Implementación de DeepSpeech, un modelo de reconocimiento del habla end to end

Jaime Ferrando Huertas, Javier Martínez Bernia

Febrero 2021

Contents

1 Introducción

En este trabajo hemos evaluado un sistema de reconocimiento del habla completo (end-to-end) basado en redes neuronales. Estos sistemas implementan todo el modelado en una única red neuronal que recibe la señal acústica y devuelve la transcripción. El sistema elegido ha sido DeepSpeech [3] y lo hemos implementado en el toolkit PyTorch [9].

En los últimos años ha habido un gran aumento de modelos completos de reconocimiento del habla basados en redes neuronales, a partir de los grandes avances en el campo de la traducción automática. Estos modelos llegan para combatir con los llamados modelos "híbridos", que son modelos que se dividen en dos partes: un modelo acústico y un modelo de lenguaje.

$$\hat{W} = \underset{W}{\operatorname{argmax}}\, P(O \mid W)P(W)$$

Esta arquitectura híbrida tiene una gran historia y se ha ido perfeccionado a lo largo del tiempo. Inicialmente el modelo acústico era un *Hidden Markov Model* donde se estimaban las probabilidades de transición con mixturas de gaussianas. Ahora, se ha sustituido esa estimación por modelos de redes neuronales recurrentes. El modelo de lenguaje solía estar basado en n-gramas y ha evolucionado también a modelos de redes recurrentes o Transformer[11].

Los modelos híbridos han evolucionado poco a poco hacia las arquitecturas neuronales y solo guardan la separación entre modelo acústico/lenguaje como principal distinción a los modelos end-to-end.

Existen multitud de arquitecturas end-to-end basadas en redes neuronales. Por ejemplo, compañías como Google o Facebook dedican muchos recursos a la investigación en este área. Se ha elegido DeepSpeech debido a su sencilla implementación, aunque haremos una implementación propia modificando algunas partes de la arquitectura original.

2 Modelo DeepSpeech original

Los mejores sistemas de reconocimiento del habla se basan en módulos compuestos de múltiples algoritmos y niveles con mucha mano de obra ingenieril para su construcción. El modelo DeepSpeech reemplaza estos niveles de procesamiento utilizando técnicas de Deep Learning.

Este modelo se basa en una red neuronal recurrente (RNN) de gran tamaño, que se entrena con miles de horas de datos. Con esta construcción se toma ventaja de la capacidad que proporcionan los sistemas de Deep Learning para aprender a partir de grandes volúmenes de datos. El modelo se ha entrenado para generar transcripciones, pero con suficientes datos y potencia computacional, es capaz de aprender a ser robusto frente al ruido o la variación del locutor. La arquitectura RNN se ha elegido específicamente para que acoplara bien con las GPUs, utilizando un esquema de partición para mejorar el paralelismo. En los siguientes subapartados se presenta el modelo con más detalle.

2.1 Arquitectura RNN

El modelo tiene como entrada las pronunciaciones. Cada pronunciación $x^{(i)}$ es una secuencia temporal donde cada trozo es un vector de características del sonido. Se usan los espectrogramas como características, por lo que cada valor de cada vector de características en un tiempo determinado es la potencia de la señal en una determinada banda de frecuencia. El objetivo de la red recurrente es convertir la secuencia de entrada en una secuencia de probabilidades asociadas a los caracteres.

La arquitectura original de la red está formada por cinco capas. Las tres primeras capas no son recurrentes, y actúan en paralelo. Cada una de ellas tiene como entrada el espectrograma de la señal junto con un contexto en ambos lados. A continuación, se pasa a una capa recurrente bidireccional, la cual tiene dos conjuntos de neuronas: uno que procesa la entrada de izquierda a derecha y otro de derecha a izquierda. Finalmente, la última capa es una capa densa totalmente conectada y toma las salidas de la parte bidireccional como entradas. La salida tiene una función de activación *Softmax* que convierte las salidas en probabilidades asociadas a los caracteres del alfabeto.

Dada una predicción a partir de una entrada, se calcula la función de pérdida CTC, la cual mide el error en la predicción. Este error se propaga por la red con el algoritmo *back-propagation*.

Este modelo de red es más simple que los modelos híbridos. Se utiliza solamente una capa recurrente para adelantar el proceso de entrenamiento y facilitar el paralelismo. Las redes LSTM requieren más cómputo y espacio en cada paso de ejecución, por lo que no se utilizan en esta arquitectura.

