Universidad Autónoma de Madrid
Escuela politécnica superior

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Trabajo Fin de Grado

AUTENTICACIÓN DE USUARIO BASADA EN INTERACCIÓN TÁCTIL SOBRE SMARTPHONES CON REDES NEURONALES PROFUNDAS

Autor: Víctor Martínez Guérin
Tutor: Julián Fiérrez Aguilar
Mayo 2021

AUTENTICACIÓN DE USUARIO BASADA EN INTERACCIÓN TÁCTIL SOBRE SMARTPHONES CON REDES NEURONALES PROFUNDAS

Autor: Víctor Martínez Guérin
Tutor: Julián Fiérrez Aguilar

Grupo BiDA Lab
Dpto. Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Mayo 2021

Resumen

Resumen

Este trabajo de fin de grado pretende estudiar en qué medida pueden servir los avances recientes en redes neuronales profundas en autenticación activa. Específicamente se hace uso de redes LSTM con el objetivo de autenticar usuarios en base a datos de interacción táctil obtenida del uso habitual y no restringido de smartphones. Concretamente se desarrolla una red neuronal siamesa que aprende de la base de datos para uso no comercial llamada UMDAA-2 (University of Maryland Active Authentication Dataset 02) de la Universidad de Maryland. Esta red será testada de dos formas, una en la que la red es entrenada con información de todos los usuarios de la base de datos, y otra en la que algunos usuarios son desconocidos, siendo este ultimo un escenario más realista y exigente. Los resultados obtenidos indican que la red tiene un cierto grado de éxito y sugieren que se deben de seguir profundizando en este tipo de técnicas frente a enfoques más tradicionales.

Palabras Clave

Aprendizaje profundo, LSTM, Biometría, teléfono inteligente, redes siamesas, seguridad, privacidad, autenticación, redes neuronales recurrentes, autenticación activa

Abstract

This final degree project aims to study to what extent new advancements in deep neural networks can serve in active authentication. Specifically LSTM networks are used with the objetive of authenticate user based on touch interaction data obtained from the usual and unrestricted use of smartphones. Concretely a siamese neural network is developed that learns from a data base for non-coemercial use named UMDAA-2 (University of Maryland Active Authentication Dataset 02) from University of Maryland. This network will be tested in two ways, one way in wich the network has received information about all the data base user's. And other way in which some of those users are completely unknown for the network, being this last test much more realistic and challenging. The results obtenied indicate that the network has some degree of success, and suggest that it should be continued to deepen this type of techniques compared to more traditional approaches.

Key words

Deep Learning, LSTM, Biometrics, smartphone, siamese network, security, privacy, authentication, recurrent neural networks, active authentication

Agradecimientos

En primer lugar, quería agradecer a Julián la oportunidad de hacer este trabajo y a Alejandro su ayuda durante su desarrollo. La elaboración de este trabajo me ha permitido acceder a las oportunidades laborales que he tenido incluso mucho antes de haberlo terminado.

También es obligado agradecer a mi madre y a mi hermana su apoyo excesivo e incondicional durante toda la carrera.

Por último, me gustaría agradecer también a mis dos tías abuelas por su apoyo y su ejemplo, que, aunque pocas veces he seguido, si me ha inspirado.

Índice general

Índice de Figuras

Índice de Tablas

1 Introducción

1.1. Motivación del proyecto

En los últimos años los dispositivos móviles se han transformado en objetos de uso cotidiano para una gran parte de la población. A través de smartphones y tabletas podemos realizar una gran cantidad de tareas tales como leer libros, usar redes sociales, jugar a videojuegos... pero también tareas menos triviales como realizar operaciones bancarias, gestionar una cita médica, monitorizar nuestra salud o realizar compras. Toda esto hace que un teléfono inteligente sea un elemento crítico, no solo a la hora de preservar la privacidad, quizás también nuestro patrimonio e incluso la salud.

A pesar de esto, los usuarios frecuentemente descuidan las protección de sus dispositivos, ya sea con el uso de contraseñas demasiado simples, o directamente no utilizando ningún tipo de sistema de autenticación. Algunos usuarios justifican esta dejadez en la molestia que supone el hecho de tener que utilizar un mecanismo de autenticación de forma constante (16).

El concepto de de Autenticación Activa ha surgido en los últimos años como solución a este problema (9). En la Autenticación Activa la autenticación se produce de forma continua en un segundo plano utilizando datos biométricos que el usuario proporciona mediante el uso del dispositivo. Estos datos biométricos pueden ser obtenidos desde los múltiples sensores que tienen estos dispositivos, tales como: la cámara frontal, pantalla táctil, giroscopio, localización GPS, patrones de escritura...

Debido a la naturaleza de este problema, una solución de software tradicional podría resultar muy complicada en caso de ser posible. Sin embargo una solución basada en Aprendizaje Automático parece más adecuada. Dentro del campo del Aprendizaje Automático, la revolución en los últimos años de las Redes Neuronales Profundas hacen necesario estudiar este tipo de técnicas en problemas biométricos (10).

1.2. Objetivos

En este trabajo se estudia el rendimiento de una red neuronal profunda sobre una base de datos destinada al estudio de la autenticación activa. El rendimiento es estudiado en dos

escenarios, uno en el que la red ha aprendido sobre todos los usuarios y otro en el que algunos usuarios no han pasado por la red.

1.3. Organización de la memoria

En el capítulo 2 se repasan los fundamentos teóricos utilizados en este trabajo. en la sección 2.2 se revisa el proceso de aprendizaje de una red neuronal con redes neuronales densas, se introducen de forma general las redes neuronales recurrentes y las particularidades de las redes LSTM. En la sección 2.3 se describe el funcionamiento de las redes siamesas utilizadas en este trabajo. En la sección 2.4 se comenta brevemente cómo evaluar el rendimiento de una red y cómo corregir algunas situaciones indeseadas durante el proceso de aprendizaje. En el capítulo 3 se describe la base de datos utilizada en este trabajo. En el capítulo 4 se explican los experimentos realizados y se comentan sus resultados. Por último en el capítulo 5 se extraen conclusiones y se proponen nuevas vías de trabajo.

2
Métodos

2.1. Introducción

En esta sección se repasa la teoría de las técnicas utilizadas para este trabajo, lo que incluye nociones básicas del aprendizaje profundo, el funcionamiento de las redes neuronales profundas, las redes neuronales recurrentes y redes siamesas.

2.1.1. Nociones de aprendizaje profundo

En este trabajo se han utilizado técnicas de aprendizaje profundo. Concretamente el problema estudiado entra dentro de la categoría del aprendizaje supervisado, es decir, un problema en el que queremos que un modelo aprenda de unos datos de entrenamiento para que, posteriormente, el modelo sea capaz de clasificar datos no vistos antes. Con el objetivo de comprobar el resultado del entrenamiento se reserva un conjunto de datos para test, es decir, se pide a la red que, una vez ha sido entrenada, intente etiquetar correctamente un conjunto de datos nunca antes visto, pero similares a los del conjunto de entrenamiento.

La red intenta aprender de los datos modificando sus parámetros internos durante el entrenamiento. El proceso de entrenamiento será controlado por unos hiperparámetros que, a diferencia de los parámetros, no pertenecen a la red. Estos hiperparámetros tienen un impacto importante en el aprendizaje de la red, y pueden ser optimizados para obtener mejores resultados. Normalmente se optimizan de forma iterativa por prueba y error, analizando los resultados que nos da la red después de entrenar (Figura 2.1).

Existen distintas formas de introducir los datos de entrenamiento en la red. El proceso que se ha seguido para este trabajo es Mini-Batch Gradient Descent (2). Consiste en en alimentar a la red con subconjuntos de datos de entrenamiento llamados batches. La red para cada batch modificara sus parámetros internos para intentar producir la salida deseada, este proceso se realizará hasta pasar todos los batches por la red. Después se repetirá todo el proceso tantas veces como indique el hiperparámetro "época"(Figura 2.2).

Figura 2.1: Proceso de optimización de hiperparámetros

2.2. Redes neuronales

Informalmente podríamos decir que el aprendizaje profundo es un tipo de algoritmo vagamente inspirado en la biología del cerebro, que consiste en acumular distintas capas de neuronas para extraer información de una entrada hasta obtener una salida deseada optimizando una función de coste. En esta sección veremos las redes neuronales densas, las recurrentes y las redes LSTM.

2.2.1. Redes neuronales densas

La red neuronal más simple se denomina red neuronal densa y, aunque finalmente no fue utilizada en el modelo final de este trabajo, ejemplifica de forma sencilla el proceso de aprendizaje de una red neuronal y deja claros conceptos que serían más complejos de entender en otras redes utilizadas. Una red neuronal densa se define como una red en la que todas sus neuronas están conectadas entre sí y en la que que no se hace ningún juicio previo sobre los datos que la

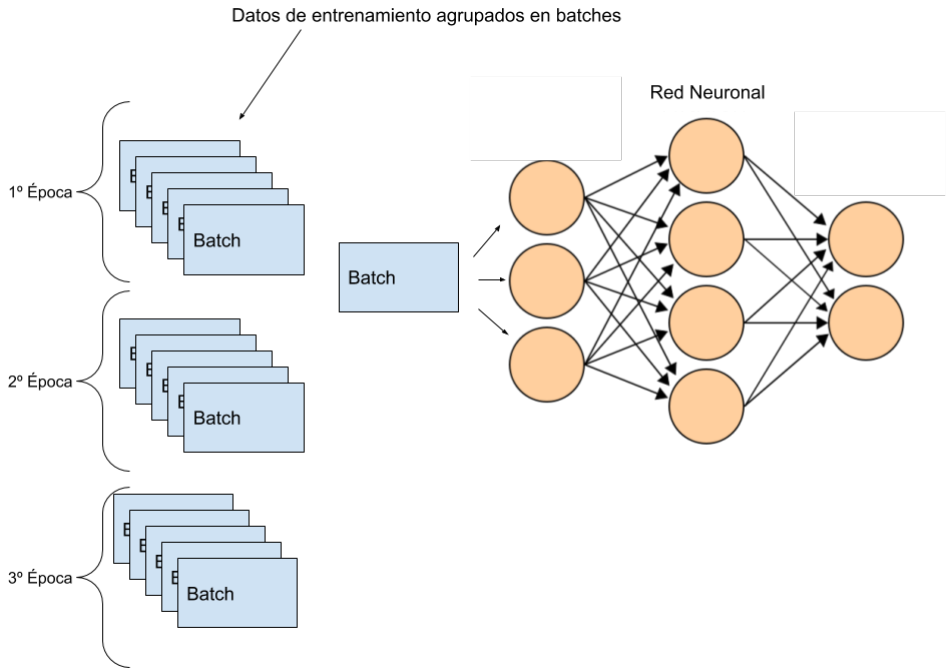


Figura 2.2: Batches y épocas. Imagen obtenida en base a (4)

atraviesan. Otros tipos de redes si que presuponen alguna características de los datos de entrada, como que estos tengan varias dimensiones (CNN) o datos con correlación temporal (RNN).

Las ecuaciones que gobiernan esta red se irán derivando al estilo de las referencias (19) y (12). El primer paso en el entrenamiento de una red neuronal es la introducción de datos y la obtención de una predicción, esta primera parte es conocida como forward propagation.

