

# MODELO END-TO-END DE RECONOCIMIENTO DEL HABLA AUTOMÁTICO

Procesamiento del Habla - 86.53

# TABLE OF CONTENTS

01

Problema a  
resolver

02

Arquitectura de  
la red

03

Pre-  
procesamiento  
del input

04

Redes Neuronales  
Convolucionales

05

Redes Neuronales  
Recurrentes

06

Entrenamiento  
Criterio: CTC

07

INFERENCIA

08

Resultados  
obtenidos

09

Problemas y  
posibles  
mejoras



# O1

## PROBLEMA A RESOLVER

# RECONOCIMIENTO AUTOMÁTICO DEL HABLA

DADA UNA SEÑAL ACÚSTICA DE VOZ, SE  
DESEA ENCONTRAR LA SECUENCIA DE  
PALABRAS EMITIDAS



# ¿POR QUÉ REDES NEURONALES?

## OPTIMIZACIÓN

La optimización se realiza de manera completa con una única función objeto

## LEXICON

No es necesario obtener un diccionario de pronunciación que mapee "palabras" con "estados".  
Simplifica el modelo de decodificación

## GENERALIZACIÓN

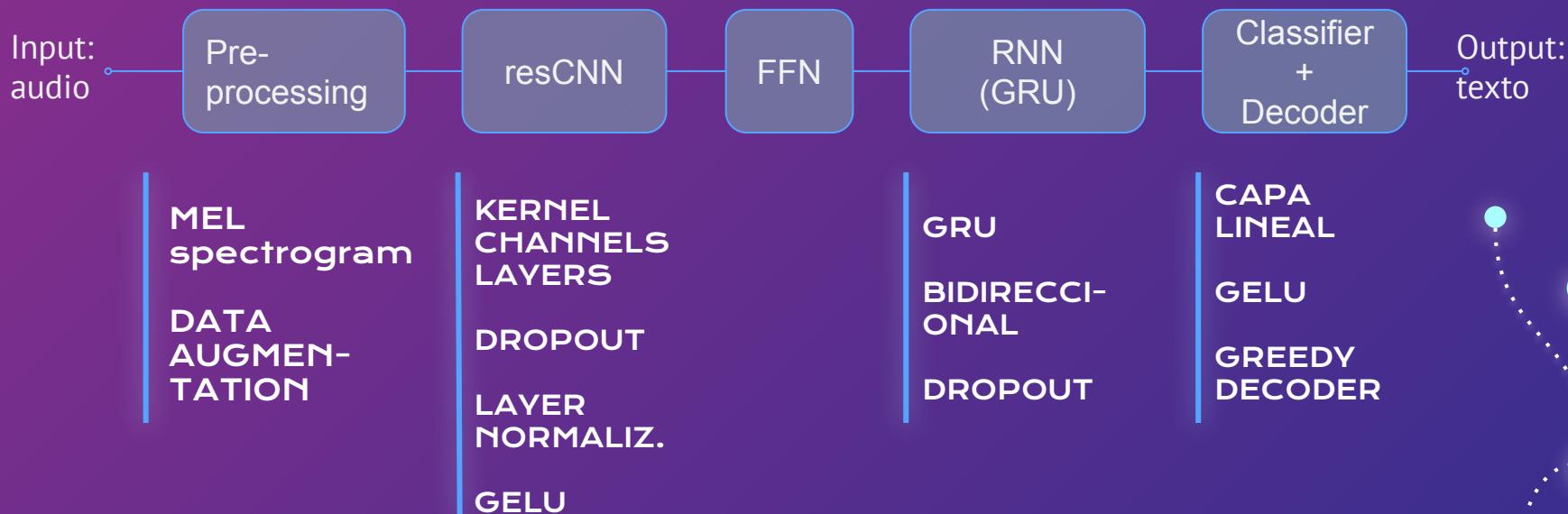
Aprende relaciones complejas entre la entrada y la salida.

Mejora predicciones con datos no vistos previamente.

