Universitat Autònoma de Barcelona

Trabajo final de grado

Grau de Ingeniería Electrónica de Telecomunicaciones

Desarrollo de Modelos de Redes Neuronales para la Identificación Eficiente de Señales Aleatorias

Pablo Gutiérrez Bueno

Director: Javier Martín Martínez

Departamento de telecomunicaciones e Ingeniería de Sistemas

Escola d'Enginyeria

Universitat Autònoma de Barcelona (UAB)

Febrero de 2024

Contenido

1. Agradecimientos 4
2. Introducción 5
3. Fundamentos teóricos y señales RTN 6
   3.1. ¿Qué es la aleatoriedad? 6
   3.2. PseudoRandom Number Generator 10
   3.3. Señales RTN 13
4. Redes neuronales y aprendizaje automático 16
   4.1. ¿Qué es una red neuronal? 16
   4.2. Red neuronal Feedforward 17
   4.3. RNN – Long Short-Term Memory (LSTM) 21
5. Desarrollo del proyecto 27
   5.1. Preprocesamiento de los datos 27
   5.2. Red neuronal Feedforward 34
   5.2.1. Diseño y entrenamiento 34
   5.2.2. Prueba de la red con nuevos datos 39
   5.3. Red neuronal LSTM 39
   5.3.1. Basada en búsqueda de patrones 40
   5.3.2. Basada en cálculo de porcentaje individual 41
6. Aplicaciones 42
   6.1. Hipótesis 44
7. Conclusión 45
8. Bibliografía 46

1. Agradecimientos

Antes de comenzar, me gustaría agradecer Dr. Javier Martin Martínez por la guía y ayuda durante este proyecto. Al inicio, mi perspectiva era limitada y mis ideas, aunque llenas de entusiasmo, carecían de la dirección necesaria. Ha sabido hacerme ver un campo desconocido para mí y que ha hecho que me replantee ideas y conceptos que pocas veces solemos parar a pensar.

Gracias también a mi familia, pareja y amigos por esas tardes que no pude estar con ellos, por esos "más adelante" o el "ya lo haremos", este proyecto requiere muchas más horas de las que yo he podido dedicar.

Con este proyecto cierro una etapa, sin lugar a duda la mejor etapa educativa que he tenido. El hambre de conocimiento día tras día en las aulas, la complejidad de la teoría, los exámenes imposibles y los proyectos inacabables han sentado una visión del mundo diferente que ahora, habrá que asumir. Nuevos retos, experiencias y aventuras nos esperan y saber que el futuro está en nuestras manos aviva la llama que tenemos dentro.

Siento que este proyecto puede ser grande y único, y que lo retomare para dar un punto final durante el Trabajo Final de Máster, tengo delante de mí un campo capaz poco conocido que puede traer grandes beneficios, es por ello por lo que seguiré aprendiendo e informándome dado que toca de lleno la rama a la que siempre me he querido dedicar.

Atentamente

Pablo Gutiérrez Bueno

2. Introducción

El proyecto tiene como objetivo el desarrollo de una red neuronal capaz de separar señales binarias según la aleatoriedad que estas tengan.

En un instante se estudiará la idea de aleatoriedad y azar, seguido de diferentes casos históricos de algoritmos usados con el fin de crear datos aleatorios para conocer mas en profundidad como trabajan. Pararemos a ver los diferentes tipos de redes neuronales, diferencias con la IA y características y veremos el estudio practico realizado, a demás de los resultados, soluciones e hipótesis. Finalmente se evaluará la necesidad actual de este tipo de redes, las aplicaciones, ventajas y avances que pueden ofrecernos.

Antes de empezar, debe informarse que este trabajo no esta finalizado, es una primera parte para posteriormente hacer un estudio mas riguroso para un Trabajo Final de Máster o Doctorado que, sin estos conocimientos no serian posibles de realizar en una instancia.

El proyecto ha sido íntegramente desarrollado por Matlab, si bien es cierto que Python es mas popular y ágil a la hora de entrenar redes neuronales, Matlab es mas que suficiente para adentrarse en el estudio y aplicar los conocimientos aprendidos durante el grado.

3. Fundamentos teóricos y señales RTN

3.1. ¿Qué es la aleatoriedad?

La aleatoriedad es la cualidad de aleatorio, que significa que depende del azar, y el azar, según la RAE significa *sin rumbo ni orden,* es decir, cualquier suceso que no se basa ni es afectado por acciones del pasado o del futuro.

¿Es correcta esta idea?

Si, tiene un significado y por tanto un sentido, pero el concepto del azar es algo que aún no logramos comprender.

Si el azar no existe, la aleatoriedad tampoco, por lo que podemos decir que lo que va a pasar en unas horas, el día de mañana, la semana que viene o de aquí cinco años, no es aleatorio. Podemos formular esta idea ya que, actualmente, sabemos que el universo que conocemos este compuesto por doce partículas fundamentales que interactúan de cuatro formas predecibles.

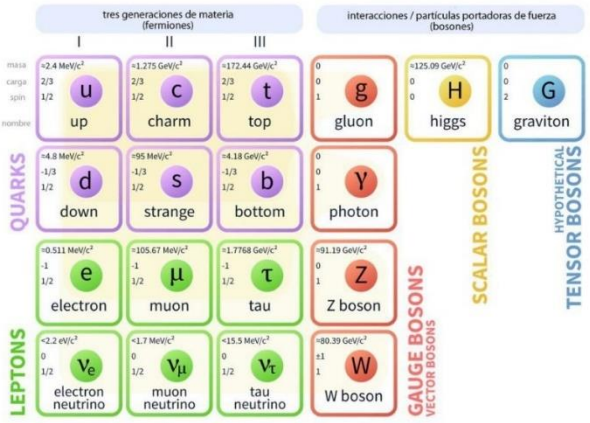

Ilustración 3.1 partículas fundamentales y partículas portadoras

---

Si pudieras saber dónde está todo y a qué velocidad se mueve sabrías el futuro del universo dado que sabrías como cada partícula interactúa con las otras, afirmando que nada es impredecible y por tanto no existe la aleatoriedad.

- Pierre-Simon Laplace -

Este concepto puede aplicarse también al comportamiento humando dado que estamos formados de las mismas doce partículas con las mismas cuatro interacciones, un ejemplo muy claro es saber el estado anímico o la reacción que va a tener una persona cercana a cierta situación, comúnmente se dice “como te conozco” pero científicamente hablando, las partículas de la otra persona están influyendo en tus partículas, accionando tus campos

y moviendo información en tus neuronas para hacerte sentir como esta o reaccionara esa persona en ese instante, ese es un claro ejemplo de ver el futuro.

Todo lo que hagamos o hicimos está determinado por la información de ese instante.

¿Qué es la información?

La información parece ser, fundamentalmente, el orden, podemos ver como el orden de las moléculas de ADN contiene la información necesaria para crear a un ser vivo, al igual que el orden de los 0s y 1s circulando a través de internet permiten obtener toda la información requerida para reproducir un video, una canción o escribir. Nada sucede por arte de magia, en la tecnología, a diferencia del universo, somos nosotros quienes dictamos la posición de las partículas, o en este caso, del código binario.

Por ejemplo, si la información en forma binaria no pudiera sobrescribirse, borrar una letra sería algo así:

			Acciones
Original	Byte	01000010 01111001 01110100 01100101	4
Borrado	Byt|e	01000010 01111001 01110100 01100101 00001000	5
Final	Byte	01000010 01111001 01110100 01100101 00001000 01100101	6

Tabla 3.1 Ejemplo de uso del código binario en memorias sin capacidad de almacenamiento

Una línea de código infinita donde podría verse todas las acciones que se han ejecutado gracias al código ASCII. Escribir determinada letra, espaciar, retroceder, hacer un enter… Todas estas acciones quedarían grabadas en un historial binario infinito, en donde no podría entenderse nada. Gracias a la capacidad de guardar información, sobrescribir, eliminar, etc. Somos capaces de no ver ese historial de las acciones que se llevaron a cabo para teclear.

Podemos afirmar que la información es orden, dado que el orden de los 0s y 1s crean códigos, que se interpretan por letras, las letras forman palabras, las palabras oraciones y por último la información.

Que la implicación implique orden se conoce como *regularidad*, pero, retomando el tema principal, el orden no es aleatorio.

El fundador de la teoría matemática de la información, Claude Shannon estimó un 75% de redundancia en el inglés dado que no todas las letras contienen la misma cantidad de información y por tanto no son aleatorias. En inglés es susceptible pensar que después de *th* vamos a encontrar una *e,* o que después de *q* encontraremos una *u*. Este simple análisis deja ver que la *e* o la *u* tiene una mayor probabilidad de aparecer después de estas letras, indicando que tienen muy poca información debido a la facilidad de predecirse de antemano.

Albert Einstein creía que la información, por muy grande que fuera, puede comprimirse, por ejemplo, leamos esta frase:

¡s tú pds lr est, pdrs cnsgr un trbj my bn rmunrd!

Leer y entender la frase no es por el azar o por inteligencia, esta frase puede comprimirse sin dejar de informar lo que quiere informar porque la información no es aleatoria, tiene patrones.

Pasa lo mismo en un video, que podamos ver a través de una pantalla se debe a que todos los pixeles tienen un orden que seguir para dar la información que quieres ver.

Si se quiere jugar con esto se puede hacer *datamoshing* que se usa en la edición de video para alterar la información de un video a los pixeles de otro.

Ilustración 3.2 Datamoshing

En esta imagen forma parte de un video en movimiento, donde, en ciertos momentos se altera el color de ciertos pixeles, alterando la información. Es común que los pixeles alterados sean otras imágenes, con el objetivo de que en el video original puedan apreciarse varias imágenes superpuestas. Como curiosidad, no todo el mundo es capaz de ver estas imágenes secundarias que se aplican, al fin y al cabo, ningún ojo humano ve por igual los colores, por lo que pequeños cambios de tonalidad en según que pixeles pueden generar debates adversos.

Volviendo a la frase que describimos en la página anterior podemos dejar claro que la información puede comprimirse, ¿pero hasta dónde? Sabemos que cualquier cosa que no sea aleatoria puede ser comprimida, los patrones o regularidades pueden comprimirse

debido a que son predecibles, por lo que cualquier información puede ser comprimida hasta que sea aleatoria, y ese pequeño dato comprimido contendrá toda la información original pero *destilada* o, mejor dicho, como *información pura*, implicando que esto es aleatoriedad, por lo que si queremos saber cuánta información tiene algo debemos saber cuan aleatorio es, es decir, que desorden tiene o mejor dicho que *entropía* tiene esa información pura.

La idea de que la información es entropía puede verse fácilmente en la disposición de la información de un disco duro:

		Se repite
Disco duro sin información	00000000 00000000 00000000 00000000	0
Disco duro con información	10011001 10011001 10011001 10011001	1001
Disco duro información aleatoria	01000111 00000101 10110100 01011101	Nada

Tabla 3.2 Ejemplo de cómo la información es entropía

Podemos ver que el último disco duro, contiene mucha más información que el resto, pero no tiene orden, afirmando que la información es entropía. Por lo que la cadena binaria que contiene más información es la que contiene una cantidad aleatoria de ceros y unos, ya que no puede ser comprimida, las dos primeras pueden enviarse únicamente con su regularidad, es decir, 0 o 1001, mientras que la última no puede ser comprimida y si se quiere conocer toda la información estas obligado a enviar toda la cadena de números.

Esto no tiene sentido para los humanos, un claro ejemplo es el ruido blanco en donde no hay un orden binario y la señal simplemente se limita a moverse sin ningún patrón.

Si volvemos a la cita de Laplace podemos entender que tiene una pequeña conjetura, si su teoría es cierta implica que la información en el universo es siempre la misma, pero ahora, después del paso de los años y de conocer la entropía, sabemos que la información en el universo aumenta con el tiempo, por lo que realmente no podemos predecir el futuro dado que ese futuro tiene más información que nuestro presente.

Esto se ha podido observar en la mecánica cuántica donde, a través de experimentos, no podemos predecir donde va a estar un electrón, pero si podemos calcular donde es más probable que aparezca, la falta de precisión siempre estará ahí, dado que ese electrón tiene información que nosotros no teníamos al momento de predecirlo.

La mecánica cuántica puede comprimirse más, mucho más. Aún no hemos conseguido que la información sea aleatoria, por lo que aún no tenemos la información pura que predice el futuro.

- Albert Einstein-

Sin embargo ¿Acaso es posible comprimir más la mecánica cuántica? La respuesta, hoy, es un no, dado que la mecánica cuántica ya trabaja por aleatoriedad.

3.2. PseudoRandom Number Generator

Los Generadores de Números Pseudoaleatorios, también conocidos como DRBG (Deterministic Random Bit Generators), son algoritmos diseñados para producir secuencias de números cuyas propiedades se asemejan a las de secuencias aleatorias, pero no lo son. Estos generadores no son verdaderamente aleatorios, ya que dependen de un valor inicial o *semilla*, la dependencia de cualquier valor pierde aleatoriedad, pero para las aplicaciones de uso actuales, es más que suficiente.

Funcionamiento básico de los PRNGs:

- Inicialización con una Semilla: Comienza con un valor inicial o semilla. Este valor puede ser un número fijo, el resultado de alguna medición (como la hora del sistema), o derivado de alguna otra fuente.
- Algoritmo de Generación: Utiliza un algoritmo matemático para generar una nueva secuencia de números a partir de la semilla. Este algoritmo es determinista, lo que significa que, dada una semilla específica, siempre producirá la misma secuencia de números.
- Secuencia Pseudoaleatoria: Los números generados por este proceso tienen la apariencia de ser aleatorios. Esto significa que pasan ciertas pruebas estadísticas para la aleatoriedad, como tener una distribución uniforme o carecer de patrones predecibles.
- Repetición y Periodicidad: Aunque los son buenos imitando la aleatoriedad, eventualmente la secuencia se repetirá, ya que el número de estados posibles es finito (limitado por su arquitectura, como el número de bits en su representación). Este período puede ser extremadamente largo para algoritmos bien diseñados.

