Università Ca' Foscari Venezia

Corso di Laurea in Informatica

Tesi di Laurea

ADVERSARIAL NATURAL LANGUAGE PROCESSING

Relatore: Prof. Claudio Lucchese

Laureando: Riccardo Spolaor 864877

Anno Accademico 2019/2020

Abstract

La robustezza delle Deep Neural Networks (DNNs), come per ogni altro modello di Machine Learning, può essere compromessa da ”attacchi” che spingono il modello ad una errata classificazione di istanze che sono state soggette a piccole perturbazioni da parte dell'attaccante. Queste istanze prendono il nome di Adversarial Examples. Nell'area del Natural Language Processing (NLP), a differenza di altri campi, un'istanza perturbata anche in maniera minimale può risultare evidente all'occhio umano: modificare una singola parola in un testo può portare alla variazione o alla perdita del suo significato originale. La presente tesi di laurea propone l'utilizzo di un algoritmo genetico di ottimizzazione che permette la generazione di Adversarial Examples che mantengono un significato sintattico e semantico simile al testo originale, ma che riescono ad ingannare un modello di Sentiment Analisys di documenti con architettura DNN.

Indice

Introduzione

Con lo sviluppo di strumenti di calcolo in grado di ottenere capacità di elaborazione elevate le *Deep Neural Networks (DNNs)* sono al giorno d'oggi applicate in vari campi dell'*Intelligenza Artificiale (IA)* quali: *Computer Vision* [1], *Game Theory* [2] o *Natural Language Processing* [3]. È stato osservato come questi modelli possano essere facilmente ingannati da utenti malintenzionati utilizzando una serie di tecniche con lo scopo di generare risultati falsati. Tali metodi di attacco unitamente a quelli finalizzati alla difesa dei modelli prendono il nome di *Adversarial Machine Learning (Adversarial ML)* [4]. Gli *Adversarial Examples* sono un tipo di attacco di Adversarial ML che utilizza istanze generate appositamente per confondere un modello di previsione nella loro classificazione. Queste vengono costruite applicando una perturbazione alla struttura dei dati di istanze precedentemente classificate correttamente dal modello. La perturbazione, affinché gli Adversarial Examples siano efficaci, deve risultare impercettibile all'occhio umano. Nel campo del *Natural Language Processing (NLP)* cambiare anche una singola parola di una frase può portare all'alterazione o perdita di significato dell'intero testo. Nell'articolo *Generating Natural Language Adversarial Examples* pubblicato dalla *Association for Computational Linguistics* [5] è stata dimostrata la possibilità di generare Adversarial Examples nel contesto dell'NLP che mantengano intatto il significato del testo originale e approssimativamente anche la sua correttezza sintattica.

Lo scopo della tesi è quello di ripercorrere l'esperimento svolto all'interno dell'articolo sopra citato che prevede l'attacco tramite Adversarial Examples di un modello allenato per svolgere *Sentiment Analysis* delle recensioni dei film del dataset *IMDB Large Movie Review Dataset* [6]. Nel ripercorrere l'esperimento, ho ritenuto di applicare alcune modifiche a dei passi di implementazione dell'algoritmo, per affinare i termini utilizzati nell'attacco e ridurre i tempi di computazione.

La tesi è stata strutturata come segue:

- **Capitolo 1. Natural Language Processing e Deep Neural Networks:** introduzione all'NLP e alle DNN illustrando gli elementi che le costituiscono, quali i layer e i neuroni. Viene inoltre descritto dettagliatamente il funzionamento dei layer interni al modello utilizzato nell'esperimento e le prassi per allenare una DNN.

- **Capitolo 2. Generazione del modello di Sentiment Analysis di documenti:** descrizione della DNN per il Sentiment Analysis delle recensioni di film da sottoporre ad attacchi avversari e del processo di allenamento ad essa applicato. Si illustra successivamente il lavoro di pre-processing delle recensioni finalizzato al corretto allenamento del modello di previsione, viene inoltre definita la struttura del sistema Black-Box utilizzato per l'attacco. Si presentano infine gli strumenti software impiegati.

- **Capitolo 3. Attacco Avversario:**
  introduzione alla tecnica degli Adversarial Examples e descrizione dei principali metodi per costruirli. Si illustra nel dettaglio il funzionamento dell'algoritmo genetico utilizzato per attaccare il modello di Sentiment Analysis e delle strutture dati che utilizza. Vengono inoltre elencate e motivate le modifiche applicate all'algoritmo dell'esperimento originale. Sono infine descritti gli strumenti software adottati per costruire l'architettura dell'algoritmo.

- **Capitolo 4. Valutazione Sperimentale:**
  In quest'ultimo capitolo sono visualizzati i risultati dell'elaborazione dell'algoritmo di attacco in termini di tempistica ed efficacia al variare di alcuni dei suoi iperparametri e sono argomentate le conclusioni di merito sia dal punto di vista tecnico che qualitativo.

Capitolo 1

Natural Language Processing e Deep Neural Networks

1.1 Natural Language Processing

Il *Natural Language Processing (NLP)* è un campo di applicazione dell'IA, che include nozioni sia di Computer Science che di Linguistica. Attualmente presenta una vasta diffusione pratica, anche in strumenti di utilizzo quotidiano tra i quali: filtri per spam, traduttori di documenti o assistenti digitali. Il suo scopo è quello di permettere ad un calcolatore di gestire in modo automatico delle informazioni espresse in forma scritta o verbale in una lingua naturale come può essere l'Inglese o l'Italiano. Questo campo si suddivide a sua volta in due macroaree: *Natural Language Understanding (NLU)* e *Natural Language Generation (NLG)*.

- L'**NLU** è una tecnologia che prevede la capacità da parte di un elaboratore di ricavare informazioni da un testo scritto. La sua applicazione può risultare in task complessi quali la comprensione di un documento oppure più semplici che non richiedono una conoscenza approfondita del significato di un testo come la Document Classification, che consiste nell'assegnazione di un documento ad una o più categorie di appartenenza [7]. Lo *Speech Recognition* è un subfield dell'NLU che ricava informazioni dal linguaggio orale. L'NLU è stato impiegato nell'esperimento descritto nella tesi sotto forma di *Sentiment Analysis*, *Document Classification*, *Tokenization* e *Named-Entity Recognition*. Lo scopo del Sentiment Analysis è quello di identificare opinioni favorevoli o

meno all'interno di un testo ed è stato sfruttato dal modello di previsione assieme alla Document Classification per classificare le varie recensioni dei film come positive o negative. La Tokenization gestisce un documento complesso suddividendolo in simboli o parole significative - denominati *token* - ed è stata utilizzata sia per pre-processare le recensioni prima di allenare il modello di analisi, che per ricavarne una previsione, inoltre l'algoritmo di attacco la sfrutta per organizzare in modo strumentale le recensioni. Anche la Named-Entity Recognition, che ha il compito di identificare e classificare elementi di un testo in varie categorie, come per esempio nomi propri di persona o entità geopolitiche, è stata impiegata nell'algoritmo di attacco per contribuire a garantire la correttezza semantica del testo.

- L'**NLG** è il processo automatizzato di produrre frasi di senso compiuto in una lingua naturale da parte di un calcolatore. Un tipico esempio di applicazione è quello della generazione di conversazioni con un utente da parte di un chatbot. Nell'esperimento l'NLG è stata utilizzata dall'algoritmo di attacco per produrre gli *Adversarial Examples*.

1.2 Deep Neural Networks

Per eseguire procedure di NLP, come per ogni altro campo dell'IA, è necessario l'utilizzo di strumenti di elaborazione complessi tra cui: *Artificial Neural Networks (ANNs)*; *Random Forests*; *Decision Trees*; ecc. Nell'esperimento descritto nella tesi sono state utilizzate delle sottocategorie delle ANNs, che sono modelli di calcolo ispirati al sistema nervoso animale, la cui architettura è organizzata come una serie estesa di layer di unità computazionali, denominate neuroni, in grado di compiere calcoli paralleli complessi. Ognuna di esse è collegata ai neuroni di un layer successivo ai quali trasmette i risultati della propria funzione di attivazione come input. Il layer iniziale presenta al suo interno i dati forniti in ingresso, quello finale il risultato di computazione del modello e quelli intermedi diverse trasformazioni dei dati originali. Una ANN si definisce Profonda o *Deep Neural Network (DNN)* quando contiene più di un singolo layer intermedio.

Esistono diversi tipi di layer che concorrono a costituire l'architettura di un

modello. Alcuni devono occupare posizioni prestabilite nella struttura o devono essere preceduti o seguiti da specifici altri layer per il loro corretto funzionamento. Una ANN che contiene al proprio interno alcuni tipi di layer può essere denominata in base al loro nome. In questa sezione sono descritte le tipologie di layer utilizzati dal modello di Sentiment Analysis di documenti adottato.

- **Embedding Layer:** è utilizzato per gestire dati di matrice testuale e occupa necessariamente la prima posizione tra i layer intermedi del modello. Accetta in input una serie di documenti testuali Integer Encoded, dove ogni parola o simbolo, definiti token, sono rappresentati da un unico numero intero. L'Embedding Layer prevede al suo interno uno spazio vettoriale in cui ogni token codificato è rappresentato da una successione di parametri definita *Word Embedding Vector*. Questi parametri sono inizialmente casuali, fatto salvo non siano stati specificati in precedenza, e ad iterazioni successive formeranno una rappresentazione vettoriale delle informazioni semantiche e sintattiche dei vari token sempre più accurata.

- **Dense Layer:** è uno dei layer più frequentemente utilizzati nelle ANN e applica ad un tensore di dati in input una trasformazione che consiste nel prodotto scalare tra esso ed una matrice di parametri attribuitagli, seguito da un aggiunta di bias e l'applicazione di una funzione d'attivazione. In output è restituito un tensore di una certa dimensione specificata. Se posizionato come ultimo layer restituisce la previsione del modello rappresentata in base al tipo di funzione di attivazione. Ad esempio la *Sigmoid Activation Function* è usata per i problemi di classificazione binaria e restituisce un valore scalare tra 0 e 1.

- **Dropout Layer:** utilizza la *Dropout Regularization* all'output del layer precedente. Questa tecnica prevede di settare a 0 in modo casuale alcuni elementi del vettore in output secondo una frequenza specificata prevenendo così un eventuale *Overfitting*, ovvero un eccessivo adattamento rispetto ai dati con i quali viene allenato il modello.

- **Recurrent Layer:** permette alla ANN di mantenere, tramite un loop interno, lo stato delle informazioni gestite in precedenza per ogni gruppo di sequenze

di istanze elaborato. Ogni elemento di una sequenza è considerato indipendente dagli altri e viene processato unitamente allo stato memorizzato nello step precedente che lo riguarda specificatamente. Inizialmente lo stato è assente ed è visualizzato come un vettore contenente soli zeri. Il processo permette alla ANN di analizzare nuove informazioni in modo fluido mantenendo un modello interno di ciò che è stato già visualizzato e che viene aggiornato ad ogni iterazione. Il ***Long Short-Term Memory (LSTM) Layer*** (visualizzato in Figura 1.1) è una sottocategoria di Recurrent Layer che permette alla ANN di implementare lo stesso procedimento sopra descritto considerando uno storico più ampio di stati precedenti per gli elementi processati di ogni sequenza. Una ANN che utilizza uno o più Recurrent Layer all'interno della propria architettura è definita *Recurrent Neural Network (RNN)* e se il tipo dei layer utilizzati è LSTM essa può essere descritta in modo più specifico come *LSTM Neural Network (LSTM NN)*. Le ANN che non utilizzano queste tecniche sono definite invece *Feed-Forward Neural Networks*.

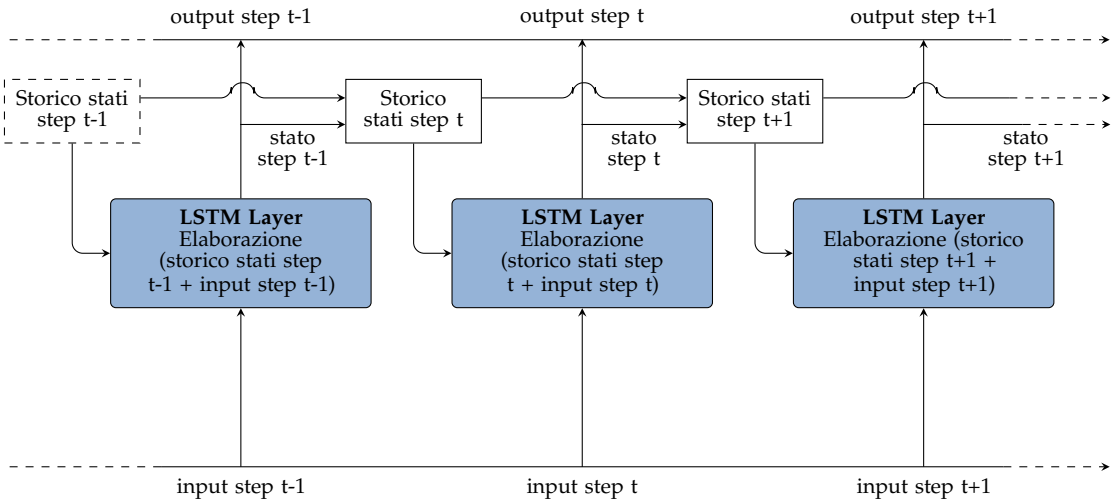


Figura 1.1: Funzionamento di un layer LSTM.

- **Bidirectional Layer:** frequentemente utilizzato nell'NLP è composto da due Recurrent Layer, ognuno dei quali processa le informazioni di una sequenza in ordine direzionale opposto per poi unificare la loro rappresentazione come illustrato in Figura 1.2. Nel contesto del linguaggio naturale una frase viene elaborata considerando sia il suo ordine cronologico che quello inverso.

L'utilizzo di questo layer non è efficiente per la gestione di serie di dati in cui l'ordine sequenziale è fondamentale. Le ANN che lo utilizzano assumono il nome di *Bidirectional Recurrent Neural Networks (BRNNs)*.

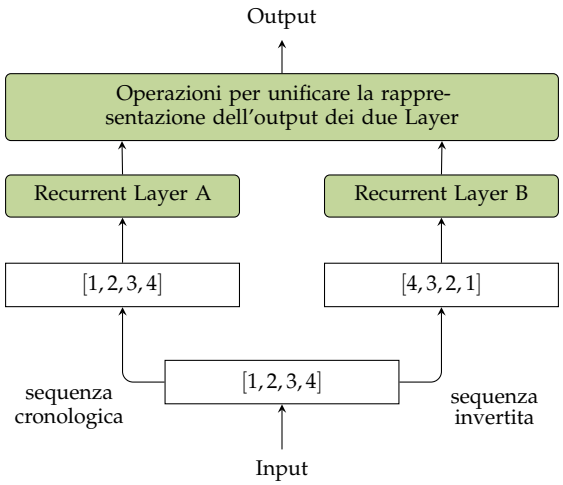


Figura 1.2: Struttura e funzionamento di un Bidirectional Layer.

- **Pooling Layer:** è utilizzato per ridurre progressivamente lo spazio di rappresentazione dell'output calcolato dai neuroni di un layer precedente. La sua funzione è di limitare il numero massimo di feature che descrivono la trasformazione dei dati in input, questo permette di rappresentare le informazioni in modo più compatto. Il *Pooling* può essere applicato a tensori cubici, matriciali o vettoriali. Se prendiamo per esempio in considerazione l'elaborazione di un contenuto testuale, la sua rappresentazione sarà visualizzata da una serie di vettori di n feature. Applicando il Pooling a finestre di 2 parametri, otterremo per ogni coppia selezionata un unico valore. Un'immagine sarà invece memorizzata come una serie di matrici con n × n feature ognuna e applicando il Pooling a finestre di 2 × 2 feature otterremo come risultato un unico valore.

  Esistono diverse tecniche di Pooling (Figura 1.3):

  - **Max Pooling:** per una finestra di feature ritorna quella con valore massimo;
  - **Average Pooling:** per una finestra di feature seleziona il loro valore medio;

- **Global Pooling:** calcola il valore medio oppure massimo tra tutte le feature. Può essere utilizzato dopo un Bidirectional o Recurrent Layer per restituire una rappresentazione vettoriale delle feature che può essere processata per la previsione oppure fornita in input ad una Feed-Forward Neural Network.

