ESCOLA TÉCNICA SUPERIOR
DE ENXEÑARÍA

TRABALLO FIN DE MÁSTER

MÁSTER EN TECNOLOXÍAS DE ANÁLISE DE DATOS MASIVOS: BIG DATA

Detección de anomalías usando Deep Learning

2018/2019

Autor: César Manuel Paz Guzmán
Titor: Manuel Mucientes Molina
Cotitor: Javier Cacheiro López
Santiago de Compostela, Julio 2019

*

Índice xeral

Capítulo 1

INTRODUCIÓN

O Centro de Supercomputación de Galicia conta cun CPD con máis de 12.000 cores e máis de 2 PB de almacenamento, ademais de numerosos equipos de rede correspondentes a Recetga. Todos estes sistemas xeran un grande volume de métricas (uso de CPU, temperatura, memoria, etc) que son recollidas e almacenadas.

Continuamente, tanto a universidade como as empresas externas envían traballos ó CESGA para que sexan executados, a través dun conxunto de nodos previamente asignados. Todos estes nodos teñen varios sensores que xeran un grande volume de métricas mentres o traballo é executado. O valor destas métricas dependerá tanto da natureza do traballo como do número de nodos asignados. Toda esta información almacénase nun clúster Hadoop con Hbase e Spark que dispón de 456 cores, 440 discos e rede de 10Gbit/s.

Polo tanto, é de grande importancia ter unha forma de detectar anomalías en cada un dos nodos, co obxectivo de predicir cando un nodo vai fallar. Na actualidade, as métricas xeradas por cada nodo son analizadas usando series temporais modeladas con métodos de Machine Learning clásicos para detectar tales anomalías. Estes métodos dan bos resultados pero requiren de moito esforzo á hora de xerar as características. Polo tanto, queremos buscar unha nova forma de obter resultados tan bos como os obtidos co sistema actual, pero moito máis flexibles.

O obxectivo xeral é avaliar o potencial que ofrecen as novas técnicas baseadas en deep learning á hora de detectar anomalías nun entorno real como o CPD do CESGA. En concreto, avaliarase o modelado de series temporais mediante o uso de redes neuronais recorrentes LSTM e GRU, que se están a aplicar con éxito en problemas similares. Os datos a analizar correspóndense coas series temporais asociadas ás distintas métricas que se recollen dos nodos durante a execución dos traballos.

Capítulo 2

ESTADO DO ARTE

Neste apartado, vanse introducir as principais tecnoloxías utilizadas no proxecto: as redes neuronais recorrentes LSTM e GRU, as librarías Keras e Tensorflow e ferramentas para a xestión do desenvolvemento.

2.1. REDES NEURONAIS RECORRENTES

As redes neuronais recorrentes son un tipo de rede neuronal que son moi útiles para o análise de series temporais ou de calquer tipo de datos que teña unha estrutura temporal xa que poden manter información pasada, conseguindo crear temporalidade e permitindo polo tanto que a rede teña memoria.

Na figura 2.1 podemos ver a representación dunha serie temporal que vai dende o paso $t-1$ ata o paso $t+1$ usando as redes RNN. Podemos interpretar a estrutura da RNN como varias copias da mesma rede, onde cada rede pasa a súa saída a unha rede sucesora. Esta arquitectura permite tomar decisións baseadas na entrada actual e na información dos seus estados anteriores combinados. Desta forma permite que a información máis relevante persista. [3]

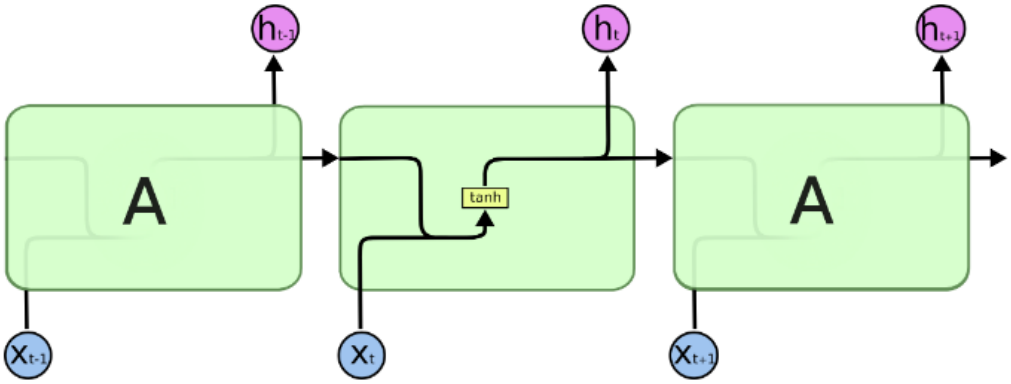


Figura 2.1: RNN interconectadas

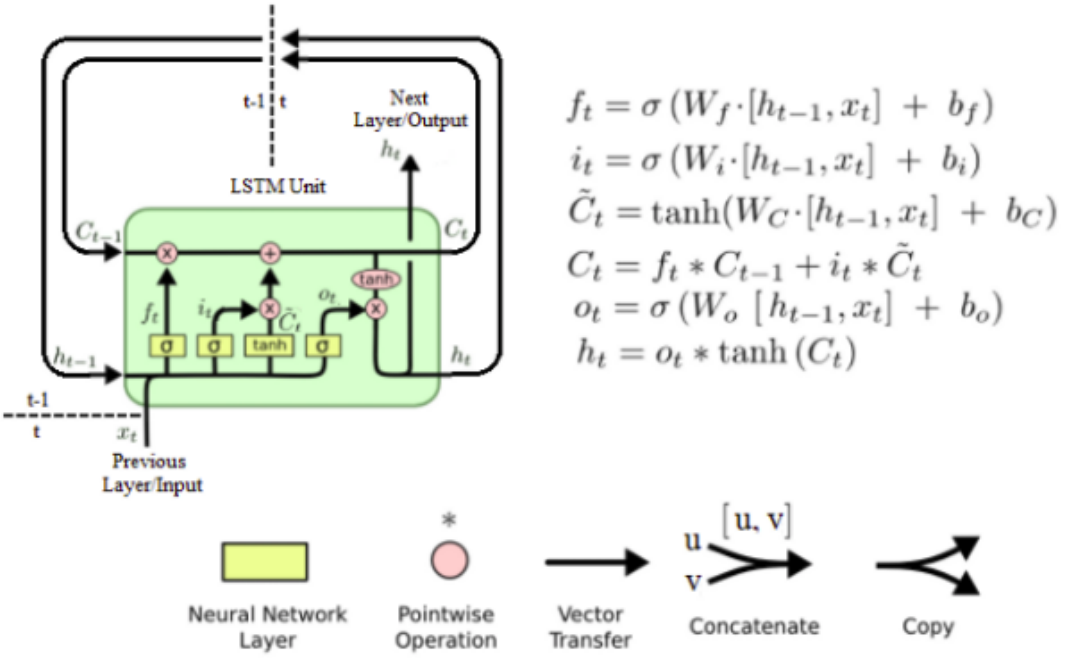
As RNN poden usar unha información recente para predicir o valor do paso seguinte, pero cando se necesita aprender dependencias temporais longas pérdese a relación entre a información relevante e o valor o cal queremos predicir. Isto é debido a que os gradientes propagados tenden a crecer enormemente ou a desvanecerse co tempo debido a que o gradiente depende non só do erro actual senón tamén dos erros pasado, o cal provoca dificultades para memorizar dependencias a longo prazo.

Neste proxecto, como é necesario adestrar usando series temporais sen unha tendencia clara, necesitaremos dunha solución que evite o problema da dependencia a longo prazo. Aquí entran un tipo especial de rede recorrente: as LSTM e GRU. [1]

2.1.1. LSTM

É un tipo de rede neuronal recorrente capaz de aprender dependencias temporais longas, xa que conservan o erro e ademais o propagan a través do tempo e as capas. Consideramos unha serie temporal X=$\{x^{(1)}, x^{(2)}, ..., x^{(l)}\}$ onde cada punto $x^{(t)}$ é un vector m dimensional $\{x_1^{(t)}, x_2^{(t)}, ..., x_m^{(t)}\}$, e onde cada un dos seus elementos corresponden a cada unha das variables de entrada.

A figura 2.2 móstranos como é a arquitectura dunha rede LSTM no paso $t-1$. Tal e como se dixo con anterioridade, a entrada e a saída da rede son vectores de m dimensións. O círculo representa unha operación punto a punto entre dous vectores, como a multiplicación ou a suma. E os rectángulos representan as capas. En lugar de ter unha única capa como ocorre coas RNN (capa tanh - Figura 2.1), as LSTM contan con catro, as cales interactúan dunha maneira particular. Como vemos, a entrada da primeira capa é a concatenación da entrada actual coa información dos seus estados anteriores combinados, representado por f_t.:



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figura 2.2: Arquitectura dunha rede LSTM [2]

A clave destas redes é o estado da cela. O estado inicial C_{t-1} correspóndese co estado final C_t do anterior paso. As redes LSTM inicializan este estado de maneira aleatoria e en cada paso elixen de maneira dinámica como actualizalo con cada nova mostra, de tal forma que hai unha capa para borrar partes do estado anterior $(f_t * C_t)$ e outra para actualizar o estado actual.

A diferencia entre o estado da cela C_t e o estado interno da rede h_t é que o primeiro actúa como unha memoria, mentres que o segunda actúa como unha copia da mesma que pode ser pasada para procesar as entradas no seguinte paso. Como veremos na explicación das redes GRU, estes dous estados fusiónanse.

O primeiro paso é decidir que información se vai a eliminar do estado interno do paso anterior h_{t-1}. Esta decisión é levada a cabo por unha capa sigmoide chamada porta de esquecemento. Colle a concatenación dos vectores h_{t-1} e X_t e produce un número entre 0 e 1 para cada número do estado. O un representa manter completamente a información mentres que un cero representa a súa completa eliminación.

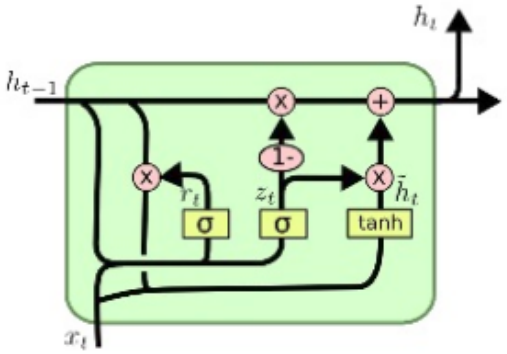
O seguinte paso é decidir que nova información se vai almacenar na cela de estado. En primeiro lugar, unha capa sigmoide chamada porta de entrada, decide que valores sen van a actualizar. A continuación, unha capa de tanh crea un vector de novos valores candidatos C_t, que se poden engadir ao estado. No seguinte paso, combínase estes dous para crear unha actualización do estado.

Finalmente, a saída estará baseada na cela de estado, pero será unha versión filtrada. En primeiro lugar, execútase unha capa sigmoide que decide que partes do estado interno se vai a eliminar o_t. Entón, o estado da cela atravesa unha capa tanh e multiplícase pola saída da porta sigmoide, de forma que só se emitirán as partes que se decidan.

2.1.2. GRU

As redes GRU pódense considerar como unha variación da arquitectura LSTM porque ambos están deseñados de maneira similar, aínda que son máis simples e fáciles de adestrar que as LSTM. Combina as portas de esquecemento e de entrada nunha única porta de actualización. Tamén combina o estado da cela e o estado interno.

Para resolver o problema do gradiente acontecido nas redes recorrentes estándar, GRU usa as portas de actualización z_t e de reaxuste r_t. Basicamente, estas portas deciden que información debe pasarse á saída h_t. A porta de reaxuste indica en que grado queremos preservar o estado e a porta de actualización indica en qué grado queremos actualizalo (Figura 2.3).



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

Figura 2.3: Arquitectura dunha rede GRU

Este tipo de redes usan menos parámetros de adestramento, polo que usan menos memoria, e se executan e adestran máis rápido cas redes LSTM. Sen embargo, teñen a desvantaxe de que son un pouco menos precisas en conxuntos de datos de moi longa temporalidade.

2.2. FRAMEWORKS

Para a implementación deste proxecto faremos uso da ferramenta *Jupyter Notebook* para o desenvolvemento interactivo, en *Python 3.6*. E para a construción das redes neuronais utilizaranse as seguintes librarías [7]:

2.2.1. TensorFlow

TensorFlow é una libraría de código aberto usado para a creación de redes neuronais. Está a un nivel máis baixo e profundo que Keras, proporcionando máis control e flexibilidade á hora de construír estas as redes. Polo tanto, é máis recomendable para proxectos grandes e complexos.

2.2.2. Keras

Keras é unha libraría de alto nivel de Python que proporciona de forma limpa e sinxela a creación de redes neuronais. Permite construír ditas redes con menos coñecementos e código que TensorFlow, xa que está construída encima desta.

2.3. OUTRAS TECNOLOXÍAS

2.3.1. GitLab

Neste proxecto usaremos GitLab por diversos motivos:

- Poder ter un control das versións.
- Ter o proxecto centralizado, de forma que poidamos acceder a el dende o supercomputador Finisterrae ou dende o cluster Hadoop.
- Cambios e integración dende ambos supercomputadores ou dende o portátil persoal.

