UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Síntesis de voz con deep learning

T E S I S

QUE PARA OBTENER EL TÍTULO DE

Ingeniero en Computación

PRESENTA

Emilio Alejandro Morales Juárez

DIRECTOR DE TESIS

Dr. José Abel Herrera Camacho

Ciudad Universitaria, Cd. Mx., 2019

Resumen

La síntesis de voz es una tarea imprescindible en la interacción usuario-máquina, con el objetivo de sintetizar audios que sean inteligibles y naturales, indistinguibles de las grabaciones humanas. En los últimos años, las técnicas de Deep Learning han obtenido importantes resultados en muchos campos. En la síntesis de voz, las redes neuronales se han propuesto para simplificar el proceso tradicional mediante la sustitución de la producción de características lingüísticas y acústicas con una sola red neuronal. En éste trabajo se exploran las capacidades de ésta disciplina para obtener la síntesis de voz del español hablado en la región central de México.

Agradecimientos

En primer lugar quiero agradecer a mi madre y a mi padre, por todo su apoyo durante mis estudios, a ellos les debo todo.

Agradezco a mi director de tesis Dr. José Abel Herrera Camacho, por todas sus recomendaciones, enseñazas y su incondicional apoyo al momento de realizar este trabajo. Al Mtro. Victor German Mijangos De La Cruz quien siempre tuvo una excelente disposición al momento de hacer observaciones y mejoras a los experimentos realizados, sin mencionar que su clase, una de las mejores que se imparten en la FI, fue un precedente para que realizara este trabajo.

Índice general

Índice de figuras

Índice de cuadros

Introducción

Traducir texto a señales de audio (síntesis de voz) es un proceso complejo y desafiante (Taylor, 2009), ya que el significado expresado por la pronunciación se encuentra inherentemente poco especificado por el texto (Skerry-Ryan et al., 2018). Además, a diferencia del reconocimiento de voz y la traducción de texto, las salidas de los sistemas *text-to-speech* (TTS) son continuas y generalmente mucho más largas que las entradas (Wang et al., 2017).

Existen dos objetivos principales en la síntesis de voz: inteligibilidad y naturalidad. La *Inteligibilidad* describe la claridad y nitidez del audio, específicamente que tan bien el mensaje puede ser extraído por el usuario. La *Naturalidad* describe la información no directa capturada por la inteligibilidad, como la consistencia estilística global, matices a nivel regional o lingüístico, entre otros (Sotelo et al., 2017).

Los métodos tradicionales de síntesis de voz se basan en etapas de procesamiento complejas. Usualmente se tiene un frontend que transforma el texto en representaciones lingüísticas, para después producir el sonido con un vocoder (Zen et al., 2009; Ze et al., 2013; Agiomyrgiannakis, 2015). Estos componentes se entrenan de forma independiente, son laboriosos de diseñar y su desarrollo requiere de una amplia experiencia.

Las técnicas de *deep learning* se han adoptado ampliamente en diversas aplicaciones del *aprendizaje máquina*. En la síntesis de voz, los componentes de los sistemas TTS tradicionales han sido reemplazados por redes neuronales, simplificando cada vez más los modelos. Primero se usaron *deep feedforward networks* en *Statistical parametric speech synthesis* (SPSS), donde se superaron los resultados de los sistemas basados en modelos ocultos de Markov con un similar número de parámetros (Ze et al., 2013). Más tarde, *recurrent neural networks* (RNNs) basadas en *long short-term memory* (LSTM) se emplearon para capturar información desde cualquier lugar de la secuencia de características (Fan et al., 2014); sin embargo, los experimentos realizados usando el español hablado de México son escasos debido a

que existen problemas por el procesamiento complejo y/o las grandes cantidades de datos requeridos. Además, los resultados más espectaculares han sido desarrollados por empresas privadas, lo que ha dificultado la propagación de estos sistemas hacia otros laboratorios.

Este trabajo se basa en el estado-del-arte de las técnicas de síntesis de voz con deep learning. Su objetivo es obtener la síntesis de voz del español hablado en la región central de México que resulte lo más natural posible. Se explorarán las capacidades de estos modelos y se propondrán mejoras.

La descripción del método empleado para obtener la síntesis de voz es la siguiente:

- Selección de un modelo basado en redes neuronales profundas para obtener la síntesis de voz.
- Diseño del corpus compatible con el modelo.
- Análisis y entrenamiento del modelo, hasta lograr un sintetizador de voz natural del español hablado en la región central de México.
- Conclusiones, se dará una interpretación de los resultados y se propondrán mejoras.

La estructura de los capítulos es la siguiente: En el capítulo 1 se presentan los fundamentos de deep learning, explicando la teoría detrás de esta técnica, para así, comprender el funcionamiento de los modelos e interpretar sus resultados. En el capítulo 2 se muestran los modelos de estado-de-arte en la síntesis de voz, para después, en el capítulo 3, presentar la arquitectura del modelo utilizada. En el capítulo 4 se presentan los experimentos realizados y, para finalizar se muestran en el capítulo 5 las conclusiones.

Capítulo 1

Deep Learning

Los sistemas de inteligencia artificial requieren de la habilidad para adquirir su propio conocimiento mediante la extracción de patrones en los datos. Esta capacidad se conoce como *aprendizaje máquina*. El desempeño de estos modelos depende de la elección de la representación de los datos (Bengio et al., 2013). Una solución a este problema es usar aprendizaje máquina para encontrar la representación. La computadora puede crear representaciones que pueden ser expresadas en términos de representaciones más simples, tal que estas pueden interpretarse en forma de un grafo profundo. Es por esto que este enfoque es llamado *deep learning* (Goodfellow et al., 2016).

1.1. Deep Feedforward Networks

Deep feedforward networks, también llamadas perceptron multicapa (MLPs), son los modelos base en deep learning. Su objetivo es aproximar una función f^* definiendo un mapeo $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ donde se aprende el valor de los parámetros $\boldsymbol{\theta}$ que resulten en la mejor aproximación. Esto se logra minimizando una función de costo $J(\boldsymbol{\theta})$ calculando el gradiente (back-propagation) (Goodfellow et al., 2016).

Por ejemplo, si se tiene una capa oculta y una neurona de salida el modelo tiene la forma

$$\boldsymbol{y} = \boldsymbol{w}^T \boldsymbol{z}(\boldsymbol{x}) \tag{1.1}$$

$$\boldsymbol{z}(\boldsymbol{x}) = g(\boldsymbol{V}\boldsymbol{x}) \tag{1.2}$$

donde g es una función no lineal, $\boldsymbol{z(x)}$ es la capa oculta, $\boldsymbol{V}$ es la matriz de pesos que va de la entrada a los nodos ocultos y $\boldsymbol{w}$ es el vector de pesos de la capa oculta a la salida. Los parámetros del modelo son $\boldsymbol{\theta} = (\boldsymbol{V}, \boldsymbol{w})$ (Murphy, 2012). La figura 1.1, muestra el ejemplo de una red feedforward.

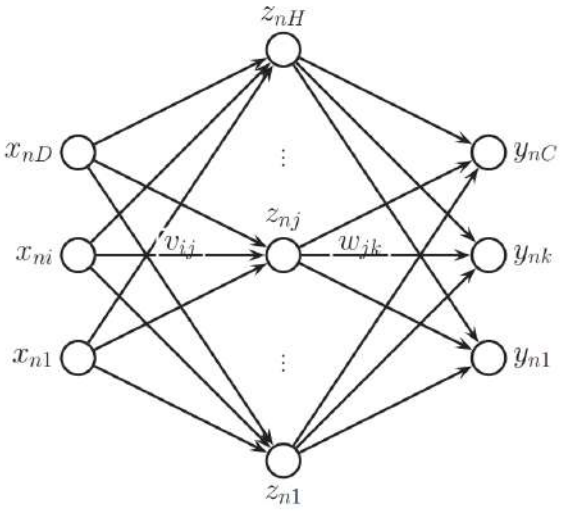


Figura 1.1: Red feedforward con una capa oculta (Murphy, 2012).

Se puede decir que un modelo es considerado profundo cuando el objetivo es producir representaciones abstractas calculando múltiples transformaciones no lineales (Bengio et al., 2013).

1.2. Redes Recurrentes

Recurrent neural networks o *RNNs* son redes neuronales para procesar secuencias de datos. Dada una secuencia de entrada $\boldsymbol{x} = (x_1, \ldots, x_T)$, una red RNN calcula una secuencia de salida $\boldsymbol{y} = (y_1, \ldots, y_T)$.

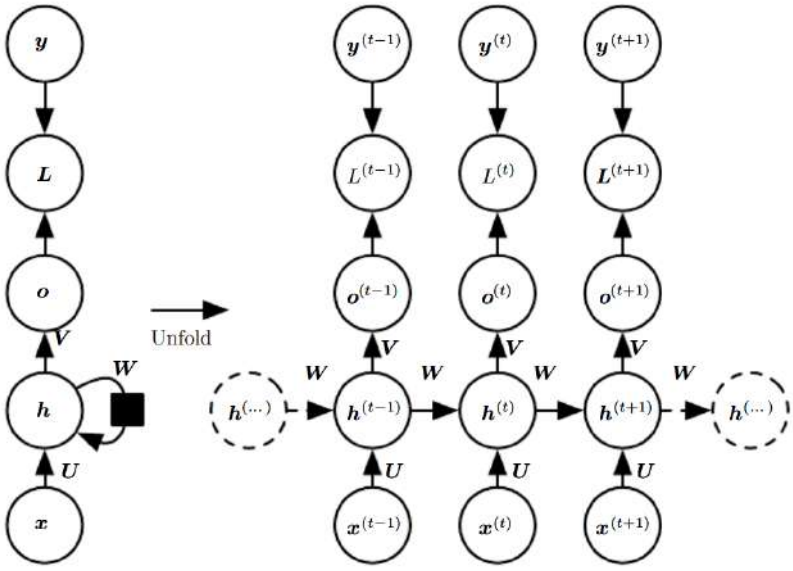


Figura 1.2: Mapeo de la secuencia x a una correspondiente secuancia o. El costo L mide la diferencia de o y el correspondiente objetivo de entrenamiento y (Goodfellow et al., 2016).

Para cada instancia de tiempo t, produce el siguiente estado oculto h_t aplicando recursivamente las siguientes operaciones:

$$h_t = \tanh(\boldsymbol{U}x_t + \boldsymbol{W}h_{t-1}) \tag{1.3}$$

$$o_t = \boldsymbol{V}h_t \tag{1.4}$$

$$\hat{y}_t = \text{softmax } o_t \tag{1.5}$$

La figura 1.2 muestra una instancia de tiempo de una RNN con conexiones recurrentes entre los nodos ocultos.

1.2.1. Gradiente en RNN

Usando el algoritmo general de back-propagation se puede obtener el llamado algoritmo *Back-Propagation Through Time* (BPTT) para entrenar RNNs. Dado que cada instancia de tiempo toma el estado oculto anterior, el entrenamiento es secuencial y por lo tanto poco eficiente. Además es difícil modelar dependencias de largo-rango debido al desvanecimiento/explosión del gradiente (Bengio et al., 1994; Hochreiter, 1998). El desvanecimiento de gradiente provoca que sea difícil conocer la forma en la que los parámetros minimizan la función de costo, por otro lado, la explosión del gradiente ocasiona que el entrenamiento sea inestable.

1.2.2. LSTM

Long short-term memory o *LSTM* (Hochreiter and Schmidhuber, 1997) soluciona el problema de dependencias de largo-rango con un vector de estado s_t y con celdas de entrada (input), olvido (forget) y salida (output) aplicando los siguientes cálculos:

$$i_t = \sigma(\boldsymbol{U}_i x_t + \boldsymbol{W}_i h_{t-1}) \tag{1.6}$$
$$f_t = \sigma(\boldsymbol{U}_f x_t + \boldsymbol{W}_f h_{t-1}) \tag{1.7}$$
$$o_t = \sigma(\boldsymbol{U}_o x_t + \boldsymbol{W}_o h_{t-1}) \tag{1.8}$$
$$s_t = \tanh(\boldsymbol{U}_s x_t + \boldsymbol{W}_s h_{t-1}) \circ i_t + s_{t-1} \circ f_t \tag{1.9}$$
$$h_t = \tanh(s_t) \circ o_t \tag{1.10}$$

donde el producto de Hadamard se denota como $\circ$. Las operaciones que se agregan solucionan el desvanecimiento del gradiente teniendo efecto en el tiempo de entrenamiento.