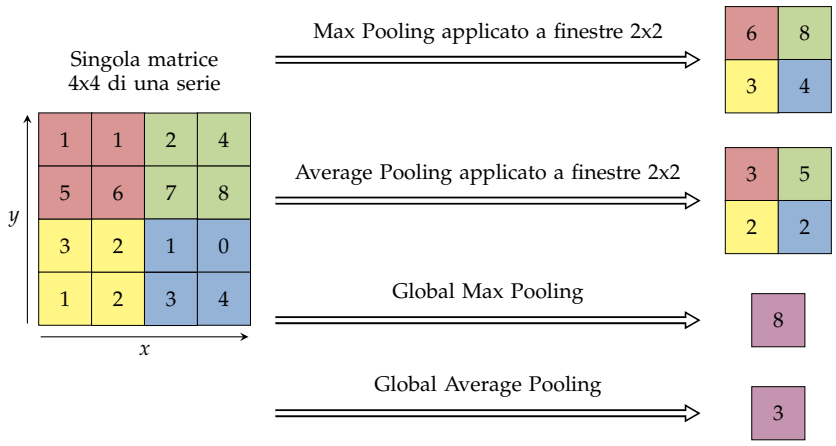


Figura 1.3: Applicazione di diversi tipi di Pooling.

1.3 Deep Learning

Il metodo definito *Machine Learning* permette l'implementazione di algoritmi in grado di svolgere compiti di apprendimento specifici e capaci di migliorare automaticamente le proprie prestazioni. Quando sono utilizzate le DNN questi processi sono descritti più specificatamente tramite la locuzione *Deep Learning*. Gli algoritmi di Machine Learning e Deep Learning si suddividono in: *Supervised Learning*; *Unsupervised Learning* e *Self-Supervised Learning*.

- Il **Supervised Learning** è il tipo di apprendimento più diffuso nel Deep Learning. Consiste nell'allenare un modello ad assegnare gli elementi processati a specifiche classi predeterminate (*label*). Per la costruzione di un modello di Supervised Learning vengono in genere utilizzati tre insiemi di elementi disgiunti denominati *Train*, *Validation* e *Test Data Sample*. Il *Train Data Sample* è utilizzato per gestire l'apprendimento da parte del modello che si concretizza attraverso la ricerca iterativa ed automatica di una specifica combinazione di parametri dei layer (*weights*) che minimizzano

la discrepanza tra i risultati di computazione degli elementi del Train Data Sample e la loro classe di effettiva appartenenza. La funzione che visualizza tale risultato, definito *loss score*, è denominata *Loss Function* e varia a seconda del genere di classificazione che il modello deve eseguire, ad esempio per la classificazione binaria viene utilizzata la *Binary Cross-Entropy*. Dopo ogni iterazione viene applicata la tecnica del *Backpropagation* che aggiorna i weights dei layer selezionati in senso inverso attraverso una *Optimizer Function* che prende in input il loss score precedentemente calcolato. Gli elementi del Train Data Sample possono essere forniti unitamente al modello in una sola iterazione oppure possono essere suddivisi casualmente in sottoinsiemi distinti, denominati *batch*, e forniti in più iterazioni. L'aggiornamento dei weight dei layer avviene dopo l'elaborazione di ogni singolo batch. Il completamento dell'elaborazione dei batch che compongono l'intero Train Data Sample viene denominato *epoch* ed il modello potrebbe essere allenato ripetutamente generandone un numero potenzialmente infinito. Produrre una quantità elevata di epoch è comunque inefficiente in quanto può provocare l'*Overfitting*, ovvero l'adattamento eccessivo del modello di previsione rispetto agli elementi del Train Data Sample. Il *Validation Data Sample* si usa per valutare l'accuratezza di previsione rispetto alle varie combinazioni dei weights dei layer ad ogni elaborazione di tutti i batch del Train Data Sample e indica la percentuale di istanze correttamente classificate rispetto ai loro label. Viene selezionata la combinazione di parametri dell'epoch che porta all'accuratezza maggiore del Validation Data Sample. Il modello finale viene testato con il *Test Data Sample*, la cui accuratezza di previsione rappresenta l'effettiva capacità previsionale del modello. Il processo di Supervised Learning di una DNN è illustrato in Figura 1.4.

- L'**Unsupervised Learning**, quando utilizzato nel Deep Learning, contempla la ricerca di informazioni rilevanti all'interno dell'insieme dei dati trasformati da una DNN senza avere a disposizione target label per le varie istanze. Questa tecnica è stata utilizzata in letteratura per eseguire task di *Clustering*, che si prefigge di realizzare il raggruppamento di istanze in diverse categorie in base alle loro caratteristiche strutturali [8]. Il principio su cui è basato il

Clustering è quello di massimizzare la similarità tra elementi all'interno di una classe e minimizzarla per elementi di classi diverse.

- Il **Self-Supervised Learning** segue gli step proposti dal Supervised Learning senza servirsi di target forniti in precedenza. I label delle varie istanze sono autogenerati da algoritmi euristici.

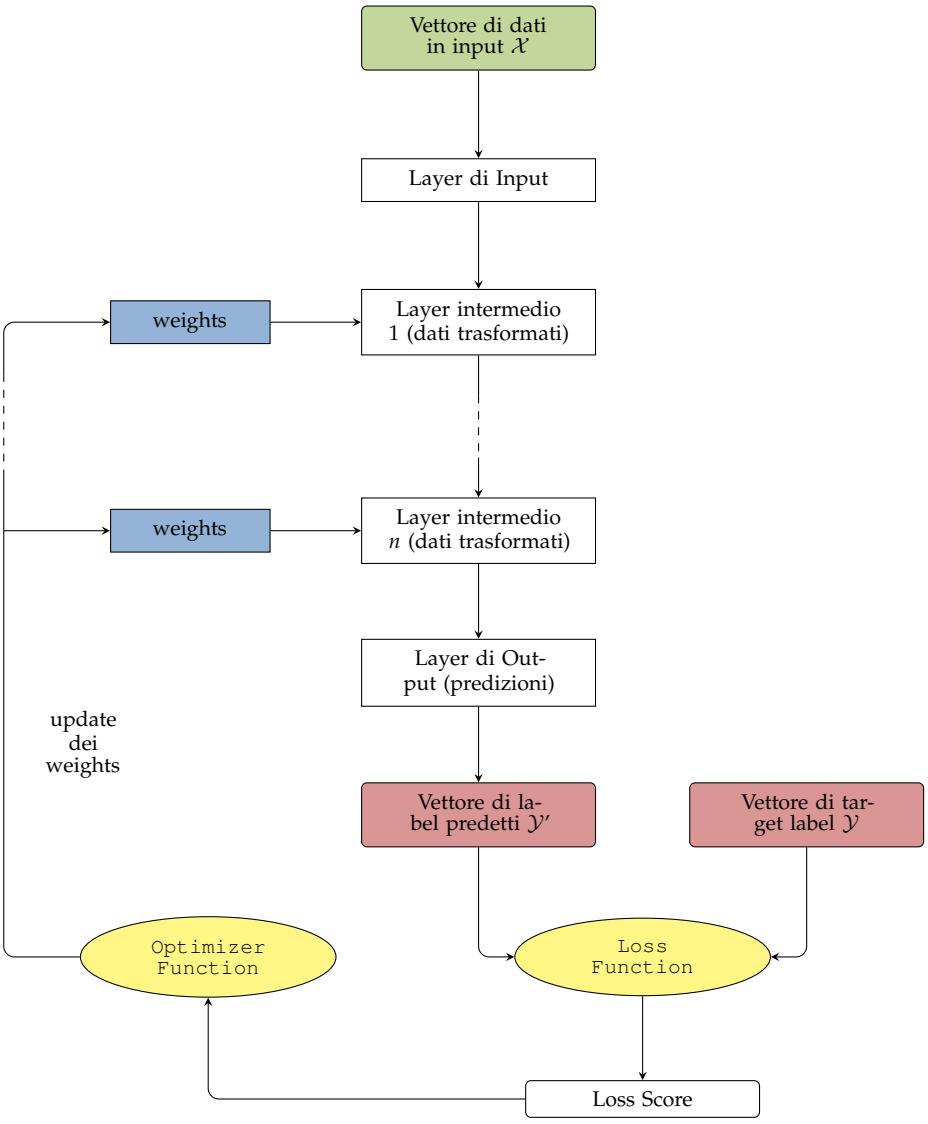


Figura 1.4: Processo di Supervised Learning di una DNN.

Capitolo 2

Generazione del modello di Sentiment Analysis di documenti

2.1 Data pre-processing

L'esperimento in [5], che la presente tesi vuole ripercorrere, consiste nell'attaccare tramite *Adversarial Examples* un modello di classificazione con architettura DNN allenato per svolgere Sentiment Analysis delle recensioni dei film del dataset *IMDB Large Movie Review Dataset* [6] categorizzandole come positive o negative. Questo dataset presenta un totale di 50.000 recensioni cinematografiche, la metà delle quali sono categorizzate come positive (label 1) e la cui altra metà sono catalogate come negative (label 0). Allenare il modello di classificazione impone l'applicazione di un processo Supervised Learning descritto al capitolo precedente. Ho quindi selezionato casualmente ⅔ delle recensioni per costituire l'insieme di elementi da utilizzare per allenare il modello. Il Train data Sample è stato generato utilizzando l'80% di esse ed il Validation Data Sample il rimanente 20%. Le recensioni residue sono state impiegate per creare il Test Data Sample, finalizzato a testare l'accuratezza di previsione del modello. Una parte di questi elementi è stata selezionata in seguito per generare gli Adversarial Examples da parte dell'algoritmo di attacco. Il Train, Validation e Test Data Sample sono stati formati da elementi scelti casualmente come specificato, ma con l'accorgimento che ognuno contenesse una quantità bilanciata di recensioni con sentiment label positivo (1) e negativo (0). Prima di procedere all'allenamento del modello di

previsione, sono state pre-processate le recensioni al fine di garantire che la loro struttura fosse adatta ai vettori pre-allenati dall'algoritmo *GloVe (Global Vectors for Word Representation)* chiamati *GloVe Embedding Vectors* [9] che sarebbero stati poi utilizzati per allenare la DNN. I vettori considerati compongono un vocabolario di 1.900.000 token privi di lettere maiuscole e costituiti da 300 dimensioni ciascuno. Il processo di adattamento delle recensioni è stato realizzato seguendo la guida illustrata nell'articolo di *Kaggle*: *How To: Preprocessing for GloVe Part1: EDA* [10]. Per gestire il lavoro di pre-processing sono stati elaborati i documenti dei Train e Validation Data Sample utilizzati per allenare il modello e rimossi gli eventuali tag html presenti all'interno delle recensioni (Tabella 2.1).

Rimozione tag html
Recensione in input: *Everything is so well done: acting, directing, visuals... If you can enjoy a story of REAL people and REAL love - this is a winner.*
Recensione in output: *Everything is so well done: acting, directing, visuals... If you can enjoy a story of REAL people and REAL love - this is a winner.*

Tabella 2.1: Esempio di rimozione tag html al testo di una recensione.

Il pre-processing delle recensioni gestito con tecniche classiche quali *Stemming*, *Lemmatization* o *rimozione delle Stopword* avrebbe contribuito alla perdita di informazioni rilevanti sulla struttura delle stesse recensioni e i token presenti non avrebbero trovato un corrispettivo riscontro nel vocabolario dei GloVe Embedding Vectors. L'obiettivo finale del pre-processing è invece quello di guidare la struttura delle recensioni ad essere costituita da token il più possibile corrispondenti a quelli dei vettori pre-allenati. Inizialmente gli Embedding ricoprivano solamente il 16,28% del vocabolario di token presenti nell'insieme di recensioni considerato e il 79,70% di tutto il testo. Queste percentuali non erano sufficienti a garantire un'adeguata corrispondenza tra la struttura delle recensioni e il vocabolario dei GloVe Embedding Vectors. Per conseguire un risultato soddisfacente sono stati seguiti i seguenti step:

1. **Lowercasing:** consiste nell'uniformare tutti i termini trasformando tutte le lettere maiuscole in minuscole. È stato applicato in quanto il vocabolario dei GloVe Embedding Vectors è composto da token contenenti esclusivamente lettere minuscole.

2. **Isolamento o eliminazione dei caratteri speciali:** oltre ad alcuni termini del linguaggio naturale (ad esempio parole Inglesi) i vettori di embedding utilizzati si riferiscono anche ad altri token, quali segni di punteggiatura o caratteri speciali che possono concorrere alla definizione del Sentiment Analysis di un documento. Diviene pertanto indispensabile considerare la presenza di questi tipi di simboli, mantenendo ed isolando dalle parole delle recensioni quelli presenti nei Word Embedding Vectors ed eliminando quelli mancanti (Esempio in Tabella 2.2).

Lowercasing e isolamento di caratteri speciali
Recensione in input: *Everything is so well done: acting, directing, visuals... If you can enjoy a story of REAL people and REAL love - this is a winner.*
Recensione in output: *everything is so well done : acting , directing , visuals . . . if you can enjoy a story of real people and real love - this is a winner .*

Tabella 2.2: Esempio di lowercasing seguito da isolamento di caratteri speciali.

3. **Isolamento delle contrazioni lessicali:** successivamente si rendeva necessario isolare le contrazioni grammaticali delle parole, in quanto non presenti all'interno dei vettori pre-allenati, ad esempio il token "she'll" viene diviso nei due token "she" e "'ll" tramite l'utilizzo di un tokenizer (Esempio in Tabella 2.3).

4. **Normalizzazione di termini non conformi:** è stata riscontrata all'interno delle recensioni la presenza di diversi token che non trovavano un corrispettivo tra quelli dei vettori pre-allenati in quanto presentavano come loro primo carattere il simbolo ′ che è stato eliminato manualmente.

Isolamento delle contrazioni lessicali
Recensione in input: *is it a poorly acted , cliche - ridden pile of trash ? of course . anyone who doesn't realize that when they pick up the box in the video store probably doesn't have any right judging movies in the first place .*
Recensione in output: *is it a poorly acted , cliche - ridden pile of trash ? of course . anyone who does n't realize that when they pick up the box in the video store probably does n't have any right judging movies in the first place .*

Tabella 2.3: Esempio di isolamento delle contrazioni lessicali di una recensione alla quale sono stati applicati gli step di preprocessing precedenti.

Il risultato di tale intervento è stato positivo evidenziando una copertura da parte degli Embedding dell'88,64% dei token delle recensioni presi singolarmente e del 99,86% del loro contenuto testuale considerato nel suo complesso.

2.2 Modello di Previsione

La DNN di classificazione adottata sfrutta inizialmente le funzionalità di una Bidirectional Long Short-Term Memory Neural Network mantenendo lo storico delle informazioni elaborate all'interno dei propri layer. Successivamente la DNN prosegue l'elaborazione come una Feed-Forward Neural Network evitando connessioni cicliche tra i layer allo scopo di provare ad affinare ulteriormente l'accuratezza di previsione senza gravare troppo sui tempi di calcolo (Figura 2.1). Le recensioni da elaborare non sono fornite al modello nella loro forma originale, ma sono prima pre-processate come descritto nella sezione precedente e successivamente opportunamente trasformate in vettori numerici di dimensione uniforme. Il motivo di queste operazioni iniziali è dovuto al fatto che il layer d'ingresso della DNN accetta esclusivamente vettori composti da numeri e non caratteri di testo.