El siguiente diagrama de flujo, describe un proceso para analizar el rendimiento aleatorio de secuencias generadas por mapas caóticos. Empieza con valores iniciales que alimentan dos procesos: un mapa de Chebyshev y un mapa de CML (Coupled Map Lattices). El mapa de Chebyshev, tras pasar por una función de retraso temporal, genera secuencias de Chebyshev. Por otro lado, el mapa CML produce secuencias CML y se mejora a través de un "Improved CML". Este proceso mejorado también contribuye al mapa CML original.El mapa CML y el mapa de Chebyshev se combinan en un mapa DCML (Delayed Coupled Map Lattices), que a su vez produce sus propias secuencias. Tanto las secuencias de Chebyshev como las DCML se analizan para evaluar el rendimiento aleatorio.

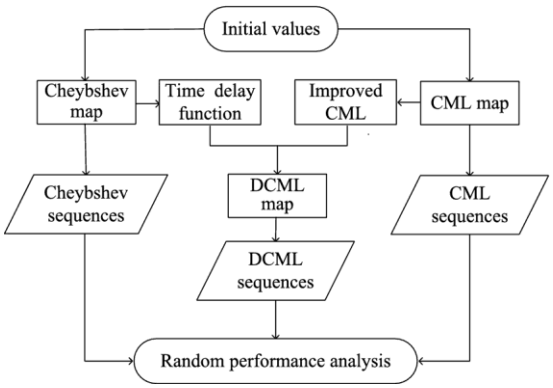

Ilustración 3.3 Diagrama de flujo de un PRNG basado en mapas caóticos

Son de gran utilidad en múltiples aplicaciones debido a su rapidez y reproducibilidad y suele acompañarse de un HRNG (Hardware Random Number Generator), que funcionan generando números aleatorios a partir de procesos físicos, generalmente cuánticos o térmicos, en lugar de utilizar algoritmos matemáticos como en los PRNGs.

Funcionamiento básico de los HRNGs:

- Procesos Físicos: Se basan en fenómenos físicos impredecibles como el ruido térmico, ruido cuántico, procesos radioactivos, o efectos relacionados con la mecánica cuántica como el efecto túnel.
- Medición y Conversión: El dispositivo mide estas variaciones físicas, que son intrínsecamente aleatorias, y las convierte en datos digitales. Por ejemplo, podría medir las fluctuaciones en la tensión eléctrica o en la intensidad de la señal.
- Digitalización: La señal analógica resultante de este proceso es entonces digitalizada, generalmente en binario.
- Post-Procesamiento: A menudo, los datos crudos generados de esta manera son procesados para mejorar ciertas características como la uniformidad y la independencia estadística. Sin embargo, este paso debe realizarse con cuidado para no introducir patrones predecibles en los datos.

El siguiente diagrama de flujo describe un HRNG basado en un oscilador de Josephson.

El proceso comienza con una corriente de entrada que se convierte a pulsos SFQ (Single Flux Quantum) mediante un convertidor DC/SFQ. Estos pulsos SFQ activan un flip-flop de tipo toggle a través de una entrada de disparo. Paralelamente, los pulsos SFQ continuos generados por el oscilador de Josephson se introducen en el sistema. El flip-flop toggle cambia su estado con cada pulso de reloj, produciendo dos salidas complementarias (Q y $\bar{Q}$). Estas salidas se alimentan junto con la señal de reloj a una puerta AND, que finalmente emite un número aleatorio (0 o 1) como salida.

Este mecanismo es típico en la criptografía y sistemas de comunicación seguros que requieren fuentes de aleatoriedad confiables.

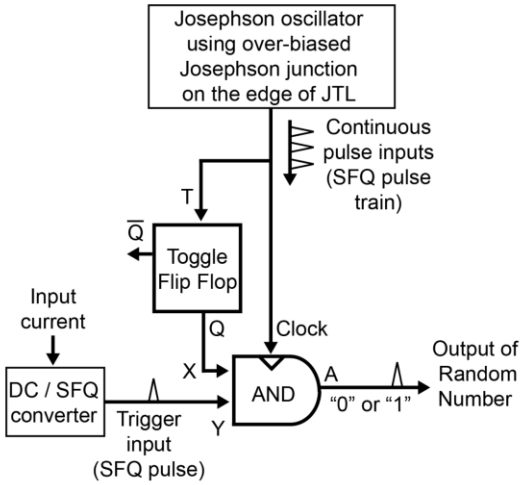

Ilustración 3.4 Diagrama de flujo de un HRNG basado en un oscilador Josephson

Los HRNGs son valorados por su alta entropía y falta de previsibilidad, lo que los hace ideales para aplicaciones como la criptografía, donde la seguridad depende de la imposibilidad de predecir la clave generada. A diferencia de los PRNGs, que pueden, en teoría, reproducir secuencias si se conoce su estado inicial, los HRNGs no sufren de este problema debido a la naturaleza fundamentalmente impredecible de los procesos físicos que utilizan.

Sin embargo, los HRNGs pueden ser más lentos en la generación de números que los PRNGs y pueden requerir hardware especializado, lo que limita su uso en algunos contextos. Además, su correcto funcionamiento depende de la calidad y el mantenimiento del hardware, así como de la precisión en la medición de los fenómenos físicos.

¿Para qué se usan los PRNG?

Estos generadores son fundamentales en aplicaciones como la simulación (con el método Montecarlo), los videojuegos (para generar entornos y eventos procedimentales) o la criptografía. Para este último, requerimos el uso de PRNGs más complejos y elaborados donde su salida no es predecible en base a las salidas obtenidas con anterioridad, además de uso de HRNGs para complementar las salidas del software.

Se requiere un cuidadoso análisis matemático para tener la certeza de que nuestro PRNG genera números lo suficientemente aleatorios para adaptarse al uso previsto.

Ventajas y desventajas

Son rápidos, eficientes y reproducibles, lo que hace que sean ideales para aplicaciones que necesitan generar grandes cantidades de números pseudoaleatorios de manera constante. Pero tiene cuatro problemas principales:

- El valor inicial
- Falta de uniformidad en la distribución de grandes cantidades de números generados.
- Correlación de valores sucesivos
- Mala distribución dimensional de la secuencia de salida
- Las distancias entre los lugares donde se producen determinados valores se distribuyen de forma diferente a las de una distribución de secuencia aleatoria.

Aun conociendo estos y más problemas que podemos llegar a encontrarnos, el principal, obviando la semilla, es que los defectos pueden calificarse de impredecibles a muy evidentes.

Un ejemplo histórico de las limitaciones de los PRNGs es el algoritmo RAND-U, de IBM, un generador de números pseudoaleatorios del tipo lineal congruencial, usado principalmente en la década de los 60. Se define por una recurrencia específica y genera enteros pseudoaleatorios uniformemente distribuidos. Sin embargo, es ampliamente considerado como uno de los generadores de números aleatorios más mal concebidos, fallando notablemente en la prueba espectral para dimensiones mayores a 2, uno de los métodos utilizados para evaluar la calidad de los PRNGs. Su función es examinar como los números generados llenan un espacio tridimensional o de dimensiones superiores. Un buen PRNG debería llenar el espacio de manera uniforme y aleatoria, por lo que las creaciones de patrones en ciertas áreas indica falta de aleatoriedad y uniformidad. Los valores usados para el multiplicador y el módulo fueron elegidos por conveniencia computacional, no por calidad estadística. Esto resultó en que muchos resultados científicos de la época en que se usó son vistos con sospecha.

Con esta premisa, podemos asegurar que la lista de generadores ampliamente utilizados que deberían descartarse es mucho más larga que la lista de generadores buenos.

Durante el último decalustro se ha recomendado a las empresas no comprar RNG sin antes haberlo probado en su sistema. Seguro que todos conocemos la empresa Java y por ende su lenguaje de programación. Hasta 2020 ha estado confiando en un LCG (Generador Congruente Lineal) que es un RNG de baja calidad ya que genera pseudo números aleatorios basados en una función lineal definida a trozos discontinua, es decir, a diferencia de la primera descripción explicada, donde el problema la semilla, este RNG se basa en una función, determinando más el camino del próximo valor a calcular y por ende de perder aleatoriedad. A partir de la versión 17 Java cambió su RNG a JEP 356.

Un ejemplo de un buen PRNG conocido por evitar problemas importantes y funcionar muy bien pese al paso del tiempo es el *Mersenne Twister* que se publicó en 1998. Este generador es uno de los citados en la *List of Random Number Generators* que registra, desde el primer PRNG, los más importantes y usados generadores hoy en día.

3.3. Señales RTN

En el estudio de las señales y su aleatoriedad, es esencial detenernos a conocer más sobre las señales de Ruido Telegráfico Aleatorio (RTN), ya que serán las señales con las que entrenaremos a la red.

Las señales RTN se distinguen por sus cambios abruptos y aleatorios entre dos o más niveles fijos de voltaje o corriente, recordándonos al sonido de un telégrafo. Lo peculiar del RTN es que, aunque sus transiciones son aleatorias, los niveles entre los que cambia son constantes y discretos. Aquí podemos ver un ejemplo:

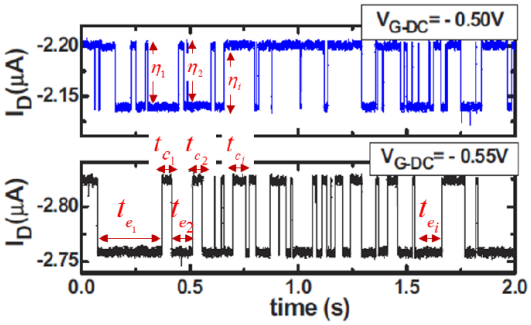

Ilustración 3.5 Ejemplo de señales RTN

Estas fluctuaciones de corriente están relacionadas con la captura y emisión de portadores de carga por defectos de óxido e interfaz, mostrando una gran dependencia de las condiciones de polarización y temperatura del dispositivo.

La variabilidad del fenómeno RTN aumenta inversamente con la escala de área, afectando negativamente, por ejemplo, a dispositivos analógicos y circuitos lógicos digitales como memorias flash.

El RTN no solo está presente en transistores MOSFET convencionales, sino también en memorias de acceso aleatorio resistivas (RRAM), FET completamente agotados (FD-SOI), FinFET, FETs de múltiples puertas y dispositivos y circuitos digitales de nanohilos.

Los FET mencionados son transistores de efecto de campo, un tipo que regula el flujo de corriente mediante un campo eléctrico. Se usan para amplificar o conmutar señales y, a diferencia de los transistores bipolares, los FET se controlan por la tensión aplicada a la puerta y modulan la conectividad entre drenador y fuente.

El origen del RTN se encuentra, como se ha comentado, en los dispositivos semiconductores y conductores, donde los electrones interactúan con defectos o impurezas en el material. A nivel microscópico, estos defectos actúan como trampas para los electrones, causando fluctuaciones en la corriente o el voltaje. Por ejemplo, en un semiconductor, estos defectos pueden ser sitios donde los electrones quedan temporalmente atrapados, alterando las propiedades eléctricas del material.

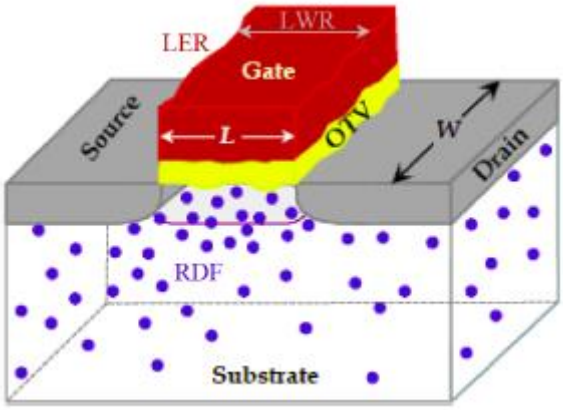

Ilustración 3.6 Defectos de un semiconductor

Desde un punto de vista matemático, el RTN se modela como un proceso estocástico, descrito por procesos de Markov o de Poisson. Estas herramientas estadísticas nos permiten describir las características del RTN, como la tasa de cambio entre estados y la duración promedio en cada estado.

Aunque a menudo consideramos el RTN como un tipo de ruido indeseado, su estudio es fundamental en diversos campos. Es clave para comprender y mejorar la fiabilidad y el rendimiento de los dispositivos. También se utiliza en la investigación de materiales y en el desarrollo de sensores.

En el contexto de nuestro proyecto, analizar las señales RTN significa identificar estas fluctuaciones características y diferenciarlas de otras formas de ruido o señales aleatorias. La detección precisa del RTN es desafiante debido a su naturaleza aleatoria y a la presencia de otros tipos de ruido. Para entrenar eficazmente una red neuronal en esta tarea, es crucial tener un conjunto de datos bien caracterizado y un diseño cuidadoso del algoritmo de aprendizaje.

El proyecto se ha desarrollado en base a señales RTN procedentes de transistores tipo P.

¿Por qué usar señales RTN de transistores tipo P y no de tipo N?

La razón principal de esta diferencia reside en las características inherentes de los materiales y la forma en que los portadores de carga se comportan en cada tipo de transistor. Desglosemos los puntos importantes:

- Diferencia en los portadores de Carga:

  En los transistores tipo N, los electrones son los portadores de carga principales. Éstos tienen una movilidad más alta que los huecos (portadores de carga en los transistores tipo P), lo que significa que pueden moverse más rápidamente a través del material semiconductor.

- Interacción con Defectos en el Material:

  Los electrones en los transistores tipo N son más susceptibles a interactuar con los defectos o impurezas en el semiconductor. Estas interacciones pueden causar fluctuaciones abruptas y aleatorias en la corriente eléctrica, que son la base del RTN.
  Por otro lado, debido a la menor movilidad de los huecos en los transistores tipo P, la probabilidad de interacción con defectos es menor, que se traduce en una menor incidencia de fluctuaciones aleatorias, y, por tanto, menos RTN.

- Efectos de la Densidad de Estados:

  La densidad de estados en la banda de conducción (donde se mueven los electrones en los transistores tipo N) es mayor que en la banda de valencia (donde se mueven los huecos en los transistores tipo P). Esto significa que hay más estados energéticos disponibles para los electrones, aumentando la probabilidad de interacciones que generan RTN.

- Influencia de la Tecnología de Fabricación:

  La tecnología y los procesos utilizados para fabricar transistores tipo P y tipo N también juegan un papel importante. Los procesos de fabricación pueden influir en la cantidad y tipo de defectos presentes en el semiconductor, lo que a su vez afecta la generación de RTN.

