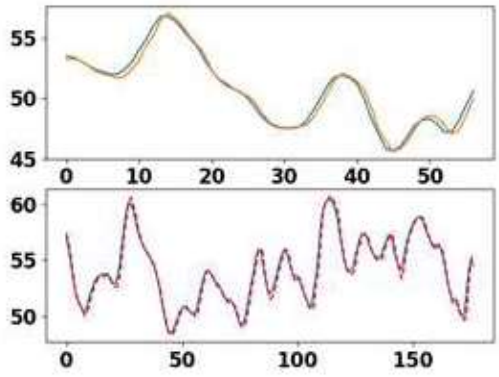
Sin embargo, aún así presenta limitantes por ejemplo, en este ejemplo del horno de gas, cuando se quieren mejorar los resultados y se trabaja con los modelos $\hat{y}(k) = f(y(k-1), \ldots, y(k-7)$ y $\hat{y}(k) = f(y(k-1), \ldots, y(k-15)$, en primer lugar se tiene que crecer mucho el tamaño de la red de una sola celda a dos capas de 200 y los resultados del MSE mejoran de 0.0027 a 0.0021 por lo tanto la mejora es casi inexistente. Pues bien esta arquitectura recurrente propuesta resuelve este problema permitiendo crecer indefinidamente la profundidad de la red.

LSTM-NN



Modelo: 2 capas de 200 celdas
Número de parámetros:505801
Tiempo de entrenamiento:16s
MSE de entrenamiento:0.0005
MSE de prueba:0.0136

GRU profunda con reforzamiento



Modelo: 10 celdas
Número de parámetros:491
Tiempo de entrenamiento:35s
MSE de entrenamiento:0.0004
MSE de prueba:0.0007

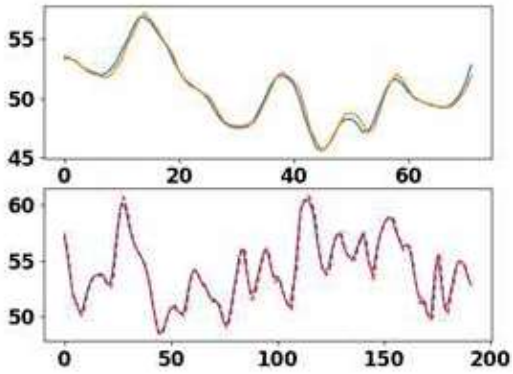
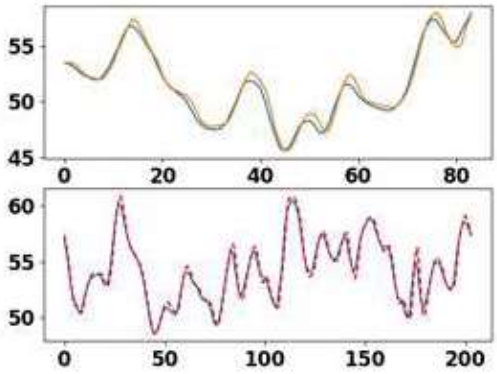
En la comparativa anterior se muestra una comparación entre el modelo LSTM-NN contra la estructura GRU profunda con reforzamiento para $\hat{y}(k) = f(y(k-1), \ldots, y(k-30)$, en la figura se muestra claramente que el modelo LSTM-NN tiene problemas para ese tamaño de secuencia y el aprendizaje se ve afectado mientras que la estructura GRU profunda con reforzamiento no solo soporta este tamaño de secuencia sino que además muestra un error significativamente menor incluso frente a la estructura GRU profunda sin reforzamiento. En realidad en las pruebas la profundidad puede crecer no solo a

30 sino indefinidamente, sin embargo, esto no siempre representa de mejor manera el comportamiento del sistema por lo que sigue siendo un parámetro que debe ser ajustado de acuerdo al problema a resolver.

Además de resolver el problema de secuencias muy largas, tambien sobresale el hecho de que en lugar de utilizar 400 celdas como en el caso de la LSTM-NN este usa solo 10, reduciendo notablemente la cantidad de parámetros que deben ser ajustados y sacando el máximo provecho de la recurrencia.

5.4.3. Estructura secuencia a secuencia profunda

Tomando el conjunto de datos del horno de gas de Box, Jenkins, y Reinsel, 1994 [15], a continuación se muestra el mejor resultado obtenido con un modelo $\hat{y}(k) = f(y(k-1), \ldots, y(k-15)$, es decir, que la red tendrá una profundidad de 15 y se compara contra la estructura GRU profunda con reforzamiento que es el que mejores resultados había obtenido.

GRU profunda con reforzamiento	secuencia a secuencia profunda
	
Modelo: 10 celdas	Modelo: 10 y 10
Número de parámetros: 491	Número de parámetros:1331
Tiempo de entrenamiento:25s	Tiempo de entrenamiento:24s
MSE de entrenamiento:0.0005	MSE de entrenamiento:0.00041
MSE de prueba:0.0008	MSE de prueba:0.00041

En la comparativa anterior se puede observar que si bien la estructura secuencia a secuencia ocupa lógicamente mas parámetros que la estructura GRU profunda con reforzamiento, el tiempo de entrenamiento es similar y además de obtener un error mas pequeño se observa que el resultado del error de entrenamiento y de prueba son en extremo similares lo que indica que el comportamiento del sistema fue mejor aprendido.

Prediccion de s pasos adelante

Se ha demostrado la superioridad de este modelo frente a las demás propuestas cuando se trata de predecir $\hat{y}(k) = f(y(k-1), \ldots, y(k-n)$, sin embargo, uno de los problemas que suponen una dificultad superior y que, sin embargo, son de extrema utilidad en aplicaciones reales es cuando se trata de predecir $\hat{y}(k+s) = f(y(k-1), \ldots, y(k-n)$, lo cual con las arquitecturas anteriores tiene un error bastante grande cuando $s = 3$ como se muestra en la siguiente figura.

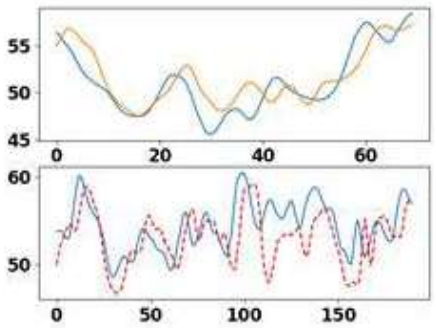
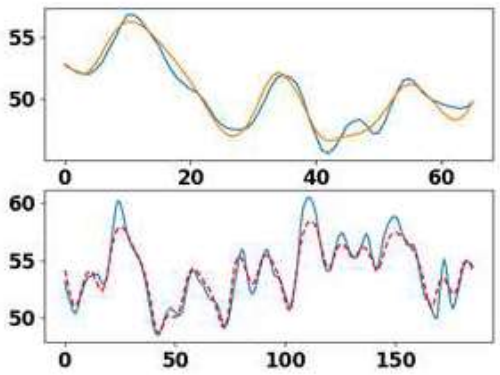
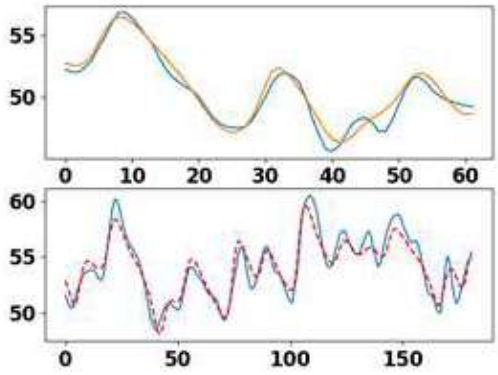


Figura 5.7: Modelo LSTM-NN, s=3, MSE=0.08

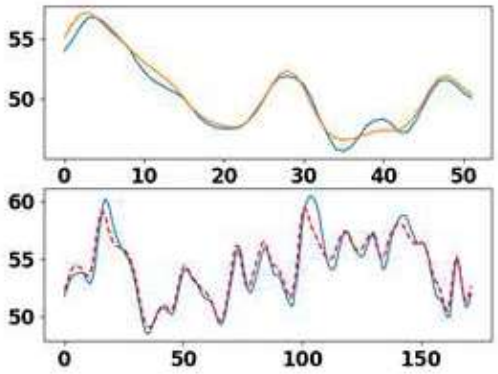
Sin embargo, con la estructura secuencia a secuencia propuesta se resuelve este problema para valores grandes de s, como se muestra en la siguientes figura.