1.3. Optimización

Entrenar redes neuronales implica la optimización de alguna función objetivo parametrizada $J(\boldsymbol{\theta})$. La mayoría de los estos métodos se basan en un algoritmo de optimización llamado *Stochastic gradient descent* (SGD).

1.3.1. Stochastic gradient descent

SGD es un método iterativo para optimizar alguna función objetivo. Esta función la define Goodfellow et al. (2016) como

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=0}^{m} L(x_i, y_i, \boldsymbol{\theta}), \tag{1.11}$$

donde L es el costo por ejemplo. El gradiente se calcula de la siguiente forma

$$\boldsymbol{g} = \frac{1}{m'}\nabla_{\boldsymbol{\theta}}\sum_{i=0}^{m'} L(x_i, y_i, \boldsymbol{\theta}) \tag{1.12}$$

donde m' es el tamaño del mini-batch. El algoritmo de SGD actualiza los parámetros con el siguiente cálculo:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\boldsymbol{g}, \tag{1.13}$$

donde ϵ es el *learning rate*.

1.3.2. Adaptive Moment Estimation

Adaptive Moment Estimation (Adam) (Kingma and Ba, 2014), es un algoritmo de optimización que es una extensión de SGD. Este puede describirse como la combinación de las ventajas de:

- *Adaptive Gradient Algorithm* (AdaGrad) que usa un learning rate diferente para cada parámetro en cada instancia de tiempo t. Se realizan actualizaciones grandes para parámetros asociados a características poco frecuentes y actualizaciones pequeñas para parámetros asociados a características frecuentes.

- *Root Mean Square Propagation* (RMSProp) que utiliza la magnitud de los gradientes recientes para normalizarlos, también seleccionando un learning rate para cada parámetro.

El algoritmo actualiza las medias móviles del gradiente con los hiperparámetros $\beta_1, \beta_2 \in [0, 1)$ que controlan el decaimiento exponencial de estas medias. En general, Adam supera a los demás optimizadores como se muestra en la figura 1.3.

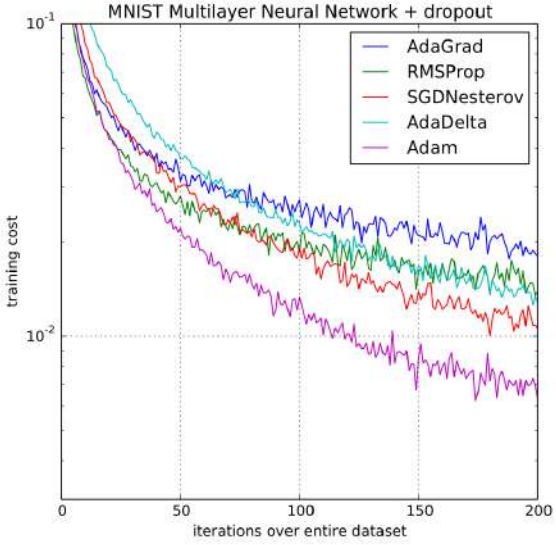


Figura 1.3: Entrenamiento de perceptrón multicapa en MNIST (Kingma and Ba, 2014).

1.4. Regularización

En aprendizaje máquina, las técnicas de *regularización* tienen como objetivo apoyar el aprendizaje del algoritmo con el fin de generalizar y evitar *sobreajuste*.

1.4.1. Batch Normalization

Durante los entrenamientos la distribución de cada entrada de las capas cambia. Esto hace que se requiera de un learning rate bajo y una buena inicialización de pesos. *Batch Normalization* (Ioffe and Szegedy, 2015) soluciona este problema normalizando las entradas de las capas. Se agregan dos parámetros por activación para preservar la capacidad de representación de la red. Esta técnica acelera el entrenamiento.

1.4.2. Dropout

Dropout (Srivastava et al., 2014) consiste en entrenar el ensamble de sub-redes que se forma ignorando algunas neuronas. Esta técnica aumenta el tiempo de entrenamiento requerido, debido a el ruido en la actualización de los parámetros, sin embargo, reduce el sobreajuste.

1.4.3. Zoneout

Zoneout (Krueger et al., 2016) es un método para regularizar RNNs. En cada instancia, zoneout provoca que algunas unidades ocultas mantengan sus valores previos. Como en dropout, se entrena un sub-ensamble, mejorando la generalización.

1.5. Neural Machine Translation

Neural Machine Translation (NMT) es la aplicación de redes neuronales profundas para la traducción de texto. Este enfoque es propuesto por Kalchbrenner y Blunsom (Kalchbrenner and Blunsom, 2013), Sutskever (Sutskever et al., 2014) y Cho (Cho et al., 2014). La mayoría de los modelos propuestos en NMT pertenecen a las arquitecturas *encoder-decoder* (Kalchbrenner and Blunsom, 2013; Cho et al., 2014). Estos modelos se entrenan para maximizar la probabilidad de la secuencia de salida correcta, dada una secuencia de entrada que la red aprende a representar en un vector de contexto de tamaño fijo. El modelo de Luong et al. (2014) lee todas las palabras de entrada hasta que se alcanza el símbolo de fin de oración <eos>, después comienza a generar una palabra objetivo a la vez, como se ilustra en la figura 1.4.

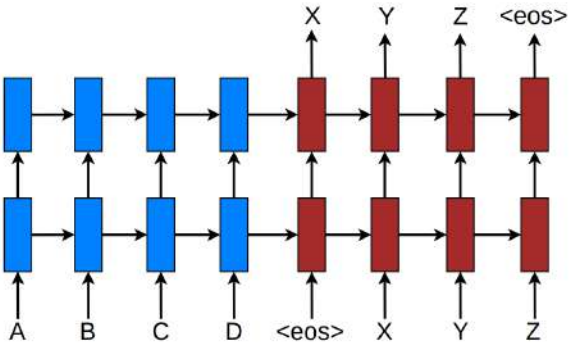


Figura 1.4: Arquitectura recurrente para traducir la secuencia fuente A B C D en la secuencia objetivo X Y Z. Aquí <eos> indica el final de la secuencia (Luong et al., 2015).

1.5.1. Embeddings

Los *Embeddings* son representaciones vectoriales generalmente de palabras (word embeddings). Cada dimensión del embedding representa una característica que contiene propiedades sintácticas y semánticas (Turian et al., 2010). Estas representaciones modelan el lenguaje de manera efectiva usando los vectores como entrada de una red neuronal (Bengio et al., 2003; Mikolov et al., 2010). También se puede modelar el lenguaje con entradas de tipo caracter (Kim et al., 2016).

Para obtener una intuición de la representación que los modelos aprenden, se pueden visualizar los vectores usando *Principal Component Analysis* (PCA) como

se muestra en la figura 1.5.

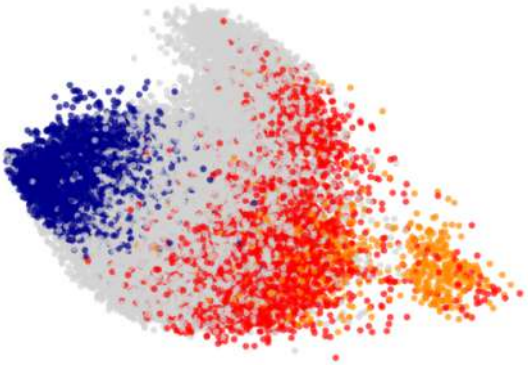


Figura 1.5: PCA de n-gramas de caracteres: prefijos (rojo), sufijos (azul), con guión (naranja) y otros (gris) (Kim et al., 2016).

1.5.2. RNN Encoder-Decoder

Arquitectura encoder-decoder propuesta por Sutskever (Sutskever et al., 2014) y Cho (Cho et al., 2014) donde un encoder lee la secuencia vectores de entrada $\boldsymbol{x} = (x_1, \ldots, x_{T_x})$, a una representación g que Sutskever et al. (2014) denota como el último estado oculto de una red recurrente. El enfoque más común, como lo describe Bahdanau et al. (2014), es usar una RNN tal que,

$$h_j = f(x_j, h_{j-1}) \tag{1.14}$$

y

$$g = q(\{h_1, \ldots, h_{T_x}\}) \tag{1.15}$$

donde h_j es un estado oculto en el tiempo j, y g es un vector de contexto generado de la secuencia de estados ocultos, f y q son funciones no lineales. La probabilidad de la secuencia de salida se define descomponiendo la probabilidad conjunta en condicionales:

$$p(\boldsymbol{y}) = \prod_{i=1}^{T_y} p(y_i | y_i, \ldots, y_{i-1}, g), \tag{1.16}$$

donde $\boldsymbol{y} = (y_1, \ldots, y_{T_y})$. Cada probabilidad condicional se define como:

$$p(y_t|y_1, \ldots, y_{i-1}, g) = f(y_{i-1}, s_i, g), \tag{1.17}$$

donde f es una función no lineal y s_i es el estado oculto de la RNN.

El desempeño de la traducción deteriora a medida que aumenta la longitud de la secuencia de entrada (Kalchbrenner and Blunsom, 2013; Cho et al., 2014). Para abordar este problema, el modelo de atención parametrizado por un MLP propuesta por Bahdanau et al. (2014) aprende la relación entre los elementos de las secuencias de entrada y salida.

1.6. Alineación

Esta arquitectura aprende a seleccionar información de una secuencia fuente oculta durante la decodificación de la secuencia de salida (Bahdanau et al., 2014). En la figura 1.6 se muestra la ilustración gráfica del modelo propuesto por Chorowski et al. (2015). La selección del modelo de atención para una arquitectura encoder-decoder depende de la diferencia entre la dimensión de la secuencia de entrada y salida. El modelo de Bahdanau et al. (2014) es propuesto para NMT.

1.6.1. Decoder

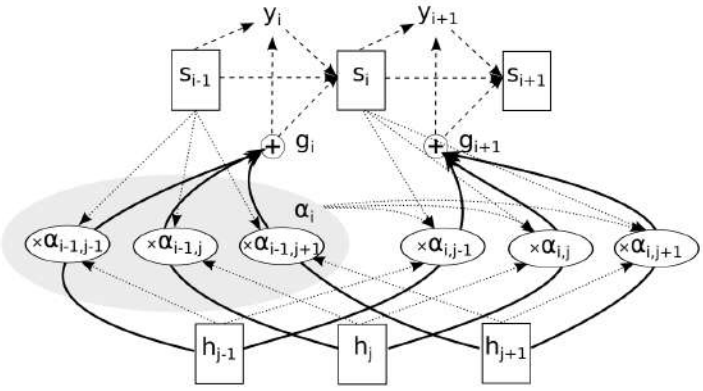


Figura 1.6: Generación del elemento y_i dada una secuencia de características oculta $h_1, \ldots, h_{T_x}$ (Chorowski et al., 2015).

