Documentazione di Progetto del Corso Ingegneria della Conoscenza

A.A. 2020/2021

Automatic Speech Recognition (ASR) attraverso la creazione, training ed inferenza di un Recurrent Neural Network per la generazione automatica di sottotitoli

Di:

Luigi Porcelli
Matricola: 701719
l.porcelli7@studenti.uniba.it

Nicolò Cucinotta
Matricola: 717287
n.cucinotta@studenti.uniba.it

Repo Github:

https://github.com/uigiporc/icon-sr

Obiettivo:

L'obiettivo del nostro lavoro è stato quello di costruire un modello di tipo Recurrent Neural Network (RNN) e applicare metodi di apprendimento supervisionato su un dataset composto da file audio e relative trascrizioni al fine di ottenere, dati in input audio in lingua inglese di qualsiasi lunghezza, la loro trascrizione.

Questa funzionalità è stata poi applicata per generare automaticamente ed offline sottotitoli per video.

Strumenti utilizzati:

Il progetto è stato sviluppato in linguaggio Python con l'utilizzo di Tensorflow, framework open-source di Google per il machine learning.

Ulteriori librerie utilizzate sono state:

- Keras, come wrapper attorno a molte operazioni di Tensorflow, utile principalmente per la definizione del modello
- Librosa, in fase di preprocessing ed inferenza per generare gli spettrogrammi
- jiwer, per effettuare il calcolo del WER
- pyAudio, per catturare l'audio del PC in fase di inferenza

Per il training, è stato utilizzato Google Colab con accelerazione hardware TPU. Per necessità di Colab, è stato richiesto anche l'uso di un bucket Google Cloud Storage come sorgente del dataset.

Dataset e Preprocessing:

E' stato scelto come dataset di training un sottoinsieme della collezione LibriSpeech, in particolare "train-clean-360", composto da 104014 file audio di parlato inglese in accento americano per un totale di 360 ore. Quest'ultimo è inoltre privo di rumori di sottofondo e tratto da vari audiolibri di diversi lettori per evitare bias.

Per determinare l'efficacia del modello, si è usato invece "test-clean", sempre dalla stessa collezione.

Ogni traccia ha un sample rate di 16000 Hz e una durata compresa tra gli 1 e i 32 secondi, ed ha associata la relativa trascrizione contente solo i 26 caratteri dell'alfabeto inglese (tutti in maiuscolo), con l'aggiunta di apostrofo e spazio.

Sia i file audio che l'intera stringa della trascrizione non sono adatti come input di un neural network e hanno quindi bisogno di essere preprocessati. In particolare:

- Audio:
  Un buon modo di rappresentare un segnale audio è la sua decomposizione nella somma di multiple onde più semplici che lo compongono, trasformando il dominio del tempo in quello delle frequenze. Nell'atto pratico, questo viene fatto applicando a un'onda una trasformata di Fourier e, poiché il nostro segnale è campionato e quantizzato, sarà una funzione discreta invece che continua (Discrete Fourier Transform, DFT). Così facendo però, andremmo a perdere la variazione nel tempo di questo segnale. Poichè questo elemento è spesso necessario, come nel nostro caso, si opta invece per applicare questa trasformata su più finestre di tempo, solitamente sovrapposte. Il risultato è una matrice bidimensionale che associa ad ogni intervallo di tempo l'intensità delle varie frequenze che lo compongono.
  Questo processo si chiama **Short Time Fourier Transform (STFT)** e ci restituisce uno **spettrogramma**.

  In base a varie ricerche nella letteratura, invece di fermarci a questa rappresentazione, applichiamo invece una ulteriore trasformazione. Invece di utilizzare una scala in Hertz, dove le differenze tra frequenze sono costanti e uniformi, utilizziamo una scala Mel, che meglio rappresenta la percezione logaritmica delle frequenze da parte degli umani (una differenza tra 500Hz e 1000Hz è molto più marcata in una scala Mel rispetto a una differenza tra i 10500Hz e gli 11000Hz). Questo vuol dire che le frequenze del parlato verranno enfatizzate in uno spettrogramma con scala Mel rispetto a quello originario.

