Language Translator

Rocco Iuliano, Simone Della Porta

Intelligenza Artificiale A.A. 2023/2024

Contents

1 Introduzione

Il progetto prevede la realizzazione di un language translator in grado di tradurre in modo automatico una frase scritta in qualsiasi linguaggio in lingua inglese. Per la sua realizzazione è stata effettuata una ricerca di fonti online per l'acquisizione dei dati necessari per definire un dataset di addestramento, ovvero un corpora. Successivamente è stato realizzato un modello di NLP per la realizzazione del task calcolando le opportune metriche di valutazione. Infine, sono state confrontate le prestazioni del modello ottenuto rispetto ai modelli dello stato dell'arte selezionati.

2 Dataset

Dopo una ricerca di fonti online per la realizzazione del corpora del progetto, abbiamo individuato Hugging Face come una valida fonte per la definizione del dataset. Tra i vari corpora messi a disposizione dalla piattaforma, abbiamo selezionato il dataset denominato *opus-books* che contiene vari sottodataset di traduzione da un linguaggio sorgente a un linguaggio di destinazione identificati con [linguaggio_sorgente]_[linguaggio_destinazione]. A partire da questo corpora, abbiamo deciso di selezionare i principali linguaggi europei, quindi abbiamo estrapolato i seguenti sottodataset:

- *en-fr* contenente 127085 corpus;
- *en-ru* contenente 17496 corpus;
- *en-it* contenente 32332 corpus;
- *en-pl* contenente 2831 corpus;
- *en-fi* contenente 3645 corpus;
- *en-nl* contenente 38700 corpus;
- *en-pt* contenente 1404 corpus;
- *de-en* contenente 51500 corpus.

Infine, per addestrare i vari modelli di traduzione abbiamo suddiviso ogni dataset in tre sottodataset, ovvero train, test e valid assegnando rispettivamente il 70%, 20% e 10% dei dati.

3 Implementazione

3.1 Language translator: Ensamble learning

Sulla base dei dati a disposizione abbiamo deciso di applicare la tecnica di *Ensamble learning* in modo da avere a disposizione diversi language translator,

ognuno specializzato per un determinato task di traduzione. Questa scelta è dovuta principalmente alla dimensione dei dati a disposizione, alla differenza di lunghezza dei testi tra i vari dataset e alla potenza di calcolo a disposizione. Se avessimo optato per la realizzazione di un unico modello, avremmo dovuto unire i vari corpora e definire un'unica lunghezza per gli embedding causando la presenza di un eccessivo padding in molte frasi di train. Questa tecnica, oltre a mitigare il problema precedente, ci ha consentito di poter addestrare più modelli contemporaneamente su macchine differenti e ci consentirà, eventualmente in futuro, di poter riaddestrare un singolo language translator in modo da poter migliorare le proprie prestazioni senza la necessità di riaddestrare il tutto, evitando così spreco di tempo e risorse. Successivamente, è stato deciso di unificare questa serie di modelli in un singolo blocco di language translator, il quale è preceduto da un language identifier in grado di riconoscere la lingua del testo in input e instradare il testo al modello specializzato. Questo consente all'utente di percepire l'interazione con un unico modello di traduzione.

3.2 Language translator: Transformer

Per la realizzazione del modello di traduzione, abbiamo deciso di realizzare un'architettura *Transformer* in quanto presenta vari vantaggi come una buona gestione del contesto di una frase e l'estrapolazione di informazioni rilevanti all'interno di quest'ultima. Per definire gli embedding da fornire in input al modello, abbiamo utilizzato la classe *Embedding* fornita dalla libreria *torch.nn* e abbiamo definito un *Positional embedding* per tenere traccia della posizione delle parole nel testo. L'output di queste due classi verrà sommato in modo da ottenere l'embedding finale da fornire in input al blocco di encoding della rete transformer. Per quanto riguarda la rete, abbiamo definito 6 blocchi consecutivi di encoding e 6 blocchi consecutivi di decoding definiti nel seguente modo:

- **Encoder block**
  - Multihead attention layer per catturare il contesto della frase;
  - Residual connection layer con normalizzazione in modo da inoltrare le informazioni ai layer successivi e di normalizzare i dati;
  - Feed forward layer.
- **Decoder block**
  - Multihead attention layer;
  - Cross multihead attention layer che combina l'output del blocco di encoder con i dati processati dal blocco di decoder della rete in modo da considerare l'intera frase durante tutta la fase di decoding. Questo consente di estrapolare informazioni importanti dal testo in input per la generazione della parola successiva in fase di decoding;
  - Feed forward layer;
  - Resisual connection con normalizzazione.