El aprende a generar la secuencia de salida $\boldsymbol{y} = (y_1, \ldots, y_{T_y})$ condicionada en la secuencia oculta $\boldsymbol{h} = (h_1, \ldots, h_{T_x})$:

$$p(y_1, \ldots, y_{T_y} | h_1, \ldots, h_{T_x}) = \prod_{i=1}^{T_y} p(y_i | y_1, \ldots, y_{i-1}, \boldsymbol{h}), \tag{1.18}$$

cada probabilidad condicional se define como:

$$p(y_i | y_1, \ldots, y_{i-1}, \boldsymbol{h}) = f(y_i, s_{i-1}, g_i) \tag{1.19}$$

Aquí la probabilidad está condicionada en un vector de contexto diferente g_i que depende de la secuencia $(h_1, \ldots, h_{T_x})$ para cada instancia y_i. El decoder (Bahdanau et al., 2014) actualiza s_i dado y_{i-1}, s_{i-1} y g_i de la siguiente forma:

$$s_i = f(s_{i-1}, g_i, y_{i-1}) \tag{1.20}$$

El vector de contexto g_i, es el resultado una suma ponderada de las representaciones h_j:

$$g_i = \sum_{j=1}^{T_x} \alpha_{i,j} h_j. \tag{1.21}$$

donde $\alpha_i \in \mathbb{R}^{T_x}$ es un vector de pesos de atención, también llamado alineación. El peso $\alpha_{i,j}$ para cada estado h_j es calculado por

$$\alpha_{i,j} = \frac{\exp(score(s_{i-1}, h_j))}{\sum_{j=1}^{T_x} \exp(score(s_{i-1}, h_j))}. \tag{1.22}$$

donde

$$score(s_{i-1}, h_j) = w^T \tanh(\boldsymbol{U} s_{i-1} + \boldsymbol{W} h_j) \tag{1.23}$$

es un mecanismo de atención *content-based* que evalúa la relación entre las instancias de entrada y salida (Bahdanau et al., 2014). El mecanismo de atención híbrido (Chorowski et al., 2015) toma en cuenta a ubicación de la instancia de tiempo anterior α_{i-1} como Graves et al. (2014) realizando convolución con una matriz $\boldsymbol{T}$:

$$y_i \sim f(s_{i-1}, g_i) \tag{1.24}$$

$$s_i = f(s_{i-1}, g_i, y_i) \tag{1.25}$$

$$p_i = \boldsymbol{T} * \alpha_{i-1} \tag{1.26}$$

$$score(s_{i-1}, h_j, \alpha_{i-1}) = w^T \tanh(\boldsymbol{U} s_{i-1} + \boldsymbol{W} h_j + \boldsymbol{V} p_{i,j}) \tag{1.27}$$

Aquí la evaluación depende de s_{i-1}, h_j y el vector α_{i-1}. La convolución se denota como $*$, $\boldsymbol{U}, \boldsymbol{W}, \boldsymbol{V}, \boldsymbol{T}$ son parámetros y $p_{i,j}$ son características convolucionales. Los pesos $\alpha_{i,j}$ de la ecuación 1.22 como se muestra en la figura 1.7 y 1.8 visualizan la alineación o atención entre las instancias de salida y entrada.

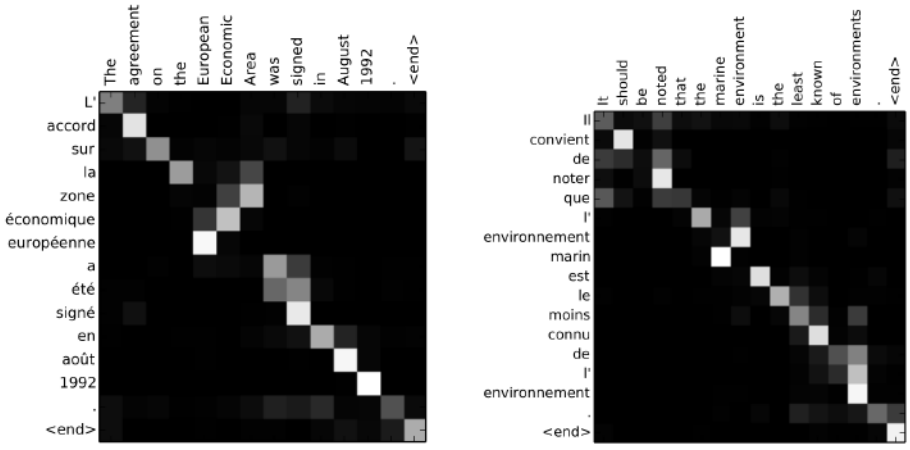


Figura 1.7: Alineación content-based de traducción (Bahdanau et al., 2014).

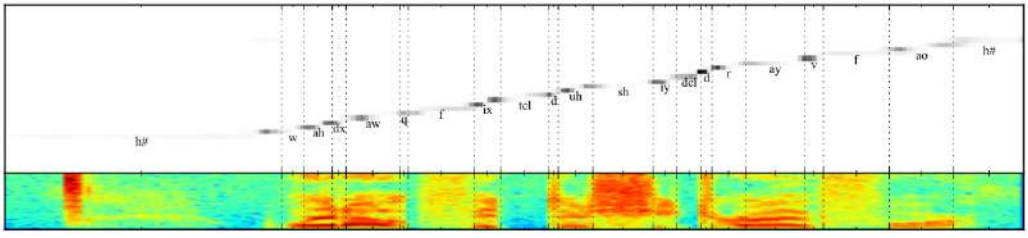


Figura 1.8: Ventanas seleccionadas por el mecanismo de atención (Chorowski et al., 2015).

Capítulo 2

Modelos de síntesis de voz

En los recientes trabajos de TTS se han utilizado redes neuronales para reemplazar los componentes tradicionales de los sistemas, entre estos se encuentran los modelos de predicción de duración de fonemas (Zen and Sak, 2015) y de síntesis de audio (Van Den Oord et al., 2016).

Los modelos *end-to-end* con alineación han sido los más prometedores en los últimos años, ya que se pueden entrenar en pares <texto, audio> con mínima anotación humana, lo cual elimina la necesidad de un conocimiento lingüístico experto, además de facilitar la generación de sintetizadores para nuevos idiomas.

2.1. Wavenet

Wavenet[1] (Van Den Oord et al., 2016) es una red neuronal profunda para generar formas de onda de audio. El modelo es totalmente probabilístico y autorregresivo. En la figura 2.1 se muestra el bloque residual del modelo, este se apila varias veces en la red.

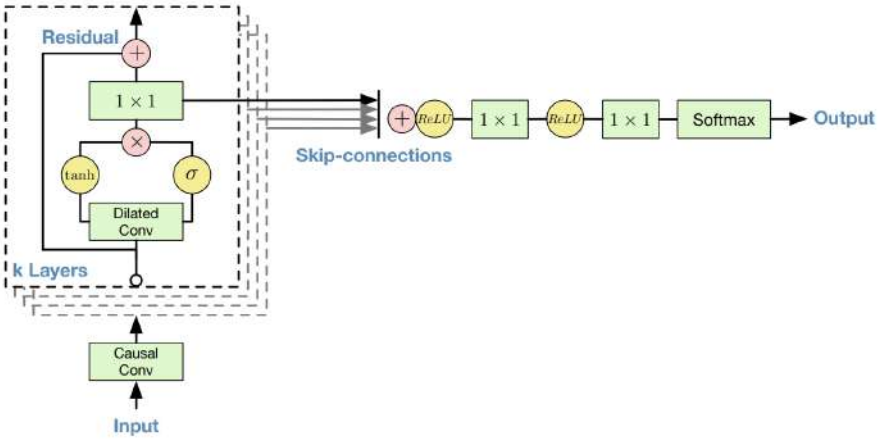


Figura 2.1: Descripción general del bloque residual y la arquitectura completa (Van Den Oord et al., 2016).

En este modelo, cada muestra de audio está condicionada por la muestra de audio anterior. La probabilidad condicional es modelada por una pila de capas convolucionales como se muestra en la figura 2.2. Esta red no tiene capas de pooling.

[1]Audios generados por Wavenet. `https://deepmind.com/blog/article/wavenet-generative-model-raw-audio`

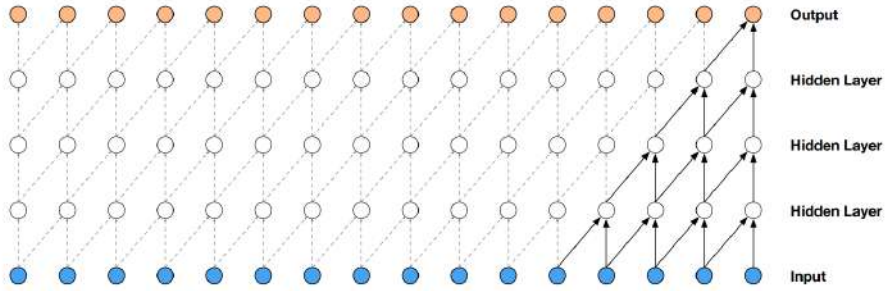


Figura 2.2: Visualización de una pila de capas convolucionales causales (Van Den Oord et al., 2016).

Las convoluciones causales requieren de muchas capas para aumentar el campo receptivo. Para resolver este problema, se utilizan convoluciones dilatadas como se muestra en la figura 2.3. Las convoluciones dilatadas permiten un gran campo receptivo con menor número de capas.

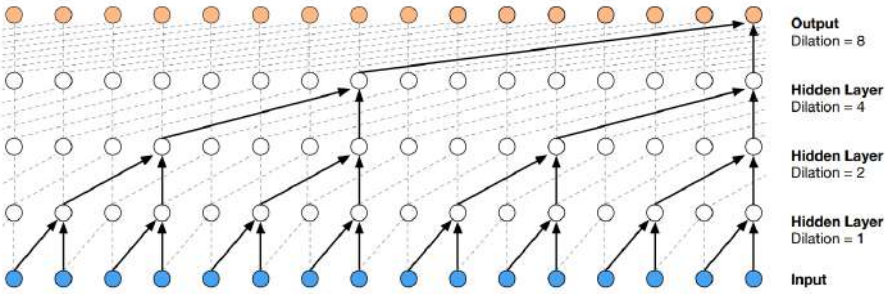


Figura 2.3: Visualización de una pila de capas convolucionales causales dilatadas (Van Den Oord et al., 2016).

Los resultados de Wavenet comparado con los enfoques tradicionales paramétrico y concatenativo se muestran en la figura 2.4. Para la evaluación se toma como medidor un estándar conocido como *Mean Opinion Score* (MOS). Los resultados varían dependiendo del idioma.

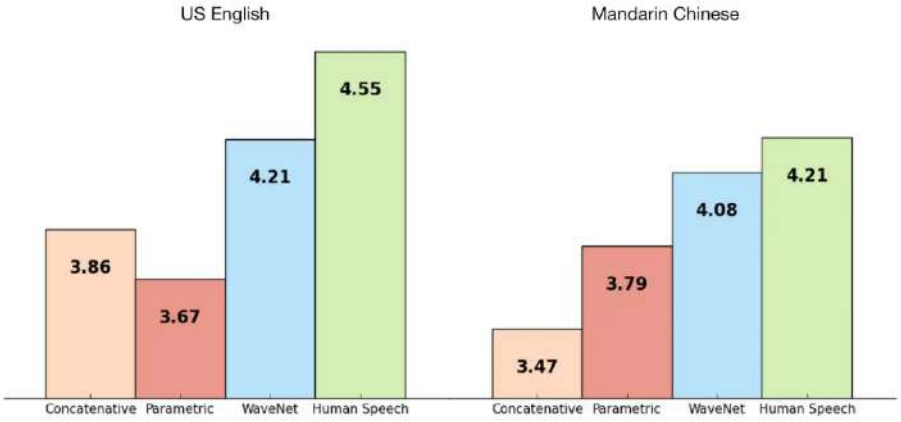


Figura 2.4: Desempeño de Wavenet comparado con el enfoque paramético y concatenativo en Inglés y Chino mandarín. `https://deepmind.com/blog/article/wavenet-generative-model-raw-audio`

2.2. Deep Voice 3

Deep Voice 3 (Ping et al., 2017) es un sistema TTS totalmente convolucional basado-en-atención. La arquitectura propuesta puede convertir características textuales como caracteres y fonemas en diferentes parámetros de vocoder, además evita el uso de RNN para acelerar el entrenamiento. Sin embargo, su fidelidad de audio no ha demostrado rivalizar con el habla humana como se muestra en la figura 2.6. La figura 2.5 muestra la arquitectura del modelo.

