UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA
Año 2017 - 2do Cuatrimestre

(86.53) PROCESAMIENTO DEL HABLA

TRABAJO PRÁCTICO FINAL
TEMA: Reconocimiento de habla mediante redes neuronales
FECHA: 22 de septiembre de 2018

ALUMNO:
ZORZANO, Nicolás - #95494

Índice

1. Objetivo

En este trabajo se plantearon dos estrategias basadas en redes neuronales para el problema de reconocimiento de habla. La primer estrategia consistió en un clasificador de fonemas, el cual permite indicar en cada frame de audio el fonema pronunciado. El segundo problema planteado fue el de la utilización de una función de costo capaz de trabajar con secuencias no alineadas para poder implementar un reconocedor de habla a partir de transcripciones no alineadas.

El objetivo de este trabajo es el de analizar ambas estrategias, comparar resultados entre ellas y con resultados obtenidos en trabajos previos. Por otra parte, es deseable analizar la influencia de la existencia de herramientas de diseño de redes neuronales, que permiten reducir los tiempos y la complejidad del desarrollo del código necesario. Para el desarrollo de redes neuronales existen distintos tipos de librerías como Keras, Caffe, etc. Sin embargo, se optó por utilizar TensorFlow, ya que se trata de una librería diseñada por Google, y por ende presenta una mayor probabilidad de establecerse como estándar para el diseño de herramientas basadas en redes neuronales.

2. Introducción

2.1. La señal de habla

La generación de las señales de habla es un mecanismo complejo que se efectúa en la laringe, donde se encuentran los pliegues vocales que regulan el paso del aire mediante el cierre y abertura de la glotis. La señal de habla resulta en una señal no estacionaria, que no solo depende de las palabras pronunciadas, sino también del tracto vocal de la persona parlante, de la duración, del tono, y demás factores que hacen difícil la tarea de encontrar un denominador común entre distintas pronunciaciones de una misma palabra. Por otra parte, las señales de habla pueden ser pensadas como una

combinación de unidades básicas conocidas como fonemas, los cuales son la articulación mínima de un sonido vocálico[1] y consonántico[2].

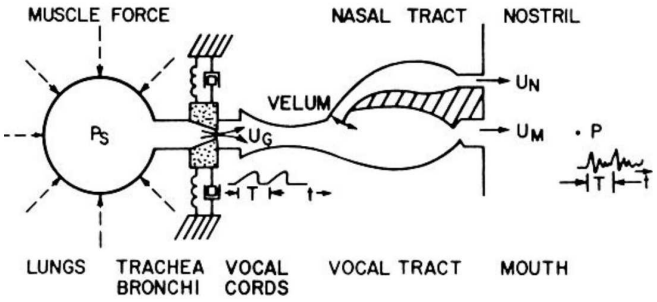


Figura 2.1.1: Representación del tracto vocal.

De todo esto, resulta que la señal de habla es una señal no estacionaria que puede ser modelada mediante filtros no lineales. Sin embargo, utilizando ventanas de aproximadamente 10 ms o 20 ms, es posible aproximar variaciones mínimas y así utilizar filtros lineales de la forma:

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 - \sum_{k=1}^{N} a_k z^{-k}}$$

Debido a que las señales de habla incluyen información irrelevante para su reconocimiento (duración de los fonemas, tono, sexo, etc.), es necesario utilizar un método de representación que permita extraer aquellas características introducidas por el tracto vocal que son dependientes de cada persona en particular. Para ésto es conveniente utilizar los **Mel Frequency Cepstral Coefficients (MFCC)**, donde no solo se utiliza el cepstrum sino que también se introduce la escala no lineal Mel que aproxima el comportamiento del sistema auditivo. La obtención de estos coeficientes se realiza de la siguiente manera:

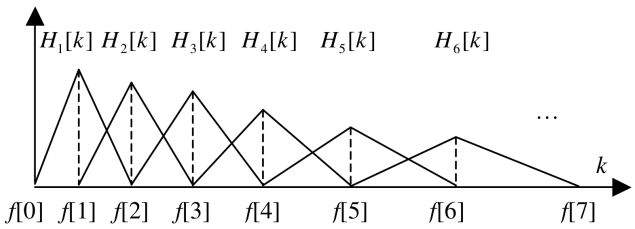


Figura 2.1.2: Banco de filtros en la escala Mel.

- Se ventanea la señal en distintos frames.
- A cada frame se le aplica la DFT:

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j\frac{2\pi nk}{N}} \qquad 0 \leq k < N$$

- Se define el banco de filtros en la escala Mel. Estos filtros calculan el espectro promedio alrededor de cada frecuencia central, utilizando anchos de banda cada vez mayores (ver figura 2.1.2). Una posible definición para dichos filtros es:

$$H_m[k] = \begin{cases} 0 & k < f[m-1] \\ \frac{2(k - f[m-1])}{(f[m+1] - f[m-1])(f[m] - f[m-1])} & f[m-1] \leq k < f[m] \\ \frac{2(f[m+1] + k)}{(f[m+1] - f[m-1])(f[m] - f[m-1])} & f[m] \leq k < f[m+1] \\ 0 & k > f[m+1] \end{cases}$$

- Se calcula el logaritmo de la energía de salida de cada uno de los filtros:

$$S[m] = \ln\left(\sum_{k=0}^{N-1} |X[k]|^2 H_m[k]\right) \qquad 0 \leq k < M$$

[1] Al pronunciar los fonemas vocálicos el aire no encuentra ningún tipo de obstáculo en su salida hacia el exterior. Estos fonemas son: /a/, /e/, /i/, /o/ y /u/.

[2] En la articulación de los fonemas consonánticos, se ponen distintos obstáculos al aire para salir por la boca, ya sea con los labios,la lengua, los dientes, etc.

- Finalmente se aplica la Transformada Coseno Discreta para encontrar los coeficientes:

$$c[n] = \sum_{m=0}^{M-1} S[m] \cos\left(\frac{\pi n}{M}\left(m + \frac{1}{2}\right)\right)$$

En la figura 2.1.3 se tiene una representación de los datos obtenidos de cada una de las señales de audio.

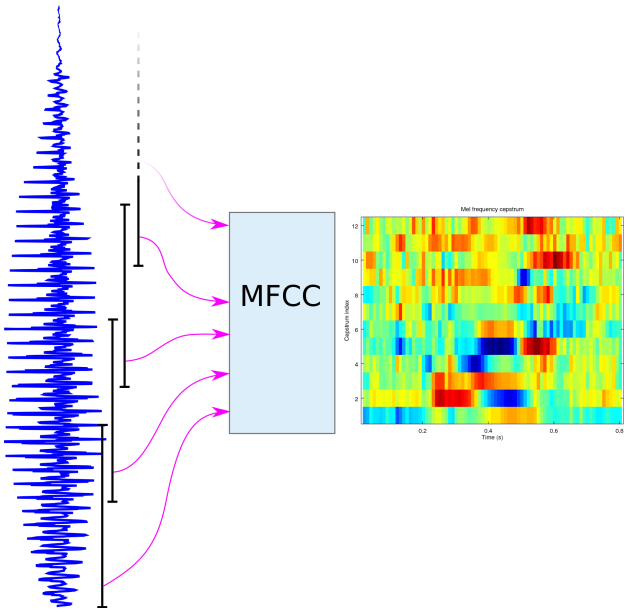
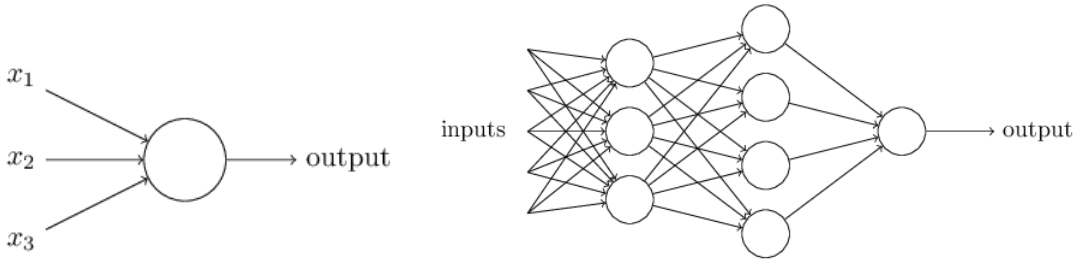


Figura 2.1.3: Procesamiento de las señales de habla para la obtención de los MFCC.

2.2. Redes neuronales

Si bien las redes neuronales comenzaron a ser investigadas por la década del 40, no fue sino hasta los últimos años donde éstas cobraron mayor importancia. Gracias al incremento en los datos disponibles y al mejoramiento del hardware específico, principalmente las GPU, se hizo posible el diseño y creación de sistemas capaces de resolver problemas complejos y hasta en algunos casos superar el rendimiento humano en tareas específicas.

2.2.1. Perceptrón multicapa



(a) Imagen de un perceptrón.