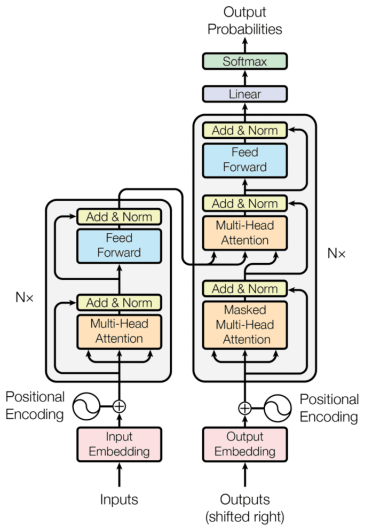


Figure 1: Architettura transformer

- **Projection layer** per mappare l'output della rete nelle rispettive parole del vocabolario della lingua inglese.

La Figura 1 mostra l'architettura interna di una rete transformer. Per ogni modello sono state eseguite 20 epoche di addestramento con una *batch size* pari a 8, un *learning rate* pari a 10^{-4} e una dimesione dell'head pari a 512. Come riportato in precedenza, per evitare embedding con eccessivo padding sono state definite lunghezze diverse per ogni modello. La tabella 1 riporta le informazioni relative alla dimensione degli embedding e le relative lunghezze massime delle sentenze sorgenti e target.

Table 1: caption

Language translator	Lunghezza max. sentenza sorgente	Lunghezza max. sentenza target	Dimensione embeddign
fr-en	482	471	485
ru-en	195	229	350
it-en	274	309	350
pl-en	158	556	560
fi-en	64	141	200
nl-en	298	656	660
pt-en	196	204	210
de-en	479	466	485

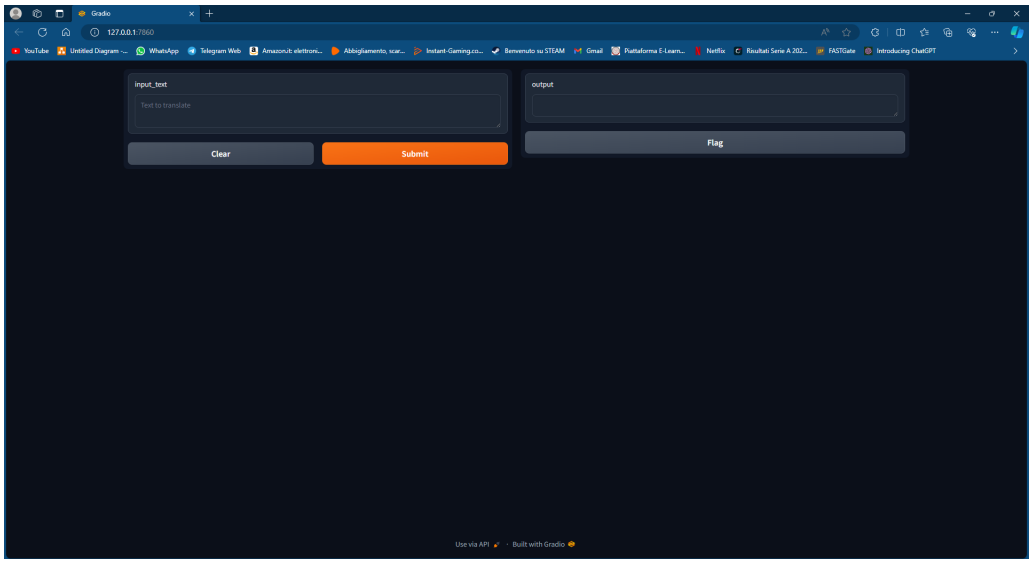


Figure 2: Interfaccia grafica con Gradio

3.3 Language translator: Language identifier

Come menzionato nella Sezione 3.1, per applicare la tecnica dell'ensamble learning, abbiamo unificato i vari modelli addestrati in un singolo blocco di language translator, il quale è preceduto da un language identifier. Quest'ultimo è stato realizzato tramite l'utilizzo del modello preaddestrato fornito dalla libreria Python *langid* che fornisce in output l'acronimo della lingua predetta e la relativa confidence.