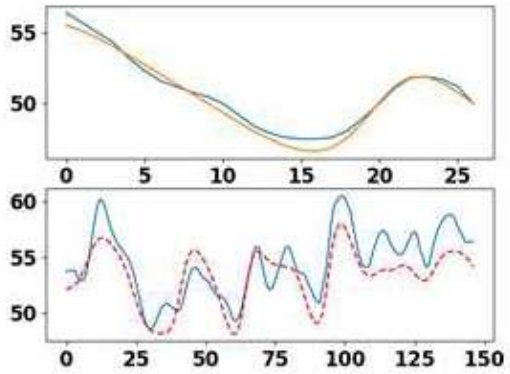
s=3, MSE=0.02



s=5, MSE=0.03



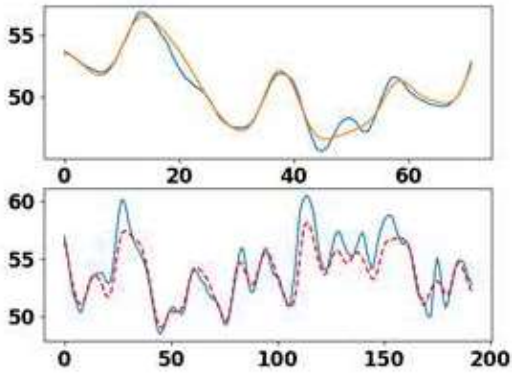
s=10, MSE=0.04



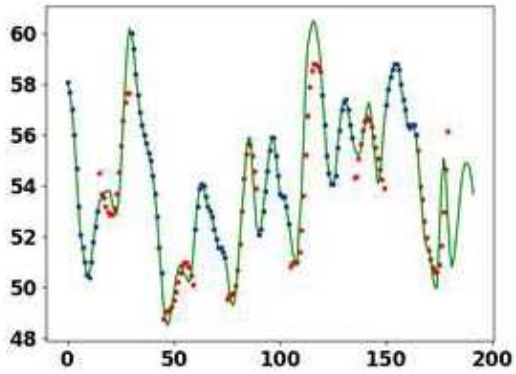
s=15, MSE=0.09

Predicción de múltiples pasos

Aquí es donde se demustra el potencial superior de esta arquitectura secuencia a secuencia, y esque otro de los problemas en extremo complejos es tratar de predecir los s pasos siguientes con los n pasos anteriores, es decir, $\hat{y}(k+1), \ldots, \hat{y}(k+1) = f(y(k-1), \ldots, y(k-n))$, el resultado se muestra a continuación.



Predicción 1 paso



Predicción múltiples pasos

En la gráfica anterior se observa el resultado de una predicción multiple con $n = 15$ y $s = 15$, a la izquerda se tiene el resultado continuo de los datos de entrenamiento (arriba izquierda) y prueba(abajo izquierda), con un error de entrenamiento de 0.0013 y de prueba de 0.005. En la imagen de la derecha se muestra un ejemplo con puntos azules los datos reales y con color rojo los datos predichos.

Capítulo 6

Predicción de sismos con RNN profundas

Hace mucho tiempo que los científicos han trabajado profundamente la cuestión del pronóstico de terremotos. Algunos esquemas de predicción se basaron en el análisis de anomalías geofísicas que se encontraron relacionadas con terremotos inminentes, como el cambio del nivel del agua en los pozos[44][45], anomalías electromagnéticas [46], anomalías térmicas [47], anomalías de radón, anomalías hidrológicas [48], etc. Se encontraron también otros signos precursores de los terremotos al analizar sus propiedades estadísticas y dinámicas mediante el uso de varios enfoques metodológicos como fractales [49], patrones de informática [50], [51], estadísticas no extensivas [52], gráficos de visibilidad [53], etc. La gran variedad de enfoques utilizados para tratar la predicción del terremoto es, *per se*, una indicación de cuán complejo es el problema.

Muchos sistemas complejos han encontrado en las redes neuronales artificiales (ANN) una forma de ser modelados eficientemente. Las redes neuronales se utilizaron con éxito para resolver problemas complejos de reconocimiento y clasificación de patrones en diferentes campos como el reconocimiento de imágenes y objetos, reconocimiento de voz, robótica y visión por computadora, procesamiento de señales, ingeniería biomédica, etc.

En el contexto de la predicción de terremotos, las redes neuronales mostraron ser una herramienta eficiente de predicción para predecir terremotos de laboratorio [54]. HojjatAdeli y AshifPanakkat [55] utilizan una red neuronal probabilística para predecir la magnitud del mayor terremoto en un período de tiempo futuro predefinido en una región sísmica. K. M. Asim [56] aplicó cuatro enfoques de redes neuronales, incluida la red neuronal de reconocimiento de patrones, la red neuronal recurrente, el bosque aleatorio y

el clasificador de conjunto de impulso de programación lineal, por separado para modelar las relaciones entre los parámetros sísmicos calculados y los sucesos futuros de terremotos. Varios otros trabajos trataron la predicción de varios parámetros de terremotos mediante el uso de redes neuronales artificiales [57][58][59][60]. También se han dedicado algunos esfuerzos, especialmente durante la última década, al uso de redes neuronales profundas para la predicción de terremotos [61][62][63][64][65][66].

Una clase de redes neuronales está representada por las Redes Neuronales Recurrentes (RNN), que son una herramienta muy importante en la predicción de problemas, porque están especialmente diseñadas para manejar tareas que usan información "dinámicamente"; mientras que las redes neuronales tradicionales, también conocidas como redes neuronales de retroalimentación", se caracterizan por entradas y salidas independientes entre sí, lo que permite, por lo tanto, solo el modelado de relaciones estáticas. Como herramienta de pronóstico, se utilizaron RNN en diferentes campos de investigación, como determinar el precio de la energía eléctrica [67], o modelar dinámicas no lineales extremadamente complejas en varios fenómenos geofísicos para predecir, por ejemplo, trayectorias de huracanes [68], velocidad del viento [69], intensidad de ciclón tropical [70] e incluso irradiancia solar [71].

Para muchas series de tiempo, los valores de datos actuales dependen de sus datos pasados, como oraciones y ondas de sonido. Las redes neuronales recurrentes (RNN) tienen propiedades similares. La propagación hacia atrás a través del tiempo (BPTT) es un método de entrenamiento efectivo para RNN. Sin embargo, el entrenamiento BPTT tiene muchos problemas, como la pérdida de gradiente y la convergencia lenta. La red de memoria a largo plazo (LSTM) en Hochreiter [29][72] es RNN muy popular en los últimos años.

La primera parte de este capítulo explora la posibilidad de usar la LSTM–NN, para predecir la magnitud máxima de los terremotos en el corto plazo de un día. Prácticamente responderá a la pregunta "¿Cuál será la magnitud máxima de los terremotos que ocurrirán mañana?", para posteriormente usar la estructura secuencia a secuencia profunda para predecir la magnitud del próximo evento, así como obtener una aproximación de donde sucederá y cuál será su profundidad.

Para explorar tal posibilidad, se utiliza el catálogo sísmico de Italia desde 1995 hasta 2018. La magnitud mínima es 1.5, que representa la magnitud de completitud del catálogo, y la profundidad máxima de todos los eventos es 60 km, lo que limita el análisis a la sismicidad de la corteza, el catalogo incluye también latitud y longitud de cada evento.

6.1. RNN LSTM-NN profunda para predecir la magnitud máxima diaria de terremotos

Para cada observación o evento sísmico, se guardan variables como ubicación geográfica, latitud y longitud, fecha y hora, magnitud y profundidad del evento. Se propone entonces tratar cada una de las observaciones o eventos sísmicos como una función $E(P(k))$ que representa un conjunto de muestras a lo largo del tiempo donde $k = 1, ..., N$ donde N es el número total de muestras tomadas en ese período de tiempo, y $P(k)$ es un vector de parámetros derivados de esas mediciones.

El objetivo es encontrar relaciones existentes entre eventos pasados y futuros con el fin de predecir la magnitud de los próximos eventos con un margen de error aceptable utilizando la información actual. Dada la naturaleza aparentemente aleatoria del problema, es difícil conocer las relaciones entre las diferentes variables y sus derivados y en qué medida influyen en la magnitud de un evento futuro, por lo tanto, el siguiente modelo básico de comportamiento se debe aprender en el modelo neuronal propuesto.

$$y(k) = N(y(k-1), \ldots, y(k-n_y))$$

Donde $y(k)$ representa la magnitud de un evento en el tiempo k. En la Figura 6.1 se muestra una representación gráfica del modelo para el entrenamiento y la predicción usando la estructura LSTM-NN.

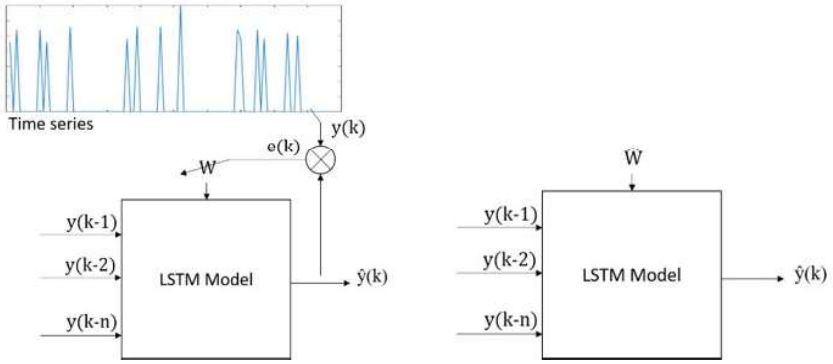


Figura 6.1: Representación gráfica del modelo para el entrenamiento (izquierda) y la predicción (derecha)

La distancia temporal que separa los eventos sísmicos observados, conocida como la distancia entre eventos es variable, lo que significa que algunos eventos podrían estar separados por minutos, mientras que otros están separados por horas o incluso días, aquí se considera una serie de tiempo que va desde el primer evento registrado hasta el último en intervalos de una hora y cuyo valor es la magnitud del evento en ese momento k, en caso de tener más de un evento registrado en la misma hora, el que tenga la mayor magnitud se toma como magnitud representativa

En la Figura 6.2 se observa que algunos eventos tienen una magnitud 0, lo que indica, por ejemplo, que no hubo terremotos o que el equipo estaba apagado por alguna razón, por lo que estos valores no son parte del comportamiento del sistema y realmente dificultan la capacitación de la red, por lo que como la mayoría de los valores están entre 2 y 6, los que están fuera de este rango pueden verse como capas externas y no se aprenderán.