2.3.2. Pivotal Tracker

É unha ferramenta de xestión de proxectos áxiles. Podemos facer historias de usuario para indicar as tarefas que debemos de facer, e permítenos ter unha visual das tarefas terminadas, pendentes e en proceso. Tamén podemos ver a importancia de cada unha, o tempo estimado e o rol que a ten que levar a cabo.

Sírvenos en definitiva, para saber en que punto do proxecto estamos, e planificar as tarefas para facer entregas periódicas dunha forma eficiente.

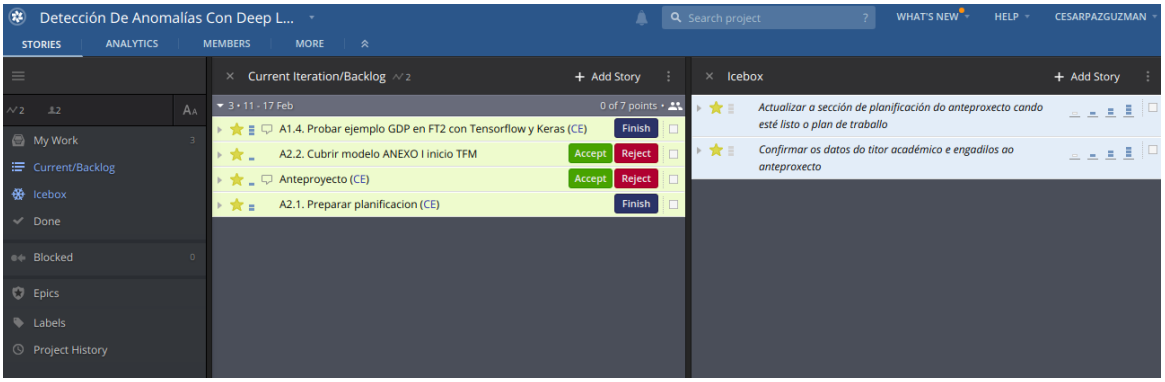


Figura 2.4: Captura do proxecto no pivotal tracker

Capítulo 3

ENTENDEMENTO DA ÁREA DE NEGOCIO

Trátase da primeira fase da metodoloxía TDSP e consiste na identificación dos obxectivos e dos KPI's, na definición do alcance, e na descrición das fontes de datos. En base ó análise realizado nesta fase, poderemos facer unha estimación máis precisa na estimación temporal.

Neste proxecto, esta fase, xunto coa fase de modelado, é a máis importante xa que nos permite entender de forma máis completa posible o que se desexa obter. Permitiranos a orientación cara ó resultado final desexado en base ó análise realizado.

3.1. Contexto inicial

Na actualidade, os valores xerados polos sensores son recollidos dende o supercomputador Finisterrae a través dunha libraría de Python, accedendo ós datos mediante un API Rest de OpenTSDB. Será neste supercomputador onde adestraremos e avaliaremos os modelos xerados. Polo tanto, haberá que analizar o código actual co obxectivo de aprender a recoller os datos necesarios do HBase.

Os sensores producen datos cada dez segundos, polo que á hora de realizar o modelado será necesario facer un remuestreo dos datos co fin de reducir a variabilidade dos mesmos.

3.2. Obxetivos

Debido a gran cantidade de cores (>12.000) é de gran importancia ter unha forma de detectar as anomalías que acontezan co obxectivo de predicir cando e de que maneira vai fallar.

Será necesario avaliar o potencial que ofrecen as novas técnicas baseadas en deep learning, en concreto usando as redes neuronais recorrentes LSTM e GRU que están sendo aplicadas con éxito en problemas similares, e comparar os resultados obtidos co modelo clásico actual.

Para lograr o obxectivo principal, como primeiro paso, construiremos varios modelos diferentes ata atopar o que mellor se adapte ó problema. É importante que o modelo sexa altamente preciso, dado que neste primeiro paso o que nos interesa é modelar da mellor maneira posible as series temporais. Posteriormente, mediante técnicas estadísticas determinaremos se vai acontecer unha posible anomalía, comparando o resultado co resto dos nodos nos que se executa o traballo.

3.3. Definición de KPI's

Coa meta de determinar se os obxectivos descritos se cumpriron ou non, e que parte do que vamos a obter na avaliación nos vai a servir, vanse a describir os identificadores KPI's.

- Que a taxa de falsos positivos non superen o 10 %. Ten que ser altamente preciso, xa de nada nos vale un sistema onde se detecten e se traten falsas anomalías.
- O modelo debe ser capaz de predicir o seguinte paso cun erro menor do 10 %.

- O modelo debe ter unha taxa de recall do 30 %, é dicir, debe atopar un mínimo do 30 % das anomalías que acontezan.
- Que o modelo sexa capaz de analizar os datos e dar o resultado en menos de 5 minutos

3.4. Definición do alcance

A continuación describiranse os requisitos do proxecto. Compre mencionar que ó tratarse dunha avaliación non temos un usuario sobre o cal dirixir o resultado, polo que se vai a limitar a especificar as condicións que debe cumprir dita avaliación para satisfacer as necesidades do proxecto. Que é o que queremos lograr avaliando as métricas xeradas en base as técnicas a empregar.

ID	Grado	Título	Descrición
RQ1	Imprescindible	Lista das posibles anomalías detectadas	Como administrador, desexase poder ter todas as anomalías nunha lista, ca hora e no core no cal tivo lugar.
RQ2	Imprescindible	Ter un indicador da fiabilidade da anomalía	Como administrador, poder ver o grado de fiabilidade das anomalías.
RQ3	Imprescindible	Taxa de falsos positivos baixa	Como administrador, prefírese poder ver unha lista con menor número de anomalías, onde o número de falsos positivos é baixo. É máis importante a precisión que o recall, xa que como administrador non quero comprobar demasiados rexistros.
RQ4	Imprescindible	Mecanismo de comparación co modelo clásico	Como administrador, quérese un mecanismo de comparación para avaliar un modelo e comparalo co modelo clásico
RQ5	Desexable	Xerar unha agrupación das anomalía de natureza similar	Como administrador e co obxectivo de simplificar o tratamento das anomalías, resulta útil que todas as anomalías que describan comportamentos similares aparezan agrupadas, de forma que se poidan tratar conxuntamente.

3.5. Identificación das fontes de datos

A continuación vaise explicar a arquitectura de como se recollen as métricas almacenadas nos distintos dispositivos, sen entrar en moito detalle o que fan os distintos compoñentes.

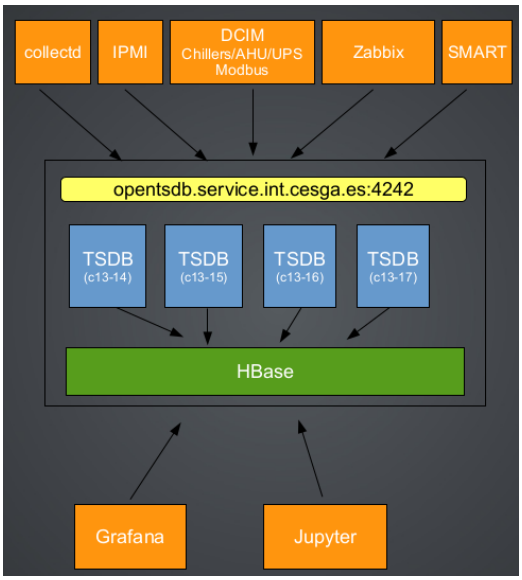


Figura 3.1: Arquitectura recolectora dos datos

As distintas fontes de datos identificadas descríbense a continuación:

- **Collectd:** Recolle as métricas dos servidores. Exemplo: Uso de CPU, uso da memoria, etc.
- **IPMI:** Recolle as métricas dos procesadores adicionais que traen os servidores. Exemplos: Consumo, temperatura, etc.
- **DCIM:** Recolle as métricas de todos os equipos correspondentes á infraestrutura de soporte.
- **Zabbix:** Recolle as métricas de rede, como o ancho de banda.
- **SMART:** Recolle as métricas dos discos.

Estas fontes de datos recollen as métricas e almacénaas no HBase a través dunha capa extra TSDB. Esta capa representa unha base de datos optimizada para o tratamento dos datos de series temporais, distribuída e escalable, permitindo recolectar miles de métricas de moitos dispositivos a unha alta frecuencia.

Para este proxecto, contamos con módulos previamente implementados en Python para a consulta de ditas métricas, polo que a recompilación inicial dos datos xa está feita de inicio. O único que necesitamos coñecer é como funcionan ditos módulos.

3.6. Identificación das métricas a analizar

A continuación explicaranse as distintas métricas que se van a analizar co fin de conseguir predicir unha delas. Serán as seguintes nove:

- Carga de la CPU (load): Carga de traballo que ten o procesador. Os procesadores dos nodos que se van analizar dispoñen de 24 núcleos, polo que o valor desta métrica, en condicións normais, debería estar en torno este valor.
- Memoria caché (mem_cache): Indica a memoria caché que está sendo usada.
- Memoria usada (mem_use): Indica a memoria RAM que está senda usada.
- Espera de CPU (cpu_wait): É o tempo que unha tarefa ten que esperar para acceder aos recursos da CPU.
- Tempo de CPU usado polo kernel (cpu_system).
- Tempo de CPU usado polo usuario (cpu_user).
- Potencia de CPU (power)
- Temperaturas da CPU1 e CPU2 (cpu1_temp, cpu2_temp).

Capítulo 4

PLANIFICACIÓN

A continuación detállase a definición e estimación da duración das actividades, o cronograma do proxecto e as metodoloxías áxiles utilizadas.

4.1. Definición das actividades

O primeiro que se debe facer é identificar e documentar cada unha das accións necesarias para levar a cabo o proxecto. Debido a que se trata dun proxecto relacionado co análise de datos, adoptaremos a metodoloxía TDSP. Esta metodoloxía proporciona un ciclo de vida composto de 5 fases, aínda que neste proxecto só farán falta as 3 primeiras debido a que non se ten o propósito de implementar a solución nun entorno real [4].

1. **Coñecemento do negocio (Bussiness Understanding):** Consiste na descrición do contexto inicial e dos obxectivos, na definición do alcance (requirimentos e EDT), e na descrición das fontes de datos.
2. **Adquisición e comprensión dos datos (Data Understanding):** Consiste na recompilación inicial dos datos e no seu entendemento e descrición. Neste caso, os datos necesarios xa están todos accesibles nunha base de datos HBase, e coa importación dunha librería permítenos con funcións de Python acceder ós datos mediante un API Rest de OpenTSDB.
3. **Modelado (Modeling):** Consistirá na implementación de varios modelos usando diferentes técnicas. Para cada prototipado, teremos que seleccionar a técnica de modelado e construír o modelo con certos valores de hiperparámetros e avalialo en base ó conxunto de test. Na avaliación empregaremos técnicas de estadística descritiva co obxectivo de atopar as anomalías nas series temporais.

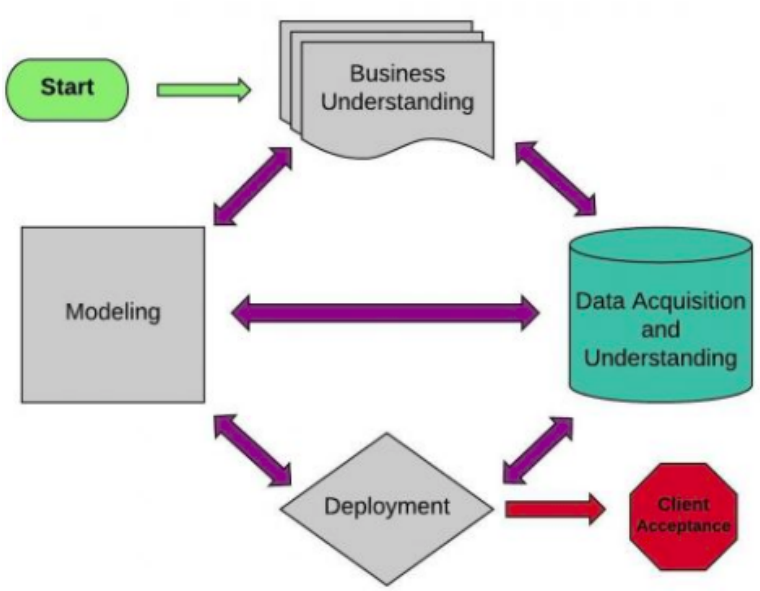


Figura 4.1: Ciclo TDSP

Ademais, ó tratarse dun TFM, debo incluír as seguintes dúas fases adicionais:

- **Lanzamento do TFM:** Consiste na realización do anteproxecto, no aprendizaxe das principais tecnoloxías, na elaboración da introdución e o estado do arte na memoria, e na preparación do entorno de desenvolvemento.
- **Avaliación comparativa:** Consiste na comparación co modelo actual de machine learning, nas conclusións e posibles melloras, no desenvolvemento final da memoria, na realización da presentación e na entrega do proxecto.