3.4 Language translator: User Interface

Per permettere all'utente di utilizzare le nostre reti transformer e dargli la percezione di star interagendo con un singolo modello, abbiamo utilizzato la libreria *gradio* che fornisce una semplice interfaccia grafica attraverso la quale può fornire l'input al traduttore e visualizzare l'output. La Figura 2 mostra un esempio di interazione.

3.5 Language translator: pretrained model

Per confrontare le prestazioni dei language translator da noi realizzati con lo stato dell'arte, abbiamo cercato modelli preaddestrati per verificare le loro prestazioni sugli stessi dati di test utilizzati per i nostri modelli. Per la ricerca di un traduttore automatico preaddestrato, abbiamo utilizzato la piattaforma

Hugging Face riscontrando la disponibilità di vari modelli. Di quest'ultimi abbiamo provato ad utilizzare i seguenti:

- facebook/mbart-large-50-many-to-many-mmt è un modello multilingua capace di effettuare traduzioni tra qualsiasi coppia delle 50 lingue su cui è stato addestrato. Abbiamo provato ad utilizzare questo modello indicando il linguaggio target di traduzione ma nonostante ciò esso forniva come output la stessa frase di input;
- facebook/nllb-moe-54b è un modello multilingua sviluppato da Facebook e richiede uno spazio di archiviazione di 350 GB. Il suo download ha richiesto 11 ore e 43 minuti e provandolo in fase di test, esso non produceva nessun output;
- google/madlad400-3b-mt è un modello multilingua basato sull'architettura T5 ed è stato addestrato su 450 linguaggi. Data la sua complessità e dimensione, richiedeva troppo tempo per l'elaborazione di una singola frase. Di conseguenza non è stato possibile tracciare le sue metriche in fase di test a causa dell'eccessivo tempo richiesto.

A valle di ciò, abbiamo deciso di applicare lo stesso approccio utilizzato per i nostri language translator anche per i modelli preaddestrati, ovvero abbiamo utilizzato *langid* per classificare la lingua del testo in input e successivamente abbiamo utilizzato un modello preaddestrato per ogni task di traduzione. I modelli selezionati sono riportati nella Tabella 2.

Table 2: Modelli preaddestrati utilizzati

Task di traduzione	Modello utilizzato	Dataset di addestramento
fr-en	Helsinki-NLP/opus-mt-fr-en	opus
ru-en	Helsinki-NLP/opus-mt-ru-en	opus
it-en	Helsinki-NLP/opus-mt-it-en	opus
pl-en	Helsinki-NLP/opus-mt-pl-en	opus
fi-en	Helsinki-NLP/opus-tatoeba-fi-en	opus
nl-en	Helsinki-NLP/opus-mt-nl-en	opus
pt-en	unicamp-dl/translation-pt-en-t5	6 dataset citati nel paper
de-en	Helsinki-NLP/opus-mt-de-en	opus

3.6 Struttura del progetto

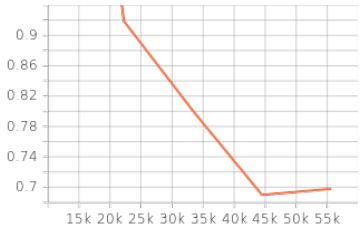
Il progetto è così strutturato:

- **.venv** che rappresenta il virtual environment di Python con le opportune librerie installate per permettere l'esecuzione del progetto;
- **opus_books_weights** contiene una sottocartella per ogni modello addestrato, all'interno della quale sono contenute le configurazioni del modello ad ogni epoca;