- Implicaciones Prácticas:

  En aplicaciones donde el ruido de baja frecuencia como el RTN es una preocupación, los transistores tipo P pueden ser preferidos debido a su menor susceptibilidad a generar este tipo de ruido.

Sin embargo, la elección entre transistores tipo P y tipo N también depende de otros factores, como la eficiencia, la velocidad y el costo.

En resumen, los transistores tipo P tienden a generar menos señales RTN en comparación con los transistores tipo N debido a la menor movilidad de los huecos, menor densidad de estados en la banda de valencia y diferencias en las interacciones electrón-defecto. Esta característica los hace más adecuados para ciertas aplicaciones donde el ruido de baja frecuencia es una preocupación.

4. Redes neuronales y aprendizaje automático

4.1. ¿Qué es una red neuronal?

Una red neuronal es un modelo de computación avanzado que simula la forma en que el cerebro humano procesa la información. Se utiliza en el campo de la inteligencia artificial (IA) para imitar la capacidad del cerebro humano de reconocer patrones y tomar decisiones basadas en datos.

Las redes neuronales consisten en unidades de procesamiento, llamadas neuronas, organizadas en capas. Estas neuronas imitan las neuronas biológicas del cerebro humano, procesando y transmitiendo señales a través de la red.

Cada neurona recibe señales de entrada, las procesa utilizando una función de activación, y luego envía una señal de salida a otras neuronas.

Tipos de Redes Neuronales

Las redes neuronales pueden clasificarse en cuatro grupos principales:

- Redes Feedforward: Son las más simples, donde la información se mueve en una sola dirección, hacia adelante, desde las capas de entrada, a través de las capas ocultas, hasta la capa de salida.
- Redes Neuronales Convolucionales (CNN): Especializadas en procesar datos con una topología en forma de cuadrícula, como imágenes.
- Redes Neuronales Recurrentes (RNN): Tienen conexiones que forman ciclos, permitiendo que la información persista, lo que las hace adecuadas para tareas como el reconocimiento de voz o el análisis de series temporales.
- Redes Neuronales Profundas (DNN): Son redes con múltiples capas ocultas, lo que les permite modelar relaciones complejas.

Dentro de cada una de ellas hay subgrupos que se clasifican en base a su arquitectura, configuración, el tipo de problema que resuelven o las técnicas de aprendizaje que utilizan. Estos subgrupos permiten adaptar las redes neuronales a tareas específicas, aprovechando las características únicas de cada tipo.

Un ejemplo conocido son las ResNet (Redes de residuos) que es un subgrupo dentro de las CNN. Esta red introdujo el concepto de "conexiones residuales" que permite a las señales saltarse ciertas capas de la red, aumentando su velocidad de respuesta y permitiendo construir redes neuronales más profundas sin perder eficiencia en el entrenamiento, lo que resulta en un mejor rendimiento en tareas complejas de visión por computador. Es altamente usada en tareas de clasificación y detección de imágenes.

Aplicaciones Comunes

- Reconocimiento de Imágenes y Visión Computarizada: Las CNN son ampliamente utilizadas para etiquetar objetos en imágenes y videos.
- Procesamiento del Lenguaje Natural (PLN): Las RNN y las DNN permiten a las máquinas comprender, interpretar y responder a texto y voz humana.
- Predicción y Análisis de Datos: En finanzas, meteorología, y salud, las redes neuronales analizan grandes conjuntos de datos para predecir tendencias futuras.

Diferencia entre Red Neuronal e Inteligencia Artificial

Las redes neuronales son modelos o técnicas dentro del campo más amplio de la IA. Se enfoca en imitar la forma en que los humanos procesan la información a través de redes de neuronas.

La inteligencia artificial es un término más amplio que abarca cualquier técnica que permite a las máquinas imitar la inteligencia humana, incluyendo el aprendizaje, el razonamiento y la autocorrección. Las redes neuronales son solo una de las muchas herramientas utilizadas en IA.

Ahora entendemos que redes neuronales son componentes fundamentales en el desarrollo de sistemas de inteligencia artificial avanzados. Permiten a las máquinas procesar datos de manera similar a como lo hacen los humanos, aprendiendo y adaptándose a partir de la información que reciben. Sin embargo, representan solo una parte del vasto campo de la IA, que incluye muchas otras técnicas y metodologías.

Veamos entonces las dos redes aplicadas durante el estudio antes de ver sus diseños, entrenamientos y resultados.

4.2. Red neuronal Feedforward

Las redes neuronales Feedforward (Feedforward Neural Networks, FNNs) o de Alimentación hacia Adelante, representan una de las formas más básicas y esenciales en el campo de las redes neuronales artificiales. En estas redes, la información se mueve en una sola dirección, de la entrada a la salida, a través de una o más capas ocultas, sin retroalimentación o conexiones de retorno. Esta estructura lineal simplifica tanto el diseño como el análisis de las redes. Veamos por partes que hay que tener en cuenta:

Estructura

1. **Capa de Entrada**: Donde cada neurona representa una variable de entrada. No hay procesamiento en esta capa; simplemente pasa las entradas a la siguiente capa.

2. **Capas Ocultas**: Estas capas son el núcleo computacional de la red. Cada neurona en estas capas recibe entradas de todas las neuronas de la capa anterior, suma de manera ponderada (usando un conjunto de pesos y sesgos), y luego aplica una función de activación para generar una salida no lineal. Veamos más en detalle estos parámetros:

    a. W(Pesos): Son coeficientes que se aplican a las entradas de la red. En una red feedforward, cada neurona en una capa determinada recibe entradas de todas las neuronas en la capa anterior, y cada una de estas conexiones tiene un peso asociado. El efecto de una entrada en la activación de una neurona se determina multiplicando el valor de la entrada por el peso correspondiente. Los pesos son ajustados durante el proceso de entrenamiento para minimizar la función de pérdida, lo que permite a la red aprender patrones de los datos de entrenamiento.

    b. b(Sesgos): Es un parámetro adicional en las redes neuronales que se suma a la suma ponderada de las entradas antes de pasar por la función de activación. Los sesgos permiten que las neuronas se activen (o no) con mayor facilidad. Por ejemplo, incluso si todas las entradas son cero, el valor del sesgo aún podría activar la neurona si es suficientemente grande. Los sesgos también se ajustan durante el entrenamiento y

ayudan a la red a modelar patrones que de otro modo serían difíciles o imposibles de capturar si solo se usaran pesos.

c. Función de activación: Las funciones de activación introducen no linealidades en la red, lo que es esencial para aprender y modelar relaciones complejas. Algunas de las funciones de activación más comunes en las redes neuronales feedforward incluyen:

   - Sigmoide: Produce una salida entre 0 y 1, lo que la hace útil para problemas de clasificación binaria.
   - Tangente Hiperbólica (tanh): Similar a la función sigmoide, pero produce salidas en un rango de -1 a 1, centrado en cero.
   - Unidad Lineal Rectificada (ReLU): Proporciona una salida que es igual a la entrada si la entrada es positiva, y cero en caso contrario. Es la función de activación más utilizada debido a su simplicidad y eficiencia computacional.
   - Leaky ReLU: Una variante de ReLU que permite una pequeña pendiente para valores negativos, evitando así el problema de las neuronas "muertas" que pueden ocurrir con ReLU.
   - Exponential Linear Unit (ELU): Similar a ReLU, pero suaviza la aproximación para valores negativos.
   - Softmax: Especialmente utilizada en la capa de salida de redes para clasificación multiclase, convierte las salidas en una distribución de probabilidad.

   Cada una de estas funciones tiene propiedades y usos específicos, y la elección de la función de activación puede depender de la naturaleza del problema y del tipo de datos que se están modelando. La combinación de pesos, sesgos y funciones de activación permite que las FNN realicen tareas complejas de modelado y predicción en una amplia variedad de campos, desde reconocimiento de imágenes hasta procesamiento del lenguaje natural

3. **Capa de Salida**: La salida de la última capa oculta es transformada en la capa de salida, que está diseñada según la tarea específica (por ejemplo, clasificación o regresión).

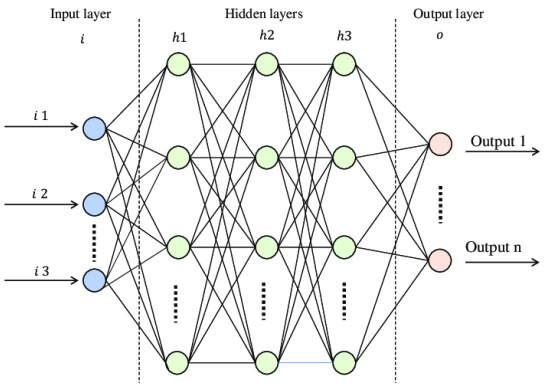

Ilustración 4.1 Diagrama de una FNN

Algoritmo de entrenamiento y retropropagación

El entrenamiento de una FNN implica ajustar sus pesos y sesgos para minimizar el error en sus predicciones. Esto se realiza a través de algoritmos de entrenamiento específicos, siendo el más común el descenso del gradiente en combinación con la retropropagación. Aquí explicaré este proceso y otros algoritmos relevantes:

- Descenso del Gradiente:
  - **Concepto**: Es un algoritmo de optimización que busca minimizar una función de coste o pérdida, que mide el error entre las predicciones de la red y los valores reales.
  - **Proceso**: Ajusta los pesos en la dirección opuesta al gradiente de la función de coste con respecto a los pesos, lo que disminuye gradualmente el error.
  - **Tasa de Aprendizaje**: Un parámetro crucial que determina el tamaño de los pasos en la actualización de los pesos. Una tasa demasiado alta puede sobrepasar el mínimo, mientras que una tasa demasiado baja puede hacer que el entrenamiento sea muy lento.

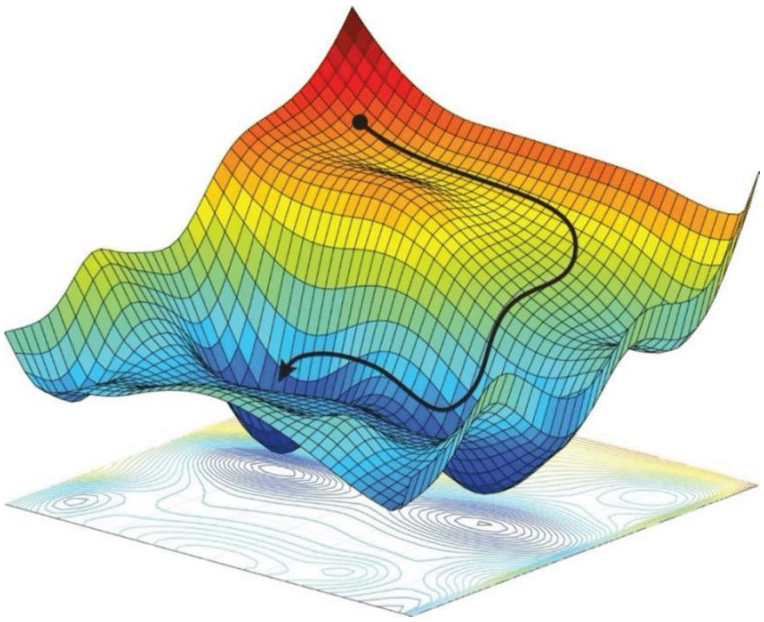

Ilustración 4.2Representación de un descenso del gradiente

- Retropropagación:
  - **Función:** Es el método utilizado para calcular el gradiente de la función de coste. La retropropagación lleva este gradiente a través de la red, desde la salida hacia la entrada, actualizando los pesos y sesgos en cada capa.
  - **Cálculo de Gradientes:** Utiliza la regla de la cadena del cálculo diferencial para calcular los gradientes de la función de coste con respecto a cada peso y sesgo en la red.

- Variantes del Descenso del Gradiente:

- **Gradiente Descendente Estocástico (SGD):** En lugar de utilizar todo el conjunto de datos para calcular el gradiente de la función de coste, el SGD utiliza un solo ejemplo o un pequeño lote (batch) en cada iteración, lo que hace que el entrenamiento sea más rápido y menos propenso a quedarse atascado en mínimos locales. Previamente hemos hablado del ResNet, que es un tipo de SGD.

- **Momentum**: Añade una fracción del gradiente de la actualización anterior a la actualización actual, lo que ayuda a acelerar el SGD en la dirección correcta y amortiguar las oscilaciones.

- **Adagrad, RMSprop, Adam:** Son algoritmos más avanzados que ajustan la tasa de aprendizaje durante el entrenamiento para cada peso, lo que mejora la convergencia.

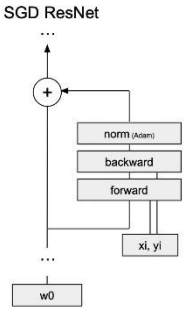

Ilustración 4.3 Diagrama de ResNet

- Regularización:
  Para evitar el sobreajuste, donde la red aprende los datos de entrenamiento demasiado bien y no generaliza correctamente a nuevos datos, se utilizan técnicas de regularización como el Dropout o la Regularización L2. Las veremos más adelante.

- Evaluación y Ajuste:
  Durante y después del entrenamiento, se evalúa el rendimiento de la red utilizando un conjunto de datos de validación, y se ajustan los hiperparámetros como la tasa de aprendizaje, el tamaño del lote y la arquitectura de la red para optimizar su rendimiento.

El entrenamiento de una FNN es un proceso iterativo y, a menudo, experimental. La elección del algoritmo de entrenamiento y la configuración de los hiperparámetros dependerán en gran medida del problema específico y de la naturaleza de los datos disponibles.

Ventajas y Limitaciones

- Ventajas: Las FNNs son intuitivas, relativamente fáciles de programar y eficaces para una amplia gama de problemas lineales y no lineales.

- Limitaciones: No son óptimas para tareas que requieren memoria o contextualización de los datos, como el procesamiento del lenguaje natural o las series temporales, donde otras arquitecturas como las RNN (Redes Neuronales Recurrentes) son más adecuadas.

Aplicaciones

Las FNNs son utilizadas en una variedad de aplicaciones, desde la predicción de tendencias del mercado hasta el diagnóstico médico y la detección de fraudes.

Aunque las FNN son relativamente simples en comparación con arquitecturas más avanzadas, siguen siendo una herramienta poderosa y fundamental en el aprendizaje automático y la inteligencia artificial, proporcionando una base sólida sobre la cual se construyen modelos más complejos.

4.3. RNN – Long Short-Term Memory (LSTM)

Las Redes Neuronales Recurrentes (RNN) con Long Short-Term Memory (LSTM) son una variante avanzada de las RNN tradicionales, diseñadas para capturar dependencias a largo plazo en secuencias de datos. Veamos sus aspectos clave:

Estructura

Las redes LSTM son un tipo especial de RNNs diseñadas para recordar información durante largos períodos de tiempo. Son particularmente útiles para secuencias de datos donde es importante mantener información de estados anteriores, como en el procesamiento del lenguaje natural o en series temporales. Al igual que en las FNNs, las LSTMs utilizan pesos, sesgos y funciones de activación, pero su estructura es más compleja debido a la recurrencia y a los mecanismos de puertas que controlan el flujo de información.

En las LSTMs, hay tres tipos de puertas: la puerta de olvido, la puerta de entrada y la puerta de salida.

- **Puerta de olvido:** Decide qué información se descarta del estado de la celda.

- **Puerta de entrada:** Decide qué nueva información se añade al estado de la celda.

- **Actualización del estado de la celda:** Se combina la información antigua (modulada por la puerta de olvido) y la nueva información candidata (modulada por la puerta de entrada) para actualizar el estado de la celda.

- **Puerta de salida:** Decide qué parte del estado de la celda se pasa al siguiente paso de tiempo o a la siguiente capa.

Cada una de estas puertas tiene su propio conjunto de pesos y sesgos.

- **Pesos de las puertas:** Son matrices que determinan la importancia de las entradas y los estados anteriores para el estado actual de la puerta. Los pesos en una LSTM se dividen en dos grupos: los que se aplican a las entradas (pesos de entrada) y los que se aplican a los estados ocultos anteriores (pesos recurrentes).

- **Sesgos de las puertas:** Cada puerta tiene su propio vector de sesgos, que se suma al producto de los pesos y las entradas para ayudar a decidir cuánto de la señal pasará a través de la puerta.

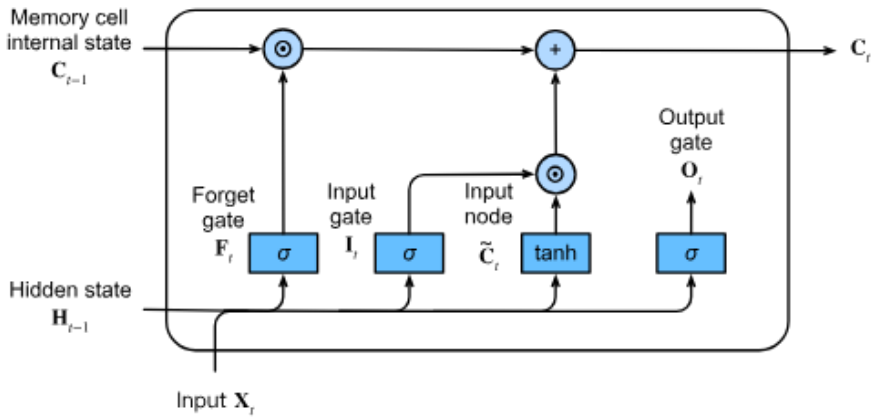

Ilustración 4.4 Diagrama de una red neuronal RNN-LSTM

Las LSTMs utilizan varias funciones de activación, las dos más comunes pueden apreciarse en el diagrama:

Sigmoide: Utilizada en las puertas de olvido, entrada y salida, esta función decide cuánta información se debe dejar pasar a través de cada puerta. La función sigmoide es ideal aquí porque su salida está entre 0 y 1, lo que puede interpretarse como una probabilidad de "cuánto" se debe permitir que pase la información.

Tangente Hiperbólica (*tanh*): Utilizada para crear un nuevo vector de candidatos que podría agregarse al estado de la celda, y también se aplica al estado de la celda antes de multiplicarlo por la activación de la puerta de salida para obtener el nuevo estado oculto. La función *tanh* es útil porque su salida varía de -1 a 1, lo que ayuda a regular la naturaleza de los valores en el estado de la celda.

En cada paso de tiempo, la LSTM realiza estas operaciones, permitiendo que la información fluya a través de la red de una manera controlada, con la capacidad de mantener y descartar información a lo largo del tiempo. Esto es lo que permite a las LSTMs abordar problemas de dependencias a largo plazo, donde las redes RNN estándar suelen tener dificultades.

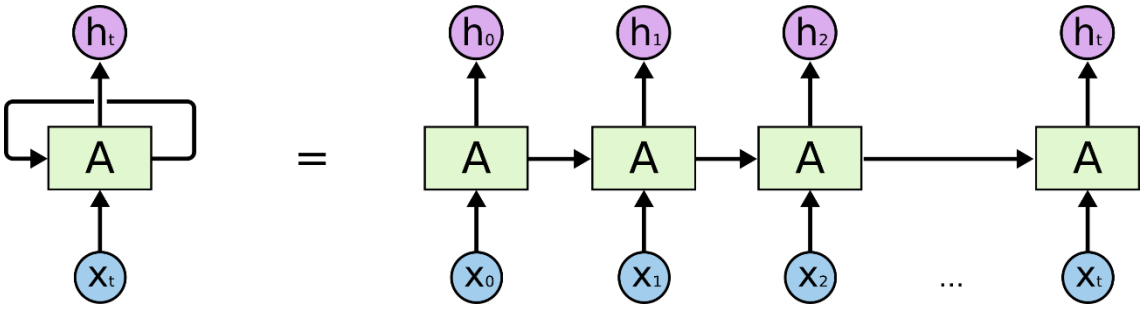

Ilustración 4.5 Diagrama de RNN estándar

El diagrama representa una RNN estándar, donde tiene capacidad de pasar la información entre etapas, pero sus debilidades se destacan a primera vista.

La LSTM tiene varias puertas que permiten regular el flujo de información, aquí la información siempre es completa, no existen filtros, produciendo que la señal de salida de una etapa sea la de entrada de la siguiente, de forma que la información no se gestiona igual, dando como resultados entrenamientos mas largos. Otra diferencia importante es la facilidad de las RNN para tener problemas de desvanecimiento del gradiente ya que la información de etapas anteriores se pierde rápidamente.

Algoritmo de entrenamiento y retropropagación

Al igual que en las FNNs, se utiliza el descenso del gradiente y la retropropagación para entrenar las redes LSTM. Sin embargo, la retropropagación en LSTM es más compleja debido a las conexiones recurrentes. Se hace uso de la BPTT (Retropropagacion a través del tiempo), una variante que implica desplegar la red a través del tiempo y luego retropropagar el error desde el final hasta el principio de la secuencia.

El entrenamiento implica ajustar los pesos y sesgos, pero debido a su arquitectura especializada para manejar dependencias temporales, el proceso incluye pasos adicionales y consideraciones. Veamos cómo se adaptaría el proceso para una LSTM:

- Descenso del Gradiente:
  - **Concepto:** Este algoritmo se mantiene igual que en las FNN. Busca minimizar una función de coste, que en el caso de las secuencias podría ser la entropía cruzada en tareas de clasificación o el error cuadrático medio en tareas de regresión.
  - **Proceso:** Para las LSTMs, esto incluye no solo las conexiones estándar sino también las puertas y los estados de la celda.
  - **Tasa de Aprendizaje:** Similar a las FNN, la tasa de aprendizaje es un parámetro crítico. Sin embargo, puede requerir más ajuste debido a la complejidad adicional de las secuencias temporales.

- Retropropagación:
  BPTT (Backpropagation Through Time):
  - **Función:** BPTT es una adaptación de la retropropagación para redes recurrentes. En lugar de propagar hacia atrás solo a través de las capas, BPTT también propaga el error a través del tiempo.

- **Cálculo de Gradientes:** Utiliza la regla de la cadena para calcular los gradientes, pero teniendo en cuenta las conexiones temporales. Esto significa que el gradiente de la función de coste se calcula en cada paso de tiempo y se acumula a través de todos los pasos de tiempo.

Existen variaciones, pero la más común y popular es la:

- **Truncated BPTT:** Una variante de BPTT donde el gradiente se propaga hacia atrás solo por un número limitado de pasos de tiempo a configurar por el usuario. Esto es útil para secuencias muy largas donde el BPTT completo sería computacionalmente costoso.

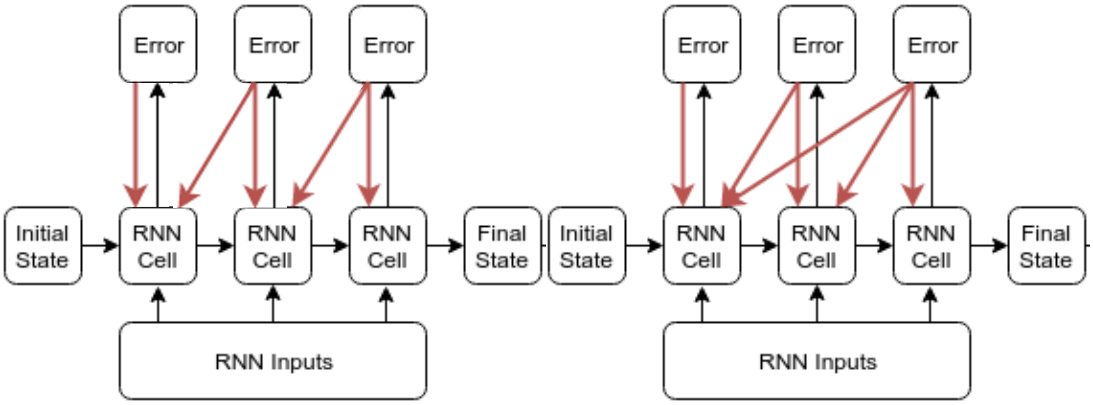

Ilustración 4.6 Diagrama de funcionamiento de BPTT y Truncated BPTT

En este diagrama podemos apreciar la Trucated BPTT a la izquierda, con su numero limitado de pasos de tiempo, mientras que a la derecha apreciamos una BPTT completa sin limitación.

Aquí podemos apreciar la diferencia en una línea de tiempo:

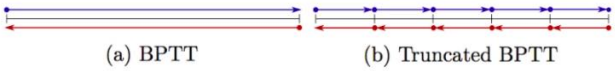

Ilustración 4.7 Comparativa temporal entre BPTT y Truncated BPTT

- Variantes del Descenso del Gradiente:
  - **Gradiente Descendente Estocástico (SGD)**: La variante estocástica puede ser particularmente útil cuando se manejan grandes conjuntos de datos secuenciales.
  - **Momentum**: Ayuda a las LSTMs a superar los puntos planos en superficies de error complejas que son comunes en las secuencias temporales.
  - **Adagrad, RMSprop, Adam:** Estos métodos son beneficiosos ya que adaptan la tasa de aprendizaje de cada parámetro a través del tiempo, lo cual es útil dado que diferentes partes de la secuencia pueden requerir diferentes ajustes en los parámetros.

- Regularización:
  Técnicas como *Dropout* adaptado para LSTMs (donde el Dropout se aplica solo a las conexiones no recurrentes) o la Regularización L1/L2 pueden ser necesarias para evitar el sobreajuste en datos secuenciales complejos. Veamos en qué consisten:

  - Dropout: Es una técnica de regularización que implica "desactivar" aleatoriamente un subconjunto de neuronas durante cada iteración del entrenamiento. Esto significa que, en cada paso del entrenamiento, cada neurona tiene una probabilidad p de ser ignorada, lo que evita que participe en la propagación hacia adelante y en la retropropagación. Al hacer esto, Dropout previene que las neuronas se adapten demasiado entre sí, forzando a la red a aprender representaciones más robustas que son independientes de las contribuciones particulares de cualquier subconjunto de neuronas. Durante la inferencia o evaluación del modelo, todas las neuronas se utilizan, pero sus salidas se escalan por p para compensar el hecho de que más neuronas están activas que durante el entrenamiento.

  - Regulación L1: También conocida como Lasso (Least Absolute Shrinkage and Selection Operator), penaliza la suma del valor absoluto de los pesos del modelo. Matemáticamente, se añade un término de penalización al coste que es proporcional a la suma de los valores absolutos de los pesos:

    $$Coste\ Total = Coste\ de\ Pérdida + \lambda \sum |w|$$

    donde w son los pesos del modelo y λ es un hiperparámetro que controla la fuerza de la penalización. La regularización L1 tiene la propiedad interesante de producir soluciones dispersas, lo que significa que puede hacer que algunos pesos sean exactamente cero. Esto puede ser útil para la selección de características o para crear modelos más simples y eficientes.

  - Regulación L2: También conocida como Ridge, penaliza la suma de los cuadrados de los pesos del modelo. El término de penalización que se añade a la función de coste es proporcional a la suma de los cuadrados de los pesos:

    $$Coste\ Total = Coste\ de\ Pérdida + \lambda \sum w^2$$

    La regularización L2 tiende a dispersar el error entre todos los términos, lo que significa que en lugar de tener pesos que son cero como en L1, los pesos no se vuelven exactamente cero, pero los valores extremos son penalizados, lo que lleva a que los pesos sean generalmente pequeños. Esto puede ayudar a mejorar la generalización del modelo al evitar que cualquier característica tenga un peso demasiado grande.

  Ambas regularizaciones, L1 y L2, pueden usarse juntas, ElasticNet es un método conocido que combina las propiedades de selección de características de L1 con las propiedades de regularización de L2.

- Evaluación y Ajuste:
  Durante y después del entrenamiento, se evalúa la capacidad de la LSTM para generalizar utilizando un conjunto de validación. Se pueden ajustar hiperparámetros como la tasa de aprendizaje, el tamaño de los lotes y la arquitectura de la red, lo que incluye el número de celdas LSTM y la profundidad de la red.

Ventajas y Limitaciones

- Ventajas: Son especialmente buenas para aprender dependencias a largo plazo y son menos propensas al problema del desvanecimiento del gradiente, común en RNN estándar.

- Limitaciones: Son computacionalmente más intensivas que las RNN tradicionales y pueden ser más difíciles de entrenar. También pueden ser propensas al sobreajuste, especialmente en conjuntos de datos pequeños.

Aplicaciones

- Procesamiento del Lenguaje Natural (PLN): Las LSTM son ideales para tareas como la traducción automática, el modelado del lenguaje y la generación de texto.
- Series Temporales: Se utilizan para predecir tendencias del mercado de valores, pronóstico del tiempo y más.
- Reconocimiento de Voz y Música: Pueden modelar secuencias acústicas para el reconocimiento de voz o generar música.

En resumen, las LSTM ofrecen una solución poderosa y flexible para modelar secuencias de datos, superando muchos de los desafíos que presentan las RNN estándar, especialmente en tareas que requieren comprender dependencias a largo plazo en los datos.

5. Desarrollo del proyecto

Con los fundamentos teóricos interiorizados, podemos pasar a la práctica, que constara de tres partes. La primera es común para las dos redes, mientras que las otras dos son completamente diferentes y nos dejara conocer y ver mejor el comportamiento de cada una de ellas. Empecemos entonces preparando los datos.

5.1. Preprocesamiento de los datos

Para los datos de las redes haremos uso de cinco paquetes de 504 señales RTN con 25.000 datos cada una. Estos paquetes se diferencian entre sí por el voltaje al que se encontraban los transistores a la hora de tomar las medidas, por lo que tenemos:

- Paquete de muestras a 0.6V
- Paquete de muestras a 0.7V
- Paquete de muestras a 0.8V
- Paquete de muestras a 1.0V
- Paquete de muestras a 1.2V

Estas señales analógicas deben ser previamente tratadas antes de poder usarse para el entrenamiento. Durante los entrenamientos, haremos uso del primer paquete debido a que, la teoría nos dice que, las señales de bajo voltaje son susceptibles a mezclarse con el ruido del circuito, provocando una detección de datos errónea. Considerando esta idea, las muestras de 0.6V contendrán tiene mayor cantidad de señales no aleatorias y por tanto la red neuronal podrá aprender antes cuales ha de descartar.

Antes de ver el proceso de tratamiento de la señal, haremos hincapié en que datos podríamos usar en caso de no disponerlos. Para esta cuestión existen diferentes respuestas, como generar señales analógicas (por ejemplo, con un micrófono) o tomarlas de un sensor de vibración (que detecte las señales de un motor). Pero, las señales aleatorias mas precisas pueden extraerse de la paradoja del gato de Schrödinger, donde, sin abrir la caja el gato esta en dos estados a la vez, vivo y muerto. Con esto nos topamos con en la mecánica cuántica y mas concretamente los *escuchadores cuánticos*, maquinas que detectan el *ruido cuántico*, la energía mínima (cercana a la mitad de la energía de un fotón) que puede ser detectada.

El método para hacer visible este ruido cuántico implica dividir un haz de láser en dos partes iguales con un divisor de haz.

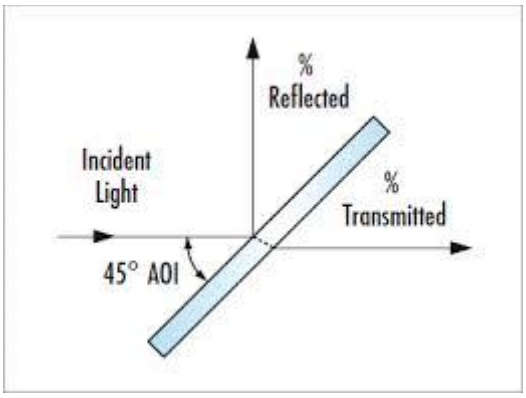

Ilustración 5.1 Representación de un divisor de haz de luz

Al bloquear uno de los puertos de entrada del divisor, las fluctuaciones del vacío afectan la salida de los dos haces parciales. Estos son luego enviados a detectores que miden la intensidad de la corriente de fotones. El resto de las mediciones de estos dos detectores deja como resultado el ruido cuántico. Este ruido, que surge al azar durante las mediciones, se utiliza para generar los números aleatorios.

En las referencias se adjunta un escuchador cuántico online en donde se puede ver, a tiempo real, qué está detectando en ese momento, por lo que, no haber tenido datos, habríamos escogido la idea de tratar señales cuánticas. Es más, la idea de usar señales cuánticas en redes neuronales de detección de señales aleatorias podría darnos información desconocida hasta el momento, porque, ¿Qué pasaría si una red neuronal perfectamente capaz de detectar señales aleatorias no pudiera detectar aleatoriedad en el ruido cuántico? Esta cuestión no ha sido tratada en este proyecto, pero podría darnos información de si realmente las partículas se comportan de forma aleatoria o mediante algún patrón ya que sabemos que las partículas se "comportan diferente" cuando son observadas.

Conversión de analógico a digital

Para transformar las señales analógicas a digitales requerimos de cuatro pasos que, como resultado, nos da una señal ligeramente aleatoria con respecto la señal de origen. Esta conversión podría haberse efectuado en un solo paso estableciendo un umbral medio entre el primer valor y el ultimo valor de cada señal, de forma que si está por encima de dicho umbral se escribiera un 1 y si no, un 0. Sin embargo, muchas de estas señales carecen de información y a veces no pueden llegar a ese umbral, dejando muchas señales fuera del entrenamiento de la red debido a que, a primera vista, ya se puede ver que son señales no aleatorias. Algunos ejemplos son por ejemplo la señal 4 o la 401:

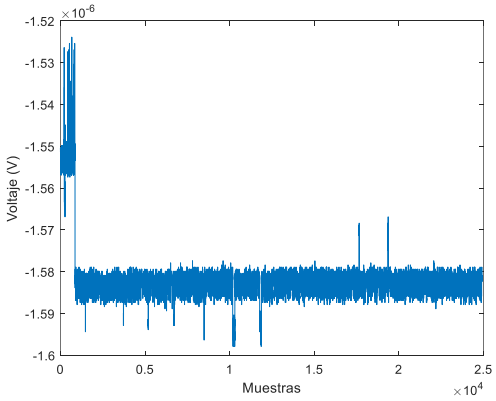

Ilustración 5.2 Señal n.4 a 0.6V

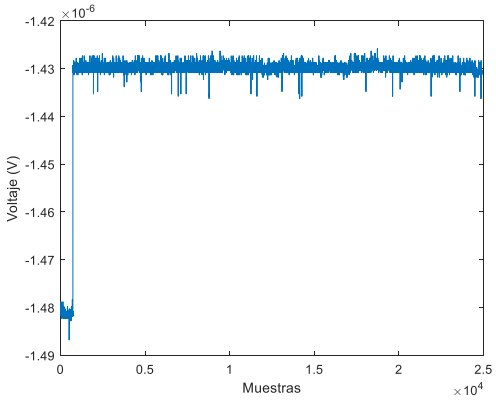

Ilustración 5.3 Señal n.401 a 0.6V

En estas dos señales carece de sentido hacer uso de un umbral medio, dado que la diferencia de los primeros valores con el resto deja una consecutiva linea de 0s seguida de una de 1s y viceversa. Podriamos eliminar el primer pico de las dos señales y establecer un umbral medio de solamente los datos que parecen estar entre dos amplitudes constantes. El problema de esto es, que al probarlo, obtenemos una consecucion de 0s y 1s, algo muy poco aleatorio debido a que repetimos constantemente el mismo patron, la probabilidad de que detrás de un 0 se escriba un 1 es muy alta, por lo que estas señales son predecibles y por tanto, no aleatorias.

Sin embargo, si tenemos señales aleatorias que con un simple umbral podrian crear señales binarias semi-aleatorias, un ejemplo de ello es la señal numero 20, que se usara para el resto de la explicacion de conversion:

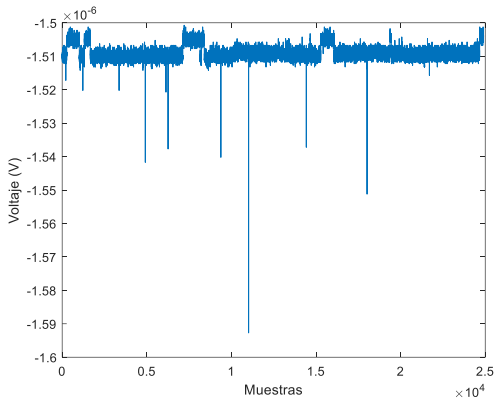

Ilustración 5.4 Señal n.20 a 0.6V

Si partimos de la base de que tanto el inicio como la salida de la señal trabajan sobre el mismo rango de amplitud y que, en determinados puntos se establecen picos inusuales, el umbral aquí no tiene por qué estar situado entre el valor mínimo y el máximo, simplemente podríamos establecerse el umbral en esta área:

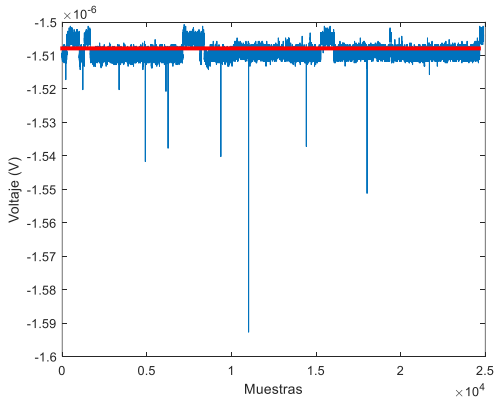

Ilustración 5.5 Señal n20 a 0.6V con un umbral medio

La línea roja podría ser el umbral, dictaminando que toda señal por debajo del umbral se escribiera como un 0, mientras que las que estén por encima se escribieran como un 1.

Esta manera tan básica de conversión de datos no es convincente, como hemos visto con las señales 4 y 401 esas no hay manera alguna de que puedan llegar a generar una señal binaria aleatoria puesto que padecen de pocas irregularidades.

Por lo que se hizo uso de un algoritmo usado en economía que hoy en día se aplica a través de redes neuronales conocido como *Media Móvil*.

Este algoritmo permite ver a tiempo real en el mercado, independientemente de la temporalidad que el sujeto este observando, la tendencia del precio a través de un simple algoritmo que calcula el precio medio de la cantidad de cotizaciones previas que decida el usuario. Si decide hacer una media móvil de 50 periodos implica que, sobre el grafico del activo se dibuja una gráfica que

constantemente calcula las 50 cotizaciones previas del precio, permitiendo así a los inversores saber la tendencia del mercado.

Este concepto viene genial para poder convertir señales analógicas prácticamente planas en señales digitales *semi-aleatorias*. Veamos los cuatro pasos a seguir:

1- Preparación de los datos a convertir

   Este primer paso ya lo hemos visto, consiste en cargar la señal que queremos convertir, en este caso la numero 20.

2- Cálculo de una media móvil aleatoria mediante la función rand().

   En este paso obtendremos dos señales diferentes. La primera, y más importante consiste en hacer uso de la media móvil aleatoria mediante una función rand(). La idea de hacer uso de una media móvil es calcular un umbral en base a las variaciones que hay cada *x* valores al azar entre el 1 al 20. Veámoslo con más detalle:

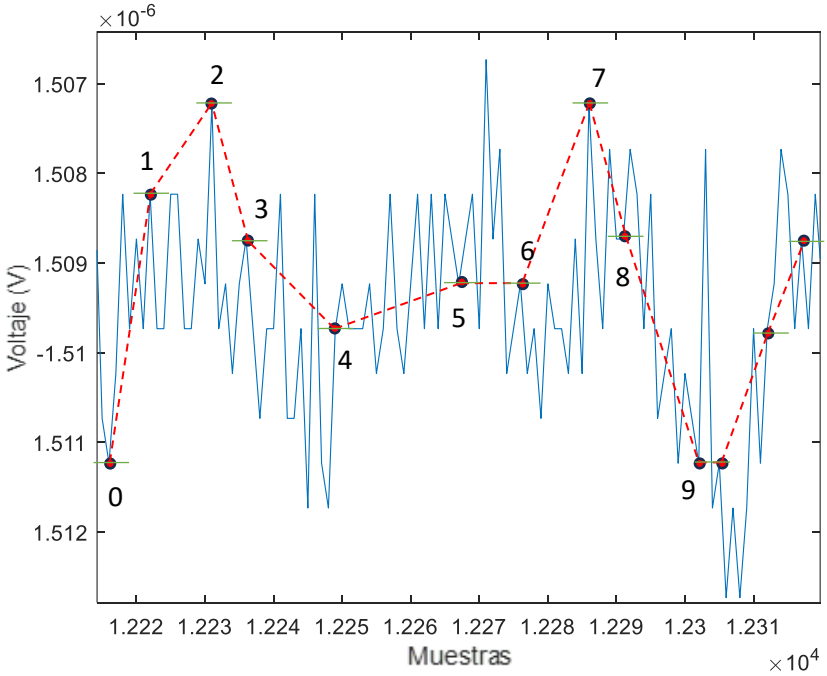

Ilustración 5.6 Ejemplo de cálculo de una media móvil aleatoria en la señal n.20 a 0.6V

Podemos ver cómo funciona el cálculo de la variación. Si nos fijamos, el primer punto se ha tomado como valor 0 y en los siguientes 20 datos deberá parar y decir la variación en voltaje que hay hasta ahí. En este caso ha parado en el quinto dato, pero, como hacemos uso de la función rand, cada vez que ejecutamos el código obtendremos variaciones diferentes y por tanto señales digitales diferentes. Ahora, este segundo punto pasa a valer 0 y desde este punto al siguiente pasan 9 datos, viendo que ahora hay un incremento mucho más ligero que el anterior. Los siguientes dos tienen un decrecimiento, el punto 5 y 6 valen 0, al igual que el 9 y 10, etc.

De esta manera obtenemos la segunda señal necesaria en este paso:

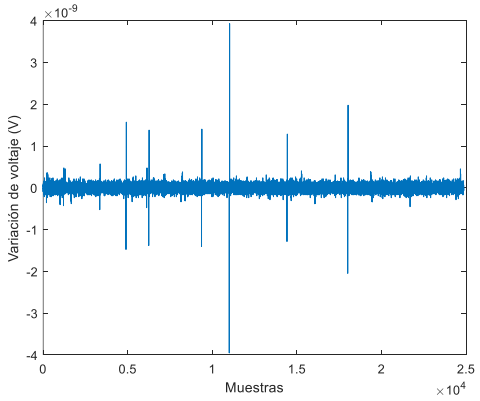

Ilustración 5.7 Variación calculada entre datos de la señal n.20 a través de la Media Móvil

Esta señal representa todas las variaciones que hemos calculado, pero tiene un problema, trabaja tanto para valores negativos como positivos, por lo que debemos pasar toda la señal a valores positivos elevándola al cuadrado:

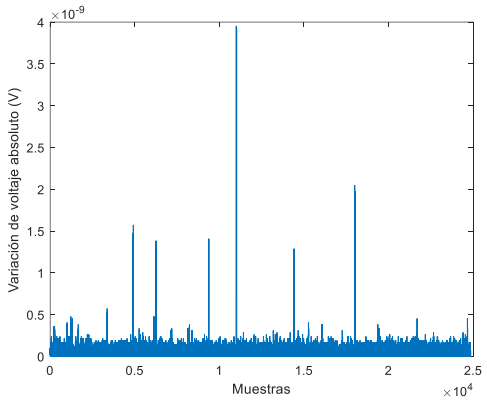

Ilustración 5.8 Variación cuadrática de la señal n.20

Antes de pasar al siguiente paso, veamos cómo se ven las señales 4 y 401 después de este procedimiento:

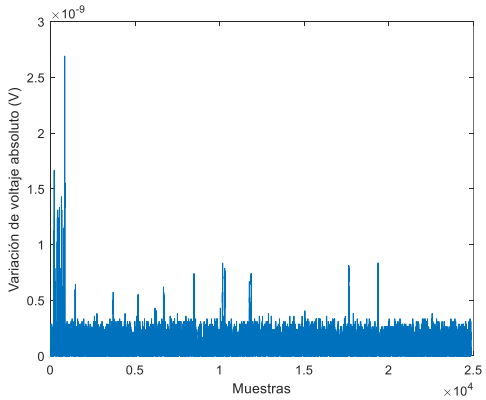

Ilustración 5.9 Variacion cuadrática de la señal n.4

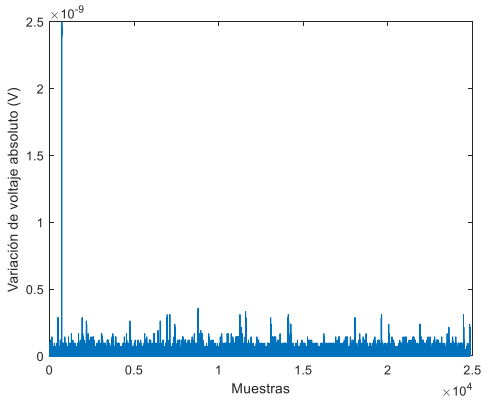

Ilustración 5.9 Variacion cuadrática de la señal n.401

3- Calculo binario de las variaciones

Gracias al código anterior podemos establecer para cada una de las señales una señal basada en la variación de los datos de forma aleatoria. Ahora si podemos hacer uso de un umbral, pero es difícil encontrar un umbral que sirva para todas las señales. Si nos fijamos en las tres previas podemos establecer como umbral el 0.5 nV pero no sería justo. Las señales 20 y 401 tendrán una cantidad de 0s muy superior a la señal 4, por lo que dejarían inmediatamente de ser aleatorias. Es por eso por lo que el umbral no determinara el valor binario simplemente por estar por encima o por debajo.
El valor binario de la señal se determinará en base a si la señal ha cruzado dos veces por el umbral que será de 0.3 nV. Veamos un ejemplo:

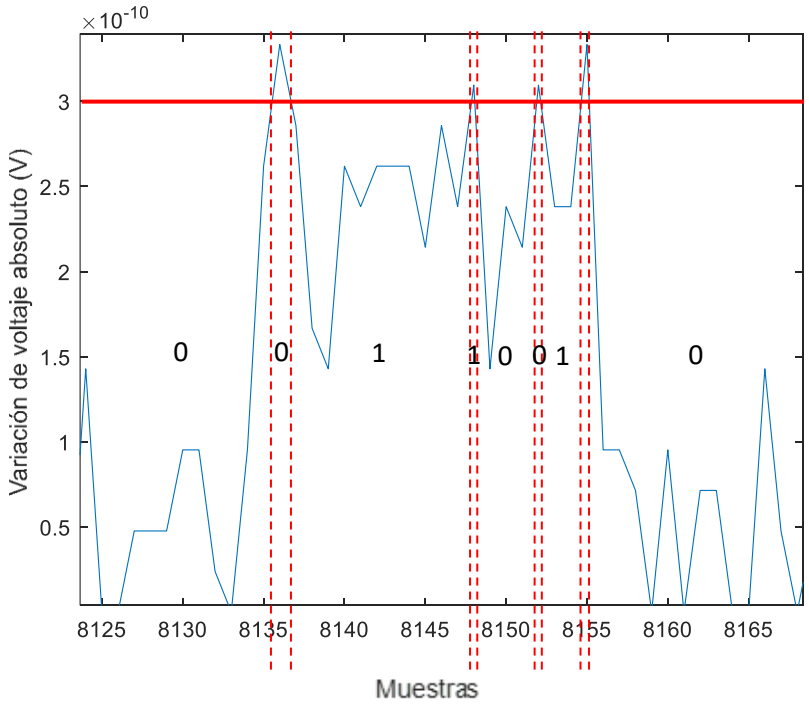

Ilustración 5.10 Fragmento de la señal n.20 con la secuencia binaria basada en dos transiciones

En este tramo de la señal suponemos que los valores hasta el primer cruce son 0s. Podemos ver cómo, al cruzar una primera vez, el valor se mantiene, pero al cruzar una segunda pasamos a tener 1s hasta que cambiamos de nuevo.

Para entenderlo mejor, este código quedaría tal que:

0000000000000 1 111111111111 0 00000 1 1111 0 00000000000000

Los números espaciados representa la posición del dato de la imagen en donde se produce una transición. En este caso la señal, a primera vista, puede no parecer aleatoria, pero hay que considerar que se ha escogido como ejemplo un área de la señal que con pocas transiciones para que fuera de fácil entendimiento y que la señal se compone de 25.000 datos, aquí puede apreciarse aproximadamente un 0.1% de información de la señal.

Al unir todas estas transiciones obtenemos de la señal 20 la siguiente señal digital:

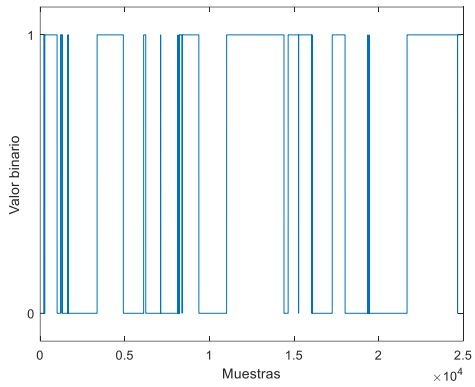

Ilustración 5.11 Señal digital basada en la señal n.20 a 0.6V

Las siguientes dos señales binarias corresponden a las señales 4 y 401, dejando ver, cómo, gracias a este método, cualquier señal analógica, por muy información que contenga, pueden convertirse en señales binarias semi-aleatorias:

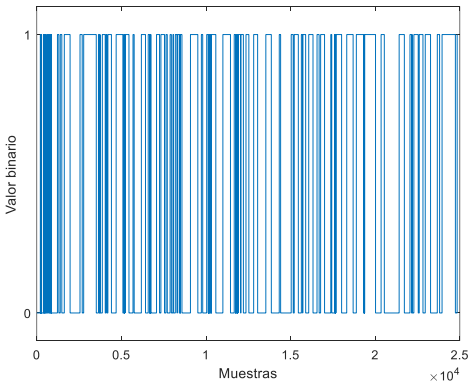

Ilustración 5.Señal digital de la señal n.4

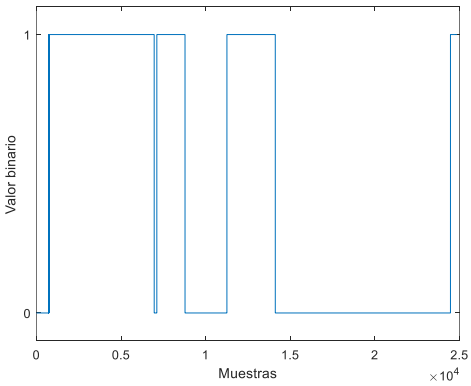

Ilustración 5.12 Señal digital de la señal n.401

Este método permite generar infinitas señales partiendo de una sola señal gracias al uso de la función rand() que afecta a la amplitud de las variaciones y por tanto afecta al umbral. Además, es curioso ver como señales que en un origen parecen aleatorias no lo son, y como otras que parecen planas, acaban teniendo más aleatoriedad de lo que podemos imaginar, como puede pasar con la señal número 4.

5.2. Red neuronal Feedforward

Durante todo el proceso de la conversión de datos se han estado guardando las señales binarias en una matriz para usarse ahora como datos de entrenamiento.

Esta matriz de nuevo consta de 504 señales con sus 25.000 datos, si hacemos un cálculo rápido nos damos cuenta de que la red debe ser capaz de procesar un poco más de 12 millones de datos, algo que, si bien a más datos mejor rendimiento, es poco práctico, ya que deja como resultado un largo entrenamiento.

Había que encontrar la manera de optimizar la red, reduciendo los métodos de entrenamiento y sacrificando un poco de rendimiento.

5.2.1. Diseño y entrenamiento

La primera red neuronal para diseñar, como hemos comentado anteriormente, es la más básica de las redes neuronales existentes, consta de un entrenamiento lineal, sin hacer uso de datos pasados, siendo rápida de ejecutar y práctica.

Esta red solo debe cumplir una orden, calificar en aleatoria aquellas señales cuyo porcentaje de 1s estuviera dentro del rango de 45%-55%.

Previamente a entrenar la red programamos un código donde, bajo esta condición, los paquetes de datos tenían:

Señal RTN [V]	**Señales aleatorias**	**Señales no aleatorias**
0.6	199	305
0.7	262	242
0.8	315	189
1.0	406	98
1.2	461	43

Tabla 5.1 Tabla de señales aleatorias y no aleatorias mediante algoritmo

Aquí podemos observar la primera suposición que tuvimos sobre las señales, donde, a más voltaje mayor cantidad de señales aleatorias, debido a que el ruido no afecta tanto la señal y las variaciones calculadas con la Media móvil son más grandes dado que son valores reales, permitiendo que estas crucen más veces el umbral, haciendo mayores transiciones y, por tanto, mayor combinación de dígitos binarios

Estos datos nos servirán como referencia para saber si la red está trabajando como nosotros buscamos.

Al finalizar el entrenamiento nos aparece el resumen.

Antes de siquiera probar la red con nuevos datos, ya podemos saber que el entrenamiento ha sido un éxito.

Si nos fijamos en la columna *Stopped Value* veremos como el entrenamiento ha parado en la decimosexta época de las 1000 que habían asignadas. Un poco más abajo vemos el rendimiento, del orden de 10^{-12}. También podemos ver el gradiente (marcado en vede).

Solamente viendo estos datos sabemos que el entrenamiento ha salido como buscábamos, pero vamos a ver un poco más en detalle la información que podemos obtener de aquí.

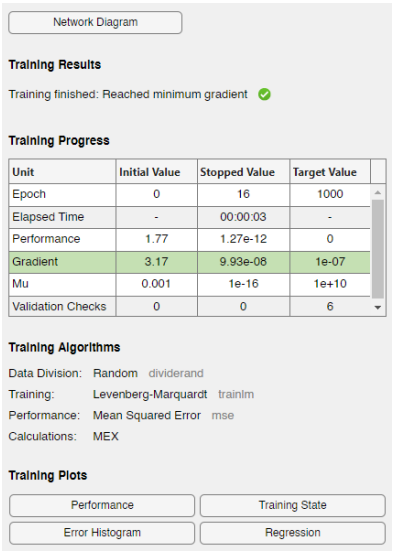

Ilustración 5.13 Resumen de entrenamiento

Empezando por el diagrama. Podemos ver que la configuración usada contiene una entrada, diez capas ocultas y una capa de salida. Veamos que sucede en cada capa:

Input: El dato de entrada de la red es un array que contiene las probabilidades de las 504 señales. Este array se ha obtenido mediante el código que hemos hablado anteriormente.

Hidden Layers: La capa oculta es la que se encarga de procesar los datos de entrada a través de:

> **W(Pesos):** Matriz de pesos que conectan la entrada con las neuronas de la capa oculta. Se ajustan automáticamente durante el entrenamiento para capturar los patrones en los datos.
>
> **b(Sesgo):** Vector de sesgo para la capa oculta.
>
> **Función de activación:** Hacemos uso de una función no lineal que permite a la red aprender relaciones complejas. Las neuronas en la capa oculta sumarían las entradas ponderadas y les aplicarían la función de activación.

Output Layer: La capa de salida procesaría la salida de la capa oculta para producir una decisión binaria: aleatoria o no aleatoria.

> **W(Pesos)** y **b(Sesgo)** Serían específicos para la capa de salida, ajustando la salida de la capa oculta para llegar a la decisión final.
>
> **Función de activación:** Hacemos uso de una función de activación sigmoidea que convierta el resultado en una probabilidad entre 0 y 1.

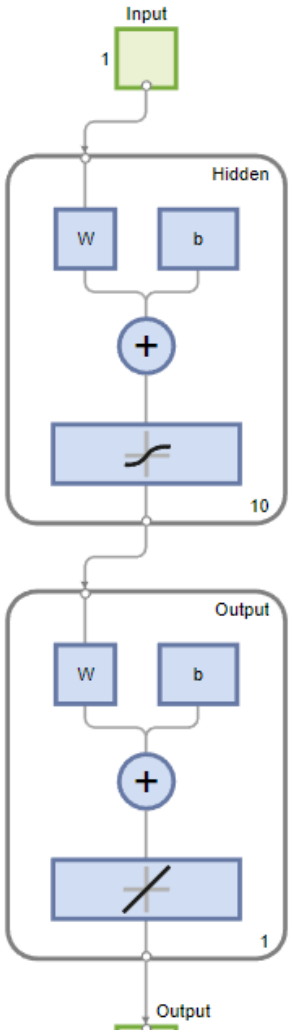

Ilustración 5.14 Diagrama FNN

La red neuronal aprendería durante el entrenamiento a distinguir las secuencias en las que el porcentaje de 1s cae dentro del rango deseado (45% - 55%) de aquellas que no lo hacen. Sería entrenada con ejemplos de ambos tipos de secuencias y ajustaría sus pesos y sesgos para minimizar algún tipo de función de pérdida que mide el error en sus predicciones.

Veamos ahora el rendimiento:

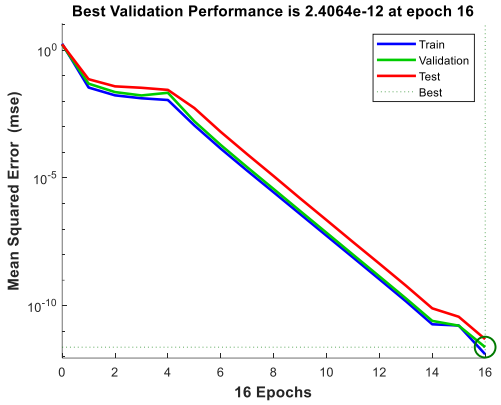

Ilustración 5.15 Rendimiento del entrenamiento

El rendimiento se calcula mediante el error cuadrático medio (Mean Squared Error, MSE) contra el número de épocas durante el entrenamiento. El gráfico muestra cuatro líneas diferentes, cada una representando un conjunto de datos diferente:

- **Train (Azul):** Muestra el MSE para el conjunto de entrenamiento. Este error se calcula durante la fase de entrenamiento y se utiliza para ajustar los pesos de la red.
- **Validation (Verde):** Representa el MSE para el conjunto de validación. Este conjunto no se utiliza para el entrenamiento directo, sino para ajustar los hiperparámetros y evitar el sobreajuste. El hecho de que esta línea siga de cerca la línea de entrenamiento sin divergir indica que el modelo está generalizando bien y no está sobreajustado.
- **Test (Rojo):** Muestra el MSE para el conjunto de pruebas. Este conjunto se utiliza para evaluar el modelo después de que el entrenamiento ha finalizado y proporciona una evaluación de cómo se desempeñará el modelo en datos no vistos.
- **Best (Punto negro):** Este punto podría representar el mejor rendimiento de la red neuronal en algún conjunto de datos durante el entrenamiento. Suele ser el punto donde se consiguió el menor error de validación antes de que comenzara a aumentar nuevamente (un indicio de sobreajuste), de ahí la razón de porque el entrenamiento paró en la decimosexta época, pudo anticiparse a obtener peores rendimientos.

Todas las líneas muestran una disminución constante en el MSE a medida que avanzan las épocas, lo que sugiere que la red está aprendiendo de manera efectiva. No hay una divergencia significativa entre el entrenamiento y la validación, lo que es un buen signo de que el modelo no está memorizando los datos, sino que está generalizando para hacer predicciones sobre datos no vistos.

El gráfico utiliza una escala logarítmica en el eje Y, lo que es común cuando se quieren mostrar cambios en órdenes de magnitud en los errores y se quiere enfocar en las diferencias de errores

cuando son muy pequeños. La escala logarítmica también ayuda a visualizar mejor el error cuando hay cambios drásticos en su magnitud.

Veamos el *Trainig State:*

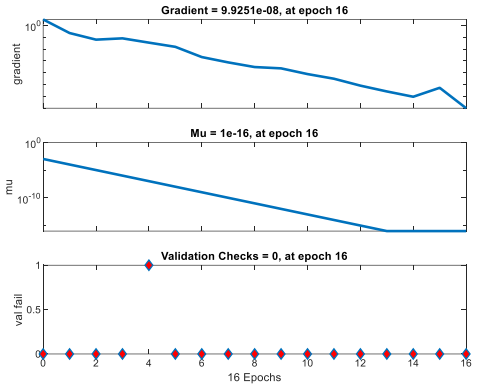

Ilustración 5.16 Graficas de Gradiente, error y validación del entrenamiento

En el estado de entrenamiento podemos ver tres métricas de rendimiento a lo largo de las épocas:

1. Gráfico superior (Gradiente): Muestra la magnitud del gradiente durante el entrenamiento. El eje Y está en escala logarítmica, y la línea azul representa cómo la magnitud del gradiente disminuye a lo largo de las épocas. Una disminución en la magnitud del gradiente es típica a medida que el modelo se acerca a un mínimo en la función de pérdida. Si la línea se vuelve plana, puede significar que el modelo ha alcanzado un punto donde realizar más ajustes en los pesos no resulta en mejoras significativas en la función de pérdida (posible convergencia). En este caso la señal no es plana, por lo que la red no es "perfecta", quizás con mayor cantidad de datos, señales y entrenamientos, quizás la curva del gradiente pudiera llegar a ser plana, confirmando que el porcentaje de error de la red es cercano a cero.

2. Gráfico medio (Error): Este gráfico muestra el error a lo largo de las épocas. Al igual que en el gráfico del gradiente, el eje Y está en escala logarítmica y la línea azul muestra una disminución en el error a medida que avanza el entrenamiento, lo cual es esperado y deseado en el proceso de optimización.

3. Gráfico inferior (Validación): Representa la precisión de la red, dado que los valores están entre 0 y 1, lo cual es común para las métricas de clasificación. Los puntos rojos representan la precisión de validación en cada época, y los diamantes azules indican el mejor valor de precisión alcanzado hasta ese momento. Si la precisión de validación se mantiene constante y no mejora, podría ser un indicio de que el modelo no está aprendiendo más de los datos o que hay un sobreajuste en el conjunto de entrenamiento que no permite una mejora en el conjunto de validación.

En resumen, el descenso constante en el gradiente y el error sugiere que el modelo está mejorando y aprendiendo de los datos. Sin embargo, la precisión en el conjunto de validación no parece mejorar, lo que podría ser motivo de una investigación más detallada para ajustar el modelo, los hiperparámetros o para proporcionar más datos de entrenamiento, como hemos comentado en la primera grafica.

Por último veamos el histograma:

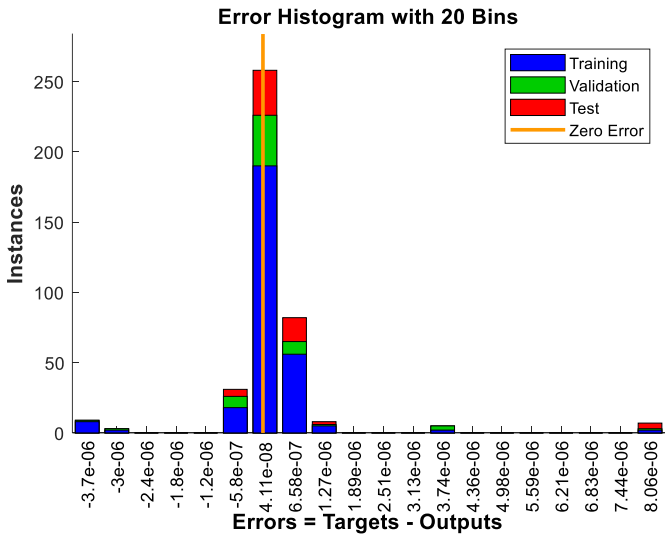

Ilustración 5.17 Histograma del entrenamiento

El histograma, o distribución de errores, dibuja unas barras que representan el número de instancias (o ejemplos) que resultaron en un determinado rango de error durante la evaluación de la red.

- **Azul (Training):** El número de instancias del conjunto de entrenamiento que tuvieron un cierto rango de error.
- **Verde (Validation):** El número de instancias del conjunto de validación que tuvieron un cierto rango de error.
- **Rojo (Test):** El número de instancias del conjunto de pruebas que tuvieron un cierto rango de error.

La línea amarilla marcada como *Zero Error* indica el punto de error cero, cercano a 4.11e-8, donde las predicciones del modelo son perfectamente precisas. Que la mayoría de las instancias en todos los conjuntos de datos (entrenamiento, validación y pruebas) están acumuladas cerca del error cero, indica que la red neuronal está realizando predicciones muy precisas en la mayoría de los casos.

Sin embargo, hay un pico significativo, especialmente en el conjunto de entrenamiento, donde el número de instancias con un error muy alto es notable. Esto podría ser un indicador de algunos ejemplos atípicos o *outliers* en el conjunto de datos que la red no está manejando bien. También podría ser un signo de sobreajuste si el modelo está funcionando excepcionalmente bien con el conjunto de entrenamiento.

El hecho de que las barras de validación y prueba sean más bajas en altura sugiere que hay menos instancias con errores altos en estos conjuntos, lo cual es positivo y apunta a un buen rendimiento general del modelo.

Sin embargo, de nuevo, es importante notar que la presencia de barras rojas y verdes en el área de error alto sugiere que el modelo no es perfecto y que hay algunas instancias en la validación y pruebas donde el modelo se equivoca.

5.2.2. Prueba de la red con nuevos datos

Después de haber visto los resultados y rendimientos de la red, ha llegado el momento de ejecutarla.

Se cargan en la red unos datos diferentes a los que se han usado en el entrenamiento. Como resultado debe calcular su porcentaje de 1s y después ira descartando todos aquellos que no cumplan el requisito, además de clasificar en aleatorias o no aleatorias las señales. Este es un fragmento de los que nos devuelve la red por pantalla:

```
Columns 40 through 52

    0     0     0     0     1     0     0     1     1     1     1     0     0

Señal 40: No Aleatoria
Señal 41: No Aleatoria
Señal 42: No Aleatoria
Señal 43: No Aleatoria
Señal 44: Aleatoria
Señal 45: No Aleatoria
Señal 46: No Aleatoria
Señal 47: Aleatoria
Señal 48: Aleatoria
Señal 49: Aleatoria
Señal 50: Aleatoria
Señal 51: No Aleatoria
Señal 52: No Aleatoria
```

Ilustración 5.18 Datos impresos por la red con la muestra de 0.8V

Puede darse por finalizada la parte del diseño de una red neuronal Feedforward funcional para la clasificación de señales aleatorias. Pasemos a ver las redes neuronales recurrentes LSTM.

5.3. Red neuronal LSTM

La red neuronal FNN creada hasta ahora ha dado los resultados esperados, pero no es un detector convincente. Un ejemplo de ello es que si de los 25.0000 datos, los primeros 12.000 son 0s y los siguientes 1s, la red FNN detectaría como señal aleatoria una señal escalón.

Las FNN no son útiles para el fin del proyecto, por lo que hay que complicar la red.

Las redes LSTM, por sus características, entrenamientos, estructuras, etc… serán de más utilidad.

Sin embargo, el desarrollo de este tipo de red no ha sido posible a lo largo del estudio, los resultados prácticos obtenidos ya se han presentado, dado que únicamente tenemos los de la red FNN. Sin embargo, las redes LSTM son mejores, es por ello que veremos dos ejemplos para detectar señales aleatorias. Veámoslo en detalle:

5.3.1. Basada en búsqueda de patrones

Las redes neuronales LSTM sabemos que son recurrentes, por lo que pasan varias veces por los mismos datos, almacenando información hasta obtener la salida deseada.

La búsqueda de patrones podría ser una muy buena forma de detectar señales aleatorias y evitaría caer en un error en el que podría caer la señal FNN diseñada.

Veamos varios ejemplos:

0101010101010001001110101

Esta señal contiene trece 0s y doce 1s. Si se usara esta señal en la red FNN, nos daría como resultado que es aleatoria, pero no es cierto. Hasta el doceavo digito se establece un patrón repetitivo de 01:

0101010101010001001110101

Esta es la mitad de la señal, en este punto podemos determinar que la señal no es aleatoria ya que al final se puede volver a detectarse este patrón:

0101010101010001001110101

Si bien es cierto que las señales tratadas no son de veinticinco dígitos, estos patrones pueden buscarse en todas, pero el número de dígitos del patrón deberá ajustarse a la cantidad de datos a tratar, veámoslo:

- Caso 1:

  10010010111111100101000001100111001011000100100100

  00100111100100111011111011001001101111101111001010 1

- Caso 2:

  10010010111111100101000001100111001011000100100100

  00100111100100111011111011001001101111101111001010 1

Aquí podemos ver como el número de dígitos debe ajustarse a los datos a evaluar. En el primer caso, la búsqueda de patrones se centra en dos dígitos, dando un total de veinticinco patrones detectados, esto quiere decir que cincuenta de los cien dígitos son predecibles, por tanto, la señal no es aleatoria.

El segundo caso es igual que el primero, pero la detección de datos se ha efectuado en base a cinco dígitos, dejando treintaicinco datos predecibles. Solo el cambio del tamaño del patrón puede hacer que una señal sea más o menos aleatoria.

Con 25.000 datos por señal, lo más razonable es ejecutar una búsqueda de patrones irregulares del 0.1%, es decir, veinticinco dígitos seguidos. Si la red es capaz de encontrar entre un 10%-15% de dígitos predichos querrá decir que la señal no es aleatoria.

La decisión de establecer estos porcentajes es meramente teórica, al no poder realizar una parte practica en donde entrenar a la red neuronal, pero para diferentes porcentajes de salida creo que predecir un 10% de información no es relevante para el conjunto. Este valor puede ser modificado según la delicadeza de la información.

Hay que remarcar que esta red si llegó a ser diseñada, pero únicamente para un patrón concreto, es decir, la red no detectó un patrón, simplemente cogió un patrón determinado y lo busco por las distintas señales. Este comportamiento podría usarse con señales FNN, por lo que carecía de sentido añadirlo en el proyecto. La complejidad de esta red es la capacidad de detectar patrones de *x* dígitos a escoger por el usuario.

5.3.2. Basada en cálculo de porcentaje individual

Esta, sin lugar a duda, sería la red neuronal perfecta para la detección de señales aleatorias debido a que basaría la clasificación digito por digito, de esta manera exprimiremos la máxima potencia de las redes neuronales LSTM.

La teoría principal es similar a la anterior, búsqueda de patrones, pero no siempre debe ser el mismo patrón.

La red debe, antes de leer el dato, revisar toda la secuencia que lleva, buscar posibles patrones y establecer la probabilidad que hay de que en esa posición haya un 1.

La red dictaminara que la señal entrante es aleatoria cuando haya habido una secuencia de probabilidades cercana al 50% a partir de la mitad de la secuencia. ¿Por qué a partir de la mitad de la secuencia? Cuando la señal se carga en la red, esta no tiene aún información, si parara al primer 50% de posibilidades pararía en a la segunda posición puesto que la única información que tiene es la del primer digito, es decir, nada.

