
UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE CIENCIAS

ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

# *Redes LSTM para el reconocimiento de voz aplicado un conjunto de dígitos*

**SEMINARIO DE TESIS II**

*Autor: Víctor Jesús Sotelo Chico*

*Asesor: Dr. Antonio Morán Cardenas*

Diciembre, 2018

# *Resumen*

En las últimas décadas el campo de la inteligencia artificial se ha desarrollado rápidamente. Estudios sobre reconocimiento de imágenes y voz han sido realizados en los últimos años. Actualmente este campo requiere realizar un gran números de cálculos para entrenar redes neuronales. Este proceso puede tomar mucho más tiempo dependiendo del tamaño del tipo de dato.

El análisis de voz es una tarea importante dentro del campo de la inteligencia artificial debido a que la señal de voz varía con el tiempo. Además de que es necesario un pre-procesamiento para extraer las características de esta. Métodos como el MFCC han permitido que tareas como el reconocimiento de voz sean posibles. Encontrar modelos adecuados para el procesamiento de estos datos es una tarea importante.

Este seminario presenta modelos de redes neuronales como el RNN, LSTM como una alternativa para tratar datos dinámicos y realizar tareas de clasificación para un conjunto de datos de la pronunciación de los números del 0 al 9. Además mostrará las precisiones de los distintos modelos.

**KEYWORDS**: MFCC, RNN, LSTM, SGD.

# Índice general

# Índice de figuras

# Índice de cuadros

# Índice de Código

# Índice de Acrónimos

| | |
|---|---|
| **SVM** | Super Vector Machine |
| **SGD** | Stochastic gradient descent |
| **DNN** | Deep Neural Network |
| **CNN** | Convolutional Neural Network |
| **RNN** | Recurrent Neuronal Network |
| **LSTM** | Long short-term memory |
| **DFT** | Discrete Fourier Transform |
| **DCT** | Discrete Cosine Transform |
| **ETC** | Etcétera |

# *Agradecimientos*

Agradezco a mis padres por todo el apoyo incondicional durante todos estos años de estudio, a mis compañeros de clase en especial aquellos que me brindaron su voz para construir el conjunto de datos empleado en este seminario y a mi asesor por ayudarme en este proyecto.

# Capítulo 1

# Introducción

Dentro de la inteligencia artificial, las redes neuronales profundas desempeñan un papel muy importante, debido a que éstas permiten entrenar a las computadoras para que realicen tareas que nuestros cerebros realizan de manera natural como el reconocimiento de voz, imágenes y patrones. Una característica de las redes neuronales profundas es la gran cantidad de capas que poseen. Esto permite que las redes sean capaces de extraer características de los datos ya sean imágenes o voz.
La comunicación entre los seres humanos se realiza por medio del habla, la cual es emitida como una señal de voz, debido a la cantidad de información que esta señal posee ha permitido el estudio y desarrollo de aplicaciones como identificadores biométricos, conversión de voz en texto, etc. Tareas como éstas requieren un análisis complejo debido a que se necesita tratar problemas como la reverberación y el ruido presentes en el entorno.

## 1.1. Motivación

La inteligencia artificial constituye una base muy importante en el campo de la computación, ésta mezcla un conjunto de disciplinas como la estadística y ciencia de la computación con el objetivo de construir modelos que puedan permitir a las computadoras realizar tareas que hace algunos años hubiesen sido consideradas imposibles. El avance de la inteligencia artificial ha permitido el desarrollo de programas que sean capaces de realizar tareas sin haber sido explícitamente programadas para hacerlas.

Entre los modelos existentes en la inteligencia artificial las redes neuronales han tenido un amplio desarrollo en las últimas décadas debido que han sido utilizadas para tratar datos más complejos como las imágenes o señales de voz. El tratamiento de la voz es una tarea más compleja que ha sido estudiada en los últimas décadas debido a sus aplicaciones en la identificación biométrica, robótica y procesamiento de lenguaje natural.

Actualmente, los sistemas de reconocimiento de voz están presentes en distintos software, muchos de ellos permiten romper algunas barreras presentes como por ejemplo, para personas con alguna discapacidad física existen software que permiten a las personas ciegas leer la pantalla de su computador transformando el texto en voz, otro ejemplo son las personas sordas dado que existen software capaces de transformar voz en texto.

Dentro del reconocimiento de voz existen problemas presentes como el ruido de voz y reverberación, fenómeno sonoro producido por la reflexión, los cuales deben ser considerados para que una red neuronal sea robusta ante estos problemas.

Además, debido a la gran cantidad de información que la señal de voz transmite es importante utilizar las herramientas adecuadas para reducir costo computacional ya sea mediante el uso de algoritmos eficientes o hardware más potentes.

En la actualidad existen APIs para el reconocimiento de voz, pero estás siempre poseen limitaciones por lo cual en esta investigación se busca desarrollar un sistema de reconocimiento de voz que nos permita reconocer el habla y resuelva los problemas ya mencionados, es decir, que sea robusta, esto nos permitirá entender y comprender como funciona el lenguaje humano. Este nuevo conocimiento nos permitirá desarrollar en este seminario una red neuronal capaz de reconocer palabras del lenguaje español con un conjunto de datos de números.

## 1.2. Objetivos

El objetivo de este seminario es el diseñar un sistemas capaz de procesar la voz y transformarla en texto mediante el uso de distintos de modelos de redes neuronales recurrentes aplicado a un conjunto de datos del lenguaje español.

Específicamente, los objetivos de este trabajo con respecto al sistema son:

- Entender el funcionamiento de las redes neuronales profundas.
- Conocer el proceso involucrado en el habla humana.
- Estudiar procesamiento de las señales de voz.
- Diseñar un sistema robusto capaz de reducir los problemas de reverberación y ruido.
- Mostrar los resultados obtenidos y explicarlos basándonos en la información estudiada.

Y los objetivos con respecto a las competencias académicas desplegadas en el trabajo son:

- Desarrollar un mejor entendimiento de las redes neuronales y sus aplicaciones, para así poder lograr afrontar problemas en el campo de la inteligencia artificial.
- Obtener la capacidad de discriminar entre los algoritmos para tratar las señales de voz.
- Desarrollar el criterio necesario para trabajar con datos de audio.
- Obtener un conocimiento de las herramientas y recursos que existen actualmente para abordar problemas de aprendizaje profundo, además de poder analizar que herramientas son adecuadas para algunos problemas.

## 1.3. Estructura del Seminario

- **Introducción:**

  En este capítulo introductorio se comenta sobre el tema a tratar, las motivaciones, intereses, objetivos con los cuales se planteo el presente seminario.

- **Estado del Arte:**

  Este capítulo muestra los trabajos e investigaciones ya realizadas, además de algunas aplicaciones que motivaron al presente seminario y las investigaciones mostrarán el interés del problema planteado.

- **Redes Neuronales:**

  En este capítulo daremos una introducción a las redes neuronales de manera general, además veremos los tipos de redes existentes y nos enfocaremos más en las redes neuronales recurrentes.

- **Procesamiento de la señal de voz**

  En este capítulo conoceremos más el proceso del ingreso de la señal a nuestra red como. También describiremos los algoritmos usados en este proceso.

- **Resultados:**

  Se mostrarán los resultados obtenidos en las pruebas de los optimizadores además de describir los resultados.

- **Conclusiones y Trabajo Futuro:**

  En este capítulo se plantean las conclusiones y se detalla algunos inconvenientes encontrados durante el trabajo. Además que se comprueba la teoría descrita en el capítulo 3.

# Capítulo 2

# Estado del Arte

En este capítulo se describirán las investigaciones anteriores con relación al Aprendizaje Automático, además de sus aplicaciones. También se verán algunas investigaciones referente al reconocimiento de voz y los algoritmos usados para estas tareas.

Este trabajo también presentará investigaciones referentes a Aprendizaje Profundo, exclusivamente nos enfocaremos a la Redes Neuronales Recurrentes (RNN), ya que son usadas en este seminario.

## 2.1. Aprendizaje Automático

El uso del Aprendizaje Automático representa una gran ventaja para empresas que manejan gran cantidad de datos debido a que permiten descubrir patrones y analizar estos.

### 2.1.1. GPGPU Performance and Power Estimation Using Machine Learning

Un equipo conformado por investigadores[1] de AMD y The University of Texas at Austin, fueron quienes propusieron el uso de redes neuronales para predecir el rendimiento de una GPU. En la actualidad existen empresas dedicadas a la creación de GPUs, en el proceso una parte fundamental es la verificación del rendimiento de las GPUs. Actualmente existen simuladores conocidos como GPGPU-SIM que permiten realizar estimaciones precisas pero

estos presentan algunas dificultades como el tiempo empleado en configurarlos en base al hardware real, no obstante, este proceso se encuentra propenso a errores.

### 2.1.2. Handshape recognition for Argentinian Sign Language using ProbSom

Investigadores de la Universidad de La Plata, en Argentina conformado por Franco Ronchetti, Facundo Quiroga, César Estrebou, y Laura Lanzarini[2], desarrollaron un sistema que permite el reconocimiento de lenguaje de señas argentino. Esta investigación fue realizada usando una técnica llamada ProbSom, esta puede ser comparada con otros métodos como las Máquinas de Soporte Vectorial, Bosques Aleatorios y Redes Neuronales.

## 2.2. Aprendizaje Profundo

Dentro del área de Aprendizaje Automático encontramos Deep Learning o Aprendizaje Profundo el cual consiste en un conjunto de algoritmos que modela abstracciones de alto nivel.
En esta sección hablaremos de un paper que nos sirvió de introducción al campo del aprendizaje profundo.

### 2.2.1. Deep Machine Learning - A New Frontier in Artificial Intelligence

Este trabajo de investigación fue realizado por investigadores Thomas Karnowski, Derek Rose - Oak Ridge National Laboratory y Itamar Arel - University of Tennessee [3], el objetivo principal de este trabajo fue presentarnos el aprendizaje profundo como un camino para la imitación del cerebro humano y sus principales cualidades como el reconocimientos de objetos, rostros, etc.
En este paper presenta una introducción a los temas de *Convolutional Neural*

*Network (CNN)* y *Deep Belief Network*, nos describe a las CNN como una familia de redes neuronales multicapas que fueron diseñadas para tratar datos de dimensionalidad 2 como lo son las imágenes y los videos.
Por otro lado, también nos muestra las aplicaciones del aprendizaje profundo como: análisis de documentos, detección de voz, rostro, procesamiento natural del lenguaje, etc.

La aplicación de la inteligencia artificial no solo se realizó para investigaciones, sino también en algunas empresas privadas que apoyan el campo del Aprendizaje Profundo con el objetivo de buscar sus aplicaciones comerciales, entre estas empresas tenemos a: Numenta y Binatix.

### 2.2.2. On Optimization Methods for Deep Learning

Un equipo de la Universidad de Standford realizó unas pruebas con el objetivo de encontrar métodos adecuados para un entrenamiento en aprendizaje profundo. El equipo se percato de lo común que resulta el uso de gradiente de descenso estocástica (SGD por sus siglas en inglés) en aprendizaje profundo. Se realizaron pruebas con otros métodos de optimización como la gradiente conjugada y Limited memmory BFGS(L-BFGS) los cuales permitieron acelerar el proceso de entrenamiento de algoritmos de Aprendizaje Profundo mostrando en su mayoría mejores resultados que el SGD. "Usando L-BFGS el modelo CNN alcanza el 0.69 % en el estándar del MNIST dataset. "[4]

## 2.3. Reconocimiento de Voz

### 2.3.1. Review of Algorithms and Applications in Speech Recognition System

Este trabajo fue realizado por CR Rashmi del *Cork Institute of Tecnology (CIT)* en la investigación se describe el reconocimiento del habla como un método para poder realizar distintas aplicaciones como: reconocimiento

del hablante (Identificación Biométrica), emociones, acento, etc. Además se presentan distintos algoritmos que usan transformada de fourier y modelos probabilísticos que son aplicados a tareas de reconocimiento de voz.
Esta investigación se centra en los algoritmos para la extracción de características y coincidencia de patrones.
Entre principales algoritmos para la extracción de características que muestran tenemos: RCC, MFCC, LPC, etc. Siendo el MFCC uno de los mejores para realizar tareas de reconocimiento del hablante. Por otro lado, en coincidencia de patrones tenemos algoritmos como VQ, HMM, SVM, MLP, GMM, etc. Para tareas de reconocimiento de emociones y géneros destaca el GMM.

### 2.3.2. SPEECH RECOGNITION WITH DEEP RECURRENT NEURAL NETWORKS

Esta investigación realizada por los investigadores Alex Graves, Abdel-rahman Mohamed y Geoffrey Hinton de la Universidad de Toronto. El principal objetivo de esta investigación fue utilizar las redes neuronales recurrentes para el reconocimiento de fonemas utilizando el conjunto de datos TIMIT, el uso de las redes neuronales resulta adecuado debía a la naturaleza dinámica del habla.
El tipo de entrenamiento utilizado fue *end-to-end training* además se utilizan distribuciones diferenciales para todas las posibles salidas fonéticas. Entre los métodos estudiados para estas distribuciones de salida tenemos *Connectionist Temporal Classification (CTC)* este método decide si emitir una etiqueta de acuerdo al fonema identificado o no emitirla.
"Las redes neuronales entrenadas con CTC son generalmente bidireccionales para asegurar que la distribución de salida dependa únicamente de la secuencia de entrada ". Otro método estudiado en esta investigación de RNN transducer el cual predice un fonema dado uno previo. En la experimentación fueron usados 2 reguladores *parada temprana* y *pesos de ruidos*. "CLC se obtiene un porcentaje de error de 23.9 % a 18.4 % a medida que se aumentan los niveles

ocultos de la red ".

### 2.3.3. EFFICIENT CEPSTRAL NORMALIZATION FOR ROBUST SPEECH RECOGNITION

Esta investigación estuvo a cargo de Fu-Hua Liu, Richard M. Stern, Xuedong Huang y Alejandro Acero del departamento de ingeniería eléctrica e informática de la Universidad de Carnegie Mellon. El objetivo de esta investigación fue comparar los procedimientos basados en cepstrum usados en el reconocimiento de voz sobre variedad de entornos acústicos.

Para los autores "Los estudios muestran que los sistemas de reconocimiento de voz que son diseñados para ser independientes del hablante funcionan de manera incorrecta cuando se prueban distintos micrófonos o entornos con los cuales fueron entrenados ". Una solución a este problema es la aplicación de técnicas de normalización cepstral como *SNR - Dependet Cepstral Normalization (SDCN)*, *Fixed Codeword-Dependet Cepstral Normalization (FCDCN)* y *Multiple fixed codeword-dependent cepstral normalization (MFCDCN)*.

Entre el FCDCN y MFCDCN este último resulta ser mejor debido a que no requiere del entrenamiento del especifico como lo hace el FCDCN.

### 2.3.4. DELTA-SPECTRAL CEPSTRAL COEFFICIENTS FOR ROBUST SPEECH RECOGNITION

El presente paper fue realizado por Kshitiz Kumar, Chanwoo Kim y Richard M. Stern del departamento de ingeniería eléctrica e informática de la Universidad de Carnegie Mellon. En esta investigación se busca diseñar un sistema de reconocimiento de voz más robusto mediante el uso de las características delta-espectrales. En palabra de los autores "*A pesar que las características delta cepstrales capturan la información dinámica y mejoran la precisión del reconocimiento de voz no son robustos contra el ruido y reverberación.*"

### 2.3.5. Voice Recognition Algorithms using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques

Esta investigación fue realizada por Lindasalwa Muda, Mumtaj Begam y I. Elamvazuthi, en esta se describen los principios involucrados en el reconocimiento de voz como la extracción de características y patrones. Al igual que otras investigaciones los autores afirman *"La voz es una señal de infinita información. Un análisis directo y sintetizado es requerido para la gran cantidad de señal contenida en la señal. "*

El reconocimiento de voz puede ser divido en 2 fases una fase de entrenamiento en que el hablante provee información o muestras de su voz y una fase de testeo donde se verifica que la voz de entrada coincida con la del hablante del proceso de entrenamiento. Los investigadores utilizan MFCC para la extracción de características y DTW para reconocimiento de los patrones. En las pruebas de identificación se logro encontrar una distorsión optima en DTW que permitió reconocer al hablante.

### 2.3.6. Convolutional Neural Networks for Speech Recognition

Entre los distintos modelos existentes en redes neuronales profundas ha surgido la necesidad de realizar modelos híbridos como *Deep Neural Network-Hidden Markov Model (DNN-HMM)* y *Gaussian Mixture Model - Hidden Markov Model (GMM-HMM)*. La presente investigación encontró que la primera es más robusta debido a la capacidad que poseen las redes neuronales al momento de modelar correlaciones complejas de las características de la voz.

Los resultados mostraron que usando CNN se obtuvo una mejora de 6-10 % en tareas de phone recognition y búsqueda de voz en un gran vocabulario.

### 2.3.7. TIME-FREQUENCY CONVOLUTIONAL NETWORKS FOR ROBUST SPEECH RECOGNITION

Este estudio usa a las redes neuronales convolucionales profundas (DCNN) para generar una arquitectura más robusta contra el ruido y otras variaciones del entorno. DCNN usa los filtros de convolución para así poder remover las distorciones cross-spectral una cualidad de este tipo de red es que no utiliza la convolución sobre el tiempo. Por lo cual esta convolución es trata en esta investigación definiendo un esquema llamado time-frequency convolutional network (TFCNN), esta red aplica 2 capas de convolución paralelas una para la frecuencia y otra para el tiempo.
En las pruebas realizadas se nota una mejora al probar con ruidos, canales y datos dañados con reverberación a comparación del uso de DCNN.

## 2.4. Conclusiones

A medida que tratamos muchos problemas vemos la necesidad de encontrar optimizadores adecuados para los diferentes tipos de problemas. En el área de Aprendizaje Profundo comúnmente se trabaja en el campo de reconocimiento de imágenes.
A pesar de las mejoras mediante el uso de GPUs este tipo de problemas necesitan soluciones óptimos para obtener un mejor rendimiento. Métodos como Nesterov Momemtum, RMSProp y Adam surgen como principales opciones para realizar optimizaciones de la gradiente de descenso.