(b) Figura de un perceptrón multicapa. La primer capa es la capa de entrada, la última capa es la de salida, y la del medio es la capa oculta (hidden layer).

Figura 2.2.1: Representación de un perceptrón.

El perceptrón es la unidad básica de una red neuronal, y constituye un modelo matemático basado en el comportamiento de las neuronas reales. Un perceptrón es un componente que presenta una serie de entradas $\{x_1, \ldots, x_n\}$, y una única salida. Cada una de estas entradas tiene asociado un peso $\{w_i\}$ (conocidos también como pesos sinápticos) que

indica la importancia de dicha entrada a la salida. A esta suma ponderada de las entradas se les suele agregar un término de threshold o bias b, así como también una función de activación $\sigma(\cdot)$ que permite trabajar con salidas no lineales, y solucionar muchos problemas que surgen durante el entrenamiento. La salida de un perceptrón se puede expresar entonces como:

$$\mathcal{O} = \sigma\left(\sum_{j=1}^{N} w_j x_j + b\right) = \sigma\left(\mathbf{w} \cdot \mathbf{x} + b\right)$$

Un claro ejemplo de uso de un perceptrón es un clasificador lineal. En estos casos la función de activación puede ser simplemente un escalón, y siempre y cuando las entradas pesadas por sus correspondientes pesos superen el umbral b, serán clasificadas en una categoría u otra.

Sin embargo, utilizar un perceptrón como única unidad de cálculo no permite trabajar con problemas linealmente separables. Para estos casos se suelen armar estructuras conocidas como **perceptrones multicapa** donde se agrupan en varias capas distintas cantidades de perceptrones, permitiendo que en cada capa se tengan niveles de abstracción diferentes, y así poder trabajar con problemas cada vez más complejos. En la figura 2.2.1b se tiene un esquema típico de esto.

La estrategia más sencilla para trabajar con perceptrones multicapa es la conocida como **capas fully connected (FCL)**. En este tipo de capas se observa que todas las neuronas de una capa se encuentran interconectadas con todas las neuronas de la siguiente capa. Esto permite que todos los elementos dentro de grandes vectores de información puedan ser interrelacionados, y así permitir que la red neuronal sea la encargada de encontrar la mejor representación de la información o los features más preponderantes, en lugar de tener que ser realizado de manera manual (que en general es mucho menos eficiente).

2.2.2. Redes convolucionales

Como se mencionó previamente las FCL permiten correlacionar gran cantidad de datos, evitando que en muchos casos se deba hacer un procesamiento previo o al menos que éste sea solo para eliminar casos que aportan poca información. Sin embargo, dado que todos los datos son cruzados entre sí, la información de la ubicación de cada uno de los datos se pierde. El caso más común de esto es en las redes de reconocimiento de imágenes, donde cada píxel se encuentra altamente correlacionado con los píxeles vecinos, no así con aquellos que se encuentran más alejados espacialmente.

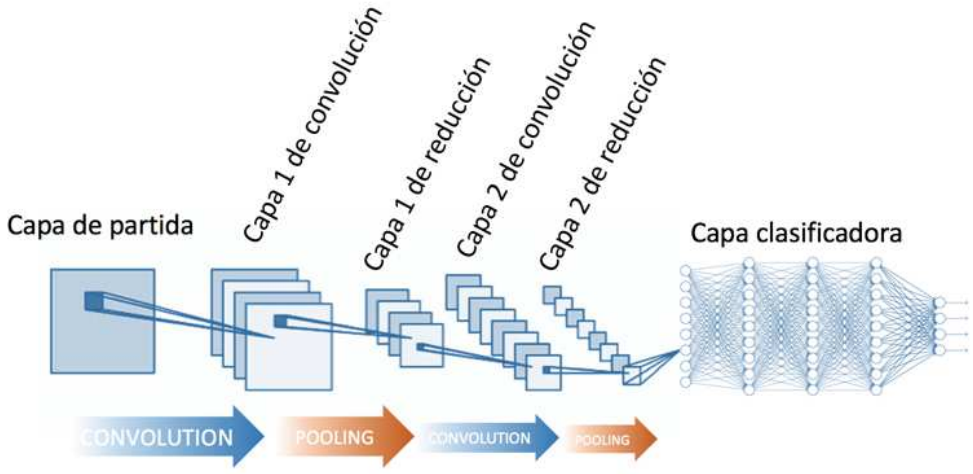


Figura 2.2.2: Arquitectura de una red convolucional.

Las redes convolucionales se caracterizan por tener una arquitectura que fomenta a correlacionar entradas cercanas espacialmente. Esto lo logra mediante pequeñas matrices llamadas **kernels**, generalmente de 3×3 o 5×5, que son aplicadas mediante un barrido a toda la matriz de entrada. El proceso en una red convolucional se puede resumir de la siguiente manera:

- Se aplica el kernel de $n \times n$ al primer conjunto de muestras. Para esto se multiplica elemento por elemento contra una submatriz también de tamaño $n \times n$ de la entrada.

- A la matriz de tamaño $n \times n$ que se obtiene como resultado, se le aplica una transformación, conocida como **pooling**, donde se reduce la dimensionalidad de acuerdo con algún criterio (se retiene el máximo, el mínimo, el promedio, etc.).
- El resultado de la capa de **pooling** se agregado a la matriz de salida en la posición que se corresponde con el desplazamiento del kernel.
- Se desplaza el kernel una posición y se repite el proceso.

Se puede ver que el mecanismo descrito tiene una gran similitud con una convolución, donde el kernel barre la matriz de entrada y presenta a su salida una nueva matriz de menor dimensión, pero donde la información fue destacada por sobre la redundancia. En general en las redes convolucionales se suelen aplicar varios kernels diferentes a una misma imagen, por lo que a la salida de cada capa se tiene una nueva matriz de menor dimensión por cada kernel aplicado (ver figura 2.2.2).

Mediante el diseño de los distintos kernels es posible identificar distintos patrones en las imágenes de entrada: lineas verticales, lineas horizontales, esquinas, etc. Sin embargo, dado que en general no se conoce cuál es la mejor representación de la información disponible, los kernels suelen ser de valores aleatorios, nuevamente permitiendo que la red encuentre la representación más conveniente para resolver el problema.

Generalmente los valores de los kernels no son modificados durante el entrenamiento. Por lo tanto, para que una red convolucional sea capaz de aprender a realizar una tarea se suele agregar una FCL a su salida, la cual se encargará de ajustar sus pesos sinápticos de manera que los features encontrados por las capas convolucionales puedan ser ajustados a los resultados esperados.

Las redes convolucionales han demostrado ser altamente eficaces en tareas como en clasificación e identificación de imágenes. Por otra parte, dado que los kernels no son entrenables, la cantidad de parámetros a entrenar suele reducirse considerablemente, reduciendo el costo computacional requerido.

2.2.3. Redes recurrentes

En muchos casos la correlación entre muestras no se encuentra debida a su ubicación espacial, sino que a su dependencia temporal. En muchas situaciones como en el procesamiento de habla, estudio del clima, etc., la información de muestras pasadas incide directamente sobre la información recibida en el instante actual. Para este tipo de problemas surgen arquitecturas conocidas como redes recurrentes, donde al momento de realizar una inferencia se pesan los valores actuales y pasados (y hasta en ciertos casos los futuros).

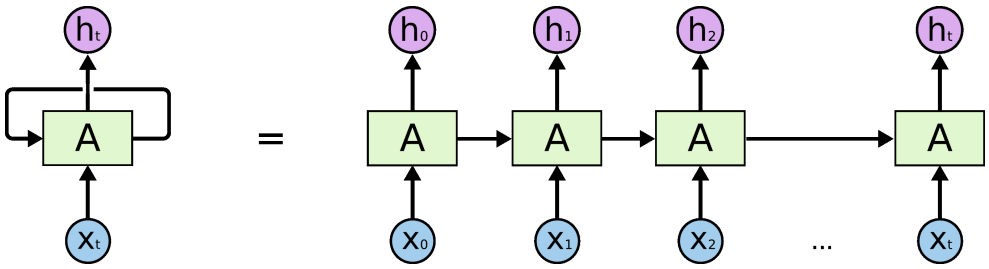


Figura 2.2.3: Red recurrente desdoblada a lo largo del tiempo.

Las redes recurrentes pueden ser pensadas como una secuencia de copias de ellas mismas, que pasan mensajes pasados a sus sucesores, como se puede apreciar en la figura 2.2.3. Si bien esta arquitectura permite relacionar temporalmente distintas entradas, surge el problema de determinar a priori que tan persistente debe ser dicha dependencia.