4.2. Metodoloxías áxiles

Na actualidade resulta imprescindible o uso de metodoloxías áxiles para a realización dun proxecto. Está demostrado que o seu uso reduce o tempo de desenvolvemento, xa que podemos identificar máis rapidamente os erros, permítenos a ampliación de funcionalidades con moito menor costo, e aumentar a calidade final do produto. Polo tanto, para a realización deste proxecto, faremos uso das seguintes metodoloxías:

- **SCRUM:** Entregas periódicas e reunións semanais, mesmo día, mesma hora, co obxectivo de identificar os erros, posibles melloras, e a seguinte entrega moldeada segundo o resultado da actual. Isto permite reducir tempo de desenvolvemento e mellorar a calidade final do produto, ó estar continuamente tomando as mellores decisións.
- **Kanban:** Para definir cales son as entregas periódicas dunha forma eficiente, faremos uso da ferramenta *Pivotal Tracker*. Con esta ferramenta, podemos ver as actividades que se encontran pendentes, en proceso ou finalizadas.

4.3. Cronograma

Definiuse o cronograma do proxecto (por cuestións de espazo queda sen mostrar), co obxectivo de seguir e controlar a execución de cada unha das actividades de forma que poidamos concluír o proxecto e acadar o alcance do mesmo antes da data de entrega estipulada. Nel, poderemos ver a secuenciación das actividades, a data de inicio e fin de cada unha delas, os hitos e os entregables.

O cronograma estará representado mediante un diagrama de Gantt, cunha duración de 450 horas. A data de comezo está marcada para o 01/02 e a data de finalización para o 15/07, estimando un traballo diario de 3 horas de luns a venres, e de 5 horas os sábados e domingos.

Cabe destacar, que neste cronograma tense en conta os días de exames (18 de maio - 21 de maio), polo que non haberá nada planificado. Ademais tense en conta os resgos aleatorios que poidan acontecer, de forma que a data inicial de finalización do proxecto se marca o 15 de xullo, de forma que teremos 13 días de marxe por se foran necesarios (data fin = 28/07).

Debemos de ter en conta que ó empregar SCRUM, este cronograma é susceptible de sufrir continuas modificacións. Por último, mencionar que as reunións periódicas establécense cada semana ou cada dúas semanas, quedando éstas para os venres.

Capítulo 5

ADQUISICIÓN E COMPRENSIÓN DOS DATOS

5.1. Exploración dos datos

Antes de adestrar ós modelos, debemos de facer unha compresión sólida dos datos. Para lograr isto, visualizaremos os datos para determinar a súa calidade e dar a información necesaria para o seu procesamento co obxectivo de deixalos preparados para o modelado.

A continuación amósanse unha gráfica onde poderemos observar os valores de cada unha das métricas correspondentes a un nodo dun traballo en particular. É con pasos de 30 minutos onde poderemos observar a distribución dos valores e a súa variabilidade. As diferentes métricas que colleremos para realizar a predición son as seguintes: load, mem_cached, mem_used, cpu_wait, cpu_user, cpu_system, power, cpu1_temp, cpu2_temp (Descritas na sección 3.6).

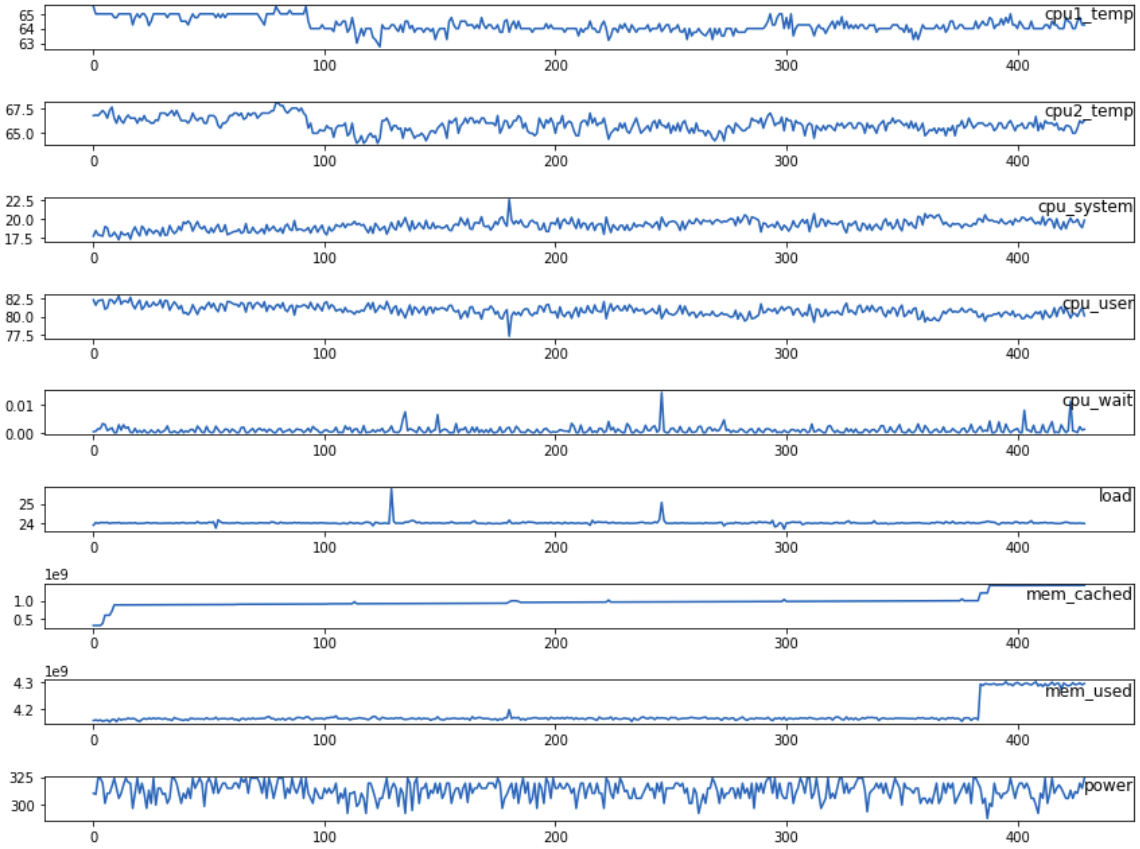


Figura 5.1: Exploración de todas as métricas dun nodo dun traballo en particular

Pódese observar como hai certas métricas que dependen unha das outras. Por exemplo, obsérvase como a suma dos valores entre o cpu_system, cpu_user e cpu_wait atópase arredor de 100, indicando que está usando o 100 % da CPU repartido de varias formas. Así como a temperatura de ambas CPU's sitúanse no mesmo rango de valores. Como cada CPU ten 24 núcleos resulta lóxico ver como o load se sitúa neste valor.

Por cuestións de espazo e tempo, neste proxecto ímonos a dedicar a predicir os valores para unha única métrica e determinar as posibles anomalías que acontezan sobre ela. Esta métrica será **load** debido a que é a que menos variabilidade representa, ademais de ser unha das máis representativas dentro das anomalías atopadas no modelo clásico.

Na exploración dos datos obsérvase que o número de nodos e o número de pasos temporais son diferentes en cada traballo, polo que deberemos ter en conta este problema. Queremos un modelo que teña en conta todos os traballos, co obxectivo de non ter que definir un modelo por cada un.

Na seguinte táboa, poderemos ver os datos da métrica load para un traballo en particular. Vemos como se executa en dez nodos, onde os datos están remostreados a intervalos de 15 minutos. Como vemos, cada nodo representa unha serie temporal diferente dentro dun mesmo traballo:

	c7313	c7308	c6923	c7315	c7314	c7326	c7312	c7341	c7342	c6941
2018-02-24 22:00:00	21.906000	23.225334	21.482000	21.496667	21.527333	22.940000	21.912667	21.494667	21.488667	21.543333
2018-02-24 22:15:00	24.011333	24.002667	24.024000	24.024000	24.018000	24.026000	24.094000	24.029333	24.024667	24.026667
2018-02-24 22:30:00	24.014000	24.002667	24.006667	24.009333	24.006000	24.037333	24.006000	24.048000	24.013334	24.022667
2018-02-24 22:45:00	24.006000	24.003333	24.013333	24.008667	24.015333	24.055999	24.011333	24.061333	24.006667	24.012667
2018-02-24 23:00:00	24.014000	24.000000	24.020000	24.006000	24.010667	24.018000	24.017333	24.011333	24.004667	24.062000

Figura 5.2: Métrica load para un traballo en particular, con dez nodos

Na figura 5.3 móstrase as distintas métricas que teñen lugar nun nodo en particular, distribuídas xa por pasos:

	cpu1_temp	cpu2_temp	cpu_system	cpu_user	cpu_wait	load	mem_cached	mem_used	power
0	65.5	66.75	17.740054	82.235843	0.000333	23.904	322746777.6	4.158764e+09	310.5
1	65.0	66.80	18.467253	81.522986	0.000494	24.038	322963865.6	4.160768e+09	309.6
2	65.0	66.75	17.922315	82.105698	0.001347	23.992	323218636.8	4.158190e+09	324.0
3	65.0	67.00	17.824747	82.176027	0.001387	24.038	323507814.4	4.160465e+09	324.0
4	65.0	67.25	17.740183	82.243661	0.003213	24.036	379660697.6	4.155952e+09	319.5

Figura 5.3: Métricas para un nodo en particular

Tal e como se describiu no apartado do estado do arte, a entrada dunha rede recorrente é un vector de m dimensións. Neste problema, como se explicará máis adiante, a entrada corresponderase cun vector de 9 dimensións, onde cada dimensión é o valor de cada unha das métricas no paso actual (t). E a saída corresponderase cun vector de lonxitude 1, que será o valor no paso seguinte ($t+1$) da métrica que se queira predicir.

5.2. Pipeline dos datos

A continuación vaise a explicar o fluxo dos datos dende a súa orixe ata o final, durante o proceso de predición. A parte exploratoria dos datos e da parametrización do modelado farase no cluster Hadoop, que é dende onde podemos executar o Jyputer-Notebook. Co obxectivo de aproveitar as GPU's e acelerar o tempo de adestramento executarase un script en Python que lanzará dito modelo no Finisterrae sobre varios nodos, como un traballo máis.

Unha vez realicemos o adestramento, o modelo xerado gardarase nun ficheiro con formato HDF5, o cal almacenará a seguinte información: a arquitectura do modelo, permitindo a súa recreación; os pesos do modelo; parámetros de configuración (p.e: función de optimización); e o estado do modelo, permitindo a continuación do adestramento tendo en conta os novos traballos entrantes.

Polo tanto, definiremos dous pipelines diferentes, operativos nun entorno de produción: un correspondente ó adestramento e outro correspondente á predición.

5.2.1. Pipeline de adestramento

A primeira vez que se realiza o adestramento para a construción do modelo, extráense todos os traballos de HBase usando a API OpenTSDB. Estes datos son transformados e dados como entrada inicial a rede neuronal.

Unha vez temos o modelo construído, usámolo para realizar as predicións. A medida que os traballos novos vanse completando, estes vanse a usar para seguir adestrando o modelo. O pipeline deste proceso será o seguinte:

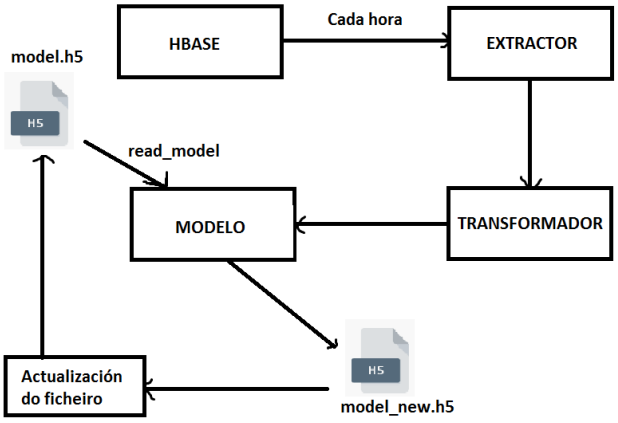


Figura 5.4: Pipeline da datos para o adestramento

1. Cada hora, un proceso denominado extractor encargarase de recoller os novos modelos xerados.
2. Un proceso transformador, transformará estes datos recollidos nunha matriz válida para envialas a rede neuronal e seguir co adestramento.
3. O modelo reconstrúese a partir do ficheiro correspondente. E mediante o datos enviados polo transformador, comeza o adestramento.
4. Unha vez termina o adestramento, escríbese nun novo ficheiro.
5. A medida que acontezan novas predicións, partirase do modelo recentemente creado. Unha vez o antigo modelo non sexa usado polas antigas predicións, este substituirase polo novo.

5.2.2. Pipeline de avaliación

Unha vez temos o modelo construído e gardado nun ficheiro, poderémolo usar para realizar as predicións dos novos traballos. Para cada predición crearase unha instancia do modelo, por se este fora actualizado.