# O2

## ARQUITECTURA DE LA RED

# OVERVIEW DE LA RED



# 03

## PRE- PROCESAMIENTO DEL INPUT

# DATASET: LATINO 40

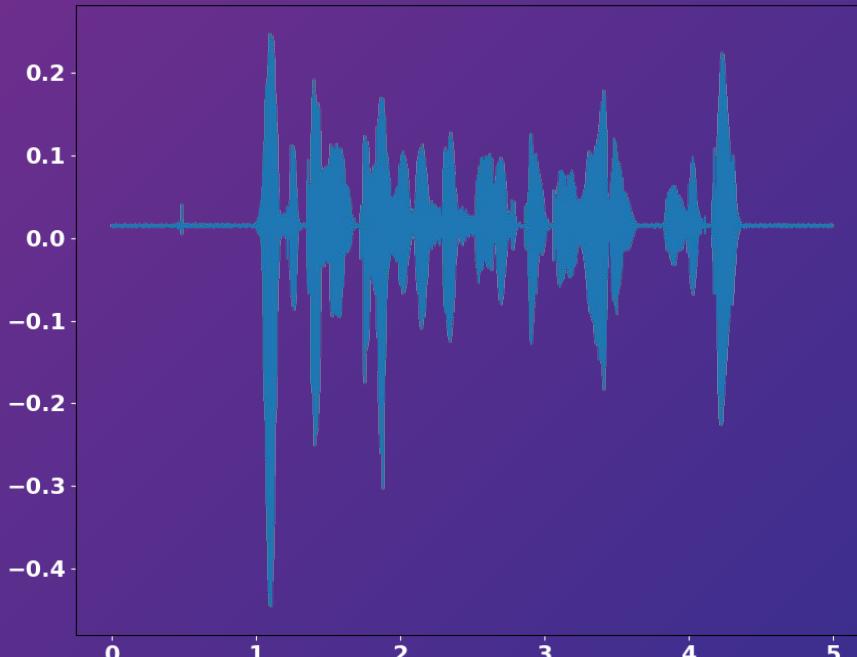
**3994**

Señales de  
entrenamiento

**500**

Señales de  
validación/test

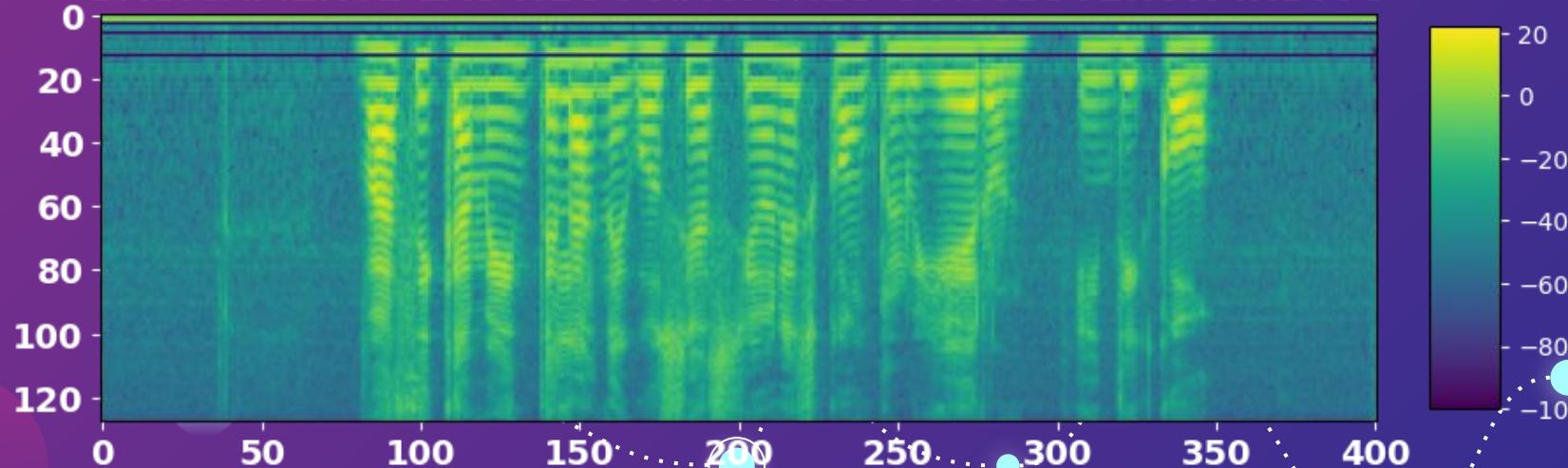
BASICAMENTE LAS NEGOCIACIONES CONCLUYERON INDICO



# MEL SPECTROGRAM

- **Escala Mel:** Transformación logarítmica de la frecuencia, asemejándose a la manera humana de discernir entre frecuencias
- Entonces, el input de la red es una imagen.

BASICAMENTE LAS NEGOCIACIONES CONCLUYERON INDICO



# DATA AUGMENTATION

**¿Qué método? Spectrogram Augmentation**

- Consiste en cortar bloques aleatorios del espectrograma tanto en tiempo como en frecuencia

**¿Cuándo se aplica? Durante el entrenamiento**

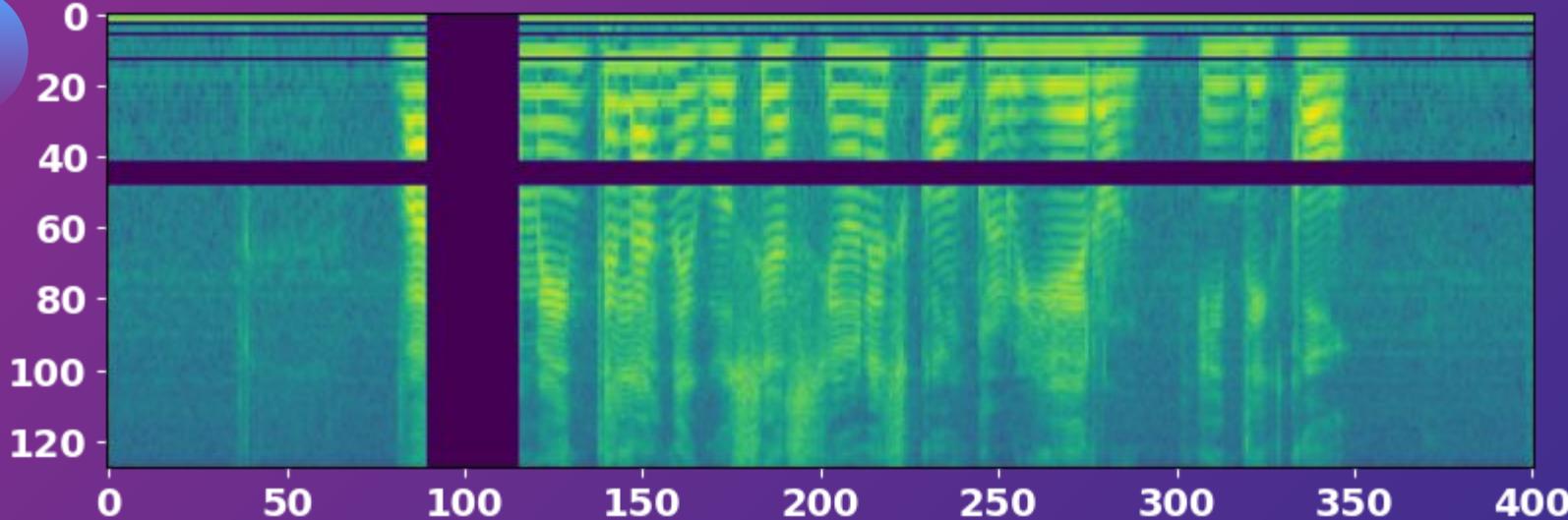
- En cada epoch, sobre cada sample

**Resulta en un aumento en la diversidad del dataset.**

**A efectos prácticos se aumenta el tamaño.**

# DATA AUGMENTATION: EJEMPLO

BASICAMENTE LAS NEGOCIACIONES CONCLUYERON INDICO



# 04

## REDES CONVOLUCIONALES

# CNN: INTRODUCCIÓN

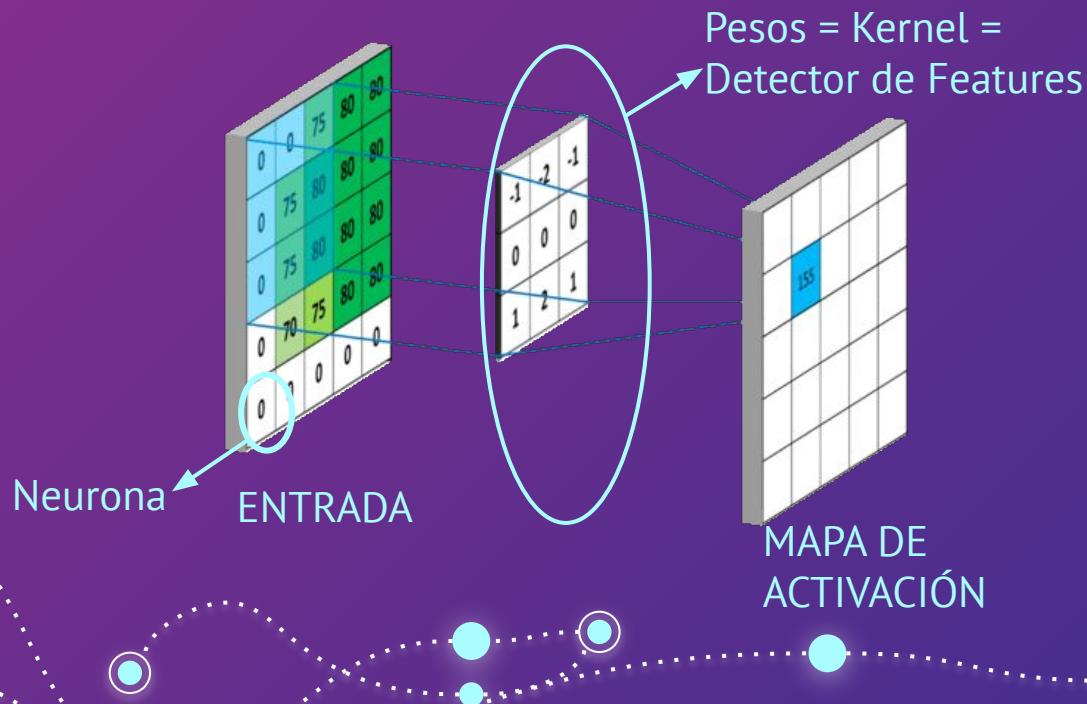
Las redes convolucionales son tipos de redes neuronales que se utilizan para reconocimiento de imágenes

Las redes convolucionales consideran la relación espacial que guardan las imágenes

La forma de capturar estas relaciones es mediante un *detector de features*, los cuales se aplican sobre todo el *input*

# CNN: Arquitectura

## 1 Capa convolucional



GRUPOS DE NEURONAS  
EN UNA CAPA ESTÁN  
CONECTADOS A GRUPOS  
DE NEURONAS EN LA  
PRÓXIMA CAPA

EL MISMO KERNEL SE  
APLICA SOBRE TODAS  
LAS NEURONAS,  
DESLIZÁNDOSE POR  
GRUPOS

ESTO REDUCE EL NÚMERO  
DE PARÁMETROS A  
APRENDER.