Forward Propagation

En una red neuronal densa, cada capa está compuesta por un conjunto de neuronas. Cada una de estas neuronas recibe un vector de valores, y realiza la siguiente operación:

$$z = w_1 x_1 + w_2 x_2 + ... + w_n x_n = w^T \cdot x + b \tag{2.1}$$

$$\widehat{y} = g(z) \tag{2.2}$$

Siendo x los datos de entrada, w los pesos de la neurona y b el sesgo.

En esta operación también tenemos la función de activación g, como se ve en la ecuación 2.2. Esta función permite a la neurona (y a la red) ajustarse a funciones no lineales. Hay muchas opciones para elegir la función de activación, cada una con diferentes implicaciones. Una de las funciones más populares es la ReLU mostrada en la figura 2.4.

El sesgo b dentro de la ecuación 2.1 se ajusta para poder desplazar el resultado de estas funciones de activación a izquierda o derecha en el eje horizontal.

Describimos ahora el funcionamiento de una capa con las siguientes ecuaciones:

$$z = w_i^T \cdot a^{[l-1]} + b_i \tag{2.3}$$

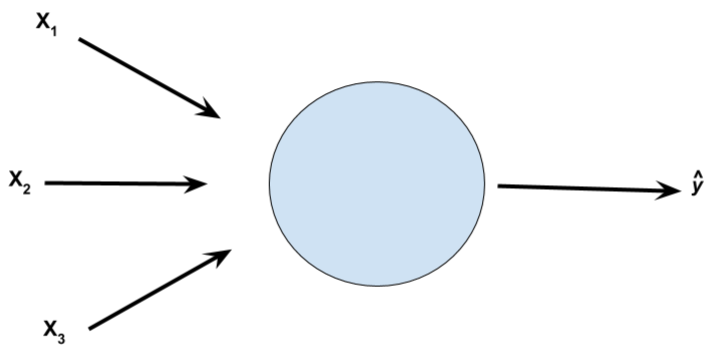


Figura 2.3: Neurona

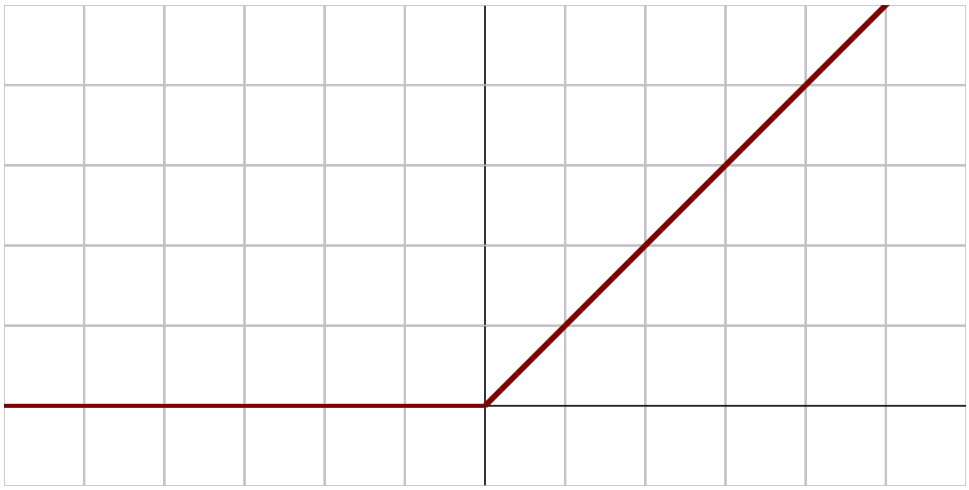


Figura 2.4: Rectified linear unit (ReLU). Imagen obtenida de (7)

$$a_i^{[l]} = g^{[l]}(z_i^{[l]}) \tag{2.4}$$

Donde $[l]$ es la capa donde se realiza la operación y el subíndice i la neurona.

En la ecuación 2.3, $a^{[l-1]}$ hace referencia al resultado de toda capa anterior que será la entrada de todas las neuronas de la capa actual al ser una red neuronal densa. En la ecuación 2.4 $a_i^{[l]}$ es el resultado de la neurona i-esima en la capa l.

Las ecuaciones 2.3 y 2.4 se suelen presentar de forma vectorizada para que hagan referencia a toda la capa, y no a cada una de las neuronas de la misma. Apilando los pesos transpuestos de cada neurona y sus sesgos verticalmente podemos construir una versión vectorizada del algoritmo y omitir el subíndice i

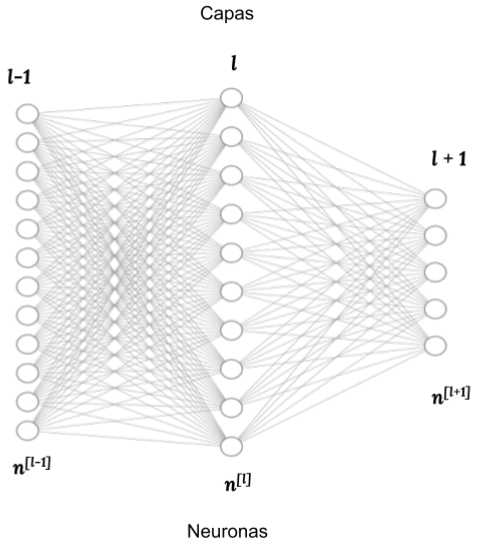


Figura 2.5: Capas en una red neuronal densa. Imagen obtenida en (1)

$$W^{[l]} = \begin{bmatrix} w_1^{[l]T} \\ w_2^{[l]T} \\ w_3^{[l]T} \\ \dots \\ w_{n-1}^{[l]T} \\ w_n^{[l]T} \end{bmatrix} \tag{2.5}$$

Esta matriz de pesos W tiene dimensiones $n^{[l]} \times n^{[l-1]}$.Donde $n^{[l]}$ se corresponde con el numero neuronas en la capa actual y $n^{[l-1]}$ con el numero de neuronas de la capa anterior, como implica la ecuación 2.3.

$$b^{[l]} = \begin{bmatrix} b_1^{[l]} \\ b_2^{[l]} \\ b_3^{[l]} \\ \dots \\ b_{n-1}^{[l]} \\ b_n^{[l]} \end{bmatrix} \tag{2.6}$$

Y un vector de sesgos de dimensiones $n^{[l]} \times 1$

Con estos elementos la versión vectorizada de las ecuaciones 2.3 y 2.4 es:

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^l \tag{2.7}$$

$$a^{[l]} = g^{[l]}(z^{[l]}) \tag{2.8}$$

Donde $z^{[l]}$ y $a^{[l]}$ tiene dimensiones $n^{[l]} \times 1$ y $a^{[l-1]}$ dimensiones $n^{[l-1]} \times 1$

Las ecuaciones vistas hasta ahora trabajan con un único ejemplo, es decir un único vector x de características. Como se ha comentado anteriormente, a una red neuronal se le pueden introducir varias muestras simultáneamente agrupadas en batches.

Supongamos que tenemos una batch de m muestras x de dimensiones $n_x \times 1$. Representamos este batch con la matriz X de dimensiones $n_x \times m$, con lo que para la primera capa de la red tendríamos:

$$Z^{[1]} = W^{[1]}X + b^1 \tag{2.9}$$

$$A^{[1]} = g^{[1]}(Z^{[1]}) \tag{2.10}$$

Teniendo en cuenta que X es sencillamente la matriz de entrada, podríamos entenderla como $A^{[0]}$. Entonces podemos escribir las anteriores ecuaciones 2.7 y 2.8 como:

$$Z^{[l]} = W^{[l]}A^{[l-1]} + b^l \tag{2.11}$$

$$A^{[l]} = g^{[l]}(Z^{[l]}) \tag{2.12}$$

Donde ahora $A^{[l-1]}$ tiene dimensiones $n^{[l-1]} \times m$ y $A^{[l]}$ $Z^{[l]}$ y tienen dimensiones $n^{[l]} \times m$. Estas ecuaciones son las que utilizara la red para intentar obtener un resultado que se ajuste con el deseado. Se necesita evaluar la bondad de este resultado antes de poder optimizar los parámetros (pesos de la red, W). Con este objetivo se usa la función de perdida.

Función de perdida y función de coste

Existen distintas funciones de pérdidas, para ejemplificar su funcionamiento general describiremos la función de entropía cruzada binaria:

$$L(\widehat{y}^{(i)}, y^{(i)}) = -(y \log \widehat{y} + (1-y) \log (1-\widehat{y})) \tag{2.13}$$

$$J(W, b) = 1/m \sum_{i=1}^{m} L(\widehat{y}^{(i)}, y^{(i)}) \tag{2.14}$$

Esta función (2.13) sirve para evaluar un modelo de clasificación binaria, es decir, la red intenta clasificar los datos en dos posibles categorías. Supongamos que los posibles resultados de la red y van del 0 al 1. Por lo tanto, la red habrá tenido más éxito cuando el valor predicho $\widehat{y}$este más cerca del valor real (0 o 1). La ecuación 2.13 es la encargada de evaluar esta proximidad, y podemos entender que está compuesta de dos partes. El primer sumando es no nulo si el valor real de la predicción es 1, el logaritmo de este sumando transforma el valor de $\widehat{y}$a un valor entre - ∞ y 0. Si la predicción es 0 y el valor real es 1, la predicción no puede ser más errónea y el resultado del sumando sera - ∞, si la predicción es 1 y el valor real 1 la predicción es totalmente acertada, y el sumando pasara a valer 0, por lo tanto el error es nulo. En el segundo sumando se sigue la misma idea para el caso opuesto. Posteriormente la ecuación 2.14 (función de coste) realiza la suma ponderada por el número de casos, con lo que obtenemos una magnitud del error de la red, un valor que se debe intentar reducir. Para ello deberemos buscar valores para los parámetros de la red que minimicen este error mediante el algoritmo de backpropagation.