Una arquitectura especial que permite lidiar con esto es la conocida como **redes LSTM (Long Short Term Memory)**. En esta arquitectura, en lugar de utilizar una única red por cada capa, se tienen 4 redes diferentes que interactúan entre sí. Como se puede ver en la figura 2.2.4, las redes LSTM presentan dos entradas y dos salidas. La primer entrada (la que se encuentra en la parte superior del bloque) se corresponde con el **estado de la celda** C_{t-1}. La segunda entrada (la de la parte inferior) se corresponde con el **forget gate** h_{t-1}, la cual indica cuanto peso deben darse a los estados pasados: Si $h_{t-1} = 1$ significa que el estado debe mantenerse, mientras que si $h_{t-1} = 0$ el estado debe ser descartado. A la salida de la red LSTM se tienen nuevamente el estado de la celda C_t y el forget gate h_t pero para el instante siguiente. Al combinar el estado de la celda, el valor del forget gate y la entrada en dicho instante a la red, se actualizan las salidas para el instante siguiente.

Otra arquitectura que se encuentra entre las mas usadas es la de las **redes GRU (Gated Recurrent Unit)**, las cuales se diferencian de las LSTM en que el forget gate y la entrada se encuentran fusionadas en una única entrada llamada **update gate**.

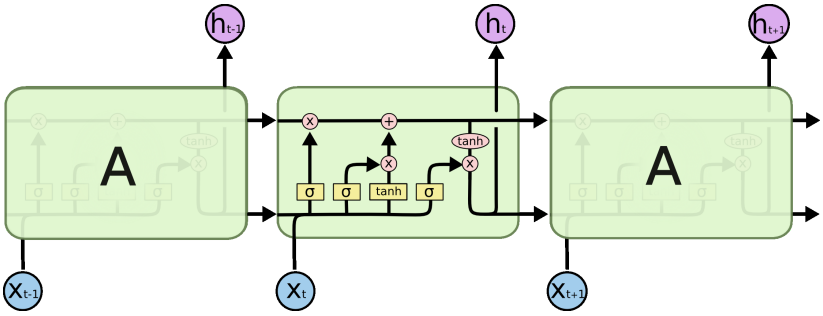


Figura 2.2.4: Red recurrente LSTM desdoblada a lo largo del tiempo.

Dado que en este trabajo se desean analizar señales de habla, las cuales presentan una gran dependencia temporal, son las redes recurrentes el candidato ideal para lograrlo. Sin embargo, este tipo de redes presentan cierto tipo de problemas a la hora de ser puestas en práctica, los cuales serán descritos en las secciones siguientes.

2.2.4. Underfitting y overfitting

Uno de los principales problemas que presentan las redes neuronales es el **overfitting**. Este problema se evidencia en los casos en que la capacidad de aprendizaje de la red es lo suficientemente grande como para memorizar todos los ejemplos de entrenamiento. De esta manera, al realizar la validación, se observan resultados muy alejados de los obtenidos durante el entrenamiento.

Otro problema importante es el de **underfitting**, el cual ocurre cuando la capacidad de aprendizaje de la red es demasiado pequeña para poder aprender los ejemplos de entrenamiento. En estos casos se observa que sin importar cuánto tiempo se entrene la red, ésta no logrará reducir su error de predicción. En la figura 2.2.5 se presentan ambos problemas como ejemplos de polinomios ajustando puntos.

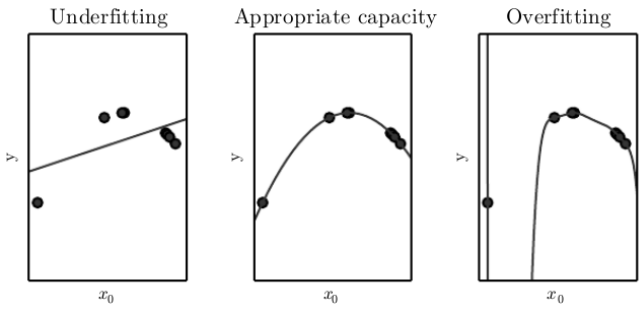


Figura 2.2.5: Problema de underfitting y overfitting.

Existen distintas maneras para solucionar estos problemas, algunas de las cuales son:

- Para evitar el problema de underfitting lo que se debe hacer es incrementar la capacidad de la red, ya sea agregando más capas o más neuronas por capa. Por otra parte, se debe tener en cuenta el problema de vanishing gradient. Este problema ocurre al propagar las actualizaciones de los pesos de la red, y al tener muchas capas los gradientes se hacen demasiado pequeños para ser representados por una computadora (durante la propagación se realizan multiplicaciones entre gradientes). Este problema puede evidenciarse como una red incapaz de ser entrenada, y confundirse con un problema de underfitting. En estos casos es necesario o bien reducir la cantidad de capas de la red, o aplicar estrategias como **residual networks**, cambios en las funciones de activación, etc.
- El problema de overfitting en general suele ser más difícil de solucionar, y las principales estrategias a plantear son:
  - Reducción de la capacidad de la red mediante la eliminación de capas ocultas o de la reducción de la cantidad de neuronas por capa.
  - Incremento del dataset de entrenamiento. Al hacer esto, no solo se evita que la red tenga la capacidad suficiente para memorizar las muestras de entrenamiento, sino que también se tiene un espacio de entrenamiento más representativo del problema a solucionar (asumiendo que las muestras sean representativas).
  - **Regularización**: Como se mencionó antes, en muchos casos las redes tienen la capacidad suficiente para memorizar las muestras. Esto puede también observarse al analizar los pesos aprendidos por las distintas neuronas,

donde podrá verse que determinadas neuronas presentan valores de activación muy altos, mientras otras se hacen casi nulas. El objetivo de la regularización es el de permitir que la red "reparta" el conocimiento en todas las neuronas, haciendo que sus pesos presenten valores similares. Para hacer esto se introducen penalizaciones a las neuronas de pesos grandes mediante términos que se suman a la función de costo de la red.

- Utilización de subgrupos o **batches** de entrenamiento. En lugar de calcular el costo de cada una de las muestras en cada iteración de entrenamiento, lo que se hace es tomar subgrupos de muestras de entrenamiento, se ejecuta la red en paralelo para todas las muestras del subgrupo, y se computa el costo como el promedio del costo de cada una de ellas. De esta manera, en lugar de aplicar backpropagation por cada muestra, se aplica por cada subgrupo de muestras, evitando así que la red ajuste sus pesos para cada muestra específica. Otra ventaja de este método es que permite paralelizar operaciones (siempre y cuando se tenga una GPU), y así acelerar los tiempos de entrenamiento.
- **Batch normalization**: Consiste en restar la media y dividir por el desvió del batch. De esta manera permite que las distintas capas se comporten de manera más independiente (se evita tener entradas o salidas con pesos muy diferentes), y se logra ayudar a la regularización.
- **Dropout**: Esta técnica consiste en eliminar distintas conexiones entre neuronas en cada iteración de entrenamiento. En general se define una probabilidad de que una conexión sea eliminada, y en cada iteración se eliminan al azar dichas conexiones. De esta manera, al entrenar la red, lo que se está haciendo es entrenarla con redes incompletas que deben actualizar sus pesos sin tener en cuenta las conexiones eliminadas. Por lo tanto, se logra que los pesos obtenidos se encuentren repartidos por la red, y el aprendizaje no se concentre en un subgrupo de neuronas.

2.2.5. Funciones de activación

Dependiendo del tipo de problema se deben utilizar distintas funciones, siendo las más comunes las siguientes:

- **Lineal**: Básicamente consiste en aplicar la función identidad. Puede tomar valores entre $-\infty$ y ∞.
- **Sigmoidea**: La función es

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

  Se puede ver que su salida varía entre 0 y 1, y lo hace de manera suave. Una ventaja de esta función es que su derivada puede ser escrita a partir de ella misma, lo cual facilita la aplicación de backpropagation.

- **Tanh**: Es una función similar a la sigmoidea, pero que en general demostró ser más eficaz. Se encuentra definida como $\sigma(x) = \tanh(x)$, y se trata de una función que varía entre -1 y 1, y también tiene ventajas en cuanto a la expresión de su derivada.
- **ReLU** (Rectified Linear Unit): Básicamente consiste en $\sigma(x) = \text{máx}(0, x)$, es decir que se trata de una función de activación lineal para los $x \geq 0$ y arroja un valor nulo para $x < 0$. Esta función se encuentra ampliamente utilizada en las redes convolucionales.
- **Softmax**: Consiste en

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \quad \forall j \in \{1, \ldots, K\}$$

  Se puede ver que sus valores se encuentran acotados entre 0 y 1, y que se aplica en casos donde se tienen únicamente K posibles salidas. Por otra parte, se puede ver que al sumar todas las posibles salidas se obtiene la unidad. Ésta función puede ser interpretada como la probabilidad sobre K diferentes clases, y por esto es que suele ser utilizada en problemas de clasificación, indicando como más probable la clase correcta.

2.3. Connectionist Temporal Classification (CTC)

Se trata de una función de costo introducida por Alex Graves en [12] en el año 2006, que permite entrenar redes neuronales del tipo sequence-to-sequence sin la necesidad de utilizar datos alineados. A diferencia del clasificador de fonemas presentado en este trabajo, al utilizar CTC no se requiere utilizar secuencias alineadas para el entrenamiento, y por ende es posible utilizar como entrada las señales de audio (o en todo caso algún tipo de transformación de éste, como un espectrograma o MFCCs), y como target para el entrenamiento las transcripciones no alineadas de dichos audios.