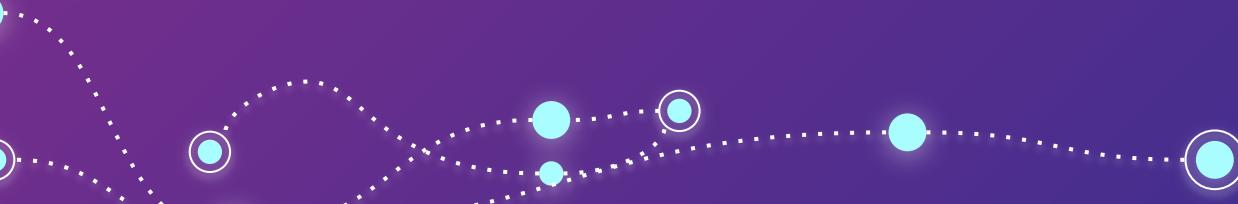
# CNN: KERNEL

EL FILTRO O KERNEL ES EL MISMO PARA TODAS LAS ENTRADAS

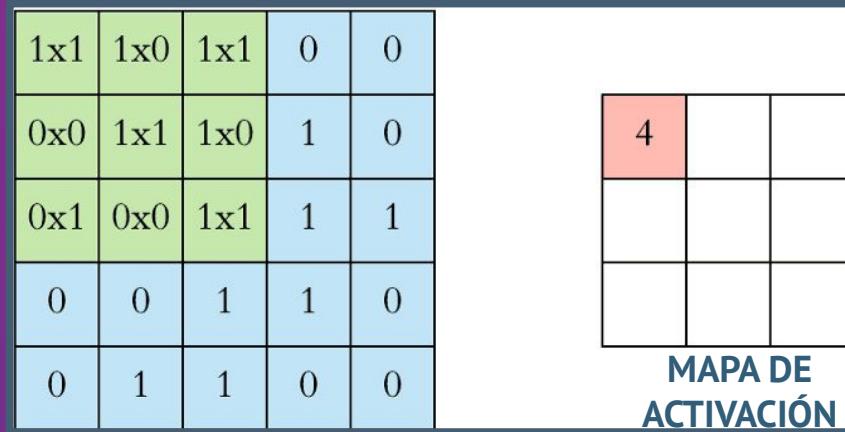
SE MULTIPLICA EL KERNEL SOBRE EL GRUPO DE NEURONAS Y SE SUMAN LOS RESULTADOS PARA OBTENER UNA NEURONA DEL MAPA DE ACTIVACIÓN

**CONVOLUCIÓN:** SE DESLIZA EL KERNEL SOBRE TODA LA IMAGEN HASTA FORMAR EL MAPA DE ACTIVACIÓN.

**CANALES:** CADA CAPA PUEDE TENER MÚLTIPLES FILTROS, QUE DETECTAN DISTINTOS FEATURES.



# CNN: MAPA DE ACTIVACIÓN



LA SALIDA TIENE MENOR DIMENSIÓN QUE LA ENTRADA

¿CÓMO CONTROLAMOS EL TAMAÑO DEL MAPA?

- Tamaño del filtro
- Stride
- Padding
- Pooling

$$L_{\text{out}} = \left\lfloor \frac{L_{\text{in}} + 2 \times \text{padding} - \text{dilation} \times (\text{kernel\_size} - 1) - 1}{\text{stride}} + 1 \right\rfloor$$

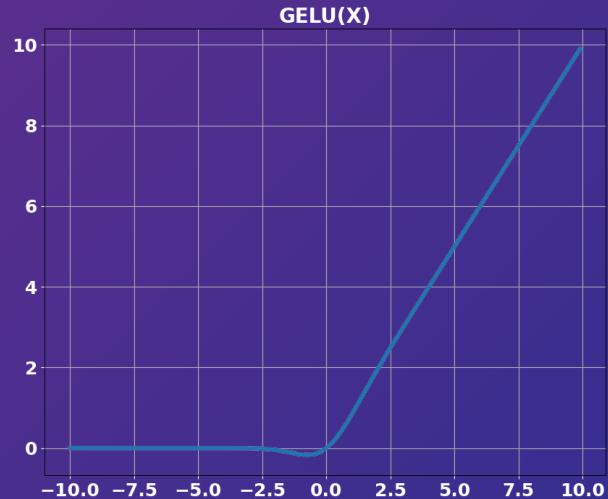
# CNN: FUNCIÓN DE ACTIVACIÓN

AGREGA LA NO-LINEALIDAD A LA RED.

SE APLICA SOBRE EL MAPA DE ACTIVACIÓN

## GELU(X)

- Es la función de probabilidad acumulada de una *normal estándar*
- Es similar a RELU, pero es más suave y diferenciable en todo el dominio



# CNN: LAYER NORMALIZATION

ESTANDARIZA LAS ENTRADAS CALCULANDO LA MEDIA Y EL DESVÍO ESTÁNDAR SOBRE LA DIMENSIÓN DE LOS FEATURES

ES MÁS APLICABLE QUE BATCH NORMALIZATION CUANDO EL TAMAÑO DEL BATCH SIZE ES PEQUEÑO

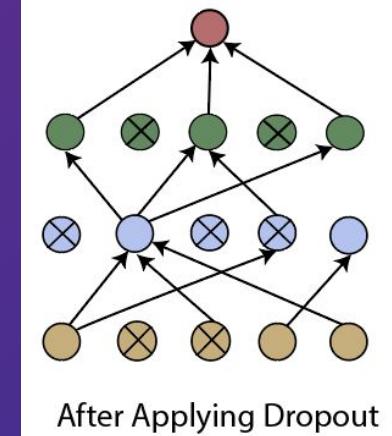
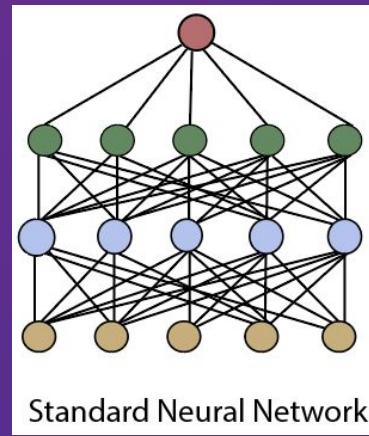
ESTA TRANSFORMACIÓN SE APLICA SOBRE EL MAPA DE ACTIVACIÓN, ANTES DE LA FUNCIÓN DE ACTIVACIÓN