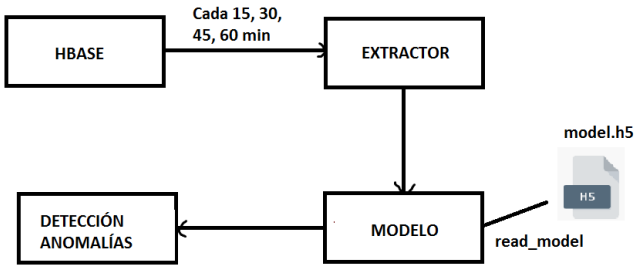


Figura 5.5: Pipeline da datos para a avaliación

1. Un proceso extractor, recolle os novos datos de cada unha das métricas para un traballo determinado. A frecuencia pola que recolle os datos ven determinado pola frecuencia de mostreo dos datos.
2. Estes datos recollidos, en conxunto cos datos dos pasos anteriores da serie úsanse para realizar a predición. En cada predición léese o modelo, por se este fora actualizado.
3. En cada predición realizada, analízanse os resultados obtidos para determinar se vai acontecer unha anomalía ou non.

Capítulo 6

MODELADO

6.1. Definición do proceso de modelado

Antes de comezar coa implementación dos diferentes modelos, vanse a definir os criterios que deben ter en común co obxectivo de establecer unha metodoloxía para predicir as anomalías.

Á hora de realizar este proxecto, non dispoñemos de información das anomalías ata agora acontecidas, polo que os modelos adestrados non disporán de dita información. Polo tanto, o primeiro que debemos de facer é atopar un modelo que nos prediga da forma máis precisa posible o seguinte valor de cada unha das series temporais. Posteriormente con este valor, mediante técnicas estadísticas, poderemos determinar se hai anomalías.

Cada traballo, dependendo da súa natureza e dos nodos sobre os cales se executa, xera valores nas métricas que poderían diferir con respecto a outros traballos. Por exemplo, o uso da CPU pode ser máis intenso nun traballo que noutro.

- Unha solución moi básica a este problema sería crear un modelo por traballo, coa consecuente perda de información. Con esta solución é probable que obteñamos unha mala estimación dos seguintes pasos, debido a que á hora de adestrar a rede non se adestrou cos suficientes datos.
- Outra solución algo máis complexa sería a de crear un modelo por tipo de traballo. Pero para isto deberíamos determinar o tipo de cada traballo mediante algún algoritmo de clasificación, cos consecuentes erros, e polo tanto, estimacións poucos precisas da serie. Ademais, cada novo traballo deberíamos de avalialo para ver en que categoría se clasifica mellor, o cal supón un atraso á hora de realizar a predición.
- Co obxectivo de ter un modelo por métrica, que cubra todos os traballos, o que deberemos de facer é **buscar unha maneira de adestrar a rede neuronal con segmentos de secuencia**. Cada segmento terá un tamaño igual á serie temporal máis longa de entre todos os traballos (tamaño determinado pola variable BATCH_SIZE), onde cada segmento representará a serie temporal dun nodo correspondente a un traballo en particular.

Co obxectivo de avaliar, debemos de ter en conta as seguintes consideracións:

- Os sensores producen datos cada 10 segundos, polo que deberemos de realizar o *resampling* dos mesmos. Probarase con intervalos de 30 e 60 minutos.
- Distintos modelos cambiando o valor dos hiperparámetros: número de capas ocultas, número de neuronas por capa e o número de épocas.
- Á hora de adestrar o modelo, colleranse os traballos dos meses de maio e xuño, que teñan unha duración mínima dun día. Isto faise así para que se teña un número mínimo de valores por cada serie que forme a entrada da rede.

6.2. Transformación dos datos

As redes neuronais son sensibles á escala dos datos de entrada, polo que deberemos de normalizar os datos. Para iso, usaremos a clase MinMaxScaler da libraría scikit-learn. Antes de normalizar os datos, separámolos en dous conxuntos: 80 % adestramento e 20 % test.

O obxectivo é buscar unha maneira de adestrar a rede neuronal con segmentos de secuencia, onde cada segmento correspóndese a unha serie temporal completa dun nodo. O tamaño deste segmento denominárase BATCH_SIZE (será igual o tamaño da serie máis longa de todos os traballos). Para que a entrada da rede sexa válida, o tamaño de todas a series temporais do conxunto de adestramento debe ser o mesmo.
Polo tanto, o seguinte paso será transformar todas as series de forma que todas teñan un tamaño igual ó BATCH_SIZE. Para cubrir o tamaño, engadiranse tantos ceros como sexan necesarios ó final de cada serie.

A entrada das redes debe ser unha matriz da forma: **[número de mostras, pasos de tempo, número de características]**: o número de mostras correspóndese co BATCH_SIZE definido, os pasos ó número de secuencias cara atrás do paso actual para predicir o paso seguinte e as características son as métricas usadas para predicir a saída (como se comentou, o valor será 9).

Como se describiu no apartado da exploración dos datos, a entrada da rede corresponderase cun vector de 9 dimensións, onde cada dimensión é o valor de cada unha das métricas no paso actual (t). Na figura 6.1, poderemos ver o resultado de transformar os datos nunha matriz válida para a rede neuronal. Neste proceso, tamén se xeran varios conxuntos (trainX, trainY, testX e testY).

Neste exemplo, por cuestións de espazo, móstrase a entrada da rede correspondente ós tres primeiros pasos dun nodo en particular. O resultado de transformar os datos é unha matriz da forma [3,2,9], de maneira que os datos que se lle pasan a rede son os correspondentes ó paso actual t e todos os anteriores acontecidos. Polo tanto, cada elemento do array correspóndese a un paso temporal (cadros vermellos), e dentro de cada elemento hai outras dúas secuencias (definidas pola variable LOOK_BACK), correspondentes á secuencia actual e á secuencia anterior (cadros azuis), para predicir o valor Y do paso seguinte (cadros verdes). Cada valor da secuencia correspóndese a unha métrica, tendo neste caso nove métricas de entrada:

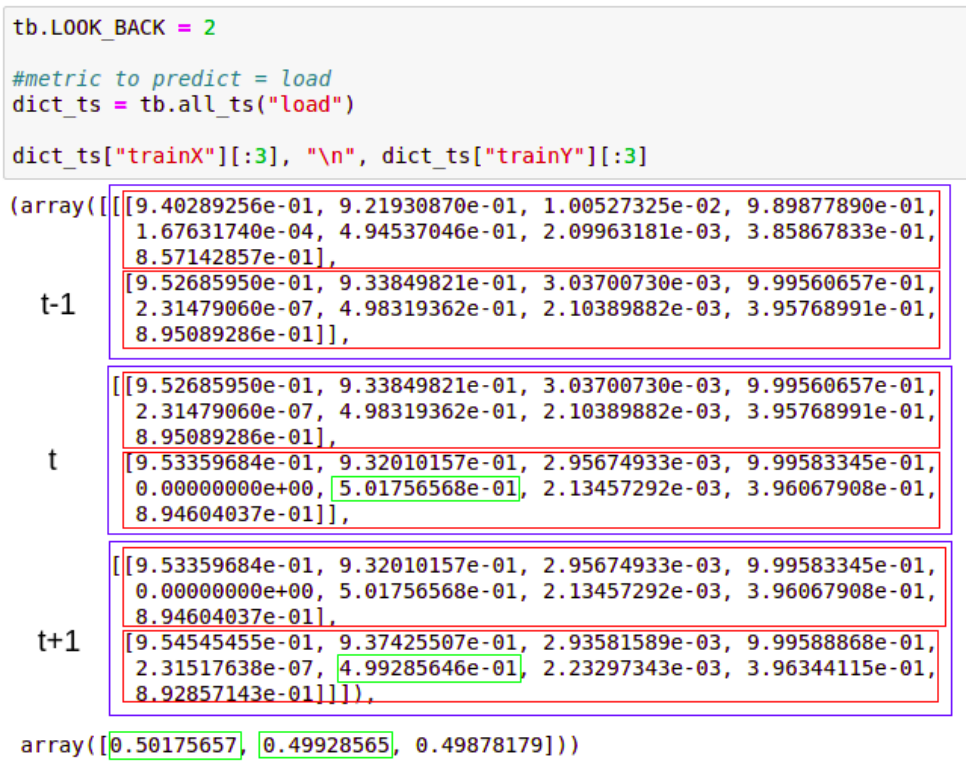


Figura 6.1: Matriz da forma [3, 2, 9]

6.3. Consideracións pre-modelado

Os problemas onde se intentan predicir o valor dos seguintes pasos dunha serie temporal e onde se adestran usando un batch_size superior a un, é necesario especificar na rede o parámetro **stateful** igual a True, para que o estado final de cada batch se pase ó estado inicial do seguinte batch e manter desta forma o estado. Isto é necesario onde os batches están relacionados de maneira secuencial.
Neste problema, a entrada da rede corresponderase a tódalas series temporais, de tódolos nodos, de tódolos traballos do conxunto de adestramento. Polo tanto, no momento de adestrar, será necesario dividir toda esta entrada en segmentos, onde cada segmento corresponderase a unha serie temporal completa (Tamaño do segmento = BATCH_SIZE). Como cada batch é independente do resto dos segmentos, haberá que especificar na rede o parámetro stateful igual a False para que non se manteña o estado entre series diferentes.

Outro parámetro a ter en conta é o **shuffle**. Este parámetro o que fai é coller todos os valores correspondentes a un batch e barallalos. Como este é un problema onde a predición dun valor depende da secuenciación dos valores pasados é requisito establecelo a False.

Co obxectivo de prever o sobre-aprendizaxe, no método de adestrar a rede, pasámoslle un callback predefinido chamado **EarlyStopping**. Este callback o que fai é cortar o aprendizaxe unha vez a precisión de validación comeza a decrecer, polo que é probable que o adestramento non chegue ó número de épocas establecido.
Veremos esta explicación de maneira máis sinxela coa figura 6.2:

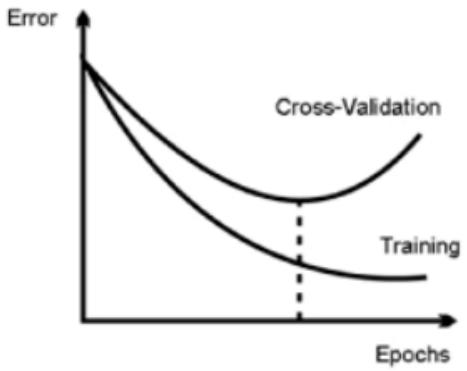


Figura 6.2: Acción do callback EarlyStopping

A continuación amósase como se chamaría ó método de adestramento tendo en conta todas estas aclaracións. O parámetro *patience* indica cantas épocas deben de pasar sen que mellore o MSE antes de parar o adestramento. Como vemos, establécese tamén un split de validación de 0.33:

```
cbs = [History(), EarlyStopping(monitor='loss', patience=10, min_delta=0.0003, verbose=0)]
for i in range(EPOCHS):
    model.fit(train_X, train_y, epochs=1, batch_size=train_basic.BATCH_SIZE, verbose=0, shuffle=False, callbacks=cbs,
              validation_split=0.33)
```

Para comparar os resultados entre os distintos modelos, usaremos o erro cuadrático medio (MSE).

6.4. Modelado

A continuación realizaremos a definición de varios modelos, co obxectivo de avalialos e escoller aquel que mellor se adapte ó problema, tendo especial coidado co sobre-aprendizaxe.
Á hora de adestrar, só teremos en conta un número limitado de traballos por razóns de tempo: seleccionáronse aqueles cunha duración superior a un día, dos meses de maio e xuño (1750 traballos aprox.).

Para cada modelo, amosaremos a variación do MSE usando diferente números de neuronas. Ademais, adestraremos usando diferentes valores do LOOK_BACK (1,2 e 3). Por cuestións de espazo, só mostrarei as gráficas para valores de 1 e 2, obtendo erros moi similares ou peores nalgúns casos con un LOOK_BACK=3. Como se comentou no apartado 6.2, o LOOK_BACK indica o número de secuencias cara atrás que serán incluídas na entrada.