Backpropagation

El algoritmo de backpropagation consiste en obtener el gradiente de la red neural sobre el que se aplicará el algoritmo de descenso por gradiente para obtener los mínimos buscados. El gradiente de la función de coste lo obtenemos con las siguientes derivadas parciales de la función de coste:

$$dW^{[l]} = \frac{\partial L}{\partial W^{[l]}} = \frac{1}{m} dZ^{[l]} A^{[l-1]^T} \tag{2.15}$$

$$db^{[l]} = \frac{\partial L}{\partial b^{[l]}} = \frac{1}{m} \sum_{i=1}^{m} dZ^{[l](i)} \tag{2.16}$$

$$dA^{[l-1]} = \frac{\partial L}{\partial A^{[l-1]}} = W^{[l]^T} dZ^{[l]} \tag{2.17}$$

$$dZ^{[l-1]} = \frac{\partial L}{\partial A^{[l-1]}} = dA^{[l]} * g'(Z^{[l]}) \tag{2.18}$$

Donde $*$ es un producto elemento a elemento

Con estos resultados podemos actualizar los pesos de las neuronas siguiendo el algoritmo de descenso por gradiente con la siguientes fórmulas.

$$W^{[l]} = W^{[l]} - \alpha dW^{[l]} \tag{2.19}$$

$$b^{[l]} = b^{[l]} - \alpha db^{[l]} \tag{2.20}$$

De esta forma se irán optimizando los parámetros internos de la red, lo que hará que se reduzca el valor de la función de coste y por lo tanto se reduzca el error en la clasificación. El proceso completo de aprendizaje se puede ver representado en la siguiente figura:

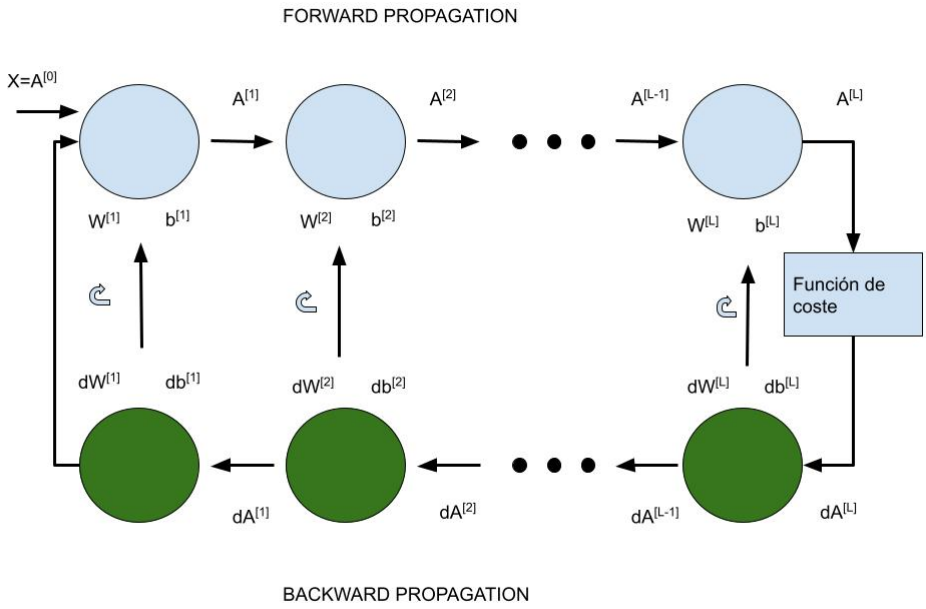


Figura 2.6: Proceso de aprendizaje.Imagen basada en figura 9 de (19)

En estas fórmulas (2.19 y 2.20) encontramos el parámetro α, llamado tasa de aprendizaje. Este parámetro rige como de rápido se efectúa el descenso por gradiente. Una adecuada selección de este parámetro es clave para todo el proceso, puesto que un valor demasiado elevado podría evitar la convergencia del algoritmo y un valor muy pequeño ralentizaría demasiado el proceso. Para la selección de este algoritmo existen distintos métodos, en este trabajo se utiliza la optimización de Adam para ir ajustando el valor α de forma adecuada durante todo el proceso.

Con esto concluimos la descripción de una red neuronal profunda densa, ahora pasaremos a describir las particularidades de las redes recurrentes utilizadas en este trabajo.

2.2.2. Redes neuronales recurrentes y redes LSTM

El objetivo buscado con el uso de las redes neuronales recurrentes es tener en cuenta en el proceso de aprendizaje la posición relativa que tienen las distintas muestras entre ellas. Un ejemplo de uso típico de este tipo de redes es el procesamiento de texto, ya sea completar un texto, identificar ciertos elementos o traducirlo. Para estas tareas el orden en el que están escritas las palabras es importante.

Las redes LSTM son un tipo de redes RNN. Primero, al estilo de la referencia (13) explicaremos primero las RNN más generales y después las LSTM, con el objetivo de tener la mayor claridad posible.

Forward Propagation en RNN

Una red neuronal recurrente (RNN por sus siglas en inglés) sigue el esquema de la figura 2.7

Viendo este esquema podemos ver que, a diferencia de las redes neuronales densas, la posición relativa de las muestras influye en el procesamiento del resto. En la figura se puede ver que una muestra influye en el procesamiento de las siguientes, pero no de las anteriores.

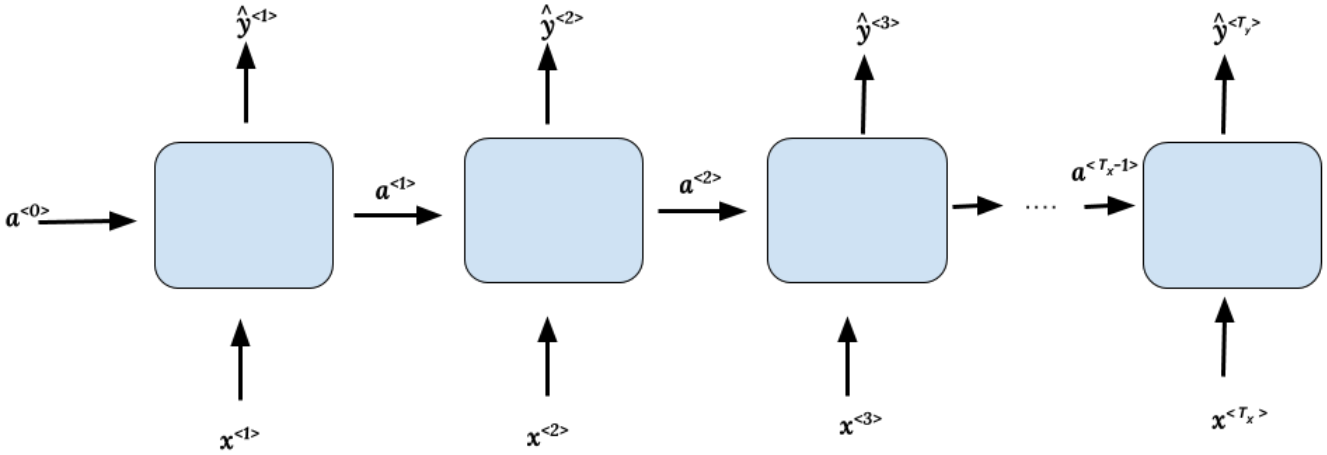


Figura 2.7: Capas en una red neuronal recurrente

Otro aspecto a tener en cuenta sobre este tipo de redes es que el tamaño de la salida no tiene por qué ser el mismo que el de entrada. En el caso de la figura 2.7 es igual pero puede haber distintos tipos como se ve en la figura 2.8.

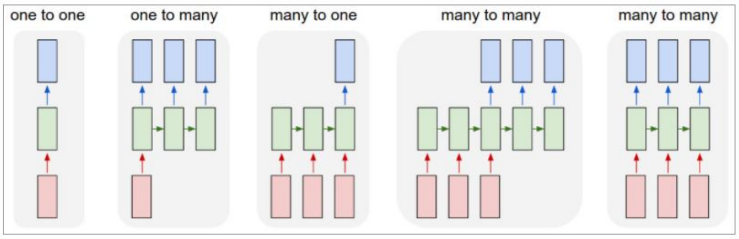


Figura 2.8: Tipos de redes RNN según sus entradas/salidas. Imagen tomada de (6)

En la figura 2.7 vemos que la entrada es una secuencia, donde cada elemento de la secuencia es denotado por x^i, siendo la secuencia total de tamaño T_x. Cada elemento de salida es denotado por $\widehat{y}^i$. Las siguientes ecuaciones muestran como se calcula la salida para el primer elemento:

$$a^{<1>} = g_1(W_{aa}a^{<0>} + W_{ax}x^{<1>} + b_a) \tag{2.21}$$

$$\widehat{y}^1 = g_2(W_{ya}a^{<1>} + b_y) \tag{2.22}$$

La función de activación g_1 se usa para obtener el vector $a^{<1>}$ y g_2 para obtener el resultado $\widehat{y}^1$, ambas funciones no tienen porque coincidir, aunque no suele ser habitual. Las cantidades W_{aa}, W_{ax} y W_{ya} son los pesos a optimizar. Para calcular la salida $\widehat{y}^{<i>}$, tenemos:

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \tag{2.23}$$

$$\widehat{y}^t = g(W_{ya}a^{(<t>)} + b_y) \tag{2.24}$$

Una forma conveniente de simplificar estas ecuaciones es:

$$W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a = \begin{bmatrix} W_{aa} & W_{ax} \end{bmatrix} \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} + b_a = W_a \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} + b_a \tag{2.25}$$

Con lo que finalmente tenemos dos matrices Wa e Wy una para la función de activación y otra para el resultado final:

$$a^{<t>} = g(W_a \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} + b_a) \tag{2.26}$$

$$\widehat{y}^t = g(W_y a^{<t>} + b_y) \tag{2.27}$$

Con esta ecuación terminamos de describir forward propagation en RNN.

Backpropagation en RNN

El algoritmo de backpropagation, al igual que en la redes neuronales densas, se calcula en dirección opuesta al algoritmo de forward propagation, en el caso de las RNN este algoritmo recibe el nombre de backpropagation througth time, puesto que la corrección de los pesos obtenidos de las matrices W_a y W_y al aplicar el descenso por gradiente se va trasladando desde las ultimas muestras a las primeras como se ve en la figura 2.9 .

El cálculo del descenso por gradiente se realiza de la misma forma que vimos en las redes densas. Suponiendo que tenemos un problema de clasificación binaria, tendríamos una función de pérdida y coste tal que:

$$L^{<t>}(\widehat{y}^{<t>}, y^{<t>}) = -(y^{<t>} \log \widehat{y}^{<t>} + (1 - y^{<t>}) \log (1 - \widehat{y}^{<t>})) \tag{2.28}$$

$$L(\widehat{y}, y) = 1/T_y \sum_{t=1}^{T_y} L^{<t>}(\widehat{y}^{<t>}, y^{<t>}) \tag{2.29}$$

Estas ecuaciones son idénticas a las que rigen las redes neurales densas, la diferencia está en que las entradas son muestras con una posición temporal definida.

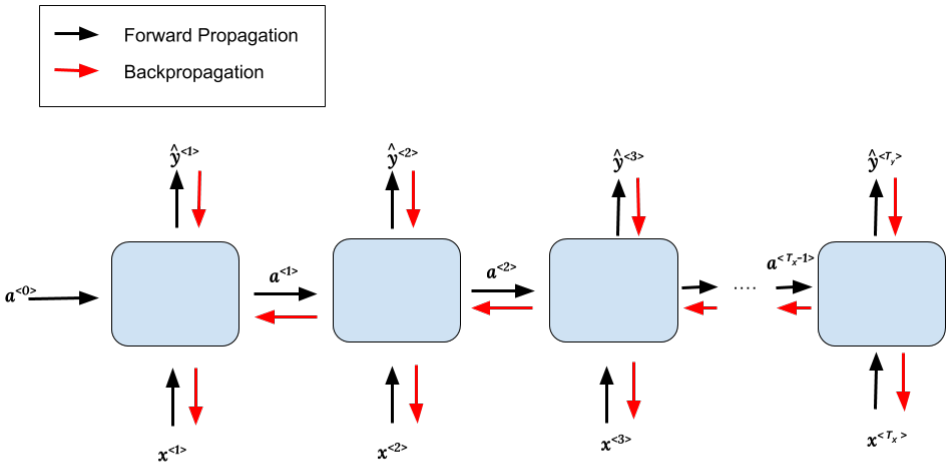


Figura 2.9: Forward y Backward propagation en una red neuronal recurrente

Desvanecimiento de gradiente en redes RNN

Las redes RNN son especialmente sensibles a el problema de desvanecimiento de gradiente, aunque este problema también se puede dar en redes densas. Cuando una red neuronal tiene mucha profundidad, es más vulnerable a problemas de desvanecimiento o explosión de gradiente. El problema se puede ver fácilmente si analizamos las ecuaciones de 2.11 y 2.12 de las redes neuronales densas.