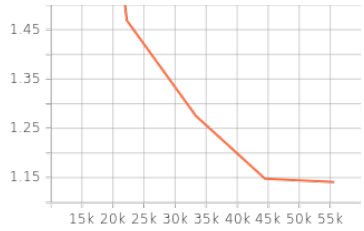
- **pretrained_models_metrics** contiene gli esperimenti TensorBoard per i modelli preaddestrati;
- **runs** suddivisa in varie sottocartelle, una per ogni modello addestrato, le quali contengono gli esperimenti TensorBoard per il tracking delle metriche durante la fase i train e test;
- **test_data** contiene i file json dei dati utilizzati per il testing dei modelli;
- **tokenikers** contiene una sottocartella per ogni modello implementato in cui sono salvati i tokeniker necessari per il determinato modello;
- **train_data** contiene i file json dei dati utilizzati per l'addestramento dei modelli;
- **valid_data** contiene i file json dei dati utilizzati per la fase di validation dei modelli;
- **config.py** utile per la corretta configurazione dei modelli;
- **dataset.py** gestisce il download dei dati da Hugging Face, il loro splitting in train, test e valid e il loro salvataggio nei corrispettivi file json. Inoltre, si occupa della creazione dei tokeniker e della tokenizzazione del testo;
- **embeddings.py** definisce l'embedding e il positional encoding dell'input;
- **model_blocks.py** definisce i blocchi del modello transformer;
- **model.py** definisce il modello transformer;
- **train.py** si occupa del training, validation e testing dei modelli calcolando le opportune metriche nelle diverse fasi;
- **__main__.py** che implementa la live demo del nostro language translator;
- **__main_pretrained_model__.py** che implementa la live demo dei modelli preaddestrati;
- **test_pretrained_model** permette di testare i modelli preaddestrati sugli stessi dati di test utilizzati per i nostri modelli, calcolando le opportune metriche di valutazione;
- **requirements.txt** contenente tutte le librerie necessarie per creare lo stesso virtual environment richiesto per il progetto.

4 Risultati

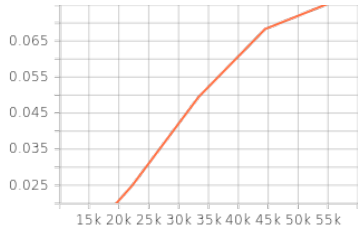
Per valutare i vari language translator, abbiamo calcolato le seguenti metriche fornite dalle librerie *torchmetrics* e *nltk*:



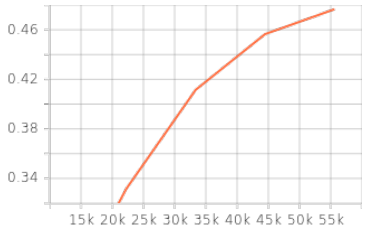
(a) CER in training per fr_en



(b) WER in training per fr_en



(c) BLEU in training per fr_en



(d) Meteor in training per fr_en

Figure 3: Metriche in train per fr_en con solo 5 epoche

- **Char Error Rate** (CER) che calcola la percentuale di caratteri erroneamente predetti dal modello;
- **Word Error Rate** (WER) che calcola la percentuale di parole errate tra la frase predetta e la frase target;
- **BLEU** per valutare la qualità della traduzione fornita dal modello rispetto a traduzioni definite da persone;
- **Meteor** per valutare la qualità della traduzione fornita dal modello rispetto a traduzioni definite da persone. In particolare, riesce a gestire anche i sinonimi. Essa è basata su media armonica, precision e recall.

Durante la fase di addestramento e di test dei nostri modelli di traduzione, abbiamo tracciato queste metriche memorizzandole in esperimenti TensorBoard. Le Figure 3 - 10 mostrano l'andamento delle metriche nella fase di training mentre la Tabella 3 mostra il punteggio ottenuto dai modelli in fase di test.