6.4.1. Modelo cunha única capa

O modelo contará cunha única capa LSTM/GRU. Na figura 6.3 poderemos ver unha comparativa de cada modelo usando diferentes números de neuronas. :

```
('Mellor resultado con 8 neuronas: ', 0.0009581964652606306, 'na epoca ', 99)
('Mellor resultado con 16 neuronas: ', 0.0009457199055117379, 'na epoca ', 99)
('Mellor resultado con 32 neuronas: ', 0.0009425942858975625, 'na epoca ', 95)
('Mellor resultado con 64 neuronas: ', 0.0009418545546833929, 'na epoca ', 97)
('Mellor resultado con 128 neuronas: ', 0.000944953089569105, 'na epoca ', 99)
```

```
['Mellor resultado con 8 neuronas: ', 0.0009520492840841266, 'na epoca ', 99)
['Mellor resultado con 16 neuronas: ', 0.0009441464729493356, 'na epoca ', 93)
['Mellor resultado con 32 neuronas: ', 0.0009458098587095305, 'na epoca ', 95)
['Mellor resultado con 64 neuronas: ', 0.0009433236388659304, 'na epoca ', 99)
['Mellor resultado con 128 neuronas: ', 0.000944727870794833, 'na epoca ', 98)
```

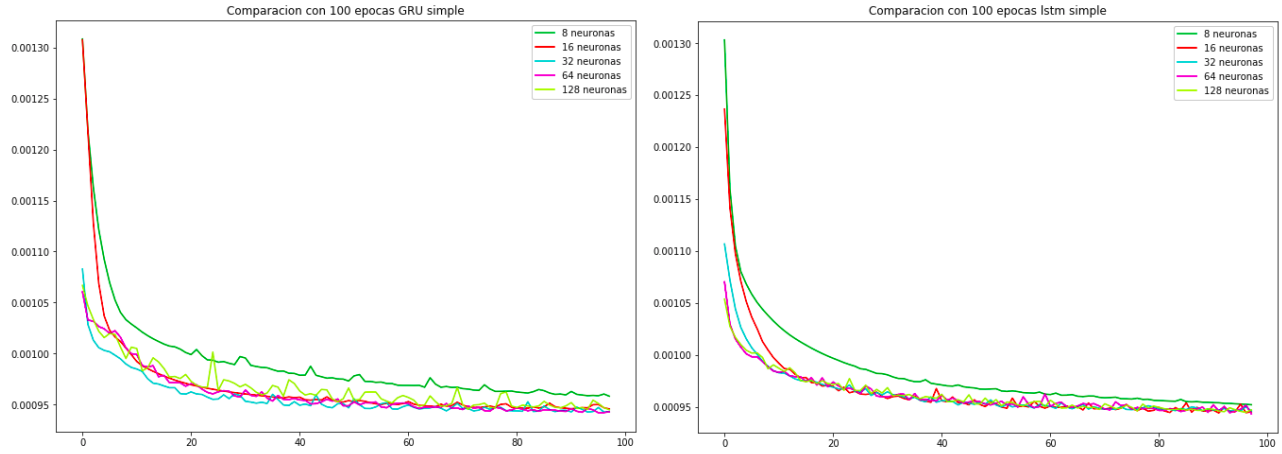


Figura 6.3: Comparación do MSE para LSTM e GRU (LOOK_BACK=1)

A simple vista, podemos observar como tódolos modelos de ambas redes converxen en poucas épocas, sendo o que máis tarda o de oito neuronas. Obsérvase tamén que ambas redes dan un erro parecido para todas as neuronas.

A continuación amosaremos as mesmas gráficas, pero adestrando a rede pasándolle como entrada a combinación de secuencias do paso actual e do paso anterior, é dicir, con un LOOK_BACK = 2. Polo tanto, a entrada da rede é unha matriz da forma [BATCH_SIZE, 2, 9]. A simple vista, parecen dar un erro moi similar o da figura 6.3.

```
('Mellor resultado con 8 neuronas: ', 0.0009398587203804576, 'na epoca ', 99)
('Mellor resultado con 16 neuronas: ', 0.0009378171225410591, 'na epoca ', 97)
('Mellor resultado con 32 neuronas: ', 0.000936940482874629, 'na epoca ', 93)
('Mellor resultado con 64 neuronas: ', 0.0009380210308772468, 'na epoca ', 97)
('Mellor resultado con 128 neuronas: ', 0.0009372866692719737, 'na epoca ', 98)
```

```
('Mellor resultado con 8 neuronas: ', 0.0009349442947779976, 'na epoca ', 91)
('Mellor resultado con 16 neuronas: ', 0.000926875040138377, 'na epoca ', 94)
('Mellor resultado con 32 neuronas: ', 0.0009333408911373081, 'na epoca ', 92)
('Mellor resultado con 64 neuronas: ', 0.0009320117099983372, 'na epoca ', 99)
('Mellor resultado con 128 neuronas: ', 0.0009316498849413032, 'na epoca ', 97)
```

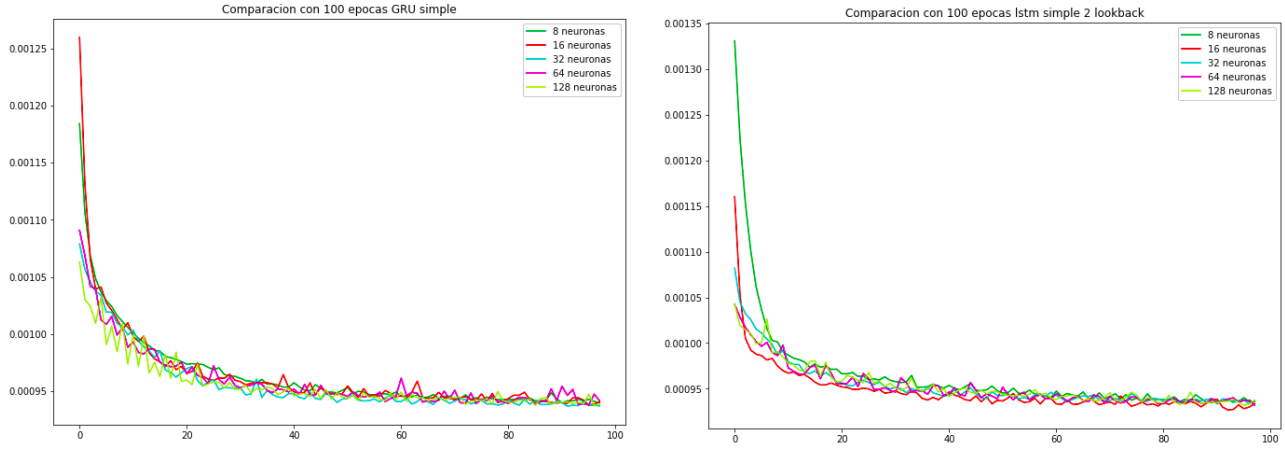


Figura 6.4: Comparación do MSE para LSTM e GRU (LOOK_BACK=2)

6.4.2. Modelo con múltiples capas con Dropout

A capa Dropout é un método que desactiva un número de neuronas de maneira aleatoria. En cada iteración, esta capa desactivará diferentes neuronas, de forma que estas neuronas desactivadas non se toman en conta nin para o forward propagation nin para o backward propagation. Polo tanto, esta capa, obriga ás neuronas máis próximas a non depender tanto das neuronas desactivadas, reducindo desta forma o sobre-aprendizaxe [12].

Este parámetro toma valores entre 0 e 1. Se os valores son próximos a 0 esta capa desactivará menos neuronas. Pódese poñer como a capa de entrada da rede, entre capas ocultas, pero nunca na capa de saída, xa que neste capa é necesario que todas as neuronas estean activas.

Este modelo executouse con dúas, tres e catro capas, onde poderemos apreciar un erro moito maor que con unha soa capa (Sección 6.4.1), e onde vemos que o erro aumenta según aumenta o número de capas. A primeira capa terá o número de neuronas que se indica nas etiquetas da lenda, mentres que na segunda será a metade da primeira, e a terceira e cuarta a metade da segunda (Ex: 64-32-16-16; 32-16-8-8)

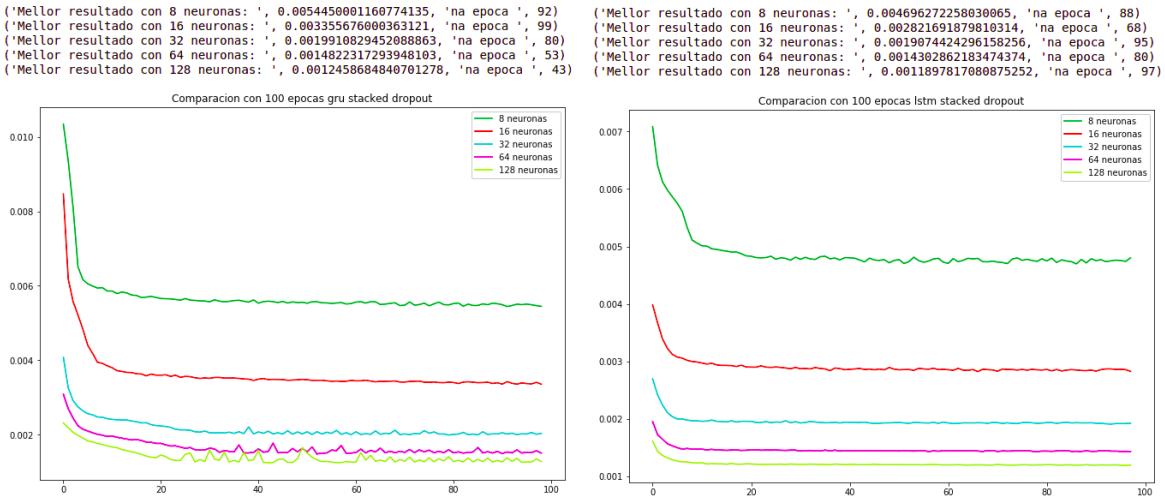


Figura 6.5: Comparación do MSE con dúas capas (LOOK_BACK=1)

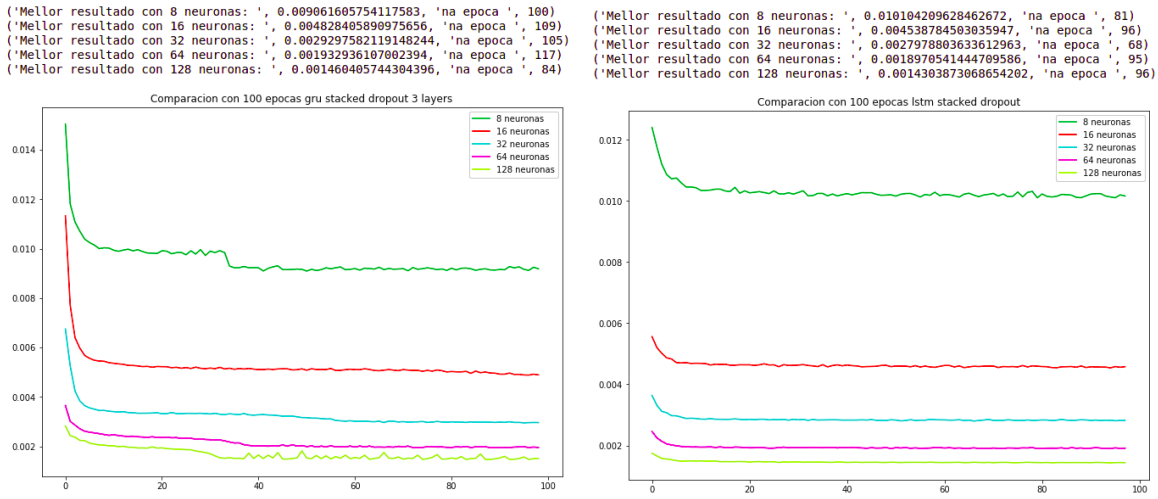


Figura 6.6: Comparación do MSE con tres capas (LOOK_BACK=1)

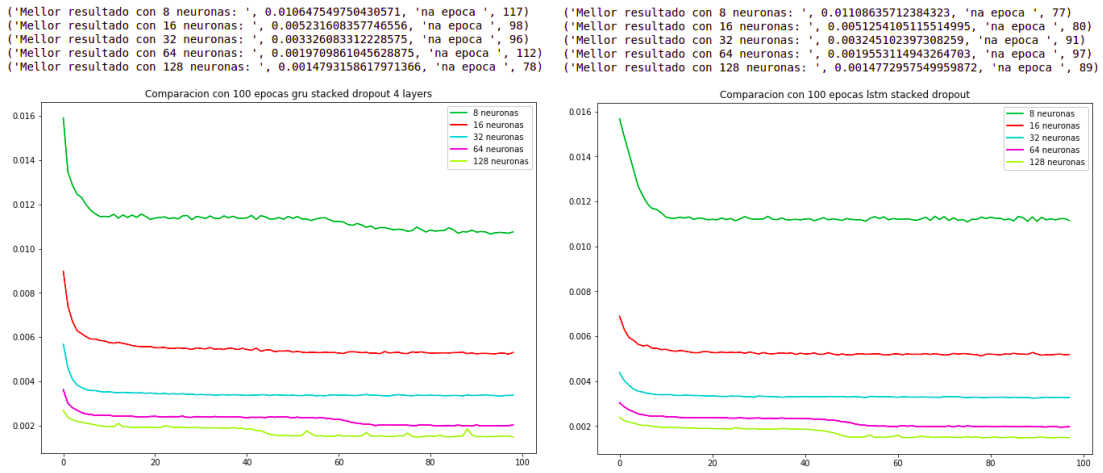


Figura 6.7: Comparación do MSE con catro capas (LOOK_BACK=1)

Concluímos que o número de neuronas que mellor se adapta para estes modelos é de 128, obtendo un erro aproximado de 0.0012 no mellor dos casos (cunha capa arredor de 0.00094 - 25 % menor). Como vemos na figura 6.8 , o mellor son dúas capas LSTM e 128 neuronas. Tamén se pode ver como o GRU traballa peor que o LSTM en dúas e tres capas, obtendo un resultado parecido en catro.