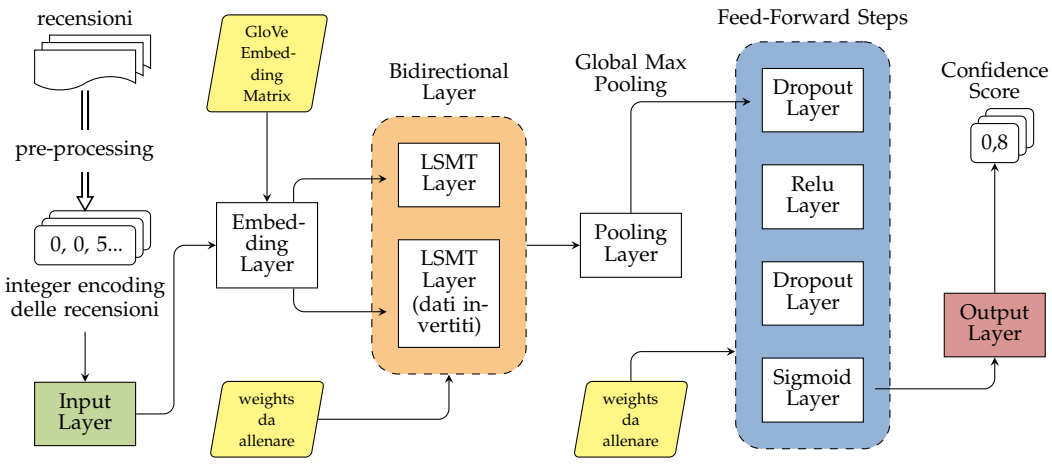


Figura 2.1: Processo di previsione della Bidirectional LSTM DNN generata.

L'architettura complessiva del modello di classificazione è composta dai seguenti layer:

1. il **layer d'ingresso** accetta in input le recensioni rappresentate sotto forma di vettori numerici di dimensione 500. Questi sono generati dall'integer encoding delle recensioni precedentemente pre-processate ottenuto assegnando ad ogni token uno specifico valore numerico intero. Questa funzione è fornita da uno strumento definito tokenizer. Se un vettore è troppo lungo viene troncato per rispettare la dimensione prevista, se è invece troppo corto viene applicato il padding, che consiste nell'inserire una serie di zeri all'inizio per incrementarne la lunghezza.

2. i dati vengono trasmessi ad un **Embedding Layer** i cui parametri sono stati già assegnati inizialmente e sono rappresentati da una matrice di *GloVe Embedding Vectors* di dimensione 50.001 x 300. Le righe della matrice corrispondono all'encoding dei token più frequentemente utilizzati nelle review del Train e Validation Data Sample pre-processato come evidenziato nella sezione precedente, mentre le colonne rappresentano i valori numerici delle dimensioni dei GloVe Embedding Vectors di ogni token. La prima riga indica l'encoding di caratteri riservati per il padding ed è espressa da un vettore formato da soli zeri. Gli encoding dei token presenti all'interno della matrice che non hanno un corrispettivo GloVe Embedding Vector sono rappresentati anch'essi da vettori di zeri. I parametri di questa matrice, in

fase di allenamento, non vengono modificati dalla Optimizer Function del modello ad iterazioni successive imponendo ai token di mantenere la stessa rappresentazione vettoriale invariata nel tempo.

3. I dati trasformati dall'Embedding Layer vengono gestiti da un **Bidirectional LSTM Layer**. In fase di allenamento ad ogni step di elaborazione dei due layer LSTM viene fornito in input, assieme agli elementi da elaborare, l'intero storico dei loro stati di rappresentazione. Questo processo garantisce che la DNN mantenga in memoria nel tempo tutte le informazioni ricavate dall'elaborazione delle recensioni e possa consultarle per affinare le proprie capacità di Sentiment Analysis. L'utilizzo di due layer LSTM che analizzano le recensioni in ordine cronologico opposto permette inoltre al modello di valutare più dettagliatamente il contesto di ogni documento.

4. Un **Layer di Pooling** gestisce i dati, che sono organizzati in serie di vettori per ogni elemento, ed applica loro il Global Max Pooling restituendo come rappresentazione un insieme di vettori contenenti un singolo valore ciascuno.

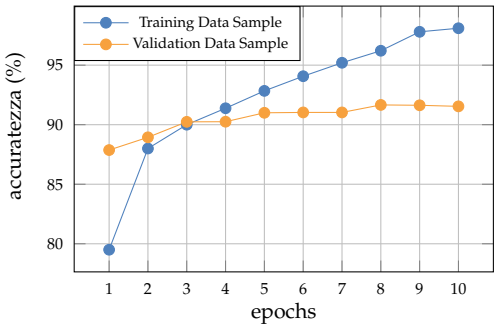
5. Da questo step a quello finale la DNN elabora ulteriormente i dati come una Feed-Forward Neural Network, evitando quindi che i successivi layer considerino anche i dati storici. Questo processo traguarda l'obiettivo di ottenere una possibile maggiore accuratezza di previsione rispetto ai risultati finora conseguiti, senza appesantire i tempi di esecuzione. I dati quindi vengono elaborati da un **Dropout Layer** che li trasforma tramite la Dropout Regularization con lo scopo di prevenire l'Overfitting durante l'allenamento.

6. L'output è gestito in seguito da un **Dense Layer** che utilizza una funzione di attivazione *relu (REctified Linear Unit)* e restituisce un risultato rappresentato da 64 unità vettoriali. Questo tipo di funzione interagisce con ogni elemento `x`, per ognuno di essi applica `max(x,0)` restituendo quindi il valore 0 per gli input negativi o nulli e per quelli positivi il numero della loro effettiva rappresentazione.

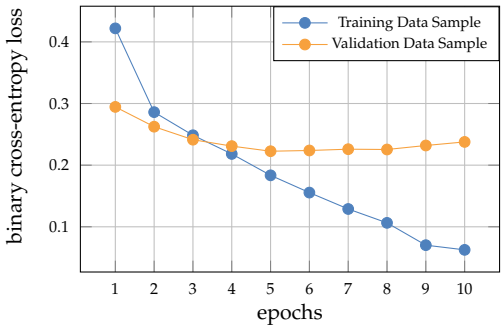
7. I dati vengono elaborati da un altro **Dropout Layer**.

8. Il **layer di output** consiste in un **Dense Layer** che elabora un'unica unità computazionale in output rappresentata dal risultato di una *funzione di attivazione Sigmoid*. Questa funzione evidenzia la confidence score del sentiment di una recensione che è espressa da un valore decimale compreso tra 0 e 1. Valori inferiori a 0,5 esprimono la previsione di una recensione con sentiment negativo (label 0) e i valori superiori o uguali a 0,5 indicano invece una recensione con sentiment previsto positivo (label 1).

Il modello è stato allenato utilizzando l'accelerazione di una *GPU Nvidia RTX 2080 8GB Super* ed è stato applicato il tuning ai seguenti iperparametri: *Optimizer Function* scelta tra *rmsprop* e *adam*; *Dropout Rate* dei due Dropout Layer con valori 0,1; 0,2 e 0,5; inizializzazione dei *weights* del primo Dense Layer fornita dalle funzioni: *uniform*; *lecun_uniform*; *normal*; *glorot_normal*; *glorot_uniform*. Il tuning è stato applicato analizzando quale fosse la migliore accuratezza del Validation Data Sample al variare del valore degli iperparametri sopra indicati. Il modello migliore riporta un dropout rate di 0,2, l'utilizzo di una Optimizer Function adam e una funzione glorot_normal per l'inizializzazione dei parametri del primo Dense Layer conseguendo un'accuratezza di previsione di circa il 91,6% per le recensioni del Validation Data Sample (Figura 2.2) e del 91,7% per quelle del Test Data Sample all'ottava iterazione (*epoch*). Il risultato non si distanzia troppo da quello previsto dall'attuale stato dell'arte per il dataset in questione [11] ed il modello generato ha comunque un'accuratezza superiore rispetto a quello costruito nell'esperimento in [5] (90%).



(a) Accuratezza dei Train e Validation Data Sample per le varie epochs di allenamento.



(b) Loss Score dei Train e Validation Data Sample per le varie epochs di allenamento.

Figura 2.2: Processo di Supervised Learning con gli iperparametri scelti.

2.3 Sistema Black-Box

Per procedere alla simulazione dell'attacco da parte di un ipotetico utente al modello di Sentiment Analysis era necessario fornire un sistema che gli permettesse di interagire con lo stesso. Quando l'utente si interfaccia ad un qualsiasi sistema di interazione con un modello questo può fornire tre diversi contesti applicativi a seconda di come è stato strutturato: *Black-Box*; *White-Box* e *Grey-Box*.

- I sistemi **Black-Box** prevedono di restituire all'utente solamente il risultato di computazione rispetto all'input che viene fornito. Sia la loro architettura che il loro comportamento interno sono nascosti e quindi solo intuibili. Questo tipo di sistema è quello più comunemente reso disponibile all'utente.

- I sistemi **White-Box** offrono, oltre ai risultati della propria computazione rispetto ad un determinato input, anche una visione completa della loro struttura e delle loro caratteristiche funzionali. Questo implica la più veloce comprensione del loro funzionamento rispetto ad un sistema Black-Box, ma rende anche più semplice l'implementazione di algoritmi di attacco o hacking ad essi rivolti.

- I sistemi **Grey-Box** costituiscono una soluzione intermedia tra i due sistemi illustrati in precedenza permettendo una visione parziale della loro architettura interna.

Il modello che utilizza l'esperimento in [5] è un sistema Black-Box che ricevendo in input una serie di recensioni ritorna in output la loro confidence score rispetto al sentiment previsto. Nel lavoro di questa tesi è stato riproposto a servizio dell'attaccante il sistema Black-Box originario al quale è stata aggiunta una funzione per valutare l'accuratezza di previsione di una serie di recensioni rispetto ai loro target label. L'architettura di questo sistema dispone internamente dei seguenti strumenti:

- `__model`: Bidirectional LSTM NN dedicata per il Sentiment Analysis delle recensioni in ingresso e descritta precedentemente.

- `__tokenizer`: struttura dati che gestisce l'Integer Encoding dei 50.000 token

più frequenti nelle recensioni del Train e Validation Data Sample utilizzati per allenare la DNN per il Sentiment Analysis (`__model`);

- `__MAXLEN`: variabile con valore 500 che indica la lunghezza massima del vettore dell'encoding di una recensione passata in ingresso;
- `__preprocesser`: classe per applicare il pre-processing alle recensioni fornite in input al sistema, il cui funzionamento è stato descritto in precedenza.

Il sistema fornisce all'utente tre funzioni per interagire con il modello:

- `predict_sentiment`: funzione che riceve in input una recensione:
  1. applica il pre-processing tramite `__preprocesser` ed esegue l'integer encoding di ogni token che la compone tramite `__tokenizer`;
  2. tronca il vettore risultante se questo risulta più lungo del valore di `__MAXLEN` (valore 500) diversamente, se risulta essere più corto, applica il padding;
  3. fornisce il vettore alla DNN (`__model`) che restituisce in output il confidence score previsto.
- `predict_all`: esegue lo stesso procedimento di `predict_sentiment` applicato però ad una serie di recensioni e restituisce per ognuna di essa il confidence score;
- `evaluate`: riceve in input una serie di recensioni ed il label di Sentiment a loro assegnato (0 per le recensioni negative e 1 per quelle positive):
  1. applica il pre-processing e l'Integer Encoding delle recensioni, aggiustando la lunghezza dei vettori risultanti come avviene nella funzione `predict_sentiment`;
  2. calcola l'accuratezza di previsione della DNN (`__model`) delle recensioni fornite in ingresso rispetto ai loro target label.

2.4 Software utilizzato

Il processo di costruzione e allenamento della Bidirectional LSTM DNN di Sentiment Analysis e quello di pre-processing delle recensioni sono stati implementati tramite l'utilizzo di librerie scritte in linguaggio *Python* [12].

2.4.1 Strumenti software per gestire il modello di previsione

Per l'implementazione dell'architettura del modello d'analisi e per allenarlo alla categorizzazione delle recensioni è stata adottata *Tensorflow2.0* [13]: libreria open source per il Machine Learning, particolarmente utilizzata per costruire e allenare DNN ed in grado di gestire in modo automatico l'accelerazione computazionale di una o più GPU presenti nel sistema. È stato utilizzato nell'esperimento il modulo *keras* con i seguenti sottomoduli:

- **keras.preprocessing.text.Tokenizer:** permette la costruzione di un vocabolario dei token presenti all'interno di un insieme di documenti tramite la funzione `fit_on_texts`, nonché di assegnare ad ognuno di essi un valore numerico intero crescente in modo inversamente proporzionale alla loro frequenza. Il valore 0 è riservato a caratteri speciali di padding. La funzione `texts_to_sequences` trasforma i documenti testuali in vettori integer encoded codificati in base al vocabolario definito da `fit_on_texts`.

- **keras.preprocessing.sequence.pad_sequences:** applica il padding a delle sequenze di vettori integer encoded troncandone la lunghezza qualora sia richiesto dall'utente.

- **tensorflow.keras.models.Sequential:** permette la generazione di una DNN e fornisce le seguenti funzioni:

  - `add`: aggiunge un layer selezionato dall'utente al modello;
  - `compile`: configura i parametri per l'allenamento del modello, ad esempio la tipologia di Optimizer Function;
  - `fit`: suddivide inizialmente una serie di istanze in ingresso in Train e Validation Data Sample, successivamente allena il modello sul Train

Data Sample, specificando il numero di campioni di elementi in cui suddividerlo ad ogni iterazione (batch_size) e il numero totale di iterazioni per cui verrà allenato (epochs);

- `predict`: ritorna i valori di previsione del modello per una serie di elementi in input;
- `evaluate`: permette di valutare l'accuratezza di previsione del modello rispetto al vettore di elementi considerati ed i loro target label.

2.4.2 Strumenti software per il pre-processing

Il pre-processing delle recensioni è stato gestito con i seguenti strumenti:

- **Beautiful Soup** [14]: libreria che permette di ricavare informazioni da file in formato *HTML* o *XML*. Fornisce inoltre strumenti per analizzare, ricercare o modificare la struttura sintattica di contenuti testuali. È stata utilizzata per rimuovere possibili tag in formato html presenti all'interno delle recensioni.
- **Natural Language Toolkit (NLTK)** [15]: libreria largamente diffusa per l'implementazione di programmi Python con lo scopo di gestire il linguaggio naturale. Dispone di modelli di analisi pre-allenati a svolgere compiti di NLP tra cui classificazione, lemmatization e stemming e permette all'utente di interfacciarsi con diverse risorse lessicali. È stato utilizzato il modulo *Penn Treebank Tokenizer* della libreria. Si serve di espressioni regolari per eseguire vari processi di tokenization all'interno di un documento, tra i quali dividere le contrazioni sintattiche in due token distinti, procedimento utilizzato nella fase di pre-processing delle recensioni.

Capitolo 3

Attacco Avversario

3.1 La Tecnica degli Adversarial Examples

La possibilità di compromettere la robustezza di una DNN di classificazione tramite *Adversarial Examples* apportando perturbazioni alla struttura delle istanze ad esso fornite e portandola all'errata categorizzazione è stata inizialmente osservata attraverso dei test applicati al campo della *Computer Vision* [16]. I primi esperimenti sono stati effettuati con sistemi *White-Box* che garantivano all'attaccante di poter accedere alla completa architettura del modello e di poter osservare il comportamento della sua *Loss Function*.

Gli Adversarial Examples sviluppati in contesti White-Box sono generalmente costruiti tramite algoritmi di ottimizzazione *Gradient-Based* che applicano una perturbazione all'istanza in input basandosi sul valore del *gradiente* (∇) della Loss Function utilizzata per allenare il modello. Il gradiente di una funzione fornisce informazioni relative alla direzione in cui essa cresce o decresce più rapidamente rispetto al valore dato. Gli algoritmi di *Deep Learning* lo utilizzano in fase di apprendimento per individuare il punto di minimo locale della Loss Function, indicatore del minimo errore di predizione da parte del modello rispetto ad una serie di istanze in input ed i loro label. Gli algoritmi di attacco Gradient-Based ricercano il gradiente della Loss Function modificando il valore dell'elemento in input in modo tale da ottenere un leggero distanziamento dal minimo locale della funzione. Questo processo è ripetuto fino a quando il modello classifica l'istanza erroneamente. Esistono varie implementazioni di questi algoritmi, tutte basate sul

processo esposto, la più comunemente adottata è la *Fast Gradient Sign Method* [17].

L'ottimizzazione Gradient-Based è efficiente quando utilizzata per applicare leggere perturbazioni alle immagini, in quanto la modifica di alcuni pixel che le compongono viene difficilmente percepita dall'uomo. Differentemente la sua applicazione al linguaggio naturale attraverso l'alterazione dell'encoding di un token all'interno di un documento può modificare il significato del testo in modo evidente. Inoltre, se l'interazione tra l'utente ed un modello di previsione avviene tramite un sistema *Black-Box*, è inibito l'utilizzo di questi metodi in quanto non è resa disponibile la Loss Function. Attaccare quindi modelli protetti da sistemi Black-Box risulta più complesso perché il loro processo di esecuzione è nascosto all'attaccante e conseguentemente il tipo di algoritmo d'attacco varia in base al campo d'applicazione. Pertanto questo tipo di sistema è stato adottato dall'esperimento in [5] e riproposto nella tesi, anche in considerazione della sua più comune applicazione in contesti reali.