Table 3: Metriche di test dei modelli addestrati

Modello	CER	WER	BLEU	METEOR
fr-en	0.6758	1.1093	0.0768	0.4779
ru-en	0.6965	1.0614	0.0331	0.3157
it-en	0.7334	1.12	0.0262	0.2984
pl-en	0.8375	1.2141	0.0038	0.1757
fi-en	0.7317	1.0708	0.0097	0.2483

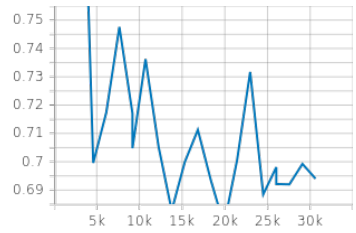
nl-en	0.7322	1.1275	0.0342	0.359
pt-en	0.6992	1.0925	0	0.2941
de-en	0.7051	1.0892	0.0403	0.3633

La Tabella 4, invece, mostra le metriche ottenute dai modelli preaddestrati sugli stessi dati di test usati per i language translator da noi definiti.

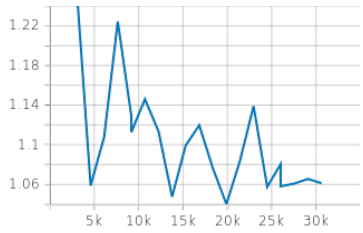
Table 4: Metriche di test dei pretrained model

Modello	CER	WER	BLEU	METEOR	Missed translation
fr-en	0.5011	0.6947	0.1869	0.526	1
ru-en	0.5226	0.7123	0.1484	0.4877	0
it-en	0.5757	0.7757	0.1267	0.4421	0
pl-en	0.7081	0.898	0.0572	0.3096	0
fi-en	0.6038	0.8084	0.1201	0.4171	0
nl-en	0.6339	0.8407	0.1334	0.4691	0
pt-en	0.7131	0.8235	0.07	0.3245	0
de-en	0.5731	0.7697	0.151	0.4987	1

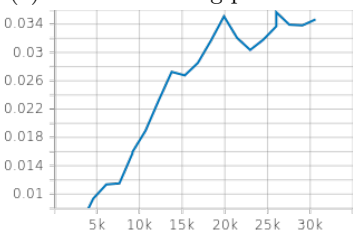
Dalle metriche ottenute dai language translator da noi definiti, riportate nella Tabella 3, possiamo affermare che i valori inerenti a BLEU e Meteor non sono ottimali. Questo è dovuto principalmente alla mancanza di traduzioni alternative per ogni corpus del dataset e alle ridotte dimensioni del corpora. Possiamo, infatti, osservare che le metriche CER e WER hanno valori abbastanza alti e quindi i modelli hanno un tasso di errore elevato.
Per quanto riguarda i modelli preaddestrati, invece, possiamo osservare che hanno ottenuto metriche migliori rispetto ai language translator addestrati. Nonostante ciò, i valori delle metriche non possono ritenersi ottimali. Infine, è da notare che alcuni modelli preaddestrati non riescono a gestire input di dimensioni maggiori a 512, infatti in tal caso veniva generato il seguente errore: *"Token indices sequence length is longer than the specified maximum sequence lenght for this model. Running this input through the model will result in indexing errors. Your input_length is bigger than 0.9 * max_length: 512."*