La gran ventaja de este enfoque es que permite reducir la complejidad del armado de los sets de entrenamiento y validación, ya que no hace falta el etiquetado para cada instante de tiempo. Por otra parte, otra ventaja frente a los

modelos tradicionales (HMM), es que no se requiere ningún tipo de presunción a priori sobre el problema que se desea modelar.

A continuación se explicará el fundamento teórico de esta función de costo.

2.3.1. Base teórica

El problema que se desea resolver es el de poder traducir la secuencia $\mathbf{x} \in \mathbb{R}^{m \times T}$, donde m es la cantidad de features por cada instante t y T es la duración temporal de la secuencia, a la $\mathbf{z} \in \mathbb{R}^n$ que representa una secuencia de clases de longitud n. En principio ambas secuencias no se encuentran alineadas, y este algoritmo puede ser aplicado siempre y cuando la secuencia de salida sea del mismo largo o menor que la de entrada: $n \leq T$.

Dejando en principio de lado la función de costo CTC, una red neuronal del tipo sequence-to-sequence lo que hace es traducir una secuencia de largo T a otra del mismo largo, pero codificada de distinta manera. Por lo tanto, la salida de esta red será del tipo $\mathbf{y} = \mathcal{N}_w(\mathbf{x})$ donde $\mathcal{N}_w : \mathbb{R}^{m \times T} \rightarrow \mathbb{R}^{p \times T}$.

Se debe aclarar que para una secuencia de salida de p clases, éste algoritmo requiere la introducción de una nueva clase **blank**, con la cual es posible realizar la separación entre estimaciones sucesivas iguales, pero que en algunos casos corresponden a repeticiones de las clases y en otros a una misma clase extendida en el tiempo. Siendo L el conjunto de clases posibles, entonces se puede definir como $L' = L \cup \{\text{blank}\}$.

Se puede denotar entonces a la salida de la red como y_k^t, donde k es la clase y t el instante de tiempo de la secuencia de salida. Ésta salida y_k^t puede ser interpretada entonces como la distribución de probabilidades de las p clases por cada instante t de tiempo[3]:

$$p(\pi|\mathbf{x}) = \prod_{t=1}^{T} y_{\pi_t}^t \quad \forall \pi \in L'^T$$

El objetivo de CTC entonces es traducir éstas salidas a la secuencia de salida, es decir $L'^T \rightarrow L'^{\leq T}$. Para hacer esta traducción lo que se hace es simplemente eliminar todas las clases correspondientes a **blank** y se agrupan los labels repetidos. Se puede definir entonces $\mathcal{B}$ como la función que permite mapear las secuencias o paths π a las secuencias reducidas: $\mathcal{B}(a - ab-) = \mathcal{B}(-aa - -abb-) = aab$. Utilizando esta función de mapeo, se puede definir la probabilidad condicional de la secuencia de salida $l \in L^{\leq T}$ como la suma de probabilidades de todos aquellos paths que se corresponden a un mismo label:

$$p(l|\mathbf{x}) = \sum_{\pi \in \mathcal{B}^{-1}(l)} p(\pi|\mathbf{x})$$

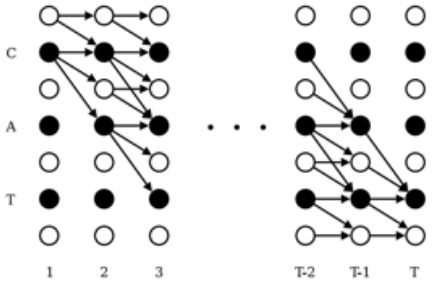


Figura 2.3.1: Decodificación de una secuencia de entrada mediante CTC.

Por lo tanto, la salida del clasificador se debe corresponder con aquella secuencia que maximiza la probabilidad:

$$h(\mathbf{x}) = \arg \max_{l \in L^{\leq T}} p(l|\mathbf{x})$$

Una manera eficiente para hallar las probabilidades $p(l|\mathbf{x})$ es utilizando el algoritmo forward-backward:

$$\begin{cases} \alpha_t(s) = \sum_\pi \prod_{t'=1}^{t} y_{\pi_{t'}}^{t'} \\ \beta_t(s) = \sum_\pi \prod_{t'=t}^{T} y_{\pi_{t'}}^{t'} \end{cases}$$

Se sabe que éstas expresiones puede ser definidas de manera recursiva. Se sabe que mediante el algoritmo forward-backward, es posible definir la probabilidad de todas las secuencias l' que definen al símbolo s en el instante t como:

$$\alpha_t(s)\,\beta_t(s) = \sum y_{l'_s}^t \prod_{t=1}^{T} y_{\pi_t}^t = \sum y_{l'_s}^t\, p(\pi|\mathbf{x})$$

Se puede demostrar que de esta expresión es posible despejar:

$$\boxed{p(l|\mathbf{x}) = \sum_{s=1}^{|l'|} \frac{\alpha_t(s)\,\beta_t(s)}{y_{l'_s}^t}}$$

[3]Para esto se utilizan salidas softmax.

De esta manera, maximizando el likelihood $\ln\left(p\left(l|\mathbf{x}\right)\right)$ es posible encontrar la secuencia de salida para la secuencia de entrada $\mathbf{x}$.

Resumen: En resumen, el método aplicado por CTC puede visualizarse en la figura 2.3.1. Cada columna corresponde con un instante de tiempo. De esta manera, se tienen múltiples caminos o *paths* posibles para la secuencia deseada. Dado que las secuencias predichas presentan los elementos blank, y que existen repeticiones, no existe un único path correcto. Por lo tanto, lo que se hace es sumar las probabilidades de cada uno de los paths que se encuentran asociados a un mismo label, y así se obtiene la probabilidad de cada uno de los posibles labels. Aquel label que presente una mayor probabilidad de ocurrencia, es elegido como el ganador. El cálculo de las probabilidades de las transiciones puede hacerse de distintas maneras, pero mediante el algoritmo forward-backward esta tarea puede realizarse de manera eficiente.

3. Desarrollo

A continuación se presentan dos enfoques diferentes para el reconocimiento del habla mediante redes neuronales. En el primer caso, se presenta un clasificador de fonemas basado en [1], donde se utilizan redes recurrentes para encontrar el fonema emitido en cada instante de tiempo. Si bien esta técnica ya no se encuentra utilizada, debido a la aparición de nuevos métodos, presenta un buen caso de estudio para introducirse en las redes recurrentes.

Por otra parte, se investiga también un método presentado en [3] para trabajar con secuencias de datos sin etiquetar. Dicho método, conocido como CTC, evita tener que etiquetar instante a instante el fonema o caracter pronunciado, lo cual reduce ampliamente el tiempo requerido para el preprocesamiento.

Para ambos casos se utilizó la base de datos TIMIT [4], la cual consiste en archivos de audio `.wav`, en las transcripciones fonéticas `.phn`, y en las transcripciones textuales `.txt` (no alineadas con los audios).

3.1. Clasificador de fonemas

3.1.1. Generación de los datos

Como se mencionó anteriormente, la base de datos de TIMIT consiste en audios y transcripciones fonéticas alineadas con dichos audios. Un proceso muy importante para el entrenamiento de redes neuronales es el correcto acondicionamiento de los datos de entrenamiento. Si bien las redes neuronales son capaces de encontrar representaciones o features de manera más eficiente que los humanos, en muchos casos una incorrecta representación (o mucha redundancia) puede llevar a que se deban recurrir a estrategias más costosas computacionalmente (como por ejemplo incrementar el tamaño de la red), o que problemas como el **overfitting** resulten más evidentes. Por lo tanto, es esencial que los datos de entrenamiento no solo sean una clara representación del problema que se desea modelar, sino que también se los exprese de manera tal que se les pueda extraer la mayor información posible al menor costo.

Como se mencionó en la introducción, cuando se trabaja con señales de habla existen distintas maneras de representar los datos. Una primer forma es introduciendo las señales de habla de manera directa a la red. La desventaja de este método es que las grabaciones presentan, además de la señal de habla que se desea analizar, ruidos de fondo, alinealidades del micrófono, diferencias por la directividad del micrófono, variaciones en el volumen, etc. Si bien es probable que la red pueda encontrar una representación que ajuste bien a los datos de entrenamiento, al proceder con la validación se observará que la generalización del modelo será pobre debido a causas que no se encuentran relacionadas con el habla en sí, sino que con problemas externos.

Para reducir este problema se suelen aplicar transformaciones a las señales de habla. Una muy común consiste en representar a las señales de audio mediante **espectrogramas**. El cálculo de un espectrograma consiste en un proceso de ventaneo sobre la señal a procesar, para luego aplicar una FFT a cada una de las ventanas. Como resultado se obtiene una matriz tridimensional donde uno de los ejes es la frecuencia, otro es el tiempo, y otro es la intensidad asignada a cada frecuencia en cada instante de tiempo. Una de las ventajas de esta representación es que permite separar el ruido de la señal útil, ya que el primero suele encontrarse a frecuencias superiores, y por ende le es más sencillo a la red aprender a ignorarlo.