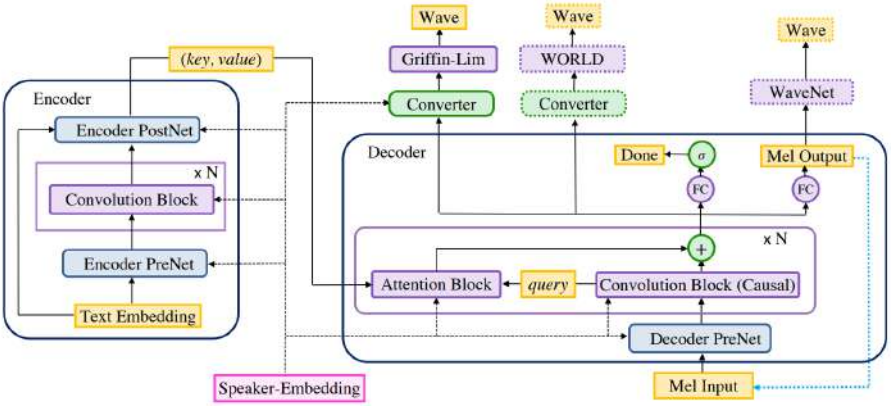


Figura 2.5: Arquitectura de Deep Voice 3 (Ping et al., 2017).

Model	Mean Opinion Score (MOS)
Deep Voice 3 (Griffin-Lim)	3.62 ± 0.31
Deep Voice 3 (WORLD)	3.63 ± 0.27
Deep Voice 3 (WaveNet)	3.78 ± 0.30
Tacotron (WaveNet)	3.78 ± 0.34
Deep Voice 2 (WaveNet)	2.74 ± 0.35

Figura 2.6: MOS de Deep Voice 3 comparado con otros modelos (Ping et al., 2017).

2.3. Tacotron

Tacotron[2] (Wang et al., 2017) es una arquitectura end-to-end para producir espectrogramas a partir de una secuencia de caracteres. Este modelo simplifica las etapas de procesamiento tradicional al reemplazar la producción de características lingüísticas y acústicas con una sola red neuronal. En las figuras 2.7 y 2.8 se muestran los componentes del modelo. Para estimar la señal de audio del espectrograma, Tacotron utiliza el algoritmo de Griffin-Lim (Griffin and Lim, 1984). La señal resultante es inteligible, aunque se pierden algunas características de la voz.

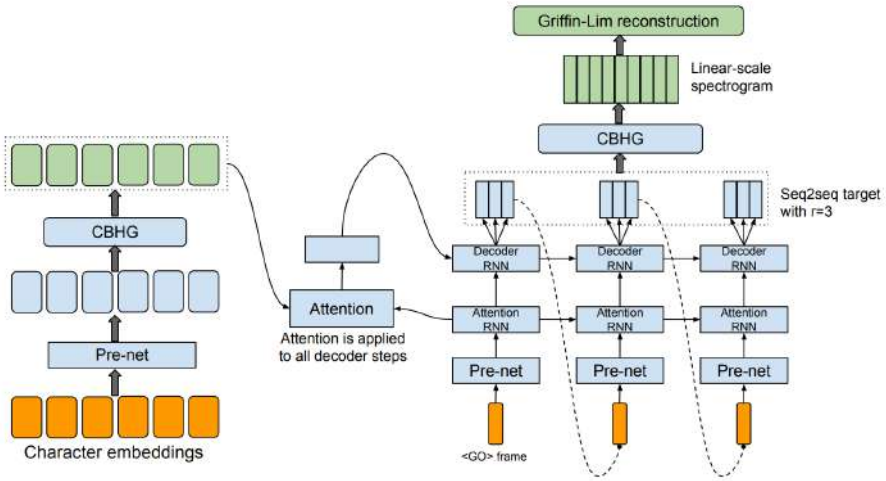


Figura 2.7: Arquitectura de Tacotron (Wang et al., 2017).

[2]Audios generados por Tacotron. `https://google.github.io/tacotron/publications/tacotron/index.html`

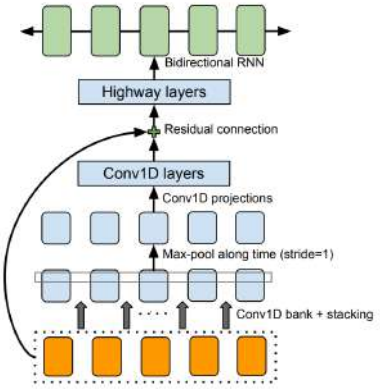


Figura 2.8: Módulo CBHG (1-D convolution bank + highway network + bidirectional GRU) (Wang et al., 2017).

Combinando lo mejor de Tacotron y Wavenet se propone Tacotron 2 (Shen et al., 2018). En la figura 2.9 se muestra cómo esta red supera a los enfoques tradicionales y a los modelos end-to-end con respecto a naturalidad en inglés, sin embargo su funcionamiento en otros idiomas sigue siendo desconocido. En el siguiente capítulo se describe a detalle la arquitectura de este modelo.

Name	MOS
Parametric	3.492 ± 0.096
Tacotron (Griffin-Lim)	4.001 ± 0.087
Concatenative	4.166 ± 0.091
WaveNet (Linguistic)	4.341 ± 0.051
Ground Truth	4.582 ± 0.053
Tacotron 2 (this paper)	$\mathbf{4.526 \pm 0.066}$

Figura 2.9: MOS de Tacotron comparado con otros modelos (Shen et al., 2018).

Capítulo 3

Arquitectura del Modelo

Tacotron 2[1] (Shen et al., 2018) es un modelo end-to-end TTS que sintetiza voz directamente de caracteres. El sistema consiste de dos componentes, los cuales son mostrados en la figura 3.1: (1) una red secuencia-a-secuencia recurrente de predicción de características con atención, que predice una secuencia de ventanas de espectrograma dada una secuencia de caracteres de entrada, y (2) una versión de WaveNet que genera formas de onda condicionadas en la predicción de ventanas de espectrograma.

El código que se usa es una implementación open-soruce en *Tensorflow*[2] (Mama, 2019) modificada para trabajar con el español de México.

[1] Audios generados por Tacotron 2. `https://google.github.io/tacotron/publications/tacotron2/index.html`

[2] `https://github.com/Rayhane-mamah/Tacotron-2`

3.1. Red para Predicción de Espectograma

La red esta compuesta de un encoder y un decoder con-atención. El encoder 3.1 convierte una secuencia de caracteres en una representación de características ocultas que el decoder utiliza para predecir un espectrograma. Los caracteres de entrada son representados usando un embedding aprendido de tipo caracter y 512, que pasa a través de una pila de 3 capas convolucionales cada una con 512 filtros de forma 5×1, seguidos de batch normalization y activaciones ReLU. Como en Tacotron, estas capas convolucionales modelan un contexto de corto-largo rango en la secuencia de caracteres de entrada. La salida de la capa final convolucional pasa a través de una capa LSTM bi-direccional de 512 unidades (256 en cada dirección) para generar las secuencia de características oculta.

```
# Embeddings ==> [batch_size, sequence_length, embedding_dim]
embedding_table = tf.get_variable('inputs_embedding',
                                  [len(symbols), hp.embedding_dim],
                                  dtype=tf.float32)


embedded_inputs = tf.nn.embedding_lookup(embedding_table, inputs
    )


#Encoder Cell ==> [batch_size, encoder_steps, encoder_lstm_units
    ]
encoder_cell = TacotronEncoderCell(
                    EncoderConvolutions(is_training,
                             kernel_size=hp.enc_conv_kernel_size,
                             channels=hp.enc_conv_channels,
                             scope='encoder_convolutions'),

                    EncoderRNN(is_training,
                             size=hp.encoder_lstm_units,
                             zoneout=hp.tacotron_zoneout_rate,
                             scope='encoder_LSTM'))


encoder_outputs = encoder_cell(embedded_inputs, input_lengths)

#For shape visualization purpose
enc_conv_output_shape = encoder_cell.conv_output_shape
```

Código 3.1: Tacotron 2 encoder (Mama, 2019).

La salida del encoder se utiliza por una red de atención que resume toda la información en un vector de contexto de tamaño fijo para cada instancia de salida

del decoder. Se usa atención sensible-a-ubicación (Chorowski et al., 2015) para añadir los pesos de atención de las instancias anteriores del decoder como una característica adicional.

```
#Decoder Parts
#Attention Decoder Prenet
prenet = Prenet(is_training, layer_sizes=hp.prenet_layers,
                        scope='decoder_prenet')


#Attention Mechanism
attention_mechanism = LocationSensitiveAttention(
                        hp.attention_dim, encoder_outputs,
                mask_encoder=hp.mask_encoder,
                        memory_sequence_length=input_lengths,
                        smoothing=hp.smoothing,
                        cumulate_weights=hp.cumulative_weights)


#Decoder LSTM Cells
decoder_lstm = DecoderRNN(is_training,
                        layers=hp.decoder_layers,
                        size=hp.decoder_lstm_units,
                        zoneout=hp.tacotron_zoneout_rate,
                        scope='decoder_lstm')


#Frames Projection layer
frame_projection = FrameProjection(
                        hp.num_mels * hp.outputs_per_step,
                        scope='linear_transform')


#<stop_token> projection layer
stop_projection = StopProjection(is_training,
                        scope='stop_token_projection')


#Decoder Cell ==> [batch_size, decoder_steps,
#                  num_mels * r] (after decoding)
decoder_cell = TacotronDecoderCell(
                        prenet, attention_mechanism,
                      decoder_lstm, frame_projection,
                stop_projection,
                        mask_finished=hp.mask_finished)
```

Código 3.2: Tacotron 2 decoder (Mama, 2019).

El decoder 3.2 es una red neuronal recurrente autorregresiva que predice un espectrograma a partir de la secuencia de entrada codificada, una ventana a la vez. La predicción de la instancia de tiempo anterior primero pasa a través de una pequeña *pre-net* que contiene 2 capas completamente conectadas de 256 unidades ocultas ReLU. La pre-net es indispensable para aprender la atención. La salida de la pre-net y el vector de contexto de atención se concatenan y pasan a través de una pila de 2 capas LSTM unidireccionales con 1024 unidades. La concatenación de la salida LSTM y el vector de contexto de atención se proyecta a través de una transformación lineal para predecir el espectrograma objetivo. Finalmente, las características predecidas pasan a través de una *post-net* 3.3 de 5 capas convolucionales donde se predice un residuo que se añade a la predicción inicial para mejorar la reconstrucción total. Cada capa de la post-net está compuesta por 512 filtros de forma 5×1 con batch normalization y activaciones tanh a excepción de la capa final.

```
#Postnet
postnet = Postnet(is_training,
                  kernel_size=hp.postnet_kernel_size,
              channels=hp.postnet_channels,
                  scope='postnet_convolutions')


#Compute residual using post-net ==> [batch_size,
                  #decoder_steps * r, postnet_channels]
residual = postnet(decoder_output)


#Project residual to same dimension as mel spectrogram
#==> [batch_size, decoder_steps * r, num_mels]
residual_projection = FrameProjection(hp.num_mels,
                          scope='postnet_projection')
projected_residual = residual_projection(residual)


#Compute the mel spectrogram
mel_outputs = decoder_output + projected_residual
```

Código 3.3: Tacotron 2 post-net (Mama, 2019).

Se minimiza el error cuadrático medio (MSE) antes y después de la post-net.

Las capas convolucionales de la red son regularizadas usando dropout con probabilidad de 0.5 y las LSTM son regularizadas con zoneout con probabilidad de 0.1.

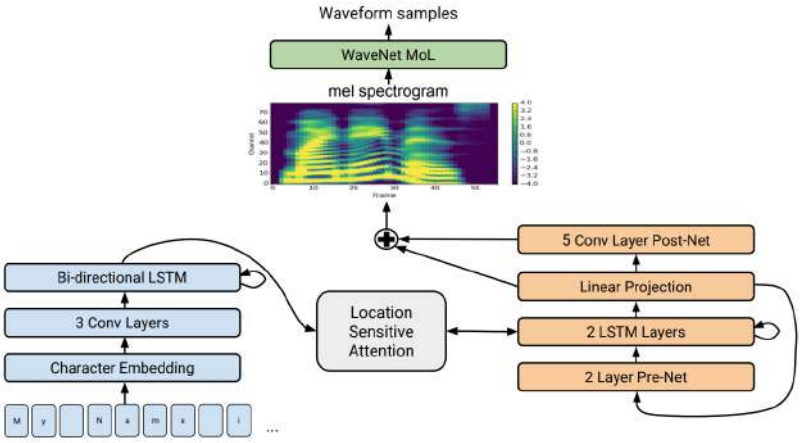


Figura 3.1: Diagrama de bloques de la arquitectura del sistema Tacotron 2 (Shen et al., 2018).