3.2 Struttura dell'Algoritmo di Attacco

L'esperimento prevede di attaccare tramite un sistema Black-Box un modello di previsione allenato per svolgere *Sentiment Analysis* delle recensioni dei film classificandole come positive (1) o negative (0) attraverso Adversarial Examples. Viste le limitazioni dei metodi di ottimizzazione Gradient-Based applicati all'*NLP* e l'impossibilità di usufruirne nel contesto adottato, è stato proposto un *algoritmo genetico* di ottimizzazione come metodo d'attacco che genera Adversarial Examples sintatticamente e semanticamente simili alle recensioni in input tramite la sostituzione di parole che le compongono.

Gli algoritmi genetici (Figura 3.1) sono metodi utilizzati per risolvere problemi di ottimizzazione e il loro flusso di esecuzione è basato sul processo di selezione naturale evolutiva. Essi agiscono su un insieme di elementi, denominati *popolazione* o *generazione*, che costituiscono possibili soluzioni al problema. Ad ogni step di esecuzione selezionano in modo casuale un certo numero di istanze della popolazione denominate "genitori". Queste vengono successivamente utilizzate, tramite una procedura definita *crossover*, per produrre una nuova popolazione

di elementi, "figli" dell'attuale generazione, che poi vengono opportunamente modificati tramite un processo chiamato *mutation*. Ciò permette che l'insieme degli elementi dell'attuale popolazione si evolva gradualmente perfezionando la soluzione del problema.

L'algoritmo di ottimizzazione utilizzato nell'esperimento in [5] è stato ripreso nel lavoro della tesi ed implementato attraverso l'applicazione di modifiche finalizzate ad affinare le tempistiche e l'accuratezza dell'attacco. Partendo dalla recensione iniziale l'algoritmo apporta una serie di perturbazioni alla sua struttura generando ad ogni step diverse popolazioni di recensioni con termini modificati al fine di ingannare il modello di Sentiment Analysis. Le perturbazioni (processo di *mutation*) consistono nella modifica di un singolo termine interno alla recensione che è considerato ogni volta casualmente. Non vengono contemplate per la sostituzione parole già modificate, vocaboli comuni e Named-Entity. La popolazione iniziale è creata perturbando la recensione originale un certo numero di volte. I "figli" delle generazioni successive sono creati da coppie di recensioni "genitori" della popolazione precedente selezionate in maniera casuale (processo di *crossover*), i migliori vengono perturbati più volte. Se durante l'elaborazione venisse individuata una recensione che riesce ad ingannare il modello, il processo terminerebbe con esito positivo, diversamente continuerebbe fino ad un numero massimo di step e qualora non trovasse soluzioni l'attacco avrebbe esito negativo.

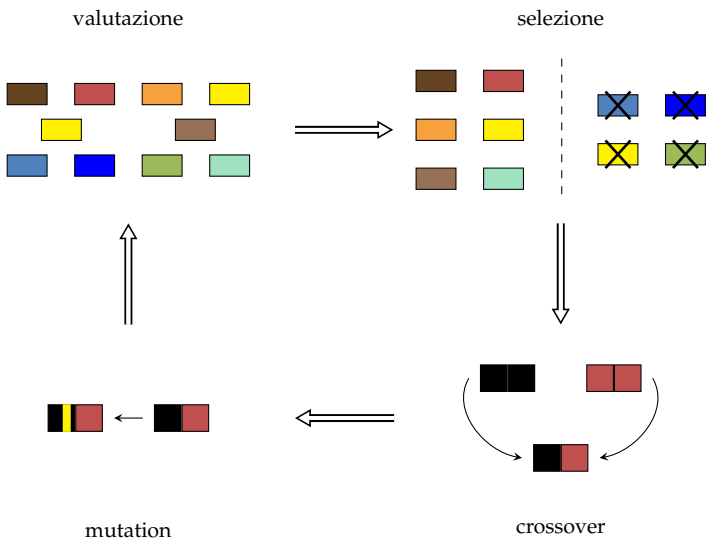


Figura 3.1: Illustrazione dell'esecuzione di un algoritmo genetico.

3.2.1 Strutture dati utilizzate per l'attacco

Per costruire degli Adversarial Examples efficienti sono stati utilizzati nell'esperimento originale e riproposte nella tesi le seguenti strutture dati: *Matrice di Distanza* tra vocaboli e *Google One Billion Words Language Model*.

Matrice di Distanza tra vocaboli

A garanzia che i vocaboli scelti in sostituzione di quelli del testo originale fossero loro sinonimi è stata generata una matrice in cui ogni cella indica numericamente, con un valore decimale compreso tra 0 e 1, la differenza semantica tra due parole definite dal loro Integer Encoding. Il numero 0 indica l'equivalenza tra vocaboli, mentre il numero 1 una loro netta discordanza. La differenza tra vocaboli è stata calcolata in base alla loro rappresentazione nello spazio vettoriale dei *GloVe Embedding Vectors* di 300 dimensioni, a cui è stato applicato il metodo denominato *counter-fitting*, presentato nell'esperimento dell'articolo *Counter-fitting Word Vectors to Linguistic Constraints* [18]. Questa tecnica impone di processare gli elementi dello spazio vettoriale inserendo vincoli di sinonimia e contrarietà, garantendo così ai vettori di incrementare la capacità di giudizio della similarità semantica tra loro. Il vocabolario dei counter-fitted Word Vectors comprende un insieme di più di 65.000 token. La matrice è stata generata con i seguenti step:

1. Data la dimensione elevata del vocabolario dei counter-fitted Word Vectors sono stati selezionati, per motivi computazionali, solo i token corrispondenti a quelli presenti nelle recensioni del Test Data Sample, costruendo così un nuovo vocabolario di dimensione più limitata.

2. Il vocabolario di Embedding generato presentava un totale di 40.567 token, al quale ne sono stati aggiunti degli altri prelevati dal vecchio vocabolario aumentando così la dimensione a 45.000 vocaboli. È stato infine applicato l'Integer Encoding ad ogni elemento.

3. Successivamente è stata generata la matrice di distanza applicando la formula della distanza al coseno (Equazione 3.1) tra ogni coppia di vettori rappresentanti l'Embedding dei token del vocabolario. Questa formula è utilizzata frequentemente nell'analisi di documenti e si basa sulla misura del

coseno degli angoli tra coppie di vettori (A, B) che rappresenta la similarità direzionale. Il risultato è un numero decimale compreso tra 0 e 1 dove valori prossimi allo 0 indicano una minore distanza tra i due vettori e valori vicini a 1 indicano la maggior distanza tra loro.

$$Distance(A,B) = 1 - \cos(\theta) = 1 - \frac{A \cdot B}{\|A\|\|B\|} = 1 - \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{3.1}$$

Google One Billion Words Language Model

Allo scopo di verificare che i sinonimi delle parole scelte per la sostituzione siano coerenti rispetto al contesto della frase è stato utilizzato il modello *Google One Billion Words Language Model* proposto nell'articolo *One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling* [19]. Esso offre funzioni di *Statistical Language Modeling* per calcolare la probabilità di una parola w_i all'interno di una frase $S = (w_1, w_2, ...w_i, ...w_n)$ di lunghezza n considerando la sequenza di quelle precedenti (Equazione 3.2).

$$P(w_i|w_1, w_2, ..., w_{i-1}) = \frac{P(w_1, w_2, ..., w_i)}{P(w_1, w_2, ..., w_{i-1})} \tag{3.2}$$

Le parole con una probabilità più elevata indicano una maggiore affinità al contesto dell'intera frase.

3.2.2 Processo di Ottimizzazione

Il processo di ottimizzazione (Algoritmo 3.1) dell'algoritmo genetico è eseguito tramite una funzione principale che accetta in input le seguenti variabili:

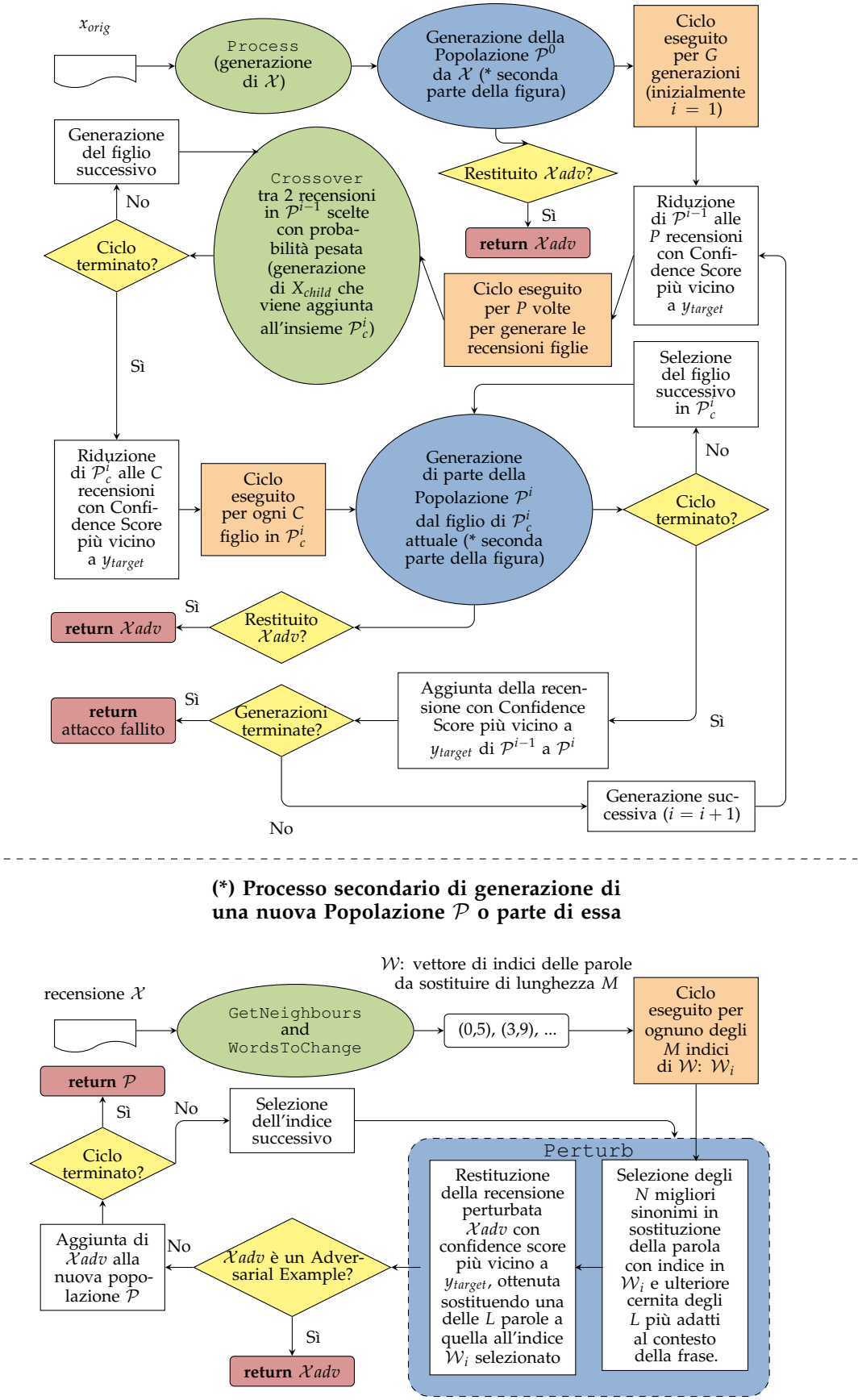
- una recensione x_{orig};
- il label ad essa riferito y_{target};
- Il numero M di possibili parole da sostituire ogni qual volta si perturba la recensione;
- Il numero N di migliori sinonimi da considerare per ogni parola da sostituire;

- Il numero L tra i migliori N sostituti di una parola che mantengono il significato e la semantica della recensione il più inalterati possibile;
- Il numero P di membri massimi della popolazione da considerare per costruire ogni generazione;
- Il numero C di figli da scegliere a cui applicare le perturbazioni funzionali a generare la nuova popolazione;
- Il numero G di generazioni massime che l'algoritmo può produrre.

La recensione x_{orig} viene inizialmente processata dalla funzione ausiliaria `Process` che restituisce $\mathcal{X}$ come rappresentazione della recensione originale attraverso una lista delle n frasi che la compongono elencate cronologicamente ($\mathcal{X} = \{i_1, i_2, ..., i_n\}$). Ogni frase i è rappresentata a sua volta da tutti i k token che la compongono in sequenza con indici in $\mathcal{X}$ $\{(i, j_1), (i, j_2), ..., (i, j_k)\}$ ed ognuno di essi viene segnalato o meno come *Named-Entity*. La recensione processata $\mathcal{X}$ è poi gestita dalla funzione ausiliaria `GetNeighbours` che restituisce un dizionario $\mathcal{N}$ al cui interno ogni indice (i, j) presenta una lista di sinonimi ordinati per similarità decrescente rispetto ad ogni token specificato dallo stesso indice (i, j) in $\mathcal{X}$. Vengono poi selezionati ed inseriti in una lista $\mathcal{W}$ dalla funzione ausiliaria `WordsToChange` gli indici in $\mathcal{X}$ delle M parole da sostituire per generare le recensioni avversarie. Successivamente vengono generate M perturbazioni di $\mathcal{X}$ tramite la funzione `Perturb`. All'interno di ogni perturbazione $\mathcal{X}adv$, la sostituzione di una parola viene definita da un diverso indice presente in $\mathcal{W}$ attraverso i suoi sinonimi in $\mathcal{N}$ e la procedura è ripetuta per ogni indice presente in $\mathcal{W}$. Ogni recensione perturbata diventa membro della popolazione iniziale $\mathcal{P}^0$ e viene calcolato il valore del suo confidence score dalla funzione f. Essa consiste nella ricostruzione della recensione sotto forma di stringa con la conseguente chiamata a `predict_sentiment` del sistema Black-Box contenente il modello di previsione descritti al capitolo precedente. Ogni confidence score è inserito nel vettore $\mathcal{F}^0$ e viene arrotondato dalla funzione `round` che restituisce 0 se è minore di 0,5 e 1 se è maggiore o uguale a 0,5. Qualora il risultato di `round` del confidence score $\mathcal{F}_i^0$ di una recensione perturbata $\mathcal{P}_i^0$ corrispondesse al target label y_{target}, l'algoritmo terminerebbe la sua esecuzione restituendo $\mathcal{P}_i^0$ ricostruita sotto forma di stringa.