Sin embargo, durante este trabajo se observó que los resultados obtenidos al utilizar espectrogramas para representar a la información de habla no fueron los esperados. Una posible causa de esto es que en las señales de habla un mismo fonema puede ser expresado en todos más graves o agudos sin modificarse. Sin embargo, ésta ambigüedad en la representación puede provocar que el entrenamiento de la red alcance un estado intermedio que minimice el error en un mínimo local, y por ende no se logre alcanzar el nivel de inferencia deseado.

El otro tipo de representación posible es mediante los coeficientes MFCC. Éstos coeficientes permiten representar a las señales de habla de una manera mucho más similar a cómo es percibido el sonido por el sistema auditivo. En este trabajo se encontró que los resultados obtenidos mediante esta representación fueron ampliamente superiores a los de las demás representaciones.

Para el cálculo de los MFCC se utilizó la librería de Python `python_speech_features`[7], mediante la cual se pueden configurar filtros de preamplificación, rangos de frecuencias, número de puntos para la FFT, etc. Por lo tanto lo que se hizo fue tomar cada una de las señales de audio, calcular sus coeficientes MFCC (resultando en matrices tridimensionales donde uno de los ejes era la cantidad de coeficientes, otro el tiempo, y otro el valor de los coeficientes), y almacenar éstos resultados en una base de datos.

Otro parámetro importante para el entrenamiento es el etiquetado de los audios. El dataset de TIMIT presenta archivos `.phn` que indican para cada muestra de los archivos de audio cuál es el fonema que está siendo articulado. De acuerdo con esta base de datos, se tienen 63 posibles fonemas[4] posibles. Dado que para el cálculo de los MFCC se realiza un proceso de ventaneo, es necesario aplicar un proceso similar a los fonemas para mantener la alineación. Por lo tanto, lo que se hizo fue aplicar una serie de ventanas a los fonemas, siguiendo el mismo formato que para los MFCC, y se eligió mediante algún criterio cuál era el fonema que debía corresponder a dicha ventana.

[4] Dentro de estos fonemas se encuentran combinaciones de varios fonemas y símbolos sin señal de habla como `h#`.

El proceso de generación y acondicionamiento de los datos de entrenamiento se realizó mediante el script `GenerateDatabase.` que se encuentra en la carpeta de `phoneme_classificator` en [6].

3.1.2. Modelo de red

Dado que los datos de entrenamiento son secuencias de MFCC, la arquitectura correcta para trabajar resulta la de las redes recurrentes. El modelo de la red implementado se encuentra parametrizado, de manera tal de que es posible modificar la arquitectura simplemente modificando determinados argumentos. La arquitectura base es la que se muestra en la figura 3.1.1, la cual presenta los siguientes componentes:

- Los placeholders de entrada son `input_feature` (por el cual se ingresa la secuencia de coeficientes MFCC), `input_context` (el cual provee información sobre la cantidad de coeficientes MFCC y el largo de la secuencia ingresada), y durante el entrenamiento también se tiene el placeholder `input_labels` que es el que contiene los valores reales de la clasificación.
- La capa llamada `input_dense` es una FCL que se encuentra parametrizada de manera que se puedan modificar la cantidad de capas, la cantidad de neuronas por capa, la función de activación, el coeficiente de regularización, etc. El objetivo de estas FCL antes de la capa recurrente es el de poder realizar pruebas permitiendo que la red encuentre su propia representación interna de los datos.
- El bloque `RNN_cell` es la red recurrente propiamente dicha. Dentro de ésta se encuentran las celdas LSTM o GRU, donde no solo es posible modificar la cantidad de celdas, sino que se puede configurar si se desea utilizar una red recurrente unidireccional o bidireccional.
- El bloque `dense_layers`, al igual que `input_dense`, está compuesto por capas FCL que pueden ser modificadas. Estas capas permiten un mayor grado de abstracción para la clasificación.
- El bloque `dense_output` es la capa encargada de la clasificación. La cantidad de neuronas se encuentra fija en 63 (una por cada clase), y la función de activación de esta capa es `softmax`. Sin embargo, dado que la función de costo utilizada ya aplica internamente ésta función de activación [9], se agregó un bloque llamado `dense_output_no_activatio` cuya salida es igual a la de `dense_output` con la diferencia de que aplica una función de activación lineal.
- Los bloques de `loss` y `accuracy` permiten medir el rendimiento de la red, donde el primero utiliza la función de costo `cross_entropy` para medir el nivel de clasificación alcanzado más términos de regularización sobre los pesos de la red, y el bloque de `accuracy` simplemente compara frame a frame la secuencia predicha y la real para dar una medida del grado de predicción alcanzado.

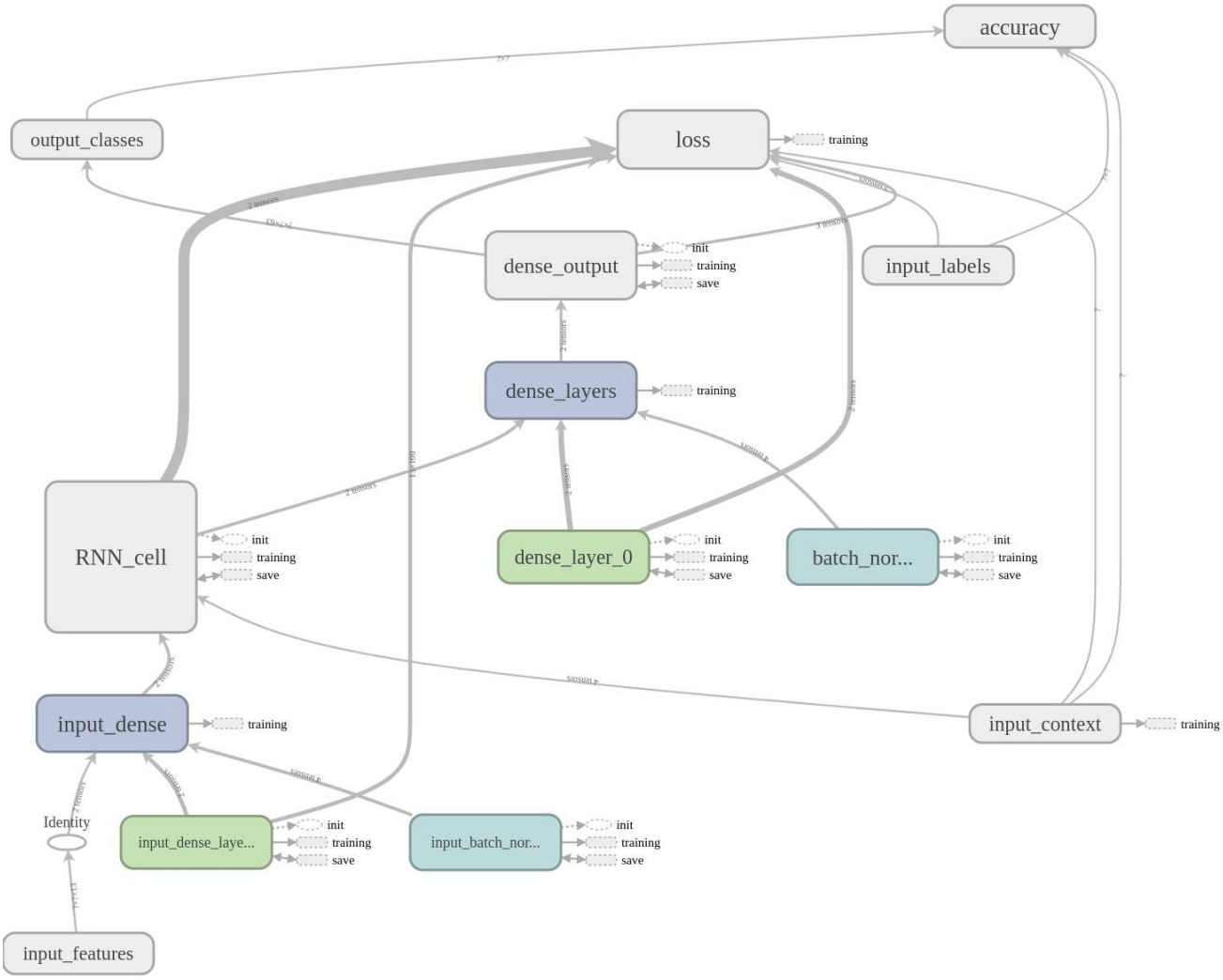


Figura 3.1.1: Grafo de la red neuronal utilizada para el clasificador de fonemas, el cual fue obtenido mediante Tensorboard [8].

