Relazione Settimanale - Sviluppo del Sistema di Fine-Tuning di BERT

Gruppo di Lavoro

1 aprile 2025

Indice

1 Introduzione

La presente relazione descrive le attività svolte durante la terza settimana di lavoro, incentrata sul perfezionamento della pulizia dei dati e sulla realizzazione dei primi test in ambiente Google Colab. In particolare, il focus principale è stato posto sull'implementazione iniziale del fine-tuning di BERT tramite la libreria `transformers` di Python e sull'elaborazione dei dati grezzi, al fine di predisporre un dataset bilanciato per i successivi esperimenti.

2 Pulizia e Preparazione dei Dati

Durante questa settimana abbiamo intensificato gli sforzi nella pulizia e organizzazione dei dati. Il lavoro si è articolato nelle seguenti fasi:

- **Estrazione e Aggregazione:** Dall'insieme dei file in formato `.txt` forniti dal Professor Ferretti, abbiamo estratto circa 2000 esempi etichettati come `jailbreak` e li abbiamo raccolti in un unico file. Per mantenere il bilanciamento del dataset, sono stati estratti altresì 2000 esempi etichettati come `confused` (intesi come non `jailbreak`).
- **Preliminare Validazione:** È importante notare che, in questa fase preliminare, non è stata ancora effettuata una verifica approfondita della correttezza delle etichette. Tale operazione verrà eseguita in iterazioni successive, poiché il primo approccio si concentra sul reperimento e l'aggregazione dei dati.
- **Utilizzo di Script Python:** Tutte le estrazioni e le operazioni di combinazione dei file sono state realizzate tramite script Python, garantendo così un elevato grado di automazione e riproducibilità del processo.

3 Implementazione dei Test su Google Colab

Un'altra parte fondamentale del lavoro di questa settimana ha riguardato l'implementazione dei primi test in Google Colab. Gli obiettivi di questi test sono:

- **Validazione dell'approccio:** Eseguire il fine-tuning di BERT su un dataset di dimensioni ridotte, al fine di monitorare le metriche di performance quali accuratezza, precisione e recall.
- **Analisi delle Risorse:** Identificare eventuali colli di bottiglia in termini di tempi di training e memoria GPU, predisponendo così il passaggio ad un modello più robusto da eseguire sul cluster del dipartimento.
- **Feedback Operativo:** I test hanno permesso di acquisire una prima esperienza sul workflow di training e di valutazione del modello, evidenziando il rischio di overfitting dovuto al testing con esempi già inclusi nel dataset di training.

4 Sviluppo di un Layer di Traduzione

Durante le operazioni di analisi dei dati è emersa la problematica della presenza di testi in lingue differenti dall'inglese. Nello specifico, una parte consistente del dataset conteneva risposte scritte in vietnamita, indonesiano, giapponese e turco. Per ovviare a questa problematica, abbiamo realizzato uno script Python sofisticato volto a tradurre in modo automatico i file contenenti gli esempi sia di `jailbreak` che di `no jailbreak`.

4.1 Obiettivi del Layer di Traduzione

Lo scopo del layer di traduzione è duplice:

1. **Standardizzazione dell'Input:** Garantire che tutte le risposte vengano convertite in lingua inglese, permettendo di utilizzare un unico modello (BERT) per il fine-tuning senza introdurre distorsioni semantiche.

2. **Compatibilità e Flessibilità:** Integrare strumenti di traduzione che spaziano dall'utilizzo di API esterne, come Google Traduttore, a soluzioni offline basate su librerie open source, per avere una copertura completa anche quando alcuni strumenti esterni si rifiutano di tradurre testi contenenti informazioni illegali.