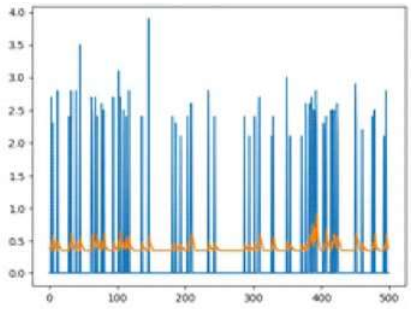


Figura 6.2: Entrenamiento de la serie temporal de magnitud del terremoto por hora

Como es posible ver en el ejemplo, la serie temporal de magnitud no solo parece comportarse de manera muy aleatoria, sino que también parece ser muy intermitente, y esto hace que la predicción sea extremadamente complicada. En nuestro estudio, el 70 % de los datos se utilizaron para el entrenamiento modelo y el 30 % restante se utilizó para evaluar la predicción.

6.1.1. Método de retención de orden cero

Por lo tanto, es posible tomar el modelo de retención de orden cero de la serie temporal generando una nueva función que, sin embargo, respeta el comportamiento original y mantiene los picos iguales, es decir, la predicción será válida como se muestra en la figura 6.3.

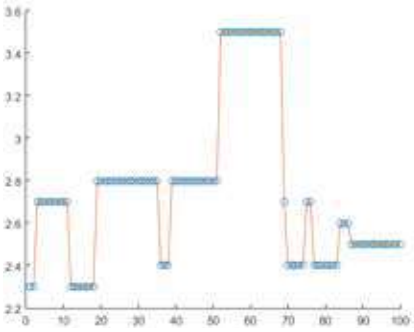


Figura 6.3: Serie temporal de magnitud del terremoto por hora sin ceros

Con estos cambios y utilizando la arquitectura propuesta anteriormente con $n = 5$, que fue el retraso mínimo para obtener un buen resultado, con el 10 % de los datos utilizados para el entrenamiento y 2 épocas, obtenemos un error de entrenamiento de 0.002 y un error de predicción de 0.003 que ya es un buen resultado, cuyos gráficos se muestran en la figura 6.4.

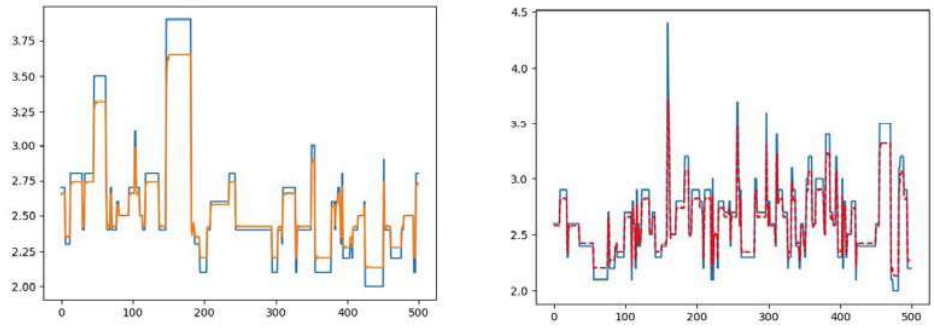


Figura 6.4: Predicción de la serie temporal de magnitud del terremoto sin ceros (izquierda) Entrenamiento (derecha) Predicción

La precisión de esta predicción depende de la ausencia de eventos contiguos en la serie de tiempo original con la misma magnitud. Al analizar los más de doscientos mil eventos durante los últimos 20 años en la península italiana, no se encontraron eventos contiguos en períodos de una hora con la misma magnitud. y de acuerdo con la desigualdad de Hoeffding (1963) [21] que se muestra a continuación, donde v representa la probabilidad de magnitudes repetidas en el conjunto de datos, μ representa la probabilidad real de magnitudes repetidas y N el tamaño de la muestra, tenemos la probabilidad de que en

los próximos 20 años haya alguna magnitud repetida es aproximadamente 10^{-5}, eso es prácticamente nulo.

$$P\left[|v-\mu| > \varepsilon\right] \leq 2e^{2\varepsilon^2 N} \tag{6.1}$$

Luego, una vez que tenemos el modelo entrenado, podemos aplicar un filtro simple a la predicción que devuelve los valores repetidos a cero, obteniendo así el modelo original como se muestra en la figura 6

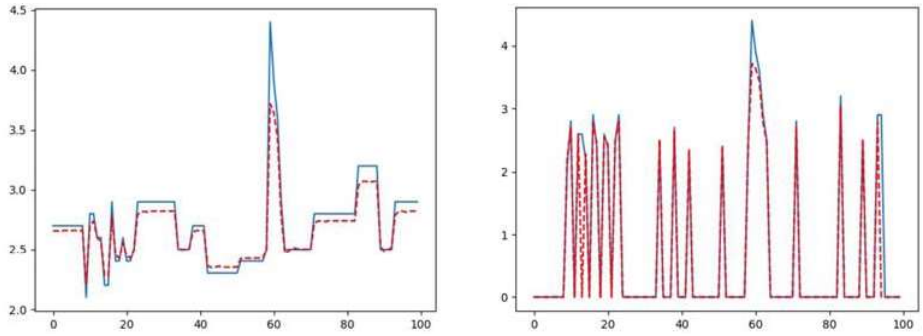


Figura 6.5: Sin predicción de ceros (Izuierda) Sin predicción de ceros sobre la serie de tiempo original por hora (Derecha)

Con este método puede predecir con un error mínimo la magnitud del evento que ocurrirá en la próxima hora, sin embargo, si así se desea, se podría predecir la magnitud en las próximas 3 horas utilizando como entrada $y(k-3), \ldots, y(k-8)$, sin embargo el resultado no es tan favorable y el error crece rápidamente.

En este trabajo se usa un tiempo entre eventos de una hora, con el cual se puede predecir la magnitud del evento de la siguiente hora, si cambiamos el intervalo k de una hora a un día, entonces el patrón de ceros que se muestra en la Figura 6.2 cambia, a medida que los ceros se vuelven más escasos y la predicción se vuelve más compleja. Sin embargo, al elevar el umbral actual de 2, el patrón se vuelve similar y se puede aplicar el mismo modelo analizado previamente. Si se quisiera predecir la magnitud máxima de los eventos del próximo mes, entonces el umbral debería subir aún más para ajustarse a un patrón similar, pero los datos no serían suficientes para tener una predicción confiable.

Por otro lado, en el presente modelo hay dos complicaciones importantes, la primera es que debido al cambio de valor de los ceros al entrenar el modelo si la magnitud de los eventos anteriores no se controla desde una magnitud distinta de cero, entonces este modelo es incapaz de hacer una predicción. El segundo problema está relacionado con el

filtro final de la predicción dado que eliminamos las magnitudes repetidas si ocurre un evento de la misma magnitud, no una sino varias horas después, entonces el modelo no podrá predecir este segundo evento.

6.1.2. Método de repetición

Como se discutió en la sección previa serie temporal que representa la magnitud de los eventos sísmicos no solo parece comportarse de manera muy aleatoria, sino que también parece ser muy intermitente, y esto hace que la predicción sea extremadamente complicada. Y con el método de retención de orden cero, parecen existir algunas complicaciones. Por lo tanto, dado el modelo de aprendizaje expresado como

$$m(k) = N_m\left[m(k-1), \ldots, m(k-n)\right]$$

Donde $m(k)$ representa la magnitud máxima de los eventos que ocurren durante el período k, que en este estudio es de 1 día y N_m representa el módulo de red neuronal recurrente LSTM-NN.

Se propone entonces aplicar una transformación T a los datos antes de realizar el modelado RNN de los datos. La transformación T es la siguiente:

$$T(x) = \{x(k)|x(k) = [x(k)_{xN}] \ \forall \ x(k) \text{ en } x\} \text{ donde } N \exists Z^+$$

La transformación T replica los eventos N veces; aunque esta transformación corresponde a generar eventos artificialmente con igual magnitud, por otro lado, hace que las series temporales sean menos intermitentes y, por lo tanto, más fáciles de investigar utilizando el RNN. Un ejemplo de esta transformación se muestra en la figura.

Esta transformación es fácilmente reversible ya que el número de repeticiones de cada evento es un entero conocido; por lo tanto, es suficiente eliminar las N repeticiones y se obtendrá la serie original.