  Ulteriormente, gli spettrogrammi rappresentano anche l'intensità del suono. Anche in questo caso, gli umani hanno una percezione logaritmica dell'intensità, quindi applichiamo il logaritmo naturale a quello che è attualmente il nostro "mel-scaled power spectrogram", ottenendo una rappresentazione in decibel (dB) dell'intensità

  Specificatamente, nel nostro caso si è scelto di effettuare una STFT con *window* di lunghezza 25ms e *stride* di 10ms. Si è poi applicata una mel filterbank a 128 mels.

- Trascrizioni:
  Abbiamo definito un vocabolario composto da tutte le lettere maiuscole dell'alfabeto con l'aggiunta dell'apostrofo (') e dello spazio ( ), assieme alla definizione di un carattere vuoto ("") per indicare una lettera o segno non presente nel vocabolario. In totale, quindi, il vocabolario è composto da 29 grafemi. Si sono poi definite due funzioni per effettuare un mapping da carattere a indice del vocabolario e viceversa. Ogni trascrizione è stata infine convertita da stringa ad array di interi compresi tra 0 e 28 attraverso la prima delle suddette funzioni.

Infine, le coppie audio-trascrizione processate sono state serializzate e salvate in file di formato TFRecord. Al fine di sfruttare al meglio la parallelizzazione delle operazioni di I/O, sono stati generati per il nostro dataset 100 file tfrecord, ognuno di dimensione compresa

tra i 600 e i 700 MB (per un totale di circa 65 GB), come consigliato nella documentazione di Tensorflow.

Modello e training:

Il modello è ispirato da quello proposto nel paper di DeepSpeech2 (Amodei et al. 2015).

Il modello è quello di un Recurrent Neural Network (RNN) con input dato da alcuni "convolutional layer", a volte definito in alcuni elementi della letteratura come CRNN (Convolutional Recurrent Neural Network), che va a combinare i punti di forza di entrambe le architetture.

I CNN sono infatti adatti ad apprendere da immagini o, in generale, da features "spaziali". Gli RNN si specializzano invece su segnali temporali. Un problema specifico degli RNN è quello della memoria a breve termine, che all'atto pratico non permette alla rete neurale di creare collegamenti tra input distanti tra loro. E' inoltre soggetto a problemi di vanishing o exploding gradient. La soluzione consiste di sostituire le classiche celle RNN con celle LSTM (Long Short-Term Memory) o GRU (Gated Recurrent Units). Nel nostro caso la scelta ricade sulle GRU, che risultano un modello semplificato delle LSTM, ma catturano sufficienti dipendenze temporali e soprattutto risultano più efficienti di queste ultime (quindi più veloci da addestrare).

Passando quindi al nostro modello: i primi due layer sono layer di convoluzione 2D (bidimensionali) che svolgono il ruolo di "feature detector", ad ognuno dei quali viene applicato un processo di batch normalization per rendere più veloce e stabile il training. Questi output vengono poi passati a 5 layer GRU bidirezionali con 256 celle ciascuno , ognuno con un relativo dropout per evitare fenomeni di overfitting. Infine, il tutto viene passato a due "fully connected layer", ottenendo in output attraverso la funzione di attivazione softmax le probabilità per ogni carattere ad ogni timestep (anche detti logits).

Si usa inoltre la Connectionist Temporal Classification Loss (CTC) come funzione di loss (Graves et al. 2006). Quest'ultima ci permette infatti di non necessitare di una separazione della label ad ogni timestep, bensì permette al network di apprendere e suddividere o raggruppare in maniera autonoma i grafemi.

Infine, viene utilizzato Adam come ottimizzatore con un learning rate costante a seguito dei buoni risultati ottenuti empiricamente rispetto a Stochastic Gradient Descent (SGD).

Il risultato finale è una rete neurale con 7,2 milioni di parametri addestrabili. L'obiettivo di questo network non è infatti un'accuratezza paragonabile a quella dello stato dell'arte attuale, ma la semplicità e velocità in fase di deployment (o anche edge computing), permettendo una veloce ed efficiente generazione di sottotitoli anche su macchine poco potenti.

Passando alla fase di training, si è optato per una suddivisione 80/20, in cui l'80% del dataset viene usato per training e il 20% in fase di valutazione, ed una batch size pari a 16 esempi per ogni unità di computazione al fine di minimizzare i tempi di training. Per valutare l'efficacia del modello, si è usato come metrica il Word Error Rate.