Capítulo 4

Experimentos y resultados

En esta sección se presentan los resultados de los experimentos. Se empieza con el entrenamineto de la red de predicción de características para después continuar con los experimentos de la generación del espectrograma. Para la generación de formas de onda se utiliza una versión de Wavenet pre-entrenada en LJSpeech-1.1 Ito (2017), que es un dataset para entrenar modelos en inglés.

4.1. Dataset

Se desarrolló el dataset KS-1.0. Este se encuentra compuesto por fragmentos de audiolibros los cuales se segmentaron cada cierta duración de silencio entre las oraciones utilizando un módulo que se desarrolló en Python para pre-procesamiento de datos[1]. Cada fragmeto cuenta con su transcripción y su respectivo espectrograma. En total 8504 (9.36 horas) segmentos de audio. La figura 4.1 muestra la distribución resultante de la duración de los audios.

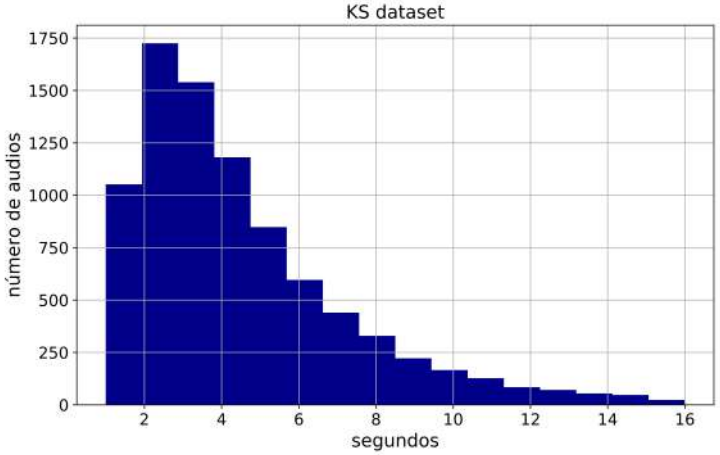


Figura 4.1: Distribución de la duración de los audios en KS-1.0 dataset.

[1] `https://github.com/Ajolotl/Kalatl.git`

4.2. Hardware

Las *Graphics Processing Units* (GPUs), son procesadores especializados originalmente creados para tareas de gráficos de computadora. Las GPUs modernas cuentan con muchos procesadores (cores) simples que pueden trabajar en paralelo, lo que es sumamente efectivo al momento de correr las operaciones matriciales de deep learning. La figura 4.2 muestra la comparación de velocidad entre GPU y CPU. Todos los experimentos se realizaron en una GPU GeForce GTX 1050.

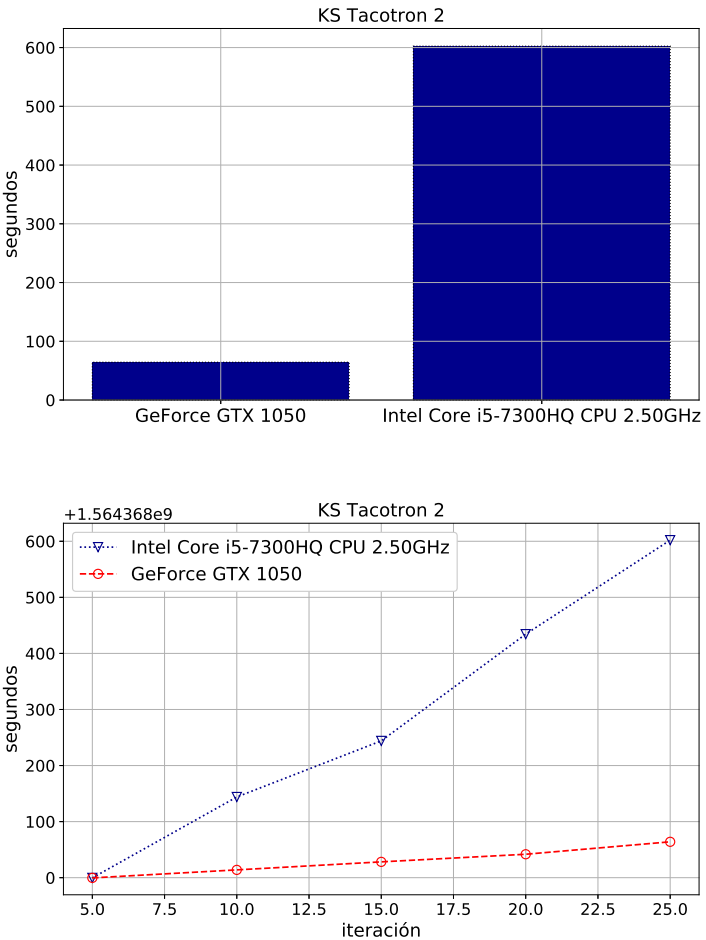


Figura 4.2: 25 iteraciones de Tacotron 2 en GPU y CPU

4.3. Entrenamiento con parámetros originales

En el proceso de entrenamiento, primero se entrena el modelo M1 que es la red de predicción de características Tacotron 2 con los parámetros originales de los autores.

Se aplica batch size = 5 en una sola GPU. Adam oprimizer con $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-6}$ y learning rate de 10^{-3} exponencialmente decayendo a 10^{-5} empezando después de $50,000$ iteraciones.

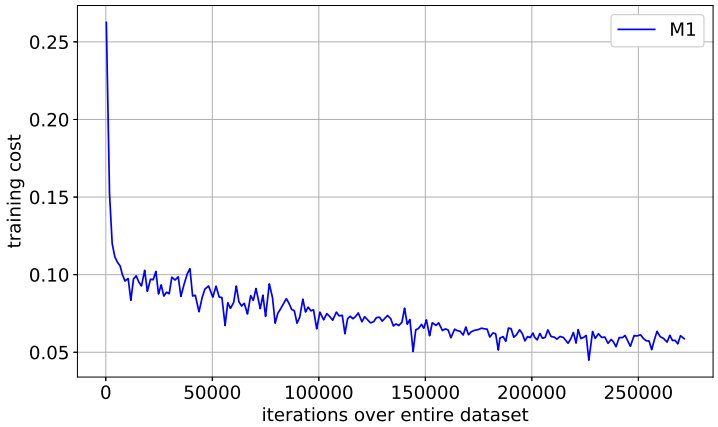


Figura 4.3: Función de costo del modelo M1.

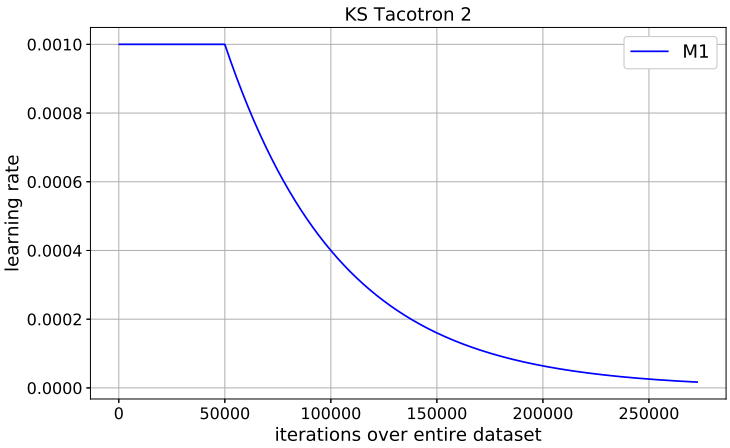


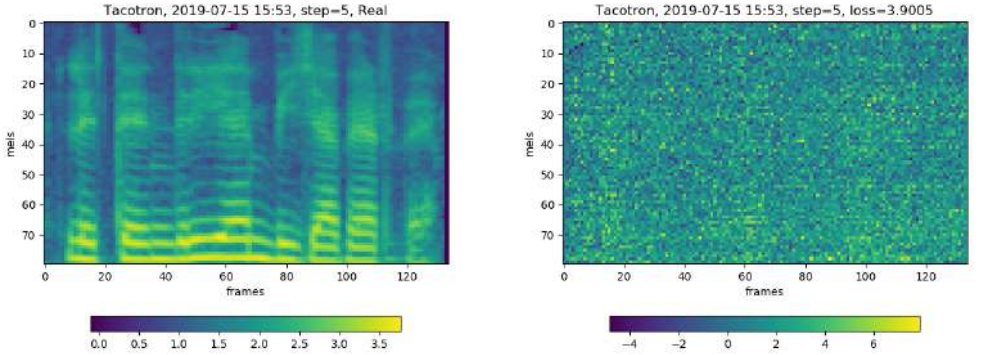
Figura 4.4: Learning rate del modelo M1.

La función de costo en la figura 4.3 se minimiza en menor tiempo cuando el learning rate es alto. A medida que el learning rate decrece como se muestra en la figura 4.4, la actualización de los pesos es menor, por lo que es mas fácil alcanzar el valor mínimo.

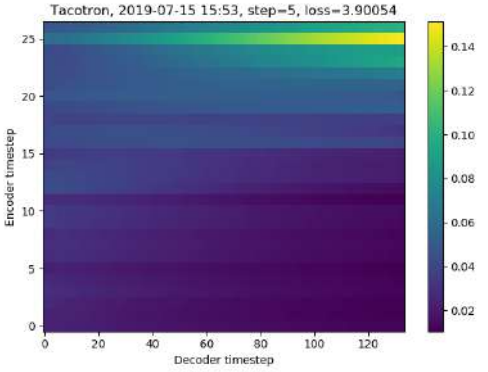
El costo no varía mucho de la iteración 170k a la iteración 270k, sin embargo los espectrogramas generados difieren bastante de los reales como se muestra a continuación.

4.3.1. Generación del espectrograma

En la figura 4.5 se muestra como la red genera ruido en las primeras iteraciones.



(a) (b)



(c)

Figura 4.5: (a) Espectrograma real, (b) espectrograma generado por M1 y (c) alineación en 5 iteraciones.

Después de entrenar 3 días y 14 horas, la red produce espectrogramas muy parecidos a los del conjunto de entrenamiento como se muestra en la figura 4.6, sin embargo la alineación es difusa, lo que provoca que no se puedan generar espectrogramas no observados durante el entrenamiento. Esto puede indicar sobreajuste en algunos parámetros.

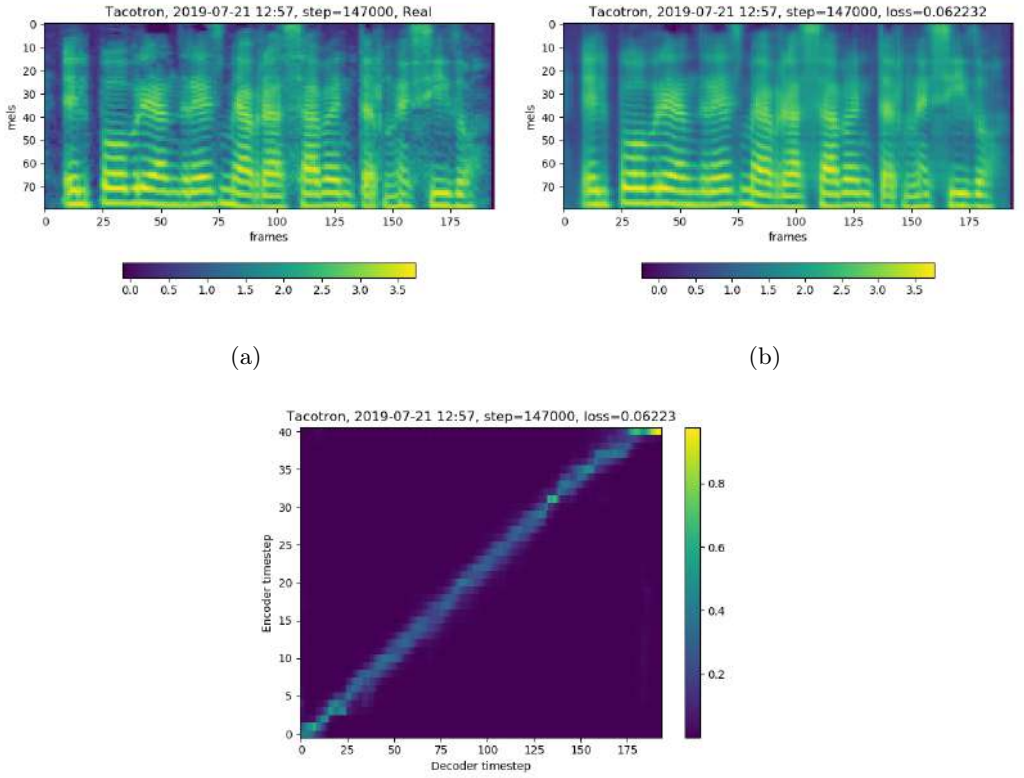


Figura 4.6: (a) Espectrograma real, (b) espectrograma generado por M1 y (c) alineación en 147k iteraciones.

