- PRÁCTICA 1 -
DEEP LEARNING IN ASR

Reconocimiento Automático del Habla

Autora
Aitana Menárguez Box
Enero 2024

1 Introducción

El objetivo de esta memoria es documentar el trabajo realizado en la primera práctica de laboratorio de Reconocimiento Automático del Habla. En esta práctica se pretende construir, a partir de los cuadernos proporcionados, un sistema que reconozca una secuencia de dígitos de un audio y que sea capaz de representarla con una codificación específica, junto con la suma de los dígitos que forman dicha secuencia, a partir de un modelo transformer.

La representación final de la secuencia reconocida debe un vector de 20 elementos con la siguiente forma: `[20,` *secuencia-reconocida* `,19,` *secuencia-suma-dos-primeros-elementos* `,18,` *secuencia-suma-dos-últimos-elementos* `,21,` *secuencia-suma-todos-elementos* `, 22, 23, 23, ..., 23]`. En concreto, si la entrada es un audio de la pronunciación de la secuencia numérica `4 8 9 6`, la salida debería ser:
`[20, 4, 8, 9, 6, 19, 1, 2, 18, 1, 5, 21, 2, 7, 22, 23, 23, ..., 23]` ,
ya que $4 + 8 = 12$, $9 + 6 = 15$ y $4 + 8 + 9 + 6 = 27$.

En caso de tratarse de una secuencia de menos de cuatro dígitos, se abordará de forma diferente según se explica, a partir del código realizado, más adelante.

2 Trabajo realizado

2.1 Partes básicas del *transformer*

Para construir el modelo transformer con el cual se trabajará, es necesario abordar las diferentes partes que lo conforman. En primer lugar, se tratará el **encoder**. Éste se trata de un módulo de *pytorch* con un número configurable de capas (por defecto 6) y una longitud de secuencia dada (por defecto 400). Cada capa contiene una capa de *self-attention*, en la cual se capturan las relaciones entre cada uno de los elementos de la secuencia y el resto, y una capa *feed-forward*. Antes de aplicar estas dos partes, se codifica cada elemento teniendo en cuenta su posición en la secuencia.
A continuación, tenemos el **decoder**. La única diferencia estructural entre ese bloque y el anterior es que en este caso también se calcula la *cross-attention*, que captura las relaciones entre los elementos de la secuencia de salida con los elementos de la secuencia de entrada.

Por otro lado, al tratarse de procesamiento del habla a partir de archivos de audio, es conveniente crear un bloque de **extracción de características** de audio (relevantes para este caso), que ayudarán en la codificación de la información de entrada. El extractor de características está compuesto por un extractor del logaritmo del *espectrograma de Mel* y una capa lineal. Éste se aplicará a la secuencia de salida antes de pasarla por el encoder.

2.2 Creación del *dataset*

En este caso se ha trabajado con los datos del directorio `data3/`, que son audios de personas pronunciando secuencias numéricas de entre 1 y 4 dígitos. Para crear el conjunto de datos, además de estos audios, necesitamos las y asociadas a cada una de las secuencias, es decir las secuencias de salida para cada uno de los audios de entrada. Aquí es donde se codificarán

las salidas para que cumpla el formato explicado en la introducción.
A partir del nombre de cada pista de audio, en el cual viene indicada la secuencia que se está pronunciando, se extraerá por escrito y se guardará en la variable `number` esta secuencia, la cual se pasará a forma de lista (cada dígito pronunciado será un elemento de la lista); la suma de los dígitos que la conforman se guardará en `sum_number`. Según la longitud de la secuencia pronunciada, la codificación será una u otra. Se hará de la siguiente manera:

```
if len(number) == 1:
    y = [20, ] + number + [19, ] + number + [21, ] + sum_number + [22, ]
elif len(number) == 2:
    y = [20, ] + number + [19, ] + num2list(sum(number)) + [21, ] +
    sum_number + [22]
elif len(number) == 3:
    y = [20, ] + number + [19, ] + num2list(number[0] + number[1]) + [18, ]
    + [number[2]] + [21, ] + sum_number + [22]
else:
    y = [20, ] + number + [19, ] + num2list(number[0] + number[1]) + [18, ]
    + num2list(number[2] + number[3]) + [21, ] + sum_number + [22]
```

De esta forma podremos entrenar el modelo para que dirija su aprendizaje a generar la salida que queremos.

2.3 Entrenamiento y evaluación

Tras entrenar el modelo con 20 *epochs* y un tamaño de *batch* de 32, se ha evaluado el modelo con los datos de entrenamiento a partir de la distancia de edición entre la salida deseada y la salida estimada. El resultado final es `error rate 143.00%, (24280/16979)`. Esto es debido a que en la mayoría de casos, la salida estimada contiene más elementos que la salida deseada.

A continuación, se explicará cómo se ha intentado solventar esta tasa de error tan elevada.

2.4 Ampliación

Una forma de intentar mejorar los resultados obtenidos anteriormente, puede ser el uso de *data augmentation*. Esta técnica se basa en *ampliar* la información de entrada, por ejemplo, añadiéndole ruido, para que el modelo pueda ver casos más realistas al entrenarse y no se sobre-adapte a los datos de entrenamiento; es decir, que pueda expandir su aprendizaje y tener un conocimiento más amplio del espacio de representación.
Con esto, se entrenará el modelo con los mismos audios a los cuales se les ha añadido ruido de fondo (a partir de los audios en `/RIRS_NOISES_small` y a partir de la simulación de diferentes reverberaciones de sala). Además, se entrenará el modelo con más *epochs*, para permitirle un resultado más elaborado.
El resultado obtenido ha sido `error rate 118.01%, (20037/16979)`. Si subimos el número de *epochs* a 75, el resultado es `error rate 92.36%, (15681/16979)` que, sin ser un buen resultado, ha mejorado bastante respecto de la primera ejecución.