# Capítulo 3

# Redes Neuronales Artificiales

En este capítulo daremos una introducción a las redes neuronales artificiales veremos su uso en el aprendizaje automático para tareas de clasificación y además estudiaremos 2 tipos especiales de redes neuronales como son las *redes convolucionales y recurrentes.*
En este seminario se dará más énfasis a las redes recurrentes y su aplicación en el procesamiento de voz para lograr la tarea de conversión voz a texto.

## 3.1. Conceptos básicos

A continuación describiremos algunos conceptos necesarios para el entendimiento de las redes neuronales artificiales más adelante se mostrarán nuevos de acuerdo al tipo de red.

### 3.1.1. Neuronas

En la biología, la neurona es conocida como la unidad fundamental del cerebro humano, el cual está compuesto por millones de neuronas interconectadas entre sí (sinapsis). El trabajo de las neuronas consiste en recibir información, procesarla y enviarla a otras neuronas.
Este modelo fue copiado en 1943 por Warren S. McCulloch y Walter H. Pitts para poder diseñar un neurona artificial que es análoga a las neuronas del cerebro humano, esta neurona artificial tomará una cantidad n de entradas $x_1, x_2, x_3, .., x_n$ las entradas serán multiplicadas (producto interno) por pesos

$w_1, w_2, w_3, .., w_n$, además se puede añadir una constante que llamaremos bias ($b$) para producir un salida.

La entrada a la neurona será la suma total de los productos z= $\sum_{i=1}^{n} w_i x_i + b$ , el valor de z , esta se evaluará con una función f de tal forma que nuestra salida será $y = f(z)$.

En la ecuación 3.1 observamos la misma salida expresada en forma vectorial para nuestros vectores $x = [x_1 x_2 x_3 ... x_n]$ y $w = [w_1 w_2 w_3 ... w_n]$

$$y = f(x \cdot w + b) \tag{3.1}$$

### 3.1.2. Funciones de Activación

La función $f$ mencionada anteriormente es una función no lineal, conocida como **función de activación.**

La tarea principal de la función de activación es introducir no linealidad a la salida de una neurona. Esto es importante debido a que la vida real no trabajamos solo con datos lineales y de esta forma la neurona puede aprender representaciones no lineales.

Entre funciones de activación tenemos algunas comúnmente usada como:

- **Sigmoid**: Toma un valor real, y lo transforma en una valor en el rango de 0 a 1.
  $\sigma(x) = \frac{1}{1+e^{-x}}$
- **tanh**: Toma por entrada un valor real y lo transfroma a un número en le rango de -1 a 1.
  $\tanh(x) = \frac{2}{1+e^{-x}} - 1$
- **ReLu**: o Unidad lineal rectificada es una función que para valores menores que 0 asigna 0 y para valores mayores

$$f(x) = \begin{cases} 0 & x < 0 \\ \text{x} & x \geq 0 \end{cases}$$

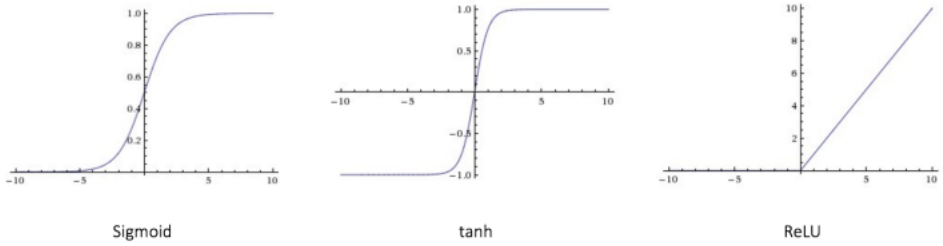


FIGURA 3.1: Funciones de activación
Fuente: *https://ujjwalkarn.me*

### 3.1.3. Redes Neuronales Artificiales

Las redes neuronales artificiales (ANN) toman la arquitectura del cerebro como inspiración para la construcción de sistemas inteligentes. Actualmente son la base para el desarrollo de la inteligencia artificial.
Una red neuronal está constituida por las uniones de neuronas.
En la figura 3.2 podemos ver la comparación entre una neurona biológica y un artificial etiquetadas con A y B respectivamente. Además observamos que las redes neuronales artificiales (etiqueta D) imitan el la unión biológicas de las neuronas o sinapsis(Etiqueta C).

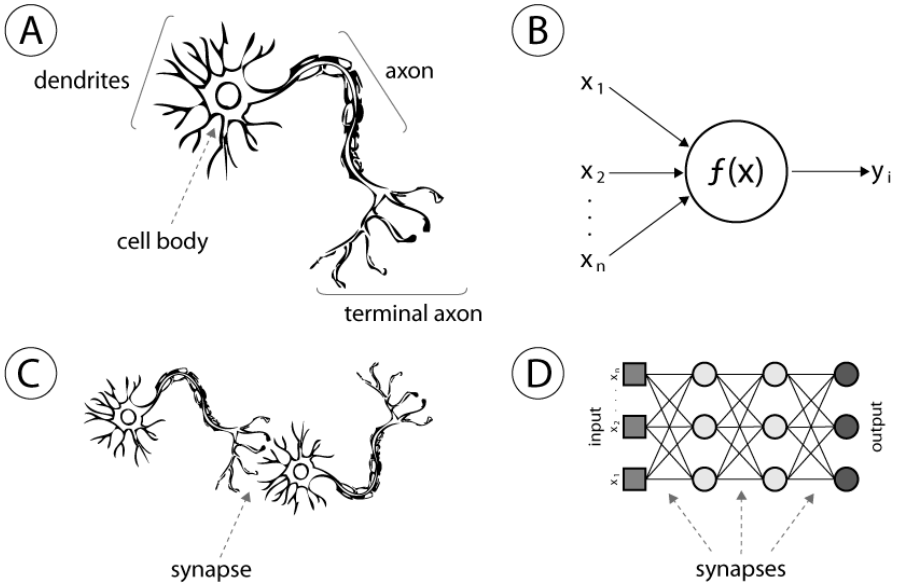


FIGURA 3.2: Redes neuronales biológicas y artificiales
Fuente: *https://medium.com*

## 3.2. Redes Neuronales FeedFoward

Estas redes fueron de las primeras y más simples. Contienen múltiples neuronas (nodos) ordenadas en capas de modo que los nodos en capas adyacentes se conectan. Cada una de estas conexiones posee un peso asociado a dicha conexión.
En la figura 3.3 mostramos el esquema de Redes FeedFoward con sus distintas capas (layer).

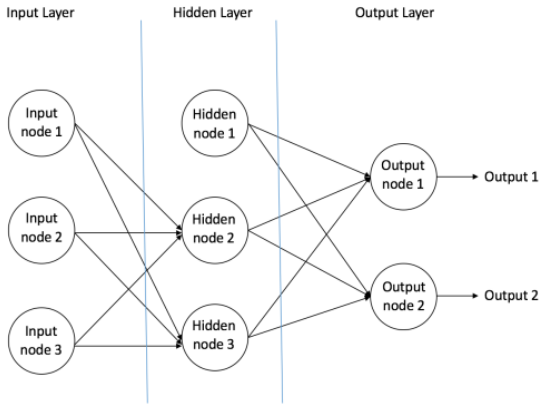


FIGURA 3.3: Esquema de Redes Neuronales FeedFoward
Fuente: *https://ujjwalkarn.me*

- **Nodo de entrada (Input Node):** Provee información a la red. En conjunto representan la capa de entrada, ningún calculo es realizado en esta capa solo se transfiere la información a la capa oculta.
- **Nodo Oculto (Hidden Node):** El trabajo de los nodos ocultos es calcular y transferir la información hacia a el nodo de salida. Una Red FeedFoward tiene solo una capa de entrada y una salida pero puede tener múltiples capas ocultas.
- **Output Node:** Su tarea principal es realizar cálculos y transferir la información fuera de la red.

En las Redes Neuronales FeedFoward, la información solo se propaga en una dirección hacia *adelante* desde los nodos de entradas pasando por los nodos ocultos hacia los nodos de salida. No existen ciclos en este tipo de red.
Dentro de las redes neuronales FeedFoward tenemos algunos ejemplos:

- **Perceptron Simple:** Es un red prealimentada simple que no posee capa oculta. Solo puede aprender de funciones lineales.
- **Perceptron Multicapas:** Esta red posee una o más capas ocultas. Este perceptron puede aprender de funciones no lineales.
- **Redes neuronales de convolución:** Este tipo de redes neuronal consta de muchas capas y resuelven problemas de definición de múltiples pesos.

### 3.2.1. Algoritmo de propagación hacia atrás

El algoritmo de propagación hacia atrás trata de aprender de los errores, en el aprendizaje supervisado los conjuntos de entrenamiento se encuentran etiquetados. Por lo cual podemos conocer la salida esperada.
El algoritmos se aplica de la siguiente forma:

1. Se toma un ejemplo y se asignan pesos aleatorios a todas las conexiones de la red. Luego por medio de las conexiones y funciones de activación se calcula la salida en las capas ocultas y de salida.
2. Se calcula el error total, luego se propagan estos errores hacia atrás a través de la red y se calcula la gradiente, luego se usan métodos como gradientes de descenso para ajustar los pesos y reducir el error en la capa de salida.
3. Se repite el proceso con los otros ejemplos

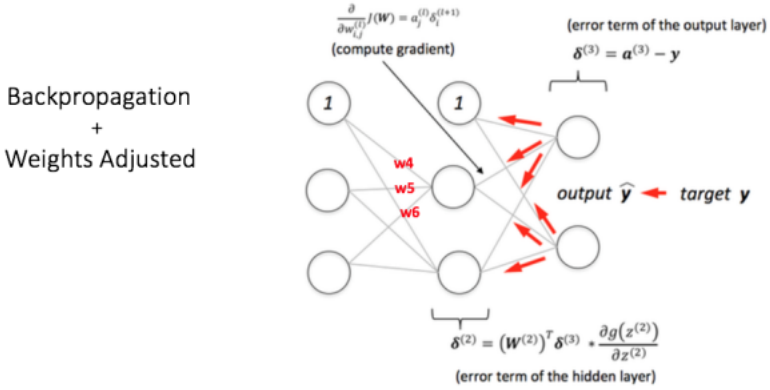


FIGURA 3.4: Propagación hacia atrás
Fuente: *https://ujjwalkarn.me*

## 3.3. Redes Neuronales Convolucionales

Las CNN son un tipo de Redes Neuronales FeedFoward que son especiales para procesar datos como imágenes las cuales son más difíciles de tratar en una red neuronal tradicional, como por ejemplo en el caso del perceptron multicapas.

El término *convolucional* hace referencia a la operación lineal matemática usada. Las redes neuronales convolucionales usan esta operación para aprender de las características de mayor orden presente en los datos. La primera CNN fue creada por Yann LeCun. Estas redes neuronales fueron inspiradas en la corteza visuales de los animales. Las células de la corteza visual, éstas se activan para realizar tareas como el reconocimiento de patrones.

Entre sus usos más comunes tenemos el reconocimiento de imágenes y lenguaje natural.

### 3.3.1. Estructura de una imagen

Debido a que las redes neuronales convolucionales son usadas comúnmente con imágenes, es importante conocer cual es la estructura de una imagen y cómo es que la computadora comprende y utiliza esta información.

Las imágenes están constituidas por una sucesión de píxeles, podemos entender el pixel como la menor unidad homogénea en color de una imagen digital. Teniendo este concepto, podemos dividir la información de una imagen de la siguiente forma:

- **Width**: El ancho de la imagen medido en pixeles
- **Height**: El alto de la imagen medida en pixeles.
- **Canales RGB**: Estos canales contiene la información de los colores y profundidad de una imagen. Este canal guarda la información en tres canales Red, Green y Blue.

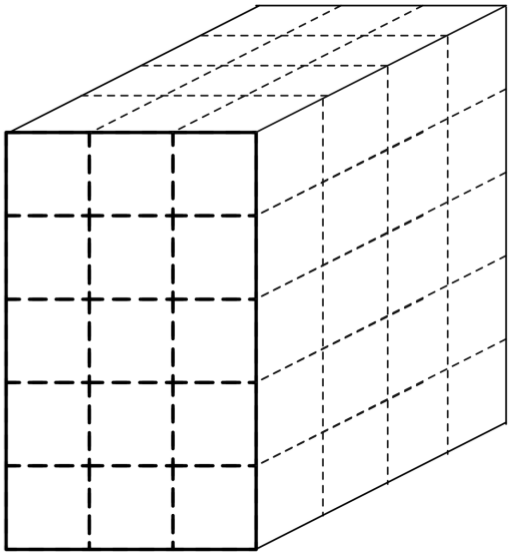

FIGURA 3.5: Estructura de la imagen de entrada
Fuente: *Deep Learning by Adam Gibson, Josh Patterson*

Teniendo en cuenta esta forma de dividir la información de una imagen podemos resaltar la ventaja de usar Redes convolucionales en lugar de usar una red neuronal multicapas.
Las redes multicapas toman un vector de una dimensión como entrada, si quisiéramos entrenar un perceptron multicapas con imágenes de 32x32 píxeles y con 3 canales RGB necesitaríamos crear 3072 pesos ($w_i$) para una sola neurona en la capa oculta. Esta generación excesiva de peso hace que la tarea resulte

complicada usando redes multicapas.
De esta forma surge la idea de recurrir a un tipo de redes neuronales que faciliten la tarea sin consumir muchos recursos.

### 3.3.2. Capas de una CNN

Las redes neuronales convoluciones pueden ser divididas en distintas capas, cada una de ellas con una tarea específica para el tratamiento de la información. En esta sección describiremos cada una de estas capas.

**Input layer**

Esta capa es la encargada de cargar y almacenar la información de las imágenes para luego procesarlas en la red. Esta información contiene detalles de ancho, alto en píxeles y el número de canales de imagen. Las entradas de esta capa corresponden a la imagen vista en la figura 3.5.

**Convolutional layers**

Es una de las capas más importante en el diseño de las CNNs, esta capa es la encargada de transformar la entrada (imagen o convolución anterior) usando las conexiones de las neuronas en capas anteriores.

Para entender más a fondo esta capa debemos definir la operación de *convolución*.
La *convolución* es una operación matemática que describe una regla de como fusionar 2 conjuntos de información.
Las convoluciones son usadas principalmente como un detectores de características cuyas entradas son la capa de entrada u otra convolución. En la figura 3.6 observamos la operación de convolución que por medio del uso de un kernel o filtro de convolución extrae características de la imagen, por ejemplo detalles como bordes de una imagen.
Haciendo analogía con los pesos en las redes neuronales convencionales, las redes convolucionales poseen el *filtro o kernel* , esto resulta beneficioso, ya que

no se tendrá que definir un peso para cada neurona.

El kernel de la figura 3.6 será desplazado a lo largo de las dimensiones espaciales. Durante el desplazamiento, el kernel se multiplicará por los datos de entrada dentro de su límite, produciendo una sola salida al mapa de características.

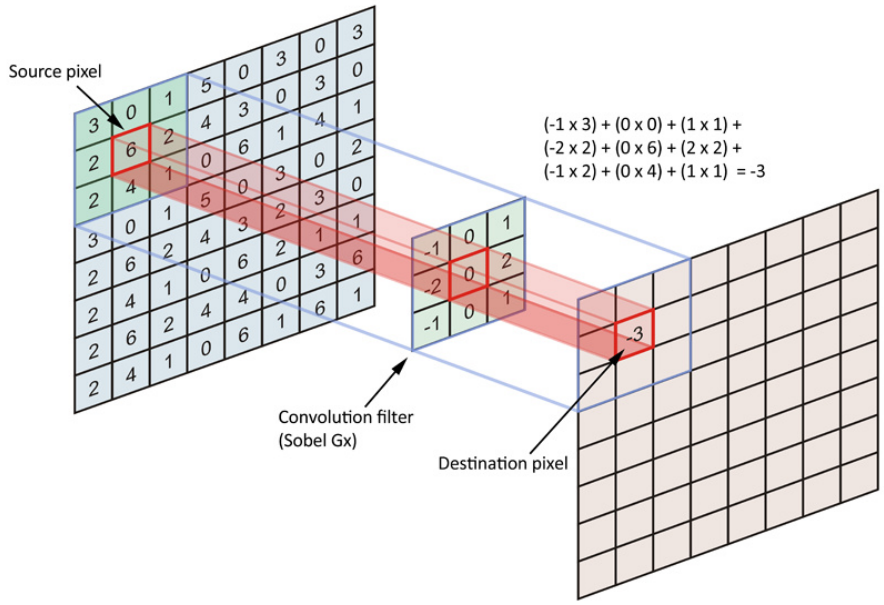


FIGURA 3.6: Operacion de convolución
Fuente: *www.openresearch.ai*

Las capas convolucionales aplican transformaciones o funciones de activación al conjunto de entrada, luego el mapa de activación generado se apilará a lo largo de dimensión de profundidad para construir el volumen de salida. **Componentes de la capa de convolución**.

Las capas convolucionales poseen parámetros. La gradiente de descenso tiene la funcion de entrenar a estos parámetros de modo que las clases sean consistentes con las etiquetas en el conjunto de entrenamiento. Entre estos parámetros describiremos a los:

**Filtros**

Los filtros son una función que posee ancho (width) y alto (height) más pequeños que la entrada. Los filtros son aplicados a través de del ancho y alto de la entrada, pero también pueden ser aplicados a lo largo de la profundidad.

**Pooling layers**

Este tipo de capas se encuentran entre las capas convolucionales. Se encarga de reducir el tamaño espacial (ancho, alto) de los datos de representación. Esta capa reduce la representación de los datos progresivamente a través de la red y ayuda a controlar el *overfitting*.
Esta capa utiliza la operación *max()* para cambiar el tamaño de los datos de entrada espacialmente, a esta operación se le conoce como max pooling. Ésta funciona de la siguiente forma toma un filtro de $nxn$, y la operación $max$ toma el mayor de los números en el área de filtro.

Por ejemplo en caso tener una imagen de entrada $32 \times 32$ píxeles y se aplica un filtro de $2 \times 2$, como resultado obtendremos una salida de $16 \times 16$ píxeles. Esto reduce cada segmento de profundidad en el volumen de entrada por un factor de 2.