Después de 6 días y 17 horas, el modelo M1 puede alinear de forma definida y crear espectrogramas casi idénticos a los reales como se muestra en la figura 4.7. A pesar de esto, aún se encuentran errores en la pronunciación de algunas oraciones.

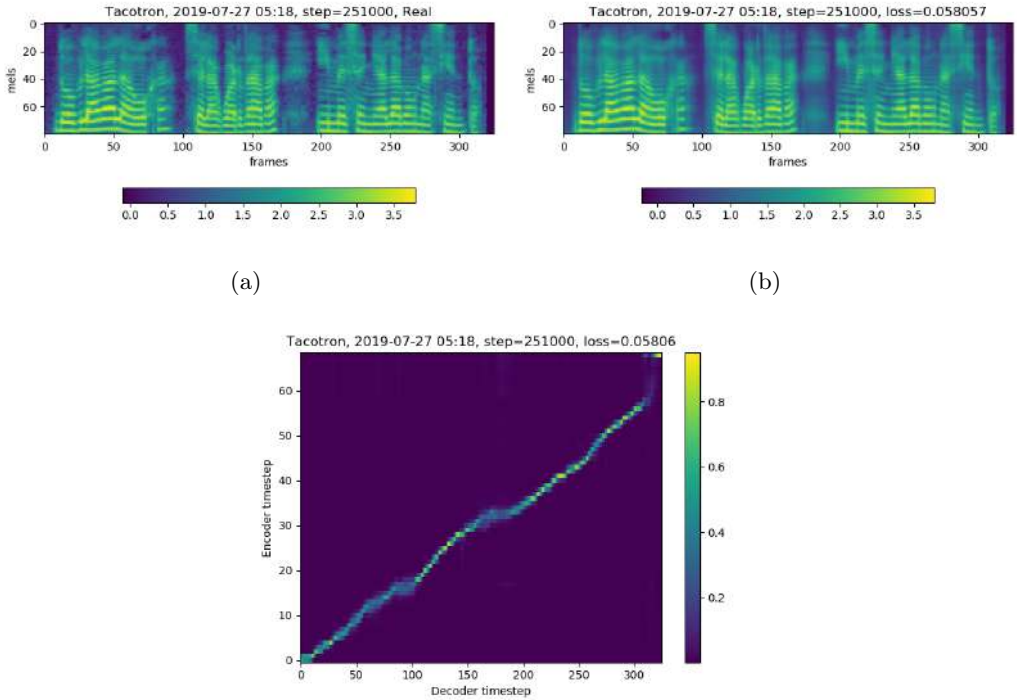
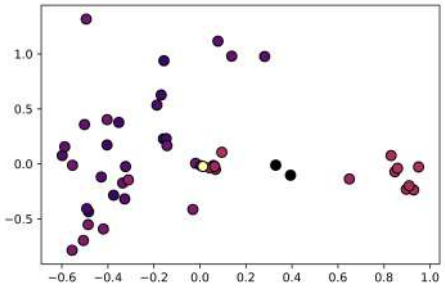


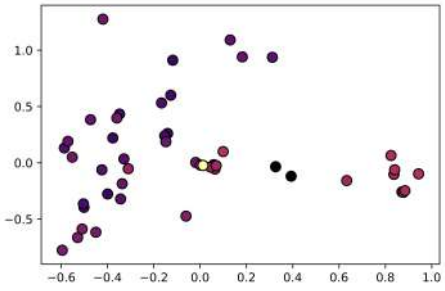
Figura 4.7: (a) Espectrograma real, (b) espectrograma generado por M1 y (c) alineación en 251k iteraciones.

4.3.2. Embeddings

Al ser el learning rate muy bajo, la actualización de los pesos de 240k a 270k es mínima, lo que genera cambios en la prosodia. Las representaciones de la figura 4.8 en $\mathbb{R}^2$ de los embeddings en $\mathbb{R}^{512}$ ayudan a visualizar de forma intuitiva estos pequeños cambios en el aprendizaje de la red.



(a)



(b)

Figura 4.8: PCA de embeddings en iteraciones (a) 240k y (b) 270k.

4.4. Entrenamiento con capas pre-entrenadas

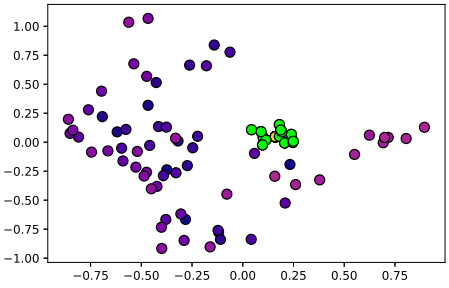
Transfer Learning (Pan and Yang, 2009) es cuando el aprendizaje obtenido en una cierta tarea es explotado para mejorar la generalización en otra tarea. Esta estrategia ha sido aplicada con éxito en NMT (Zoph et al., 2016).

Para este entrenamiento, se desarrolló un módulo de Python para transfer-learning[2] en modelos de Tensorflow. Se utilizó un modelo de Tacotron 2 entrenado en LJSpeech-1.1, al cual se le agregaron los caracteres faltantes del idioma como las vocales con acentuación. Con este aprendizaje se inicializaron los pesos de los modelos M2, M3 y M4 a los cuales de les modificó los hiperparámetros *start decay* y *decay steps* con los valores que se muestran en la tabla 4.3 con el objetivo de no perder el aprendizaje previo adquirido. El esquema de entrenamiento está inspirado en Smith et al. (2017).

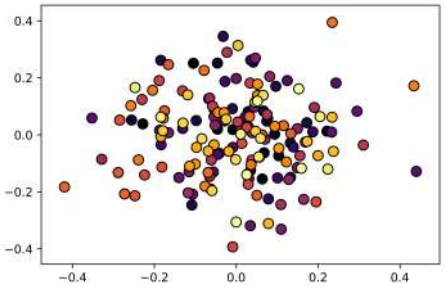
Modelo	**Start decay**	**Decay steps**	**Batch-size**	**Pre-entrenado**
M1	50k	50k	5	No
M2	”	”	6	Si
M3	1.5k	50k	”	”
M4	1	20k	”	”

Cuadro 4.1: Hiperparámetros de modelos.

[2] `https://github.com/Ajolotl/Coatl.git`



(a)



(b)

Figura 4.9: PCA de (a) Embeddings de modelo entrenado en LJSpeech-1.1 con representaciones faltantes (verde) y (b) embeddings sin entrenar.

En la figura 4.9 se observa una estructura diferente aprendida por Tacotron 2 para LJSpeech-1.1.

Con audios de duración máxima de 10 segundos como se muestra en la figura 4.10 el batch size se inceremenata a 6.

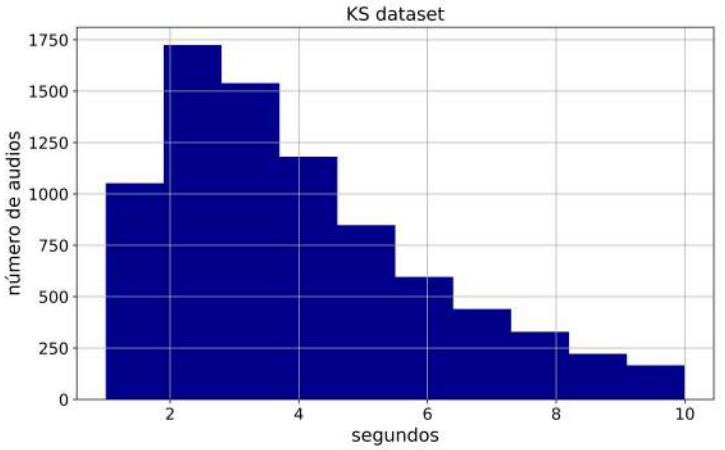


Figura 4.10: Distribución de la duración de los audios en KS-1.0 dataset hasta 10 segundos

En la figura 4.11 se muestra como los modelos M2, M3 y M4 que cuentan con capas pre-entrenadas, minimizan la función de costo en menor tiempo. En la figura 4.12 se puede observar que al decaer el learning rate desde las primeras iteraciones en los modelos M3 y M4, disminuye aún más el tiempo de entrenamiento requerido.

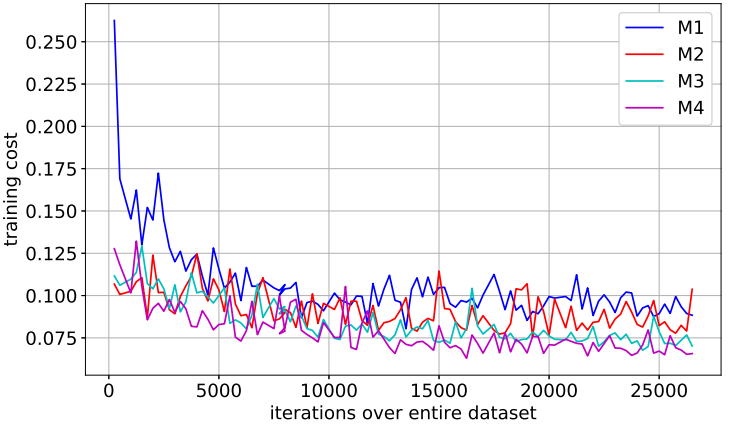


Figura 4.11: Función de costo de los modelos.

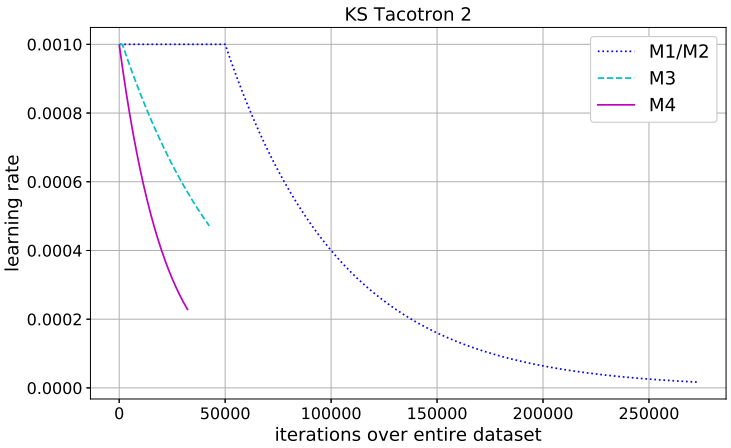
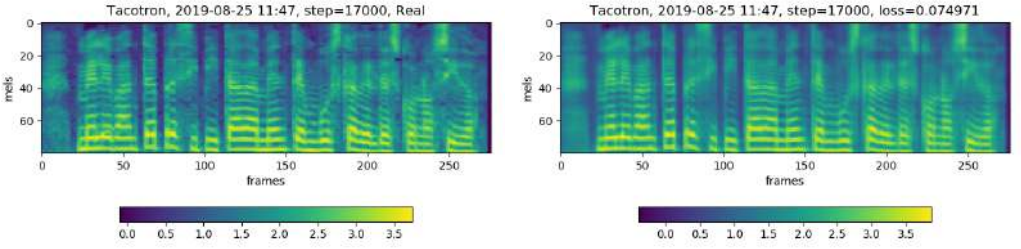


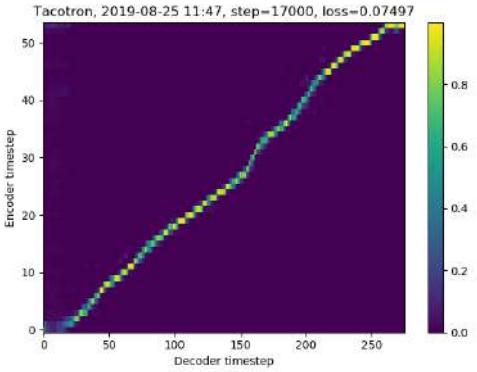
Figura 4.12: Learning rate de los modelos.