Al aplicar la transformación T, el entrenamiento y la predicción se modifican como se muestra en la figura 6.7.

Para elegir correctamente el valor de N, se compara la distribución de errores con varios valores de N y se evalua el rendimiento del entrenamiento. El cuadro siguiente muestra varios casos de $N = 3, 6, 12, 30, 60, 120, 200, 500$.

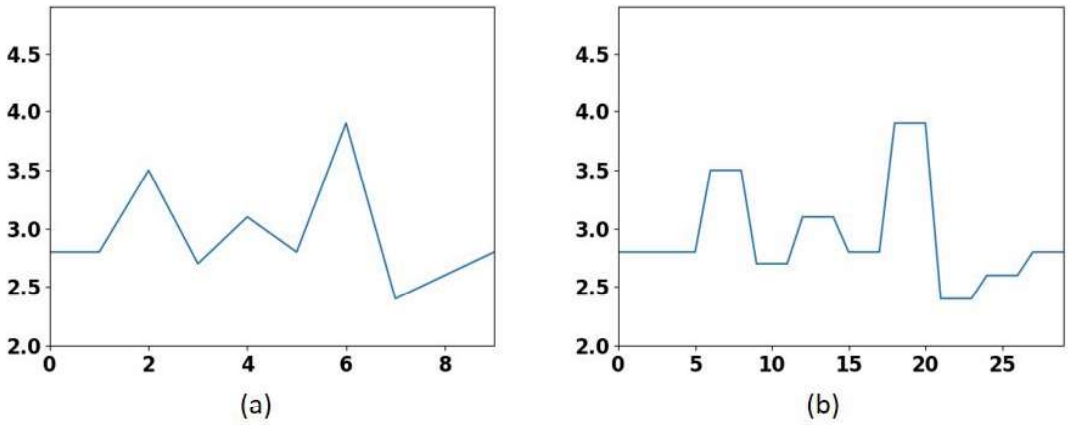


Figura 6.6: Conjunto de datos original, primeros 10 días (a) y su transformación $T(x)$ con $N = 3$ (b)

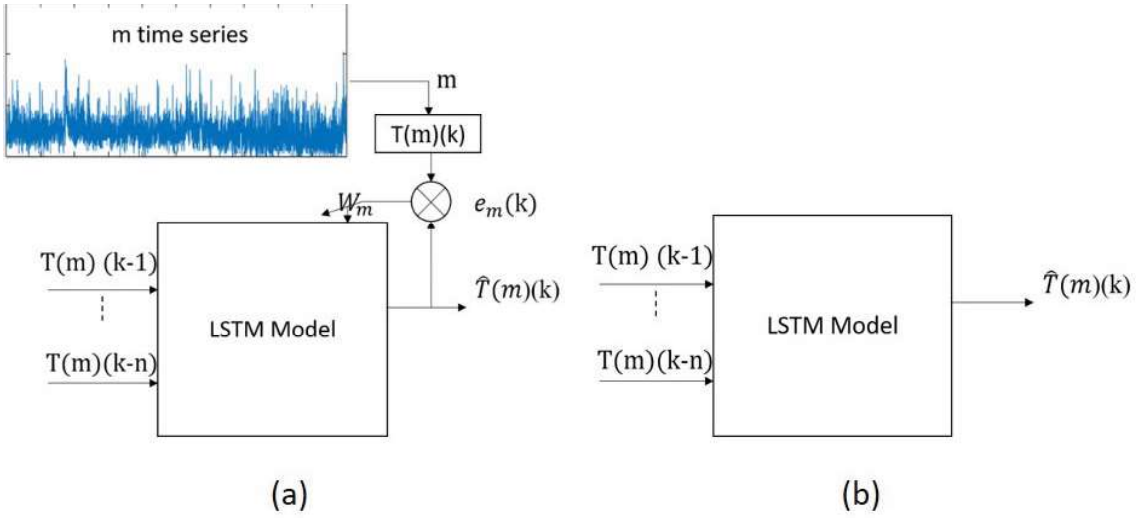
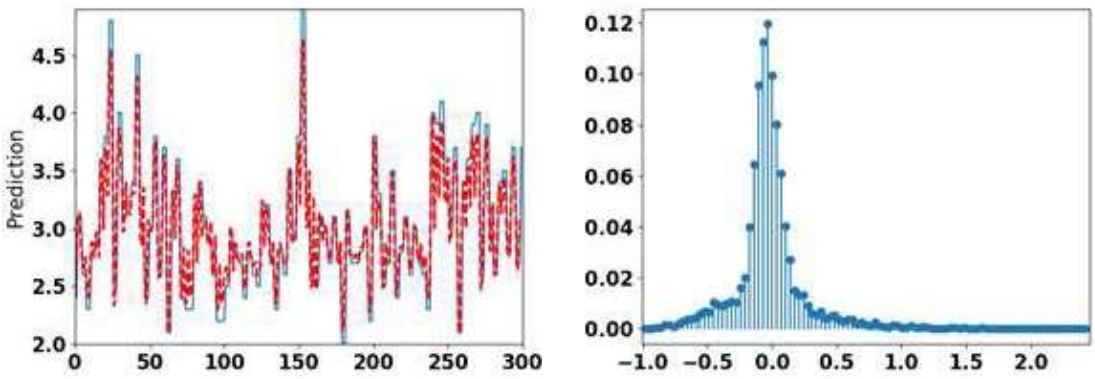
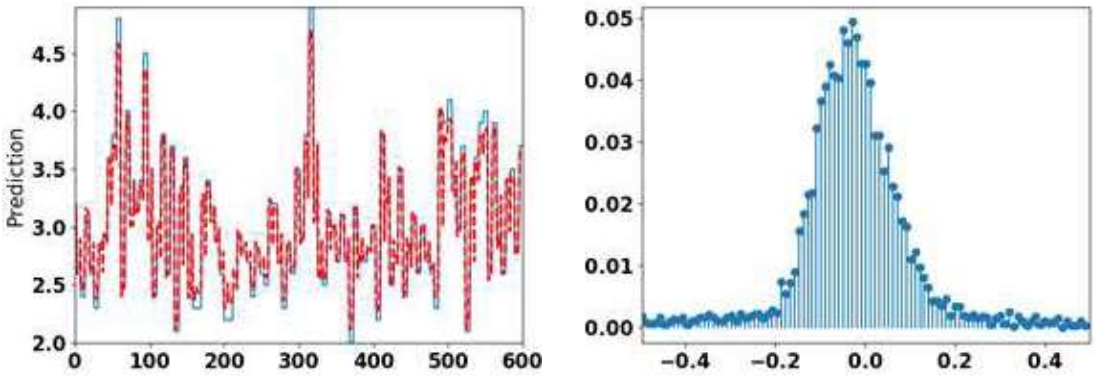


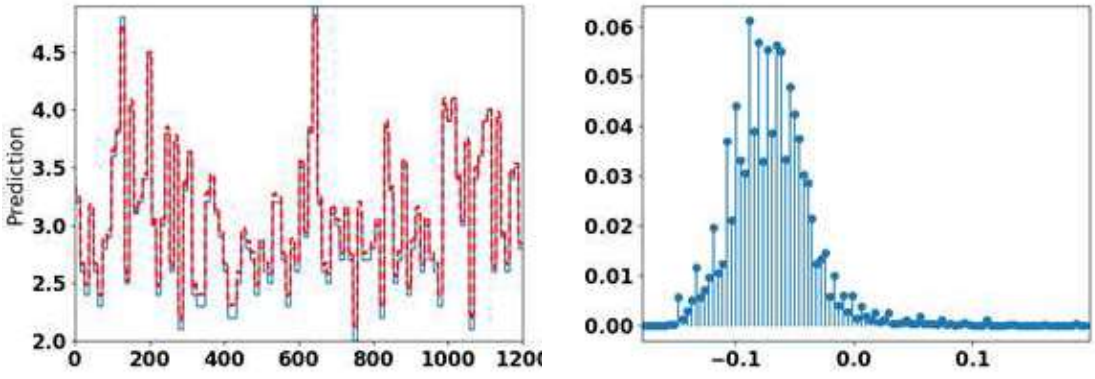
Figura 6.7: modelo LSTM-NN para entrenamiento (a) y predicción (b) usando la transformación propuesta



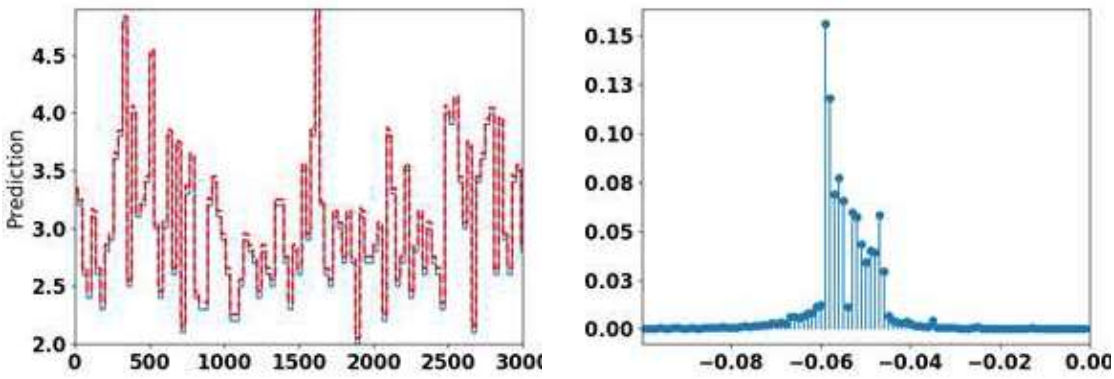
Predicción y distribución para N = 3, MSE: 0.00607