	Samples		
Features	x <sub>1</sub>	x <sub>2</sub>	x <sub>3</sub>
x <sub>1</sub>	1	3	8
x <sub>2</sub>	3	4	3
x <sub>3</sub>	5	6	2
Mean	3	4.33	4.33
Variance	2.67	1.56	6.89
Normalize			

# CNN: DROPOUT

CONSISTE EN APAGAR CADA NEURONA CON PROBABILIDAD  $\alpha$  DURANTE EL ENTRENAMIENTO

PERMITE QUE LAS NEURONAS NO SEAN TAN DEPENDIENTES DE OTRAS NEURONAS, REDUCE EL OVERFITTING

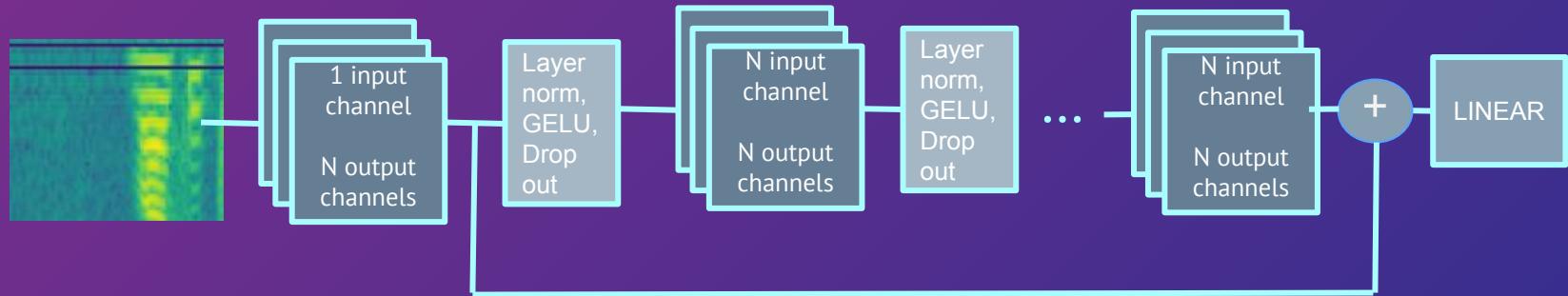


# resCNN: ARQUITECTURA COMPLETA

Luego de la última capa se agrega una capa lineal

**Residual:** se agrega una conexión directa en paralelo desde la entrada hacia la salida de la red

Evita el desvanecimiento del gradiente en redes muy profundas



# 05

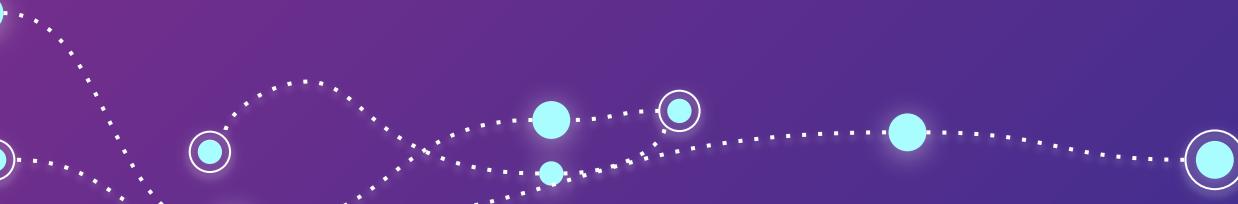
## REDES RECURRENTES

# RNN: INTRODUCCIÓN

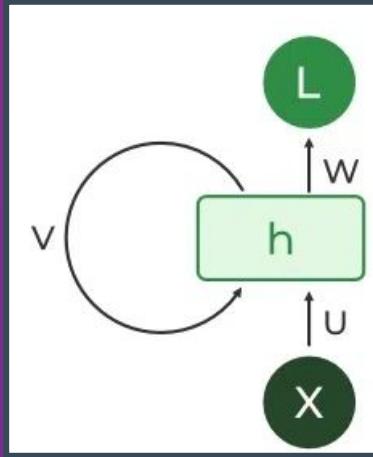
Las redes recurrentes son tipos de redes neuronales que involucran datos secuenciales o en series de tiempo

Las redes recurrentes tienen estados ocultos, que se generan tiempo a tiempo

Cada estado oculto guarda información del estado oculto en un instante anterior, esto de una noción de memoria a la red.



# RNN: ARQUITECTURA DE LA RED



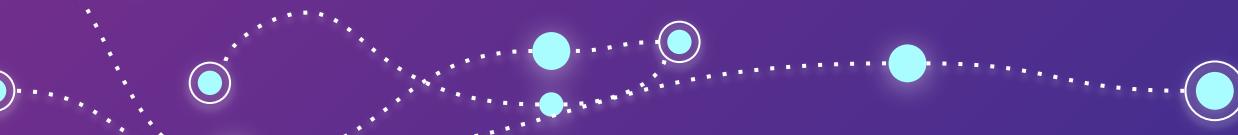
EN EL INSTANTE T LA ENTRADA X SE COMBINA CON EL ESTADO OCULTO H DEL TIEMPO T-1 PARA GENERAR EL ESTADO OCULTO EN EL TIEMPO T

$$h_{(t)} = \tanh(U \cdot x_{(t)} + Vh_{(t-1)})$$

LA SALIDA EN EL INSTANTE T SE CALCULA COMO LA MULTIPLICACIÓN ENTRE EL ESTADO OCULTO Y LA MATRIZ DE PESOS W

$$L_{(t)} = f(Wh_{(t)})$$

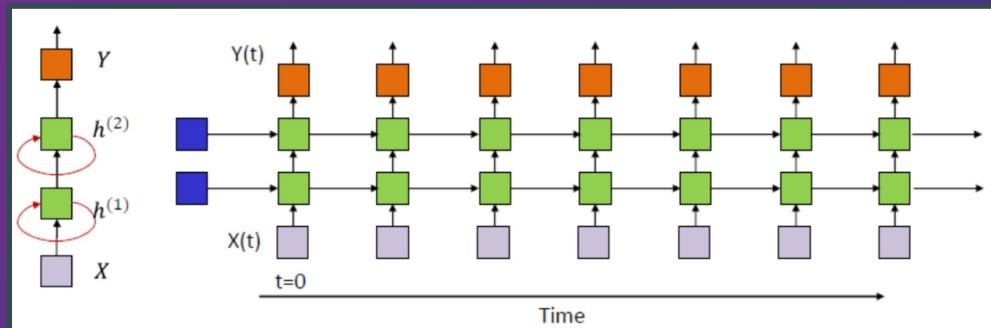
Si se expande la red a lo largo del tiempo, se obtiene una red de varias capas en la cual la salida corresponde al instante N



# RNN: PROFUNDIDAD DE LA RED

Además de expandirse en el tiempo, puede diseñarse la red con varias capas en un mismo instante, para generar la salida en cada uno.