2.2 Modelo del lenguaje

Cuando se entrena el modelo con grandes cantidades de datos etiquetados, la red recurrente aprende a producir transcripciones a nivel de carácter que son leíbles. En muchas de las transcripciones que realiza este modelo, la secuencia de caracteres más probable que predice la RNN es correcta sin la necesidad de restricciones del lenguaje externas. La mayoría de errores tienden a ser errores leves y suelen ocurrir en palabras que raramente aparecen o no aparecen en el conjunto de entrenamiento. Este problema es bastante difícil de evitar. Para solucionarlo o intentar evitarlo se puede integrar el sistema con un modelo de lenguaje de n-gramas, ya que estos modelos se pueden entrenar fácilmente a partir de datos no etiquetados. Con ello, dada una salida de la red recurrente RNN, se hace una búsqueda para encontrar la secuencia de caracteres más probable según la salida de la red y el modelo de lenguaje, donde este último interpreta la cadena de caracteres como palabras. En resumen, en este momento tenemos un problema de búsqueda de una secuencia que maximice una función multiobjetivo: la probabilidad de la secuencia según la red y según el modelo de lenguaje. Para realizar esta búsqueda típicamente se aplican técnicas como la búsqueda en haz.

2.3 Optimizaciones

Para mejorar la velocidad de ejecución del entrenamiento de las redes que se van a emplear se toman diversas decisiones de diseño. En el modelo original, se opta por una implementación con dos niveles de paralelismo. Por una parte, cada GPU procesa diversas muestras en paralelo. Por otra, cuando se procesan *minibatches* tan grandes cuyo espacio no puede soportar una única GPU, se reparten estos *minibatches* de muestras entre las distintas GPUs disponibles.

El paralelismo en los datos no es fácil de implementar, ya que cuando las pronunciaciones tienen distinta longitud no se pueden combinar todo en una simple multiplicación matricial. Este problema se resuelve ordenando las muestras por tamaños y combinando solamente las pronunciaciones que tienen tamaños similares.

El modelo es difícil de paralelizar debido a la arquitectura de las redes recurrentes y su naturaleza secuencial. Lo que se hace en el modelo original es partir la red bidireccional y calcular por separado las dos secuencias para luego combinarlas.

2.4 Datos de entrenamiento

Para extender el número de datos de entrenamiento aún más, se crean muestras sintetizadas. El objetivo es mejorar el rendimiento en entornos ruidosos, donde los sistemas tradicionales tienen un punto débil. Estas muestras de audio sintetizadas se generan mediante un proceso de superposición de señales. Se utiliza una señal de habla y una señal ruidosa y se suman ambas para simular el sonido capturado en un entorno ruidoso. Además, se pueden añadir efectos y las señales ruidosas suelen cambiar para que no sean las mismas en todas las muestras sintetizadas.

3 Implementación del modelo

En la anterior sección se ha revisado la arquitectura oficial del modelo DeepSpeech que se presenta en su artículo [3]. En esta sección se presenta la implementación realizada de este modelo para este trabajo.

3.1 DeepSpeech 2

El modelo que se ha presentado anteriormente es el modelo DeepSpeech inicial, que se presentó en 2014. Un año después, se presentó la versión DeepSpeech 2 en [1]. Esta incorporaba muchas mejoras, pero principalmente cambiaba la arquitectura. En este artículo el modelo tiene la siguiente forma:

- Una o más capas convolucionales
- Una o más capas recurrentes
- Una o más capas *fully connected*

Con ello, se pretendía aumentar la capacidad del modelo añadiendo más capas.

3.2 Implementación realizada

Para la realización de este trabajo, se ha hecho una modificación que estaría en un camino entre la versión inicial y la versión 2. Se ha modificado la arquitectura del modelo con capas más avanzadas, pero preservando la arquitectura principal. Estos cambios no son invención nuestra, sino una colección de mejoras que se han visto en el campo del Reconocimiento Automático del Habla.

Nuestra implementación comienza aplicando dos capas convolucionales a la entrada con conexiones residuales. Con ello se consiguen encontrar dependencias espaciales y temporales en las señales de entrada, además de evitar el problema del desvanecimiento del gradiente con las conexiones residuales. A continuación, se hace uso de tres redes GRU bidireccionales, las cuales son distintas de las RNN que se utilizan en la arquitectura original, pero en general consiguen mejores resultados que las redes recurrentes simples en este tipo de tareas donde se procesan secuencias que dependen de la historia pasada. Esta mejora se consigue gracias al uso de puertas que deciden si dejar (o no) pasar cierta información pasada para usarla en la predicción actual. Finalmente, se usa una capa *fully connected* tras las GRU para realizar la clasificación de los caracteres en cada paso de ejecución. Este modelo nos recuerda a las tradicionales arquitecturas Encoder-Decoder. El Encoder estaría formado por las capas convolucionales y el decoder sería la parte recurrente con GRUs bidireccinales.