4.4.1. Generación del espectrograma

En la figura 4.13 se muestra el espectrograma generado por la red en tan solo 17K iteraciones, lo que equivale a tan solo 10 horas de entrenamiento. La alineación conseguida se encuentra definida, lo que permite obtener espectrogramas no observados durante el entrenamiento con una correcta pronunciación.



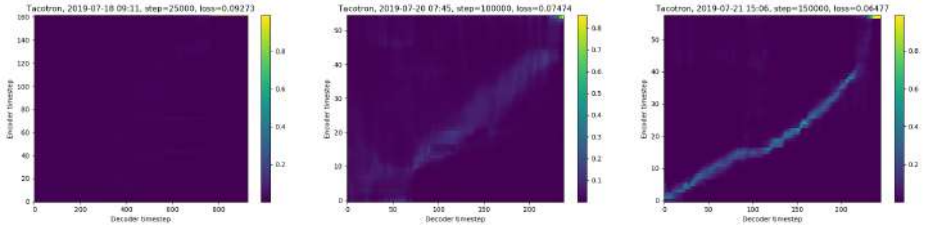
(a) (b)



(c)

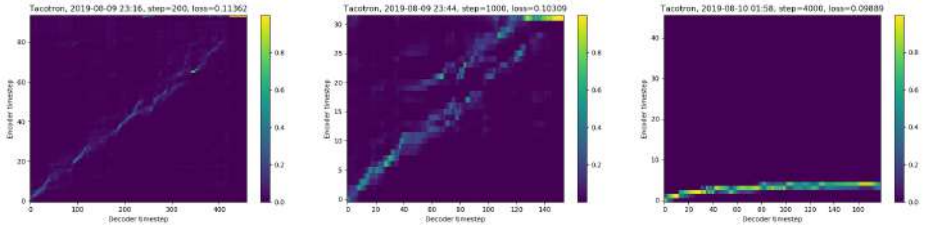
Figura 4.13: (a) Espectrograma real, (b) espectrograma generado por M4 y (c) alineación en 17k iteraciones.

En las figuras 4.14, 4.15, 4.16 y 4.17 se muestran los efectos del correcto ajuste en los parámetros start decay y decay steps en la alineación. A pesar de que parece que el modelo M3 aprende a alinear en un menor tiempo, se puede observar como en la iteración 4k el modelo M4 supera su calidad.



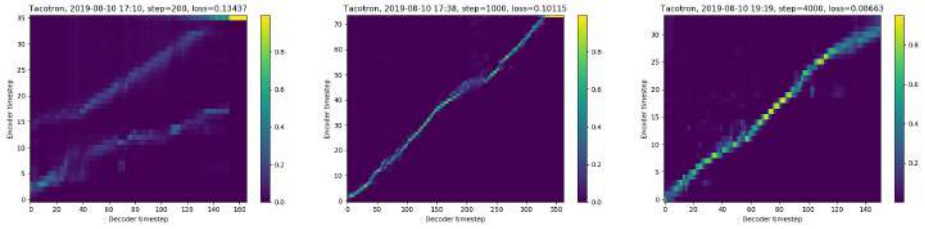
(a) (b) (c)

Figura 4.14: Alineación del modelo M1 en (a) 25k, (b) 100k y (c) 150k iteraciones.



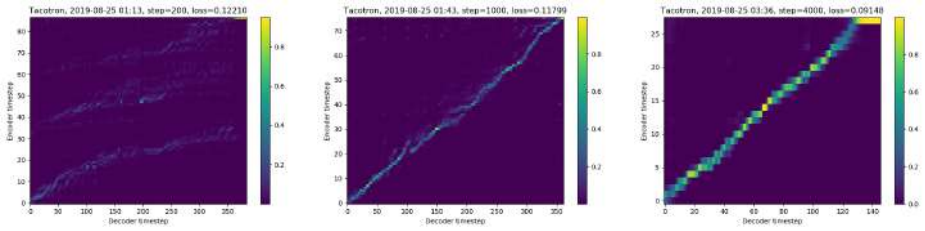
(a) (b) (c)

Figura 4.15: Alineación del modelo M2 en (a) 200 (b) 1k y (c) 4k iteraciones.



(a) (b) (c)

Figura 4.16: Alineación del modelo M3 en (a) 200 (b) 1k y (c) 4k iteraciones.

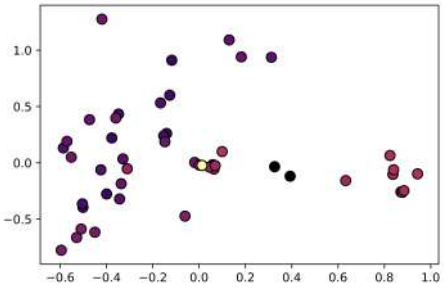


(a) (b) (c)

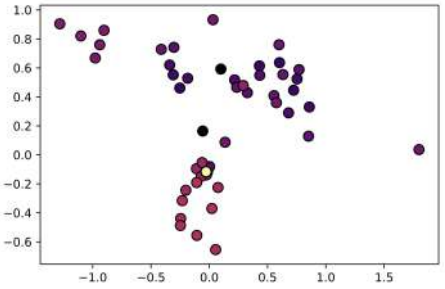
Figura 4.17: Alineación del modelo M4 en (a) 200 (b) 1k y (c) 4k iteraciones.

4.4.2. Embeddings

Los embeddings aprendidos por el modelo M4 en la figura 4.18 muestran una estructura similar a la del modelo M1 pero en una orientación diferente. Esto indica que los pesos de la red convergen siempre a una respectiva estructura.



(a)



(b)

Figura 4.18: PCA de embeddings de modelos (a) M1 en iteración 270k y (b) M4 en iteración 32k.

4.5. Entrenamiento con menor número de capas

Después de mostrar como la alineación del modelo es más relevante que otros parámetros para la correcta generación de espectrogramas, se optó por inicializar los pesos de modelos con un menor número de parámetros para optimizar la red. Se utilizó el modelo M4 para inicializar N1, N2, N3, N4 y N5, los cuales son versiones de Tacotron 2 con menor número capas convolucionales, LSTM y de post-net. Los parámetros start decay y decay steps permanecen igual a M4.

En la tabla 4.2 se observa como al reducir las capas LSTM, el número de parámetros decrece mucho más que al reducir las capas convolucionales o de post-net. La velocidad depende del tamaño de los elementos del batch, por lo que se tomó el tiempo promedio del entrenamiento con un batch de tamaño similar por modelo.

Modelo	**Encoder**	**Decoder**		**parámetros (%)**	**velocidad**
	conv	**lstm**	**post-net**		**(sec/step)**
M4	3	2	5	100	2.174 ± 0.030
N1	1	1	1	40.26	1.058 ± 0.006
N2	”	”	3	49.93	1.154 ± 0.006
N3	”	2	1	71.11	1.784 ± 0.013
N4	”	”	2	75.88	1.886 ± 0.023
N5	”	”	3	80.66	1.992 ± 0.017

Cuadro 4.2: Modelos de Tacotron 2 con diferente número de capas, número de parámetros y velocidad promedio de entrenamiento

En la figura 4.19 se observa como es más difícil minimizar el error sin la segunda capa LSTM del decoder, sin embargo el número de capas en post-net y convolucionales parecen no ser igual de relevantes.

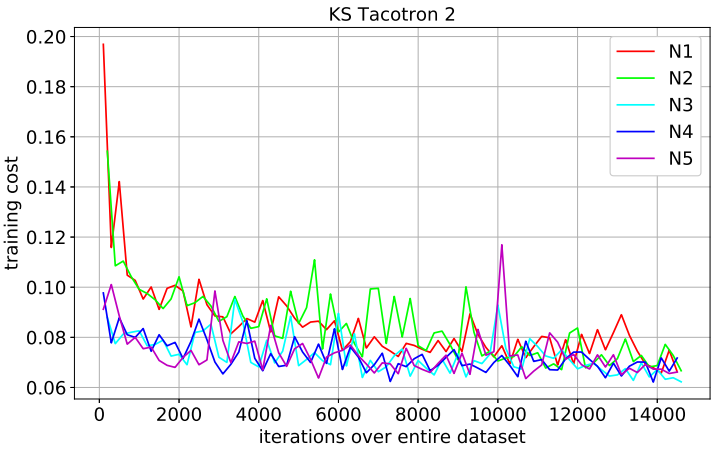
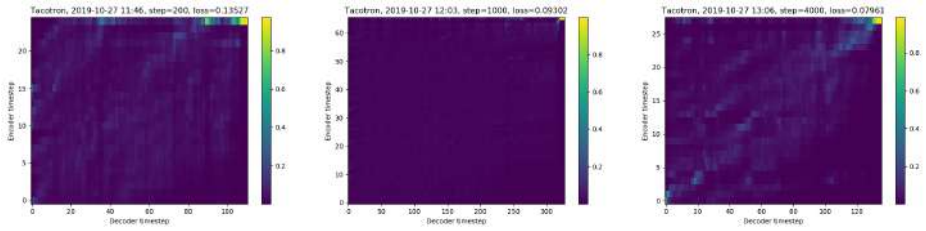


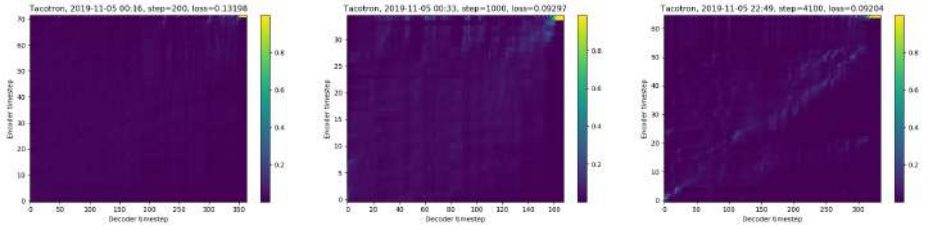
Figura 4.19: Función de costo de modelos.

Se puede notar en las figuras 4.20, 4.21, 4.22, 4.23 y 4.24 que al reducir las capas LSTM la alineación se encuentra muy afectada. La reducción de capas convolucionales y de post-net parece no afectar el resultado de los espectrogramas, lo que permite trabajar solo con el %80.7 de los parámetros originales de Tacotron 2 en el modelo N5, consiguiendo resultados muy similares a los del modelo M4.



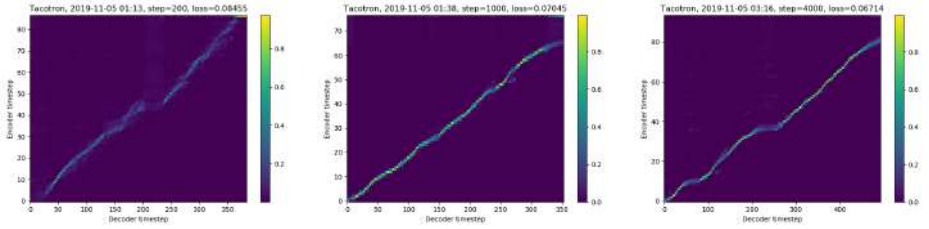
(a) (b) (c)

Figura 4.20: Alineación del modelo N1 en (a) 200 (b) 1k y (c) 4k iteraciones.