Diversamente l'algoritmo proseguirebbe la sua esecuzione attivando un ciclo per un massimo di G iterazioni dove ognuna delle quali costruisce una nuova popolazione di frasi $\mathcal{P}^i$ avversarie dalla generazione precedente. Ad ogni iterazione la generazione $\mathcal{P}^{i-1}$ ed il corrispondente vettore dei confidence score $\mathcal{F}^{i-1}$ vengono ridotti alle P recensioni con il confidence score più vicino al target label y_{target}. Questo processo è funzionale alla limitazione di $\mathcal{P}^{i-1}$ per le sole recensioni avversarie più efficaci. Il vettore di confidence score $\mathcal{F}^{i-1}$ è normalizzato dalla funzione ausiliaria `Normalize`, che restituisce il vettore p contenente la distribuzione di probabilità per le recensioni in $\mathcal{P}^{i-1}$ direttamente proporzionale alla vicinanza dei loro confidence score rispetto al target label. Sono poi selezionate con probabilità pesata in p per P volte coppie di recensioni di $\mathcal{P}^{i-1}$ e da ognuna di esse è generata una recensione figlia $\mathcal{X}child$ tramite la funzione ausiliaria `Crossover`. Ogni recensione figlia viene assegnata al vettore $\mathcal{P}c^i$ ed il suo confidence score calcolato con f nel vettore $\mathcal{F}c^i$. $\mathcal{P}c^i$ è poi ridotto alle C recensioni i cui confidence score sono più vicini a y_{target}. Per ognuna di esse è ricavato il dizionario di sinonimi tramite la funzione ausiliaria `GetNeighbours` e `WordsToChange` ricerca gli indici di M parole da sostituire. Di seguito viene applicata ad ogni recensione in $\mathcal{P}c^i$ la funzione `Perturb` per ogni M indice ricavato e la recensione così perturbata $\mathcal{X}adv$ viene aggiunta al vettore della nuova generazione $\mathcal{P}^i$. Il suo confidence score viene calcolato dalla funzione f come nei casi precedenti e, qualora il valore risultante arrotondato da `round` fosse uguale a y_{target}, la recensione verrebbe restituita sotto forma di stringa con conseguente successo dell'attacco. In ogni caso il confidence score viene aggiunto al vettore $\mathcal{F}^i$ della generazione attuale. La migliore recensione della generazione precedente $\mathcal{P}^{i-1}$ è aggiunta alla generazione corrente $\mathcal{P}^i$ ed il relativo confidence score al vettore $\mathcal{F}^i$. Questa azione si esegue al fine di garantire, nel caso non ci siano stati progressi nel processo di ottimizzazione, che la soluzione del problema fornita da $\mathcal{P}^i$ possa essere ottimale quanto quella in $\mathcal{P}^{i-1}$. Qualora l'algoritmo completasse il ciclo senza aver prodotto una recensione avversaria efficace l'attacco fallirebbe. Il processo di ottimizzazione è illustrato in Figura 3.2.

Un esempio di Adversarial Example di una recensione generato correttamente dal processo di ottimizzazione appena illustrato è visualizzato in Tabella 3.1.



(*) Processo secondario di generazione di una nuova Popolazione $\mathcal{P}$ o parte di essa

Figura 3.2: Illustrazione del Processo di Ottimizzazione.

Algoritmo 3.1: Processo di Ottimizzazione

Input : La recensione x_{orig}; il suo target label y_{target}; gli N sinonimi da considerare; le M parole da modificare; le L sostituzioni da valutare; i C figli da selezionare; i P membri della popolazione da considerare; le G generazioni massime.

Output : La recensione x_{orig} perturbata se l'attacco ha avuto successo.

$\mathcal{X} \leftarrow$ `Process(`x_{orig}`)`
$\mathcal{N} \leftarrow$ `GetNeighbours(`$\mathcal{X}$`)`
$\mathcal{W} \leftarrow$ `WordsToChange(`$\mathcal{X}, \mathcal{N}, M$`)`
for $i \leftarrow 1$ **to** M parole **do**
    $\mathcal{X}adv \leftarrow$ `Perturb(`$\mathcal{X}, y_{target}, \mathcal{N}, \mathcal{W}_i, N, L$`)`
    $\mathcal{F}_i^0 \leftarrow f$`(`$\mathcal{X}adv$`)`
    if `round(`$\mathcal{F}_i^0$`)` $== y_{target}$ **then**
        return $\mathcal{X}adv$ ricostruita sotto forma di stringa
    end if
    $\mathcal{P}_i^0 \leftarrow \mathcal{X}adv$
end for
for $i \leftarrow 1$ **to** G generazioni **do**
    Riduci $\mathcal{P}^{i-1}$ e $\mathcal{F}^{i-1}$ in base ai P valori di $\mathcal{F}^{i-1}$ più vicini a y_{target}
    $p \leftarrow$ `Normalize(`$\mathcal{F}^{i-1}$`)`
    for $j \leftarrow 1$ **to** P membri di $\mathcal{P}^{i-1}$ **do**
        Seleziona con probabilità pesata in p, $\mathcal{X}1$ da $\mathcal{P}^{i-1}$
        Seleziona con probabilità pesata in p, $\mathcal{X}2$ da $\mathcal{P}^{i-1}$
        $\mathcal{X}child \leftarrow$ `Crossover(`$\mathcal{X}1, \mathcal{X}2$`)`
        $\mathcal{F}c_j^i \leftarrow f$`(`$\mathcal{X}child$`)`
        $\mathcal{P}c_j^i \leftarrow \mathcal{X}child$
    end for
    Riduci $\mathcal{P}c^i$ in base ai C valori di $\mathcal{F}c^i$ più vicini a y_{target}
    $pos \leftarrow 2$

for $j \leftarrow 1$ **to** C figli **do**
  $\mathcal{N}^j \leftarrow$ `GetNeighbours` $(\mathcal{P}c^i_j)$
  $\mathcal{W}^j \leftarrow$ `WordsToChange` $(\mathcal{P}c^i_j, \mathcal{N}^j, M)$
  for $k \leftarrow 1$ **to** M parole **do**
    $\mathcal{X}adv \leftarrow$ `Perturb` $(\mathcal{P}c^i_j, y_{target}, \mathcal{N}^j, \mathcal{W}^j_k, N, L)$
    $\mathcal{F}^i_{pos} \leftarrow f(\mathcal{X}adv)$
    if `round` $(\mathcal{F}^i_{pos}) == y_{target}$ **then**
      return $\mathcal{X}adv$ ricostruita sotto forma di stringa
    end if
    $\mathcal{P}^i_{pos} \leftarrow \mathcal{X}adv$
    $pos \leftarrow pos + 1$
  end for
end for
$\mathcal{P}^i_1 \leftarrow \mathcal{P}^{i-1}_x | \mathcal{F}^{i-1}_x = \texttt{min}_{\forall \mathcal{F}^{i-1}_y \in \mathcal{F}^{i-1}} (| y_{target} - \mathcal{F}^{i-1}_y |)$
$\mathcal{F}^i_1 \leftarrow \mathcal{F}^{i-1}_x | \mathcal{F}^{i-1}_x = \texttt{min}_{\forall \mathcal{F}^{i-1}_y \in \mathcal{F}^{i-1}} (| y_{target} - \mathcal{F}^{i-1}_y |)$
end for
return attacco fallito

Previsione Recensione Originale = **Negativa** (Confidence Score: 0,001)
There's only one thing I need to say about this movie [...] The story is horrible, the acting is terrible (c'mon, it's Shaq!) and I'd rather see Capra in Free Willy (equally horrible) twice before ever seeing this movie.
Previsione Recensione Perturbata = **Positiva** (Confidence Score: 0,811)
There's only one thing I need to say about this movie [...] The story is ghastly, the acting is terrifying (c'mon, it's Shaq!) and I'd rather see Capra in Free Willy (equally gruesome) twice before ever seeing this cinematic.

Tabella 3.1: Esempio di Adversarial Example generato con successo.

3.2.3 Funzioni Ausiliarie

Funzione `Process`

La funzione (Algoritmo 3.2) prende in input la recensione originale x_{orig}, che viene suddivisa cronologicamente rispetto alle singole frasi che la compongono. Quest'azione renderà il processo della valutazione delle parole sostitute più semplice in quanto queste verranno valutate nel contesto della singola frase e non dell'intera recensione. Ogni frase è suddivisa a sua volta in base ai token che la costituiscono e ad ognuno di essi è assegnato un label se viene riconosciuto come una Named-Entity. I token che rappresentano Named-Entity indicano parole che appartengono a categorie specifiche, come nomi propri di persona, espressioni temporali oppure entità geo-politiche, che non presentano sinonimi nel linguaggio naturale. La segnalazione di questi token permette all'algoritmo di attacco di evitare la loro sostituzione a scopo di mantenere la correttezza semantica del documento. Il suo processo d'esecuzione è illustrato in Figura 3.3.

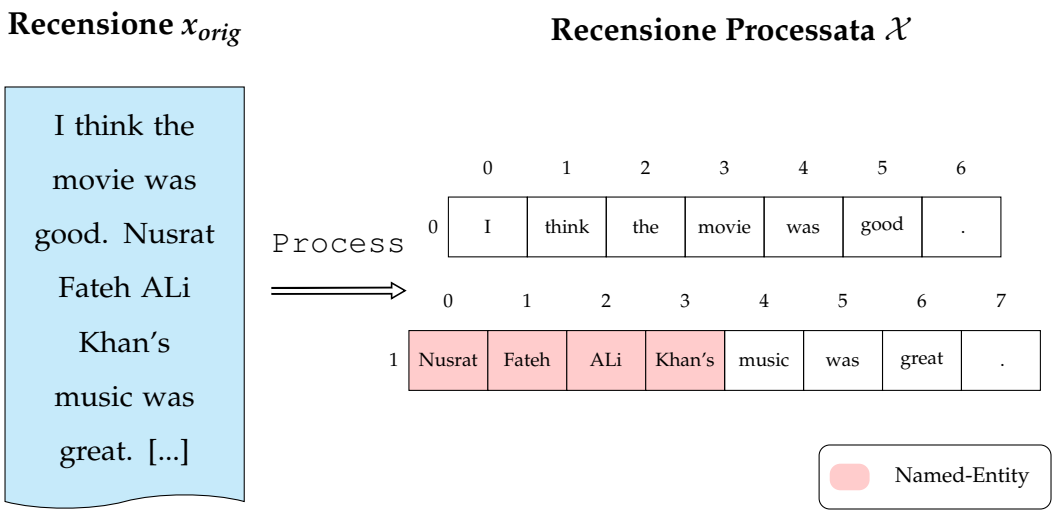


Figura 3.3: Esempio di esecuzione della funzione ausiliaria `Process`.

Algoritmo 3.2: Funzione `Process`

Input : La recensione x_{orig}

Output : $\mathcal{X}$: recensione x_{orig} suddivisa in frasi suddivise in token eventualmente segnalati come Named-Entity

$\mathcal{X} \leftarrow \emptyset$
$i \leftarrow 1$
foreach frase $s \in x_{orig}$ presa secondo l'ordine **do**
    $\mathcal{X}_{(i)} \leftarrow \emptyset$
    $j \leftarrow 1$
    foreach token $t \in s$ preso secondo l'ordine **do**
        $\mathcal{X}_{(i,j)} \leftarrow t$
        if t è una Named-Entity **then**
            Segnala $\mathcal{X}_{(i,j)}$ come posizione di una Named-Entity
        end if
        $j \leftarrow j + 1$
    end foreach
    $i \leftarrow i + 1$
end foreach
return $\mathcal{X}$

Funzione `GetNeighbours`

Questa funzione ausiliaria (Algoritmo 3.3) prende in input una recensione processata $\mathcal{X}$ e restituisce un dizionario $\mathcal{N}$ in cui ogni indice corrisponde a quelli dei token in $\mathcal{X}$. Ogni indice (i, j) in $\mathcal{N}$ è composto da una lista con un massimo di T sinonimi (50 per l'esperimento), ordinati in modo decrescente in base alla vicinanza semantica al token in $\mathcal{X}_{(i,j)}$ e scelti per maggior similarità. Per i token che sono già stati modificati in precedenza, che sono segnalati come Named-Entity o che fanno parte delle parole comuni - *stopword* - della lingua Inglese ($\mathcal{S}$) vengono restituite delle liste vuote. I sinonimi di ogni token $\mathcal{X}_{(i,j)}$ sono stati selezionati tramite la funzione `GetMostSimilar` che sfrutta la matrice di distanza tra vocaboli ($\mathcal{M}$), illustrata precedentemente ed esegue i seguenti step:

1. Controlla se l'encoding della parola $\mathcal{X}_{(i,j)}$ esiste all'interno di $\mathcal{M}$, in caso contrario restituisce una lista vuota;

2. Recupera gli encoding dei T vocaboli più simili a $\mathcal{X}_{(i,j)}$ secondo $\mathcal{M}$ ordinandoli in ordine decrescente;

3. Riduce eventualmente la lista di encoding eliminando i candidati con distanza maggiore di una soglia δ, che per l'esperimento è stata fissata a 0,5;

4. Restituisce la lista di vocaboli riferiti agli encoding calcolati al punto precedente.

Algoritmo 3.3: Funzione `GetNeighbours`

Input : La recensione processata $\mathcal{X}$.

Output : $\mathcal{N}$, dizionario contenente per ogni indice dei token in $\mathcal{X}$ un massimo di T sinonimi con distanza non superiore a δ da essi e ordinati per similarità decrescente.

Data :
- $\mathcal{S}$: set delle stopword della lingua Inglese;
- $\mathcal{M}$: matrice di distanza tra vocaboli;
- T: numero massimo di sinonimi da considerare per ogni token;
- δ: distanza massima che ogni sinonimo può avere rispetto al token a cui si riferisce.

```
1  for i ← 1 to length(X) do
2      for j ← 1 to length(X_i) do
3          if X_(i,j) è Named-Entity or X_(i,j) è modificata or X_(i,j) ∈ S then
4              N_(i,j) ← ∅
5          else
6              N_(i,j) ← GetMostSimilar(X_(i,j), M, T, δ)
7          end if
8      end for
9  end for
10 return N
```

Funzione `WordsToChange`

Dato il dizionario $\mathcal{N}$, generato dalla funzione `GetNeighbours`, viene calcolata una distribuzione di probabilità p per ogni token della recensione $\mathcal{X}$ proporzionale alla quantità di sinonimi disponibili individualmente con la seguente funzione (Equazione 3.3):

$$\forall(x,y) \in \mathcal{X} : p_{(x,y)} = \frac{\texttt{length}(\mathcal{N}_{(x,y)})}{\sum_{\forall(x',y')\in\mathcal{X}} \texttt{length}(\mathcal{N}_{(x',y')})} \tag{3.3}$$

Vengono poi selezionati casualmente, con probabilità pesata in p, gli M indici che in $\mathcal{X}$ evidenziano i token che verranno successivamente sostituiti per costruire le recensioni perturbate. Sono escluse dalla selezione le parole già modificate, le Named-Entity e le stopword della lingua inglese, in quanto le liste dei sinonimi che le rappresentano sono vuote ed hanno quindi lunghezza, e conseguente probabilità, pari a 0. La lista di indici risultante $\mathcal{W}$ sarà restituita dalla funzione.

Funzione `Perturb`

La funzione ausiliaria `Perturb` (Algoritmo 3.4) è utilizzata per generare le perturbazioni delle recensioni di una popolazione tramite la sostituzione di una parola al suo interno ed il processo è paragonabile a quello di *mutation* degli algoritmi genetici di ottimizzazione. Essa prevede in input:

- una recensione processata $\mathcal{X}$ di una popolazione;
- il target label y_{target} di $\mathcal{X}$;
- il dizionario di sinonimi $\mathcal{N}$ calcolato dalla funzione `GetNeighbours`;
- l'indice $w = (x, y)$ del token da sostituire;
- il numero massimo N di sinonimi da valutare per la sostituzione;
- Il numero L corrispondente ai migliori sinonimi tra gli N scelti che mantengono il significato e la semantica della recensione $\mathcal{X}$ maggiormente inalterati;

Algoritmo 3.4: Funzione `Perturb`

Input : La recensione processata $\mathcal{X}$; il suo target label y_{target}; Il dizionario di sinonimi $\mathcal{N}$; la tupla $w = (x, y)$ contente l'indice della frase (x) e del token al suo interno da sostituire (y); il numero massimo di sinonimi N da considerare per la sostituzione; il numero di sostituti migliori L da valutare per la sostituzione.

Output : La perturbazione di $\mathcal{X}$ con confidence score più vicino a y_{target}.

Data : $\mathcal{G}_{lm}$: *Google One Billion Words Language Model*.