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^l \tag{2.30}$$

$$a^{[l]} = g^{[l]}(z^{[l]}) \tag{2.31}$$

Si tenemos una red neuronal arbitrariamente densa (esto es con muchas capas, como se ve en la figura 2.10) y definimos una función de activación y un sesgo tal que:

$$g(z) = z \tag{2.32}$$

$$b^l = 0 \tag{2.33}$$

tendremos que

$$A^{[1]} = W^{[1]} x \tag{2.34}$$

$$A^{[2]} = W^{[2]} W^{[1]} x \tag{2.35}$$

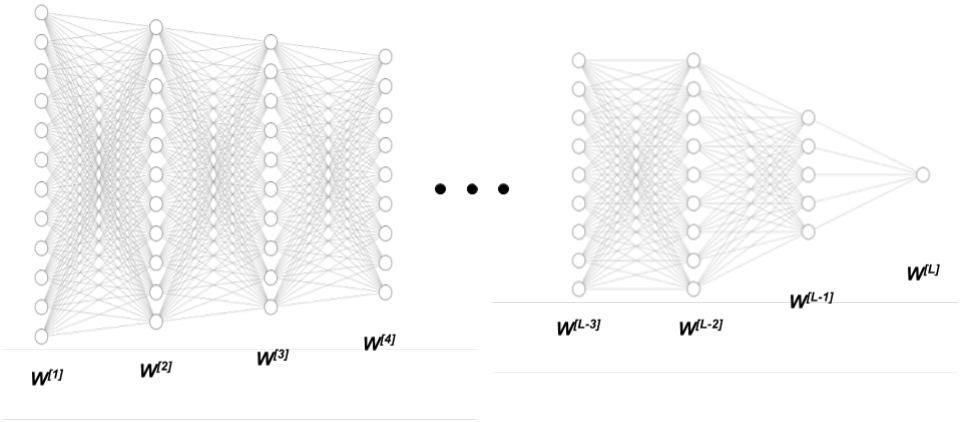


Figura 2.10: Red neuronal arbitrariamente densa. Figura obtenida en (1)

$$A^{[3]} = W^{[3]}W^{[2]}W^{[1]}x \tag{2.36}$$

es decir que la predicción $\widehat{y}$ valdrá:

$$\widehat{y} = W^{[l]}W^{[l-1]}W^{[l-2]}W^{[l-3]}\dots W^{[3]}W^{[2]}W^{[1]}x \tag{2.37}$$

si consideramos que los pesos tienen el mismo valor tal que: $W^{[l]} = W^{[l-1]} = \dots = W^{[2]} = W^{[1]}$.

Tendremos dos posibilidades, la primera:

$$W^{[l]} > I \tag{2.38}$$

Para redes neuronales muy profundas con un número de capas l muy grande, vemos que para este caso en el que la matriz es superior a la unidad nos encontraremos con que los cálculos cada vez dan valores más grandes. En un momento dado pueden superar la capacidad de la variable que los contenga en memoria y bloquear el entrenamiento de la red, esto se conoce como explosión por gradiente.

En el caso contrario, tenemos:

$$W^{[l]} < I \tag{2.39}$$

Los números se irán haciendo cada vez más pequeños, haciendo que las derivadas que se utilizan en el descenso por gradiente sean nulas, y haciendo también fracasar el entrenamiento.

El problema de explosión por gradiente tiene soluciones conocidas y simples que quedan fuera de este trabajo, pero el desvanecimiento de gradiente es un problema para las RNN más

complejo de resolver, puesto que el número de capas de la red es normalmente muy grande dependiendo del tamaño de la secuencia. Esto quiere decir que para secuencias muy grandes nos es difícil relacionar muestras que estén muy separadas espacialmente. De este problema surge la necesidad de usar redes LSTM, Long short-term memory que como su nombre indica, son redes que solucionan este problema y permite relacionar muestras muy separadas. Son además las utilizadas en este trabajo.

Redes RNN Bidireccionales

La RNN que se ha descrito hasta ahora solo puede transmitir información desde una muestra a las siguientes, es decir, para cada momento de la secuencia solo se utiliza información actual y previa. Las redes bidireccionales nos permiten también utilizar información de valores posteriores. En la figura 2.11 se presenta el esquema de una red bidireccional comparado con el de una red unidireccional.

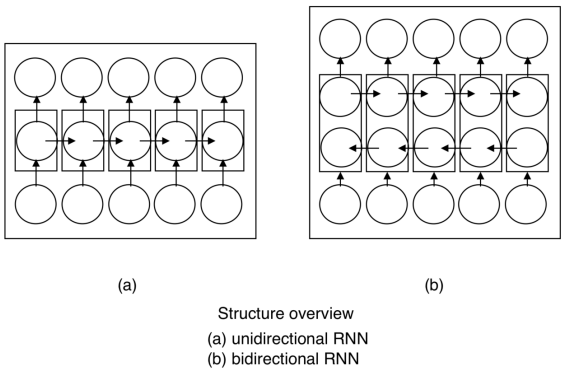


Figura 2.11: Red RNN Bidireccional. Imagen obtenida de (5)

En la imagen podemos ver que una red bidireccional es el resultado de la superposición de dos redes RNN en un grafo acíclico, donde una de las redes avanza en el tiempo y la otra retrocede. El resultado de cada muestra vendrá dado de combinar la salida de las dos redes, como se puede ver en la siguiente ecuación:

$$\hat{y^t} = g(W_y \begin{bmatrix} \overrightarrow{a}^{<t>} \\ \overleftarrow{a}^{<t>} \end{bmatrix} + b_y) \tag{2.40}$$

A pesar de que estas redes suponen una mejora en la reducción del error se ha de tener en cuenta que suponen un incremento muy considerable en los recursos utilizados, al duplicarse el número de neuronas, aumentará el tiempo que tardará en entrenarse la red, y se podrá iterar menos veces en la optimización de hiperparámetros (ver figura 2.1).

2.2.3. Redes LSTM

A continuación se explicarán las redes LSTM al estilo de las referencias (15) y (13). En una red LSTM definimos un nuevo valor c, llamado celda de memoria. Este valor permite relacionar elementos muy distantes entre si. El contenido de c puede mantenerse o actualizarse en cada neurona LSTM. La variable que contiene la posible actualización de c es $\tilde{c}$, se denomina valor candidato y se rige por la siguiente ecuación:

$$\tilde{c}^{<t>} = \tanh(W_c \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} + b_c) \tag{2.41}$$

La función de activación que se usa normalmente para este valor es *tanh*, la información será mantenida o actualizada por otro valor en cada neurona dependiendo de la siguiente ecuación:

$$c^{<t>} = \Gamma_u \cdot \tilde{c}^{<t>} + \Gamma_f \cdot c^{<t-1>} \tag{2.42}$$

Los valores Γ_u y Γ_f se conocen respectivamente como puerta de olvido y puerta de actualización. Y se rigen por las siguientes ecuaciones:

$$\Gamma_u = \sigma(W_u \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} + b_u) \tag{2.43}$$

$$\Gamma_f = \sigma(W_f \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} + b_f) \tag{2.44}$$

Por último tenemos la puerta de salida Γ_o y el vector de activación a:

$$\Gamma_o = \sigma(W_o \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} + b_o) \tag{2.45}$$

$$a^{<t>} = \Gamma_o \cdot \tanh(c^{<t>}) \tag{2.46}$$

Todas estas ecuaciones determinan el comportamiento de una neurona LSTM y se pueden ver representadas en la siguiente figura:

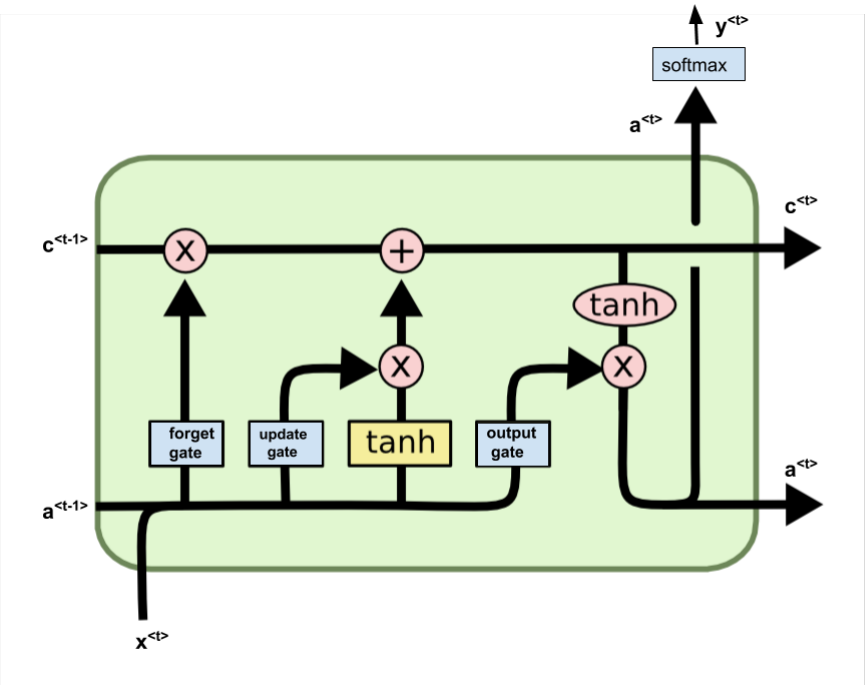


Figura 2.12: Esquema de una LSTM. Imagen obtenida en base a (15)

Como se comentó anteriormente, estas redes solucionan en parte el problema del desvanecimiento por gradiente gracias a la celda de memoria, como se puede ver en la figura 2.13, al conectar varias de estas unidades LSTM es posible mandar información desde cualquier elemento de la secuencia a otro. Esto permite que se capten mejor las dependencias a largo plazo, las puertas puerta de olvido (Γ_f) y de actualización (Γ_u) determinarán que información se mantendrá o se perderá en cada unidad, y estos valores dependerán de los pesos W_f y W_u que serán ajustados para minimizar el error, como hemos visto anteriormente.

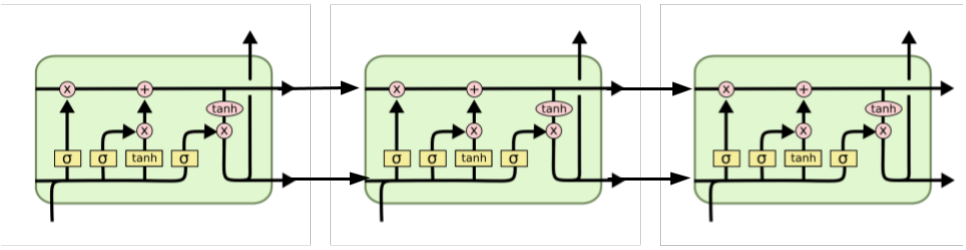


Figura 2.13: LSTM conectadas. Imagen obtenida en base a (15)

2.3. Redes Siamesas

El objetivo de este trabajo es autenticar usuarios, concretamente, dadas dos muestras de cada usuario queremos determinar si las muestras pertenecen al mismo usuario o no. Para este tipo de tareas de autenticación suelen ser utilizadas redes siamesas. Como se puede ver en la figura 2.16, las redes siamesas (11) son redes neuronales compuestas de dos redes idénticas, cada red recibirá una entrada y como resultado dará un vector de salida. Es decir, cada una de las redes habrá realizado una codificación de la entrada recibida.