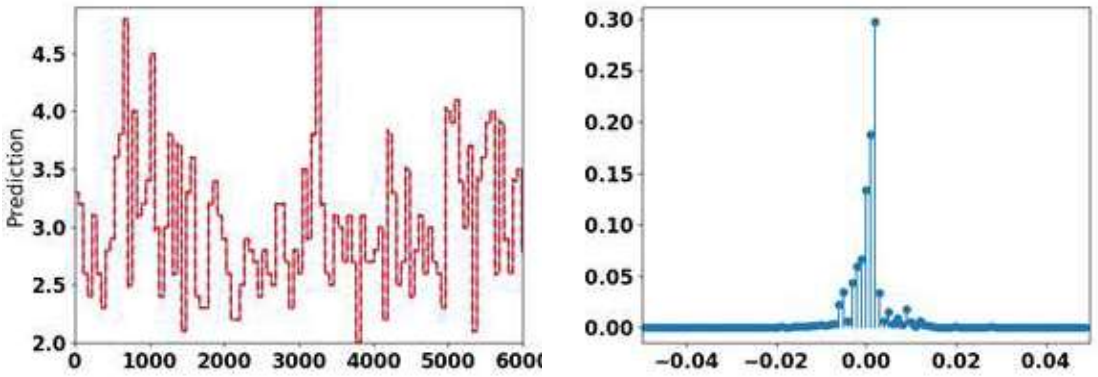
Predicción y distribución para N = 6, MSE: 0.00347



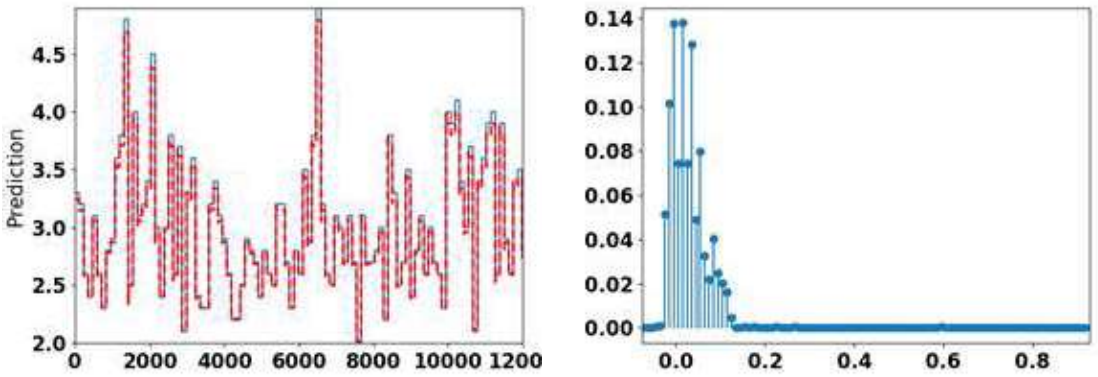
Predicción y distribución para N = 12, MSE: 0.00254



Predicción y distribución para N = 30, MSE: 0.00111



Predicción y distribución para N = 60, MSE: 0.00049



Predicción y distribución para N = 120, MSE: 0.00034

Se puede ver que el valor del error disminuye con el aumento de N. Dicha disminución del error con N, como es bien sabido en el campo del aprendizaje automático, no sería necesariamente una buena característica, porque cuanto más la red es entrenada, lo más probable es que se pueda sobre entrenar y no representar el comportamiento real del sistema. Sin embargo, es imposible obtener un valor de error óptimo y evitar incurrir en sobreajuste6.8. En el caso del presente estudio, para N mayor que 30 la variación del error es del orden de 10^{-5}, y esto ya indica un sobreajuste.

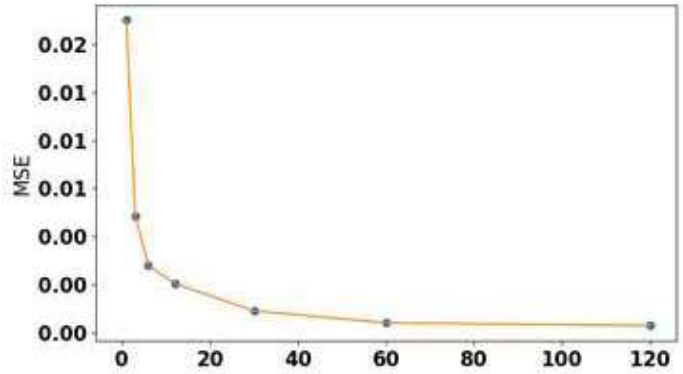


Figura 6.8: MSE para el entrenamiento de red con diferentes valores de N de 1 a 120

Por lo tanto, se considera el valor óptimo de N como 30, siendo estos los resultados de la predicción de magnitud para la sismicidad italiana utilizando el modelo con la transformación $T(x)$.

Con el modelo propuesto, demostramos que es posible predecir la magnitud máxima de los eventos que ocurrirán al día siguiente, considerando los datos de los 30 días anteriores. Sin embargo, dado que el estudio considera todo el territorio italiano, el esquema RNN propuesto no puede usarse para situaciones prácticas de emergencia, donde se debe dar una alarma con la mejor información espacial precisa posible. Aunque se planea en el futuro explorar mejor la posibilidad de pronosticar la magnitud máxima de un terremoto en áreas pequeñas, este método de pronóstico sugiere una nueva forma de pronosticar series temporales muy intermitentes como secuencias de magnitud de terremotos.

6.2. Estructura secuencia a secuencia profunda para la predicción de sismos

Se sabe qué serie temporal que representa la magnitud de los eventos sísmicos no solo parece comportarse de manera muy aleatoria, sino que también parece ser muy intermitente, y esto hace que la predicción sea extremadamente complicada. Sin embargo, con los métodos planteados anteriormente se comienza a dar una solución para realizar una predicción de la máxima magnitud de un evento diario en toda Italia. Aunque si bien se logran buenos resultados esto no se podría aplicar en una situación real, por tal motivo en esta sección se pretende predecir no solo la magnitud del próximo evento si no también su profundidad y mejor aún su latitud y longitud, lo que diría una aproximación de donde va a ocurrir el próximo evento. Esto con el uso de la estructura secuencia a secuencia profunda utilizando celdas GRUs. Por lo tanto, dado el modelo de aprendizaje quedará expresado como

$$[m(k), p(k), la(k), lo(x)] = N_m \begin{bmatrix} m(k-1), \ldots, m(k-n), \\ p(k-1), \ldots, p(k-n), \\ la(k-1), \ldots, la(k-n), \\ lo(k-1), \ldots, lo(k-n) \end{bmatrix}$$

Donde $m(k)$ representa la magnitud del siguiente evento en el tiempo k, el cual representa el tiempo inter evento y puede ser variable, $p(k)$ representa la profundidad del siguiente evento en el tiempo k, $la(k)$ representa la latitud del siguiente evento en el tiempo k, $lo(k)$ representa la longitud del siguiente evento en el tiempo k, y N_m es el modelo neuronal recurrente profundo secuencia a secuencia planteado en el capítulo 5.

De esta forma se puede observar que se logró un resultado imposible con cualquiera de las otras arquitecturas probadas en el presente trabajo, ya que ahora no solo se puede predecir la máxima magnitud de un evento por día, sino que se puede predecir la magnitud del siguiente evento junto con su profundidad y una aproximación de la ubicación lo que ya propone una solución viable a ser usada en un futuro para alertar de posibles eventos sísmicos. Esto se logró con una estructura de únicamente 100 celdas en la entrada y 100 celdas en la salida.

Suena lógico que el siguiente paso sea probar las predicciones en tiempo real, además de tratar de predecir no solo el siguiente, sino que además los siguientes 10 eventos y calcular el margen de error esto en un futuro cercano, más adelante se podría incluir más información relacionada que ayude a mejorar la predicción para una implementación real.

