Advanced NLP Exercise 1

מגישה: הגר צרפתי 322710229

שאלות פתוחות

שאלה 1

1. EntQA (Entity Linking as QA)
dataset שבו המודל מקבל שאלה והקשר, ונדרש לקשר ישות מתוך השאלה לישות מסוימת באונטולוגיה (למשל בוויקיפדיה).
הdataset מתמקד בתכונה אינטרינזית של entity linking - כלומר, הבנת הקשר בין מונחים לשוניים לבין ייצוגים חוץ-טקסטואליים.

2. BoolQ (Boolean Questions)
שאלות כן/לא על גבי טקסטים, כל דוגמה מורכבת מהשלשה- (שאלה,פסקה,תשובה). המודל נדרש לזהות האם השאלה נענית בטקסט - מה שדורש הסקה, טיפול בשלילה והבנת ניואנסים.
זו משימה אינטרינזית שכן היא מודדת הבנה לוגית ולשונית עמוקה של טקסט.

3. DROP (Discrete Reasoning Over Paragraphs)
Dataset שמיועד להעריך הבנת הנקרא ברמה יותר גבוהה משאר הdataset שקיימים (לפחות בהשוואה למה שהיה קיים בזמן שיצרו אותו), ולגרום למודל לקרוא פסקאות שלמות. על מנת לענות על השאלות בdataset צריך לדעת להתמודד עם הפניות (reference) לעיתים למספר מקומות שונים בטקסט, ולבצע פעולות דיסקרטיות מעליהן (למשל חיבור\ספירה\מיון..) פעולות הדורשות הבנה עמוקה של תוכן הפסקאות.

שאלה 2

סעיף a

<u>שיטה:</u> Chain Of Thought
תיאור: מבקשים מהמודל להסביר את reasoning שלב אחר שלב (למשל בבעיות שדורשות הסקה לוגיות).
ניתן להשתמש גם בדוגמאות (ICL) למשל- הinput למודל הוא דוגמה לשאלה ודרך לפתור אותה (בנוסף לפתרון יש הסבר מפורט איך הגיעו אליו) ואז השאלה שהמודל צריך לפתור לבד.
המודל מחזיר את התשובה בצורה greedy.
יתרונות: מאפשר יכולת חשיבה מורכבת באמצעות שלבי חשיבה ביניים. משפר משמעותית הבנה לוגית ומשימות מורכבות הדורשות חשיבה לפני מתן תגובה, ומשפר את הדיוק.
צווארי בקבוק חישוביים: הפלט ארוך יותר, לכן דורש יותר כוח חישוב (יחסית לפעולת הסקה שלא עושה זאת) וזמן עיבוד נוסף, ובנוסף דורש יותר זיכרון כי צריך לזכור את השלבי ביניים.
האם ניתן למקבל את השיטה: לא

<u>שיטה:</u> Self-Consistency
תיאור: הinput של המודל הוא אותו דבר כמו בCOT, ההבדל הוא שכאן המודל בודק מספר אפשרויות של reasoning path ועושה עליהם אגריגציה (aggregate) ולפי זה מחזיר תשובה לשאלה (למשל לפי הרוב).
יתרונות: מעלה דיוק ומפחית רעש ושונות בתשובות, משפר את הביצועים
צווארי בקבוק חישוביים: דורש הרבה הרצות, וכל הרצה היא גם יקרה כי עושים reasoning path
האם ניתן למקבל את השיטה: כן (ניתן למקבל את ההרצות ואז עושים את האגריגציה)

שיטה: Least-to-Most
תיאור: במקום לתת למודל שאלה קשה ומורכבת אנחנו מפרקים אותה לשאלות עזר פשוטות יותר, לפי סדר לוגי מהקל אל הקשה, כך שהמודל יכול לענות עליהן שלב אחר שלב, כל פעם מוסיפים לinput את השאלות והתשובות שהוא ענה עליהן עד עכשיו ואז מוסיפים שאלת ביניים נוספת, עד שהמודל פותר את כל הבעיה.
יתרונות: מקל על reasoning מורכב, מפחיד שגיאות הנובעות מעומס מידע, מאפשר שליטה בתהליך החשיבה של המודל, מתאים לשאלות עם מספר שלבים. (השיטה נועדה להוביל את המודל בהדרגתיות לפתרון).
צווארי בקבוק חישוביים: נדרשות הרצות רבות של המודל (כל פעם מריצים שאלת ביניים), דורש זמן ריצה ארוך יותר, נדרש זיכרון עבור השאלות והתשובות שנשאלו, דורש תכנון ידני של פרומפט השאלה
האם ניתן למקבל את השיטה: לא