El objetivo de la red es que las codificaciones que se realicen de cada entrada sean tales que si dos muestras pertenecen al mismo usuario, (lo que se conoce como un caso positivo) los dos vectores sean similares. Y en el caso de que las muestras provengan de distintos usuarios (un caso negativo), los dos vectores han de ser suficientemente diferentes. Podemos expresar esto matemáticamente con concepto de distancia d :

Si $x^{(i)}$ y $x^{(j)}$ pertenecen al mismo usuario, entonces:

$d = \left\| f(x^{(i)}) - f(x^{(j)}) \right\|^2$, d tiene valores pequeños.

Si $x^{(i)}$ y $x^{(j)}$ no pertenecen al mismo usuario, entonces:

$d = \left\| f(x^{(i)}) - f(x^{(j)}) \right\|^2$, d tiene valores suficientemente grandes.

Para lograr este objetivo, se entrena la red utilizando funciones de pérdida de tipo Ranking Loss (3). Concretamente en este trabajo utilizamos la función de pérdida Pairwise o función de pérdida contrastiva.

2.3.1. Función de perdida Pairwise

La función de pérdida Pairwise necesita los siguientes elementos:

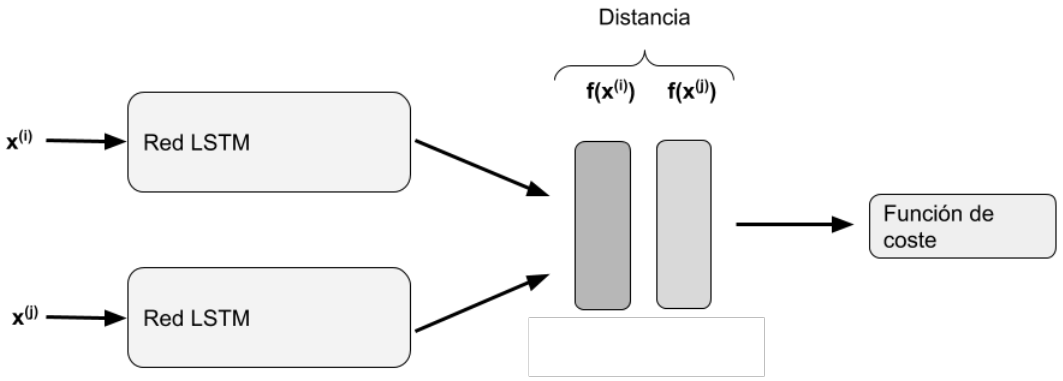


Figura 2.14: Esquema Red Siamesa

· Una muestra ancla de un usuario, denominada A

· Una muestra obtenida del mismo usuario para el caso positivo, denominada P

· Una muestra obtenida de otro usuario para el caso negativo, denominada N

Utilizando el concepto de distancia descrito anteriormente, el objetivo será obtener unas codificaciones de las muestras tales que se cumpla la siguiente desigualdad:

$$\|f(A) - f(P)\|^2 \leq \|f(A) - f(N)\|^2 \tag{2.47}$$

Para lograr esto se utiliza la función de pérdida Pairwise:

$$\mathcal{L}(A, P, N) = \begin{cases} d(A, P) & si \quad Positivo \\ \text{máx}(0, \alpha - d(A, N)) & si \quad Negativo \end{cases} \tag{2.48}$$

Con esta función de pérdida, los casos positivos serán no nulos siempre que la distancia sea mayor que cero. Para los casos negativos, la pérdida será nula cuando la distancia sea mayor que la variable α, llamada margen. De esta forma α estimula a la red para que los casos negativos tengan una diferencia de α, también garantiza que la red no derroche esfuerzos intentado hacer las distancias excesivamente grandes, puesto que una vez superada la distancia α la perdida es nula.

Finalmente la función de coste es:

$$J = \sum_{i=1}^{n} \mathcal{L}(A^{(i)}, P^{(i)}, N^{(i)}) \tag{2.49}$$

y sobre ella se aplicará el descenso por gradiente para entrenar a la red.

2.4. Diagnóstico de curvas de aprendizaje

Una vez se ha entrenado la red neural se deben analizar los resultados con el fin de determinar si el aprendizaje ha sido adecuado o no.

Existen dos casos comunes en los que el aprendizaje ha sido erróneo. Por un lado tenemos el caso de underftiting, que se da cuando tenemos un error elevado en el conjunto de entrenamiento y en el conjunto de test. Esto indica que la red no ha logrado aprender de los datos, y por lo tanto se debe replantear el modelo que se está utilizando o aumentar los datos.

El otro escenario habitual es el overfitting, que se produce cuando tenemos un error muy bajo en los datos de entrenamiento y un error elevado en el conjunto de test. Esto quiere decir que la red no es capaz de generalizar adecuadamente, y por lo tanto el modelo puede ser válido pero ha aprendido con demasiada exactitud el conjunto de entrenamiento, ya que no es capaz de generalizar correctamente lo aprendido a nuevos conjuntos de datos.

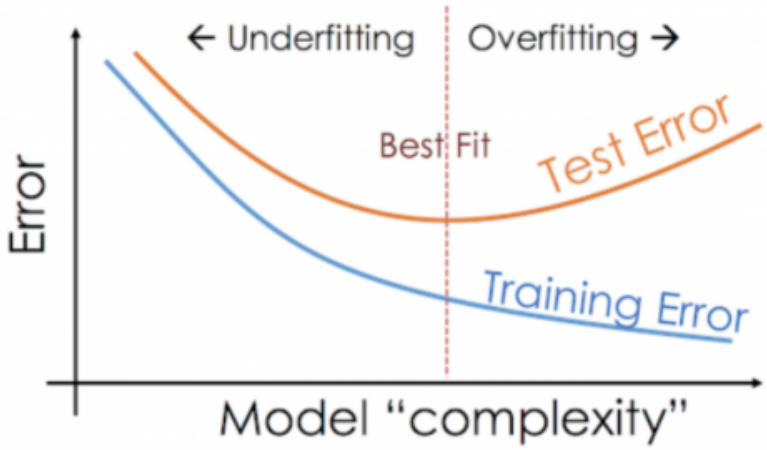


Figura 2.15: Overfitting y Underfitting según el error. Imagen obtenida en (17)

Existen muchas posibilidades para resolver este problema, en este trabajo se ha utilizado la técnica de dropout para rectificar y prevenir este escenario.

2.4.1. Dropout

El dropout es una técnica que permite reducir la capacidad de aprendizaje de la red con la esperanza de que se obtenga una mejor generalización. Como se puede ver en la figura, consiste en ignorar algunas neuronas escogidas aleatoriamente durante el entrenamiento.

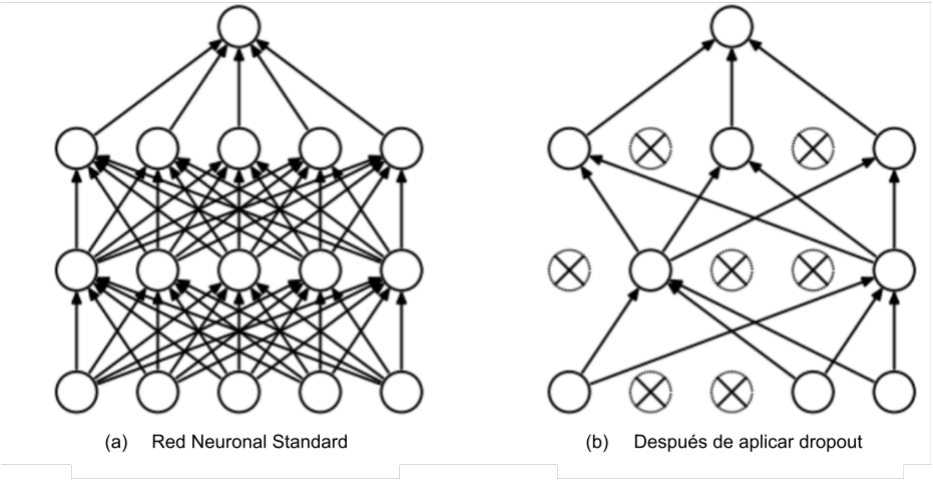


Figura 2.16: Dropout. Imagen obtenida de (20)

Para las redes LSTM se aplica el equivalente de esta técnica conocida como recurrent dropout que, como se puede ver en la figura, consiste en ignorar algunas de las entradas o actualizaciones de las puertas de las LSTM.

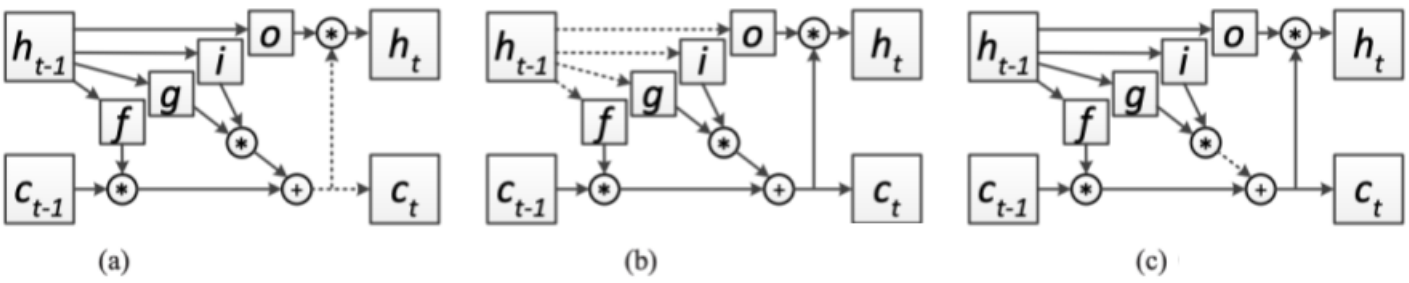


Figura 2.17: Ejemplos de Recurrent Dropout, las líneas de puntos son eliminadas. Imagen obtenida de (18)

3

Base de datos

En este trabajo se ha utilizado la base de datos para uso no comercial Unniversity of Maryland Active Authentication Dataset 02 (por sus siglas UMDAA-02, (9)). Es una base datos usada para investigación en autenticación activa multimodal con smartphones, concretamente la base da datos proporciona 141.14 GB de datos recogidos por distintos sensores como: la cámara frontal, pantalla táctil, acelerómetro, giroscopio, magnetómetro, sensores de luz, GPS, Wifi, Bluetooth, sensores de temperatura, presión y de proximidad.