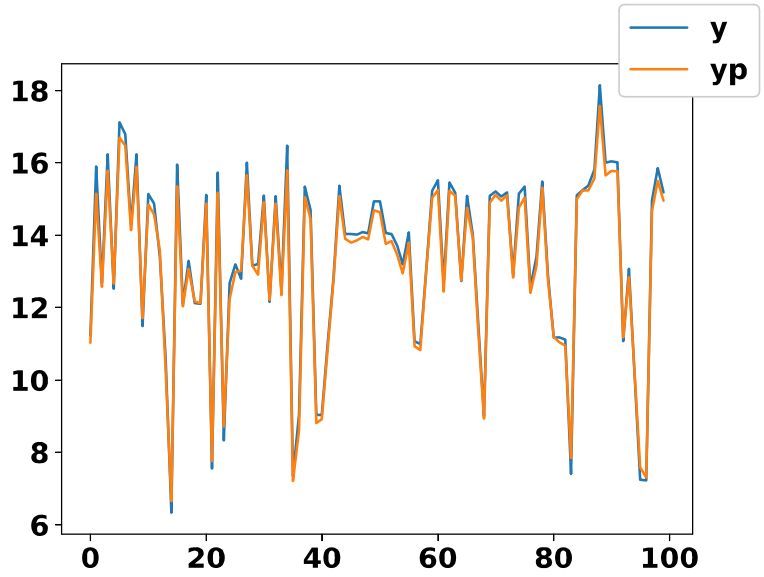


Figura 6.9: Resultados de entrenamiento para la longitud

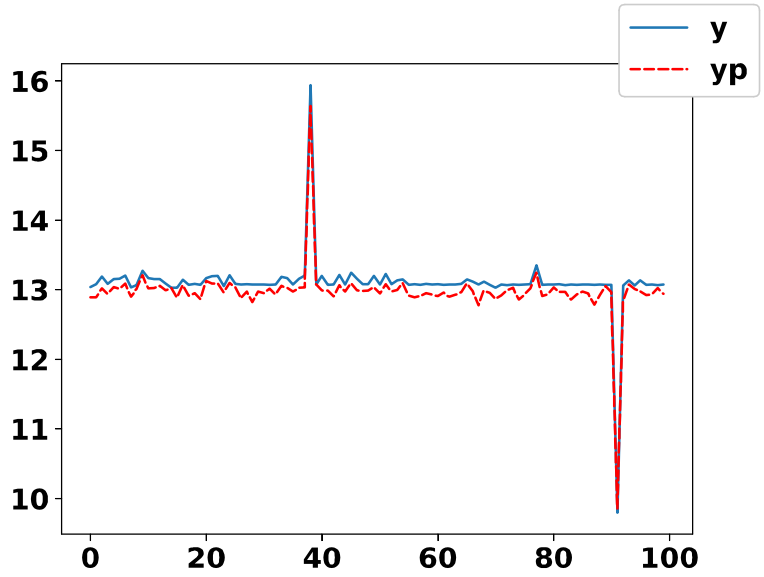


Figura 6.10: Resultados de prueba para la longitud

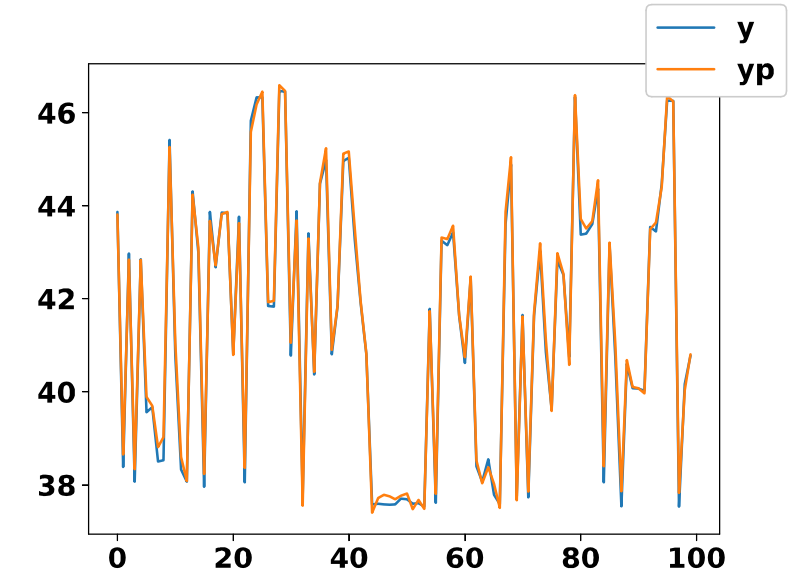


Figura 6.11: Resultados de entrenamiento para la latitud

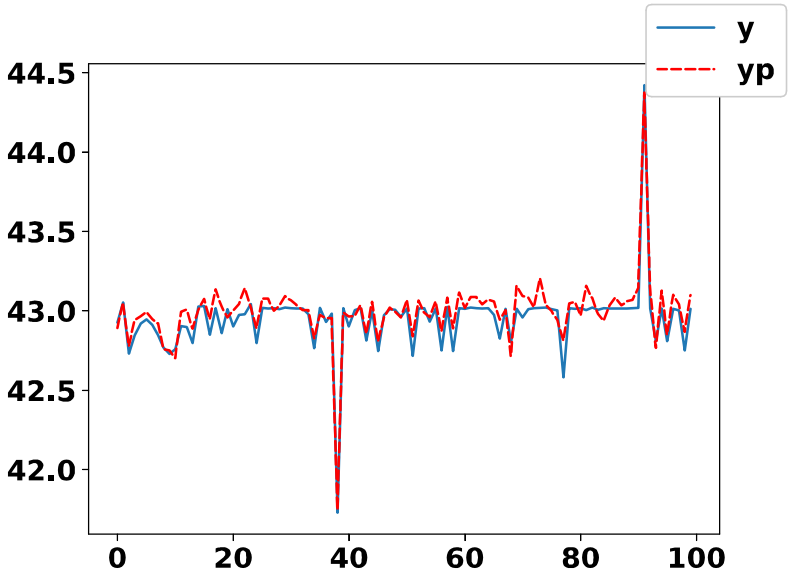


Figura 6.12: Resultados de prueba para la latitud

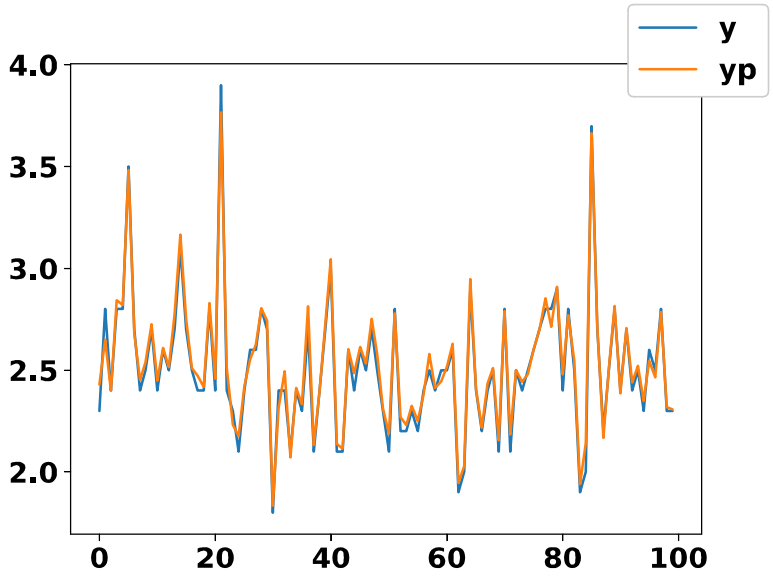


Figura 6.13: Resultados de entrenamiento para la magnitud

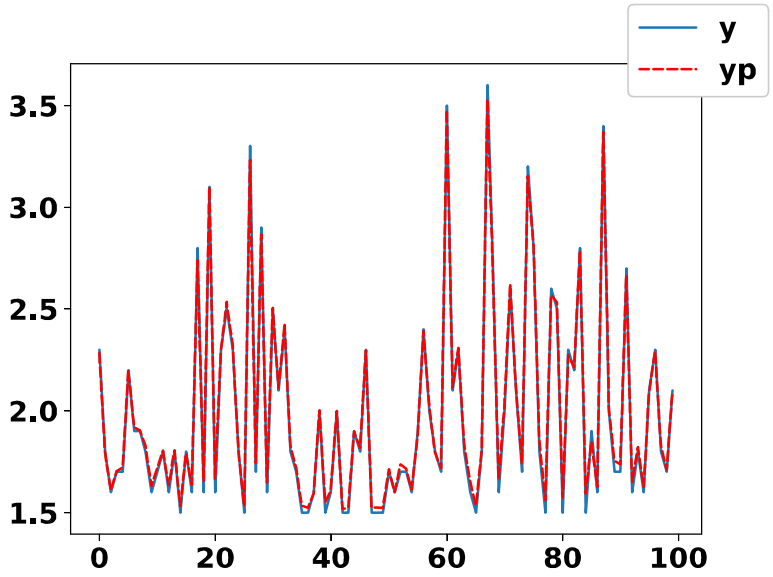


Figura 6.14: Resultados de prueba para la magnitud

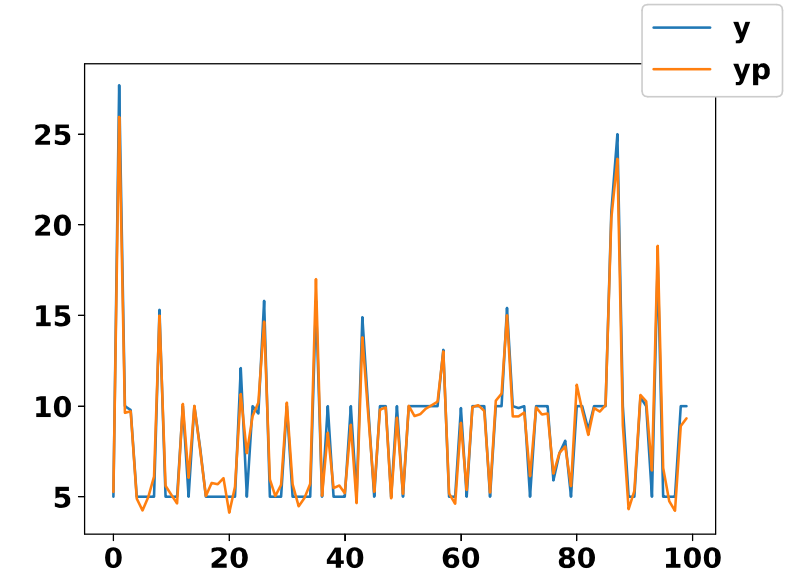


Figura 6.15: Resultados de entrenamiento para la profundidad

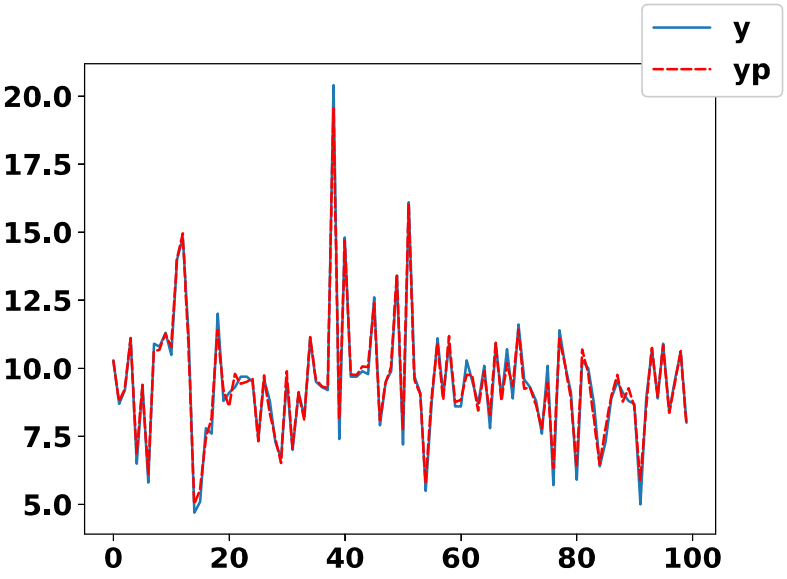


Figura 6.16: Resultados de prueba para la profundidad

Capítulo 7

Conclusiones y trabajo a futuro

Durante la presente tesis se usaron las redes neuronales recurrentes profundas con estructuras LSTM y GRU para el modelado de sistemas dinámicos, demostrando su capacidad de aprender el comportamiento de dichos sistemas. Se abordaron además con éxito problemas complejos en el modelado de sistemas que se probaron no solo teóricamente sino con una aplicación tan importante como es la predicción de sismos.

Las cuatro arquitecturas propuestas demostraron solucionar los distintos problemas propuestos que se querían solucionar, como son el modelado de un sistema en un entorno de simulación, la predicción del siguiente paso k usando solo las predicciones anteriores, la predicción de un tiempo $k+n$, la predicción de múltiples pasos adelante y se demostró en las simulaciones lo que propone la teoría respecto a el desvanecimiento o fuga del gradiente, superando este problema al tener la secuencia de entrada tan larga como sea necesaria.

La aportación de un entrenamiento estable demostró su eficiencia con la simulación de un sistema aerodinámico y el correcto entrenamiento y aprendizaje de su comportamiento pese a la problemática planteada.

Este trabajo incursiona apenas con las arquitecturas LSTM y GRU y su uso en el modelado de sistemas dinámicos, sin embargo, se obtuvieron muy buenos resultados y deja mucho camino que recorrer aún, como es probar con la arquitectura secuencia a secuencia, la predicción del siguiente paso usando las predicciones anteriores y el problema de modelado en un entorno de simulación.

Bibliografía

[1] Fishwick, P. A. (Ed.). Handbook of dynamic system modeling. CRC Press. (2007).

[2] Chen, S., & Billings, S. A. Representations of non-linear systems: the NARMAX model. International Journal of Control, 49(3), 1013-1032. (1989).

[3] Sjöberg, J., Zhang, Q., Ljung, L., Benveniste, A., Delyon, B., Glorennec, P. Y., ... & Juditsky, A. Nonlinear black-box modeling in system identification: a unified overview. Automatica, 31(12) 1691-1724.(1995).

[4] Bailer-Jones, C. A., MacKay, D. J., & Withers, P. J. A recurrent neural network for modelling dynamical systems. network: computation in neural systems, 9(4), 531-547. (1998).

[5] Han, M., Shi, Z., & Wang, W. Modeling dynamic system by recurrent neural network with state variables. In International Symposium on Neural Networks (pp. 200-205). Springer, Berlin, Heidelberg. (2004, August).

[6] Narendra, K. S., & Parthasarathy, K. Neural networks and dynamical systems. International Journal of Approximate Reasoning, 6(2), 109-131.(1992).