$\mathcal{K} \leftarrow \mathcal{N}_w$
for $i \leftarrow 1$ **to** N sinonimi **do**
    $\mathcal{K}_i' \leftarrow \mathcal{K}_i$
end for
$prefix \leftarrow$ Costruisci una stringa con i token da $\mathcal{X}_{(w[x],1)}$ a $\mathcal{X}_{(w[x],w[y]-1)}$
$suffix \leftarrow$ Costruisci una stringa con i token da $\mathcal{X}_{(w[x],w[y]+1)}$ a $\mathcal{X}_{(w[x],w[\texttt{length}(\mathcal{X}_{(x)})])}$
$p \leftarrow \texttt{GetWordsProbability}(\mathcal{G}_{lm}, prefix, \mathcal{K}', suffix)$
$\mathcal{K}'' \leftarrow$ seleziona gli L sinonimi con probabilità maggiore in p da $\mathcal{K}'$
for $i \leftarrow 1$ **to** L possibili sostituti **do**
    $\mathcal{X}adv^i \leftarrow \mathcal{X}$
    $\mathcal{X}adv^i_{(w[x],w[y])} \leftarrow \mathcal{K}_i''$
    Segnala $\mathcal{X}adv^i_{(w[x],w[y])}$ come modificata
    $\mathcal{F}_i \leftarrow f(\mathcal{X}adv^i)$
    $\mathcal{P}_i \leftarrow \mathcal{X}adv^i$
end for
return $\mathcal{P}_x | \ \mathcal{F}_x = \texttt{min}_{\forall \mathcal{F}_y \in \mathcal{F}}(|\ y_{target} - \mathcal{F}_y|)$

La funzione ricava la lista di sinonimi $\mathcal{K}$ del token con indice w dal dizionario $\mathcal{N}$ e ne riduce la dimensione agli N sinonimi più adatti costituendo $\mathcal{K}'$. Viene poi costruita $prefix$: la frase di $\mathcal{X}$ ($\mathcal{X}(w[x])$) dove è situato il token con indice in w, considerando solamente le parole che lo precedono. La frase $suffix$ è generata allo stesso modo, ma considera solamente le parole che seguono quella con indice

w. Viene poi calcolato un vettore p che indica al proprio interno lo score della probabilità di compatibilità di ogni sinonimo s considerato in $\mathcal{K}'$ all'interno della frase composta da $prefix + s + suffix$. Questo processo è eseguito tramite la funzione `GetWordsProbability` che si serve del *Google One Billion Words Language Model* ($\mathcal{G}_{lm}$) per svolgere la funzione di language modelling illustrata. Sono scelti gli L sinonimi che hanno probabilità maggiore in p, ovvero che garantiscono una minore variazione alla semantica e la sintassi della recensione originale. Si generano quindi L recensioni perturbate ognuna delle quali sostituisce il token con indice w con un diverso sinonimo scelto e lo segnala come modificato. Viene infine restituita quella il cui confidence score è più vicino al valore del target label y_{target}. Esso è calcolato tramite la funzione f che, come nell'algoritmo principale, ricostruisce la recensione sotto forma di stringa ed interroga il modello di previsione protetto dal sistema Black-Box.

Funzione `Normalize`

La funzione `Normalize` prevede di costruire il vettore di probabilità p per gli n elementi di una popolazione $\mathcal{P}$ applicando una funzione *Softmax* (Equazione 3.4) ad ogni elemento del vettore di confidence score $\mathcal{F}$, modificata per garantire la proporzionalità diretta tra probabilità e vicinanza del confidence score rispetto al target label y_{target}.

$$p_{\mathcal{P}_i} = Softmax(\mathcal{F}_i) = \frac{e^{(\frac{|1-y_{target}-\mathcal{F}_i|}{0,3})}}{\sum_{j=1}^{n} e^{(\frac{|1-y_{target}-\mathcal{F}_j|}{0,3})}} \tag{3.4}$$

Funzione `Crossover`

La funzione `Crossover` (Algoritmo 3.5) prende in input due recensioni processate ($\mathcal{X}1$ e $\mathcal{X}2$) e restituisce un'unica recensione figlia $\mathcal{X}child$. $\mathcal{X}1$ e $\mathcal{X}2$ rappresentano pertrurbazioni distinte della recensione originaria x_{orig}. Ogni indice delle due recensioni si riferisce ad un token che potrebbe essere corrispondente o diverso a seconda che sia stato o meno sostituito in una o entrambe. Il processo di generazione di $\mathcal{X}child$ prevede che, per ogni indice (i, j) delle due recensioni, venga selezionato in maniera casuale un vocabolo tra $\mathcal{X}1_{(i,j)}$ e $\mathcal{X}2_{(i,j)}$ ed assegnato all'indice (i, j) di

$\mathcal{X}child$. Qualora la parola scelta da $\mathcal{X}1$ o $\mathcal{X}2$ fosse un token modificato da una precedente perturbazione oppure una Named-Entity, questo verrebbe evidenziato anche in $\mathcal{X}child$. La selezione di ogni vocabolo è effettuata per scelta casuale da una distribuzione uniforme (U) tra i valori 0 e 1. Qualora il valore risultante fosse minore di 0,5 il vocabolo verrebbe scelto da $\mathcal{X}1$, diversamente da $\mathcal{X}2$.

Algoritmo 3.5: Funzione `Crossover`

Input : la prima recensione processata genitrice $\mathcal{X}1$; la seconda recensione processata genitrice $\mathcal{X}2$.

Output : $\mathcal{X}child$, la recensione figlia di $\mathcal{X}1$ e $\mathcal{X}2$.

```
1  for i ← 1 to length(X1) frasi do
2      for j ← 1 to length(X1_(i)) token do
3          if U(0,1) < 0,5 then
4              w ← X1_(i,j)
5          else
6              w ← X2_(i,j)
7          end if
8          Xchild_(i,j) ← w
9          if w è segnalata come modificata then
10             Segnala Xchild_(i,j) come modificata
11         end if
12         if w è segnalata come Named-Entity then
13             Segnala Xchild_(i,j) come Named-Entity
14         end if
15     end for
16 end for
17 return Xchild
```

3.3 Differenze rispetto all'esperimento originale

L'algoritmo d'attacco utilizzato ricalca la struttura principale di quello adottato dall'esperimento, ma presenta una serie di differenze applicate per provare a rendere più efficiente l'attacco e le tempistiche di generazione degli Adversarial Examples. Le modifiche sono di seguito elencate.

- Nell'esperimento originale la matrice di distanza tra vocaboli utilizza la distanza euclidea come misura di confronto tra parole. Nel lavoro della tesi si è preferito implementarla tramite la distanza al coseno generalmente più utilizzata per procedure di NLP e in grado di percepire in modo più accurato sinonimi e contrari di un vocabolo valutando la direzione della loro rappresentazione nello spazio vettoriale.

- Nel lavoro svolto la recensione viene inizialmente processata sotto forma di una lista di frasi che la compongono ed ognuna viene suddivisa nei token che la costituiscono considerati in ordine cronologico. Questo processo non avviene nell'esperimento originale ed è stato inserito per garantire un più facile riconoscimento delle parole da sostituire o che non devono essere modificate.

- È stata aggiunta un'integrazione, ovvero l'individuazione di vocaboli categorizzati come Named-Entity al fine di impedirne la sostituzione durante l'attacco. Questo garantisce che parole riferite a nomi di persone, luoghi, ecc., non siano modificate durante le perturbazioni in quanto si potrebbe compromettere il significato del testo originale.

- Il contesto considerato per la valutazione delle parole da sostituire tramite il *Google One Billion Words Language Model* è quello dell'intera frase in cui si trova il vocabolo da modificare. L'individuazione della frase risulta essere semplice, in quanto ognuna è ricavabile tramite indice. Nell'esperimento originale le parole da sostituire sono state valutate solamente in base al vocabolo precedente e successivo a quello da modificare. Questo comporta tempi di elaborazione minori, ma contesti di valutazione meno accurati.

- Originariamente la funzione ausiliaria `Perturb`, utilizzata per ricavare la

perturbazione di una recensione, selezionava al suo interno l'insieme completo di parole da considerare per la sostituzione. Una volta generata la nuova popolazione di recensioni sarebbe stata restituita quella con confidence score più vicino al target label qualora avesse fornito una predizione equivalente ad esso. Per limitare i tempi di calcolo si è preferito valutare il confidence score subito dopo ogni perturbazione effettuata tramite `Perturb`, che considera le parole da modificare singolarmente. Viene eventualmente restituita la prima recensione con predizione di Sentiment equivalente al target label. Data la differente implementazione di `Perturb`, le parole utilizzate per la sostituzione vengono valutate dalle funzioni `GetNeighbours` e `WordsToChange` prima della sua esecuzione.

- Al contrario dell'esperimento originale, che considera l'intera popolazione per la creazione della generazione successiva, si è deciso di limitarla ad i soli P membri con confidence score più vicino al target label. Questo processo è effettuato per generare figli con maggiore probabilità di costituire Adversarial Examples efficaci.

- Inizialmente ogni figlio di una nuova generazione veniva modificato tramite `Perturb` applicando la sostituzione di una sola sua parola selezionata casualmente. Nel lavoro descritto la generazione viene invece limitata a C figli con confidence score più prossimo al target label e per ognuno di essi sono generate M perturbazioni considerando parole diverse da sostituire. Questo processo è eseguito con l'obiettivo di affinare l'efficacia delle recensioni perturbate che comporranno la nuova generazione.

- Gli Adversarial Examples venivano scartati se la differenza nella loro struttura, rispetto alla recensione da cui erano stati costruiti, era maggiore del 20%. L'algoritmo d'attacco era testato su un campione di 1000 frasi. Per limitazioni computazionali, nel lavoro della tesi, si è deciso di ridurlo a solo 100 elementi suddivisi equamente tra recensioni con Sentiment positivo e negativo. A causa della limitata dimensione del campione utilizzato, la valutazione della differenza tra recensione originale ed Adversarial Examples non è stata applicata e nessun elemento è stato scartato.

3.4 Software utilizzato

L'algoritmo d'attacco è stato scritto in codice *Python* e per i processi di NLP collegati alla creazione degli Adversarial Examples è stata nuovamente utilizzata la libreria *NLTK*:

- Per segmentare la recensione nelle singole frasi che la compongono è stato utilizzato il modello **Punkt Sentence Tokenizer** pre-allenato tramite un Algoritmo di *Unsupervised Learning* per analizzare la struttura di un testo. Questo è in grado di identificare le parole ed i simboli contenuti, distinguendo quelli che fungono da delimitatori rispetto a quelli interni nelle varie frasi.

- La funzione **nltk.word_tokenize** è stata utilizzata per suddividere le frasi della recensione in token e si serve anch'essa dei modelli *Punkt Sentence Tokenizer* e *Penn Treebank Tokenizer*, quest'ultimo utilizzato anche per il preprocessing delle recensioni al fine di allenare il modello di Sentiment Analysis da attaccare.

- Per il riconoscimento delle Named-Entity è stata applicata la funzione **ntlk.pos_tag**, che si serve di un *Part-Of-Speech Tagger* (*POS-tagger*) ad assegnazione alla categoria lessicale di appartenenza dei token ricavati da *nltk.word_tokenize* (Es: verbo; aggettivo; pronome; nome; ecc.). È stata utilizzata infine la funzione **ntlk.ne_chunk** che controlla attraverso un modello di classificazione pre-allenato se i vari token sono o meno Named-Entity.

Per gestire le operazioni vettoriali e matriciali è stata utilizzata la libreria **numpy** [20], ad esempio per ordinare le recensioni di una popolazione in base al loro confidence score, ricercare i sinonimi di una parola all'interno della matrice di distanza tra vocaboli oppure scegliere con probabilità pesata casuale le recensioni da cui produrre la generazione successiva.

Capitolo 4

Valutazione Sperimentale

4.1 Esperimento

Sono stati eseguiti una serie di test allo scopo di valutare l'efficacia in termini di tempo e numero di generazioni dell'algoritmo al variare di alcuni suoi iper-parametri:

- Il numero M di possibili parole da sostituire ogni qual volta si perturba la recensione;
- Il numero N di migliori sinonimi da considerare per ogni parola da sostituire;
- Il numero L tra i migliori N sostituti di una parola che mantengono il significato e la semantica della recensione il più inalterati possibile;
- Il numero P di membri massimi della popolazione da considerare per costruire ogni generazione;
- Il numero C di figli da scegliere a cui applicare le perturbazioni funzionali a generare la nuova popolazione;

Per ognuno di essi è stato scelto un valore predefinito ($M = 5$; $N = 10$; $L = 4$; $P = 5$; $C = 2$) che a rotazione è stato incrementato per due volte mantenendo gli altri invariati. Per il test sono stati scelti due insiemi composti da recensioni che erano state correttamente classificate selezionate in modo casuale dal *Test Data Sample* utilizzato per testare il modello di *Sentiment Analysis*. Entrambi comprendono 50 recensioni con Sentiment positivo e 50 con Sentiment negativo. I

due insiemi differiscono fra loro in quanto il primo è composto da recensioni brevi, ognuna costituita da un massimo di 500 caratteri, ed il secondo è formato invece di recensioni con un numero di caratteri compreso tra 300 e 2.000. È stato selezionato questo intervallo per il secondo insieme al fine di inserire nel campione recensioni di lunghezza media, valore che corrisponde a circa 1.279 caratteri. Per ogni insieme sono state testate tutte le possibili combinazioni di iperparametri e per il primo è stato scelto per l'attacco un numero di generazioni massime per recensione G pari a 15, mentre per il secondo pari a 50. Il motivo di questa scelta è dovuto all'ipotesi, successivamente confermata, che per attaccare recensioni di lunghezza maggiore fosse necessario in media un numero più elevato di generazioni. Per l'esperimento è stata utilizzata una CPU con una frequenza di 2.596.995.000 Hz.

4.2 Analisi Tecnica

La percentuale media di successo dell'attacco per le varie combinazioni è risultata essere alta sia per le recensioni brevi (max 500 caratteri) che per quelle lunghe (300-2.000 caratteri). Per il primo insieme è stata raggiunta una percentuale di successo media del 96,18% per le recensioni con Sentiment positivo e del 99,45% per quelle con Sentiment negativo. Entrambi questi valori avrebbero raggiunto probabilmente il 100% se il valore dell'iperparametro G fosse stato più alto. Per l'insieme delle recensioni di lunghezza maggiore è stata invece raggiunta un'efficienza dell'attacco del 100% sia per quelle positive che per quelle negative. L'eventuale diminuzione di G, che era stato impostato a 50, avrebbe prodotto una percentuale di successo inferiore (Figura 4.1).

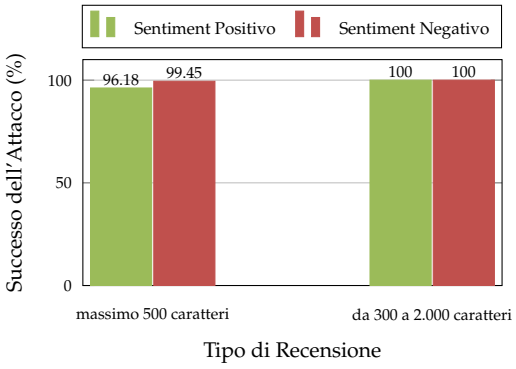
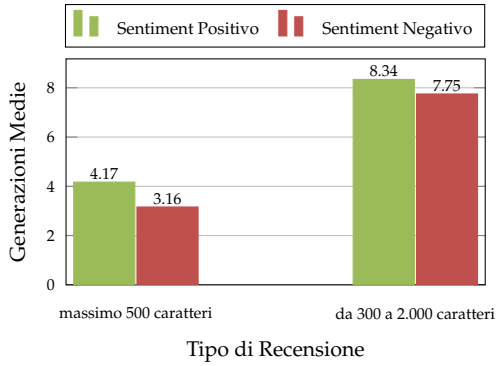


Figura 4.1: Successo medio dell'attacco per tipo di recensione.

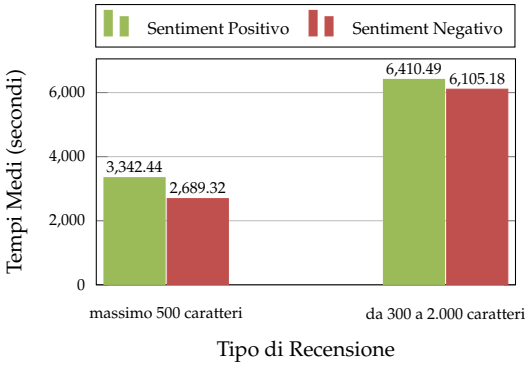
In riferimento al risultato medio dei tempi di computazione e del numero di generazioni necessarie a creare un Adversarial Example considerando le varie combinazioni di iperparametri sono state invece evidenziate due caratteristiche (Figura 4.2):

- Le recensioni più lunghe impiegano in media un numero di generazioni e tempi circa doppi rispetto a quelli delle recensioni brevi. Quelle brevi con Sentiment positivo utilizzano mediamente 4,17 generazioni e 3342,44 secondi per generare un Adversarial Example, mentre quelle lunghe spendono in media 8,34 generazioni e 6410,49 secondi. Le recensioni brevi con Sentiment negativo impiegano mediamente 3,16 generazioni e 2689,32 secondi, mentre quelle lunghe con lo stesso Sentiment 7,75 generazioni e 6105,18 secondi. Si può dedurre che all'aumentare della lunghezza delle recensioni siano necessari tempi d'attacco più lunghi ed un maggior numero di generazioni per produrre Adversarial Example. Questo è probabilmente dovuto al fatto che al fine di valutare il Sentiment di una recensione la DNN di Sentiment Analysis esclude i token successivi ad una certa posizione per motivi computazionali, anche se questi possono comunque essere selezionati dall'algoritmo di attacco per la sostituzione senza che avvenga un'alterazione del Confidence Score. In un contesto Black-Box l'attaccante non è a conoscenza della struttura del modello, e per rendere più verosimile l'attacco non è stato preso in considerazione il comportamento da parte della DNN di classificazione di esclusione di alcuni token.