Algunos de los inconvenientes encontrados a la hora de plantear este modelo fueron:

- Si bien las redes recurrentes presentan un parámetro de entrada que indica la longitud de la secuencia, trabajar con señales de habla no es sencillo. En este trabajo se utilizaron más de 4000 audios de la base de datos TIMIT, dentro de los cuales las longitudes son altamente variables. Como se mencionó anteriormente, una técnica muy utilizada para acelerar los procesos de entrenamiento y al mismo tiempo mejorar la generalización de la red, es entrenar mediante batches de muestras. El problema encontrado con trabajar con señales de longitud variable es que no es posible agruparlas en un único grupo de muestras. Por lo tanto, se tomaron las siguientes acciones:
  - Se realizaron entrenamientos con batches de tamaño 1, es decir de a una muestra de habla por vez. La desventaja de este método es el bajo nivel de generalización alcanzado, y el alto incremento en los tiempos de entrenamiento al aumentar la cantidad de muestras.
  - Se completaron los audios con la clase `h#` en el final de todas las secuencias, de manera que la señal de habla no se viera modificada. El problema encontrado fue que debido a la gran diferencia de longitudes de los audios, el mínimo local encontrado por la red era arrojar como salida siempre la clase `h#`.
  - Se realizó una estadística de las longitudes de los audios, y se tomaron audios que se encontraran dentro de un rango aceptable de longitudes para completarlos nuevamente. Si bien los resultados fueron mejores que en el caso anterior, no fueron lo suficientemente buenos.
  - Se aplicó un procesamiento previo donde se ordenaron los audios de menor a mayor con respecto a su longitud, se formaron batches de distintas longitudes, y dentro de cada batch se completaron a los audios más cortos para que tuvieran la misma longitud que los demás audios del batch. De esta manera se redujo la cantidad de redundancia agregada, y se tuvieron batches de longitudes distintas pero cada uno con muestras de igual

longitud. Los resultados fueron mejores que en los casos anteriores debido a un incremento en la generalización y una reducción en el tiempo de entrenamiento. La desventaja de este método es que requiere una gran cantidad de tiempo de procesamiento previo, ya que se deben ordenar listas de grandes cantidades de datos, modificar todos los elementos, etc. Otra desventaja encontrada es que no este método no permite reordenar las muestras de manera completamente aleatoria, lo cual también perjudica a la generalización.

- Finalmente se optó por la opción más sencilla de aplicar. Dado que inicialmente se utilizaron redes recurrentes unidireccionales, es decir que solo utilizan información pasada y presente para hacer predicciones, el hecho de recortar los audios en un determinado instante de tiempo no afecta a las predicciones previas. Por lo tanto, lo que se hizo fue escoger una longitud tal que a los audios con una longitud menor no se les requiriera agregar demasiada redundancia, y a los demás audios se los recortó a dicha longitud. Los resultados obtenidos mediante este método son los que se presentan a continuación.
  Éste método resultó útil para el caso de las redes unidireccionales. Por otra parte, al utilizar las redes bidireccionales, realizar un recorte abrupto completamente descorrelacionado del resto de la secuencia puede perjudicar el proceso de aprendizaje de la red. Sin embargo, se observó que los resultados obtenidos con redes bidireccionales resultaron incluso mejores que con las unidireccionales.

- Un problema muy común y complicado de solucionar que tienen las redes recurrentes es el overfitting. Para reducir este problema se implementaron dos soluciones:

  - Se agregaron términos de regularización a la función de costo. Además de computar que tan alejadas se encuentran las predicciones de los valores reales, se tomaron los valores de los pesos de las neuronas de las capas FCL y de las celdas RNN por separado, se los multiplicaron por coeficientes parametrizables, y todos éstos productos fueron sumados al costo del entrenamiento. De esta manera, además de reducir el costo de la predicción, y por ende aumentar la precisión de la red, se impusieron penalizaciones a aquellas neuronas que presentaran pesos de valores elevados. Este tipo de regularización permite que el "conocimiento" se distribuya a lo largo y ancho de la red, en lugar de concentrarse en un pequeño grupo de neuronas (*el comunismo de las redes neuronales*).
  - Por otro lado, se agregaron capas de **batch normalization** a todas las FCL.
  - Se agregaron capas de dropout. Mediante esta técnica se logró distribuir el conocimiento a través de la red, evitando que las predicciones dependan de un único camino a través de la red.

- Otro problema importante es el de hallar los hiperparámetros de la red. Inicialmente éstos fueron elegidos a mano, observando qué tan buenos los resultados eran. En embargo, hacerlo de esta manera es un proceso lento y problemático, ya que requiere múltiples entrenamiento de la red y de análisis de los parámetros internos (magnitud de los pesos, resultados de validación, etc.). Por lo tanto, luego de varias iteraciones a mano sin lograr superar el 68 % de accuracy, se decidió utilizar un algoritmo de optimización Bayesiano para que se encargue de hallar los mejores hiperparámetros. El algoritmo utilizado fue SMAC3 [10], donde se debe definir un subespacio de hiperparámetros a recorrer, se define una función objetivo que se desea minimizar, y dicho algoritmo se encarga de ajustar los hiperparámetros automáticamente. Los pasos seguidos fueron:

  - Dentro del subespacio de hiperparámetros a recorrer se agregaron parámetros como la cantidad de capas tanto antes como después de la capa recurrente, la cantidad de neuronas por capa, la cantidad de celdas de la red recurrente, los valores de regularización, la probabilidad de eliminar una neurona al utilizar dropout, etc.
  - La función objetivo definida consistió en realizar múltiples entrenamientos sobre el set de entrenamiento, y luego realizar una validación sobre el set de validación. Obteniendo el costo de la validación (sin los términos de regularización), se lo debe retornar a la librería como parámetro a minimizar. De esta manera, la librería minimizará el costo de la validación de la red, y por ende maximizará el accuracy.

3.1.3. Resultados

Luego de múltiples corridas del algoritmo de optimización, se obtuvieron los siguientes parámetros:

- FCL de entrada: `[250, 150]`. La función de activación fue `tanh`. Los coeficientes para dropout fueron $0{,}8$ para ambas capas.
- Red recurrente bidireccional: `bw=[150, 30]`, `fw=[240, 160]`. La función de activación fue `tanh` y las celdas eran LSTM.
- FCL de salida: `[150]`. La función de activación fue `tanh`. El coeficiente de dropout fue $0{,}8$.
- El término de regularización para las FCL fue $0{,}3$, mientra que para la capa recurrente fue de $0{,}09$.

- La cantidad de coeficientes MFCC fue de 26.
- Para la optimización se utilizó el algoritmo ADAM, con un learning rate inicial de $0{,}001$ y un épsilon de $0{,}005$.

Utilizando estos hiperparámetros se obtuvo un porcentaje de entrenamiento del $83{,}2\,\%$ y uno de validación del $71{,}5\,\%$, luego de entrenar durante 100 epochs.

3.1.4. Análisis de resultados

Se puede ver que los resultados obtenidos resultan mejores que en [1], donde el máximo porcentaje de validación obtenido fue del $70{,}2$. Una diferencia entre ambos modelos se debe a que en la red aquí presentada se introducen capas FCL previas a la red recurrente. Esto permite que la red pueda hallar una nueva representación de los datos de entrada que puede arrojar mejores resultados que la utilizada manualmente (en este caso MFCC). Por otra parte, en dicho trabajo se utilizaron funciones de activación sigmoideas en lugar de `tanh`, las cuales demostraron en el último tiempo resultar mucho más eficaces. Finalmente, cabe mencionar que en este trabajo se utilizó un algoritmo de optimización de hiperparámetros, lo cual permite explotar a mayor profundidad la capacidad de la red.

Si bien los resultados obtenidos fueron satisfactorios, en el trabajo [11] se presentan algunas modificaciones que permiten tener mejores resultados. Una posible modificación es la de reducir la cantidad de símbolos para los fonemas a 39 clases. Mediante esta modificación en dicho trabajo lograron incrementar de $68\,\%$ a $75\,\%$ el accuracy alcanzado. Otra modificación importante que introducen es la de modificar el conexionado interno de las redes recurrentes, logrando un resultado de validación del $77\,\%$ para 61 fonemas y $82\,\%$ para 39.

3.2. Reconocedor de habla continua

Para este reconocedor se hizo uso de la función de costo CTC, de manera de poder entrenar a la red neuronal con secuencias no alineadas. Mediante esta función de costo es posible armar un clasificador de fonemas como el presentado previamente, pero sin la necesidad de alinear los fonemas con la señal de audio. Sin embargo, dado que se tienen las transcripciones de los audios, se optó por utilizar como salida caracteres normales, de manera que el reconocedor tuviera una utilidad práctica mayor. De esta forma, a continuación se presenta una red neuronal capaz de traducir señales de audio a texto.

3.2.1. Generación de los datos

Al igual que en el clasificador de fonemas, se optó por representar a las señales de audio mediante sus MFCCs, ya que se observó que resultaron mas eficientes que la utilización de espectrogramas.