[7] Zhang, J., Morris, A. J., Montague, G. A., & Tham, M. T. Dynamic system modelling using mixed node neural networks. IFAC Proceedings Volumes, 27(2), 107-112. (1994).

[8] Rahrooh, A., & Shepard, S. Identification of nonlinear systems using NARMAX model. Nonlinear Analysis: Theory, Methods & Applications, 71(12), e1198-e1202. (2009).

[9] Ruder, S. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747. (2016).

[10] Kingma, D., & Ba, J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980. (2014).

[11] Gers, F. A., & Schmidhuber, J. Recurrent nets that time and count. In Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium (Vol. 3, pp. 189-194). IEEE.(2000, July).

[12] O.Ogunmolu, X.Gu, S.Jiang, and Ni.Gans, Nonlinear Systems Identification Using Deep Dynamic Neural Networks, *arXiv:1610.01439v1 [cs.NE]*, 2016

[13] Y.Wang. A New Concept using LSTM Neural Networks for Dynamic System Identification, *2017 American Control Conference*, Seattle, USA, 5324-5329, 2017

[14] Schoukens, J., & Ljung, L. Wiener-hammerstein benchmark.(2009).

[15] Box, G.E.P., Jenkins, G.M. and Reinsel, G.C. Time Series Analysis; Forecasting and Control. 3rd Edition, Prentice Hall, Englewood Cliff, New Jersey. (1994)

[16] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks, *Advances in Neural Information Processing Systems (NIPS'06)*, pp. 153-160, 2007

[17] Kagan, Yan Y. Are earthquakes predictable? (PDF), Geophysical Journal International, 131 (3): 505–525, Bibcode:1997GeoJI.131..505K, doi:10.1111/j.1365-246X.1997.tb06595.x (December 1997b).

[18] Rikitake, Tsuneji , Çlassification of earthquake precursors", Tectonophysics, 54 (3–4): 293–309, Bibcode:1979Tectp..54..293R, doi:10.1016/0040-1951(79)90372-X. (1 May 1979).

[19] Ghaedi, K., & Ibrahim, Z. Earthquake prediction. Earthquakes-Tectonics, Hazard and Risk Mitigation, 205-227.(2017).

[20] Wang, Q., Guo, Y., Yu, L., & Li, P. Earthquake prediction based on spatio-temporal data mining: an LSTM network approach. IEEE Transactions on Emerging Topics in Computing.(2017).

[21] Abu-Mostafa, Y. S., Magdon-Ismail, M., & Lin, H. T. Learning from data (Vol. 4). New York. (2012).

[22] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078. (2014).

[23] Lipton, Z. C., Berkowitz, J., & Elkan, C. A critical review of recurrent neural networks for sequence learning. arXiv preprint arXiv:1506.00019. (2015).

[24] Unbehauen, H. (Ed.). CONTROL SYSTEMS, ROBOTICS AND AUTOMATION– Vol. VI - Identification of NARMAX and Related Models - Stephen A. Billings and Daniel Coca . EOLSS Publications. (2009).

[25] J.Chung, C.Gulcehre,K.Cho,Y.Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv:1412.3555 [cs.NE]*, 2014

[26] A.Graves, A.Mohamed, G.Hinton. Speech Recognition with Deep Recurrent Neural Networks. *arXiv:1303.5778,* 2013

[27] N.Hirose and R.Tajima. Modeling of Rolling Friction by Recurrent Neural Network using LSTM, *2017 IEEE International Conference on Robotics and Automation (ICRA),* Singapore, May 29 - June 3, 6471-6475, 2017

[28] G.E.Hinton, S. Osindero, Y.W. Teh. A fast learning algorithm for deep belief nets, *Neural Computation*, 18 (7), 2006

[29] S.Hochreiter and J.Schmidhuber. Long short-term memory. *Neural computation,* 9(8) : 1735-1780, 1997

[30] G. Box, G. Jenkins, G. Reinsel. *Time Series Analysis: Forecasting and Control*, 4th Ed, Wiley, 2008.