- Sia per l'insieme di recensioni brevi che per quello di recensioni più lunghe è evidente come l'algoritmo risulti essere più efficiente sia in termini di generazioni che di tempi massimi nel costruire Adversarial Examples da recensioni con Sentiment negativo. Sia le recensioni negative brevi che quelle più lunghe impiegano in media circa una generazione in meno rispetto a quelle positive. Quelle brevi positive sono generalmente attaccate con circa 700 secondi in più rispetto a quelle negative, mentre quelle più lunghe con una maggiorazione di circa 300 secondi.



(a) Generazioni medie per effettuare un attacco per tipo di recensione.

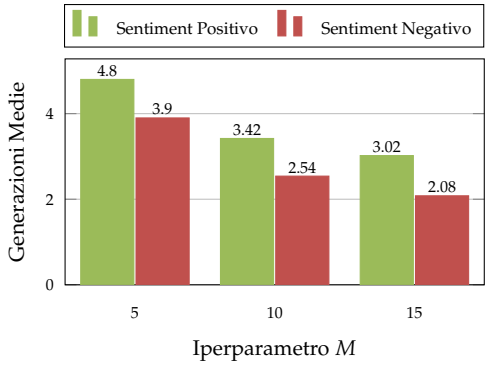


(b) Tempi medi per effettuare un attacco per tipo di recensione.

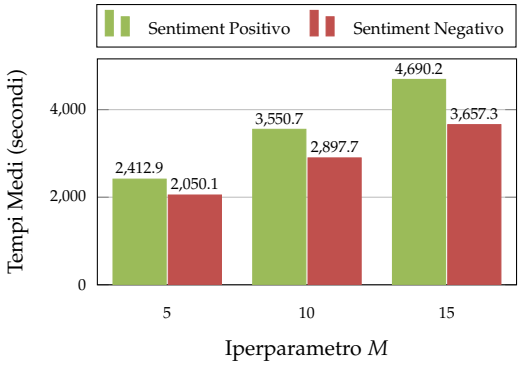
Figura 4.2: Generazioni e tempi medi per effettuare un attacco per tipo di recensione.

Si sono osservati i seguenti comportamenti all'aumentare dei valori degli iperparametri:

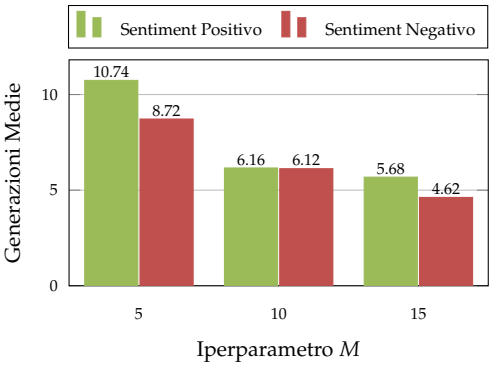
- In riferimento a M è stato osservato come, all'aumentare del numero di parole da modificare ad ogni perturbazione, si evidenzi una diminuzione del numero di generazioni medie necessarie per eseguire un attacco (Figura 4.3). Sia per le recensioni brevi che per quelle più lunghe avviene una diminuzione maggiore quando M varia da 5 a 10 e una diminuzione più lieve per il valore 15. Alla diminuzione del numero delle generazioni si contrappone però un aumento critico dei tempi di elaborazione medi. Infatti, per entrambi gli insiemi di recensioni, si raggiungono valori quasi doppi quando M varia da 5 a 15. La diminuzione del numero di generazioni medio è dovuto al fatto che vengono prodotte più perturbazioni per ogni popolazione ed è quindi più semplice che venga costruito un Adversarial Example in meno iterazioni. L'aumento critico dei tempi di computazione è dovuto al fatto che, con la presenza di più perturbazioni da generare per popolazione, aumentano il numero di operazioni necessarie a valutare l'efficienza dei sinonimi scelti per la sostituzione. Un valore di M troppo alto risulta quindi essere poco conveniente.



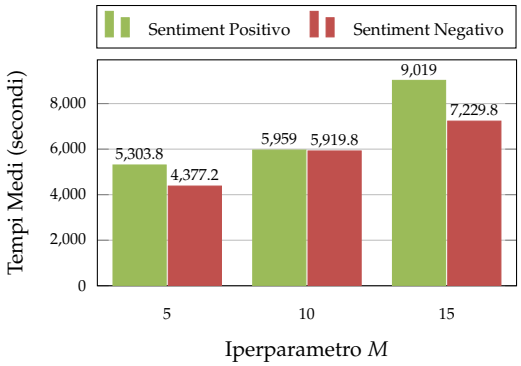
(a) Generazioni medie per attacco al variare di M (recensioni $\leq$ 500 caratteri).



(b) Tempi medi per attacco al variare di M (recensioni $\leq$ 500 caratteri).



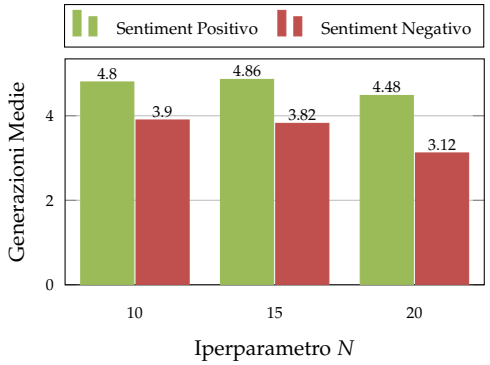
(c) Generazioni medie per attacco al variare di M (recensioni da 300 a 2.000 caratteri).



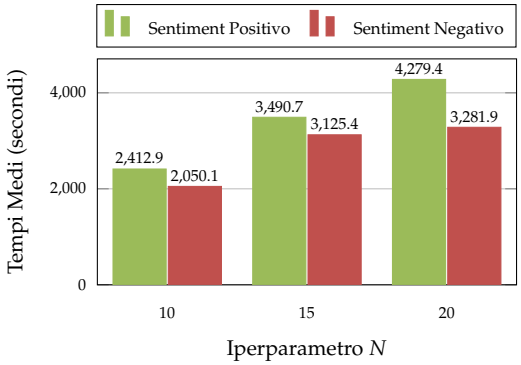
(d) Tempi medi per attacco al variare di M (recensioni da 300 a 2.000 caratteri).

Figura 4.3: Test effettuati per l'iperparametro M.

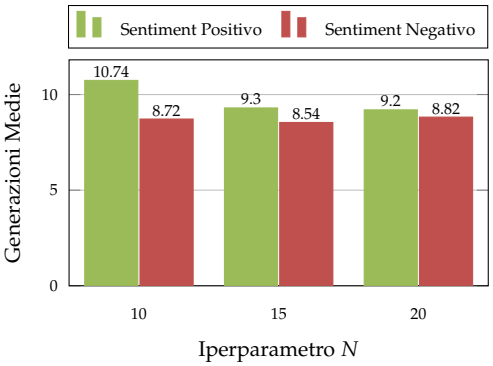
- Un andamento simile si riscontra anche per l'iperparametro N (Figura 4.4). Anche in questo caso avviene, per i due insiemi di recensioni, un aumento quasi doppio dei tempi di computazione quando il numero di migliori sinonimi scelti varia da 10 a 20, ma in genere non si accompagna ad una sostanziale diminuzione del numero di generazioni medie. La crescita dei tempi di computazione è anche in questo caso dovuta all'aumento delle operazioni necessarie per la valutazione delle varie parole candidate a sostitute, ed in particolare a quelle di interrogazione del *Google One Billion Words Language Model*. Anche in questo caso un valore troppo alto per N risulta poco efficiente ai fini dell'attacco.



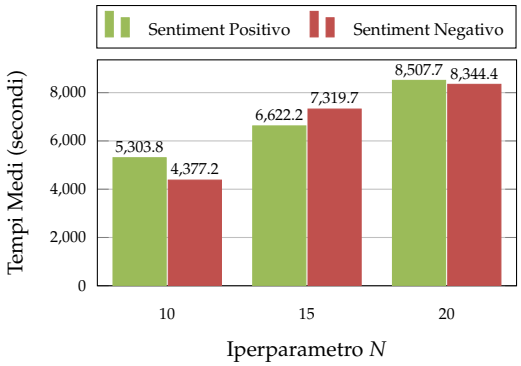
(a) Generazioni medie per attacco al variare di N (recensioni $\leq$ 500 caratteri).



(b) Tempi medi per attacco al variare di N (recensioni $\leq$ 500 caratteri).



(c) Generazioni medie per attacco al variare di N (recensioni da 300 a 2.000 caratteri).

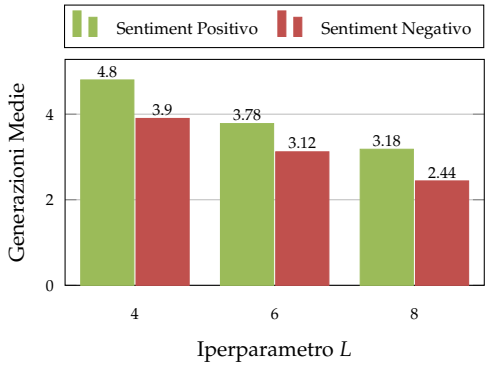


(d) Tempi medi per attacco al variare di N (recensioni da 300 a 2.000 caratteri).

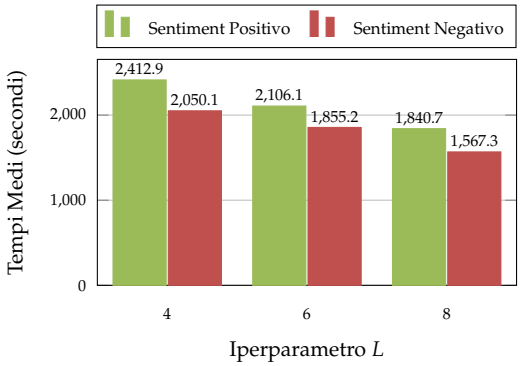
Figura 4.4: Test effettuati per l'iperparametro N.

- All'aumentare di sole poche unità del valore dell'iperparametro L avviene un miglioramento dei tempi impiegati nell'attacco oltre ad una diminuzione delle generazioni medie necessarie per portarlo a termine (Figura 4.5). Per le recensioni brevi aumentando il valore di L da 4 a 8 le generazioni medie diminuiscono di più di un'unità, mentre per quelle più lunghe vi è una diminuzione di 4 unità per le recensioni positive e di circa 2 per quelle negative. Per gli stessi valori si riscontra una diminuzione di circa 500 secondi per le recensioni brevi in genere, di circa 2.000 per quelle più lunghe con Sentiment positivo e di circa 1.000 per quelle con Sentiment negativo. La diminuzione dei tempi di computazione e delle generazioni si verifica perché il numero di operazioni effettuate per ogni popolazione rimane praticamente invariato, mentre vengono valutati un numero maggiore di possibili sostituti tra gli N sinonimi considerati. Questo permette di aumentare la probabilità che

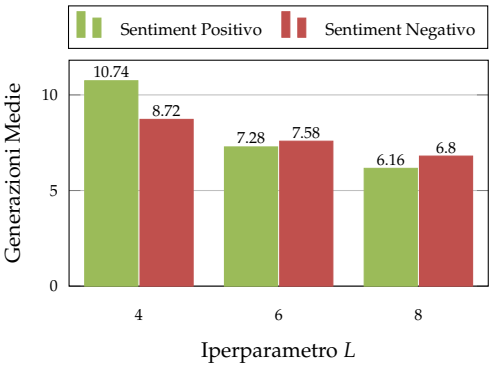
all'interno di un popolazione venga generata una recensione con Confidence Score più vicino al target label. All'aumentare del valore di L aumenta però anche il numero di sinonimi da considerare per la sostituzione che hanno poca probabilità di essere coerenti nel contesto della frase.



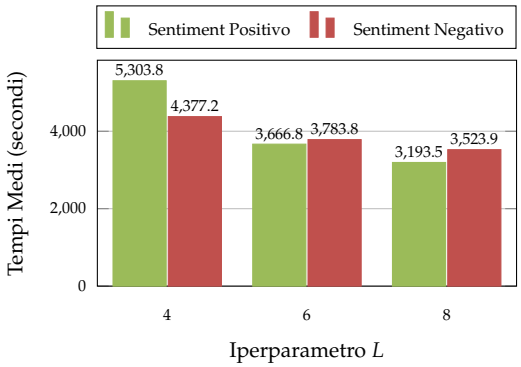
(a) Generazioni medie per attacco al variare di L (recensioni $\leq$ 500 caratteri).



(b) Tempi medi per attacco al variare di L (recensioni $\leq$ 500 caratteri).



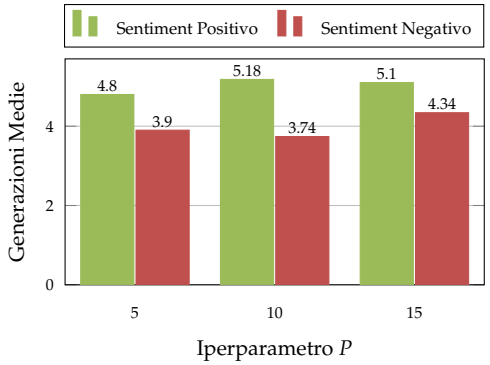
(c) Generazioni medie per attacco al variare di L (recensioni da 300 a 2.000 caratteri).



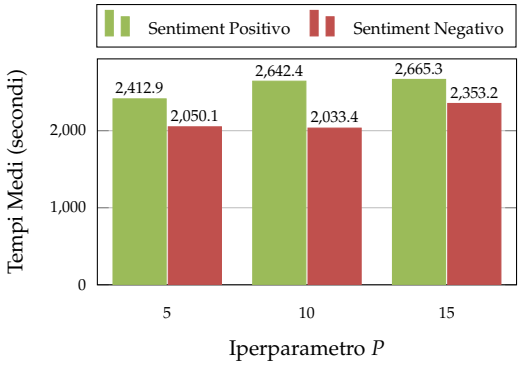
(d) Tempi medi per attacco al variare di L (recensioni da 300 a 2.000 caratteri).

Figura 4.5: Test effettuati per l'iperparametro L.

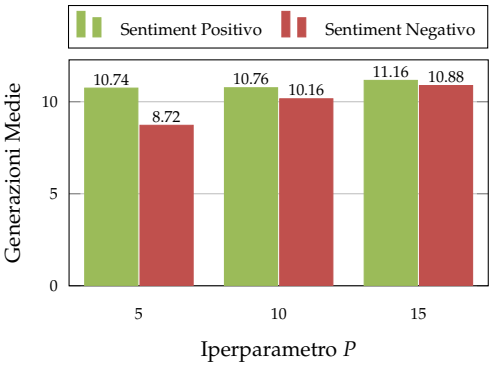
- L'applicazione di variazioni all'iperparamentro P non contribuisce all'individuazione di modifiche comportamentali specifiche per nessuno dei due insiemi di recensioni, né per quelle con Sentiment negativo né per quelle con Sentiment positivo (Figura 4.6).



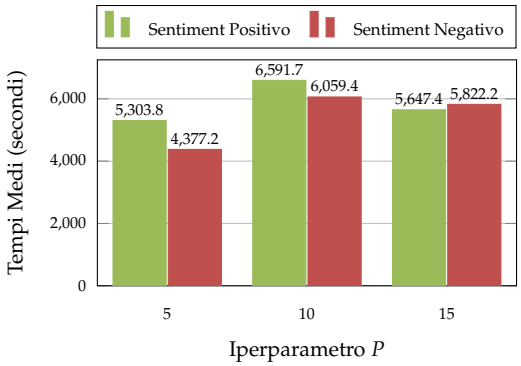
(a) Generazioni medie per attacco al variare di P (recensioni $\leq$ 500 caratteri).