Por otra parte, se modificó la base de datos para utilizar transcripciones de texto en lugar de secuencias de fonemas, por lo que en este caso no se requirió realizar operaciones de ventaneo o alineación. Sin embargo, se debieron eliminar símbolos como las comillas, comas, guiones bajos y medios, etc., ya que no aportaban información necesaria para la tarea de reconocimiento[5]. La cantidad de clases entonces se reduce a las letras del abecedario ingles (26 caracteres) más una clase para el espacio en blanco y otra para el **blank**, por lo que se tienen 28 posibles clases.

3.2.2. Modelo de red

Para esta red lo que se hizo fue reutilizar la red del clasificador de fonemas, pero adaptando la etapa de salida para la función de costo CTC. En la figura 3.2.1 se presenta un esquema de la red utilizada. Se puede ver en dicha figura que los bloques diferentes a los de la figura 3.1.1 se corresponden con la salida, donde se aplica CTC para las secuencias no alineadas. Los nuevos bloques entonces se pueden detallar como:

- El bloque `loss` se corresponde con la función de costo CTC. TensorFlow ya tiene implementada esta función de costo, llamada `tf.nn.ctc_loss`, la cual recibe la secuencia de labels, las probabilidades de cada clase en cada instante de tiempo, y la longitud de la secuencia de features. Dado que esta función de costo ya aplica internamente la función de activación `softmax`, nuevamente se debieron utilizar los bloques `dense_output` y `dense_output_no_activation` para diferenciar las operaciones que requieren determinadas funciones de activación.
- El bloque `decoder` es el encargado de encontrar la secuencia más probable de salida. TensorFlow ofrece dos posibilidades para esta capa: `tf.ctc_greedy_decoder` y `tf.ctc_beam_search_decoder`. La estrategia greedy es un caso particular de la otra, y realiza **best path decoding**, la cual resulta mucho más rápida que `tf.ctc_beam_search_decoder`.
- El bloque `label_error_rate` (LER) es el encargado de medir el error entre la secuencia predicha y la deseada. Para esto se utiliza la distancia conocida como **edit distance**, que básicamente consiste en la cantidad de operaciones que deben hacerse sobre una secuencia para llegar a la secuencia deseada.

[5] Las comas y puntos sí influyen en el reconocimiento, ya que se evidencian mediante pausas o entonaciones. Sin embargo, se optó por no utilizarlos para no incrementar la complejidad del problema.

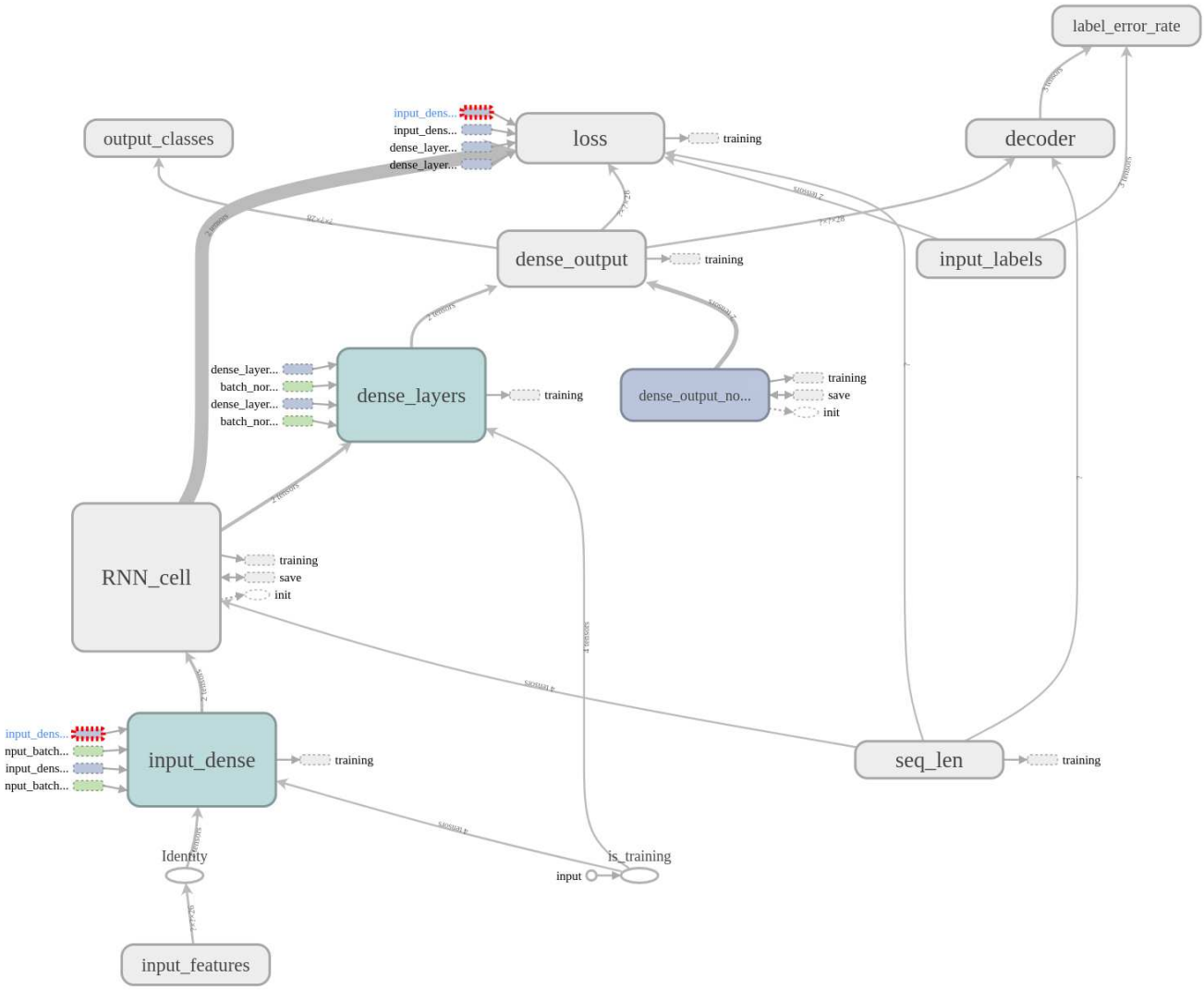


Figura 3.2.1: Grafo de la red neuronal utilizada para el reconocedor de habla continua, el cual fue obtenido mediante Tensorboard [8].

Al implementar esta red se encontraron problemas muy similares a los del caso anterior. Por lo tanto, se debieron aplicar las mismas técnicas de regularización y normalización aplicadas previamente para reducir el problema de overfitting, el cual en este caso resultó mucho más evidente debido a la reducción en la longitud de las secuencias.

Por otro lado, se observó un incremento importante en los tiempos de cómputo. Es por esta razón que no pudo ser utilizado el decodificador `tf.ctc_beam_search_decoder`, ya que éste resultaba mucho más lento que la estrategia greedy. Debido a esto, no pudo hacerse una comparación entre los resultados obtenidos con cada una de las estrategias.

3.2.3. Resultados

A diferencia del trabajo anterior, en este caso se utilizaron los audios completos, y por ende los tiempos de entrenamiento se incrementaron considerablemente. A pesar de haber dejado la computadora corriendo el algoritmo de optimización durante varios días, solo se logró explorar 28 combinaciones de hiperparámetros. Los mejores hiperparámetros encontrados permitieron alcanzar un $10{,}2\,\%$ de LER para el entrenamiento y $31{,}8\,\%$ de LER para la validación. Los parámetros utilizados en este caso fueron:

- FCL de entrada: `[150]`. La función de activación fue `tanh`. Los coeficientes para dropout fueron $0{,}6$ para ambas capas.
- Red recurrente bidireccional: `bw=[250]`, `fw=[110]`. La función de activación fue `tanh` y las celdas eran LSTM.
- FCL de salida: `[75, 180]`. La función de activación fue `tanh`. El coeficiente de dropout fue $0{,}6$.
- El término de regularización para las FCL fue $0{,}3$, mientra que para la capa recurrente fue de $0{,}9$.

- La cantidad de coeficientes MFCC fue de 26.
- Para la optimización se utilizó el algoritmo ADAM, con un learning rate inicial de $0{,}001$ y un épsilon de $0{,}005$, y se realizaron 200 epochs de entrenamiento.