EN TODAS LAS COLUMNAS, ES DECIR A LO LARGO DEL TIEMPO, LAS CAPAS SON IDÉNTICAS

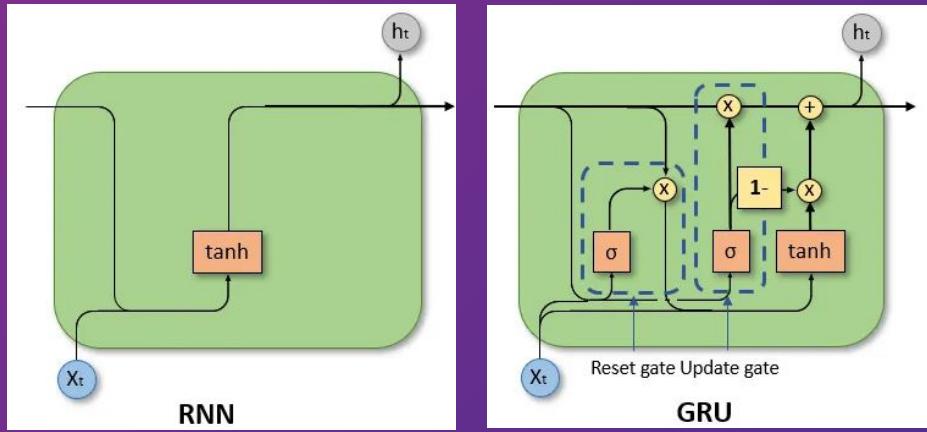


## PROBLEMAS

- En secuencias largas, al propagar el gradiente hacia el tiempo O se desvanecerá.
- A medida que se propaga hacia la salida, se perderá todo recuerdo de la entrada en instantes lejanos.

# SOLUCIÓN: GRU

ESTE TIPO DE REDES INTRODUCE COMPUERTAS QUE DICTAN CUÁNTO DEL ESTADO ANTERIOR SE RECUERDA Y CUÁNTO SE PROPAGA HACIA EL ESTADO ACTUAL



$$\begin{aligned}r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \\n_t &= \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn})) \\h_t &= (1 - z_t) * n_t + z_t * h_{(t-1)}\end{aligned}$$

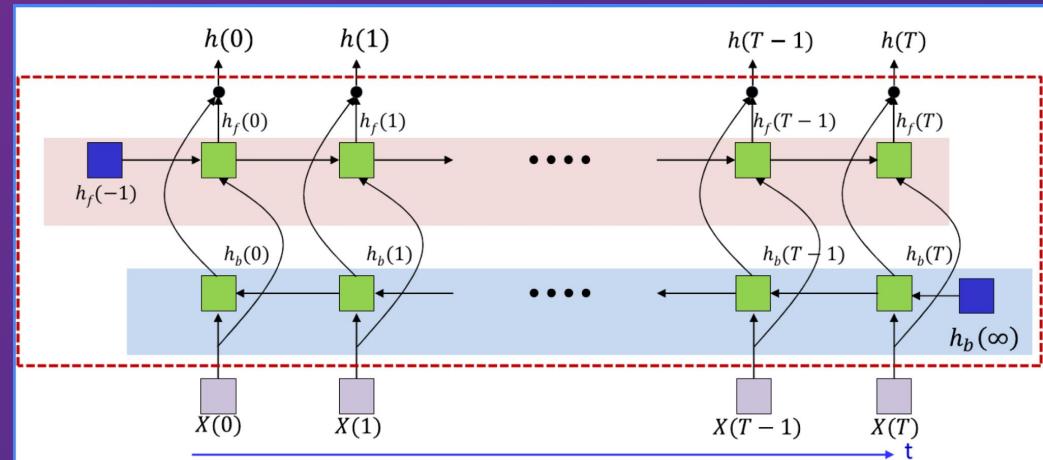
# GRU BIDIRECCIONAL

PARA REFORZAR LA MEMORIA A LARGO PLAZO SE UTILIZA UNA RED FORWARD Y UNA RED BACKWARD

La red forward procesa la entrada desde  $t = 0$  hasta  $T$

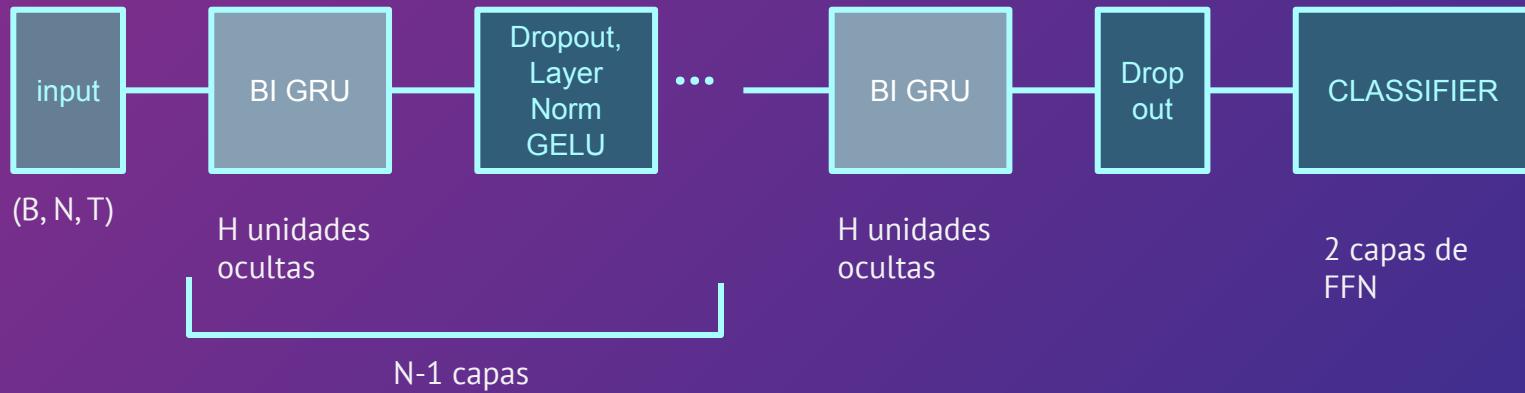
La red backward procesa la entrada desde  $t = T$  hasta 0

Se concatena las salidas antes de alimentarla a una red Feed Forward



# BiGRU: ARQUITECTURA COMPLETA

Como las entradas corresponden a la salida de la red convolucional, primero se deben aplanar para expresarlas en forma de secuencia



# 06

## ENTRENAMIENTO

# SALIDA DE LA RED



La función de activación de la última capa de la Red Feed Forward es una softmax de dimensión 29.

Para una secuencia de entrada, la red arroja una matriz de probabilidades (con la probabilidad de cada carácter en cada instante de tiempo).

# CTC Loss: Introducción

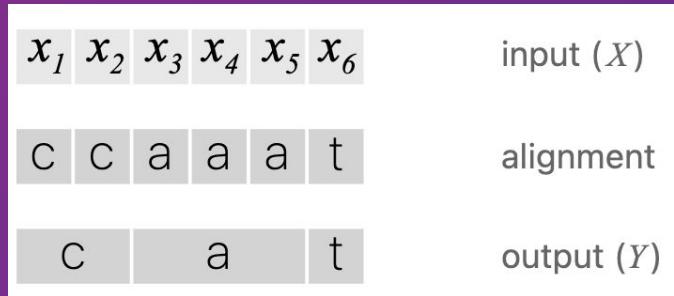
La matriz de probabilidades obtenida permite formar una secuencia de caracteres para comparar con los labels de las entradas.

El largo de la secuencia de salida obtenida es **mayor** que el largo de la secuencia label.