Figura 3.1: Nexus 5. Imagen tomada de (8)

A 48 voluntarios se les proporciono un Nexus 5 durante dos meses, que usaron como dispositivo principal durante al menos una semana. Después los usuarios tenían la opción de parar la toma de datos y revisarlos previamente antes de compartirlos. Los datos están agrupados por sesiones, que es el tiempo desde que un usuario a desbloqueado el teléfono hasta que lo vuelve a bloquear. Estas sesiones están almacenadas cronológicamente en distintas carpetas según el mes y el día de la toma de muestras. Dentro de cada sesión se almacenan en archivos de texto la información relativa a los swipes (pasadas con el dedo sobre la pantalla del smartphone).

Nº de usuarios	48
Media Sesión/Usuario con datos de swipe	196
Total de pulsaciones (dedo abajo-dedo arriba)	177417
Total swipes (incluyendo taps)	489723
Swipe de mayor longitud	3637
Nº de Swipes/Usuario	10203
Nº de Swipes/Sesión	52
Nº de Swipes (mayores de 4 puntos)	167126
Nº de Swipes/Usuario (mayores de 4 puntos)	3482
Nº de Swipes/Sesión (mayores de 4 puntos)	18

Cuadro 3.1: Información general de los swipes. (9)

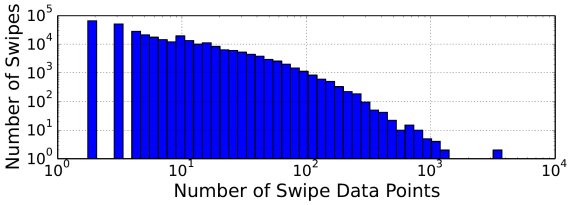


Figura 3.2: Histograma de datos por swipe. (9)

Una particularidad interesante de esta base de datos es que la recogida de muestras era pasiva, es decir, a diferencia de otras base de datos, no se les pedía a los usuarios que realizaran ninguna tarea especifica con los smartphones, por lo que la información recogida es especialmente útil para estudiar métodos de autenticación activa, dado que es un escenario mucho más realista.

4

Experimentos Realizados y Resultados

En esta sección se describirán los experimentos realizados. Para este trabajo se han realizado dos experimentos, una llamado de conjunto abierto y otro de conjunto cerrado.

Antes de detallar estos dos tipos de experimentos se explicará el procesamiento de las muestras, que fue común para ambos.

4.1. Procesamiento de los datos

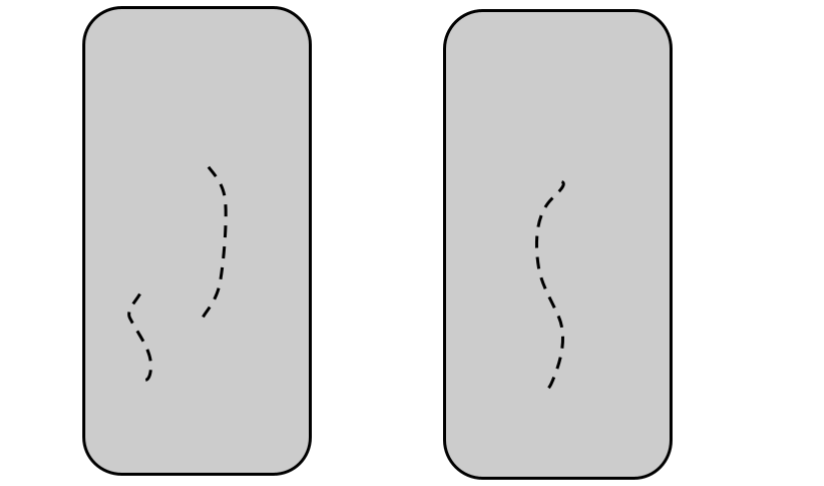
La base de datos anteriormente descrita esta organizada en ficheros CSV (por sus siglas en inglés comma separated values), los datos que encontramos en los ficheros son: el tiempo actual, la presión realizada sobre la pantalla, la posición del dedo según el eje horizontal de la pantalla, la posición según el eje vertical y el tipo de evento. Hay dos tipos de evento, uno para cuando se está ejerciendo presión en la pantalla y otro para cuando se deja de hacer presión. En base a estos dos tipos de eventos podemos extraer los swipes completos.

Antes de extraer más características de los swipes nos encontramos con un problema, las redes siamesas basadas en LSTM que se utilizan en este trabajo sólo pueden comparar muestras de igual tamaño, y como se vio en al figura 3.2 disponemos de swipes muy distintos. En este sentido se valoraron varias opciones. Una de ellas fue concatenar swipes sucesivos de cada usuario hasta obtener un número determinado de muestras pero, con esta alternativa, las características extraídas no serían de swipes y perderían cierto sentido, puesto que se estarían mezclando dos o más swipes distintos. Otra opción por la que se finalmente se optó, al obtener resultados marginalmente mejores, es el remuestreo de la señal. Aunque el remuestreo dañe en cierta medida la calidad de la muestra, mantiene la estructura del swipe, como se puede ver en la figura 4.1.

Concretamente se eligió remuestrear para que todas las señales midieran el tamaño medio del swipe, es decir 35 puntos. Para ello se utilizó la función de Python de la librería Scypy de ‘python signal.resampling()‘ que utiliza métodos de Fourier (14). Antes de remuestrear se descartaron muestras inferiores a 10 puntos de longitud.

Una vez obtenidas las muestras de igual tamaño se extraen características basadas en las señales básicas del swipe, se pueden ver en el cuadro 4.1.

Pasamos ahora a describir los experimentos de conjunto cerrado y abierto.



a) Dos swipes distintos concatenados. b) Un único swipe resampleado. Opción elegida.

Figura 4.1: Ilustración de las posibles alternativas

Características dadas por la base de datos
1º Posición en el eje X
2º Posición en el eje Y
3º Presión
Características derivadas
4º Distancia entre los puntos de un swipe en el eje X
5º Distancia entre los puntos de un swipe en el eje Y
6º Velocidad
7º Aceleración
8º Coseno de las posiciones en el eje X
9º Seno de las posiciones en el eje X
10º Coseno de las posiciones en el eje Y
11º Seno de las posiciones en el eje Y
12º Área entre cada posición X y posición Y
13º Ángulo entre cada posición X y posición Y
14º Distancia entre cada posición X y posición Y
15º Valor absoluto de la transformada de Fourier en el eje X
16º Valor absoluto de la transformada de Fourier en el eje Y
17º Correlación entra la velocidad y la presión

Cuadro 4.1: Características utilizadas

4.2. Experimento conjunto cerrado

La principal diferencia entre el experimento del conjunto abierto y conjunto cerrado, es la cantidad de usuarios utilizados en el entrenamiento. Para el experimento de conjunto cerrado se han utilizado muestras de todos los usuarios de la base de datos para entrenar la red, en conjunto abierto se excluyen algunos usuarios, como se verá más adelante.

Reparto de muestras entre entrenamiento y test

La división de los datos en entrenamiento y test se hizo por días. Concretamente el 80 % de los días de cada usuario se utilizaron para entrenamiento y el 20 % restante para test. A la hora de seleccionar los días se tuvieron dos consideraciones:

· Se observó un error ligeramente más alto si los días se escogían por orden cronológico frente a escogerlos al azar. Para evitar esto, primero se aleatorizaron los días a seleccionar.

· No se contabilizaron los días que por algún motivo no tenían ninguna muestra validad, ya fuera porque ninguna era superior a 10 puntos, el usuario detuviera la toma de datos, o por algún error de formato los archivos .CSV no fueran validos. De no tener en cuenta estos factores el porcentaje de días de entrenamiento y de test no sería real, y el experimento no sería válido.

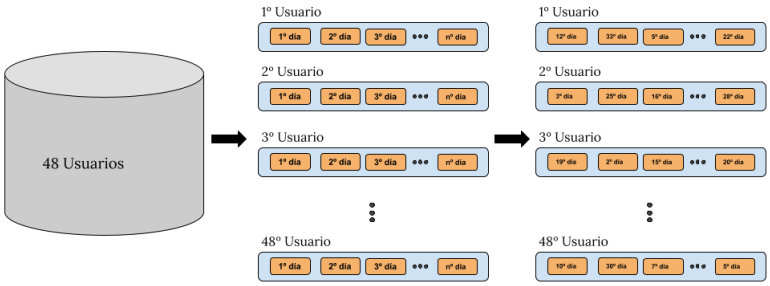


Figura 4.2: Aleatorización

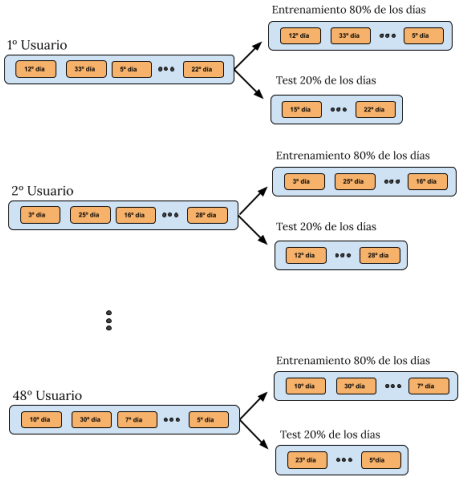


Figura 4.3: Reparto en conjunto cerrado

Una vez tenemos los días repartidos en dos grupos de entrenamiento y test, se extraen las

muestras de estos días y se procesan de acuerdo con lo visto en la sección anterior.

Entrenamiento de la red

Tanto para el experimento de conjunto cerrado como de conjunto abierto se han utilizado redes siamesas basadas en redes LSTM. Una vez tenemos las características procesadas, las agrupamos por batchs de entrenamiento.

Cada batch consistirá en 512 pares de muestras. Como se vio anteriormente, hay dos tipos distintos de pares, un par positivo, de dos muestras escogidas al azar que pertenecen al mismo usuario, y un tipo negativo en el que se escogerán dos muestras al azar de usuarios diferentes. La selección del tipo de par se hace también al azar asignando a cada caso un 50 % de probabilidad, hasta rellenar el lote de 512 pares. La red se entrenará durante 35 épocas, donde cada época consiste en 200 batch.

Para entrenar la red, se introducen consecutivamente estos pares de cada lote en la red siamesa. Para cada par devolverá dos vectores de características de un tamaño igual al número de neuronas de la red, en este caso al haber elegido 2 redes concatenadas bidireccionales de 64 neuronas LSTM serán dos vectores de salida de longitud igual 128.

Para estos dos vectores se calculará una distancia euclídea entre ellos, y sobre esta distancia se aplicará la función de pérdida pair-waise loss, intentado maximizar la distancia en los casos negativos y minimizarla en los positivos, con un margen α de 1.5.

Testado de la red. Cálculo de vectores de predicción

Una vez entrenada la red, se procederá a testarla de forma exhaustiva:

· Primero se pasarán por la red todas las muestras destinadas a entrenamiento del primer usuario.

· Después se pasarán por la red todas las muestras destinadas a test de ese mismo usuario.

· Se calculará la distancia euclídea entre la matriz de salida de las muestras de entrenamiento con la matriz de salida de las muestra de test y se calculará la media de las distancias obtenidas por muestra de test. Este nuevo vector de medias contiene las medias de las distancias entre cada muestra de test y las muestras de entrenamiento. Al ser de un mismo usuario, se almacenarán en un vector de predicción genuino, que se usará posteriormente para calcular el error.