En la siguiente figura podemos ver un esquema de la implementación que se pretende construir.

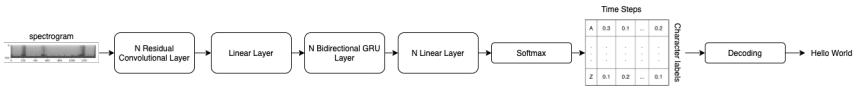

Figure 1: Esquema de la arquitectura DeepSpeech.

4 Experimentación

Tras tener claro el modelo que se va a construir, se procede a presentar el proceso de experimentación que se realizará para comprobar el funcionamiento del modelo, entrenándolo con unos datos que se comentarán a continuación, para una posterior evaluación del rendimiento.

4.1 Datos

Para probar el funcionamiento de la implementación realizada se utilizará el corpus LibriSpeech [8]. Este corpus contiene audio de lectura en inglés, y está generado a partir de audiolibros que forman parte del proyecto LibriVox. Está

formado por un total de 1000 horas de grabación muestreada a 16 kHz, que ha sido segmentada y alineada con mucho cuidado. Este corpus contiene múltiples conjuntos:

- Entrenamiento: train-clean-100, train-clean-360, train-other-500
- Validación: dev-clean, dev-other
- Test: test-clean, test-other

Se utilizarán todos los conjuntos de entrenamiento para entrenar el modelo, un conjunto de validación para la *early patience policy* y un conjunto de test para evaluar métricas en el modelo una vez entrenado.

4.1.1 Preproceso de los datos

Para entrenar el modelo tenemos que procesar los ficheros de audio para tenerlos en un formato que poder entregar a la red. Para ello, se aplicarán los coeficientes cepstrales de Mel (MFCC) [5], los cuales, a partir de una combinación de operaciones matemáticas, transforman la señal de audio en una representación multidimensional decorrelada que captura las frecuencias del oído humano.

En la siguiente figura podemos ver un ejemplo de representación de una señal de audio tras aplicarle los coeficientes cepstrales de Mel (Cepstrum).

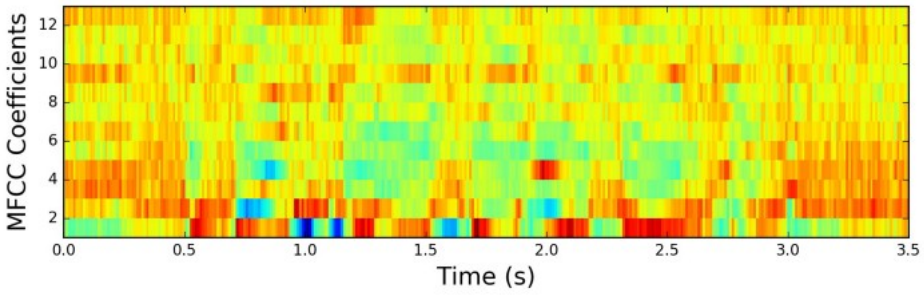

Figure 2: Representación de una señal de audio tras aplicarle el banco de filtros de la escala Mel.

4.2 Entrenamiento

Tras tener los datos preparados, es el momento de entrenar el modelo. Para la implementación del modelo como para el entrenamiento del mismo, se ha utilizado la herramienta PyTorch. En este caso, se ha utilizado la versión con soporte para CUDA [7], una librería que permite ejecutar la computación aprovechando la capacidad y rapidez de cálculo de las GPUs. Se han lanzado experimentos en una máquina local con una GPU.

Tras crear el modelo, se han definido distintos parámetros para el entrenamiento. En primer lugar, se ha decidido utilizar AdamW como optimizador

de la función objetivo. AdamW [4] es una versión del optimizador Adam con un algoritmo alternativo para la disminución de los pesos. Como *scheduler* para el factor de aprendizaje se ha decidido utilizar OneCycleLRE [10] con un valor inicial de *learning rate* de 0.0005. Este *scheduler* tiene un periodo inicial donde el factor de aprendizaje llega a un máximo y a continuación disminuye exponencialmente. Se ha entrenado el modelo con un tamaño de *batch* de 32 y se ha evaluado con el conjunto de validación al final de cada *epoch*. Se ha aplicado lo que se conoce como *patience policy* con un valor de paciencia de 5, es decir, hemos esperado 5 *epochs* como máximo sin que el modelo mejorara significativamente antes de parar el entrenamiento. Esto nos ha llevado a un entrenamiento de 40 *epochs*.