4.2 Versione 1.0 del Traduttore: Dettaglio del Codice

Di seguito viene riportata la versione 1.0 dello script del traduttore, sviluppata per essere il nucleo del futuro layer di traduzione. Questo script gestisce sia il download dei pacchetti per la traduzione offline sia la logica per suddividere il testo in blocchi traducibili, interfacciandosi con i vari strumenti disponibili.

```
import os
import time
from pathlib import Path
from tqdm import tqdm
import logging

# Configura logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler("translation_log.txt"),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)

def get_translator(translator_type="google", to_lang="en"):
    """Crea e restituisce il traduttore appropriato in base al tipo specificato"""
    if translator_type == "google":
        try:
            from deep_translator import GoogleTranslator
            # Google Translator ha una funzione di auto-rilevamento della lingua
            # che funziona meglio con contenuti multilingue
            return GoogleTranslator(source='auto', target=to_lang)
```

```
        except ImportError:
            logger.error("Per usare GoogleTranslator installa: pip install deep-translator")
            return None
    elif translator_type == "offline":
        try:
            from argostranslate import package, translate
            # Verifica se i pacchetti sono già installati
            if not package.get_installed_packages():
                logger.info("Scaricamento e installazione dei pacchetti di traduzione...")
                package.update_package_index()
                available_packages = package.get_available_packages()
                installed_count = 0
                for pkg in available_packages:
                    if pkg.to_code == to_lang:
                        try:
                            pkg.install()
                            installed_count += 1
                            logger.info(f"Installato pacchetto: {pkg.from_code} -> {pkg.to_code}")
                        except Exception as e:
                            logger.error(f"Errore installazione {pkg.from_code}: {str(e)}")
                logger.info(f"Installati {installed_count} pacchetti di traduzione verso {to_lang}")
                if installed_count == 0:
                    logger.error(f"Nessun pacchetto trovato con target {to_lang}")
                    return None
            installed_languages = translate.get_installed_languages()
            target_lang = next((lang for lang in installed_languages if lang.code == to_lang), None)
            if not target_lang:
                logger.error(f"Lingua target {to_lang} non disponibile")
                return None
            class MultilingualArgosTranslator:
                def __init__(self, installed_languages, target_lang_code):
                    self.target_lang_code = target_lang_code
                    self.installed_languages = installed_languages
                    self.models = {}
                    for lang in installed_languages:
                        if lang.code != target_lang_code:
                            translation = lang.get_translation(target_lang_code)
                            if translation:
                                self.models[lang.code] = translation
                    if not self.models:
                        logger.error("Nessun modello di traduzione disponibile")
                def translate(self, text):
                    import re
                    if self._is_probably_english(text):
                        return text
                    detected_lang = self._detect_language(text)
                    if detected_lang in self.models:
```

```
                        return self.models[detected_lang].translate(text
)
                    else:
                        best_translation = text
                        for lang_code, model in self.models.items():
                            try:
                                translation = model.translate(text)
                                if self._quality_score(translation) >
self._quality_score(best_translation):
                                    best_translation = translation
                            except:
                                continue
                        return best_translation
                def _is_probably_english(self, text):
                    common_english_words = {"the", "and", "is", "in", "
to", "of", "that", "for", "it", "with"}
                    words = text.lower().split()
                    if not words:
                        return True
                    english_count = sum(1 for word in words if word in
common_english_words)
                    return english_count / len(words) > 0.2
                def _detect_language(self, text):
                    try:
                        import langdetect
                        return langdetect.detect(text)
                    except:
                        if any(ord(c) > 1000 for c in text):
                            for candidate in ["ar", "ru", "zh", "ja", "
ko"]:
                                if candidate in self.models:
                                    return candidate
                        return next(iter(self.models.keys())) if self.
models else "en"
                def _quality_score(self, text):
                    latin_chars = sum(1 for c in text if 'a' <= c.lower
() <= 'z')
                    total_chars = max(1, len(text.strip()))
                    return latin_chars / total_chars
            return MultilingualArgosTranslator(installed_languages,
to_lang)
        except ImportError:
            logger.error("Per usare il traduttore offline installa: pip
install argostranslate langdetect")
            return None
    elif translator_type == "combo":
        google_translator = get_translator("google", to_lang)
        if google_translator:
            return google_translator
        logger.warning("Google Translator non disponibile, uso
traduttore offline")
        return get_translator("offline", to_lang)
    else:
        logger.error(f"Tipo di traduttore non supportato: {
translator_type}")
        return None