Passiamo adesso ad alcuni dettagli specifici all'addestramento su TPU:

- Su TPU non vi è il supporto a SparseTensors, ma solo a DenseTensors (tensori 0-padded). La funzione di calcolo della CTC loss di Keras ha un'implementazione unicamente con tensori sparsi, rendendo quindi necessario scrivere un wrapper attorno alla funzione a basso livello per il calcolo su tensori densi fornita da Tensorflow.
- Poichè il modello viene compilato in un grafo XLA una singola volta per motivi di ottimizzazione su TPU, è necessario specificare le dimensioni di input per ogni elemento di un batch, anche se in un addestramento su GPU sarebbe possibile ometterle e avere dimensioni di input diverse per ogni batch. Sono state scelte dimensioni multiple di 128 per ogni dimensione, poiché le TPU lavorano su matrici 128x128. Per esattezza, ogni input ha un padding per lo spettrogramma pari a 3328 timesteps (33.28 secondi), mentre la trascrizione ha un padding a 640 caratteri. Batch size e numero di feature (numero di bins nella mel filterbank) sono già pari a 128.

Abbiamo implementato inoltre una callback tale che alla fine di ogni epoch calcolasse il WER sulle predizioni di un campione da 5 batch del dataset di valutazione(nel caso di addestramento con TPU pari 128*5 = 640 esempi), ci mostrasse 2 delle predizioni, e poi salvasse in cloud il modello.
Questo è risultato fondamentale perché ogni epoch ha richiesto circa 60 minuti di training su una VM con accelerazione TPU offerta da Colab, necessitandoci quindi di suddividere il training su più giornate.

Il risultato finale, addestrando per 79 epoch e testando sul dataset test-clean, è un modello con WER pari a 21.07% (calcolato sulla media di 3 run).

Inferenza:

L'inferenza avviene mediante la creazione di due thread, entrambi con un preciso scopo: il primo thread ha il compito di registrare l'audio in uscita dalla scheda audio, e di inserire la lista di byte catturati all'interno di una Queue; il secondo thread si occuperà, invece, di processare lo spettrogramma in scala Mel del frammento di audio recuperato dalla Queue, e successivamente di inferire la frase pronunciata in tale frammento di audio.

La divisione del processo di inferenza in due thread paralleli è necessaria affinché non si verifichi una perdita di informazione (audio): senza parallelismo, ci sarebbe una perdita di

audio pari al tempo impiegato dal calcolatore a processare ed effettuare inferenza sul frammento di audio precedente.

Si è scelta, a seguito di vari test, una durata di 3 secondi per ogni finestra di inferenza per fornire un'adeguata responsiveness in quella che è progettata per essere un'applicazione "live".

Una peculiarità per l'acquisizione dell'audio di output su piattaforma Windows è la necessità di abilitare e impostare come dispositivo di default lo Stereo Mix, per cui si rimanda a questa guida.

Conclusioni:

Si è voluto dimostrare in questo caso la ancora attuale rilevanza di un modello basato su RNN con assenza di Language Models esterni, soprattutto nella rapidità del deploy. Rispetto allo stato dell'arte basato su modelli di tipo Transformer-Conformer, risulta infatti molto più leggero (il nostro modello finale pesa meno di 100MB) e più veloce in fase di inferenza, prestandosi ad un deployment su dispositivi a bassa potenza computazionale (ci permette di inferire 30 secondi di audio in circa 6 secondi su una macchina dual core), mantenendo comunque un livello rispettabile di accuratezza.

E' inoltre una proof of concept per dimostrare le possibilità di un sistema di trascrizione vocale in tempo reale offline (offline streaming ASR), utile ai non udenti che usano un PC e vogliono vedere video non sottotitolati o partecipare a conversazioni dal proprio dispositivo in modo quasi trasparente all'utilizzo del PC.

Ulteriori studi potrebbero essere fatti aumentato il numero di dati mandati in input al modello (usando ad esempio l'intero catalogo di 960h di LibriSpeech) o usando metodi di "augmenting", come SpecAugment. Risulterebbero efficaci anche test per l'introduzione di algoritmi di riduzione del rumore applicati in fase di inferenza.