**Fully Connected Layers**

En esta capa se calcula el puntaje de las clases que usaremos como salida de red, esta será la encargada de reconocer a que clase pertenece una imagen de prueba de acuerdo a su puntaje o probabilidad. Las dimensiones del volumen de la salida son [1x1xN], donde el valor de N corresponde al número de

clases de salida que se están evaluando. En el caso del MNIST (dataset para reconocimiento de dígitos), el valor de N es igual a 10, número que corresponde a los 10 dígitos distintos que posee el dataset ($0, ..., 9$).
Esta capa tiene conexión entre todas sus neuronas y las de la capa anterior. Esta capa realiza las transformaciones del volumen de datos de entrada. Estas son funciones de activación en el volumen de entrada y los parámetros (pesos y bias de las neuronas).

## 3.4. Redes Recurrentes

Las redes neuronales recurrentes aparecieron en los años 1980s, actualmente se han desarrollado más estudios debido a la mejora de hardware. Estas tipo de redes son utilizadas principalmente para tratar con una información de secuencia, por ejemplo series de tiempo, audio, sentencia de oraciones, etc.
La principal diferencia de la redes con las redes neuronales feed-foward es que en las neuronas de las redes recurrentes las salidas regresan a la entrada de esta manera mantiene información de un estado anterior. Esto permite que nuestro modelo sea cambiante cada vez que este se alimente con una secuencia nueva.

Dentro de una capa recurrente se tiene los siguientes tipos de conexiones :

- **Entrantes:** Son aquellas que emanan de la capa previa.
- **Salientes:** Estas conexiones son dirigidas a todas las neuronas de las capas consecuentes.
- **Recurrentes:** Se encargan de propagar la información entre las neuronas de las misma capa.

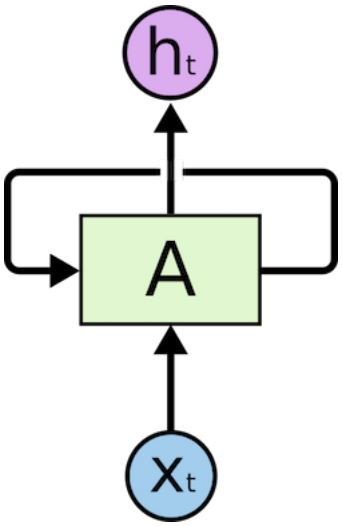


FIGURA 3.7: Neurona Recurrente
Fuente: *https://medium.com*

Es posible representar a la redes recurrentes como si fueran redes feed-fowards esto lo realizamos *desenrollando* la neurona como vemos en la figura 3.8 el $x_t$ representa un estado final de esta forma podemos representar la red como un conjunto de estados a través de un lapso de tiempo. En esta forma las técnicas de backpropagation pueden ser usadas en la redes recurrentes.

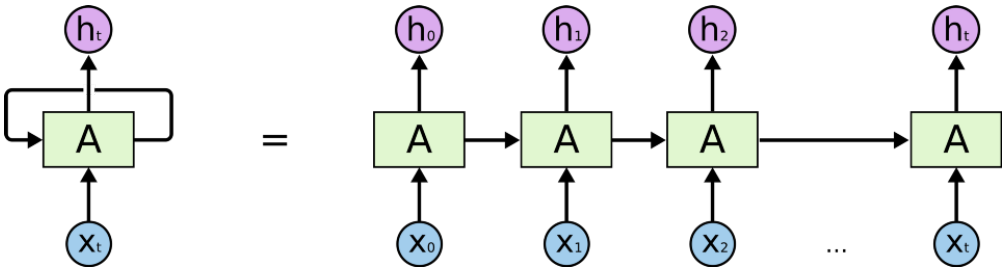


FIGURA 3.8: Versión desenrollada
Fuente: *https://towardsdatascience.com*

### 3.4.1. Propagación hacia atrás a través del tiempo (BPTT)

Las redes neuronales feedfoward, la propagación hacia atrás se encarga de transmitir el error de la salida hacia a las capas ocultas asignados a los pesos(w) responsabilidad del error general mediante el uso de la derivada parcial $\frac{\partial E}{\partial w}$. En estas redes es común usar la propagación hacia atrás debido a que el par entrada salida posee un tamaño fijo situación que no ocurre con las secuencia de datos.

Las redes neuronales recurrentes utilizan una extensión de la propagación hacia atrás llamada *Propagación hacia atrás a través del tiempo*, o BPTT por sus siglas en inglés, la cual se aplica para secuencia de datos. Este algoritmo consta de los siguiente pasos:

- Dado unos lapsos de tiempo entre la entrada y la salida de la red.
- Se desenrolla la red(Ver Figura 3.8) , calcula y acumulada los errores de cada lapso de tiempo.
- Enrolla la red y actualiza los pesos para reducir el error.

"Si la secuencia de datos consta de 1000 lapsos de tiempo, este será el número de derivadas requeridas para una simple actualización"[8]

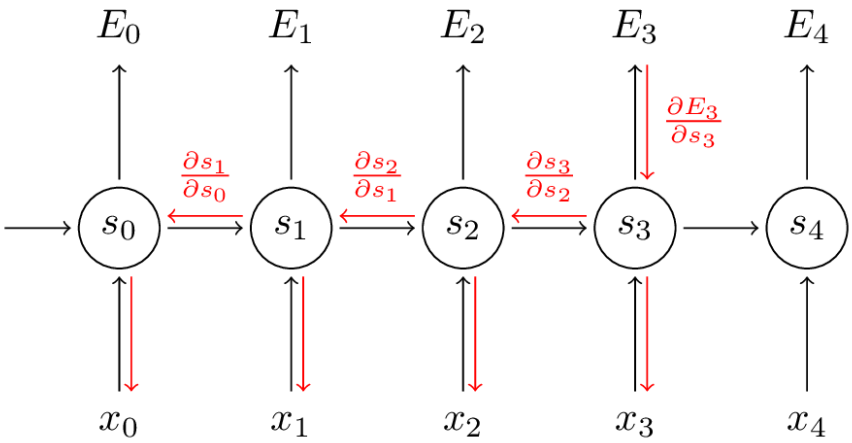


FIGURA 3.9: BPTT
Fuente: *http://www.wildml.com*

En este esquema de básico de redes neuronales recurrentes utilizamos la función de activación softmax para calcular los valores de la celda y utilizamos pesos $(U, W, V)$ de esta forma tenemos que:

$$\begin{aligned} s_t &= \tanh(Ux_t + Ws_{t-1}) \\ \hat{y}_t &= softmax(Vs_t) \end{aligned} \tag{3.2}$$

Donde $s_t$ son los valores de la celda y $\hat{y}_t$ es la predicción. A continuación calcularemos el error de nuestra red utilizando el valor de la predicción $\hat{y}_t$ y el valor real $y_t$.

$$\begin{aligned} E_t(y_t, \hat{y}_t) &= y_t log y_t \\ E_t(y, \hat{y}) &= \sum_t E_t(y_t, \hat{y}_t \end{aligned} \tag{3.3}$$

Nuestro objetivo será ajustar los pesos $(U, W, V)$ por lo cual usaremos la gradiente de descenso con respecto a los pesos los cuales serán calculados en la ecuación 3.4.

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W} \tag{3.4}$$

Para calcula esta gradiente haremos uso de la regla de la cadena para simplificar los cálculos usaremos un ejemplo el $E_t$ para $t = 2$.

$$\begin{aligned} \frac{\partial E_2}{\partial V} &= \frac{\partial E_2}{\partial \hat{y}_2}\frac{\partial \hat{y}_2}{\partial V} \\ \frac{\partial E_2}{\partial V} &= \frac{\partial E_2}{\partial \hat{y}_2}\frac{\partial \hat{y}_2}{\partial z_2}\frac{\partial \hat{z}_2}{\partial V} \\ \frac{\partial E_2}{\partial V} &= (\hat{y}_2 - y_2) \otimes s_3 \end{aligned} \tag{3.5}$$

Donde $s_2 = Vs_3$ ahora calcularemos para W como $s_2 = tanh(Ux_t + Ws_1)$ al derivar obtenemos la ecuación 3.6 luego se sumarán las contribuciones en cada lapso de tiempo.

$$\frac{\partial E_2}{\partial W} = \frac{\partial E_2}{\partial \hat{y}_2}\frac{\partial \hat{y}_2}{\partial s_2}\frac{\partial \hat{s}_2}{\partial s_k}\frac{\partial s_k}{\partial W} \tag{3.6}$$

### 3.4.2. Propagación dinámica hacia atrás (DBP)

Este algoritmo fue propuesto por Pineda[9] en 1987. Este tipo de redes neuronales son utilizados a las redes neuronales dinámica, la cual está representada por la ecuación no lineal de tiempo discreto mostrará en la ecuación 3.7

$$\begin{cases} x(k+1) = f(x(k), W, \theta) \\ y(k) = h(x(k)) \end{cases} \tag{3.7}$$

- $x = [x_1..., x_n]^T$: Vector de estado de la red dinámica
- $x_i$ : estado interno de la neurona $i$
- $W = [w_{ij}]_{nxn}$: Matriz de valores de pesos.
- $\theta = [\theta_1..., \theta_n]^T$: vector somático.
- $f : R^n \times R^{n \times n} \times R^n \to R^n$ función continua y diferenciable.
- $h(x) : R^n \to R^m$ función continua y diferenciable.

Para el caso de redes recurrentes se tienen conexiones hacia adelante y hacia atrás entre las capas y las neuronas. Ahora los pesos $w_{ij}$ representa la conexión sináptica entre la i-ésima y j-ésima y $\theta_i$ umbral en la i-ésima neurona.

$$\begin{cases} x_i(k+1) = f_i(x(k), w_i, \theta_i) \\ y(k) = h(x(k)) \end{cases} \tag{3.8}$$

Donde :

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1n} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & w_{n3} & \dots & w_{nn} \end{bmatrix} = \begin{bmatrix} w_1^T \\ w_2^T \\ \vdots \\ w_d^T \end{bmatrix} \tag{3.9}$$

$$w_i = \begin{bmatrix} w_{i1} \\ w_{i2} \\ \vdots \\ w_{in} \end{bmatrix} \; i = 1, 2...., n \tag{3.10}$$

La ecuación 3.7 nos muestra que el comportamiento dinámico de la neurona depende los pesos
($w_{ij}$) y el parámetro $\theta_i$s Dado $t = [t_1, ..., t_m]^T$ un valor objetivo de nuestra red y nuestro propósito es encontrar los valores de pesos adecuados para estimar $t = h(x^f)$ . De esta forma podemos definir el error en la ecuación 3.11 .

$$E = \frac{1}{2}\sum_{l=1}^{m}[t_l - h_l(x^f)]^2 = \frac{1}{2}\sum(J_l^f)^2 \tag{3.11}$$

Donde $J_l^f - t_l - h_l(x^f)$

A continuación veremos como se actualizarán los parámetros $w_{ij}$ y $\theta_i$ usando tasa de aprendizaje $\eta_w$ y $\eta_\theta$ respectivamente.

$$\begin{aligned}\triangle w_{ij} &= -\eta_w \frac{\partial E}{\partial w_{ij}} \\ &= \eta_w \sum_{p=1}^{n}\sum_{l=1}^{m}[t_l - h_l(x^f)]\frac{\partial h_l(x^f)}{\partial x_p^f}\frac{\partial x_p^f}{\partial w_{ij}} \\ &= \eta_w \sum_{p=1}^{n}\sum_{l=1}^{m} J_l^f \frac{\partial h_l(x^f)}{\partial x_p^f}\frac{\partial x_p^f}{\partial w_{ij}}\end{aligned} \tag{3.12}$$

$$\begin{aligned}\triangle \theta_i &= -\eta_\theta \frac{\partial E}{\partial \theta_i} \\ &= \eta_\theta \sum_{p=1}^{n}\sum_{l=1}^{m}[t_l - h_l(x^f)]\frac{\partial h_l(x^f)}{\partial x_p^f}\frac{\partial x_p^f}{\partial \theta_i} \\ &= \eta_\theta \sum_{p=1}^{n}\sum_{l=1}^{m} J_l^f \frac{\partial h_l(x^f)}{\partial x_p^f}\frac{\partial x_p^f}{\partial \theta_i}\end{aligned} \tag{3.13}$$

Derivando las ecuaciones 3.12 y 3.13

$$\begin{aligned}\triangle w_{ij} &= -\eta_w z_i^f \frac{\partial f_i(x^f, w_i, \theta_i)}{\partial w_{ij}} \\ \triangle \theta_i &= -\eta_\theta z_i^f \frac{\partial f_i(x^f, w_i, \theta_i)}{\partial \theta_i}\end{aligned} \tag{3.14}$$

Donde $z_i^f$ esta determinada por la siguiente ecuación:

$$z_i^f = \sum_{p=1}^{n}\frac{\partial f_p}{\partial x_i^f} z_p^f + \sum_{l=1}^{m} J_l^f \frac{\partial h_l(x^f)}{\partial x_i^f} \tag{3.15}$$

"Un algoritmo de propagación dinámica hacia atrás es utilizado para entrenar la red en configuraciones de bucle cerrado, cuando existe una ruta de retroalimentación entre la salida de la sección de filtro digital y las entradas a la red neuronal. "[10]

### 3.4.3. Desaparición de la gradiente

Las redes recurrentes ofrecen una gran ventaja al manejar *secuencias de datos* pero el trabajar con este tipo de datos también pueden repercutir en un problema conocido como la desaparición de la gradiente, o *Vanishing Gradient*, el cual es una de la principales dificultades en las redes recurrentes.
La gradiente de descenso nos permite actualizar los valores de nuestros pesos para que nuestra red continué aprendiendo, pero si esta gradiente desaparece, debido a que toma un valor pequeño, nuestra red deja de aprender en términos sencillos en esto consiste el problema de *desaparición de gradiente*.

Dado que en la propagación hacia atrás se calcula la gradiente usando la regla de la cadena podemos suponer que usamos la función sigmoid de la figura 3.9 notamos que su derivada para valores positivos y negativos grandes toma el valor de 0 lo que producirá que la gradiente desaparezca y nuestra red deje de aprender al no poder actualizar.

Entre la soluciones para este problema se encuentran usar otras funciones de activación como Relu o Elu, también se pueden usar métodos de normalización para los batchs.

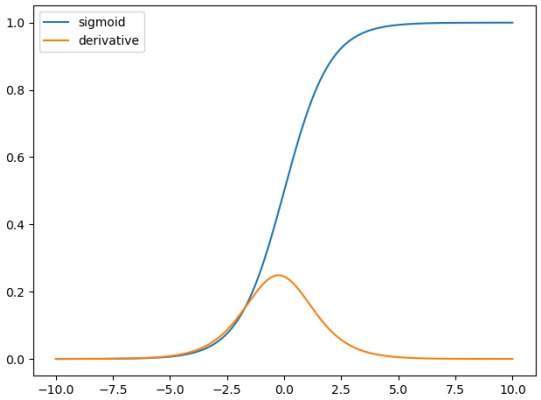


FIGURA 3.10: Función sigmoid y su derivada
Fuente: *https://medium.com*

### 3.4.4. Long Short Term Memory (LSTM)

LSTM es un tipo de red neuronal recurrente que fue propuesto por Sepp Hochreiter y Jurgen Schmidhuber.[11] para aprovechar las ventajas de las redes recurrentes y combatir el problema de la desaparición de gradiente.

En la redes recurrentes se trasmite la información pero ha medida que transcurre el tiempo están son olvidadas, a esto se le llamó el *problema de las dependencias a largo plazo*. Las LSTM buscan solucionar este problema, por lo cual estas se encargan de transmitir la información y *recordarla* mientras los lapsos de tiempo pasan imitando la *capacidad de la memoria humana para recordar.*

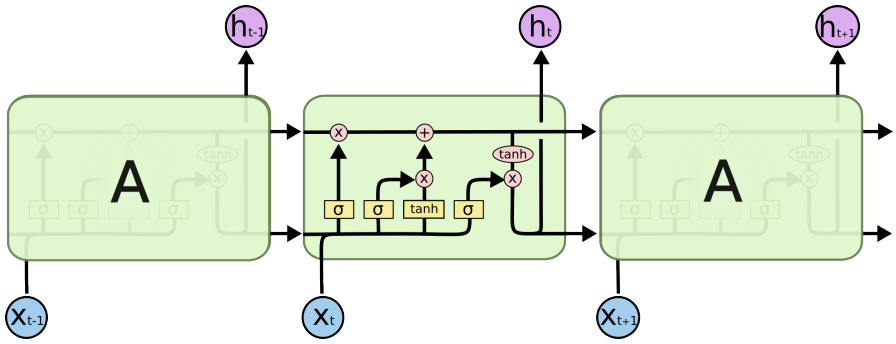


FIGURA 3.11: Estructura en cadena
Fuente: *http://colah.github.io*

En la siguiente figura observamos el esquema de una unidad, o neurona, de LSTM y estudiaremos su comportamiento.

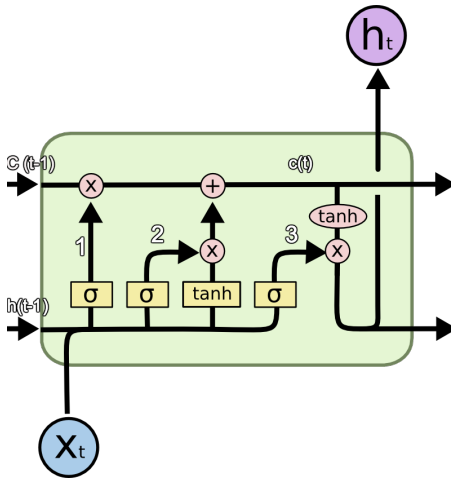


FIGURA 3.12: Unidad de LSTM
Fuente: *https://towardsdatascience.com*

- $X_t$: entrada actual
- $\sigma$: capa sigmoid
- $tanh$: capa tanh
- $h_{t-1}$: salidas de la última unidad.
- $C_{t-1}$: memoria de la última unidad.
- $h_t$: salida actual.
- $C_t$: memoria actualizada

Principalmente la idea de LSTM gira entorno a las celdas de estado $c_t$, esta será la encargada de añadir nueva información o removerla si ya no es necesaria.