· Seguidamente, se toman las muestras de test del siguiente usuario y se realiza el mismo proceso explicado, con la diferencia de que ahora el vector de medias se almacena en un vector de predicción impostor, puesto que las muestras de entrenamiento seguirán siendo del primer usuario pero las de test no. Se repite este proceso para todas las muestras de test de todos los usuarios y se concatenan los vectores de medias en el vector de predicción impostor.

· Por último, cuando se han testado todas las muestras de entrenamiento del primer usuario con todas las muestras de test de todos los usuarios, se toman como muestras de entrenamiento las del segundo usuario y se repite todo el proceso, concatenando los resultados en los vectores de predicción.

En la figura 4.4 se presenta el proceso de obtención de estos vectores de predicción para el primer usuario. Este proceso se repite hasta que **se haya pasado por todos los usuarios**.

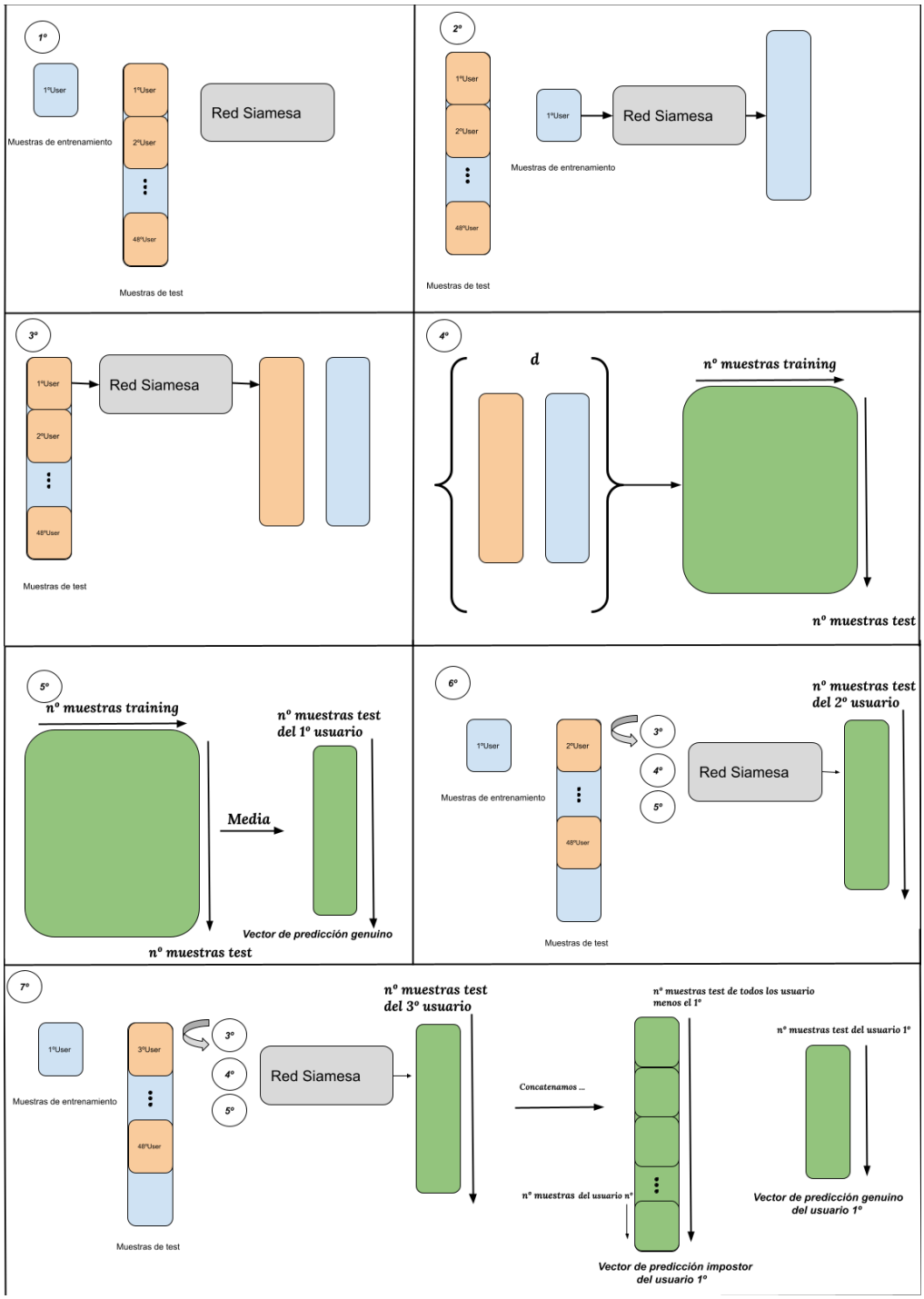


Figura 4.4: Proceso de obtención de vectores de predicción para el primer usuario

Testado de la red. Cálculo del error

Una vez tenemos los vectores genuino e impostor necesitamos calcular un valor límite de la distancia a partir del cual podamos determinar si una autenticación es errónea o no, y en base a este límite calcular cuántas autenticaciones han sido erróneas. Para obtener este límite utilizamos el valor mínimo y máximo de entre los dos vectores, genuino e impostor, entre estos dos valores estará el límite buscado, para encontrarlo calculamos el False Acceptance Rate (FAR), esto se hace contado para cada posible valor del límite, cuántos casos serían erróneamente autenticados como positivos, es decir cuántos valores son menores que el valor límite, (puesto que el algoritmo

pair-wise intenta hacer que las distancias de muestras genuinas sean cercanas a 0) y dividiendo por el número de impostores.

$$FAR = \frac{\text{Nº Falsos aceptados}}{\text{Nº de Impostores}} \qquad \text{para cada posible valor límite}$$

Y de igual forma calculamos el False Rejection Rate (FRR) contando el número de usuarios genuinos que serían erróneamente considerados impostores para cada posible valor límite, es decir, contar cuántos valores quedan por encima de cada posible valor límite.

$$FRR = \frac{\text{Nº Falsos rechazados}}{\text{Nº de Genuinos}} \qquad \text{para cada posible valor límite}$$

Una vez tenemos las curvas FAR y FRR, buscamos el valor límite que reduce al mínimo los FRR y FAR (21). Se puede obtener analíticamente restando las dos curvas y buscando su valor mínimo, ese será el error del sistema, como se ve en la siguiente figura:

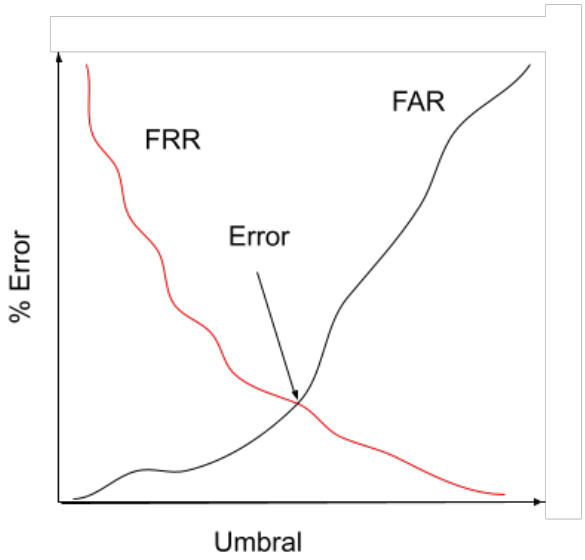


Figura 4.5: Calculo del error en base a las curvas FRR y FAR

4.3. Experimento conjunto abierto

Como se comentó anteriormente, la diferencia entre el experimento de conjunto abierto y de conjunto cerrado reside en la cantidad de usuarios que se usan para entrenar la red. En el experimento de conjunto abierto un 20 % de los usuarios no se usan en entrenamiento y son utilizados exclusivamente para test. lo que supone un reto más difícil para la red.

Como se ve en la figura 4.6, el 80 % de usuarios son usados en entrenamiento y test tal y como se ha descrito en el experimento de conjunto cerrado. Para el 20 % restante sus muestras se usan únicamente para test, se dividen sus días en dos mitades iguales y se testa de la misma forma que en conjunto cerrado (de forma exhaustiva, ver figura 4.7), pensado en la primera mitad como si fueran las muestras de entrenamiento y la segunda mitad como las muestras de test. Los usuarios excluidos del entrenamiento son escogidos al azar para cada experimento.

El error se calcula de la misma forma que en el caso cerrado:

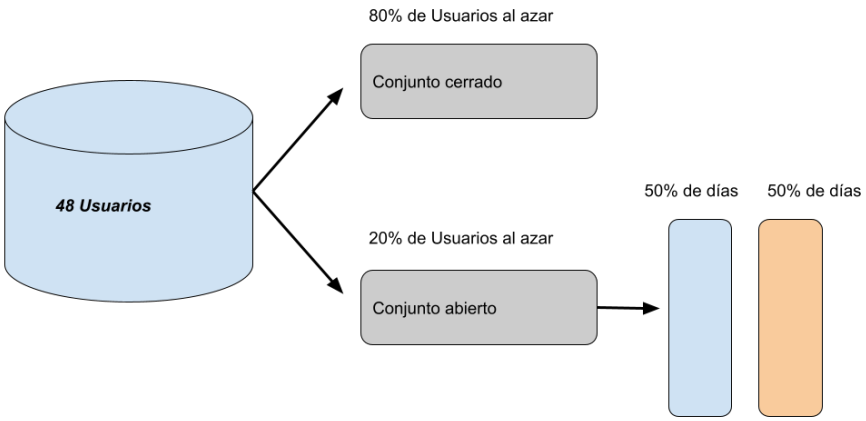


Figura 4.6: Reparto de usuarios en el experimento de conjunto abierto

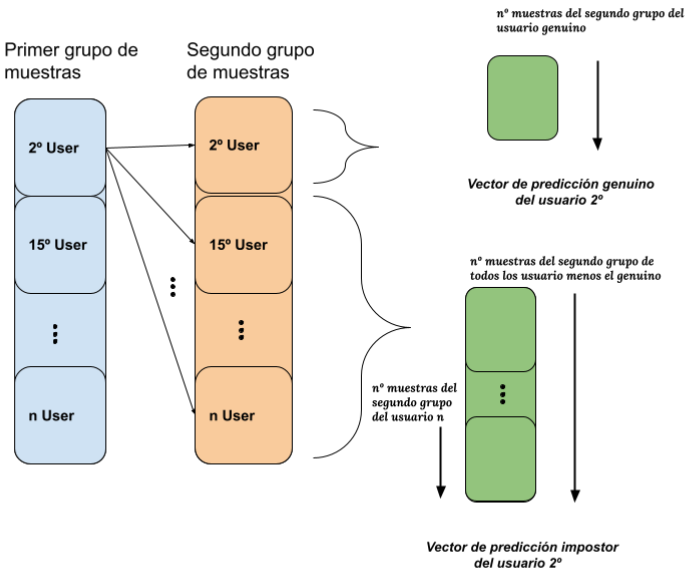


Figura 4.7: Test en experimento de conjunto abierto

4.4. Resultados

Para el experimento de conjunto cerrado se hicieron dos realizaciones, mientras que para el de conjunto abierto solo se presenta un caso de ejemplo. Todas las gráficas se dan en función de

las épocas (eje horizontal).