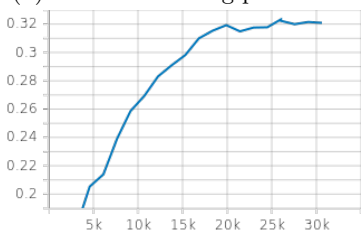
(a) CER in training per ru_en



(b) WER in training per ru_en

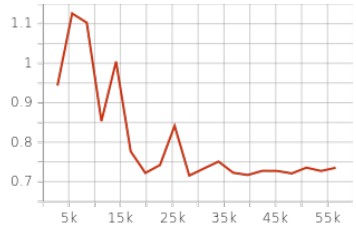


(c) BLEU in training per ru_en

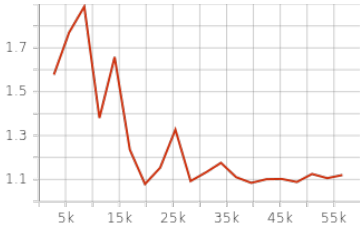


(d) Meteor in training per ru_en

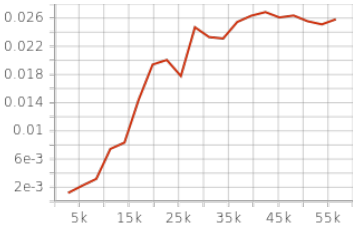
Figure 4: Metriche in train per ru_en



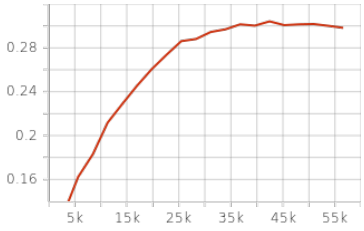
(a) CER in training per it_en



(b) WER in training per it_en

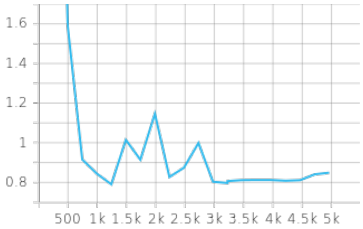


(c) BLEU in training per it_en

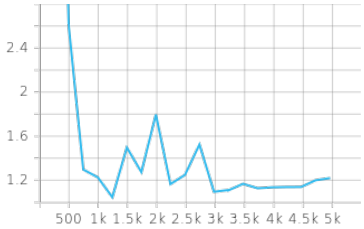


(d) Meteor in training per it_en

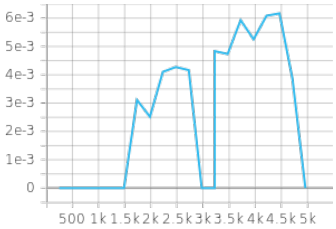
Figure 5: Metriche in train per it_en



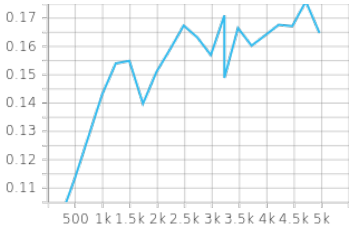
(a) CER in training per pl_en



(b) WER in training per pl_en

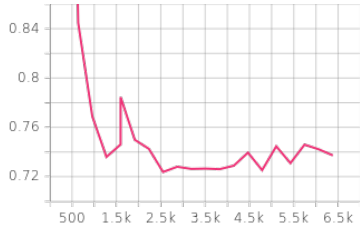


(c) BLEU in training per pl_en

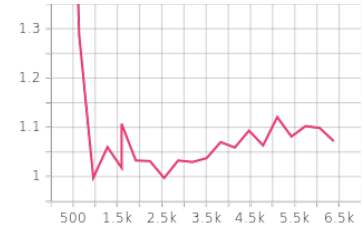


(d) Meteor in training per pl_en

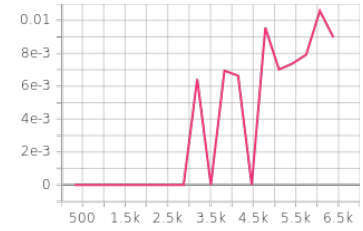
Figure 6: Metriche in train per pl_en



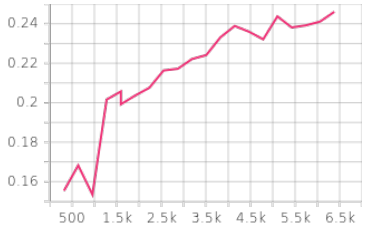
(a) CER in training per fi_en



(b) WER in training per fi_en

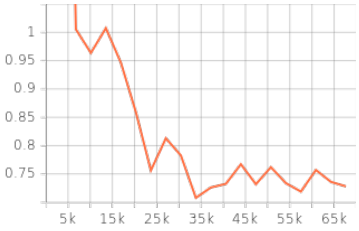


(c) BLEU in training per fi_en

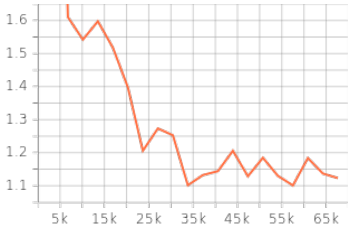


(d) Meteor in training per fi_en

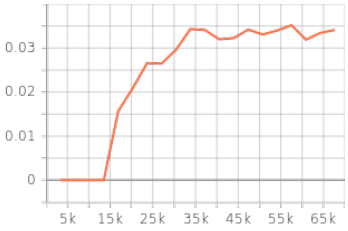
Figure 7: Metriche in train per fi_en



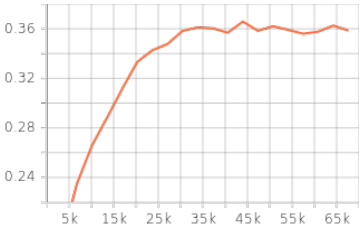
(a) CER in training per nl_en



(b) WER in training per nl_en

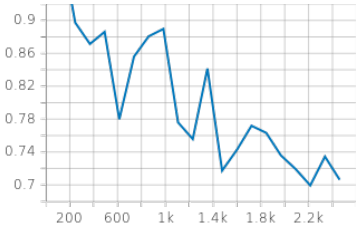


(c) BLEU in training per nl_en