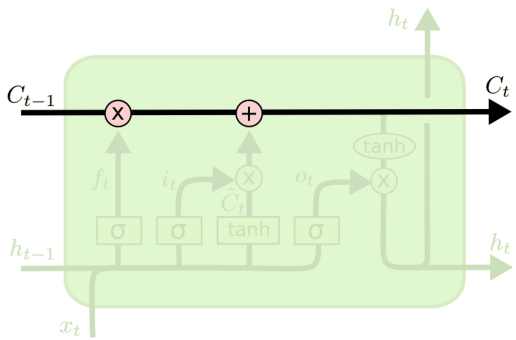


FIGURA 3.13: celda de estado
Fuente: *http://colah.github.io*

El valor de la celda de estado($C_t$) y la salida($h_t$) :

1. Nuestro LSTM decidirá que información será desechada de nuestra célula de estado. Esta decisión utilizará una capa sigmoidal llamada *forget gate layer* la cual genera un número entre 0 y 1, lo cual definirá la cantidad de información que mantendrá.

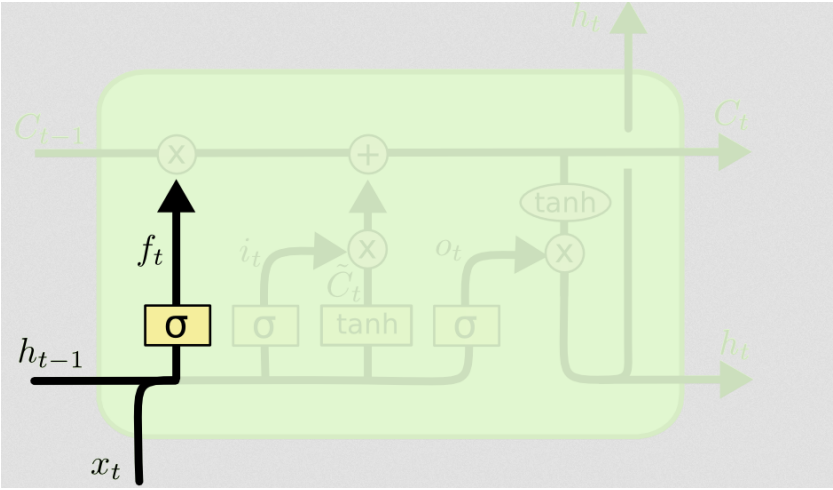


FIGURA 3.14: capa sigmoidal
Fuente: *http://colah.github.io*

$$f_t = \sigma(W_t \cdot [h_{t-1}, x_t] + b_f) \tag{3.16}$$

2. Una vez que se ha olvidado es importante es aceptar nueva información y almacenarla. Esto será realizado en los siguientes pasos.

   *a*) Una capa sigmoid llamada *input gate layer* decide que valores serán actualizados($i_t$).

   *b*) Una capa tanh crea un vector de valores posibles, o valores candidatos, $\hat{C}_t$ .

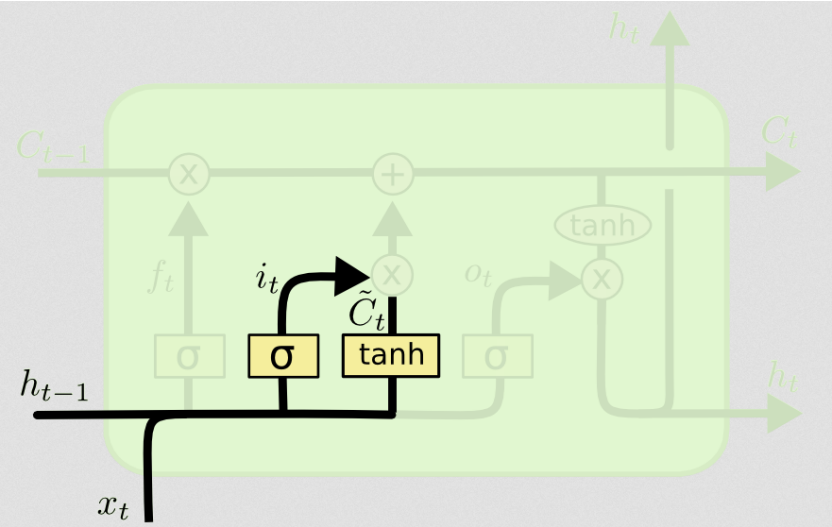


FIGURA 3.15: capa tanh
Fuente: *http://colah.github.io*

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \hat{C}_t &= \sigma(W_C \cdot [h_{t-1}, x_t] + b_C) \end{aligned} \tag{3.17}$$

*c*) Finalmente se combinan ambos para actualizar la celda de estado.

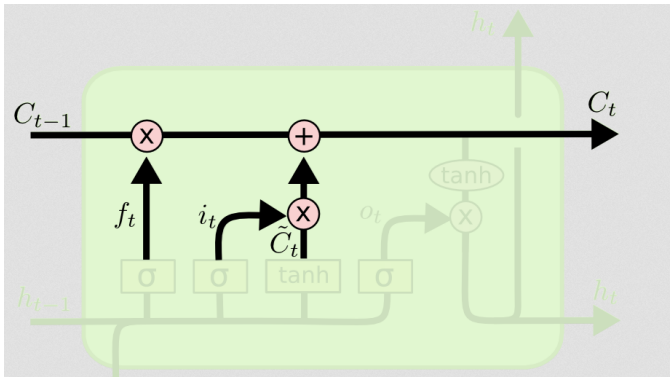


FIGURA 3.16: Actualización del $C_t$
Fuente: *http://colah.github.io*

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t \tag{3.18}$$

3. Finalmente para obtener la salida de este módulo $C_t$

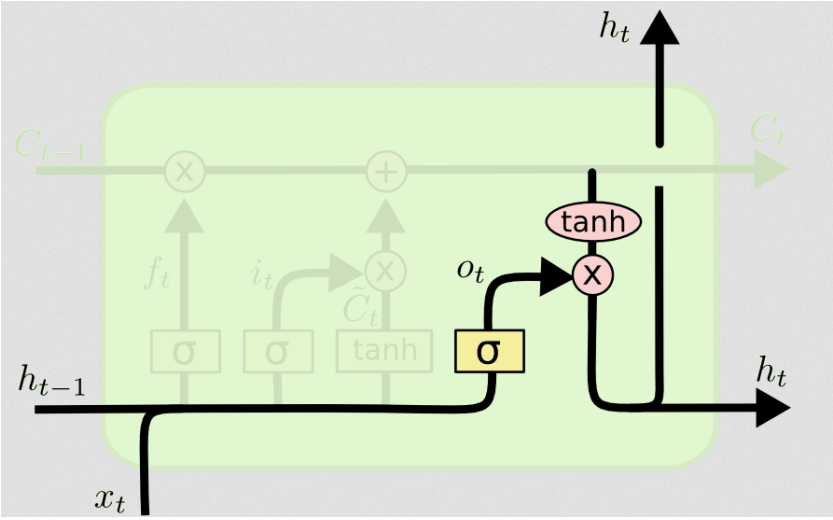


FIGURA 3.17: Calculo del $h_t$
Fuente: *http://colah.github.io*

$$
\begin{aligned}
o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
\tag{3.19}
$$

En la figura 3.18 mostramos una idea lo que son las unidades ocultas dentro de la celdas LSTM, estás unidades están relacionadas con las capacidad de aprender de nuestra red y determinan el tamaño del hidden state $h_t$.

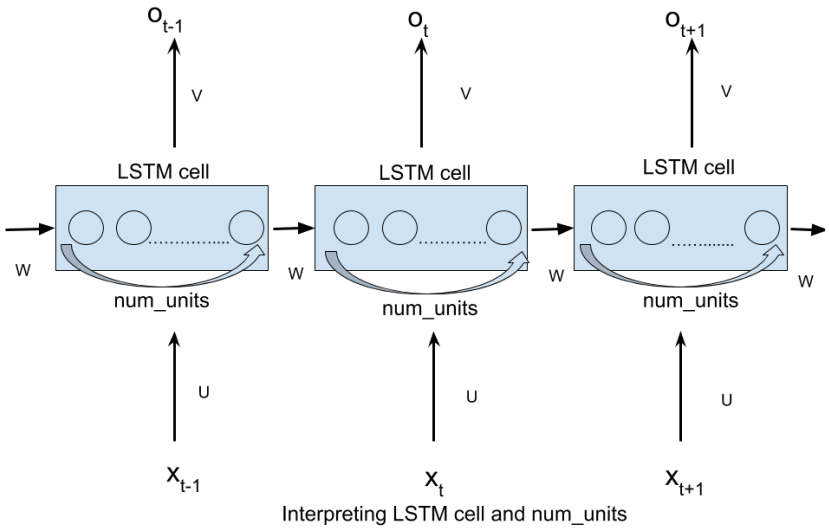


FIGURA 3.18: Número de unidades ocultas
Fuente: *https://stackoverflow.com*

Una variante de las redes recurrentes son las redes recurrentes bidireccionales, la cual consiste en aprender de una secuencia de datos en 2 direcciones siguiendo lapsos de tiempo de la secuencia hacia adelante y atrás. Al capturar la información de la secuencia invertida nos permite mejorar nuestra predicción. "Para superar las limitaciones de una RNN regular proponemos un modelo bidireccional de redes recurrentes(BRNN) la cual puede ser entrenada con la información de entrada en el pasado y en el futuro de un marco especifico de tiempo "[12]

# Capítulo 4

# Procesamiento señal de voz

En este capítulo conoceremos más acerca del procedimiento para procesar la señal de voz, además repasaremos algunos conceptos previos para entender en procesamiento y luego introduciremos 2 conceptos importantes *Extracción de características y la coincidencia de patrones.*

## 4.1. Conceptos previos

### 4.1.1. Voz

Los seres humanos diariamente nos comunicamos por medio del habla utilizando nuestras voces somos capaces de transferir información en forma de *ondas sonoras*.
Estas ondas transmiten una gran cantidad de información usando el aire como medio de transmisión. Las propiedades como variaciones en amplitud al empezar o finalizar una palabra varían en el transcurso del tiempo, por lo cual es importante analizar segmentos de tiempo donde estas propiedades se mantengan aisladas de esta manera nos aseguramos que la señal de voz no cambie.

### 4.1.2. Audios

Los audios son un tipo de *datos no estructurados*, es decir datos que no se encuentran en algún tipo de estructura de datos, estos tipos de datos son los que más se encuentran el mundo real como imágenes y audio. Una características de

estos es que son complejos en su recolección y preparación para la realización de un análisis.

El audio puede capturarse mediante la grabación de nuestro entorno pero para que este audio sea entendido por las computadoras necesita un formato adecuado como: wav, mp3 y wma.

Debido a que el sonido es una señal de onda podemos analizar y obtener valores numéricos de este. En la figura 4.1 observamos una onda de la cual obtenemos valores almacenando las alturas de puntos equidistantes de esta forma guardamos información de esta onda.

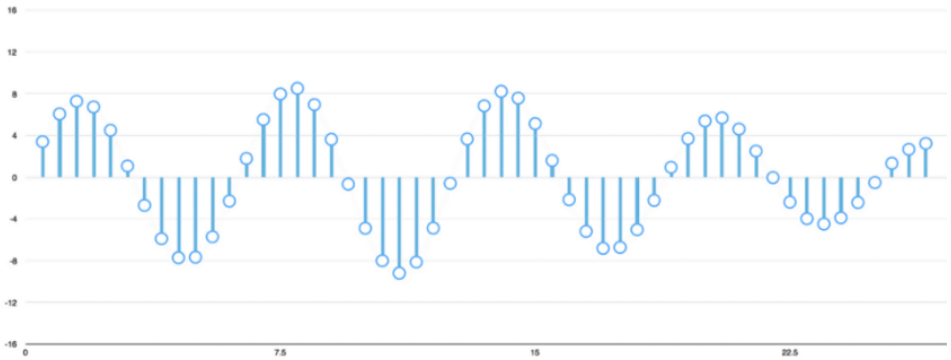

FIGURA 4.1: Onda de sonido
Fuente: *https://medium.com*

### 4.1.3. Espectro de frecuencias

Consiste en la distribución de amplitudes para un valor dado de frecuencia de un fenómeno ondulatorio en este caso ondas sonoras.

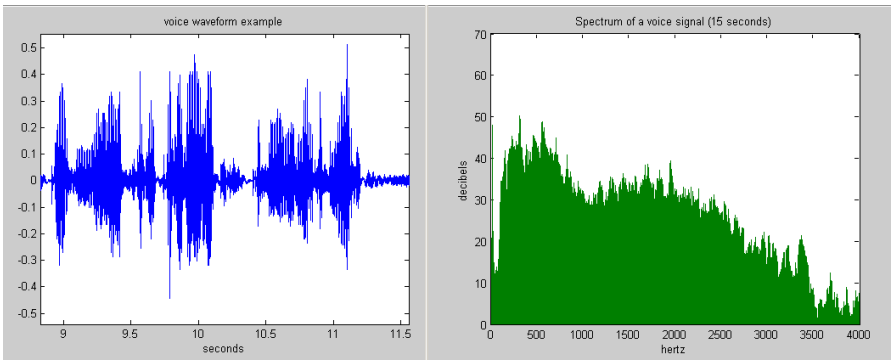


FIGURA 4.2: señal de voz y su espectro asociado
Fuente: *https://www.finaltest.com.mx*

### 4.1.4. Escala Mel

Es una escala psicoacústica propuesta por Stevens, Volkman y Newman. Esta relaciona la frecuencia percibida con su frecuencia real medida además la uso de esta escala permite abstraer las características que coinciden más con lo que los seres humanos oyen.

Esta escala es muy importante debido a que la representación más usada en las señales de voz. La ecuación 4.1 muestra la ecuación para convertir a la escala Mel.

$$m = 1127.01048 \log_e(1 + f/700)) \tag{4.1}$$

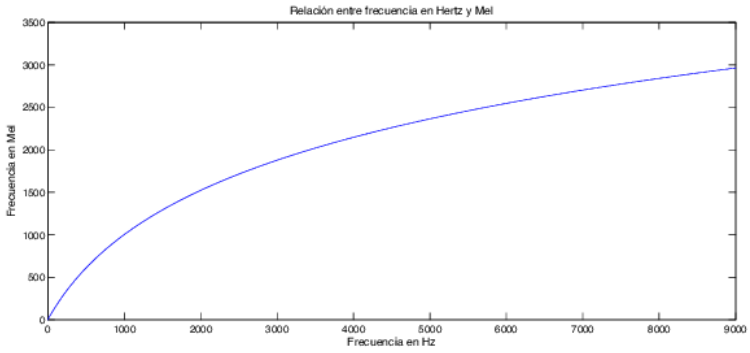


FIGURA 4.3: Relación escala mel - Frecuencia
Fuente: *https://www.researchgate.net*

### 4.1.5. Dominio de tiempo

Este dominio permite obtener la amplitud en un instante de tiempo dado existen dominio de tiempo discretos y continuos.

### 4.1.6. Dominio de frecuencia

Este dominio permite representar a nuestra frecuencia de onda como un par de amplitud y valores de la fase. Para señales periódicas se relaciona con series de fourier y para señales no periódicas se relaciona con la transformada de fourier.

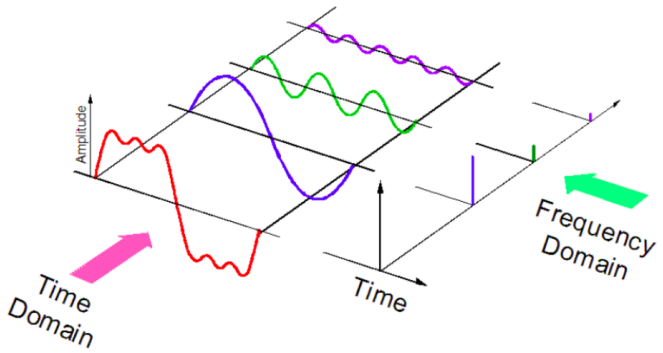


FIGURA 4.4: Dominios de tiempo y frecuencia de dominio
Fuente: *https://medium.com*

## 4.2. Proceso de extracción de características

También conocido como análisis de la señal de voz este proceso retené la información necesaria de la voz y elimina los datos redundantes y no necesarios. Este proceso es uno de los más importantes debido a que se transforma la señal en una forma adecuada para los modelos de clasificación.

### 4.2.1. Etapas del proceso de extracción de carácterísticas

El proceso de extracción de características puede ser divido en 3 etapas o subprocesos.

#### 1. Análisis Espectral

El análisis espectral consiste en descomponer algo complejo en parte o identificar propiedades de la muestra analizada. Cuando tratamos con una seña de voz esta es variable en el tiempo, por tanto podemos identificar sus características en función a la actividad espectral. El análisis espectral puede ser realizado usando los siguientes métodos:

- **Banco de filtros digitales**

  Es un sistema que divide una señal de entrada $x(n)$ en un conjunto de señales $x_1, x_2, ..$ donde cada una corresponde a una región del espectro de $x(n)$.

- **Transformada de fourier**

  Dada una señal aperiódica discreta en el tiempo $x(n)$ definimos la transformada discreta de Fourier como:

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp^{-j2\pi k \frac{n}{N}} \; k = 0, ..., N-1 \tag{4.2}$$

  Esta transformada es calculada por la transformada rápida de Fourier, esta tiene el objetivo de encontrar los componente de la frecuencia en una señal ruidosa dentro de su dominio de frecuencia.

- **Predicción Lineal**

  La predicción Lineal o LP, es una técnica para modelar el espectro de la señal de voz por medio de los espectros de todos los polos. "El método permite la conformación espectral arbitraria en el dominio de la frecuencia y el modelado de espectros continuos y discretos. "[13]

**2. Transformación de parámetros**

Los parámetros son generados mediante operaciones fundamentales como: diferenciación y concatenación. La salida será un vector de parámetros con estimaciones brutas de la señal.

**3. Modelo estadístico**

En este paso se asume que de los parámetros de las señales son producidas por procesos aleatorios multivariados. En este proceso lo único que conocemos es la entrada(parámetros) y la salida. Los vectores de los parámetros son conocidos como observaciones de señales y es usado para determinar si son parte de una palabra, frase o representan un ruido.
En la figura 4.5 mostramos el proceso que siguen algunos algoritmos de extracción de características.