```

```
def traduci_file_multilingue(file_input, file_output, etichetta_inizio, etichetta_fine,
                             translator_type="google", to_lang="en",
                             chunk_size=4500, delay=2):
    """
    Traduce un file di grandi dimensioni con supporto per testi multilingue.

    Parametri:
    - file_input: percorso del file da tradurre
    - file_output: percorso del file di output
    - etichetta_inizio: tag che segna l'inizio di un blocco da tradurre
    - etichetta_fine: tag che segna la fine di un blocco da tradurre
    - translator_type: "google", "offline" o "combo"
    - to_lang: lingua di destinazione (default "en" per inglese)
    - chunk_size: dimensione massima in caratteri per ogni richiesta di traduzione
    - delay: ritardo in secondi tra le traduzioni
    """
    backup_file = f"{file_output}.progress"
    temp_output = f"{file_output}.temp"
    translator = get_translator(translator_type, to_lang)
    if not translator:
        logger.error("Impossibile inizializzare il traduttore")
        return
    blocchi_tradotti = []
    ultimo_blocco_tradotto = -1
    if os.path.exists(backup_file):
        try:
            with open(backup_file, 'r', encoding='utf-8') as f:
                ultimo_blocco_tradotto = int(f.read().strip())
            if os.path.exists(temp_output):
                with open(temp_output, 'r', encoding='utf-8') as f:
                    blocchi_tradotti = f.read().split("\n\n--- NUOVO BLOCCO ---\n\n")
                    blocchi_tradotti = [b for b in blocchi_tradotti if b.strip()]
            logger.info(f"Ripresa dalla traduzione: trovati {len(blocchi_tradotti)} blocchi già tradotti")
        except Exception as e:
            logger.error(f"Errore durante il caricamento del progresso: {e}")
            ultimo_blocco_tradotto = -1
            blocchi_tradotti = []
    try:
        with open(file_input, 'r', encoding='utf-8') as fin:
            contenuto = fin.read()
    except UnicodeDecodeError:
        encodings = ['latin-1', 'iso-8859-1', 'cp1252']
        for enc in encodings:
            try:
                with open(file_input, 'r', encoding=enc) as fin:
                    contenuto = fin.read()
                logger.info(f"File aperto con encoding: {enc}")
                break
            except:
                continue
        else:
```

```
            logger.error("Impossibile aprire il file con gli encoding supportati")
            return
        indice_inizio = 0
        tutti_blocchi = []
        while True:
            inizio = contenuto.find(etichetta_inizio, indice_inizio)
            if inizio == -1:
                break
            fine = contenuto.find(etichetta_fine, inizio)
            if fine == -1:
                break
            fine_completa = fine + len(etichetta_fine)
            blocco = contenuto[inizio:fine_completa]
            tutti_blocchi.append(blocco)
            indice_inizio = fine_completa
        logger.info(f"Trovati {len(tutti_blocchi)} blocchi totali nel file")
        for i, blocco in enumerate(tqdm(tutti_blocchi[ultimo_blocco_tradotto+1:],
                                        desc="Traduzione blocchi")):
            indice_blocco = i + ultimo_blocco_tradotto + 1
            try:
                inizio_tag = blocco.find(etichetta_inizio)
                fine_tag = blocco.rfind(etichetta_fine)
                if inizio_tag != -1 and fine_tag != -1:
                    inizio_contenuto = inizio_tag + len(etichetta_inizio)
                    testo_da_tradurre = blocco[inizio_contenuto:fine_tag].strip()
                    if not testo_da_tradurre:
                        blocco_tradotto = blocco
                        blocchi_tradotti.append(blocco_tradotto)
                        continue
                    if len(testo_da_tradurre) > chunk_size:
                        linee = testo_da_tradurre.split('\n')
                        chunks = []
                        chunk_corrente = []
                        lunghezza_corrente = 0
                        for linea in linee:
                            if lunghezza_corrente + len(linea) > chunk_size and chunk_corrente:
                                chunks.append('\n'.join(chunk_corrente))
                                chunk_corrente = []
                                lunghezza_corrente = 0
                            chunk_corrente.append(linea)
                            lunghezza_corrente += len(linea) + 1
                        if chunk_corrente:
                            chunks.append('\n'.join(chunk_corrente))
                        traduzioni = []
                        for j, chunk in enumerate(chunks):
                            try:
                                if is_probably_english(chunk):
                                    logger.info(f"  Chunk {j+1}/{len(chunks)} già in inglese, salto traduzione")
                                    traduzioni.append(chunk)
                                else:
                                    traduzione_chunk = translator.translate(chunk)
                                    traduzioni.append(traduzione_chunk)
```

```
                                logger.info(f"  Blocco {indice_blocco+1}/{len(tutti_blocchi)}: Chunk {j+1}/{len(chunks)} tradotto")
                            time.sleep(delay)
                        except Exception as e:
                            logger.error(f"Errore durante la traduzione del chunk {j+1}: {e}")
                            traduzioni.append(chunk)
                    testo_tradotto = '\n'.join(traduzioni)
                else:
                    if is_probably_english(testo_da_tradurre):
                        logger.info(f"  Blocco {indice_blocco+1}/{len(tutti_blocchi)} già in inglese, salto traduzione")
                        testo_tradotto = testo_da_tradurre
                    else:
                        testo_tradotto = translator.translate(testo_da_tradurre)
                blocco_tradotto = f"{etichetta_inizio}\n{testo_tradotto}\n{etichetta_fine}"
                blocchi_tradotti.append(blocco_tradotto)
                with open(backup_file, 'w', encoding='utf-8') as f:
                    f.write(str(indice_blocco))
                with open(temp_output, 'w', encoding='utf-8') as f:
                    f.write("\n\n--- NUOVO BLOCCO ---\n\n".join(blocchi_tradotti))
                time.sleep(delay)
            else:
                logger.error(f"Errore: tag non trovati nel blocco {indice_blocco+1}")
                blocchi_tradotti.append(blocco)
        except Exception as e:
            logger.error(f"Errore durante la traduzione del blocco {indice_blocco+1}: {e}")
            blocchi_tradotti.append(blocco)
            with open(backup_file, 'w', encoding='utf-8') as f:
                f.write(str(indice_blocco))
            with open(temp_output, 'w', encoding='utf-8') as f:
                f.write("\n\n--- NUOVO BLOCCO ---\n\n".join(blocchi_tradotti))
    with open(file_output, 'w', encoding='utf-8') as fout:
        fout.write("\n\n--- NUOVO BLOCCO ---\n\n".join(blocchi_tradotti))
    if len(blocchi_tradotti) == len(tutti_blocchi):
        try:
            if os.path.exists(backup_file):
                os.remove(backup_file)
            if os.path.exists(temp_output):
                os.remove(temp_output)
        except Exception as e:
            logger.warning(f"Avviso: impossibile rimuovere i file temporanei: {e}")
    logger.info(f"{len(blocchi_tradotti)} blocchi tradotti e salvati in '{file_output}'")

def is_probably_english(text):
    """Determina se il testo è probabilmente già in inglese"""
    common_english_words = {"the", "and", "is", "in", "to", "of", "that", "for", "it", "with",
```

```
                                "this", "on", "are", "as", "was", "by", "be",
    "have", "you", "not"}
    words = text.lower().split()
    if not words:
        return True
    if len(words) < 5:
        return False
    english_count = sum(1 for word in words if word in
    common_english_words)
    return english_count / len(words) > 0.15

# Esempio di utilizzo
if __name__ == "__main__":
    traduci_file_multilingue(
        '2490_Confused.txt',
        '2490_Confused_Tradotto.txt',
        '###CONFUSED###',
        '###ENDCONFUSED###',
        translator_type="google",
        to_lang="en",
        chunk_size=4500,
        delay=2
    )
```