Sin embargo, se desconoce el alineamiento de la secuencia de salida con la de entrada

Este algoritmo computa la probabilidad de una secuencia de salida dada una entrada, y eso lo hace sumando las probabilidades de todos los **alineamientos posibles**:

# CTC Loss: blank token



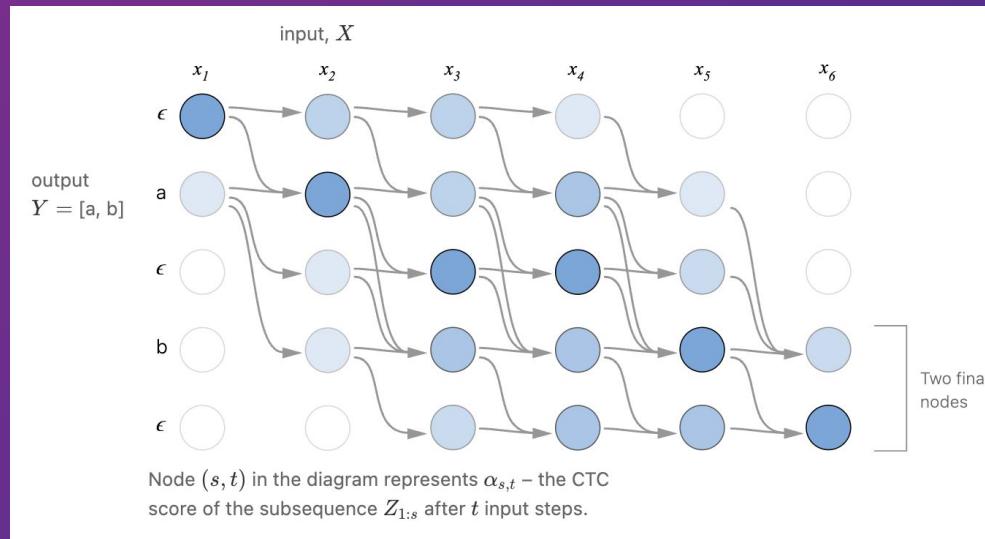
Como las secuencias generadas son tan largas como la secuencia de entrada, en los alineamientos posibles se colapsan los caracteres repetidos para obtener una secuencia válida

Se introduce un carácter *blank*, representado como  $\epsilon$ , que permite:

- Tener instantes en donde no hay carácter
- No colapsar caracteres que deben repetirse

# CTC Loss: alineamientos permitidos

Un alineamiento permitido es aquel que al reducirlo conduce a la secuencia de caracteres target



# CTC Loss: implementación

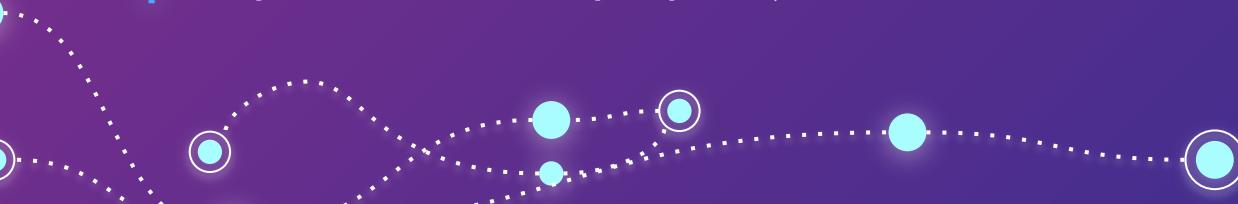
La ecuación que describe el modelo se puede escribir como:

$$P_{CTC}(C|X) = \sum_{Z \in \Lambda} \Pi_{t=1}^T P(z_t|X)$$

Donde C es la secuencia de caracteres target, X la secuencia de entrada.

La sumatoria sea realiza sobre todos los alineamientos permitidos.

La probabilidad de un carácter en tiempo t dado X se obtiene de la matriz de probabilidades que predijo la red



# CTC Loss: implementación

La función encontrada es diferenciable respecto de las probabilidades en cada tiempo.

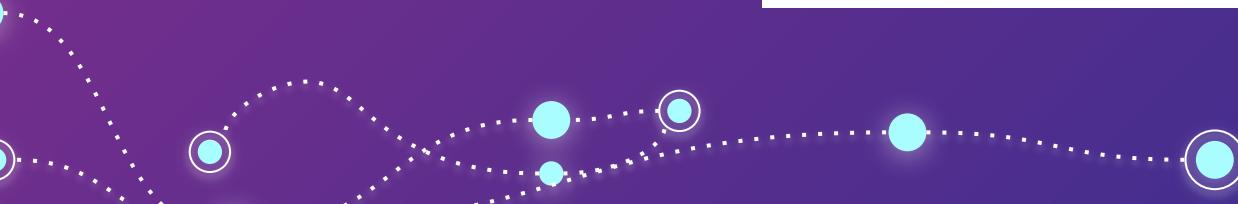
Entonces, se puede calcular el gradiente analiticamente y realizar el algoritmo de backpropagation

La función de Loss planteada es:

$$Loss(X, C) = -\log P_{CTC}(C|X)$$

El criterio de minimización:

$$Loss(W) = \frac{1}{N} \sum_{i=1}^N Loss(X^i, C^i; W)$$



# Backpropagation

**Autograd(): Pytorch** almacena los gradientes de Loss( $W$ ) con respecto a los parámetros de la red

En CNN: El gradiente es similar a una red FF pero su aplica una restricción para forzar que los filtros sean iguales en toda la capa

En RNN: Para calcular el gradiente es necesario recorrer toda la secuencia.

**Optimizer:** AdamW

Modifica SGD para producir grandes cambios en los parámetros cuando el gradiente no varía mucho y viceversa.

**Scheduler:** One Cycle Learning Rate Scheduler

Modifica la constante de aprendizaje: empieza en un mínimo, crece linealmente hasta un máximo y luego desciende linealmente hasta el mínimo

# 07

## INFERENCIA

# INFERENCIA: DECODER



Recordemos que la salida de la red es una matriz de probabilidades de cada carácter en cada instante de tiempo.

Esta red se debe poder utilizar con entradas cuyas etiquetas son desconocidas, con lo cual, no se puede utilizar el algoritmo de CTC para decodificar la salida.

Una opción es un Greedy Decoder

# INFERENCIA: GREEDY DECODER

Este decoder recorre el camino de mayor probabilidad marginal para cada instante T.

Es decir, para cada tiempo, elije el carácter con mayor probabilidad, y en caso de ser un blank token, se elimina de la secuencia final.

La inferencia es a nivel carácter, no a nivel palabra. El carácter <ESPACIO> tiene su codificación.