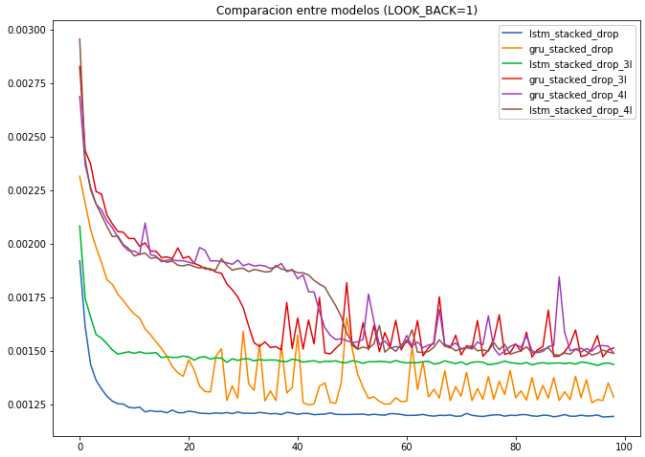


Figura 6.8: Comparación dos mellores modelos apilados (LOOK_BACK=1)

Agora ímos repetir o mesmo proceso pero cun LOOK_BACK igual a dous. Tamén obtemos que en tódolos modelos, o número de neuronas que mellor resultado presenta é de 128. Obsérvase tamén, que para este caso, ó igual que para unha capa, non se aprecia moita variación do erro entre ambos valores de LOOK_BACK:

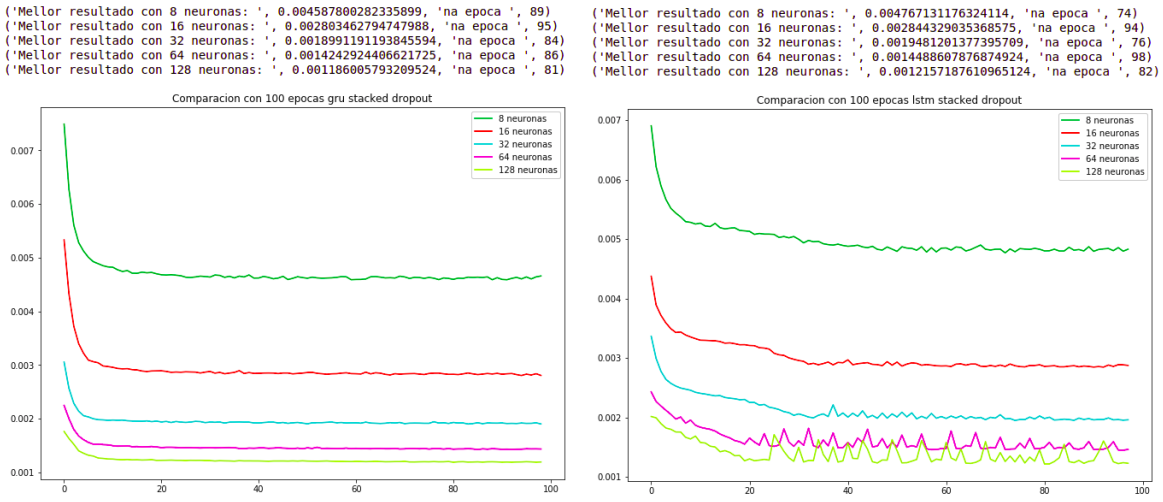


Figura 6.9: Comparación do MSE con dúas capas (LOOK_BACK=2)

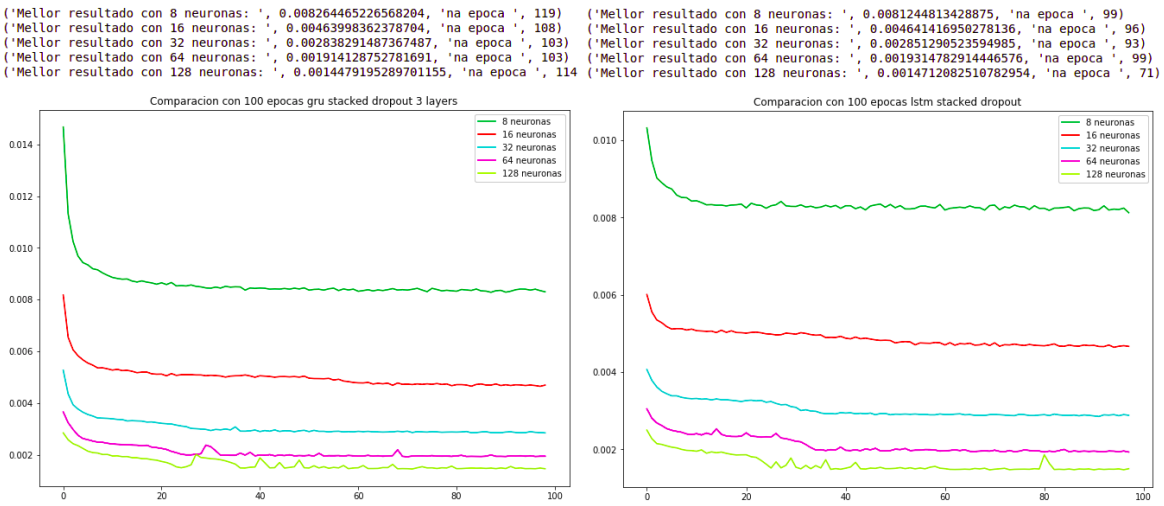


Figura 6.10: Comparación do MSE con tres capas (LOOK_BACK=2)

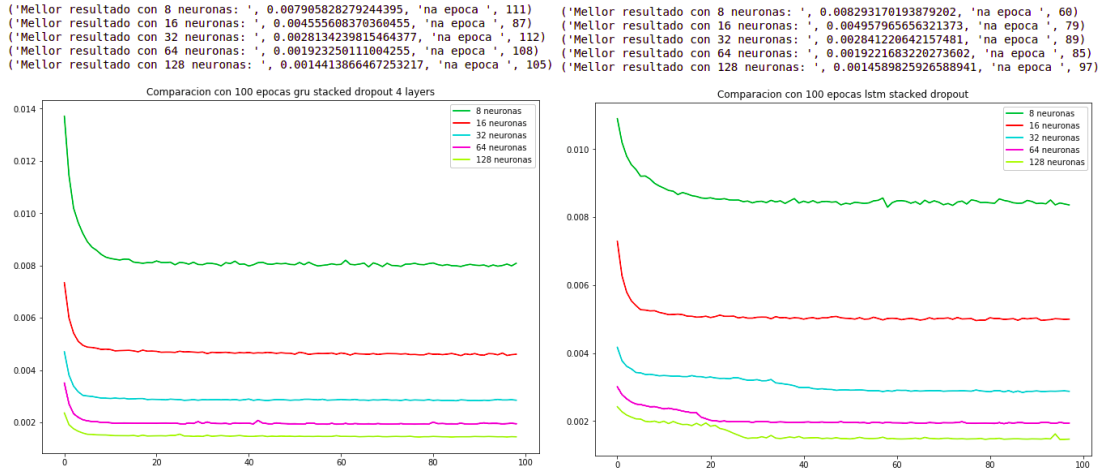


Figura 6.11: Comparación do MSE con catro capas (LOOK_BACK=2)

Como vemos na figura 6.12, a diferencia con respecto a LOOK_BACK igual a un, é que o que mellor resultado presenta é o da rede GRU con dúas capas, obtendo un erro moi similar dado na figura 6.15. Tamén vemos como apenas hai variacións de erro adestrando con tres e catro capas.

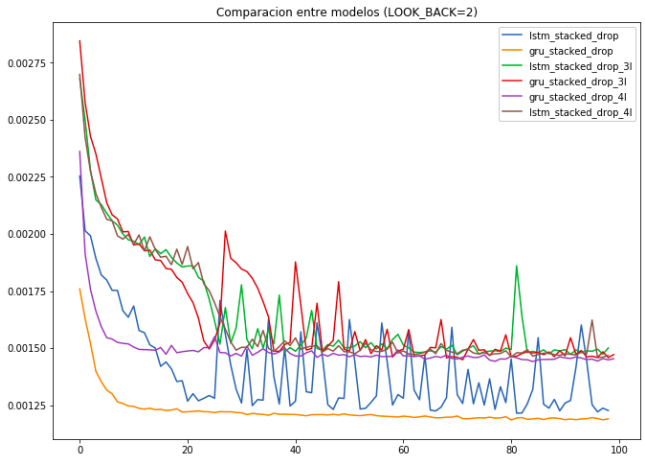


Figura 6.12: Comparación dos mellores modelos apilados (LOOK_BACK=2)

6.4.3. Modelo con múltiples capas con Batch normalization

Cando normalizamos os datos de entrada, solo a capa de entrada benefíciase de isto. Conforme os datos pasan por outras capas ocultas, esta normalización vaise perdendo e se temos unha rede neuronal con moitas capas podemos ter problemas co adestramento. Esta capa normaliza os datos antes de cada capa oculta, de forma que sempre teremos os datos normalizados [13].

Como comprobamos no modelo anterior, cantas máis capas, peor resultado. Polo que este modelo vaime a dar peores resultados, xa que en teoría, provoca mellores resultados cantas máis capas teña a rede. Amosamos a variación do erro nunha rede LSTM e tres capas, e observamos como o erro resultante é aproximadamente dez veces peor que co modelo simple:

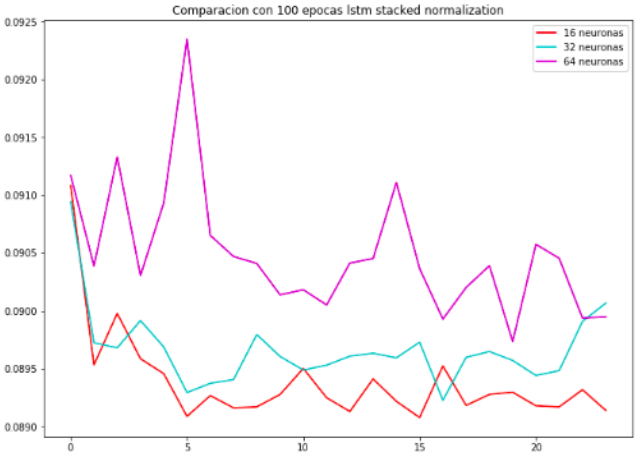


Figura 6.13: Comparación do MSE para LSTM con batch normalization

6.4.4. Modelo con autoencoders

Os autoencoders son redes neuronais co obxectivo de xerar novos datos, primeiro comprimindo a entrada nun espazo de variables útiles e despois reconstruíndo a saída en base á información recollida. Co uso de autoencoders poderemos reducir o ruído e a dimensionalidade das variables de entrada [11].

Os autoencoders nas redes LSTM son unha implementación da arquitectura Encoder-Decoder, onde primeiramente se codifica a entrada nun vector de lonxitude fixa e posteriormente decodifícase este vector para ser a entrada da capa LSTM/GRU. Por exemplo, na seguinte imaxe vemos esta implementación:

```
# design network
model = Sequential()
model.add(GaussianNoise(0.01, input_shape=(train_X.shape[1], train_X.shape[2])))

model.add(LSTM(n_neuronas[0], activation=activation, name="encoder"))
model.add(RepeatVector(train_X.shape[1]))
model.add(LSTM(n_neuronas[0], activation='relu', return_sequences=True, name="decoder"))
```

Na figura 6.14 poderemos observar como varía o erro según o número de neuronas, cun valor de LOOK_BACK igual a 1. É moi similar ó erro obtido co modelo simple, e vemos como para calquer número de neuronas, o modelo converxe en poucas épocas. Ó igual que sucede en tódolos casos estudados ata agora, o mellor erro obtido é moi similar para os diferentes números de neuronas analizadas:

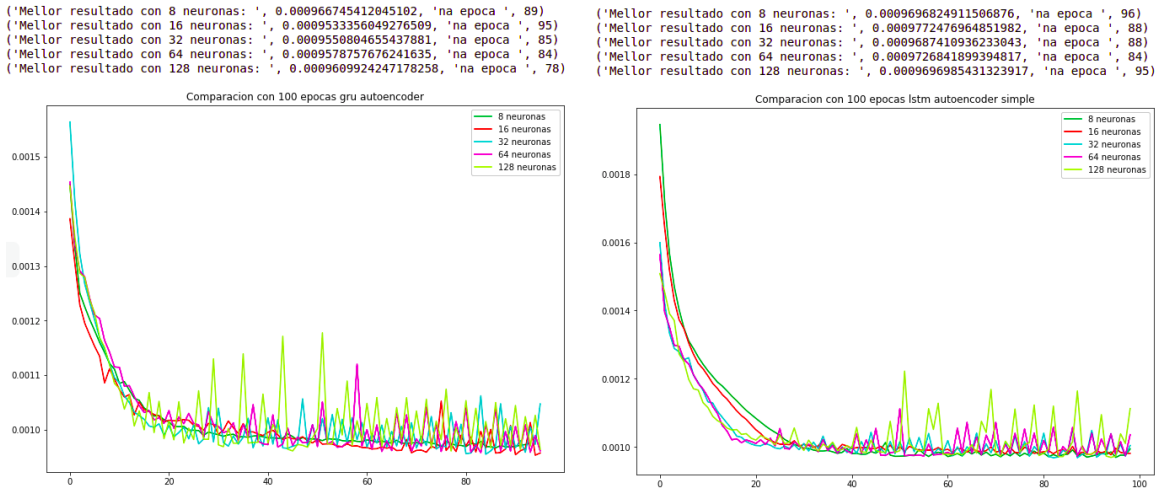


Figura 6.14: Comparación do MSE para LSTM e GRU (LOOK_BACK=1)

Con LOOK_BACK igual a dous (Figura 6.15) vemos que obtemos un erro moi similar que coa figura 6.14.