En la figura 3.2.2 se presentan algunos ejemplos de las predicciones realizadas por la red.

```
Predicted: bob foun mor clams the ocens ege
Target: bob found more clams the oceans edge

Predicted: an adult male babons teeth ere not sutable for eaing shelfish
Target: an adult male baboons teeth are not suitable for eating shellfish

Predicted: she had your dark suit in greasy wash water all year
Target: she had your dark suit in greasy wash water all year

Predicted: a lonestar shone in the erly eveing sk
Target: a lone star shone in the early evening sky

Predicted: ow about me tryn to helpp er gether job bak
Target: how about me trying to help her get her job back

Predicted: rockandrol music has a great rhetom
Target: rockandroll music has a great rhythm

Predicted: she was so beautoful so vaianso pible
Target: she was so beautiful so valiant so pitiable

Predicted: he pickedup neing parof sos forech brother
Target: he picked up nine pairs of socks for each brother

Predicted: cunt the numbe of teaspns of soyausce that you ad
Target: count the number of teaspoons of soysauce that you add

Predicted: the halwayopens into a huge chamber
Target: the hallway opens into a huge chamber
```

(a) Ejemplo de predicciones para las muestras de **entrenamiento**.

```
Predicted: dont ask me to carry an oily rag like that
Target: dont ask me to carry an oily rag like that

Predicted: she had your dark suit in greasy wash water all year
Target: she had your dark suit in greasy wash water all year

Predicted: ghreis paranoy abot ha fhuc ure cas shortach
Target: george is paranoid about a future gas shortage

Predicted: iin or my mom
Target: i honor my mom

Predicted: this wis easy fois
Target: this was easy for us

Predicted: smash lightdbls ind har cash val uwil dimenish to neing
Target: smash lightbulbs and their cash value will diminish to nothing

Predicted: the ic olrse wis mnely obe wel by e thouts imd iese
Target: the thick elm forest was nearly overwhelmed by dutch elm disease

Predicted: the rose cresarge erlswek
Target: the rose corsage smelled sweet

Predicted: chocle ind orasis never filssiromanti cift
Target: chocolate and roses never fail as a romantic gift

Predicted: pourcy pines ressemle seurchins
Target: porcupines resemble sea urchins
```

(b) Ejemplo de predicciones para las muestras de **validación**.

Figura 3.2.2: Ejemplos de resultados de predicciones de la red.

3.2.4. Análisis de resultados

Se debe aclarar que los resultados obtenidos mediante esta red podrían haber sido mejorados simplemente explorando más el espacio de hiperparámetros, lo cual no se continuó haciendo debido al tiempo de ejecución requerido[6].

En la figura 3.2.2 se puede ver que si bien las predicciones no son perfectas, en la mayoría de los casos las pronunciaciones de las secuencias predichas resultan similares a las reales. Por otra parte, si bien la tasa de error obtenida en este caso es comparable con la del clasificador de fonemas[7], el preprocesamiento necesario para las muestras de entrenamiento es muchísimo menor, y el hecho de no requerir alineamiento permite que las muestras disponibles aumente considerablemente.

Otra observación a la figura 3.2.2 es que si bien en muchos casos las palabras formadas no parecieran tener sentido, las pronunciaciones de los caracteres predichos presentan alguna cercanía con la de los caracteres reales. Esto está indicando que el modelo de habla aprendido por la red, realmente logra correlacionar las diferentes pronunciaciones dependiendo del contexto en la oración, ya que en caso contrario se observarían muchos más caracteres aleatorios sin ningún tipo de relación con las palabras reales.

Finalmente cabe mencionar que en [3] se obtuvo un LER de $31{,}47\,\%$ sobre TIMIT al utilizar CTC mediante la estrategia best path decoding, la cual es la utilizada por TensorFlow en `tf.ctc_greedy_decoder`. Por lo tanto se logró reproducir los resultados de [3], aunque es probable que si se hubiera explorado más el espacio de hiperparámetros se hubiera logrado alcanzar mejores resultados.

4. Conclusiones

En este trabajo se abordó el problema reconocimiento de habla desde dos enfoques diferentes: muestras alineadas y no alineadas. En ambos casos la estructura central que permitió lograr éstos resultados es la de las redes recurrentes. Este tipo de redes demostraron ser eficaces en el análisis de secuencias, permitiendo que problemas donde la correlación temporal es predominante puedan ser resueltos de manera simple, sin la necesidad en muchos casos de introducir presunciones a priori sobre el modelo que se desea analizar. Éste resultado es muy importante, ya que si bien para el procesamiento de habla ya se disponen de modelos ampliamente investigados, existen muchas otras tareas donde la obtención de un modelo matemático descriptivo obliga a hacer simplificaciones a casos más simples y menos abarcativos.

[6] Para obtener estos resultados se realizaron 28 iteraciones sobre el espacio de hiperparámetros, lo cual llevó alrededor de 5 días de ejecución continua.

[7] Esta comparación no debe tomarse de manera literal, ya que en el clasificador de fonemas se contaron la cantidad de labels correctamente predichos como medida de error, mientras que en el reconocedor de habla continua se utilizó la métrica de LER.

Por otra parte, como puede observarse en [6], debido a la existencia de librerías de alto nivel es posible la implementación de algoritmos altamente complejos, tanto computacionalmente como en su diseño, de manera sencilla. Es gracias a este tipo de herramientas que el desarrollo de nuevas tecnologías basadas en redes neuronales puede ser encarado por el ambiente “no científico”, y de esta manera se logra también que se requiera menos tiempo de desarrollo y más tiempo en la búsqueda de soluciones al problema en cuestión.

A. Conversor de archivos de audio

Los archivos de audio `.wav` que se encuentran dentro de TIMIT se encuentran en un formato diferente al que se suele utilizar. Éstos audios presentan encabezados del formato SPHERE en lugar de NIST, que es el utilizado generalmente por los reproductores de audio y librerías de Python. Para esto fue necesario entonces utilizar el conversor `sph2pipe_v2.5` de [5], con el cual se escribió el siguiente script:

```
#!/bin/bash
# This scripts allows to convert the Sphere NIST WAV format used in the TIMIT
    dataset to normal WAV format

search_dir=wav
this_dir=FixWav

cd ..
for file in "$search_dir"/*
do
    echo "$file"
    ./"$this_dir"/sph2pipe -t : -f rif "$file" out.wav
    mv out.wav "$file"
done
```

B. Armado de datasets

Durante este trabajo se hicieron diferentes pruebas, donde muchas consistieron en ver los resultados ante variaciones en los datos de entrenamiento. Para poder agilizar esto, se escribieron dos scripts encargados de copiar archivos de la base de datos, con o sin repetición, de manera que no hubiera conflictos con los nombres de los archivos, etc. El siguiente script lo que hace es copiar archivos de la base de datos evitando que se repitan los audios, es decir que la base de datos se encontrará conformada únicamente por audios diferentes.

```
#!/bin/bash
# This scripts takes all files in the TIMIT database and copies them without
    repeating to another folder
search_dir=TRAIN
results_dir=results
mkdir "$results_dir"

for d in $(find "$search_dir/" -maxdepth 2 -type d)
do
    for file in "$d"/*
    do
        if [ ! -e "$results_dir/${file##*/}" ]
        then
            cp "$file" "$results_dir/"
                echo "$0: '${file}'"
        fi
    done
done
```

Por otra parte, el siguiente script copia todos los archivos de la base de datos, cambiando sus nombres para evitar que se sobrescriban. La base de datos entonces queda conformada por audios de las mismas frases pero pronunciados por personas diferentes.

```
#!/bin/bash
# This scripts takes all files in the TIMIT database and copies them to another
    folder
search_dir=TRAIN
results_dir=results
mkdir "$results_dir"
```

```
for d in $(find "$search_dir/" -maxdepth 2 -type d)
do
    for file in "$d"/*

    do
        extension="${file##*.}"
        filename="${file%.*}"
        folder="$(cut -d'/' -f3 <<<${filename})"

        cp "$file" "${filename}_${folder}.${extension}"
        mv "${filename}_${folder}.${extension}" "$results_dir/"
        echo "$0: '${file}'"
    done
done
```

Referencias

[1] Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures: ftp://ftp.idsia.ch/pub/juergen/nn_2005.pdf. 3, 3.1.4

[2] Understanding LSTM Networks: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[3] Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks ftp://ftp.idsia.ch/pub/juergen/icml2006.pdf. 3, 3.2.4

[4] TIMIT Acoustic-Phonetic Continuous Speech Corpus: https://catalog.ldc.upenn.edu/ldc93s1. 3

[5] SPHERE Conversion Tools: https://www.ldc.upenn.edu/language-resources/tools/sphere-conversion-tools A

[6] Repositorio de GitHub: https://github.com/nicozorza/speech-recognition. 3.1.1, 4

[7] python_speech_features: https://github.com/jameslyons/python_speech_features. 3.1.1

[8] TensorBoard: Visualizing Learning: https://www.tensorflow.org/guide/summaries_and_tensorboard. 3.1.1, 3.2.1

[9] tf.nn.softmax_cross_entropy_with_logits: https://www.tensorflow.org/api_docs/python/tf/nn/softmax_cross_entropy_with_logits 3.1.2

[10] SMAC3: https://github.com/automl/SMAC3. 3.1.2

[11] Bidirectional Recurrent Neural Networks: https://pdfs.semanticscholar.org/4b80/89bc9b49f84de43acc2eb8900035f7d492b2.pdf. 3.1.4

[12] Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks [Alex Graves]: https://www.cs.toronto.edu/~graves/icml_2006.pdf. 2.3

[13] Supervised Sequence Labelling with Recurrent Neural Networks: https://www.cs.toronto.edu/~graves/preprint.pdf