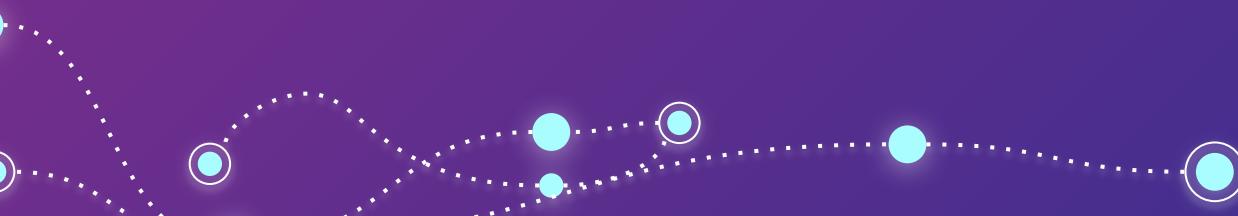
Ejemplo:

Target:

nadie ha sido condenado a la pena capital

Predictión:

nadia ha sido condenado la pena capitarl



# 08

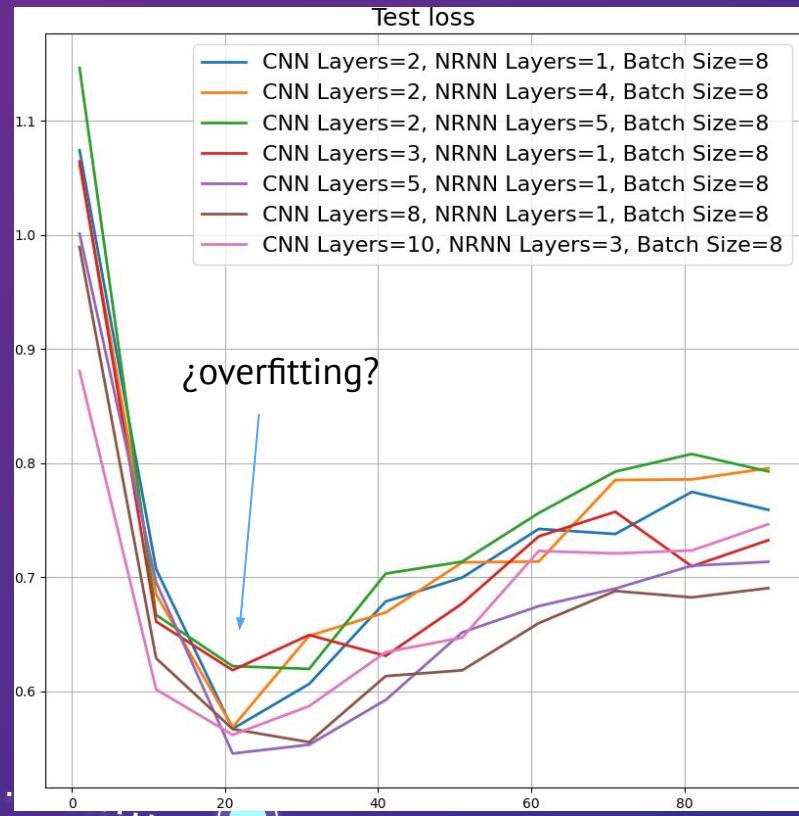
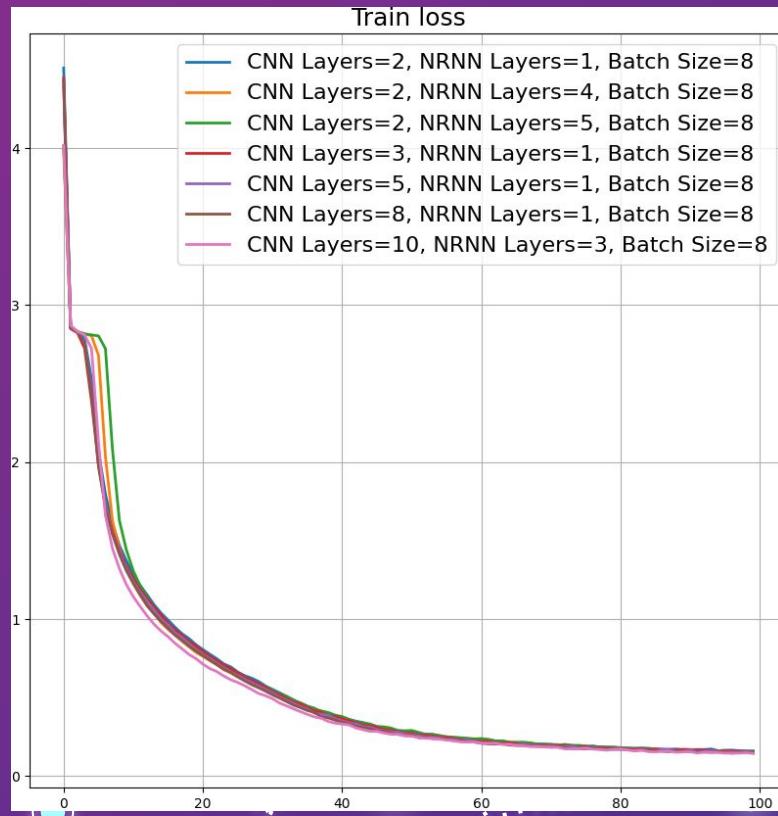
## RESULTADOS OBTENIDOS

# REDES ENTRENADAS

Se entrenaron múltiples redes variando la cantidad de capas de las CNN y RNN, siempre utilizando un `batch_size = 8`:

- Nro capas de CNN = 2, Nro capas de RNN = 1
  - Nro capas de CNN = 2, Nro capas de RNN = 4
  - Nro capas de CNN = 2, Nro capas de RNN = 5
  - Nro capas de CNN = 3, Nro capas de RNN = 1
  - Nro capas de CNN = 5, Nro capas de RNN = 1
  - Nro capas de CNN = 8, Nro capas de RNN = 1
  - Nro capas de CNN = 10, Nro capas de RNN = 1
  - Nro capas de CNN = 3, Nro capas de RNN = 4
  - Nro capas de CNN = 10, Nro capas de RNN = 3
- 
- RNN Layers
- CNN Layers
- Ambas Layers