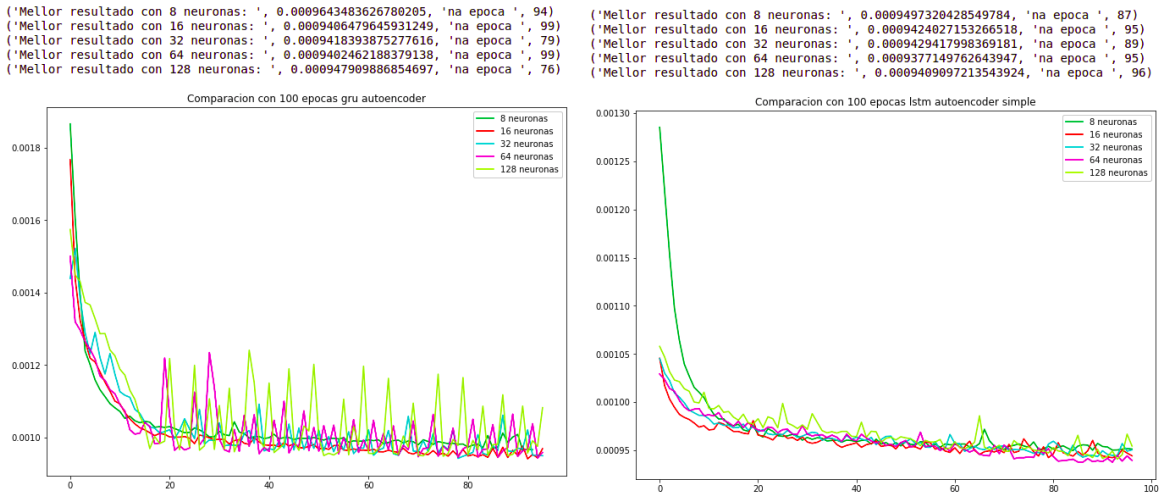


Figura 6.15: Comparación do MSE para LSTM e GRU (LOOK_BACK=2)

6.5. Selección do modelo

Tendo en conta o valor do MSE dado, escolleremos a opción co menor número de capas e co menor número de neuronas, co obxectivo de evitar o sobre-aprendizaxe. Nas seguintes dúas figuras, poderemos ver a comparación do MSE entre os diferentes modelos contemplados, para os dous valores de LOOK_BACK documentados:

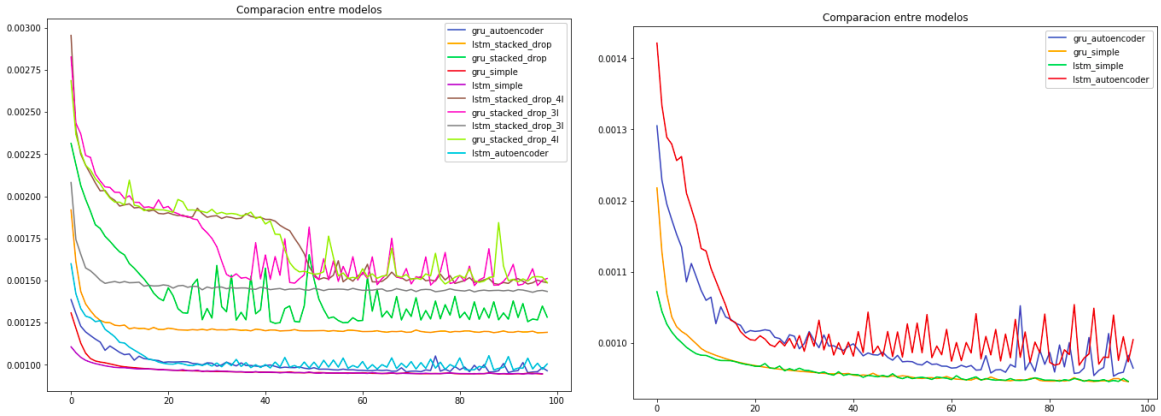


Figura 6.16: Comparación do MSE entre varios modelos (LOOK_BACK=1)

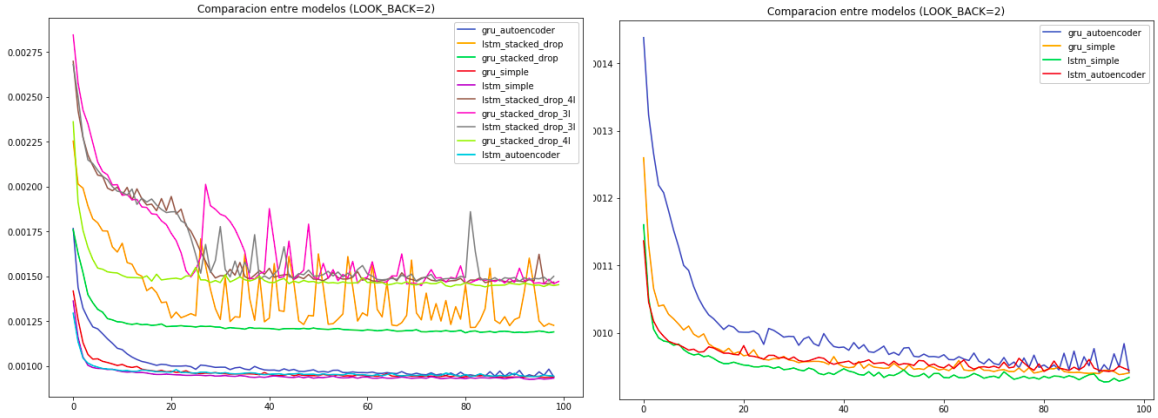


Figura 6.17: Comparación do MSE entre varios modelos (LOOK_BACK=2)

Na gráfica da esquerda vemos como os peores modelos sen dúbida son aqueles que teñen dúas ou máis capas. Na figura 6.16 os modelos que mellor resultado presentan son o gru e lstm simples, mentres que na figura de abaixo, practicamente os catro modelos presentan os mesmos valores de MSE, sendo lixeiramente mellor o da lstm simple.

Podemos concluír, que practicamente tódolos modelos apenas presentan variacións en canto o número de LOOK_BACK. Nos casos onde o erro mellora moito a medida que aumentamos o número de neuronas, é cando o número de capas é maior a un. Cando o número de capas é igual a un, como ocorre na gráfica da dereita da ambas figuras, o aumento de neuronas apenas presenta mellora no resultado, polo que á hora de escoller un modelo, escolleremos aquel que menor número de neuronas presente dentro dos mellores resultados, co obxectivo de prever o sobre-aprendizaxe.

Finalmente, amósanse un par de series temporais correspondentes á métrica **load**, onde poderemos ver a estimación producida por tres dos modelos estudados. Debido a que se adestrou a rede usando traballos de natureza diferente, onde as series temporais das métricas teñen un comportamento sen unha tendencia marcada (como ocorre coas series temporais da bolsa), a predición parece depender moito do valor anterior acontecido, tal e como podemos observar nas seguintes figuras:

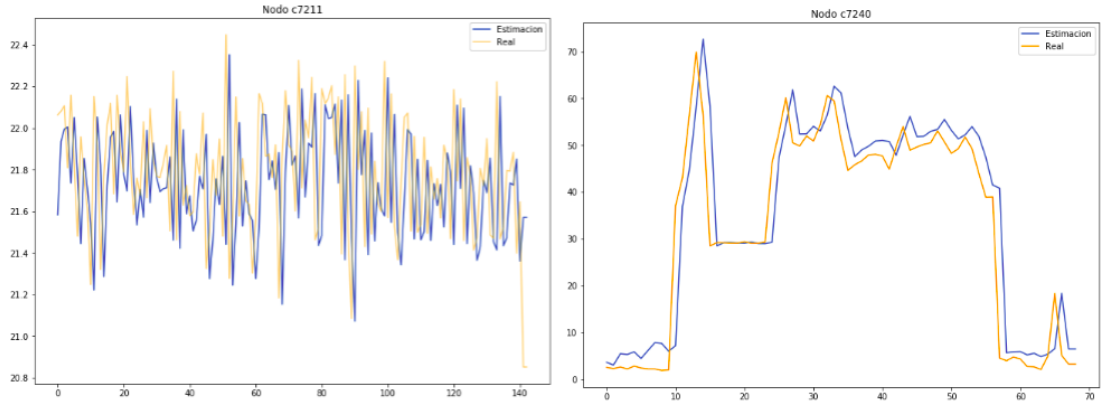


Figura 6.18: Estimación usando o modelo gru simple cun LOOK_BACK = 1

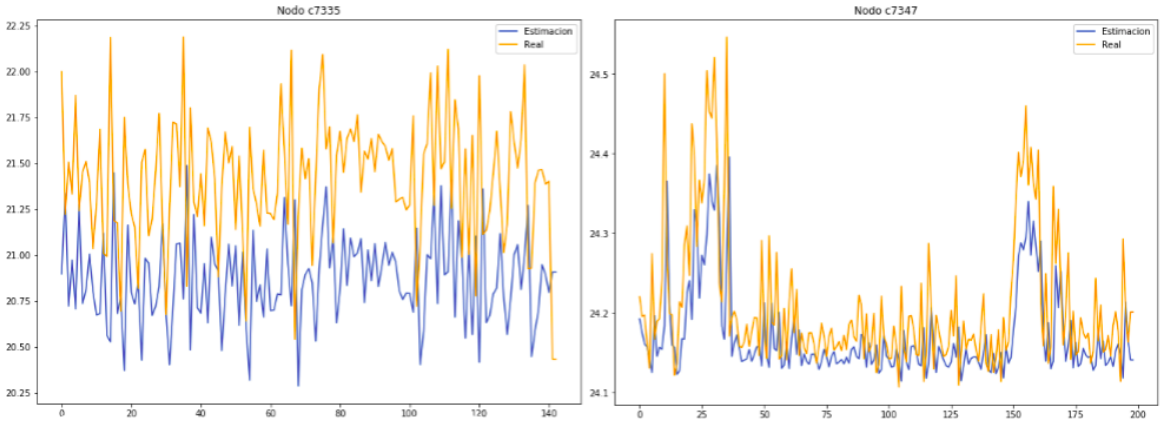


Figura 6.19: Estimación usando o modelo lstm con dúas capas e LOOK_BACK = 1

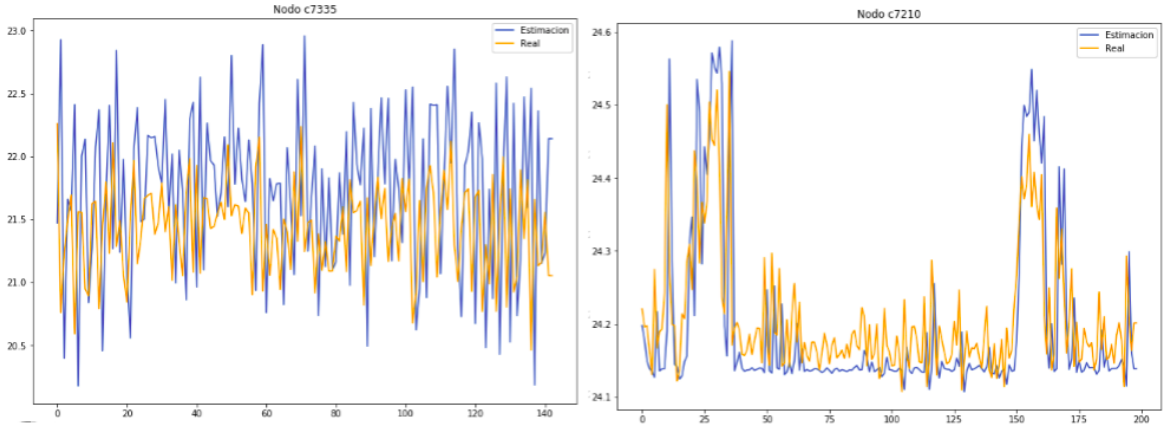


Figura 6.20: Estimación usando o modelo lstm con catro capas e LOOK_BACK = 2

Capítulo 7

AVALIACIÓN E COMPARATIVA

O obxectivo principal é concluír se se vai a producir unha anomalía nos seguintes puntos no que se atopa a serie temporal en cuestión, ou ben obter unha lista coas anomalías acontecidas nun grupo de traballos. Por isto, o que imos facer en primeiro lugar será predicir os seguintes catro puntos usando o modelo escollido (gru simple).

O que faremos para logralo, será chamar catro veces á función *predict* do modelo, poñendo como último valor de entrada o valor predito da anterior chamada. No primeiro caso, o último valor de entrada corresponderase co último valor real, que o temos.