(b) Tempi medi per attacco al variare di P (recensioni $\leq$ 500 caratteri).



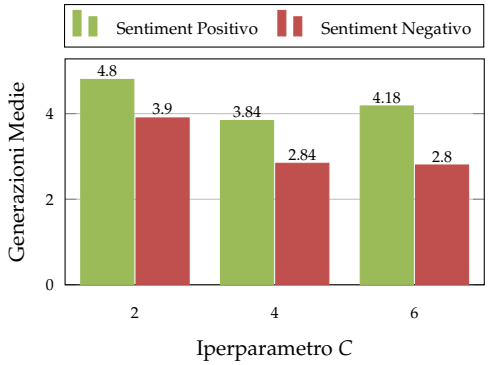
(c) Generazioni medie per attacco al variare di P (recensioni da 300 a 2.000 caratteri).



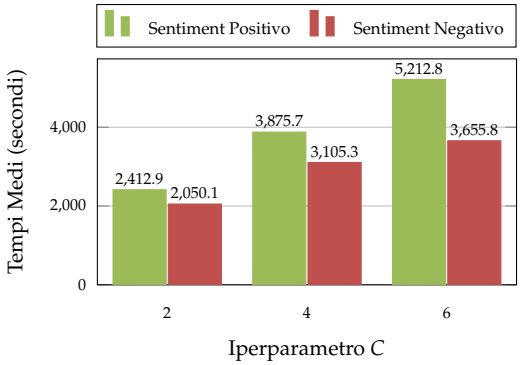
(d) Tempi medi per attacco al variare di P (recensioni da 300 a 2.000 caratteri).

Figura 4.6: Test effettuati per l'iperparametro P.

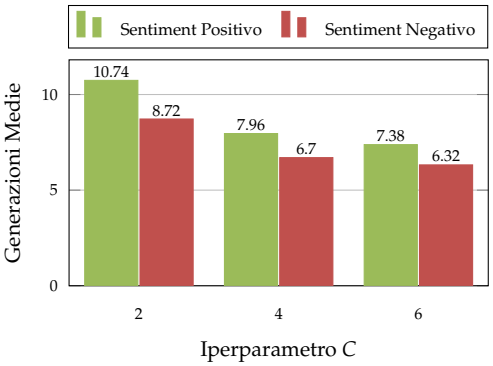
- L'iperparametro C evidenzia un comportamento simile a quello di M ed N (Figura 4.7). Aumentando il numero di figli da perturbare per ogni generazione da 2 a 6 avviene, per le recensioni brevi, una diminuzione di circa una generazione media collegata ad un aumento di circa 3.000 secondi di computazione per le recensioni positive e di 1.500 per quelle negative. Per le recensioni di lunghezza maggiore, sia positive che negative, avviene invece un decremento di circa 3 generazioni ed un aumento dei tempi di circa 3.000 secondi. Anche in questo caso l'aumento dei tempi di computazione è dovuto al fatto che cresce il numero di operazioni necessarie a valutare le varie parole nel contesto delle frasi, in quanto ad ogni figlio considerato vengono applicate M perturbazioni. Pertanto è più appropriato selezionare un valore non troppo elevato per C.



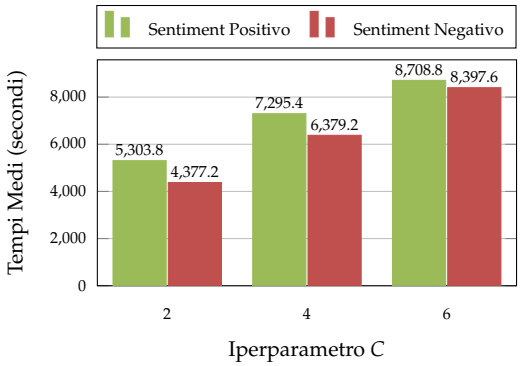
(a) Generazioni medie per attacco al variare di C (recensioni $\leq$ 500 caratteri).



(b) Tempi medi per attacco al variare di C (recensioni $\leq$ 500 caratteri).



(c) Generazioni medie per attacco al variare di C (recensioni da 300 a 2.000 caratteri).



(d) Tempi medi per attacco al variare di C (recensioni da 300 a 2.000 caratteri).

Figura 4.7: Test effettuati per l'iperparametro C.

4.3 Analisi Qualitativa

Nell'esperimento originale i test sull'algoritmo sono stati effettuati con una *GPU Titan X* mentre nell'elaborazione della tesi è stata utilizzata una sola CPU, pertanto i risultati di computazione non sono totalmente comparabili. In ogni caso i tempi di elaborazione dei test descritti nella tesi risultano maggiori rispetto a quelli dell'esperimento originale principalmente a causa dei calcoli effettuati dal *Google One Billion Words Language Model* per valutare la probabilità dei vari sinonimi da sostituire nel contesto della frase. Il motivo della più lunga computazione è dovuto al fatto che ogni parola viene valutata nel contesto dell'intera frase piuttosto che rispetto al vocabolo precedente e quello successivo come avviene nell'esperimento originale. Comunque l'applicazione di tale modifica dovrebbe aver garantito la generazione di Adversarial Examples più efficaci. Non è possibile

fare un paragone diretto con i risultati dell'esperimento originale in quanto non sono disponibili le recensioni avversarie da loro generate. Inoltre non si è potuta effettuare, come nel loro caso, una valutazione delle recensioni risultanti da parte di terzi. Nell'osservazione degli Adversarial Examples ottenuti dal lavoro svolto si evidenzia che la semantica nella maggior parte dei casi rimane inalterata, in quanto le parole sostituite non tendono a modificarla. I casi in cui la semantica viene alterata sono principalmente collegati al fatto che vengono inserite parole non appartenenti alla lingua Inglese, questo può accadere in quanto tali vocaboli sono presenti all'interno dei *Counter-Fitted Word Vectors* utilizzati per generare la matrice di distanza. Si potrebbero filtrare i vettori considerando le sole parole Inglesi prima del loro utilizzo per ovviare al problema. In riferimento alla sintassi degli Adversarial Examples l'algoritmo non risulta sempre efficiente, in quanto alcuni vocaboli rappresentanti nomi sono a volte sostituiti da altri che ne alterano la quantità: capita che nomi al singolare vengano modificati con nomi al plurale e viceversa. A volte si riscontra che la coniugazione di alcuni verbi delle recensioni perturbate non sia corretta. La modifica applicata all'algoritmo che impedisce che una popolazione superi una certa dimensione pare non aver avuto effetti sui risultati. Invece è stata sicuramente efficace l'esclusione delle *Named-Entity* dalle parole da scegliere per la sostituzione in quanto all'interno dei *Counter-Fitted Word Vectors* sono presenti anche nomi propri o entità geopolitiche che trovano esatto corrispettivo nel testo delle recensioni. Un'altra possibile integrazione si potrebbe individuare nella limitazione della probabilità di sostituzione di alcune parole frequenti nelle recensioni del *Test Data Sample*. Ad esempio il vocabolo "movie" è molto frequente e la sua sostituzione non porta ad una variazione eccessiva del valore del Confidence Score risultando quindi la sua modifica solo uno spreco computazionale. Gli Adversarial Examples generati sono efficienti e realistici nella maggior parte dei casi, nonostante le limitazioni descritte. Un qualsiasi lavoro di *NLG*, pur pianificato, attualmente non sarà perfetto per la difficoltà nel riprodurre correttamente il linguaggio umano da parte di un calcolatore. La Tabella 4.1 illustra degli esempi di Adversarial Examples generati da recensioni con sentiment positivo e la Tabella 4.2 ne evidenzia degli altri prodotti da recensioni con sentiment negativo.

Previsione Recensione Originale = **Positiva** (Confidence Score: 0,894)
I think that the movie was really good. [...] Although the director has succeeded in showing the status of women in rural areas and how they suffer at the hands of male-dominated culture, he has neglected [...]
Previsione Recensione Perturbata = **Negativa** (Confidence Score: 0,486)
I think that the movie was actually decent. [...] Although the director has succeeded in showing the status of women in agricultural areas and how they suffer at the hands of male-dominated civilization, he has neglected [...]

Previsione Recensione Originale = **Positiva** (Confidence Score: 0,992)
The acrobatics mixed with haunting music, make one spectacular show. The costumes are vibrant and the performances will just boggle your mind! Simply amazing!
Previsione Recensione Perturbata = **Negativa** (Confidence Score: 0,47)
The acrobatics mixed with mournful music, render one noteworthy show. The costumes are dynamic and the depictions will just boggle your mind! Simply staggering!

Previsione Recensione Originale = **Positiva** (Confidence Score: 0,965)
This film is a summary of Visconti's obssessions: the decadence of nobility, death, aesthetic search, [...] all its beauty is in the music and the images as well as in the story. This movie is brave, deeply personal and intelligent. [...]
Previsione Recensione Perturbata = **Negativa** (Confidence Score: 0,431)
This film is a summary of Visconti's obssessions: the slump of nobility, dead, aesthetic search, [...] all its beauty is in the music and the photograph as well as in the story. This film is bold, severely personal and intelligent. [...]

Tabella 4.1: Esempi di Adversarial Example generati da recensioni positive.

Previsione Recensione Originale = **Negativa** (Confidence Score: 0,129)
Low budget horror about an evil force. Hard to believe in this day and age, but way back when this stuff actually used to get theatrical release! [...] Shouldn't be too hard to avoid this one; who's ever heard of it?
Previsione Recensione Perturbata = **Positiva** (Confidence Score: 0,576)
Low budget horror about an evil force. Hard to believe in this day and age, but way back when this stuff truly used to get theatrical release! [...] Shouldn't be too hard to avert this one; who's ever heard of it?

Previsione Recensione Originale = **Negativa** (Confidence Score: 0,079)
This is a silly spoof of private eye thrillers [...] At times silliness becomes obnoxious. This is not Cain at his best. [...] Notable support from Lizabeth Scott, Lionel Stander and the comely Nadia Cassini. Not easy to watch.
Previsione Recensione Perturbata = **Positiva** (Confidence Score: 0,646)
This is a silly simulation of private eye thrillers [...] At times silliness becomes abominable. This is not Cain at his best. [...] Notable assist from Lizabeth Scott, Lionel Stander and the comely Nadia Cassini. Not easy to watch.

Previsione Recensione Originale = **Negativa** (Confidence Score: 0,001)
[...] Jeanine Garofalo is supposed to be the "ugly duckling" [...] would this movie work if the "ugly duckling" was really unattractive? In my opinion, despite the message that it wants to convey, this movie is simply ridiculous.
Previsione Recensione Perturbata = **Positiva** (Confidence Score: 0,649)
[...] Jeanine Garofalo is alleged to be the "nasty duckling" [...] would this movie work if the "ugly duckling" was really unsavory? In my views , despite the message that it wishes to convey, this cinematic is simply daft.

Tabella 4.2: Esempi di Adversarial Example generati da recensioni negative.

Conclusioni

L'*NLP* è un campo dell'*IA* di difficile applicazione pratica, in quanto le lingue naturali sono strutture particolarmente complesse. Presentano infatti un ampio numero di regole, sono in continua evoluzione e hanno caratteristiche di ambiguità. Un singolo concetto può infatti essere espresso in maniera differente da persone distinte e un vocabolo può assumere significati diversi in base al contesto in cui si trova. L'espressione linguistica presenta inoltre carattere soggettivo e può venire recepita ed esposta in maniera diversa a seconda dell'interlocutore. Nonostante le difficoltà evidenziate, la realizzazione dell'esperimento originale e di quello riportato nella tesi manifestano la possibilità di generare Adversarial Examples in grado di attaccare un modello di classificazione protetto da un sistema Black-Box mantenendo quasi integralmente la semantica e la sintassi delle recensioni originali. Questa tecnica d'attacco applicata al campo dell'NLP è comunque abbastanza recente: l'esperimento originale stesso è stato pubblicato nel 2018. Le implementazioni apportate nel lavoro della tesi, seppur con più alti tempi di computazione rispetto all'esperimento originale, hanno apportato l'affinazione del modello di attacco attraverso l'applicazione di alcune modifiche. In futuro sicuramente le procedure di NLP e le tecniche di attacco non potranno che diventare più efficaci.

Sviluppi futuri

Per migliorare la qualità degli Adversarial Examples prodotti nell'esperimento sarebbe opportuno applicare le tecniche indicate nel capitolo precedente. Filtrare quindi i *Counter-Fitted Word Vectors* eliminando i vocaboli non appartenenti alla lingua Inglese comporterebbe la diminuzione dei casi in cui si presentano discrepanze semantiche tra le recensioni originali e quelle perturbate. Inoltre limitare

la probabilità di sostituzione dei token che si ripetono frequentemente all'interno delle recensioni, in quanto ininfluenti alla modifica del Confidence Score, dovrebbe favorire i tempi di computazione. Infine può essere interessante valutare le tecniche utilizzabili per allenare un modello di classificazione del Sentiment Analysis delle recensioni al fine di garantire una robustezza maggiore agli Adversarial Examples generati dalle recensioni correttamente classificate.

Bibliografia

[1] D. Ciregan, U. Meier, and J. Schmidhuber. "Multi-column deep neural networks for image classification". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3642–3649. DOI: 10.1109/CVPR.2012.6248110.

[2] Dale Schuurmans and Martin A Zinkevich. "Deep Learning Games". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016, pp. 1678–1686. URL: https://proceedings.neurips.cc/paper/2016/file/c4015b7f368e6b4871809f49debe0579-Paper.pdf.

[3] Ankit Kumar et al. "Ask Me Anything: Dynamic Memory Networks for Natural Language Processing". In: *CoRR* abs/1506.07285 (2015). arXiv: 1506.07285. URL: http://arxiv.org/abs/1506.07285.

[4] Battista Biggio and Fabio Roli. "Wild patterns: Ten years after the rise of adversarial machine learning". In: *Pattern Recognition* 84 (Dec. 2018), pp. 317–331. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2018.07.023. URL: http://dx.doi.org/10.1016/j.patcog.2018.07.023.

[5] Moustafa Alzantot et al. "Generating Natural Language Adversarial Examples". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 2890–2896. DOI: 10.18653/v1/D18-1316. URL: https://www.aclweb.org/anthology/D18-1316.

[6] Andrew L. Maas et al. "Learning Word Vectors for Sentiment Analysis". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Associ-

ation for Computational Linguistics, June 2011, pp. 142–150. URL: http://www.aclweb.org/anthology/P11-1015.

[7] Peifeng Li, Jinhui Li, and Qiaoming Zhu. "An Approach to Hierarchical Email Categorization Based on ME". In: *Proceedings of the 12th International Conference on Applications of Natural Language to Information Systems*. NLDB'07. Paris, France: Springer-Verlag, 2007, pp. 25–34. ISBN: 3540733507.

[8] Mathilde Caron et al. "Deep Clustering for Unsupervised Learning of Visual Features". In: *CoRR* abs/1807.05520 (2018). arXiv: 1807.05520. URL: http://arxiv.org/abs/1807.05520.

[9] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://www.aclweb.org/anthology/D14-1162.

[10] christofhenkel. *How To: Preprocessing for GloVe Part1: EDA*. URL: https://www.kaggle.com/christofhenkel/how-to-preprocessing-for-glove-part1-eda.

[11] Tan Thongtan and Tanasanee Phienthrakul. "Sentiment Classification Using Document Embeddings Trained with Cosine Similarity". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 407–414. DOI: 10.18653/v1/P19-2057. URL: https://www.aclweb.org/anthology/P19-2057.

[12] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[13] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[14] Leonard Richardson. "Beautiful soup documentation". In: *April* (2007).

[15] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[16] Christian Szegedy et al. *Intriguing properties of neural networks*. 2014. arXiv: 1312.6199.

[17] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. arXiv: 1412.6572.

[18] Nikola Mrkšić et al. *Counter-fitting Word Vectors to Linguistic Constraints*. 2016. arXiv: 1603.00892.

[19] Ciprian Chelba et al. *One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling*. 2014. arXiv: 1312.3005.

[20] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.