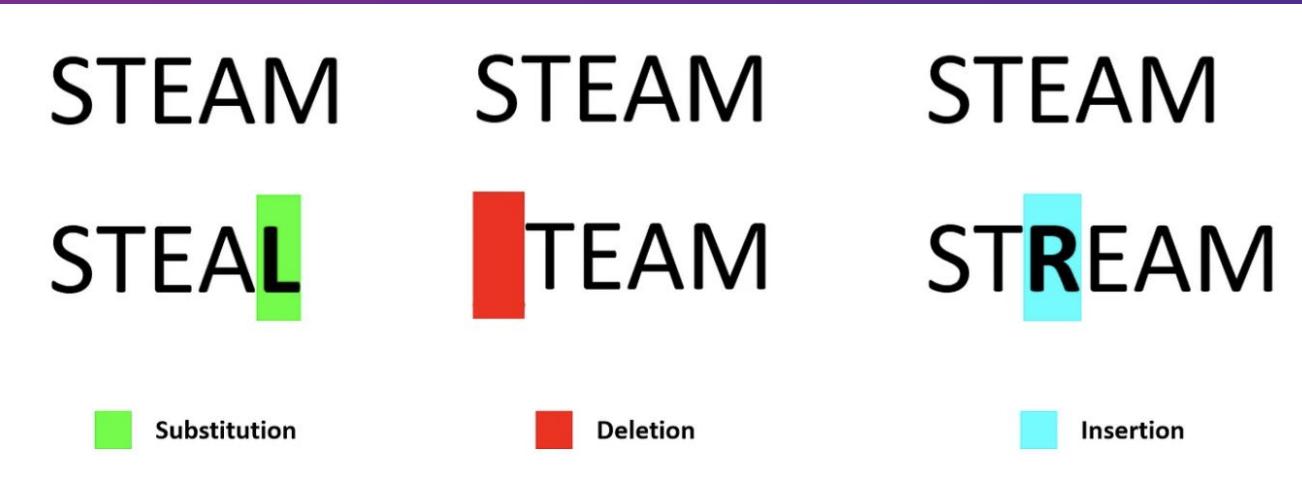
## RESULTADOS: TRAIN + VALIDATION LOSS



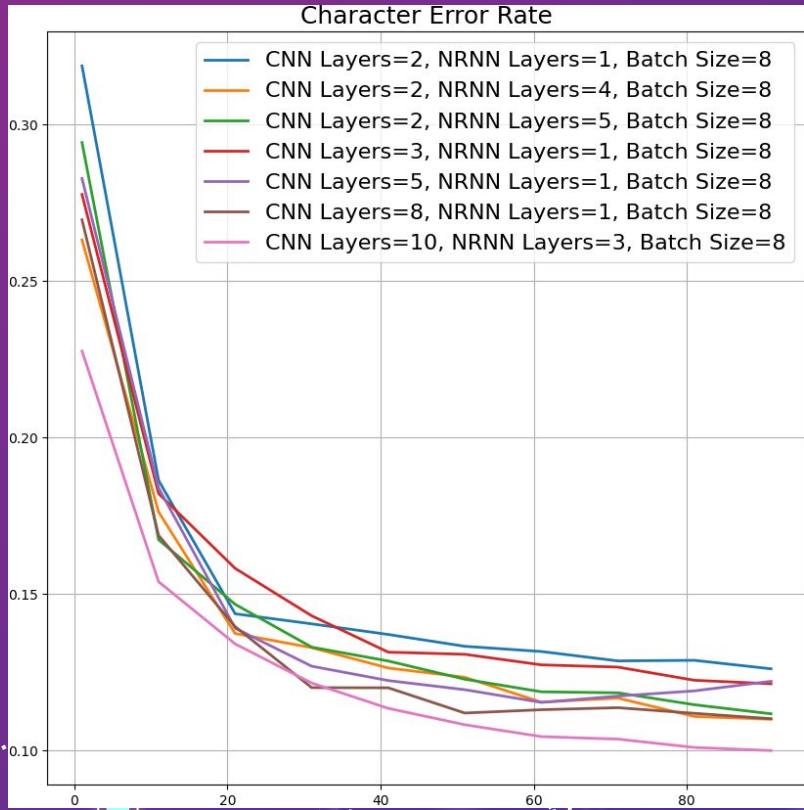
# RESULTADOS: CER P/EPOCH + AVG

Cálculo del CER:

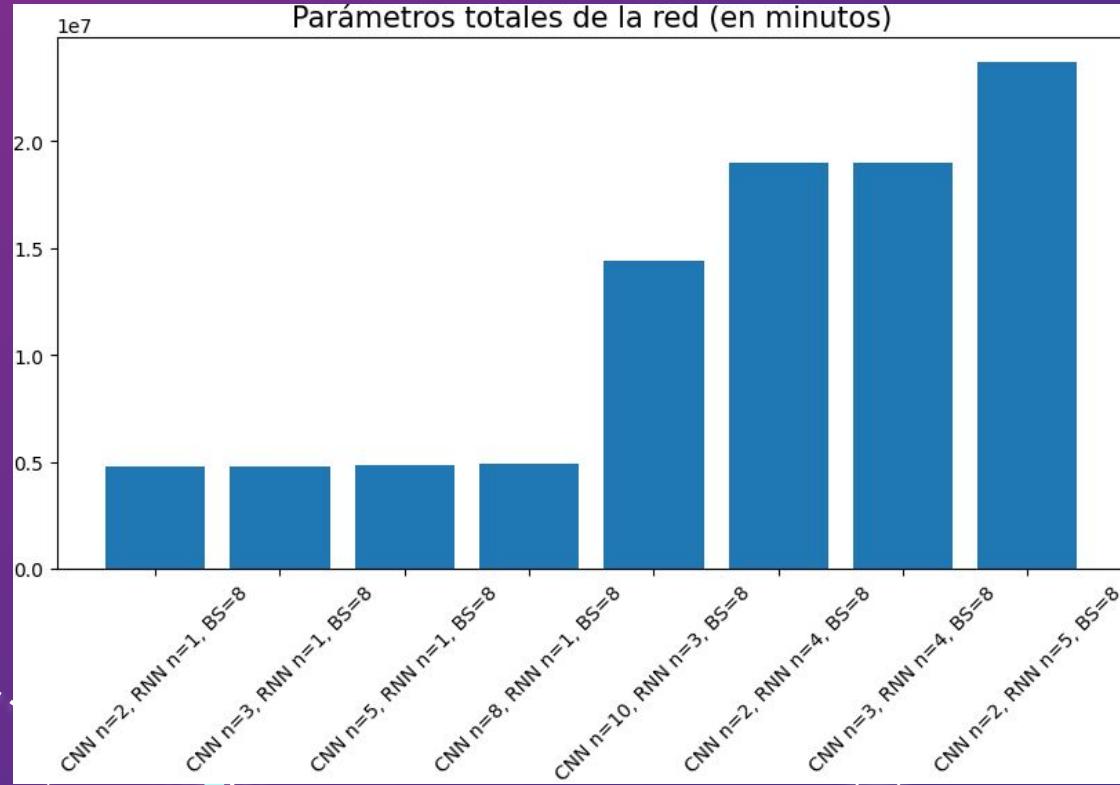
$$CER = \frac{S + D + I}{N}$$



# RESULTADOS: CER P/EPOCH + AVG



# RESULTADOS: PARAMETROS TOTALES



# RESULTADOS: APRECIACIÓN SUBJETIVA

Ejemplo: N CNN = 8, N RNN = 1

Target:

nadie ha sido condenado a la pena capital

Prediccion:

nadia ha sido condenado la pena capitari

Ejemplo: N CNN = 2, N RNN = 5

Target:

nadie ha sido condenado a la pena capital

Prediccion:

nadia ha sido con denado la pena capitarie

# RESULTADOS: APRECIACIÓN SUBJETIVA

| Ejemplo: N CNN = 8, N RNN = 1

Target:

en este debate no hay lugar para la retorica de salon

Prediccion:

en este de bater o alugar para la retorica desalon

| Ejemplo: N CNN = 2, N RNN = 5

Target:

en este debate no hay lugar para la retorica de salon

Prediccion:

en ieste de bvate no ayludar para la retoricadesalon



| ¡Muchas  
Gracias!

# REFERENCIAS

- <https://www.assemblyai.com/blog/end-to-end-speech-recognition-pytorch/>
- <https://www.pinecone.io/learn/batch-layer-normalization/>
- <https://towardsdatascience.com/why-adamw-matters-736223f31b5d>
- <https://distill.pub/2017/ctc/?undefined=&ref=assemblyai.com>
- <https://jinglescode.github.io/2020/11/01/how-convolutional-layers-work-deep-learning-neural-networks/>
- <https://medium.com/@mihirkhandekar/forward-and-backpropagation-in-grus-derived-deep-learning-5764f374f3f5>