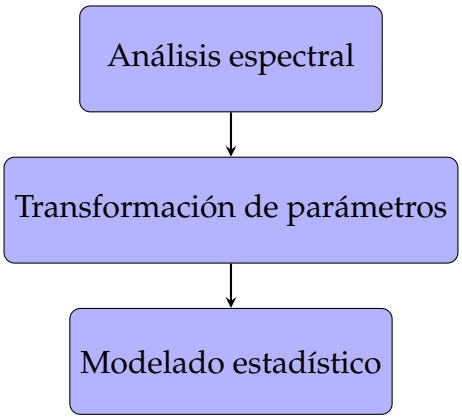


FIGURA 4.5: Etapas de extracción de características
Fuente: *Fuente Propia*

### 4.2.2. Algoritmos

A continuación describiremos algunos algoritmos que se usan para realizar este proceso, en la siguiente sección trateremos el algoritmo MFCC el cual es usado en este seminario

#### Real Cepstral Coefficient (RCC)

Este algoritmo convierte la señal del dominio de tiempo a dominio de frecuencia aplicando la transformada rápida de fourier a cada cuadro(frame). Luego se aplica le logaritmo y la transforma de fourier inversa.

$$RealCepstrum = IFFT(log(FFT(s(n)))) \tag{4.3}$$

Donde:

- IFFT: Transformada de fourier inversa
- FFT: transformada rápida de fourier.
- s(n): señal de voz.

**Codificación lineal predictiva**

Este método también conocido como LCP considera que la señales de voz pueden ser expresadas como la combinación lineal de las anteriores, entonces podemos describir a la señal de voz utilizando los coeficientes de esta combinación lineal. Este método ha sido utilizado para el procesamiento de la señal de voz debido modela el comportamiento del tracto vocal.
En el ecuación 4.3 vemos como se expresa la señal de voz $s$ usando las p anteriores señales de voz y los $a_k$ representan *coeficientes del LCP.*

$$s(n) = \sum_{k=1}^{p} a_k s(n-k) + e(n) \tag{4.4}$$

## 4.3. Mel Frequency Cepstral Coefficients (MFCC)

Es el algoritmo más usado en aplicaciones de reconocimiento de voz, esta basado en el sistema auditivo humano, el cual es sensible a 2 características dinámicas y estáticas. El MFCC se concentra principalmente en las características estáticas. A continuación describiremos el proceso de la MFCC.

### 4.3.1. Pre-emphasis

En este paso, la señal pasa a través de un filtro el cual enfatiza las altas frecuencia. Este proceso aumentará la potencia de la señal con mayor frecuencia. El pre-emphasis es necesario debido a que las señales con alta frecuencia tienen una amplitud menor y por medio este se puede compensar ese resultado. En la ecuación 4.4 vemos la aplicación de un filtro $s2$ para compensar esto donde $a$ es un valor entre $0.9$ y $1$.

$$s2(n) = s(n) - a \times (n-1) \tag{4.5}$$

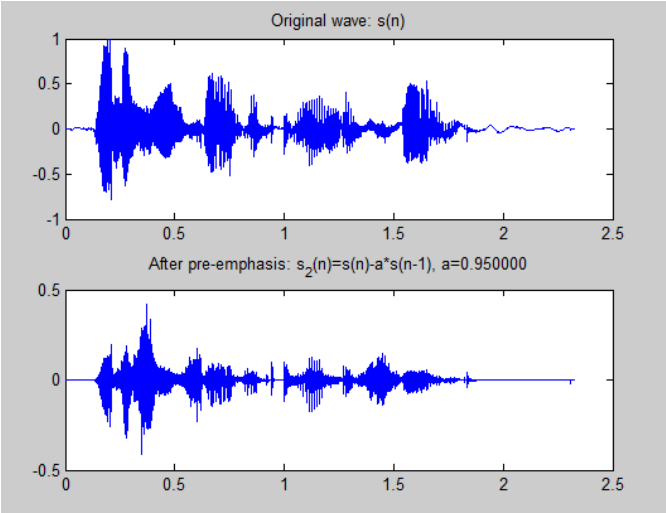


FIGURA 4.6: Aplicación del filtro Mel
Fuente: *https://www.researchgate.net*

### 4.3.2. Framing

Se realiza la segmentación de la señal de voz digital, es decir la señal es divida en pequeños intervalos(*frames*), estos generalmente son del rango de 20ms a 30ms. Un estándar usado es 25ms. Por ejemplo si la longitud de nuestro frame esta expresado en puntos de muestra 320 y nuestra frecuencia es 16kHz la duración de cada frame en segundo será $320/16000 = 0.02s$ o $20ms$ estos frames pueden solaparse. En la figura 4.7 mostramos los frame de una porción de la señal.

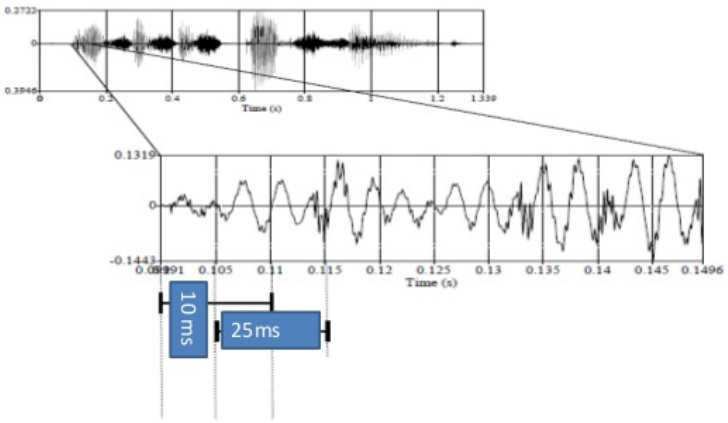


FIGURA 4.7: Framing de una señal
Fuente: *https://www.slideshare.net*

### 4.3.3. Hamming windowing

Esta proceso aplica una función usada para suavizar la señal y preparar para la aplicación de la DFT. Hamming windowing es usado en el análisis espectral de señales de voz debido a que en este método el espectro cae rápidamente por lo tanto la frecuencia resultante es mejor.

$$\begin{aligned} Y(n) &= X(n) \times W(n) \\ W(n) &= (1-\alpha) - \alpha\cos(\frac{2\pi n}{N-1}) 0 \leq n \leq N \end{aligned} \tag{4.6}$$

- $Y(n)$: Señal de salida
- $W(n)$: Hamming window
- $X(n)$: Señal de entrada
- $N$: Número de ejemplo en cada frame

En la Figura 4.8 mostramos los distintas curvas de hamming windowing.

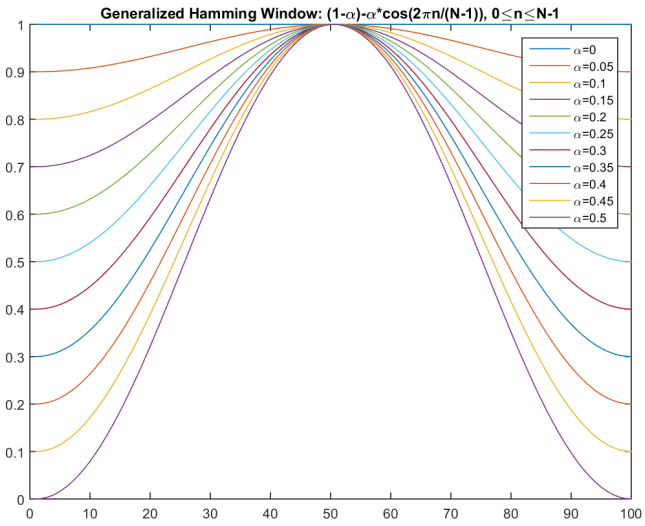


FIGURA 4.8: hamming generalizado
Fuente: *https://www.slideshare.net*

### 4.3.4. Transformada rápida de Fourier (FFT)

El análisis espectral establece que a distintos timbres en la señal de voz le corresponde una distribución de energía diferente en la señal. El FFT tiene como objetivo obtener la magnitud de la frecuencia en cada frame. Cuando se realiza FTT en un frame se asume que la señal es periódica y continua, al aplicar FFT se transforma las muestras N del dominio de tiempo al dominio de frecuencia.

$$Y(w) = FFT(H(t) * X(t)) = H(w) * X(w) \tag{4.7}$$

- $Y(w)$: Transformada de Fourier de Y(t)
- $H(w)$: Transformada de Fourier de H(t)
- $X(w)$: Transformada de Fourier de X(t)

### 4.3.5. Mel filter bank

Debido a que el rango en el espectro de la Transformada rápida de Fourier es muy amplia y la señal de voz no sigue una escala lineal. Es necesario aplicar filtros triangulares en una escala mel estos filtros son aplicado al espectro de frecuencia y extraen la banda de frecuencias.

La ecuación 4.5 se usa para calcular Mel para una frecuencia de tiempo.

$$F(mel) = (2595 * log10(1 + f)700) \tag{4.8}$$

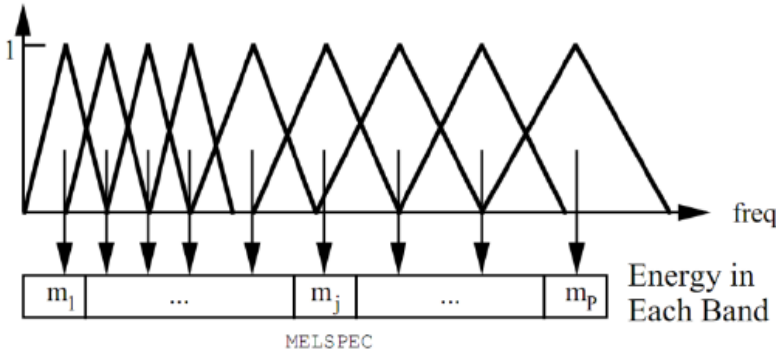


FIGURA 4.9: Aplicación del filtro Mel
Fuente: *https://www.researchgate.net*

### 4.3.6. Discrete Cosine Transform (DCT)

Este proceso se encarga de convertir el log del espectro Mel en un dominio de tiempo usando DCT. El resultado de esta conversión es llamado MFCC. El conjunto de coeficientes esa llamado vectores acústicos.

$$Energia = \sum X^2(t) \tag{4.9}$$

## 4.4. Coincidencia de patrones

A continuación se describirán algunos algoritmos usados para las tareas de las coincidencias de patrones.

#### Modelo oculto de Markov (HMM)

Es un modelo estadístico donde asumimos que nuestro sistema es un proceso de markov con parámetros desconocidos. A continuación se describirá el proceso en la figura 4.10

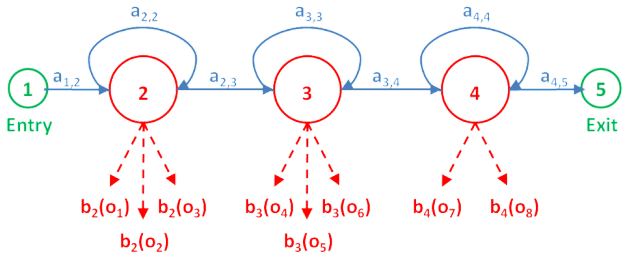


FIGURA 4.10: Esquema del HMM
Fuente: *http://gekkoquant.com*

- N: Número de estados en el HMM
- $a_{ij}$ probabilidad de transición del estado $i$ al $j$.
- $b_j(o)$ Probabilidad de generar un vector de característica en estado j.

**Máquinas de soporte Vectorial(SVM)**

Las SVM usan el concepto de planos de decisión. Un plano de decisión separa un conjunto de objetos que tienes diferentes etiquetas de clases. Las SVM no están restringidas a los problemas lineales debido a las *funciones Kernel.*

**Funciones Kernels**

Las SVM pueden tener distintos tipos de kernels que tienen como objetivo tomar la data y transformarla. Algunas funciones kernels conocidas:

- Lineal: $\ker(x_i, x_j) = x_i \cdot x_j$
- Polinomial: $\ker(x_i, x_j) = (\gamma x_i \cdot x_j + C)^d$
- Radial: $\ker(x_i, x_j) = e^{(\gamma|x_i - x_j|)}$
- Sigmoidal: $\ker(x_i, x_j) = \tanh(\gamma x_i \cdot x_j + C)$

En la figura 4.11 muestra el efecto de las funciones kernels en un conjunto de datos para que este sea linealmente separable sin necesidad de construir curvas complejas.

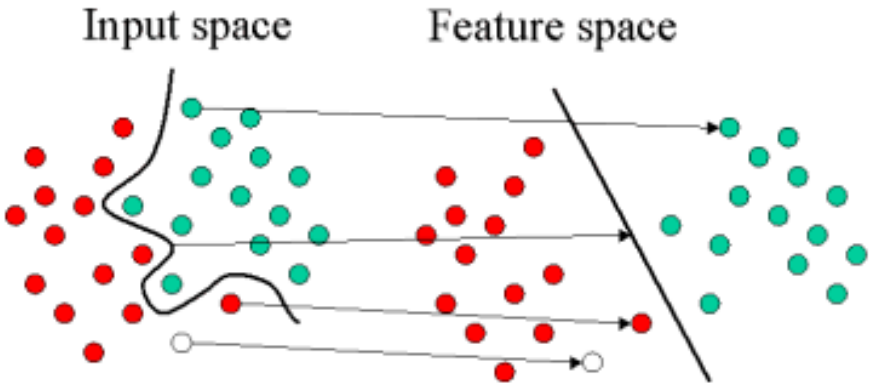


FIGURA 4.11: transformación con la función kernel
Fuente: *www.statsoft.com*

**Alineamiento temporal dinámico (DTW)**

Es un método de programación dinámica que busca encontrar una alineación óptima entre 2 series de tiempo. El algoritmo trata de buscar deformaciones entre las series de tiempo y determina la similitudes entre ambas series. Esto nos permite tener una robustez en caso de grupos fonéticos en el audio tenga una duración más prolongada.
En la figura 4.12 se muestra 2 series de tiempo donde se trazan las coincidencias entre ambas series.

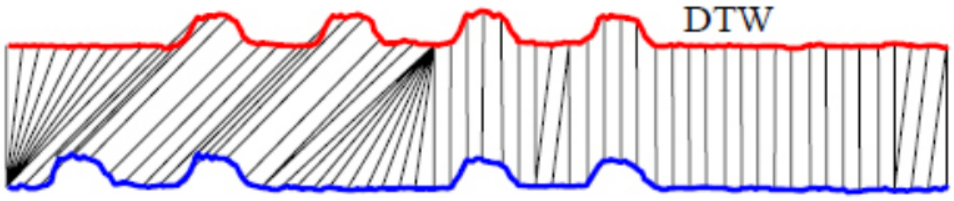


FIGURA 4.12: DTW
Fuente: *https://lemonzi.files.wordpress.com*

**Redes Neuronales**

Las redes neuronales descritas en el capítulo 3, luego de su entrenamiento serán capaces de reconocer los patrones de las señales de voz. Además las redes recurrentes nos permiten trabajar con las características dinámicas de la voz. Nuestra red será alimentadas con los coeficientes MFCC. Los esquemas de estas redes pueden ser vistas en el Apéndice A.1

# Capítulo 5

# Implementación

En este capítulo describiremos detalles de la implementación de nuestro modelo de reconocimiento de números en español. Comenzaremos describiendo el proceso de obtención y tratamiento de datos. Finalmente, mostraremos el esquema de nuestra red neuronal.

## 5.1. Datos

Para las tareas de reconocimiento de voz es importante encontrar un conjunto de datos adecuado, este servirá para alimentar nuestra red neuronal. Proyectos como *VoxForge*[14] y *Common Voice*[15] son iniciativas open source que buscan formar un gran corpus de data para distintos idiomas. Sin embargo en el lenguaje español los datos aún están en proceso de recolección. Por lo cual obtener los datos para una tarea específica resulta complicado.

### 5.1.1. Recolección de datos

Debido al problema anterior fue necesario realizar una propia recolección de datos. Durante este proceso se obtuvo la información de 12 hablantes. Los cuales proporcionaron grabaciones de voz de la pronunciación de los dígitos $0, 1, ..., 8, 9$.

**Conjuntos de datos**

Nuestro conjunto de datos consta de un total de 300 audios los cuales fueron dividos en 2 conjuntos:

- **Training**: Este conjunto es usado para entrenar nuestro modelo. Posee 24 muestra de cada número.
- **Test**: Este conjunto es usar para verificar la precisión de modelo. Posee 11 muestras de cada número.

La división del conjunto de datos es mostrada en el cuadro 5.1.

| Training | Test |
|---|---|
| 240 | 110 |

CUADRO 5.1: División del conjunto de datos

Debido a las diferencias de entre las voces de entre personas de diferentes sexos fue necesario tener una variedad de hablantes entre mujeres y varones con el objetivo de construir un modelo más robusto. La distribución de hablantes se muestra en cuadro 5.2.

| Mujeres | Varones |
|---|---|
| 5 | 7 |

CUADRO 5.2: Distribución de hablantes del conjunto de datos por sexo

La distribución de acuerdo a la cantidad de audios con Voces Femeninas(VF) y Voces Masculinas(VM) se muestran en el cuadro 5.3.

| conjunto de datos | | | |
|---|---|---|---|
| Training | | Test | |
| VF | VM | VF | VM |
| 90 | 150 | 40 | 70 |

CUADRO 5.3: División del conjunto en base a la cantidad de audios con voces femeninas y masculinas

### 5.1.2. Tratamiento de los datos

#### Conversión de formato

Los audios proporcionados por algunos hablantes se encontraban en un formato ogg, el cual tuvo que ser transformado al formato wav para su procesamiento esto debido a que ogg es más formato usado en teléfonos moviles mientras que el formato wav es más estándar. En el Apéndice A.2.1 podemos observar el código usado.

### 5.1.3. Obtención de coeficientes cepstrales

Nuestro modelo será alimentado con audios pero antes se necesita obtener la información más importante y eliminar el ruido de fondo. Para esto utilizamos el algoritmo de extracción de características MFCC (Ver Cap. 4.3).
Para estas tareas hacemos uso de las librerías: *librosa, sklearn y numpy*.
Estas son usadas en el código A.2.2 donde se obtiene 13 coeficientes cepstrales de una muestra.

### 5.1.4. One Hot encoding

Para entrenar nuestro modelo necesitamos representar nuestras clases o variables categóricas de manera numérica la cual es más útil para tareas de clasificación y regresión. Una forma de lograr esto es mediante el uso de One Hot enconding. En la figura 5.1 podemos representar las clases red, green y blue, en vectores únicos de dimensión 3.

| color | color_red | color_blue | color_green |
|---|---|---|---|
| red | 1 | 0 | 0 |
| green | 0 | 0 | 1 |
| blue | 0 | 1 | 0 |
| red | 1 | 0 | 0 |

FIGURA 5.1: Ejemplo One hot encoding
Fuente: *https://www.machinelearningplus.com/*

En nuestra implementación fue necesario encontrar una representación para nuestras 10 clases de números. El código puede ser encontrado en el Apéndice A.2.3 .

## 5.2. Modelo de la red neuronal

En esta sección veremos algunos modelos que fueron entrenados para la tarea de clasificación de audios de números estos serán probados con un solo tipo de redes RNN(ver 3.4) y en especial su derivado LSTM(ver 3.4.4).

### 5.2.1. RNN

Definimos un red neuronal con una sola capa RNN y una capa densamente conectada( ver Figura A.1.1). Esta red será probada con distintas cantidades de estados ocultos. Podemos ver el código de esta red en A.2.4

### 5.2.2. LSTM

Las redes LSTM son tipo de redes recurrentes especial que resuelven las dificultades de las RNN simples. Dentro de las redes LSTM usaremos el esquema sin Dropout(A.1.2) y con Dropout(A.1.3). Los códigos de estos modelos son encontrados en los códigos A.2.5 y A.2.6.

### 5.2.3. 2 capas LSTM y 2 Dropout

Adicionalmente se diseño una red con más de 2 capas LSTM para realizar las tareas de reconocimiento de voz. El esquema de esta red puede ser encontrado en A.1.4 y la implementación puede ser visto en el código A.2.7 del Apéndice.

**Categorical Crossentropy**

Para entrenamiento de nuestro modelos usaremos categorical crossentropy como nuestra función de perdida.

$$CCE = -\frac{1}{N}\sum_{i=0}^{N}\sum_{j=0}^{J} y_j \log(\hat{y_j}) + (1 - y_j)\log(1 - \hat{y_j}) \tag{5.1}$$

- N: cantidad del conjunto training
- J: cantidad de clases

**Adam**

Como algoritmo de optimización utilizaremos Adam, calcula una tasa de aprendizaje adaptativo para cada parámetro y es usado para optimizar la gradiente de descenso. En la ecuación 5.2 mostramos el cálculo del promedio de decaimiento de las gradientes pasadas $m_t$ y el cuadrado de las gradientes pasadas $v_t$

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \end{aligned} \tag{5.2}$$

- $m_t$ : Primer momento (media)
- $v_t$ : Segundo momento de la gradiente
- $\beta_1$ : Taza de decaimiento del primer momento.
- $\beta_2$ : Taza de decaimiento del segundo momento.

En la ecuación 5.3 mostramos la forma de calcular estimado de la primer y segundo momento.

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \end{aligned} \tag{5.3}$$

- $\hat{m}_t$ : Estimación del Primer momento (media)
- $\hat{v}_t$ : Estimación del Segundo momento de la gradiente.

La ecuación 5.4 muestra la regla de actualización en Adam. Se utiliza el $\epsilon$ para prevenir una división por cero.

$$\theta_{t+1} = \theta_{t+1} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{5.4}$$

En el código 5.7 mostramos el uso del función de perdida y el optimizador como parte de la configuración de nuestro modelo.

# Capítulo 6

# Resultados

En este capítulo se discutirán los resultados obtenidos durante el entrenamiento de nuestra red neuronal. Para estas pruebas usamos el conjunto de datos descrito en el capítulo. Estos resultados fueron obtenidos usando una computadora con tarjeta gráfica NVIDIA GTX950M con un total de 640 núcleos y una memoria de 4GB. Estos resultados fueron obtenidos utilizando distintos esquema de redes neuronales(ver Apéndice A.1)

### 6.0.1. Precisión

En el cuadro 6.1 podemos ver los resultados de probar distintos variantes de RNN las gráficas de la precisión en cada epochs se pueden ver en el Apéndice A.3.1.
Los resultados presentados a continuación fueron realizados con distintas cantidades de estados ocultos $h_t$ .

#### 64 estados ocultos

Para 64 estados ocultos observamos en las figuras del Apéndice A.3.1 mostrarán el comportamiento del training para 400 epochs. Notamos RNN oscila constantemente(ver A.5) y no supera el 16 % de presión. Al aplicar redes LSTM se alcanzan precisiones más altas pero a una cantidad de epochs determinada alrededor del epochs 200 la precisión del test se mantiene para una *LSTM simple* y *LSTM con Dropout 0.5* . Al aplicar un Dropout de 0.8 se logra superar más el problema sin acercarse más al overfitting.

El cuadro 6.1 muestra los resultados de precisión de los conjuntos test y training durante el entrenamiento luego de 300 epochs. Notamos que los mejores resultados sin llegar a overfitting son de LSTM simple y LSTM dropout (0.8).

| Modelo | Precisión test( %) | Precisión Training ( %) |
|---|---|---|
| RNN | 10.90 | 14.58 |
| LSTM | 40.00 | 73.75 |
| LSTM Dropout(0.5) MFCC | 31.81 | 98.33 |
| LSTM Dropout(0.8) MFCC | 0.55 | 77.91 |

CUADRO 6.1: Precisión de modelos para 300 iteraciones y 64 estados ocultos

**128 estados ocultos**

En el Apéndice A.3.1 encontramos las gráficas de precisión para nuestros modelos con 128 estados ocultos. Notamos que el LSTM simple llega al overfitting luego de 250 epochs, mientras que los modelos con dropout obtiene una precisión de alrededor de 40 % a 60 % durante las primeras ejecuciones. Con estos estados ocultos el LSTM resulto ser un mejor modelo durante los primeros 200 epochs(Ver figura A.10 del Apéndice ).
El cuadro 6.2 muestra los resultados luego de 300 epochs donde los dropout obtienen mejor resultado que el LSTM y RNN pero aún se acerca al overfitting.

| Modelo | Precisión test( %) | Precisión Training ( %) |
|---|---|---|
| RNN | 9.09 | 17.50 |
| LSTM | 75.45 | 99.16 |
| LSTM Dropout(0.5) MFCC | 60.00 | 93.74 |
| LSTM Dropout(0.8) MFCC | 67.27 | 93.74 |

CUADRO 6.2: Precisión de modelos para 300 iteraciones y 128 estados ocultos

**256 estados ocultos**

Los siguientes resultados son tomados de las gráficas en A.3.1. Notamos que con 256 estados ocultos rápidamente se obtuvieron resultados prometedores para una LSTM simple pero luego de los 200 epochs comenzó a caer en overfitting. Los modelos con dropout retrasaron el overfitting, este fue más notorio a partir de los 250 epochs(Ver figuras de A.3.1).
El cuadro 6.3 muestra la precisión para 300 epochs.

| Modelo | Precisión test( %) | Precisión Training ( %) |
|---|---|---|
| RNN MFCC | 9.09 | 17.50 |
| LSTM MFCC | 78.18 | 99.58 |
| LSTM Dropout(0.5) MFCC | 76.36 | 96.66 |
| LSTM Dropout(0.8) MFCC | 82.57 | 92.91 |

CUADRO 6.3: Precisión de modelos para 300 iteraciones y 256 estados ocultos

### 6.0.2. Errores

El cuadro 6.2 representa los errores que se resultaron durante el entrenamiento de nuestros modelos con distintas cantidades de estados ocultos.

**64 estados ocultos**

Los errores obtenidos fueron obtenidos en base los resultados de A.3.2. Se muestra en el cuadro 6.4 que los errores para 300 epochs en los modelos RNN y LSTM Dropout 0.5 tuvieron los errores más altos. Mientras que el modelo más estable fue LSTM con Dropout 0.8.

| Modelo | Error test | Error Training |
|---|---|---|
| RNN MFCC | 2.378 | 2.235 |
| LSTM MFCC | 1.960 | 0.598 |
| LSTM Dropout(0.5) MFCC | 3.074 | 0.073 |
| LSTM Dropout(0.8) MFCC | 1.419 | 0.581 |

CUADRO 6.4: Errores de los conjuntos test y training para 300 iteraciones y 64 estados ocultos

**128 estados ocultos**

Para 300 epochs todos lo modelos caen en overfitting(ver figuras de A.3.2) para el training. Sin embargo podemos observar en el A.3.2 que el LSTM simple resulta tener menor error durante las primeras 200 epochs(Ver figura A.21)

| Modelo | Error test | Error Training |
|---|---|---|
| RNN MFCC | 2.414 | 2.240 |
| LSTM MFCC | 1.489 | 0.011 |
| LSTM Dropout(0.5) MFCC | 1.647 | 0.230 |
| LSTM Dropout(0.8) MFCC | 1.329 | 0.2150 |

CUADRO 6.5: Errores de los conjuntos test y training para 400 iteraciones y 128 estados ocultos

**256 estados ocultos**

Al aumentar los estados ocultos se nota (ver figuras de A.3.2) que el LSTM simple resulta ser más estable que su derivados con Dropout en los primeros 100 epochs el error decrecen drásticamente y se mantiene cercano al error del training.

| Modelo | Error test | Error Training |
|---|---|---|
| RNN MFCC | 2.427 | 2.242 |
| LSTM MFCC | 1.3170 | 0.008 |
| LSTM Dropout(0.5) MFCC | 0.815 | 0.148 |
| LSTM Dropout(0.8) MFCC | 0.825 | 0.224 |

CUADRO 6.6: Errores de los conjuntos test y training para 300 iteraciones y 256 estados ocultos

### 6.0.3. Modelo para reconocimiento de voz

A continuación se presentan los resultados obtenidos por el modelo de 2 capas LSTM y 2 dropout(ver esquema A.1.4)

#### Precisión

. La figura A.29 nos muestra la precisión de nuesro modelo. Este es más controlado debido a los 2 dropout que evitan el overfitting. En el cuadro 6.7 observamos que obtuvimos una precisión de 65.45 %, este fue mejor que los anteriores debido a que no cayo en overfitting.

| Modelo | Precisión test( %) | Precisión Training ( %) |
|---|---|---|
| 2LSTM 2 D | 65.45 | 87.50 |

CUADRO 6.7: Precisión de modelos para 500 iteraciones y 64 estados ocultos

#### Errores

La figura A.30 vemos como el error varia a lo largo de las iteraciones. Apartir del epochs 300 el error no supera el valor de 2 y más controlado a medida que se realiza el entrenamiento.

El cuadro 6.8 nos muestra el error final luego de 500 iteraciones.

| Modelo | Error test | Error Training |
|---|---|---|
| 2LSTM 2D | 1.5330 | 0.3858 |

CUADRO 6.8: Errores de los conjuntos test y training para 500 iteraciones y 64 estados ocultos

# Capítulo 7

# Conclusiones y Trabajo Futuro

En este capítulo se describirán las conclusiones generales que se encontraron al probar y estudiar las variantes de RNN al momento de realizar una tarea de clasificación en el conjunto de datos utilizado.

Además, se propondrán algunas mejoras para que el trabajo obtenga mejores resultados en el futuro.

## 7.1. Conclusiones

- El uso 2 capas LSTM y dropout de 0.6 mejoran el rendimiento de nuestro modelo además manejar mejor el problema de overfitting
- Las redes LSTM obtuvieron mejores resultados contra el RNN simple debido a su capacidad de manejar el problema de desaparición de gradiente.
- El número de estados ocultos puede acelerar el proceso de entrenamiento pero puede generar efectos de overfitting, por lo cual se obtiene un mejor rendimiento el estado oculto de 64.
- El algoritmo MFCC permite extraer las características necesarias para el procesamiento de la señal de voz.
- Los audios con distintos tiempos incrementan el número de lapsos tiempos en que desenrolla la red lo cual impide una mejor precisión.

- La cantidad de epochs requeridos es menor en las redes con una sola capa LSTM pero estas tienden a caer en problemas de overfitting.

## 7.2. Trabajo Futuro

El propósito general de este seminario fue adquirir el conocimiento y experiencia necesarios para poder trabajar con datos del tipo dinámico además de conocer las ventajas de usar una red tipo LSTM. Las LSTM demostraron ser útiles para este tipo de tareas debido a que estas poseen cierta *memoria* y guardan la información pasada. Además que estas mejoran mejor el problema de desaparición de gradiente. El conjunto de datos recolectados fue suficiente para realizar el entrenamiento y las pruebas pero para un mejor resultado es necesario datos más variados con gran cantidad de hablantes. Los proyecto para generar un corpus de data aún están en desarrollo por lo cual no sé puede acceder a una data más adecuada.
En futuros trabajos se trataré de construir sistema de reconocimiento para más palabras además de tratar de tratar de construir un lenguaje básico usando LSTM. Este proyecto estuvo delimitada por la capacidad de la tarjeta gráfica y por su conjunto de datos reducido.

# Bibliografía


[1] Alexander Lyashevsky Nuwan Jayasena Gene Wu, Joseph L. Greathouse and Derek Chiou. Gpgpu performance and power estimation using machine learning. *21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 564–576, 2015.

[2] César Estrebou Franco Ronchetti, Facundo Quiroga and Laura Lanzarini. Handshape recognition for argentinian sign language using probsom. *JCST*, 16(1):1–5, 2010.

[3] Derek Rose Itamar Arel and Thomas Karnowski. Deep machine learning - a new frontier in artificial intelligence research [research frontier]. *JCST*, 16(1):1–5, 2010.

[4] Adam Coates Abhik Lahiri Bobby Prochnow Quoc V. Le, Jiquan Ngiam and Andrew Y. Ng. On optimization methods for deep learning. *International Conference on Machine Learning 2010*, pages 1–8, 2010.

[5] Josh Patterson and Adam Gibson. Major architectures of deep networks. In *Deep Learning A Practitioner's Approach*, pages 132–135. O'Reilly Media, 2017.

[6] Adrian Rosebrock. Lenet – convolutional neural network in python. `https://www.pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python`, Ago 2016. Accessed on 2018-06-15.

[7] Siddharth Das. Cnn architectures: Lenet, alexnet, vgg, googlenet, resnet and more .... `https://medium.com/@siddharthdas_32104/cnns-`

architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5, Nov 2017. Accessed on 2018-06-15.

[8] Jason Brownlee. A gentle introduction to backpropagation through time. https://machinelearningmastery.com/gentle-introduction-backpropagation-time/, Jun 2017. Accessed on 2018-10-11.

[9] Fernando J. Pineda. Generalization of back-propagation to recurrent neural networks. *PHYSICAL REVIEW LETTERS*, 59(19):2229–2232, 1987.

[10] Janos Sztipanovit. Dynamic backpropagation algorithm for neural network controlled resonator-bank architecture. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-11: ANALOG AND DlGlTAL SIGNAL PROCESSING*, 39(2):99–108, 1992.

[11] Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. In *Neural Computation*, pages 1735–1780, 1997.

[12] M. Schuster and K. Paliwal. Bidirectional recurrent neural networks. *IEEE TRANSACTIONS ON SIGNAL PROCESSING,*, 45(11):2673–2681, 1997.

[13] J. Makhoul. Spectral linear prediction: Properties and applications. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23:283–296, 1975.

[14] Carnegie Mellon University. Voxforge. http://www.voxforge.org/, 2006. Accessed on 2018-09-10.

[15] Mozilla. Common voice. https://voice.mozilla.org/, NOV 2017. Accessed on 2018-09-10.

# Apéndice A

# Esquemas de las red, Códigos y Resultados obtenidos

## A.1. Esquemas

### A.1.1. RNN simple

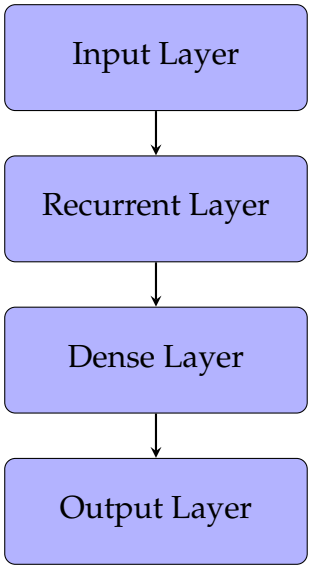


FIGURA A.1: Esquema de RNN simple
Fuente: *Fuente Propia*

### A.1.2. LSTM simple

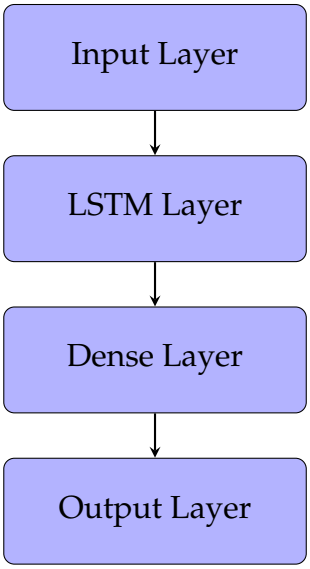


FIGURA A.2: Esquema de LSTM simple
Fuente: *Fuente Propia*

### A.1.3. LSTM con Dropout

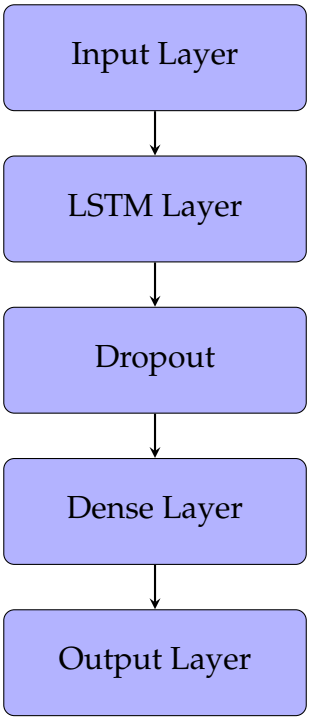


FIGURA A.3: Esquema de LSTM con dropout
Fuente: *Fuente Propia*

### A.1.4. 2 capas LSTM con 2 Dropout

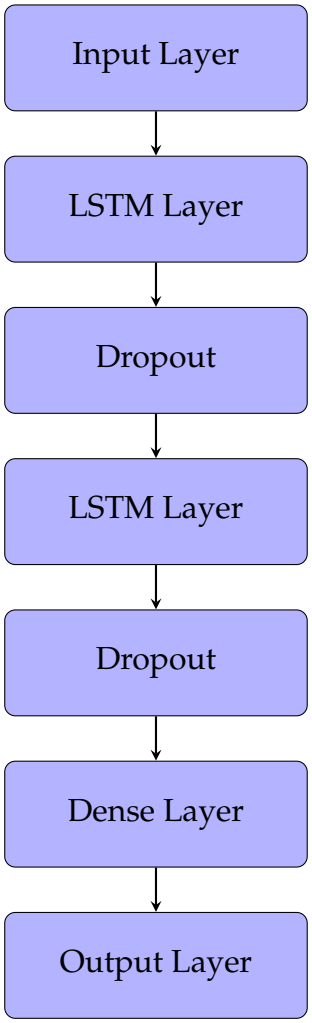


FIGURA A.4: Esquema de 2 capas LSTM con 2 dropout
Fuente: *Fuente Propia*

## A.2. Códigos

### A.2.1. Conversión de formatos

```
# itera los archivos .ogg
for i in *.ogg;
# realiza la conversion ogg a wav usando ffmpeg
do ffmpeg -i "$i" "${i%.*}.wav";
done
```

CÓDIGO A.1: Bash para conversión de formato

### A.2.2. Obtención de coeficientes MFCC

```
# cargado del audio
wave, sr = librosa.load(DIR+dir, mono=True)
# obtencion de los coeficientes
features= librosa.feature.mfcc(wave, sr,n_mfcc=20)
#padding con ceros asegurar misma dimension de salida
features=np.pad(features,((0,0),(0,160-len(features[0]))),
mode='constant',constant_values=0)
```

CÓDIGO A.2: Obtención de MFCC

### A.2.3. One hot encoding

```
from sklearn.preprocessing import OneHotEncoder
# vocabulario de clases definidas
vocabulary_words=np.array(['cero','uno','dos','tres',
'cuatro','cinco','seis','siete','ocho','nueve'])
onehot_encoder = OneHotEncoder(handle_unknown='ignore',
categories='auto')
# entrenamiento del OneHotEncoder
onehot_encoder.fit(X=vocabulary_words.reshape(-1,1))
```

CÓDIGO A.3: one hot encoding

### A.2.4. RNN

```
# establece la secuencia para empezar con el apilado de capas
model=tf.keras.Sequential()
# se apila una capa RNN simple
model.add(tf.keras.layers.SimpleRNN(128, input_shape=(time_steps,n_inp
# establece una capa densamente conectada
model.add(tf.layers.Dense(n_class, activation='softmax'))
```

CÓDIGO A.4: Modelo LSTM

### A.2.5. LSTM Simple

```
# usado para apilar las capas de la red
model=tf.keras.Sequential()
# apila una capa LSTM
model.add(tf.keras.layers.LSTM(n_units,
input_shape=(time_steps,n_inputs)))
# establece una capa densamente conectada
model.add(tf.layers.Dense(n_class, activation='softmax'))
```

CÓDIGO A.5: Modelo LSTM

### A.2.6. LSTM con Dropout

```
model=tf.keras.Sequential()
# apila una capa LSTM
model.add(tf.keras.layers.LSTM(n_units,
input_shape=(time_steps,n_inputs)))
# apila un Dropout a nuestra red para evitar overfitting
model.add(tf.keras.layers.Dropout(0.5))
# establece una capa densamente conectada
model.add(tf.layers.Dense(n_class, activation='softmax'))
```

CÓDIGO A.6: Modelo LSTM

### A.2.7. 2 Capas LSTM con 2 Dropout

```
model=tf.keras.Sequential()
#primera capa LSTM
model.add(tf.keras.layers.LSTM(n_units, input_shape=(time_steps,n
# aplicacion del Dropout
model.add(tf.keras.layers.Dropout(dropout))
# 2da capa LSTM
model.add(tf.keras.layers.LSTM(n_units, input_shape=(time_steps,n
# 2da aplicacion del Dropout
model.add(tf.keras.layers.Dropout(dropout))
model.add(tf.layers.Dense(n_class, activation='softmax'))
```

CÓDIGO A.7: Modelo 2 capas LSTM y 2 dropout

### A.2.8. Configuración del modelo

```
# configura el entrenamiento del modelo
model.compile(loss='categorical_crossentropy',
optimizer='adam',metrics=['accuracy'])
```

CÓDIGO A.8: Parámetros para el entrenamiento del modelo

### A.2.9. Entrenamiento del modelo

```
# entrenamiento del modelo pasando train and test set
history=model.fit(trainX,trainY,batch_size=batch_size,
epochs=n_epochs,validation_data=[testX,testY])
```

CÓDIGO A.9: Entrenamiento del modelo

## A.3. Pruebas con RNN, LSTM con MFCC

### A.3.1. Resultados de precisión de entrenamiento

#### 64 estados ocultos

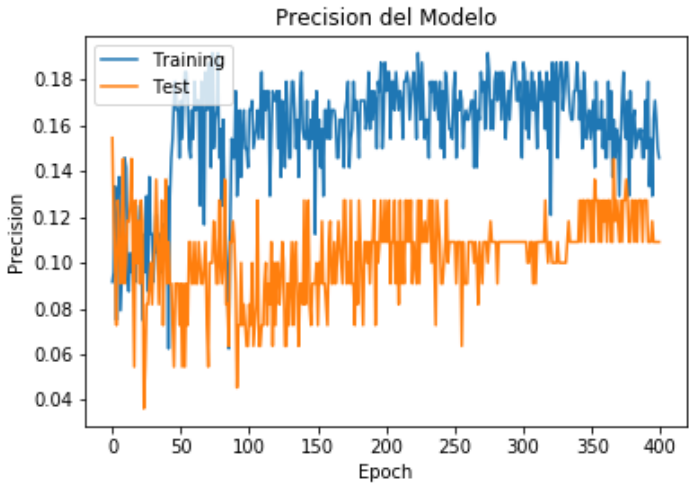


FIGURA A.5: Precisión de RNN para 64 estados ocultos
Fuente: *Fuente Propia*

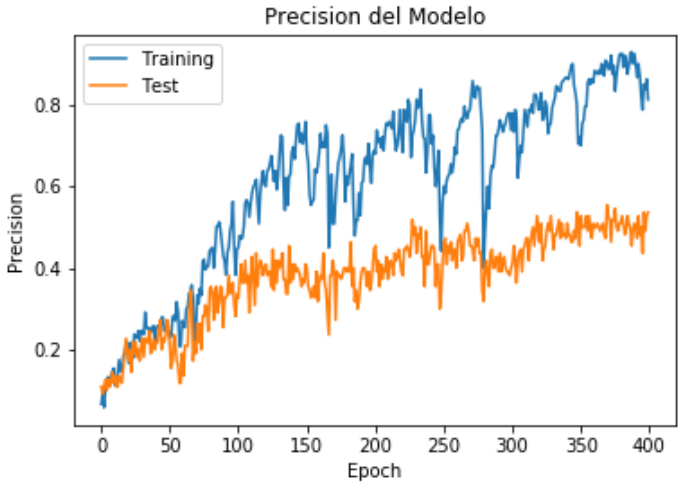


FIGURA A.6: Precisión de LSTM simple para 64 estados ocultos
Fuente: *Fuente Propia*

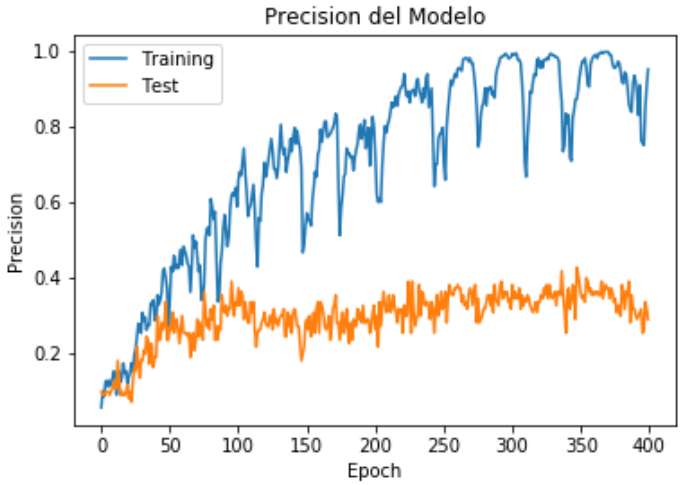


FIGURA A.7: Precisión LSTM con dropout 0.5 para 64 estados ocultos
Fuente: *Fuente Propia*

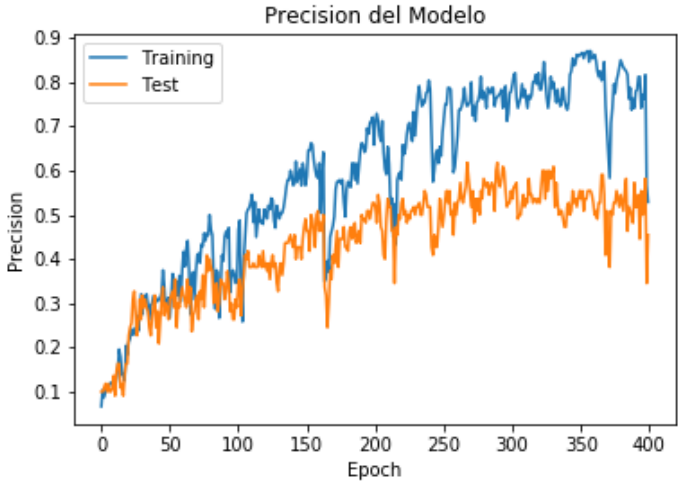


FIGURA A.8: Precisión LSTM con dropout 0.8 para 64 estados ocultos
Fuente: *Fuente Propia*

**128 estados ocultos**

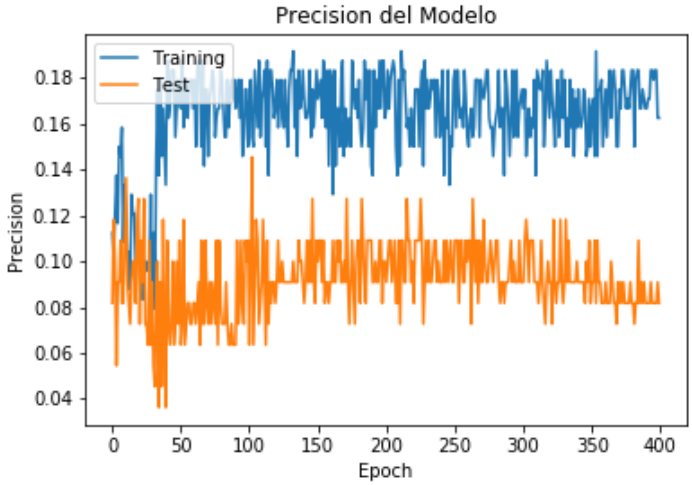


FIGURA A.9: Precisión de RNN para 128 estados ocultos
Fuente: *Fuente Propia*

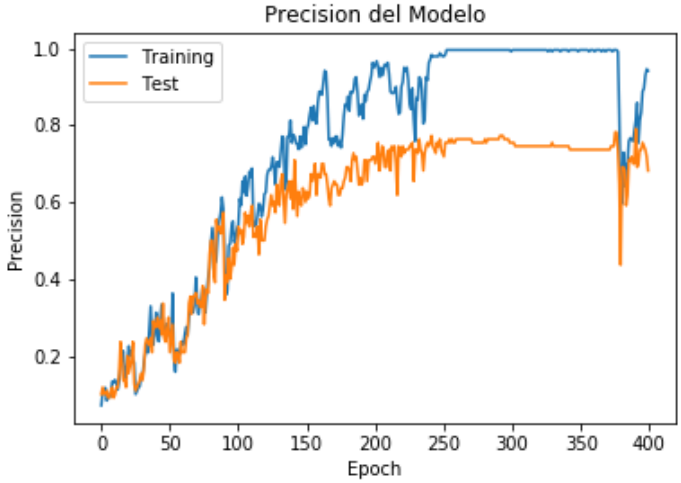


FIGURA A.10: Precisión de LSTM simple para 128 estados ocultos
Fuente: *Fuente Propia*

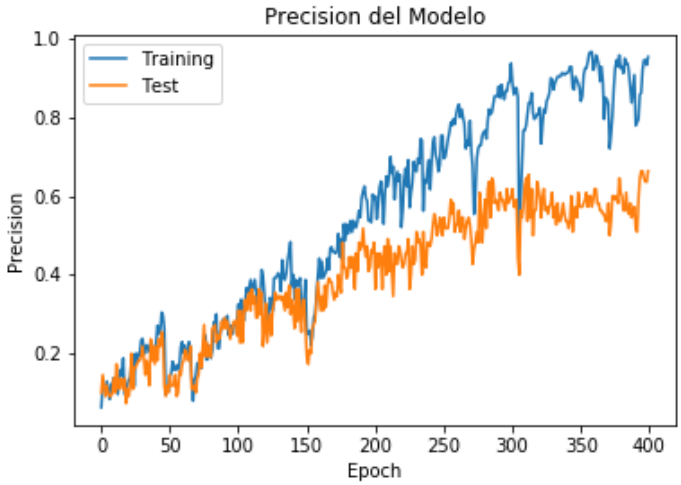


FIGURA A.11: Precisión de LSTM dropout 0.5 para 128 estados ocultos
Fuente: *Fuente Propia*

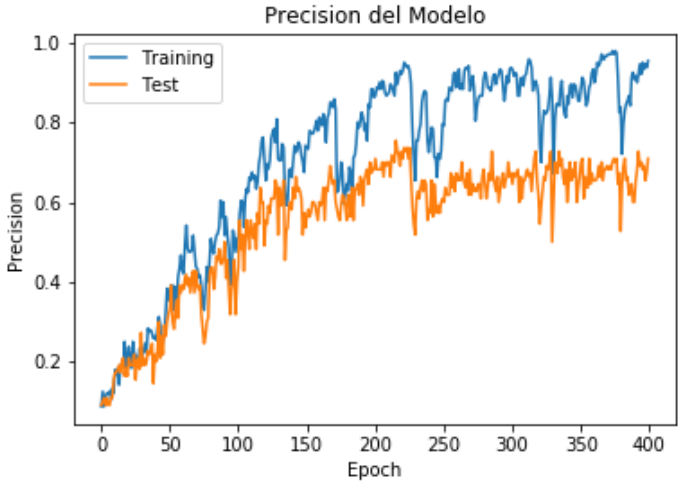


FIGURA A.12: Precisión de LSTM dropout 0.8 para 128 estados ocultos
Fuente: *Fuente Propia*

**256 estados ocultos**

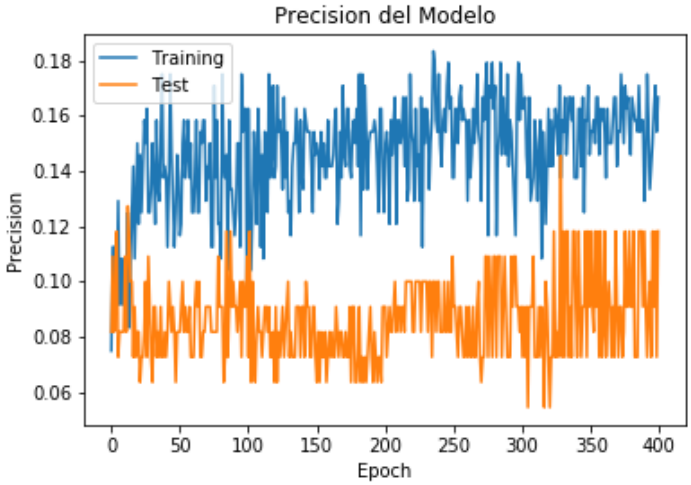


FIGURA A.13: Precisión de RNN para 256 estados ocultos
Fuente: *Fuente Propia*

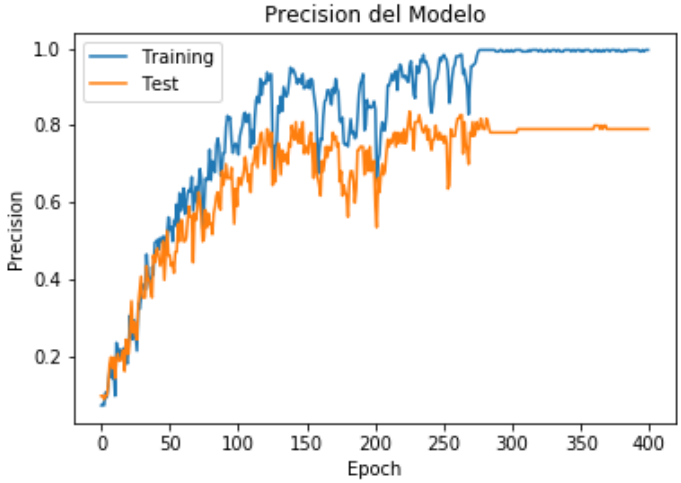


FIGURA A.14: Precisión LSTM simple para 256 estados ocultos
Fuente: *Fuente Propia*

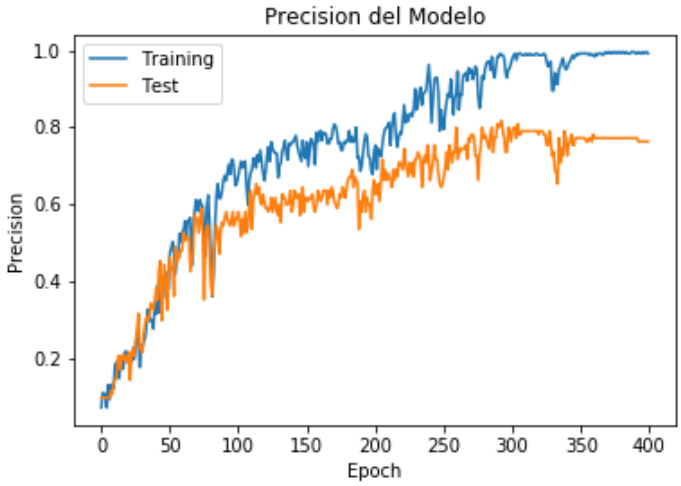


FIGURA A.15: Precisión LSTM con dropout 0.5 para 256 estados ocultos
Fuente: *Fuente Propia*