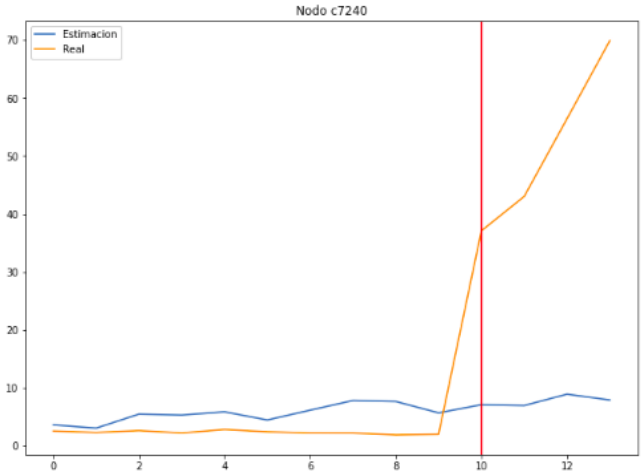


Figura 7.1: Estimación dos seguintes catro valores da métrica load nun nodo en particular

Como vemos na figura 7.1, o punto actual da serie é o 9, polo que queremos predicir os seguintes catro puntos (Cada punto é un paso de media hora). Obsérvase que a predición dista moito do real.

A continuación, aplicaranse técnicas estadísticas para determinar se se van producir anomalías a partir deste intervalo de catro puntos [14]. En primeiro lugar, calculo as seguintes variables:

- **std1**: Desviación típica dos catro puntos anteriores ó actual da serie real.
- **std2**: Desviación típica dos catro puntos anteriores ó actual da serie predita.
- **std3**: Desviación típica dos catro puntos preditos.
- **std4**: Desviación típica dos catro puntos reais correspondentes ó predito.
- **std5**: Desviación típica global da serie, ata o punto actual.
- **diff_real**: abs(std1-std2).
- **diff_pred**: abs(std3-std4).
- **res**: abs(diff_real-diff_pred).

Primeiro, calculo as desviacións típicas de varios intervalos (std1, std2, std3, std4). Posteriormente, obteño a diferencia entre as desviacións de cada intervalo da serie (diff_real, diff_pred) e obteño a desviación entre o real e o predito dentro deses intervalos (res). Finalmente comparo este valor con tres veces o valor da desviación global (3 x std5). Se este valor é maior, catalógase como anomalía.

A continuación, executamos o modelo conxunto co descrito nesta parte, sobre un traballo que sabemos que é anómalo. E obtemos oito intervalos con anomalías:

Figura 7.2: Detección dos intervalos onde teñen lugar posibles anomalías

A continuación, xeraremos unha lista dos traballos anómalos acontecidos dende o 1 de xullo ata o 15 de xullo. A xeramos usando tanto o noso sistema (esquerda) como o método de machine learning clásico actual (dereita).

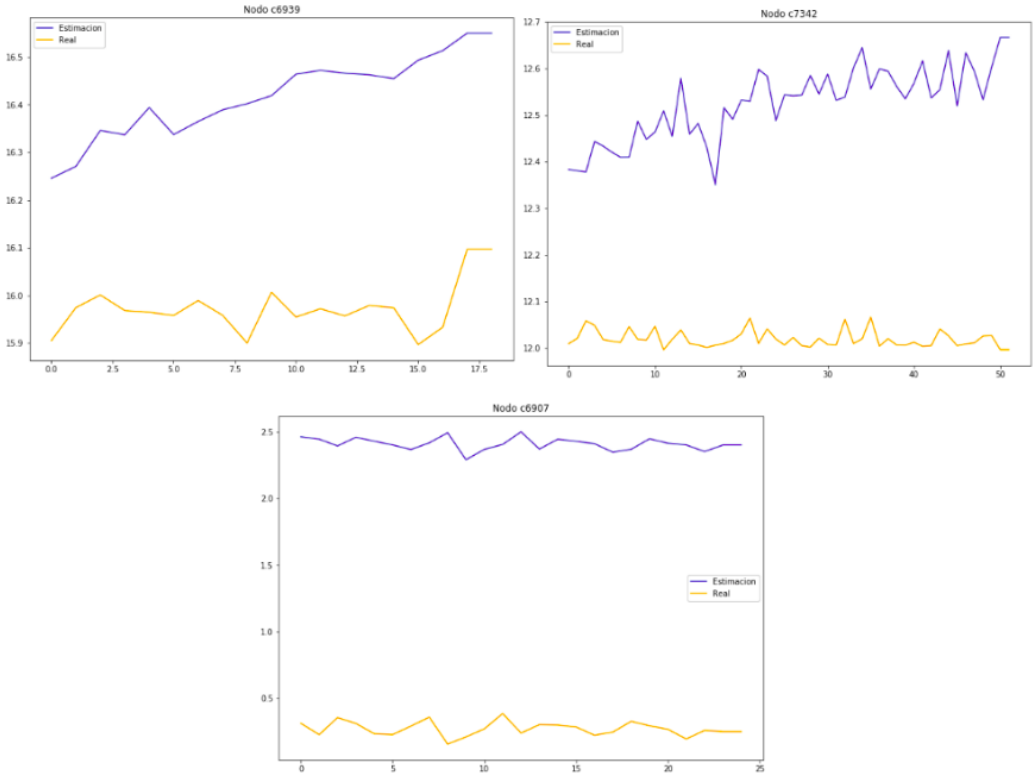
JOB_ID	NODO/S
2844601	c6902
2865229	c7207
2879120	c7345
2814723	c6622
2814717	c6946
2814725	c7244
2834539	c7332
2830538	c7308
2869397	c6925
2869399	c6935
2873757	c7319

JOB_ID	NODO/S
2844601	c6902
2865229	c7207
2879120	c7345
2864310	c7342
2873761	c7248
2812043	c6907
2814156	c6625
2848714	c7319
2856186	c6933,c7233
2856187	c6615
2869615	c7248
2869616	c6637
2831764	c6943
2853824	c7213,c7342
2869587	c7321
2883609	c7341,c6924
2883611	c7342,c6905
2841870	c7316
2841872	c7211
2809484	c6613
2809501	c7348
2812363	c7340

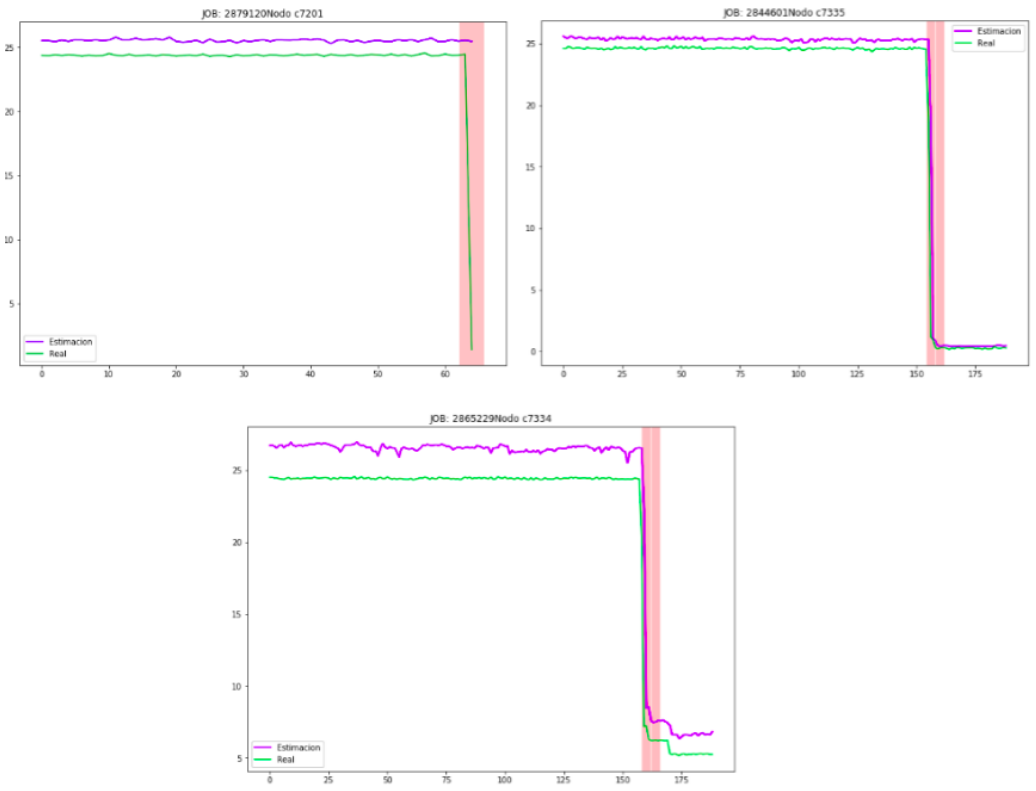
Figura 7.3: Lista de traballos anómalos da miña solución (esquerda) e do método clásico (dereita)

Como podemos comprobar, só obtemos tres traballos preditos con ambos métodos (Os tres primeiros). Os outros traballos que xeran o noso sistema contempla casos onde os valores da carga se van a 60, e parece ser que o modelo clásico xa non ten en conta estes traballos, polo que é razoable que non estean na lista.

En canto os traballos que aparecen na dereita e non están na esquerda, van ser analizados, co obxectivo de ver en que puido fallar o noso modelo. Pódese observar como a serie real dos tres analizados non parecen presentar ningunha anomalía en canto á variación. O único que difire é o valor da carga, sendo moi baixo nos tres casos, xa que o normal son valores en torno a 24 (Cada nodo ten 24 cores). Polo tanto, en todos estes casos, non se aproveita como se deben ditos cores e parece que o sistema clásico detecta esta comportamento.



E finalmente mostro a gráfica das anomalías atopadas que concordan en ambos conxuntos:



Podemos concluír polo tanto que o noso sistema atopa anomalías, e o que considera anomalía é realmente unha anomalía, sen producir falsos positivos. Sen embargo, non o podemos comparar directamente co modelo clásico actual, porque hai máis condicións que considera anomalía, como que a carga baixe dun determinado valor.

7.1. Conclusións e posibles melloras

O problema principal das predicións dos modelos e deste proxecto en xeral é a diversidade en canto a natureza dos traballos. Isto complicou bastante a forma de considerar a construción dos modelos xa que debíamos ter un único modelo para todos os traballos. Ademais debemos sumar a contra que supón adestrar series cunha tendencia nada marcada, sendo un comportamento bastante aleatorio. O gran problema que atopamos con isto foi que o modelo non foi capaz de aprender patróns, e a predición do seguinte punto era bastante dependente do punto anterior, polo cal acababa axustando bastante ben, pero sen ser capaz de adiantarnos ó que vai a acontecer.

Polo tanto, non fomos capaces de predicir se vai acontecer anomalías no seguinte intervalo, aínda que si podemos dicir que se produciron unha vez temos os datos reais.

Polo que se analizou, foi bastante preciso en canto á procura das anomalías, xa que a taxa de acertos foi bastante alta. En canto ó recall, non foi posible analizalo, debido a que o modelo clásico actual consideraba máis motivos que as variacións dos valores, atopando máis anomalías das que poderiamos atopar coa nosa solución.

Como se analizou no modelado, modelos con moitas neuronas e varias capas (Deep Learning) produciron peores resultados frente a modelos máis sinxelos con poucas neuronas e unha capa.

En canto as posibles melloras, como as series que teñen anomalías representan un porcentaxe moi baixo dos traballos, a rede non é capaz de aprendelas. Unha forma de mellorar isto, sería simulando máis traballos deste tipo, co obxectivo de que a rede comece tendo en conta este tipo de comportamentos.

Outra forma de mellorar o resultado sería facéndolle saber a rede a existencia dos outros nodos do mesmo traballo. Desta forma poderíamos indicar esa dependencia co obxectivo de predicir mellor os puntos da serie.

Finalmente, outra mellora, sería coa predición do modelo dos seguintes 4 puntos. Na nosa solución só se predice o seguinte valor, e executamos a predición catro veces. Desta forma, podería aprender mellor os patróns de comportamento e predecir por tanto se no seguinte intervalo de tempo se vai a producir algunha anomalía, aínda que a precisión da serie estimada fose moito menor.

Bibliografía

[1] https://www.researchgate.net/publication/329891470_DEEP_LEARNING_IN_INDUSTRY_40_-_BRIEF

[2] https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57

[3] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[4] https://docs.microsoft.com/es-es/azure/machine-learning/team-data-science-process/overview

[5] https://github.com/FrancisArgnR/SeriesTemporalesEnCastellano

[6] http://www.diegocalvo.es/red-neuronal-recurrente/

[7] https://medium.com/implodinggradients/tensorflow-or-keras-which-one-should-i-learn-5dd7fa3f9ca0

[8] PMBOK 6.0

[9] https://vincentblog.xyz/posts/dropout-y-batch-normalization

[10] https://towardsdatascience.com/machine-learning-for-anomaly-detection-and-condition-monitoring-d4614e7de770

[11] https://keras.io/getting-started/functional-api-guide/

[12] https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

[13] https://machinelearningmastery.com/how-to-accelerate-learning-of-deep-neural-networks-with-batch-normalization/

[14] http://www.cs.columbia.edu/ lierranli/publications/TSW2017_paper.pdf