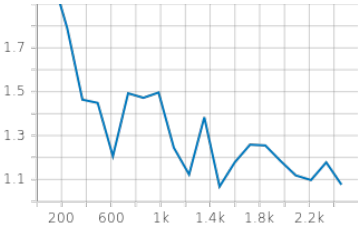


(d) Meteor in training per nl_en

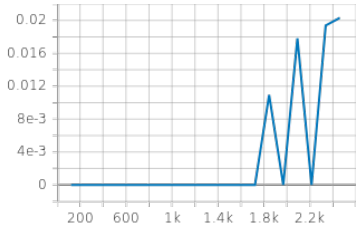
Figure 8: Metriche in train per nl_en



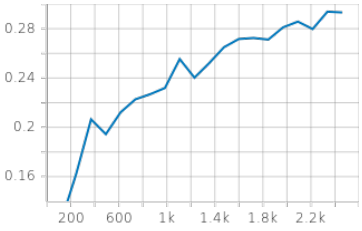
(a) CER in training per pt_en



(b) WER in training per pt_en

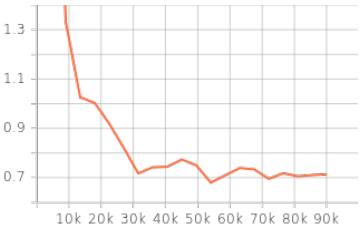


(c) BLEU in training per pt_en

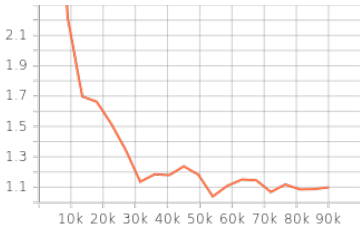


(d) Meteor in training per pt_en

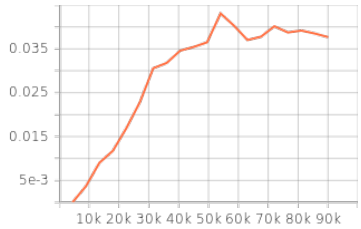
Figure 9: Metriche in train per pt_en



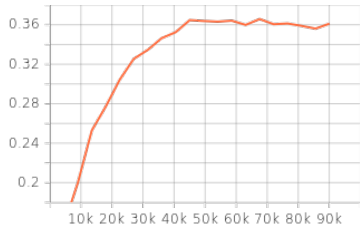
(a) CER in training per de_en



(b) WER in training per de_en



(c) BLEU in training per de_en



(d) Meteor in training per de_en

Figure 10: Metriche in train per de_en

5 Conclusioni

Sulla base dei risultati ottenuti sia dai nostri language translator che dai modelli preaddestrati, possiamo notare che i primi effettuato più errori rispetto ai secondi ma nonostante ciò i risultati ottenuti da quest'ultimi non possono ritenersi ottimali. Questo è dovuto principalmente alla dimensione del corpora (per i nostri modelli) e dalla mancanza di traduzioni alternative per ogni corpus. In conclusione, possiamo affermare che l'approccio utilizzato per lo sviluppo dei modelli può portare ad ottenere buone prestazioni, in quanto è possibile riaddestrare in futuro un singolo language translator senza la necessità di riaddestrare il tutto ma per fare ciò sono richiesti corpora più grandi con eventuali traduzioni alternative e di conseguenza una maggiore potenza di calcolo.