Si establecemos que debe determinar la aleatoriedad de la señal en base a la consecuencia de porcentajes perfectos, a partir de la mitad le hemos dado tiempo suficiente a la red para leer y entender la señal, buscar patrones predecibles y regularidades.

Cierto es que esta teoría no puede ser afirmara debido a la no realización de esta, pero, la capacidad de cómputo necesario para entrenar una red similar con la cantidad de datos que tenemos es demasiado elevada como para tenerla en casa. Este tipo de red, con un archivo de entrenamiento como nuestra matriz de señales, debería ejecutarse en un super ordenador o un ordenador cuántico y tampoco sabríamos el tiempo que podría tardar a entrenarse.

En mi experiencia dejé una torre de sobremesa con una GPU Nvidia RTX 2060 Super con una potencia de 52 Teraflops durante 17h entrenando la red sin ninguna otra tarea en primer ni segundo plano. Tan siquiera pudo llegar a la señal 42. Sabiendo que a cada ciclo que hace mantiene la información de todas las señales previas, el tiempo y coste estimado de entrenamiento es demasiado elevado como para poder realizarse en un proyecto de este nivel, de hecho, la cantidad de datos que debe procesar seria de 25.000!, es decir, los datos de una señal con la función factorial. Un numero que no puede siquiera ser calculado.

6. Aplicaciones

La capacidad de una red neuronal para generar señales aleatorias, especialmente después de haber aprendido a clasificarlas, tiene varias aplicaciones potenciales en diferentes campos. Algunos de estos son:

1. **Criptografía y Seguridad Informática:**
   La generación de secuencias aleatorias es fundamental en la criptografía. Una red neuronal capaz de generar secuencias aleatorias verdaderas podría ser utilizada para crear claves criptográficas más seguras, mejorar los algoritmos de cifrado, o desarrollar sistemas de autentificación más robustos.

2. **Simulaciones y Modelado Estadístico:**
   En áreas como la física, la economía o la biología, las simulaciones que requieren la generación de datos aleatorios pueden beneficiarse de redes neuronales que produzcan secuencias aleatorias con propiedades estadísticas específicas.

3. **Pruebas y Análisis de Sistemas:**
   En ingeniería de software y hardware, las secuencias aleatorias se utilizan para probar la robustez y el comportamiento de los sistemas en condiciones impredecibles o bajo diferentes escenarios.

4. **Juegos y Entretenimiento**:
   En el desarrollo de videojuegos, la aleatoriedad es a menudo una característica deseable para generar entornos, eventos o comportamientos que sean únicos.

5. **Investigación en Inteligencia Artificial:**
   La habilidad de generar secuencias aleatorias puede ser útil en la investigación de algoritmos de IA, especialmente en áreas como el aprendizaje reforzado, donde la aleatoriedad puede ayudar a explorar y optimizar decisiones en entornos complejos.

6. **Arte y Creatividad Digital:**
   En el ámbito del arte digital y la música, la generación de patrones aleatorios puede ser utilizada para crear obras únicas y experimentales.

7. **Finanzas y Modelado de Mercados:**
   En finanzas, la generación de series temporales aleatorias puede ayudar en la modelización de precios de activos, riesgos y en la realización de pruebas de estrés bajo escenarios económicos impredecibles.

8. **Generación de Datos para Entrenamiento de Modelos:**
   En el aprendizaje automático, especialmente en situaciones donde los datos son escasos, la generación de datos sintéticos aleatorios pero realistas puede ser útil para entrenar modelos más robustos.

La capacidad de una red neuronal para generar secuencias aleatorias confiables y de alta calidad abriría nuevas puertas en estos campos, mejorando la eficiencia, la seguridad y la creatividad en sus respectivas aplicaciones. Peor va más allá, paremos, paremos atención a la primera aplicación. En el ámbito de la seguridad informática, la capacidad de una red neuronal para generar o clasificar secuencias aleatorias puede tener sub-aplicaciones como:

- **Generación de Claves Criptográficas:** Una de las aplicaciones más directas sería en la generación de claves criptográficas. Las claves fuertes y aleatorias son fundamentales para la seguridad de los sistemas de cifrado. Una red neuronal que pueda generar secuencias verdaderamente aleatorias podría ser utilizada para crear claves más seguras y difíciles de predecir o descifrar por métodos convencionales.
- **Mejora de los Algoritmos de Cifrado:** En los algoritmos de cifrado, la aleatoriedad juega un papel crucial. La generación de secuencias aleatorias complejas y no predecibles por redes neuronales podría incorporarse en algoritmos de cifrado para mejorar su robustez contra ataques criptoanalíticos.
- **Autenticación y Protocolos de Seguridad:** En los sistemas de autenticación, como la autenticación de dos factores (2FA) o los tokens de seguridad, la generación de códigos o tokens aleatorios es esencial. Las redes neuronales podrían ser utilizadas para generar estos códigos de manera más segura, reduciendo el riesgo de predicción o replicación.
- **Detección de Anomalías y Prevención de Intrusiones:** Las redes neuronales, especialmente aquellas entrenadas para reconocer patrones aleatorios, podrían ser útiles en la identificación de comportamientos anómalos en redes y sistemas. Esto incluye la detección de intentos de intrusión, actividades sospechosas o malware, basándose en desviaciones de los patrones normales o esperados de tráfico y uso de datos.
- **Pruebas de Penetración y Evaluación de Vulnerabilidades:** En pruebas de penetración, la generación de acciones o datos aleatorios puede ser utilizada para probar la robustez de los sistemas contra ataques impredecibles. Una red neuronal podría generar secuencias de prueba que ayuden a identificar vulnerabilidades desconocidas.
- **Generación de Ruido para Privacidad de Datos:** En escenarios donde la privacidad de los datos es crucial, como en la comunicación segura o en sistemas de almacenamiento de datos, la generación de "ruido" aleatorio por parte de redes neuronales puede ayudar a ofuscar los datos sensibles, haciéndolos menos susceptibles a ser descifrados o analizados.
- **Blockchain y Criptomonedas:** En el ámbito de blockchain y criptomonedas, la generación de números aleatorios es vital para varios procesos, como la creación de direcciones de cartera o en el mecanismo de consenso. Las redes neuronales podrían proporcionar un método más seguro y eficiente para generar estos números.

Si bien el potencial es considerable, también es crucial garantizar que los métodos basados en redes neuronales no introduzcan vulnerabilidades inadvertidas. Por lo tanto, cualquier aplicación en seguridad informática debe ser exhaustivamente probada y validada.

Antes de finalizar este apartado, permitidme exponer la hipótesis que dio rienda a este proyecto:

Actualmente no hay nada que no pueda ser hackeado, constantemente, día tras día, se libra una lucha interna binaria de datos, en la que se crean nuevos ataques y, por ende, nuevas defensas. Nadie puede imaginarse algo que sea “inhackeable”, al igual que el ser humano no puede entender el concepto de infinito o inmortalidad.

Partiendo de esta idea llegue a los números aleatorios y a un proyecto futuro, veámoslo.

6.1. Hipótesis

Usaremos como ejemplo un dron y un control remoto. Actualmente la comunicación entre estos dos dispositivos es sencilla, hablando en términos de seguridad informática, si tuviéramos la señal que los comunica es fácil interceptarla y hackear al dron.

Para evitar este problema teorizamos una manera de crear un nuevo protocolo de comunicación entre dispositivos que cambiara la señal constantemente de manera aleatoria y únicamente el receptor y emisor supieran la señal que deben enviarse para funcionar. A esto, en seguridad informática se le denomina *seguridad mediante la obscuridad* o *seguridad a través de la aleatoriedad*, vamos a analizarlas y ver la veracidad de esta hipótesis:

Viabilidad y Consideraciones

- **Cambio Aleatorio de Señales:** La idea de cambiar las señales de comunicación aleatoriamente es conceptualmente similar a las técnicas utilizadas en criptografía, como los sistemas de cifrado de clave pública o los protocolos de intercambio de claves como Diffie-Hellman. La aleatoriedad mejora la seguridad al hacer que sea mucho más difícil para un atacante predecir o interceptar la comunicación.
- **Sincronización y Gestión de Claves**: Un desafío clave en este enfoque es cómo ambos dispositivos (el dron y su control) pueden sincronizar y conocer las señales aleatorias mutuamente sin que un tercero pueda predecirlas o interceptarlas. Esto normalmente implica algún tipo de intercambio de claves o un acuerdo previo sobre un método para generar y validar estas señales aleatorias.
- **Seguridad a Través de la Obscuridad:** Aunque cambiar las señales aleatoriamente puede añadir una capa de seguridad, es importante no confiar únicamente en la obscuridad como defensa. La seguridad robusta generalmente requiere más que solo aleatoriedad; también implica el uso de algoritmos criptográficos probados y técnicas de autenticación fuertes.
- **Retos Prácticos:** Implementar un sistema así en la práctica presentaría varios retos. Por ejemplo, la necesidad de asegurar que la comunicación sea resistente a interrupciones y que los dispositivos puedan recuperarse rápidamente de errores de sincronización.
- **Vulnerabilidad a Ataques:** Aunque cambiar las señales aleatoriamente puede dificultar el hackeo, no es infalible. Los ataques de repetición, por ejemplo, podrían seguir siendo una preocupación si un atacante logra interceptar y retransmitir una señal válida.

Viendo esto, la idea de utilizar señales que cambian aleatoriamente para la comunicación entre un dron y su control es conceptualmente viable y refleja principios utilizados en sistemas de comunicación seguros. Sin embargo, su implementación efectiva requeriría una consideración cuidadosa de la sincronización, la gestión de claves, y la resistencia a varios tipos de ataques.

La seguridad robusta suele requerir un enfoque multifacético que incluya tanto la aleatoriedad como técnicas criptográficas sólidas, es decir, estamos aún lejos de poder ver este tipo de comunicación o de vivir en un sistema en donde la información pueda ser segure e invulnerable, pero, la perfección no existe, por lo que nos queda la ciencia y la probabilidad.

7. Conclusión

Como bien se dijo en un inicio, este proyecto es una pequeña parte de un gran proyecto futuro, los conocimientos aquí obtenidos son mas que suficientes para adentrarse en el estudio de las redes neuronales que cada vez son mas complejas y con muchas mas capas de neuronas. Es por ello por lo que los resultados del proyecto, aun teniendo fallos, pueden mejorarse notablemente, las hipótesis o ideas planteadas son teóricamente viables. Además, el uso de funciones *rand()* durante el preprocesado de los datos permite tener infinitas señales de entrenamiento diferentes y, aun no siendo aleatorias, tienen la capacidad de confundir y entrenar las redes. Por el momento la red FNN diseñada no tiene la estricta necesidad de usar este tipo de preprocesados complejos, sin embargo, las futuras LSTM aprovecharan esta pseudo aleatoriedad algorítmica para entrenarse con el fin de obtener detecciones mas precisas y con un grado de error más pequeño.

El desarrollo de redes neuronales capaces de detectar aleatoriedad binaria ya es una realidad. Sin embargo, el siguiente paso aún está lejos. Los próximos retos para superar son la creación de redes neuronales capaces de generar señales aleatorias y posteriormente el desarrollo de nuevos protocolos de comunicación mas seguros. Sin embargo, aun no tenemos una infraestructura capaz de aguantar este tipo de protocolos y siquiera sabemos si pueden llegar a ser posibles en el corto plazo. Es importante investigar y desarrollar este tipo de tecnologías para garantizar una mayor seguridad de los datos que, desde hace unos años, pueden considerarse una moneda de cambio.

8. Bibliografía

Referencias

[1] J. D. Fortuny, A versatile framework for the statistical characterization of CMOS time-zero and time-dependent variability with array-based ICs, Cerdanyola del Vallés, 2019.

[2] «Wikipedia,» 24 Enero 2024. [En línea]. Available: https://en.wikipedia.org/wiki/Pseudorandom_number_generator.

[3] «Wikipedia,» 17 Enero 2024. [En línea]. Available: https://en.wikipedia.org/wiki/Hardware_random_number_generator.

[4] «Wikipedia,» 25 Enero 2024. [En línea]. Available: https://en.wikipedia.org/wiki/List_of_random_number_generators#Pseudorandom_number_generators_(PRNGs).

[5] E. S. d. C. Irigaray, «Youtube,» 09 Marzo 2022. [En línea]. Available: https://www.youtube.com/watch?v=RzEjqJHW-NU.

[6] «ANU QRNG - Escuchador cuántico,» [En línea]. Available: https://qrng.anu.edu.au/.

[7] A. Luengo, «Youtube,» 08 Noviembre 2022. [En línea]. Available: https://www.youtube.com/shorts/tk_A4uze8rk.

[8] ". Baker", «Youtube,» 23 Mayo 2021. [En línea]. Available: https://www.youtube.com/watch?v=yuZDZehV8Tg.

[9] D. A. Muller, «Youtube,» 16 Julio 2014. [En línea]. Available: https://www.youtube.com/watch?v=sMb00lz-IfE.

[10] Java, «Jep356: Enhanced Pseudo-Random Number Generators,» 01 Febrero 2023. [En línea]. Available: https://openjdk.org/jeps/356.

[11] D. Strmecki, «Baeldung,» 08 Enero 2024. [En línea]. Available: https://www.baeldung.com/java-17-random-number-generators.

[12] Wikipedia, «Mersenne Twister,» 27 Enero 2024. [En línea]. Available: https://en.wikipedia.org/wiki/Mersenne_Twister.

[13] Universitam, «CREAN GENERADOR CUANTICO DE NUMEROS ALEATORIOS: ENCRIPTARA DATOS,» 30 Septiembre 2010. [En línea]. Available: https://universitam.com/academicos/noticias/crean-generador-cuantico-de-numeros-aleatoria-encriptara-datos/.

[14] U. o. C. Boulder, «Quantum Tunneling and Wave Packets,» [En línea]. Available: https://phet.colorado.edu/en/simulations/quantum-tunneling.

[15] M. G. Castell, «Predicción de índices bursátiles por medio de redes neuronales artificiales. Aplicación al caso del IBEX-35,» 2020. [En línea]. Available: https://m.riunet.upv.es/bitstream/handle/10251/152246/Guimer%C3%A1%20-%20Predicci%C3%B3n%20de%20%C3%ADndices%20burs%C3%A1tiles%20por%20medio%20de%20redes%20neuronales%20artificiales.%20Aplicaci%C3%B3n....pdf?sequence=2&isAllowed=y.