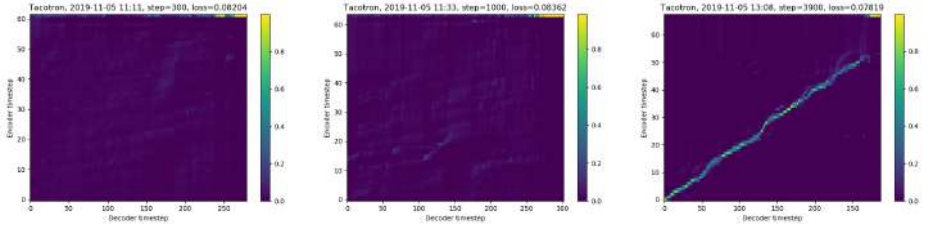
(a) (b) (c)

Figura 4.21: Alineación del modelo N2 en (a) 200 (b) 1k y (c) 4k iteraciones.



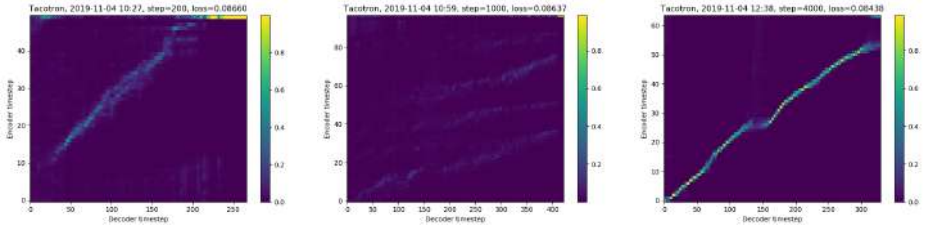
(a) (b) (c)

Figura 4.22: Alineación del modelo N3 en (a) 200 (b) 1k y (c) 4k iteraciones.



(a) (b) (c)

Figura 4.23: Alineación del modelo N4 en (a) 300 (b) 1k y (c) 3.9k iteraciones.



(a) (b) (c)

Figura 4.24: Alineación del modelo N5 en (a) 200 (b) 1k y (c) 4k iteraciones.

4.6. Evaluación de los modelos

Similar a la evaluación realizada por los autores de Tacotron 2, se seleccionaron 5 ejemplos generados por cada modelo no observados durante el entrenamiento[3] . Cada ejemplo fué calificado por 10 personas en escala de 1 a 5, siendo 1 la calidad más baja y 5 la más alta. Los resultados se muestran en las tablas 4.3.

Modelo	**Inteligibilidad**	**Naturalidad**
M1 (Griffin-Lim)	3.42 ± 0.666	3.24 ± 0.708
M2 (Griffin-Lim)	3.64 ± 0.624	3.88 ± 0.430
M4 (Griffin-Lim)	4.02 ± 0.373	3.92 ± 0.391
M4 (WaveNet)	4.48 ± 0.499	4.46 ± 0.573
N5 (WaveNet)	4.42 ± 0.493	4.38 ± 0.561

Cuadro 4.3: Evaluación MOS.

[3]Audios generados por Tacotron 2 en el español de la región central de México. `https://github.com/Ajolotl/Tacotron-2-mx`

Capítulo 5

Conclusiones y trabajo futuro

El objetivo era obtener resultados naturales que fueran indistinguibles de las grabaciones humanas del español hablado en la región central de México, y como se muestra en las evaluaciones MOS de los modelos M4 y N5 en el capítulo 4, el objetivo puede considerarse como cumplido.

Dentro de las mejoras, se encontró que el correcto ajuste de los hiperparámetros start decay y decay steps es una ventaja en cuanto al tiempo de entrenamiento y calidad de los resultados en los modelos pre-entrenados. Esto se puede notar en la diferencia de la naturalidad del modelo M1 y el modelo M4. La pendiente en el decaimiento del learning rate parece ser una interesante oportunidad de estudio y mejora para las estrategias de transfer-learning.

Una recomendación es siempre usar capas pre-entrenadas en los sistemas TTS, sin importar que se encuentren entrenadas en otro idioma, esto disminuye el tiempo de entrenamiento y mejora la calidad de la voz. El entrenamiento del modelo M4 fue tan exitoso, que éste al trabajar sin Wavenet, da como resultado un sintetizador sumamente rápido al momento de generar audios, con una inteligibilidad y naturalidad aceptable.

También se encontró una importante relación entre los hiperparámetros start decay, decay steps y el interesante comportamiento en la reconstrucción de la alineación location-based pre-entrenada, la cual varía respecto al valor de los hiperparámetros y el número de capas. Detectar una temprana y correcta alineación puede servir como indicador al momento de abortar o continuar un entrenamiento, lo que puede ahorrar mucho tiempo en modelos que requieren días o semanas para entrenarse.

Este trabajo también presenta el modelo N5 que es una mejora de Tacotron 2,

el cual consigue una inteligibilidad y naturalidad similar a la del modelo original con solo el %80.66 de los parámetros. La propuesta de transferir el conocimiento de una red neuronal a una copia de sí misma con menos capas parece funcionar cuando se busca ahorrar tiempo de entrenamiento para nuevos experimentos y encontrar la arquitectura óptima para reducir los recursos de cómputo necesarios, siempre y cuando se preserven las capas de mayor importancia para el aprendizaje. Se propone entrenar el modelo N5 sin capas pre-entrenadas para mostrar si se puede obtener la misma calidad, además de experimentar con todas las capas hasta encontrar la configuración de Tacotron 2 con el menor número de parámetros posibles.

Finalmente, ya conociendo el comportamiento de estos sistemas, el trabajo a futuro plantea proponer una mejor arquitectura de predicción de características lingüísticas y acústicas que trabaje aún con menos parámetros. En los experimentos con reducción de capas se muestra como algunos bloques de la red como el mecanismo de atención, son más importantes que otros, los resultados indican que cuando la red aprende una correcta alineación, no son necesarias tantas capas de convoluciones para mantener dependencias de largo rango en las salidas del modelo. Por esta razón, también se propone dar más prioridad al estudio y mejora de los mecanismos de atención en redes neuronales.

Abreviaturas

GPU Graphics Processing Units. 31, *Glossario:* GPU

NMT Neural Machine Translation. 10, *Glossario:* NMT

PCA Principal Component Analysis. 11, *Glossario:* PCA

SGD Stochastic gradient descent. 7, *Glossario:* SGD

SPSS Statistical parametric speech synthesis. 1, *Glossario:* SPSS

TTS text-to-speech. 1, *Glossario:* TTS

Glosario

decay steps Instancias de tiempo que tarda el learning rate inicial en alcanzar un learning rate de menor valor. 38

end-to-end Modelo que se entrena para aprender a mapear una entrada de longitud fija a una salida de longitud fija donde la longitud de la entrada y de la salida difieren. 15

GPU Procesadores especializados originalmente creados para tareas de gráficos de computadora. 31

learning rate Hiperparámetro que controla cuanto se actualizan los pesos en un modelo de acuerdo al error estimado. 7

NMT Uso de redes neuronales profundas para la traducción de texto. 10

PCA Método que encuentra las direcciones de máxima varianza en datos de alta dimensión para proyectarlos en un nuevo sub-espacio de menor dimensión. 11

SGD Método iterativo para optimizar alguna función objetivo. 7

sobreajuste Fenómeno que ocurre cuando la brecha entre el error de entrenamiento y el error de prueba es muy grande. 9

SPSS Uso de modelos paramétricos para generar conjuntos de segmentos de voz que suenan de manera similar. 1

start decay Instancia de tiempo durante el entrenamiento donde el learning rate empieza a decaer. 38

Tensorflow Biblioteca open-source para el desarrollo de aprendizaje máquina. 23

TTS Sistema que transforma texto a señales de audio. 1

Referencias

Agiomyrgiannakis, Y. (2015). Vocaine the vocoder and applications in speech synthesis. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4230–4234. IEEE. 1

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*. , 11, 12, 13, 14, 15

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828. 3, 4

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155. 10

Bengio, Y., Simard, P., Frasconi, P., et al. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166. 6

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*. 10, 11, 12

Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition. In *Advances in neural information processing systems*, pages 577–585. , 13, 14, 15, 27

Fan, Y., Qian, Y., Xie, F.-L., and Soong, F. K. (2014). Tts synthesis with bidirectional lstm based recurrent neural networks. In *Fifteenth Annual Conference of the International Speech Communication Association*. 1

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`. , 3, 4, 5, 7

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*. 14

Griffin, D. and Lim, J. (1984). Signal estimation from modified short-time fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):236–243. 22

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116. 6

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780. 6

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*. 9

Ito, K. (2017). The lj speech dataset. `https://keithito.com/LJ-Speech-Dataset/`. 31

Kalchbrenner, N. and Blunsom, P. (2013). Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709. 10, 12

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2016). Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*. , 10, 11

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. , 7, 8

Krueger, D., Maharaj, T., Kramár, J., Pezeshki, M., Ballas, N., Ke, N. R., Goyal, A., Bengio, Y., Courville, A., and Pal, C. (2016). Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*. 9

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*. , 10

Luong, M.-T., Sutskever, I., Le, Q. V., Vinyals, O., and Zaremba, W. (2014). Addressing the rare word problem in neural machine translation. *arXiv preprint arXiv:1410.8206*. 10

Mama, R. (2019). Rayhane-mamah/tacotron-2. `https://github.com/Rayhane-mamah/Tacotron-2`. 25, 26, 27, 28

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*. 10

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press. , 4

Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359. 40

Ping, W., Peng, K., Gibiansky, A., Arik, S. O., Kannan, A., Narang, S., Raiman, J., and Miller, J. (2017). Deep voice 3: Scaling text-to-speech with convolutional sequence learning. *arXiv preprint arXiv:1710.07654*. , 21

Shen, J., Pang, R., Weiss, R. J., Schuster, M., Jaitly, N., Yang, Z., Chen, Z., Zhang, Y., Wang, Y., Skerrv-Ryan, R., et al. (2018). Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4779–4783. IEEE. , 23, 25, 29

Skerry-Ryan, R., Battenberg, E., Xiao, Y., Wang, Y., Stanton, D., Shor, J., Weiss, R. J., Clark, R., and Saurous, R. A. (2018). Towards end-to-end prosody transfer for expressive speech synthesis with tacotron. *arXiv preprint arXiv:1803.09047*. 1

Smith, S. L., Kindermans, P.-J., Ying, C., and Le, Q. V. (2017). Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*. 40

Sotelo, J., Mehri, S., Kumar, K., Santos, J. F., Kastner, K., Courville, A., and Bengio, Y. (2017). Char2wav: End-to-end speech synthesis. 1

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958. 9

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112. 10, 11

Taylor, P. (2009). *Text-to-speech synthesis*. Cambridge university press. 1

Turian, J., Ratinov, L., and Bengio, Y. (2010). Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics. 10

Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *SSW*, 125. , 17, 18, 19

Wang, Y., Skerry-Ryan, R., Stanton, D., Wu, Y., Weiss, R. J., Jaitly, N., Yang, Z., Xiao, Y., Chen, Z., Bengio, S., et al. (2017). Tacotron: Towards end-to-end speech synthesis. *arXiv preprint arXiv:1703.10135*. , 1, 22, 23

Ze, H., Senior, A., and Schuster, M. (2013). Statistical parametric speech synthesis using deep neural networks. In *2013 ieee international conference on acoustics, speech and signal processing*, pages 7962–7966. IEEE. 1

Zen, H. and Sak, H. (2015). Unidirectional long short-term memory recurrent neural network with recurrent output layer for low-latency speech synthesis. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4470–4474. IEEE. 17

Zen, H., Tokuda, K., and Black, A. W. (2009). Statistical parametric speech synthesis. *speech communication*, 51(11):1039–1064. 1

Zoph, B., Yuret, D., May, J., and Knight, K. (2016). Transfer learning for low-resource neural machine translation. *arXiv preprint arXiv:1604.02201*. 40