[31] A.Kumar and Y.S.Chandel. Solar radiation prediction using Artificial Neural Network techniques: A review, *Renewable and Sustainable Energy Reviews*,Volume 33, Pages 772-781, 2014

[32] P.Kingma and J.Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs.LG]*, 2014

[33] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, *NIPS*. 2012.

[34] Y. LeCun, L. Bottou, Y. Bengio, P. Haffne, Gradient based learning applied to document recognition, *Proceeding of the IEEE*, 1998.

[35] L.Ljung. *System Identification-Theory for User*, Prentice Hall, Englewood Cliffs, NJ 07632, 1987

[36] O.Nelles. *Nonlinear system identification: from classical approaches to neural networks and fuzzy models*. Springer Science & Business Media, 2013

[37] J. Schoukens, J. Schoukens, and L. Ljung. Wiener-Hammerstein benchmark, *15th IFAC Symposium on System Identification*, Saint-Malo, France, 2009

[38] N.Srivastava, G.Hinton, A.Krizhevsky, I.Sutskever, R.Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15 (1), 1929-1958, 2014

[39] M. Sugeno, T. Yasukawa, A Fuzzy Logic Based Approach to Qualitative Modeling, *IEEE Trans.on Fuzzy Systems*, Vol.1, No.1, pp.7-31, 1993

[40] L. Wang and R. Langari, Complex Systems Modeling via Fuzzy Logic, *IEEE Trans. on Syst., Man, and Cybernetics*, Vol.26, No.1, pp.100-106, 1996

[41] W.Yu, Nonlinear system identification using discrete-time recurrent neural networks with stable learning algorithms, *Information Sciences*, Vol.158, No.1, 131-147, 2004.

[42] W.Yu, Multiple recurrent neural networks for stable adaptive control, *Neurocomputing*, vol 70, Issues 1-3 , pages 430-444, 2006

[43] W.Yu, Jose de Jesús Rubio, Recurrent neural networks training with stable bounding ellipsoid algorithm, *IEEE Transactions on Neural Networks*, Vol.20, No.6, 983-991,2009.

[44] Lapenna V., Martinelli G. and Telesca L.; Long-range correlation analysis of earthquake-related geochemical variations recorded in Central Italy. Chaos, Solitons. 2004

[45] Matsumoto N., Kitagawa Y. and Koizumi N.; Groundwater-level anomalies associated with a hypothetical preslip prior to the anticipated Tokai earthquake: detectability using the groundwater observation network of the Geological Survey of Japan, AIST. Pure Appl. Geophys., 164, 2377-2396. 2007

[46] PetrakiE , Nikolopoulos D, Nomicos C , Stonham J , Cantzos D , Yannakopoulos P and Kottou S, Electromagnetic Pre-earthquake Precursors: Mechanisms, Data and Models-A Review, J Earth Sci Clim Change, 6:1. 2015.

[47] Rajesh P, H. N. Srivastava, Thermal anomalies in relation to earthquakes in India and its neighbourhood, CURRENT SCIENCE, VOL. 108, NO. 11, pages 2071-2082. 10 JUNE 2015.

[48] Martinelli, G., 2015, Hydrogeologic and geochemical precursors of earthquakes: an assessment for possible applications, Bollettino di GeofisicaTeorica ed Applicata Vol. 56, n. 2, pp. 83-94; June 2015

[49] Telesca and Lovallo, Geophys. Res. Lett., 36, L01308, 2009

[50] Y. Toya , K. F. Tiampo, J. B. Rundle, Chien-chih Chen, Hsien-Chi Li, W. Klein, Pattern informatics approach to earthquake forecasting in 3D, Concurrency Computat.: Pract. Exper.2010;22:1569–1592. 2010.

[51] Sobolev, G., Tyupkin, Y.: Low-seismicity precursors of large earthquakes in Kamchatka. Volcanol. Seismol. 18, 433–446 (1997)

[52] Telesca, L., A non-extensive approach in investigating the seismicity of L'Aquila area (central Italy), struck by the April 6th 2009 earthquake (ML5.8), Terra Nova, 22, 87-93, 2010

[53] Telesca, L., Lovallo, M., Aggarwal, S. K., Khan, P. K., Precursory signatures in the visibility graph analysis of seismicity: an application to the Kachchh (Western India) seismicity, Phys. Chem. Earth, 85-86, 195-200, 2015

[54] Corbi, F., Sandri, L., Bedford, J., Funiciello, F., Brizzi, S., Rosenau, M., & Lallemand, S. Machine Learning can predict the timing and size of analog earthquakes. Geophysical Research Letters, 46(3), 1303-1311. (2019).

[55] Adeli, H., &Panakkat, A. A probabilistic neural network for earthquake magnitude prediction. Neural networks, 22(7), 1018-1024. (2009).

[56] Asim, K. M., Martínez-Álvarez, F., Basit, A., & Iqbal, T. Earthquake magnitude prediction in Hindukush region using machine learning techniques. Natural Hazards, 85(1), 471-486. (2017)

[57] Asencio-Cortés, G., Martínez-Álvarez, F., Troncoso, A., & Morales-Esteban, A. Medium–large earthquake magnitude prediction in Tokyo with artificial neural networks. Neural Computing and Applications, 28(5), 1043-1055. (2017).

[58] Florido, E., Aznarte, J. L., Morales-Esteban, A., & Martínez-Álvarez, F. Earthquake magnitude prediction based on artificial neural networks: A survey. Croatian Operational Research Review, 7(2), 159-169.(2016).

[59] Narayanakumar, S., & Raja, K. A BP artificial neural network model for earthquake magnitude prediction in Himalayas, India. Circuits Syst, 7(11), 3456-3468. (2016).

[60] Asim, K. M., Javed, F., Hainzl, S., & Iqbal, T. Fault Parameters-Based Earthquake Magnitude Estimation Using Artificial Neural Networks. Seismological Research Letters. (2019).

[61] Panakkat, A., &Adeli, H. Neural network models for earthquake magnitude prediction using multiple seismicity indicators. International journal of neural systems, 17(01), 13-33. (2007).

[62] Al Ibrahim, M., Park, J., & Athens, N. Earthquake warning system: Detecting earthquake precursor signals using deep neural networks. (2018).

[63] Huang, J. P., Wang, X. A., Zhao, Y., Xin, C., & Xiang, H. Large earthquake magnitude prediction in Taiwan based on deep learning neural network. Neural Network World, 28(2), 149-160. (2018).

[64] Perol, T., Gharbi, M., & Denolle, M. Convolutional neural network for earthquake detection and location. Science Advances, 4(2), e1700578. (2018).

[65] Kirschner, D., Howes, N., Daly, C., Mukherjee, J., & Li, J. Detecting P-and S-wave arrivals with a recurrent neural network. In SEG Technical Program Expanded Abstracts 2019 (pp. 2659-2662). Society of Exploration Geophysicists.(2019).

[66] Wang, Q., Guo, Y., Yu, L., & Li, P. Earthquake prediction based on spatio-temporal data mining: an LSTM network approach. IEEE Transactions on Emerging Topics in Computing. (2017).

[67] Ugurlu, U.; Oksuz, I.; Tas, O. Electricity Price Forecasting Using Recurrent Neural Networks. Energies ,1255. 2018 11.

[68] Alemany, S., Beltran, J., Perez, A., &Ganzfried, S. Predicting hurricane trajectories using a recurrent neural network. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 33, pp. 468-475). (2019, July).

[69] Liu, H., Mi, X. W., & Li, Y. F. Wind speed forecasting method based on deep learning strategy using empirical wavelet transform, long short term memory neural network and Elman neural network. Energy conversion and management, 156, 498-514. (2018).

[70] Pan, B., Xu, X., & Shi, Z. Tropical cyclone intensity prediction based on recurrent neural networks. Electronics Letters, 55(7), 413-415.(2019).

[71] Qing, X., &Niu, Y. . Hourly day-ahead solar irradiance prediction using weather forecasts by LSTM. Energy, 148, 461-468.(2018)

[72] Hochreiter, S., &Schmidhuber, J. LSTM can solve hard long-time lag problems. In Advances in neural information processing systems (pp. 473-479). (1997)

[73] Yu, W., Li, X., & Gonzalez, J. Fast Training of Deep LSTM Networks. In International Symposium on Neural Networks (pp. 3-10). Springer, Cham.(2019, July)

[74] González, J., Yu, W., & Telesca, L. Earthquakes magnitude prediction using recurrent neural networks (2018)

[75] Abdel-Nasser, M., & Mahmoud, K. Accurate photovoltaic power forecasting models using deep LSTM-RNN. Neural Computing and Applications, 31(7), 2727-2740. (2019).

[76] Tsai, Y. T., Zeng, Y. R., & Chang, Y. S. Air pollution forecasting using RNN with LSTM. In 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech) (pp. 1074-1079). IEEE

[77] Bakir, H., Chniti, G., & Zaher, H. (2018). E-Commerce price forecasting using LSTM neural. networks. Int. J. Mach. Learn. Comput, 8, 169-174. (2018, August).

[78] Gonzalez, J., & Yu, W. Non-linear system modeling using LSTM neural networks. IFAC-PapersOnLine, 51(13), 485-489.(2018).