שיטה: Self-Ask
תיאור: המודל לא מנסה לענות מיד על שאלה מורכבת, במקום זאת המודל שואל את עצמו שאלות follow-up כדי לפרק את שאלה מורכבת לשאלות ביניים, ובסוף נותן תשובה סופית.
יתרונות: משפר את היכולת של המודל לבצע reasoning מורכבות שדורשות מספר שלבים של reasoning כדי להגיע לתשובה, ולחבר (compose) ידע כדי לענות על שאלות מורכבות.
צווארי בקבוק חישוביים: יקר חישובית (זמן כוח חישוב וזיכרון). דורש שהמודל ידע לנסח לעצמו את השאלות, זה לא תמיד מדויק. לא מתאים לשאלות פשוטות.
האם ניתן למקבל את השיטה: אפשר למקבל רק אם יש שאלות ביניים שלא תלויות זו בזו, לרוב זה לא המצב ויש תלות סדרתית בין השאלות ביניים ואז לא יהיה ניתן למקבל.

שיטה: Verifiers
תיאור: כלים שיודעים לקבל output ולהגיד אם הוא נכון או לא. מייצרים כמה outputים ובמקום לקחת את הרוב אז לוקחים את הכי טוב מבניהם או את הרוב שעברו ואריפיקציה.
יתרונות: אם יש משימה שאי אפשר להוציא output לפי מה שהרוב חושבים (למשל קוד\המשך לסיפור) אז אפשר להשתמש בזה כדי לוודא שמחזירים תשובה תקינה. מבטיח תשובות איכותיות יותר (כמובן תלוי באיכות הvarifiers)
צווארי בקבוק חישוביים: צריך להריץ את המודל כמה פעמים ואז גם לבדוק כל output, דורש יותר זמן וכוח חישוב וזיכרון. צריך לדעת איך לעשות אגריגציה של הoutputים השונים. צריך לכל סוג משימה verifier מתאים.
האם ניתן למקבל את השיטה: כן, ניתן למקבל את ההרצות השונות ואת הואריפיקציה ואז לעשות אגריגציה על מה שקיבלנו.

שיטה: Planning
תיאור: שיטה בה המודל לא מנסה ישר לפתור את השאלה, אלא קודם מתכנן את שלבי הפעולה שלו ורק לאחר מכן מבצע את השלבים כדי להחזיר תשובה.
יתרונות: מפריד בין שלבי הreasoning לביצוע. מקל על טעויות שנובעות מהסקה מהירה מידי. מגדיל את הדיוק, ניתן להבין "איך המודל חושב" ולמה הוא הגיע לתוצאה שקיבלנו, משפר ביצועים.
צווארי בקבוק חישוביים: דורש יותר זמן וכוח חישוב כי התשובה יותר ארוכה, זיכרון (לזכור את השלבים שתכנן ואת התוצאות שהוא קיבל בשלבים השונים). דורש יכולת לפרק את המשימה לתתי משימות.
האם ניתן למקבל את השיטה: תלוי בתוכנית, אם הפעולות שהמודל מתכנן לא תלויות אחת בשנייה ניתן למקבל אותן, אחרת לא ניתן.

שיטה: backtracking
תיאור: תוך כדי פתרון השאלה אם המודל מזהה שהוא עשה משהו לא טוב, הוא חוזר אחורה לשלבים קודמים ומנסה פתרון אחר.

יתרונות: מאפשר למודל לתקן את עצמו בלי התערבות ולכן מפחית טעויות בתשובה הסופית. מדמה תהליך אנושי של ניסוי וטעיה.
מגדיל את הדיוק, משפר ביצועים, עוזר (גם אם לא באופן מוחלט) להתמודד עם הבעיה שלעיתים המודל מנחש תשובה שהוא לא יודע מה הפתרון, כי הוא מאפשר לעצמו לנסות מספר כיוונים ואם הוא רואה שהוא הלך בכיוון לא טוב הוא עוזר וחוזר אחורה.
צווארי בקבוק חישוביים: דורש יותר זיכרון על מנת שתהיה האפשרות "לחזור אחורה" שטועים וגם בגלל שהמודל בודק את עצמו תוך כדי וגם כי הפלט יותר ארוך, דורש כוח עיבוד נוסף על מנת שהמודל יבדוק את עצמו תוך כדי, וזה לוקח יותר זמן. (זו שיטה סדרתית שאינה ניתנת למקביליות, כי כל ניסיון תלוי בניסיון הקודם.)
האם ניתן למקבל את השיטה: לא