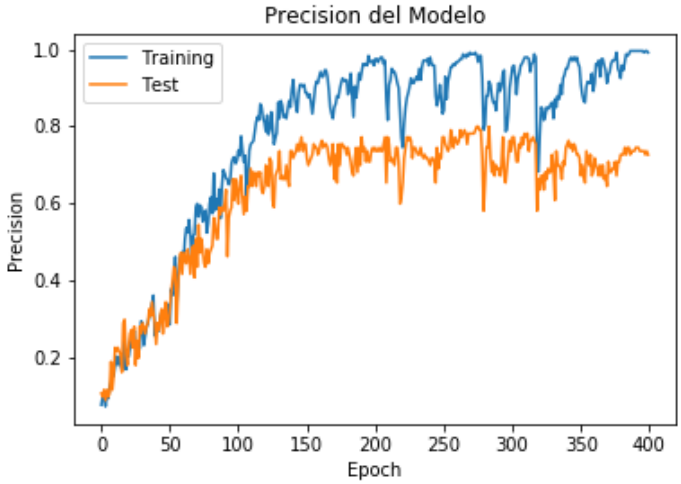


FIGURA A.16: Precisión LSTM con dropout 0.8 para 256 estados ocultos
Fuente: *Fuente Propia*

## A.3.2. Resultados de función de perdida

### 64 Estados ocultos

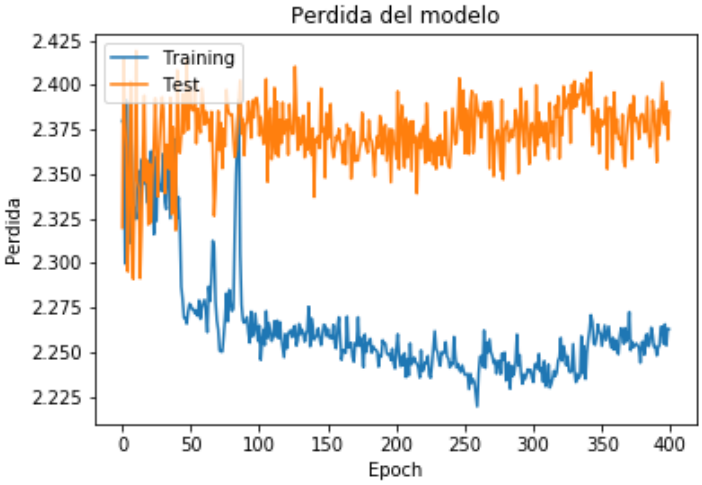


FIGURA A.17: Perdida LSTM para 64 estados ocultos
Fuente: *Fuente Propia*

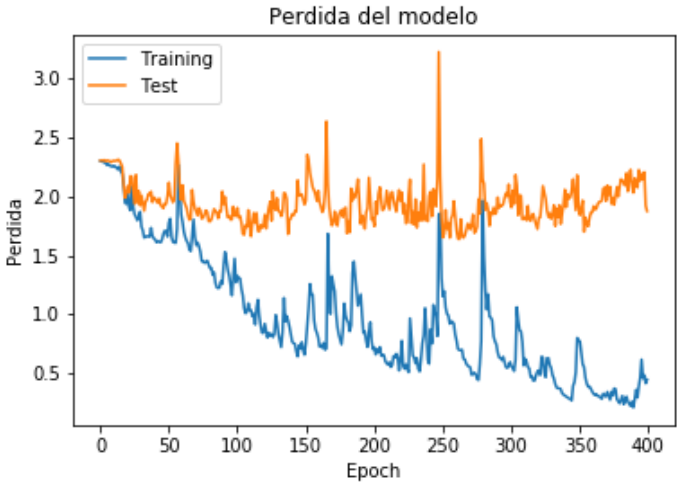


FIGURA A.18: Perdida LSTM para 64 estados ocultos
Fuente: *Fuente Propia*

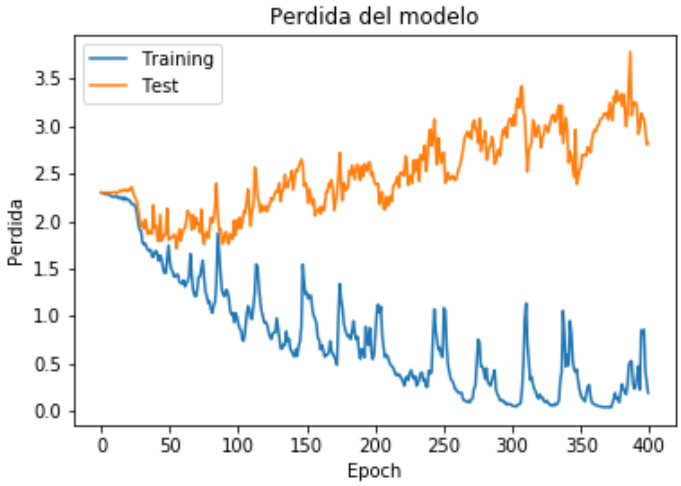


FIGURA A.19: Perdida LSTM con dropout 0.5 para 64 estados ocultos
Fuente: *Fuente Propia*

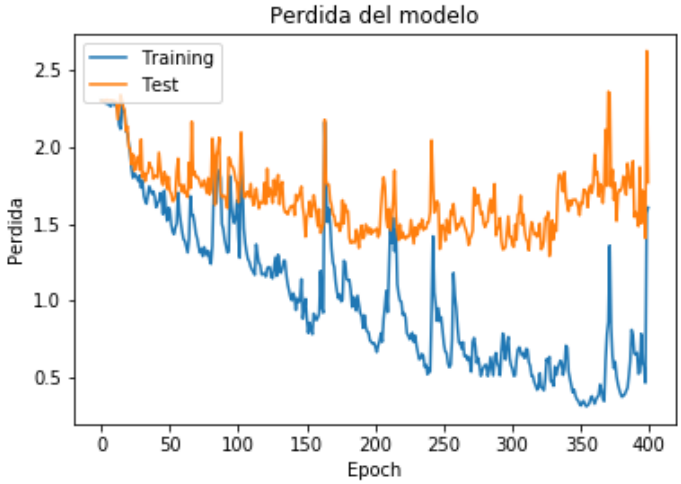


FIGURA A.20: Perdida LSTM con dropout 0.8 para 64 estados ocultos
Fuente: *Fuente Propia*

**128 estados ocultos**

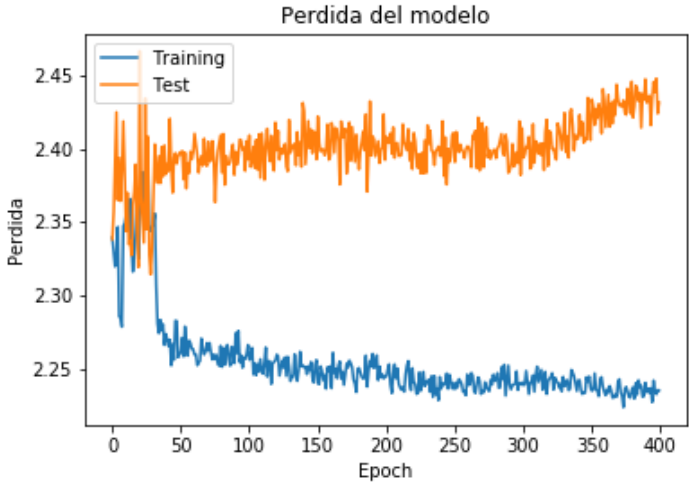


FIGURA A.21: Perdida de RNN para 128 estados ocultos
Fuente: *Fuente Propia*

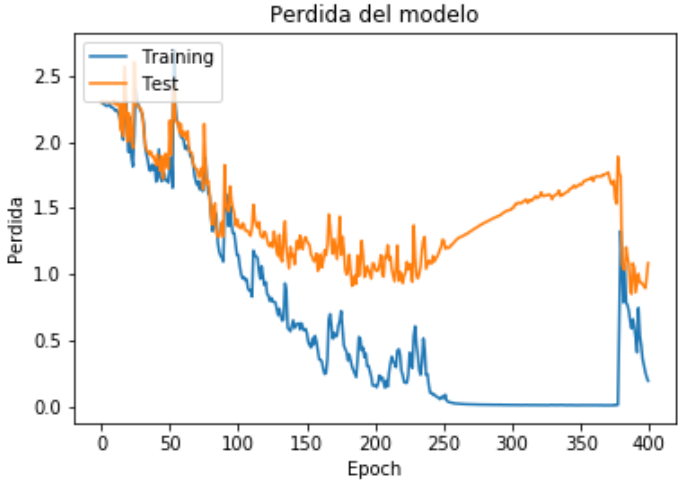


FIGURA A.22: Perdida de LSTM para 128 estados ocultos
Fuente: *Fuente Propia*

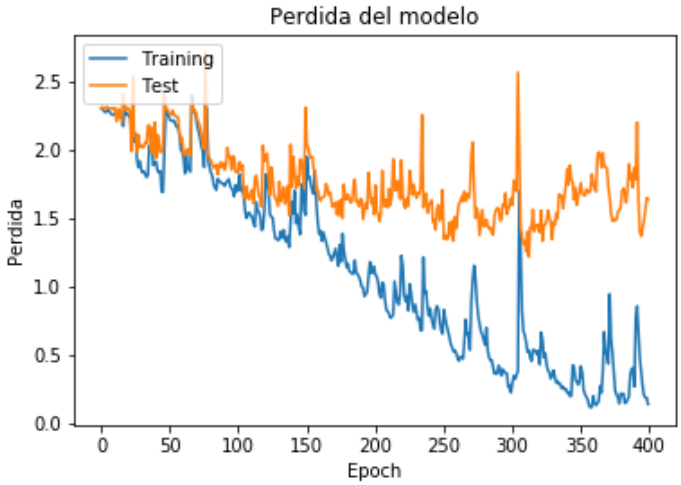


FIGURA A.23: Perdida de LSTM dropout 0.5 para 128 estados ocultos
Fuente: *Fuente Propia*

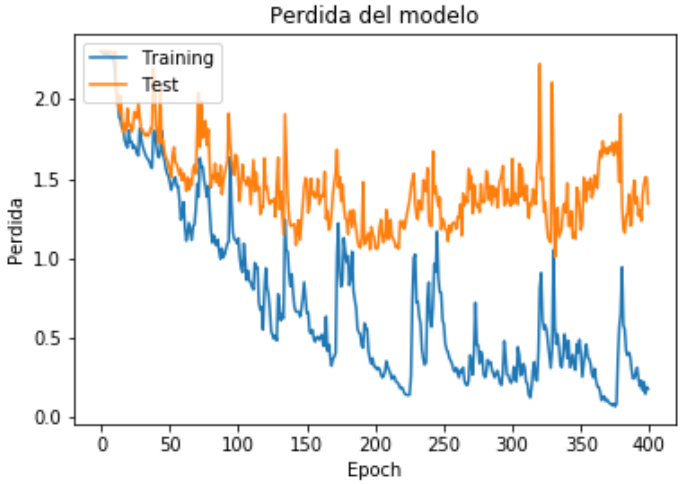


FIGURA A.24: Perdida de LSTM dropout 0.8 para 128 estados ocultos
Fuente: *Fuente Propia*

**256 estados ocultos**

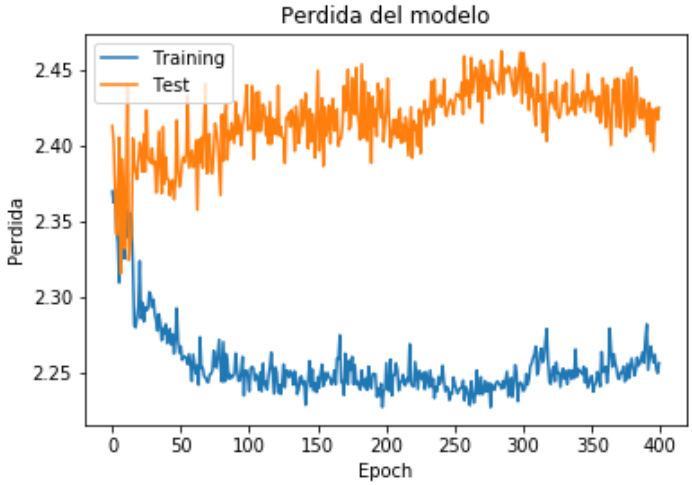


FIGURA A.25: Perdida de RNN para 256 estados ocultos
Fuente: *Fuente Propia*

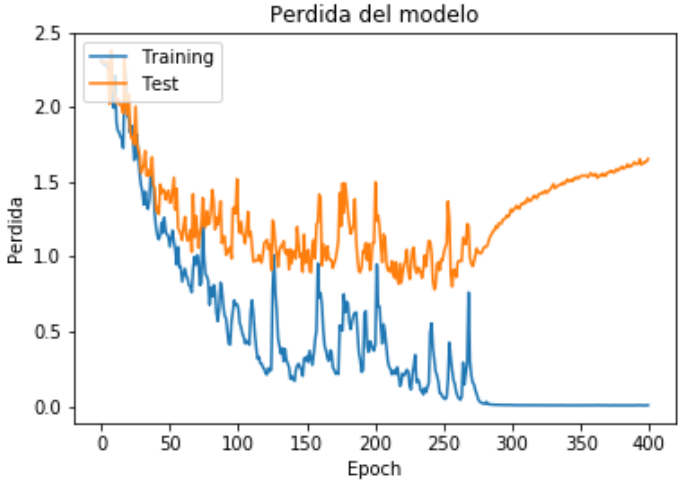


FIGURA A.26: Perdida de LSTM para 256 estados ocultos
Fuente: *Fuente Propia*

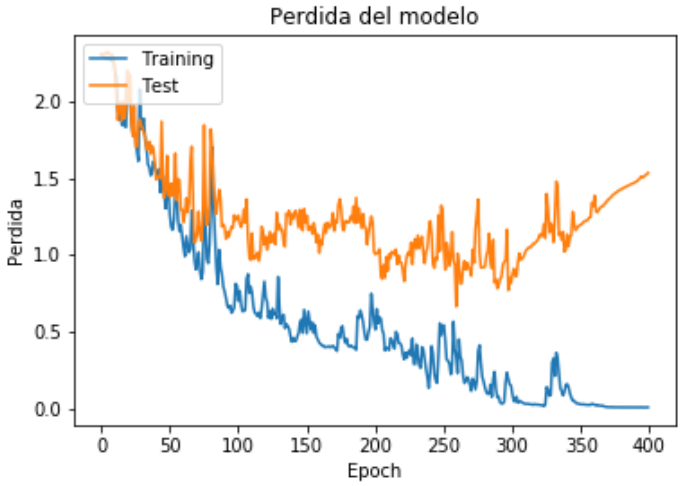


FIGURA A.27: Perdida de LSTM con Dropout 0.5 para 256 estados ocultos
Fuente: *Fuente Propia*

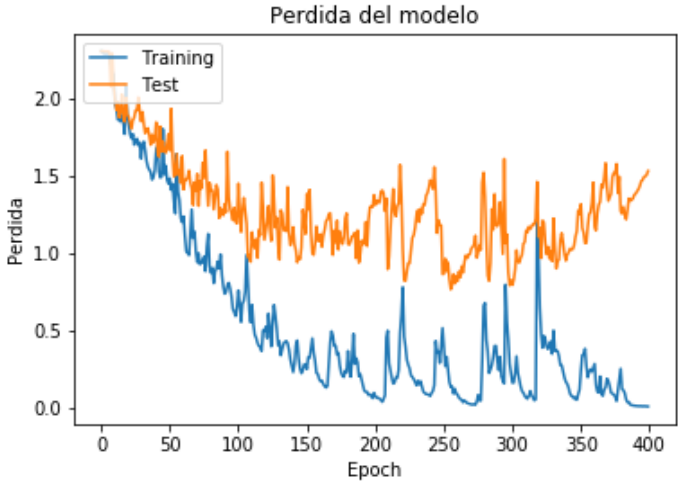


FIGURA A.28: Perdida de LSTM con Dropout 0.8 para 256 estados ocultos
Fuente: *Fuente Propia*

## A.4. Propuesta de modelo para reconocimiento de voz

### A.4.1. Precisión

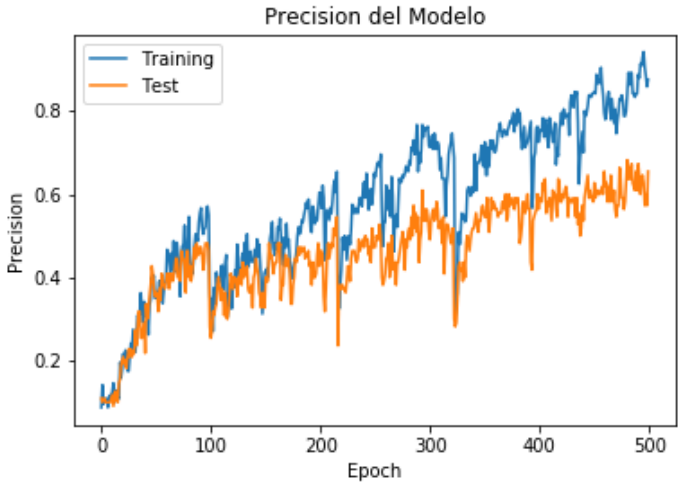


FIGURA A.29: Precisón de modelo con 2 LSTM y 2 Dropout 0.6

### A.4.2. Error

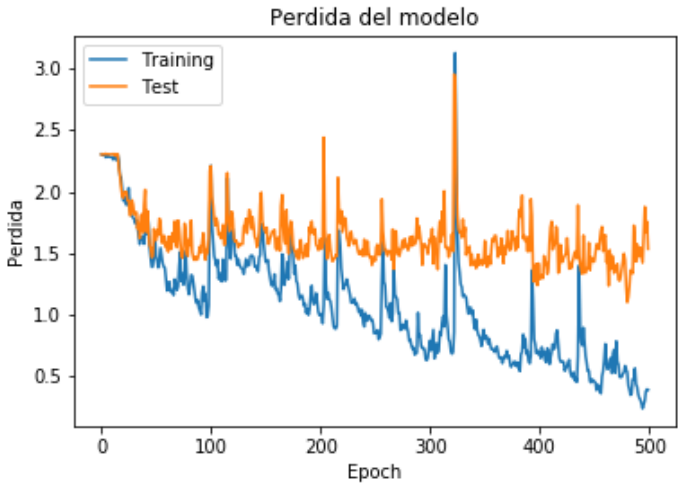


FIGURA A.30: Error de modelo con 2 LSTM y 2 Dropout 0.6