4.3 Métricas de evaluación

Las métricas más utilizadas en Reconocimiento Automático del Habla son *Character Error Rate* (CER) y *Word Error Rate* (WER). Estas métricas representan el error de clasificación de nuestro modelo a dos niveles, a nivel de carácter y a nivel de palabra. La métrica CER es comúnmente utilizada cuando los modelos no tienen un modelo de lenguaje al final que ayuda a que las palabras de salida sean palabras que formen de un vocabulario. Por otra parte, la métrica WER toma más importancia cuando se utilizan estos modelos de lenguaje, ya que ayuda a evaluar tanto el rendimiento del sistema de reconocimiento del habla como el modelo de lenguaje.

En la siguiente sección, se presentarán los resultados del modelo que hemos construído utilizando ambas métricas, CER y WER, aunque se esperan peores resultados en WER debido a que no se está utilizando un modelo de lenguaje al final del modelo.

4.4 Resultados

Tras entrenar el modelo implementado, se han evaluado las métricas comentadas en el apartado anterior con los dos conjuntos de test y se han obtenido los siguientes resultados:

Conjunto de test	WER	CER
test-clean	13.67%	4.87%
test-other	32.33%	14.95%

Table 1: Resultados con nuestra implementación del modelo DeepSpeech.

No hay benchmarks públicos del corpus LibriSpeech de los autores originales, pero se han encontrado otras implementaciones [6] que están evaluadas con el corpus LibriSpeech. Sus resultados son los siguientes:

Conjunto de test	WER	CER
test-clean	10.46%	3.39%
test-other	28.28%	12.03%

Table 2: Resultados públicos sobre el conjunto LibriSpeech

Como se puede ver en la tabla anterior, los resultados obtenidos por otras implementaciones superan a los que hemos obtenido en este trabajo, aunque las diferencias no son demasiado grandes. Sabemos que la optimización de los hiperparámetros de las redes neuronales puede afectar fuertemente a su rendimiento, especialmente con redes del tamaño de DeepSpeech, por lo que creemos que con una mejor optimización de los parámetros del modelo se podrían llegar a conseguir mejores resultados. Desgraciadamente, esto estaría fuera de los objetivos de este trabajo, ya que este simple experimento llevó tres días y medio para su ejecución, por lo que para conseguir un mejor ajuste de parámetros necesitaríamos un cluster de GPUs o más tiempo para realizar pruebas.

5 Conclusiones

Este ha sido nuestro primer contacto con arquitecturas end-to-end para el reconocimiento del habla y estamos muy contentos con el contenido explorado. En los laboratorios de clase solo se vieron modelos híbridos y gracias a este trabajo hemos visto la otra cara de la moneda. Los resultados de DeepSpeech no son mejores que los obtenidos con modelos híbridos pero marcaron una base sobre la que los modelos end-to-end han ido creciendo hasta conseguir liderar el ranking de WER en LibriSpeech con modelos como el Conformer[2]. Implementar esta arquitectura ha sido un desafío para nosotros pero hemos aprendido bastante sobre PyTorch. Ahora nos gustaría intentar replicar la mejor arquitectura híbrida (HMM-DNN con modelo de lenguaje transformers) y la mejor arquitectura end-to-end (conformer) para evaluar cómo de viable es su implementación y ventajas/desventajas de entrenar estos modelos. Estos aspectos también deber ser evaluados cuando elegimos nuestro modelo y no solo fijarnos en el WER. Pensamos que ahora, gracias a la popularidad de modelos neuronales, puede que existan más herramientas para desarrollar modelos end-to-end pero Kaldi también es un tool-kit muy famoso para arquitecturas híbridas y merece la pena explorarlo. En conclusión, ha sido un proyecto extremadamente ilustrador y nos has servido de primer contacto con modelos neuronales end-to-end.

References

[1] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse H. Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Y. Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman,

Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. *CoRR*, abs/1512.02595, 2015.

[2] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang. Conformer: Convolution-augmented transformer for speech recognition, 2020.

[3] Awni Y. Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014.

[4] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.

[5] S. Molau, M. Pitz, R. Schluter, and H. Ney. Computing mel-frequency cepstral coefficients on the power spectrum. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, volume 1, pages 73–76 vol.1, 2001.

[6] Sean Naren. Deepspeech. https://github.com/SeanNaren/deepspeech.pytorch, 2020.

[7] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020.

[8] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, 2015.

[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[10] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of residual networks using large learning rates. *CoRR*, abs/1708.07120, 2017.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.