Listing 1: Versione 1.0 del traduttore

4.3 Evoluzione del Workflow e Integrazione

Una volta verificato il corretto funzionamento dello script di traduzione, abbiamo testato la sua efficacia passando i file di esempio per `jailbreak` e `no jailbreak` per tre volte consecutive. Quest'approccio ha portato a due ulteriori sviluppi:

- **Adozione del Formato JSON:** Considerata la necessità di gestire file di grandi dimensioni e di integrare agevolmente i dati con i nostri script Python, abbiamo convertito i file iniziali in formato JSON. Tale formato si rivela più adatto rispetto al CSV perché permette una migliore strutturazione e manipolazione dei dati.
- **Preparazione del Dataset Finale:** Successivamente, abbiamo sviluppato un ulteriore script per combinare i due file JSON, alternando gli esempi etichettati (1 per esempi `jailbreak` e 0 per esempi `no-jailbreak`). Inoltre, è stato prelevato un sottoinsieme di 300 esempi per classe da utilizzare come dataset di test, sebbene tale operazione si sia poi rivelata non ottimale in quanto comportava l'inclusione di esempi già presenti nel dataset di training.

5 Test Preliminari e Analisi dei Risultati

Una volta completata la fase di preparazione del dataset, sono stati eseguiti i primi test di fine-tuning di BERT su Google Colab, configurati per l'uso di GPU. Le metriche ottenute, seppur solo indicative, hanno mostrato risultati sorprendentemente ottimi. Tuttavia, si sospetta che il modello stia andando in overfitting, poiché i test sono stati effettuati su esempi già visti durante la fase di training. A seguito di questi test preliminari, abbiamo avuto modo di interfacciarci con il professore per ottenere un feedback mirato e definire i prossimi step della ricerca:

- La realizzazione di un dataset di training più pulito ed affidabile.
- La definizione di un dataset di test separato per evitare sovrapposizioni e ridurre il rischio di overfitting.

6 Conclusioni e Prospettive Future

Riassumendo, durante questa settimana sono stati compiuti importanti passi avanti, che includono:

- L'approfondimento del processo di pulizia dei dati e l'aggregazione dei file grezzi in dataset bilanciati.
- L'implementazione dei primi test in ambiente Google Colab per il fine-tuning iniziale di BERT, con una valutazione preliminare delle performance del modello.
- Lo sviluppo e il collaudo di un complesso script Python destinato alla traduzione automatica dei testi, che costituirà il nucleo del futuro layer di traduzione.
- La conversione in formato JSON dei file di dataset e la successiva combinazione per la creazione di un dataset finale alternato, sebbene con alcuni limiti riscontrati nella fase di test.

In prospettiva, nei prossimi step ci concentreremo su:

- Il perfezionamento del processo di scrematura del dataset per eliminare dati ridondanti e non validi.
- L'ottimizzazione del modello di fine-tuning per evitare l'overfitting, tramite una revisione attenta dei dataset di training e test.
- L'implementazione di ulteriori funzionalità nel layer di traduzione per garantire una gestione ancora più robusta delle diverse lingue.

Note Finali

Il lavoro di questa settimana ha fornito una solida base per le fasi successive del progetto, evidenziando sia i punti di forza del nostro approccio, sia le criticità da affrontare nelle prossime iterazioni. L'integrazione dei vari script Python e l'adozione di un formato dati più strutturato rappresentano elementi fondamentali per garantire la scalabilità e la riproducibilità del sistema.