Resultados conjunto cerrado

Experimento en conjunto cerrado	
Porcentaje de días utilizado para entrenamiento	80 %
Porcentaje de días utilizado para test	20 %
Normalización de Batch	Si
Tamaño del Batch	512
Batches por época	200
Épocas	35
Tamaño mínimo de muestra	10
Tamaño de cada muestra	35
Primera LSTM	
Nº de unidades LSTM	64
Recurrent dropout	0.4
Segunda LSTM	
Nº de unidades LSTM	64
Recurrent dropout	0.4
Optimizador de Adam	
Tasa de aprendizaje (α)	0.01
β_1	0.9
β_2	0.999
ε	10^{-8}

Cuadro 4.2: Hiperparámetros del experimento cerrado

Para el primer experimento de conjunto cerrado tenemos los siguientes resultados:

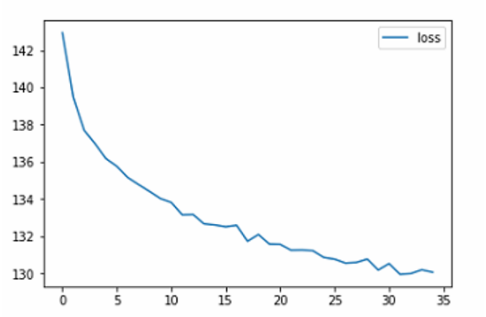


Figura 4.8: Función de pérdida del primer experimento de conjunto cerrado

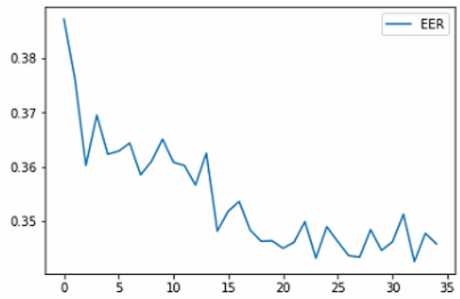


Figura 4.9: Error del primer experimento de conjunto cerrado

Se observa una correcta minimización de la función de pérdida y un error 34 %

Para el segundo experimento:

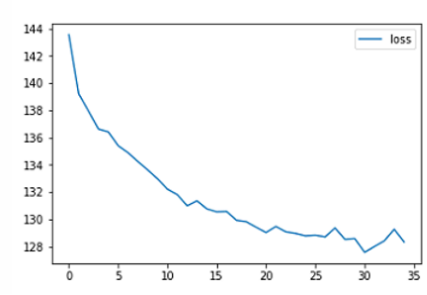


Figura 4.10: Función de pérdida del segundo experimento de conjunto cerrado

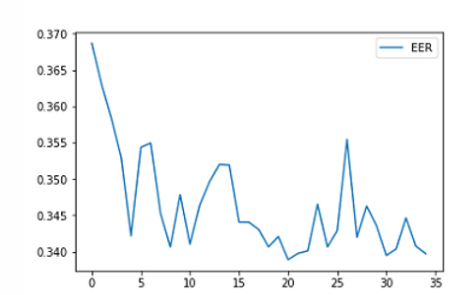


Figura 4.11: Error del segundo experimento de conjunto cerrado

Se observa también una correcta minimización de la función de pérdida y un error del 34 %

Resultados conjunto abierto

Experimento en conjunto abierto	
Porcentaje de usuarios utilizados para entrenamiento	80 %
Porcentaje de días utilizado para entrenamiento	80 %
Porcentaje de días utilizado para test	20 %
Normalización de Batch	Si
Tamaño del Batch	512
Batches por época	200
Épocas	25
Tamaño mínimo de muestra	10
Tamaño de cada muestra	35
Primera LSTM	
Nº de unidades LSTM	64
Recurrent dropout	0.5
Segunda LSTM	
Nº de unidades LSTM	64
Recurrent dropout	0.5
Optimizador de Adam	
Tasa de aprendizaje (α)	0.01
β_1	0.9
β_2	0.999
ε	10^{-8}

Cuadro 4.3: Hiperparámetros del experimento abierto

En el experimento de conjunto abierto se redujo el número de épocas al haber reducido el número de datos de entrenamiento. Para el experimento de conjunto abierto tenemos:

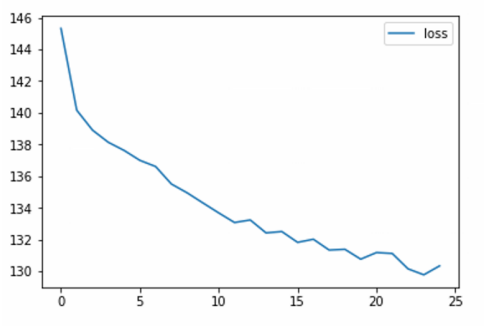


Figura 4.12: Función de perdida del experimento de conjunto abierto

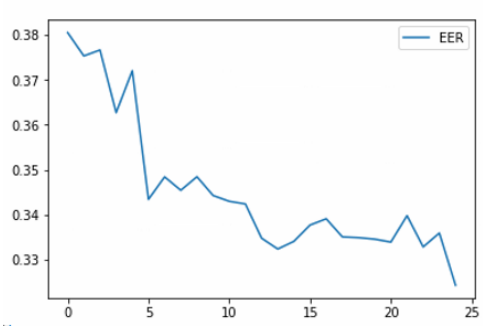


Figura 4.13: Error del experimento de conjunto abierto, en la parte cerrada

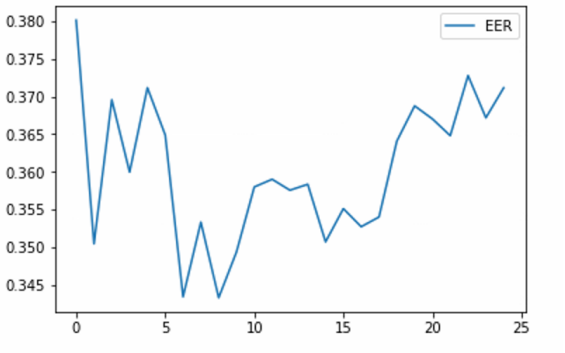


Figura 4.14: Error del experimento de conjunto abierto, en la parte abierta

Vemos que la función de pérdida se reduce adecuadamente la parte de test del conjunto cerrado tiene un error similar al alcanzado en el experimento de conjunto cerrado, pero el test de conjunto abierto no se estabiliza, y oscila entre 38 % y el 34 %. se observó el mismo resultado con gran variabilidad entre distintas realizaciones.

Sumario de los experimentos

Resultados	
Experimento	**Error**
1º Conjunto cerrado	34 %
2º Conjunto cerrado	34 %
Conjunto abierto, parte cerrada	33 %
Conjunto abierto, parte abierta	34-37 % (No concluyente)

Cuadro 4.4: Sumario de los experimentos

5

Conclusiones y trabajo futuro

Conclusiones

Podemos concluir que la red es capaz de aprender a discernir con cierto grado de éxito entre usuarios genuinos e impostores en el experimento de conjunto cerrado. En el experimento de conjunto abierto el error no se estabiliza, y por lo tanto no podemos extraer conclusiones. Los resultados obtenidos están lejos de poder ser utilizados en un escenario real.

Trabajo futuro

Una de las dificultades que se ha encontrado en este trabajo reside en el hecho de que no podíamos comparar swipes de tamaños diferentes, lo que nos lleva a tener que procesar las muestras eliminando el concepto de swipe (si optamos por concatenar muestras) o alterando la señal (si optamos por remuestrear, ver figura 4.1).

Una posibilidad que solventaría este problema es renunciar a utilizar redes neuronales que estudien secuencias y en su lugar tratar los distintos swipes como imágenes en redes CNN. Esto nos permitiría comparar swipes de distintos tamaños sin alterar la señal,e incluso se podrían agrupar las imágenes en secuencias y estudiarlas combinando redes CNN y RNN, de forma que podríamos incluir también la evolución temporal.

Bibliografía

[1] Publication-ready nn-architecture schematics. URL: http://alexlenail.me/NN-SVG/.

[2] Jason Brownlee. Difference between a batch and an epoch in a neural network, 2019. URL: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/.

[3] Raúl Gómez Bruballa. Understanding ranking loss, contrastive loss, margin loss, triplet loss, hinge loss and all those confusing names. URL: https://gombru.github.io/2019/04/03/ranking_loss/.

[4] Cburnett. Artificial neural network. URL: https://en.wikipedia.org/wiki/Artificial_neural_network#/media/File:Artificial_neural_network.svg/.

[5] Incfk8. Structure of rnn and brnn. URL: https://en.wikipedia.org/wiki/Bidirectional_recurrent_neural_networks#/media/File:Structural_diagrams_of_unidirectional_and_bidirectional_recurrent_neural_networks.png.

[6] Andrej Larpathy. The unreasonable effectiveness of recurrent neural networks. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[7] Laughsinthestocks. Activation rectified linear. URL: https://en.wikipedia.org/wiki/Activation_function#/media/File:Activation_rectified_linear.svg.

[8] LG. Parte frontal, trasera y perfil del nexus 5., 2013. URL: https://www.flickr.com/photos/lge/10600449773/.

[9] Upal Mahbub, Sayantan Sarkar, Vishal M. Patel, and Rama Chellappa. Active user authentication for smartphones: A challenge data set and benchmark results. *2016 IEEE 8th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, 2016.

[10] Shervin Minaee, Amirali Abdolrashidi, Hang Su, Mohammed Bennamoun, and David Zhang. Biometric recognition using deep learning: A survey. 2020.

[11] Andrew Ng, Kian Katanforoosh, and Bensouda Mourri. Convolutional neural networks. week 4 [mooc]. URL: https://www.coursera.org/learn/convolutional-neural-networks/home/week/4.

[12] Andrew Ng, Kian Katanforoosh, and Bensouda Mourri. Neural networks and deep learning [mooc]. URL: https://www.coursera.org/learn/neural-networks-deep-learning.

[13] Andrew Ng, Kian Katanforoosh, and Bensouda Mourri. Sequence models [mooc]. URL: https://www.coursera.org/learn/nlp-sequence-models/home/welcome.

[14] Numpy and Scipy Documentation. scipy.signal.resample, 2020. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.resample.html.

[15] Christopher Olah. Understanding lstm networks. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[16] Vishal M. Patel, Rama Chellappa, Deepak Chandra, and Brandon Barbello. Continuous user authentication on mobile devices: Recent progress and remaining challenges. *IEEE SIGNAL PROCESSING MAGAZINE, VOL. X, NO. X*, 2016.

[17] Gaurav Rajpal. Underfitting vs overfitting (vs best fitting) in machine learning. URL: `https://medium.com/analytics-vidhya/underfitting-vs-overfitting-vs-best-fitting-in-machine-learning-1af6c6961d89`.

[18] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. Recurrent dropout without memory loss. *Universitat zu Lubeck, Institut fur Neuro- und Bioinformatik, Google Research*, 2016.

[19] Piotr Skalski. Deep dive into math behind deep networks, 2018. URL: `https://towardsdatascience.com/https-medium-com-piotr-skalski92-deep-dive-into-deep-networks-math-17660bc376ba`.

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 2014.

[21] Carmen Sánchez Ávila. Aplicaciones de la biometría en seguridad. ciclo de conferencias upm tassi (temas avanzados en seguridad y sociedad de la información), 2012. URL: `http://oa.upm.es/20071/`.