<u>שיטה:</u> self-evaluation
תיאור: המודל בוחן את התשובה של עצמו לפי קריטריונים פנימיים, ויכול להעריך את הביצועים של עצמו.
יתרונות: מגביר את אמינות התשובות. מאפשר לסנן תשובות לא נכונות.
צווארי בקבוק חישוביים: דורש חישוב נוסף אחרי התשובה לכן דורש עוד כוח חישוב, זמן וזיכרון.
המודל עשוי לטעות בהערכת התשובה של עצמו- הערכה עצמית היא לא בהכרח מדויקת וקשה להגדיר קריטריונים אחידים.
האם ניתן למקבל את השיטה: לא

<u>שיטה:</u> Prompt Ensembling
תיאור: מריצים את המודל עם ניסוחים שונים של אותה השאלה, ואז עושים אגריגציה על הoutputים השונים.
יתרונות: מפחית רגישות לניסוח ומעלה את הדיוק.
צווארי בקבוק חישוביים: דורש יותר כוח עיבוד וזיכרון (כי צריך להריץ את המודל מספר פעמים), דורש לייצר מספר פרומפטים שונים עבור אותה המשימה.
האם ניתן למקבל את השיטה: כן

סעיף b

במקרה של משימה מדעית מורכבת הדורשת הסקה מרובת שלבים, ויש ברשותי GPU יחיד עם זיכרון גדול, אבחר להשתמש בשיטת Chain of Thought (CoT).
הסיבה לכך היא ששיטה זו מאפשרת למודל לפרק את המשימה לשלבים לוגיים פנימיים, ובכך לשפר את איכות ההסקה - במיוחד במשימות שדורשות reasoning מורכב כמו שאלות מדעיות, מתמטיות או ניתוח לוגי. היות ששיטה זו דורשת הרצה אחת בלבד של המודל לכל שאלה והיא לא ניתנת למקבול היא מתאימה למקרה בו יש GPU יחיד. בנוסף, מאחר שהתשובות כוללות שלבי ביניים ארוכים, השיטה דורשת זיכרון גדול - מה שהופך את סביבת העבודה הזו (GPU עם הרבה זיכרון) לאידיאלית ליישום השיטה.

חלק 2

קישור לgit:

https://github.cs.huji.ac.il/hagar/ANLP-EX1
https://github.com/hagarzarfati/ANLP-EX1

- הקונפיגורציה שהשיגה accuracy הכי טוב על סט הvalidation גם השיגה accuracy הכי טוב על סט הtest.

	Number of epochs	Learning rate	Batch size	validation accuracy	test accuracy
1	2	0.0001	64	0.8480	0.8290
2	2	1e-05	64	0.7181	0.7014
3	4	0.1	32	0.6838	0.6649

- הקונפיגורציה שהכי הצליחה היא קונפיגורציה 1 (לפי הטבלה), ושהכי פחות הצליחה היא קונפיגורציה 3.
- המודל עם הביצועים הנמוכים תמיד חוזה את הlabel להיות 1 ללא קשר לדוגמה, היות שבvalidation set יש יותר labelים שערכם 1 מאשר 0, המודל למד שתמיד עדיף לו לחזות את הערך 1 ללא קשר למשפטים שהוא קיבל, לעומת זאת המודל שמצליח הכי טוב משתמש במשפטים בפרדיקציה שלו והוא חוזה גם 0 וגם 1.
- ניתן לראות שזה מה שקורה מהנתונים הבאים:
  - Total validation samples: 408
    Label = 1 (Paraphrase): 279
    Label = 0 (Not Paraphrase): 129

  - Total examples where Best model was right and Worst model was wrong: 89

המודל הכי טוב חזה את הלייבל 0: 111 פעמים, את הלייבל 1: 297 פעמים
המודל הכי גרוע חזה את הלייבל 0: 0 פעמים, את הלייבל 1: 408 פעמים

המודל הכי טוב צדק 346 פעמים, טעה 62 פעמים
המודל הכי גרוע צדק 279 פעמים, טעה 129 פעמים.

הערה: בנוסף לtrain_loss.png , אני מוסיפה פה את הplot של הtrain loss רק על שני המודלים הראשונים (לפי המספור בטבלה), בגלל שיש הבדל גם במספר הצעדים וגם בטווח הערכים של הloss כאשר מציגים אותם עם המודל השלישי לא רואים אותם טוב בגרף: