**הבנת שפה טבעית**

**אביב תשפ"ג**

מודלי N-grams

נקודת המפנה הראשונה בתחום ה-NLP הייתה שימוש במודלים סטטיסטיים לטובת עיבוד השפה. כאשר משתמשים בהסתברות ניתן לענות על שאלות כגון: עד כמה סביר ניסוח מסוים של משפט, בהינתן התחלה של משפט מהי ההסתברות למילה הבאה וכדו'. המטרה של **מודל שפה** היא בניית מודל הסתברותי של שפה טבעית. שימושים מידיים שניתן לעשות למודל כזה הם למשל השלמת מילים בזמן הקלדה, תיקון שגיאות כתיב או שגיאות תחביר וכדו'. עם זאת, מודלים כאלה כיום מהווים בסיס לכמעט כל הבעיות בעיבוד שפה טבעית. כעת נתייחס לשאלה כיצד ניתן ללמוד מודל כזה של הסתברויות השפה.

ראשית נגדיר מהו token. Token הוא מילה או חלק מילה. הגדרה של "מילון ה-token-ים" נעשה פעם אחת, ולאחר מכן ניתן לבצע "טוקניזציה" – כלומר חלוקה של רצפים (משפטים) ל-tokens. ישנם כמה אפשרויות לבניית מילון ה-tokens, למשל ניתן להגדיר כל תו כ-token, כל מילה, חלקי מילים. החלוקה נקבעת לפי המשימה שאותה רוצים לפתור.

מודל השפה הפשוט ביותר מבוסס N-gram. N-gram הוא רצץ של N tokens. ספציפית, unigram הוא רצף של token אחד, bigram – שתי tokens ו-trigram שלוש tokens. מודל unigram ממדל הסתברות של כל מילה בנפרד, מודל bigram בודק את ההסתברות של כל זוגות המילים במשפט וכן הלאה. ככל ש-N גדול יותר המודל יכול למדל בצורה טובה יותר את השפה כרצף tokens. מצד שני, ככל ש-N גדול יותר, כל רצף של N tokens הוא נדיר יותר ולכן קשה לדעת מהי ההסתברות של אותו רצף. לדוגמה, בהינתן התחלה של משפט: "can you give me a cup of" – אם נרצה שמודל השפה יחזה מהי המילה הבאה, במודל unigram המודל פשוט יתן את המילה הנפוצה ביותר בקורפוס, למרות שהיא לא קשורה למילים הקודמות, מודל bigram יבדוק את כל זוגות המילים שהמילה הראשונה היא of: (of, \_), מודל trigram יבדוק את כל שלישיות המילים המתחילות ב-cup of: (cup, of,\_) ויבחר את השלישיה המופיעה הכי הרבה.

פורמלית יותר, במודל bigram אנו מעוניינים לדעת מהי המילה הבאה בהינתן המילה הקודמת:   
כאשר היא מספר הפעמים שבהם הזוג מופיע בקורפוס, ו- הוא מספר הפעמים ש- מופיע. במילים אחרות הוא אחוז הפעמים שבהם הוא ה-token שמופיע אחרי .

דבר נוסף שניתן לעשות עם מודל שפה הוא לחשב את **ההסתברות של משפט מסויים**, כלומר כמה סביר שיהיה בקורפוס משפט כזה. את ההסתברות ניתן לחשב באמצעות הנוסחה:  
ההנחה מאחורי שימוש ב-N-gram היא שהמידע על ההסטוריה שלפני N tokens לא רלוונטית, למשל עבור bigram:

בדומה לכך, ניתן להגדיר עבור מודל trigram:

*וכן עבור N-gram כללי:*

במקרה כזה הסתברות למילה מסוימת תלויה ב- המילים שלפניה.

שיטה זו של N-grams מבוססת על maximum likelihood. הסיבה לך היא ש-ML מחפשת את ההתפלגות שתמקסם את ההסתברות לקבל את הדגימות שדגמנו. בהינתן דגימות, ההתפלגות שעבורה ההסתברות לקבל את אותן דגימות מקסימלית, היא ההתפלגות הנ"ל (כלומר מספר הדגימות שהתקבלו בפועל חלקי כל הדגימות). נראה הוכחה ספציפית ל-bigram model כי המודל שהוצג הוא אכן פתרון ה-maximum likelihood. נניח משפט באורך N ומילון באורך n מילים . לצורך ההוכחה לא נשתמש ב-token מיוחד לתחילת המשפט, אלא נחשב בנפרד את ההסתברות של המילה הראשונה ושל שאר המשפט. ניתן לבטא את ההסתברות בצורה הבאה.

כאשר הוא אינדיקטור אם היא המילה הראשונה במשפט, ו- הוא מונה (counter) לכמות הפעמים ש- מופיע במשפט.

*המטרה היא למצוא את המקסימום של ההתפלגות תחת האילוץ:*

*ניתן להשתמש בכופלי לגראנז':*

*נגזור את הלגראנז'יאן לפי כל אחד מהמשתנים:*

*בנוסף, קיים האילוץ:*

*כפי שהסקנו לעיל:*

*כלומר, מודל ה-bigram הינו maximum likelihood. (מש"ל וכו'). ניתן לגזור זאת גם למקרה הכללי של N-gram.*

*כעת נדון בשאלה כיצד ניתן לבצע אבלואציה של מודל שפה. קיימות שתי גישות, אבלואציה אינטרינזית ואקסטרינזית. האבלואציה האקסטרינזית מתבססת על אבלואציה של מטלות (tasks) המבוססות על מודל השפה. למשל, פיתוח מודל question answering או של תרגום (machine translation) המבוסס על מודל השפה. בצורה כזאת, ניתן לכמת את הצלחת מודל השפה במטלות אלה, ובעזרתן להסיק את טיב מודל השפה. הייתרון בגישה זו הוא שאם המטרה הסופית של המפתח היא להשתמש במודל השפה למטלה זו, אבלואציה כזאת משקפת בצורה טובה לאילו ביצועים ניתן לצפות. החיסרון העיקרי בגישה זו הוא שהבדיקה אינה ישירה אלא עקיפה.*

*הגישה האינטרינזית מנסה לתת מדד כמותי לטיב המודל בצורה ישירה באמצעות ההסתברויות של המודל. השיטה הנפוצה ביותר היא ה-perplexity (PP). בשיטה זו, המפתח שומר test set עליו הוא לא מאמן את מודל השפה. לאחר האימון מתבצעת בדיקה של ההסתברויות שמודל השפה נותן למשפטים מתוך ה-test set. ההנחה היא שעבור מודל שפה טוב ההסתברויות תהינה גבוהות. ספציפית, מדד ה-PP הינו ביחס הפוך להסתברות שהמודל נותן:*

*למשל, עבור הנחת bigram מתקבל:*

*ככל שמדד ה-perplexity נמוך, כך ההסתברות למשפט גבוהה, מה שמעיד על מודל שפה איכותי יותר. הייתרון בשיטה זו הוא שניתן לבדוק את טיב המודל בצורה "אינטרינזית" בלי להסתמך על מטלות עקיפות אחרות. חיסרון בשיטה זו הוא שלא תמיד הוא משקף כמה המודל יהיה טוב במטלות אחרות. חיסרון נוסף בשיטה זו הוא שבמקרה ש-N-gram מסויים לא מופיע ב-train set ההסתברות שלו תהיה שווה לאפס, וה-perplexity יהיה אינסופי. פיתרון אפשרי לבעיה זו הוא שבמקרה שבו N-gram לא קיים נתבסס על ה-(N-1)-gram. למשל, במקרה ששלישיית מילים לא נמצאת ב-train set אך נמצאת ב-test set ניתן לחלק אותה לשני זוגות מילים. אם גם הזוגות לא נמצאים ניתן לחלק ל-unigrams.*

*לסיכום, מודל שפה ממדל את ההסתברויות של שפה טבעית, הן הסתברות של משפט שלם והן הסתברות של המילה הבאה במשפט. מודל N-gram מניח תלות סטטיסטית של מילה במשפט רק ב-N מילים שלפניה. תחת הנחה זו, ניתן למצוא משערך maximum likelihood על בסיס ספירה של N-grams הקיימים בקורפוס. שיטה זו היא בסיסית ופשוטה אך יכולה לתת ביצועים טובים במשימות פשוטות.*

מודל שפה מבוסס רשת נוירונים

*הערה: בשיעור המרצה העמיק במבוא לרשתות נוירונים (פרספטרון, פונקציות אקטיבציה, ועוד). לא נסקור מבוא זה בסיכום.*

*הרשת הפשוטה ביותר בה ניתן להשתמש כמודל שפה היא feedforward network. רשת כזאת תקבל בכניסה ייצוג של N מילים (בדומה ל-N-gram) ותחזה את המילה הבאה. הייתרון של מודל זה על פני N-gram הוא שהעיבוד נעשה על וקטורי embedding של מילים, ולא המילים עצמן. דבר זה מאפשר הכללה יותר טובה, למשל המודל יוכל לחזות בצורה טובה גם רצפים שלא הופיעו בקורפוס האימון, אם היו קיימים רצפים דומים ב-embedding שלהם. למשל, במשפט הבא:*

*I forgot to make sure that the dog gets…*

*המילה "fed" צריכה להיות בעלת הסתברות גבוהה. במודל שפה מבוסס N-grams, אם הרצף לא היה קיים (למשל ב-trigram – (dog gets fed)) המודל לא ידע להשלים את המשפט. לעומת זאת, במודל מבוסס רשת אשר מתבסס על embeddings, במידה והיה קיים בקורפוס המשפט הבא:*

*I forgot to make sure that the cat gets fed.*

*המודל ידע להשלים את המשפט בהסתברות גבוהה, מכיוון שה-embeddings של dog ושל cat קרובים. הרעיון ב-embeddings הוא שניתן לייצג מילים במרחב וקטורי ולהגדיר משמעות של מילה באמצעות ההתפלגות הסטטיסטית של השפה במרחב זה. למשל מילים בעלות משמעות סמנטית או תחבירית דומה יהיו קרובות במרחב זה. רעיון זה מופיע עוד בחיבור של Osgood et al., 1957 אשר מייצג משמעות של מילה כנקודה במרחב. בפרסום המקורי המחבר התייחס לשלושה ממדים של משמעות:*

* ***Valence*** *– עד כמה המילה "נעימה"*
* ***Arousal*** *– כמה "רגש" יש במילה*
* ***Dominance*** *– כמה "שליטה" יש במילה (ציווי)*

עם זאת, כיום משתמשים בוקטורים מרובי ממדים לצורך אפיון של מילה.

קיימים שני סוגי embeddings – "דלילים" (sparse) ו"צפופים" (dense).

וקטורים "דלילים"

הוקטורים ה"דלילים" מיוצגים על ידי ספירה של מילים אשר מופיעות בקרבת המילה בקורפוס. בצורה טיפוסית, הוקטור יהיה בממד שגודלו כגודל המילון, כאשר כל אבר מייצג מילה אחרת, והערך המופיע בו הוא counter של כמות הפעמים שמילה זו מופיעה בקרבת המילה שאותה מקודדים. צורה זו היא "דלילה" מכיוון שלרוב, רוב הערכים בוקטור יהיו אפסים (או לפחות נמוכים מאוד), למשל, הסיכוי שתופיע המילה "אנציקלופדיה" ליד המילה "בלגיה" נמוך מאוד.

בשיטת word counts, מחושבת טבלה בגודל כאשר הוא מספר המילים במילון. איבר בטבלה מכיל את מספר הפעמים שמילה מופיעה במרחק מסוים קבוע (למשל 6 מילים) ממילה . מכיוון שיחס זה הוא גם הפוך, כלומר אם מילה קרובה ל- במקום מסוים בקורפוס אז גם מילה קרובה ל- באותו המקום, המטריצה סימטרית (). המטריצה נקראת word-word matrix או term-context matrix. שתי מילים תחשבנה קרובות במרחב ה-embeddings אם שורה דומה (למשל קרובה בנורמת ) לשורה .

וקטורים "צפופים"

שיטות לייצוג "צפוף" משתמשות בוקטור אשר הממד שלו קטן משמעותית מגודל המילון. בשיטות אלו נדרש להשתמש בשיטות חכמות יותר על מנת לקודד את המידע על המילה בצורה "דחוסה" יותר. הדוגמה המפורסמת ביותר לשיטה כזו היא Word2Vec. הרעיון בשיטה זו הוא לאמן מסווג שיחזה האם מילה תופיע בקרבת מילה . בצורה זו, ניתן להשתמש בקורפוס על מנת ליצור dataset גדול של זוגות מילים בעלות תיוג "חיובי" ללא צורך בתיוג (self-supervised), וכל המילים שאינן מופיעות ביחד תהיינה דוגמאות שליליות. לאחר מכן, מאומן מודל logistic regression שיחזה את המילים החיוביות, כאשר הפרמטרים הנלמדים הם הייצוגים של המילים. לאחר האימון, ייצוגים אלה נלקחים כ-embeddings.

*ישנם שני אלגוריתמים שהוצגו במאמר של Word2Vec: skip-gram ו-Continuous Bag of Words (CBOW).ב-skip-gram מנסים לחזות את סביבות המילה על בסיס המילה האמצעית (למשל עבור סביבה של – את על בסיס ), וב-CBOW להיפך – מנסים לחזות את המילה האמצעית () בהינתן סביבות המילה (). בשתי השיטות, לאחר למידת מודל כזה מתקבל ייצוג וקטורי של כל מילה במילון.*

*אחת מהאפשרויות לחישוב ההסתברויות היא מהצורה הבאה:  
כלומר, חישוב softmax בין ייצוג הווקטור הנתון לבין כל הוקטורים במילון . במהלך האימון, תתבצע אופטימיזציה כך שההסתברות תהיה גבוהה עבור זוגות מילים אשר קיימים בקורפוס.*

*הבעיה בשיטה זו היא שעבור כל חישוב של אנו נדרשים לחשב את המכפלות של עם הווקטורים המייצגים של כל המילים במילון. חישוב זה הינו אינטנסיבי עבור מילון גדול.* ***Negative sampling*** *היא שיטה הנועדה לפתור את הבעיה (נקראת גם SGNS - Skip-gram with Negative Sampling). Mikolov et al., 2013 הציעו את הרעיון הבא, במקום לחשב את ההסתברות על פני כל המילים במילון, ניתן לדגום k מילים אקראיות ולהשתמש רק בהן כדוגמאות שליליות. בצורה כזאת, הדמיון בין הווקטורים החיוביים יתקרב, והמרחק בין הווקטורים השליליים יתרחק בממוצע, מכיוון שהדוגמאות השליליות נדגמות באקראי. פונקציית המטרה אשר הוצגה במאמר היא:  
כאשר היא פונקציית הסתברות שנקבעה על בסיס ניסויים. היתרונות באלגוריתם הם מהירות האימון שלו והיעילות שלו. בנוסף, קיימים מימושים רבים שלו באינטרנט וכן "מילוני embeddings" מאומנים שלו.*

*נעיר כי בשיטות ה-Word2Vec כל מילה (או token) מקבלת ייצוג סטטי, אשר אינו תלוי בקונטקסט. עובדה זו יכולה להוות בעיה למשל כאשר יש למילה מספר משמעויות. לפעמים חשוב לתת למילה ייצוג לפי המשמעות שלה כתלות בקונטקסט. למשל היינו רוצים כי הייצוג של המילה "orange" במשפטים "A tasty orange" ו-“An orange pillow” יהיה שונה מכיוון שהמשמעות שונה. קיימים מודלים אחרים אשר באים להתמודד עם בעיה זו. למשל BERT נותן ייצוג לכל מילה על פי הקונטקסט של המשפט.*

*אחד מהתכונות המעניינות בווקטורים הנלמדים מ-Word2Vec היא הלינאריות של ה-embeddings (parallelogram model). למשל, הראו במאמר מוקדם בתחום כי:  
דבר זה מעיד כי ל-embeddings הנלמדים ייצוגים לינאריים בקירוב.*

*לסיכום, היתרונות של מודל שפה נלמד מבוסס feed forward אל מול N-gram:*

* *קל יותר למדל רצפים ארוכים של tokens*
* *הכללה טובה יותר של מילים דומות קונטקסטואלית (הדוגמה שהובאה לעיל עם cat ו-dog)*
* *ביצועים טובים יותר בחיזוי מילה*

*חסרונות:*

* *מימוש מורכב יותר, נדרש ללמוד מודל, לעומת N-gram בו רק נדרש לספור מילים בקורפוס*
* *איטי יותר ב-inference (לעומת N-gram בו מספיק להשתמש ב-lookup table)*
* *יותר אנרגיה לאימון*
* *פחות אינטרפרטבילי – קשה להבין את משמעות הווקטורים הנלמדים*

שימוש במודל feed-forward

*הארכיטקטורה של מודל שפה מסוג feed-forward הוא יחסית פשוט. ניקח לדוגמה מודל שמטרתו לחזות את ה-token בהינתן שלושת ה-tokens האחרונים: . נניח שנתון וקטור embeddings עבור כל token במילון , כאשר הוא הממד של וקטור ה-embeddings. בנוסף, נניח כי מספר ה-tokens במילון הוא . מכל הוקטורים . ניצור מטריצה כך שכל עמודה במטריצה מייצגת את ה-token ה- במילון ומכילה את ה-embeddings של אותה המילה. כעת, נייצג את ה-tokens של שלושת המילים האחרונות במשפט שאותו מעוניינים לחזות באמצעות one hot encoding, כלומר וקטור בגודל המילון המכיל אפסים בכולו מלבד המילה הנוכחית. את המילים המקודדות כ-one hot נייצג באמצעות . נשים לב כי מתקיים:  
כלומר, אם נכפיל את מטריצה ה-embeddings בייצוג ה-one hot של ה-token, נקבל את ה-embeddings של ה-token הנבחר. לאחר קבלת מתבצע שרשור (concatenation) של שלושת הווקטורים לווקטור embeddings אחד: :  
את הווקטור הזה מעבירים בשכבת fully connected:  
כאשר מספר העמודות ב- הוא , ומספר השורות הוא כמספר הנוירונים בשכבה. לבסוף מעבירים את לשכבת המוצא:  
הווקטור יהיה וקטור שגודלו כגודל המילון ומכיל את ההסתברויות של כל אחת מהמילים להיות המילה הבאה. סכמה של התהליך מוצגת באיור 1.*

**

איור 1: סכמה של מודל שפה מסוג feed-forward. כל מילה ממופית לאינדקס שלה במילון, לאחר מכן מומרת ל-one hot vector. לאחר מכן מוכפלת במטריצה E כך שמתקבלים וקטורי ה-embeddings של המילים. וקטורים אלו משורשרים ועוברים בשכבת fully connected ושכבת output, כך שמתקבל בסופו של דבר וקטור הסתברויות המכיל את ההסתברות לכל אחת מהמילים במילון.

*ניתן לאמן מודל כזה בצורה פשוטה כ-self supervised. בהינתן קורפוס, ניתן להתייחס לכל רביעיית מילים כדגימות (שלושת המילים הראשונות) ותיוג (המילה האחרונה). האימון מתבצע באמצעות cross entropy loss באמצעות gradient descent. בהרצאות כל משוואות ה-loss function וה-backpropagation מוצגים בהרחבה, אך לא נרחיב בסיכום זה.*

*לבסוף, קיימות שתי גישות לאימון מודל כזה. גישה אחת משתמשת בווקטורי embeddings קבועים (pre-trained) למשל, באמצעות Word2Vec, והאימון מתבצע אך ורק עם הפרמטרים של הרשת (). לעומת זאת, בגישת ה-end-to-end מתבצע backpropagation גם לווקטורי ה-embeddings, כך שהאימון מתבצע גם על הייצוגים של ה-tokens. נעיר כי קיימות גישות היברידיות אשר מאמנות בתחילה על וקטורי embeddings קבועים, ורק בסוף מבצעות fine tuning על כל הרשת כולל הייצוגים.*

RNNS

מודל שפה מבוסס RNN

*הערה: בשיעור המרצה העמיק בארכיטקטורת RNN. לא נסקור מבוא זה בסיכום.*

*רשת RNN מעבדת רצפים של inputs, גם אם האורך שלהם לא קבוע, על ידי וקטור "זיכרון" . תכונה זו הופכת את ה-RNN למתאימה באופן טבעי לעיבוד שפה טבעית, מכיוון שבאופן עקרוני, יכול להכיל את כל ההיסטוריה של הרצף . בשונה מרשת feed-forward לא נדרש להגדיר בצורה מפורשת את אורך ההיסטוריה הרצויה, כלומר כמה מילים להכניס לרשת. משוואות הרשת הן מהצורה הבאה:*

*עם מודל כזה אפשר להעריך בצורה ישירה את ההסתברות של המילה הבאה:  
וכן את ההסתברות לרצף של מילים:*

*אימון המודל מתבצע בדומה למתואר בפרק הקודם בצורה self-supervised על חיזוי המילה הבאה, כאשר ה-backpropagation מתבצע אחורה דרך רצף הפרדיקציות (backpropagation through time – BPTT). BPTT נצרך, מכיוון שהפרדיקציה בזמן t תלויה ב- אשר תלוי ב- וכן הלאה, ולכן העדכון צריך להתבצע עבור המשקולות שבעזרתן כל ה- מחושבים. מכיוון שכל החישובים של ה-RNN גזירים, ניתן לבצע BP גם על פרדיקציות שהתבצעו בעבר באותו הרצף. האימון מתבצע על ידי* ***teacher forcing****, שמשמעותו היא שה-loss עבור מילה מתבססת על הרצף האמיתי, ולא על פרדיקציית המודל. שיטה זו מאפשרת למידה יציבה ומהירה יותר.*

RNN למטלות NLP נוספות

*קיימות מטלות נוספות ב-NLP אשר ניתן למדל בצורה קלה באמצעות RNN. מטלה אחת כזאת היא sequence labeling – חיזוי של class עבור כל מילה במשפט. דוגמה לבעיה ספציפית של sequence labeling היא part-of-speech tagging, תיוג חלק דיבר (כלומר שם עצם, פועל וכדו') עבור כל מילה במשפט. RNN היא ארכיטקטורה קלאסית לבעיה זו, מכיוון שהמודל מקבל רצף של מילים, יודע להתחשב במילים בסביבת המילה כדי להסיק את משמעותה, וכן ניתן באמצעותו להחזיר פרדיציה על כל מילה על ידי הוספת שכבה אשר מקבלת את וחוזה – התיוג המתאים.*

*בעיה נוספת היא sequence classification. בעיות ספציפיות של sequence classification הן למשל ניתוח רגשות (sentiment analysis) – האם ביקורת מסויימת על סרט היא חיובית או שלילית, Spam detection – האם מייל מסויים הוא spam או לא. בבעיות אלה הרשת נדרשת לעבור על כל הרצף ולתת תיוג. ניתן לבצע זאת על ידי RNN – העברת כל הרצף ברשת ושימוש בוקטור האחרון לביצוע הקלסיפיקציה (הוספה של שכבת קלסיפיקציה שמקבלת את ומסווגת את הרצף). במהלך האימון, הרשת תלמד לקודד לתוך את האינפורמציה הרלוונטית לסיווג הרצף.*

*עוד שימוש אפשרי ל-RNN הוא text generation – יצירה של טקסט. Text generation הוא חיוני בבעיות כגון question answering, תרגום, תמצות טקסט, תיקון שגיאות כתיב, ועוד. בבעיות אלה הטקסט שנוצר נדרש להיות נכון מבחינה תחבירית וטבעי לאדם הקורא אותו. ניתן לבצע זאת באמצעות שערוך ההסתברויות למילה הבאה בהינתן המילים הקודמות, ודגימה של המילה הבאה מתוך אותה ההתפלגות. לאחר שנבחרה המילה הבאה ניתן לשערך מחדש את פונקציית ההתפלגות למילה שאחריה ולחזור על התהליך עד לסיום. תהליך זה נקרא* ***autoregressive generation*** *או*  ***causal LM generation****.*

שיפורים ל-RNN

*ארכיטקטורת ה-RNN הינה מאוד פשוטה ובסיסית, המשקולות היחידות שנלמדות הן - ה-embedding של המילון, W,U המטריצות שמכפילות את ו-V שכבת המוצא. לכן, כוח הייצוג שלהם מוגבל, כלומר יהיה קשה לבצע איתן מטלות אשר אינן קרובות ללינאריות. קיימים מספר שיפורים לרשת הבסיסית. שיפור ראשון הוא stacked RNN. בשיטה זו, במקום להשתמש ברשת יחידה, משורשרות מספר רשתות RNN כך שכל רשת מקבלת ככניסה את המוצא של הרשת הקודמת. בשיטה כזאת ניתן לבצע עיבוד מורכב יותר, מכיוון שהמוצא עובר מספר שכבות (מספר רשתות).*

*בעיה נוספת הקיימת ב-RNN היא שהרשת בזמן t "ראתה" רק את העבר ולא את העתיד. עם זאת, ישנן משימות בהן המשפט כולו קיים, ולכן קיימת אינפורמציה שאנו לא משתמשים בה לצורך הפרדיקציה. פתרון אפשרי לבעיה הוא bidirectional RNN, רשת המורכבת משני RNNs נפרדים אשר אחד מקבל את הרצף מהתחלה לסוף ואחד מהסוף להתחלה. שני הייצוגים (וקטורי הזיכרון) משורשרים לוקטור אחד אשר נעשה בו שימוש לטובת הפרדיקציה:  
כאשר הם הייצוגים של הרשת הראשונה (forward) והשניה (backward) של המילה t. בצורה כזאת, מכיל אינפורמציה הן מהעבר והן מהעתיד.*

*בעיה נוספת ב-RNN הבסיסי הוא ההתמודדות עם רצפים ארוכים. מכיוון שהוקטור עובר טרנספורמציה כל נקודת זמן, קשה למידע להישאר בו לאורך רצפים ארוכים. לצורך העניין, אינפורמציה שקודדה בזמן בתוך ונדרשת בזמן , תצטרך לעבור פעמים את פונקציית הזהות (או פונקציה דומה), על מנת שיהיה אפשר להשתמש בה ב-. בעיה זו חריפה עוד יותר כאשר הרשת נדרשת בנוסף להשתמש ב- לחיזוי בזמן , למשל במקרה של sequence labeling. בעיה נוספת הקיימת כאשר הרצפים ארוכים היא שה-BPTT צריך לעבור שכבות רבות ולכן קיים חשש ל-vanishing gradients.אחד מהפתרונות לבעיה הוא ה-LSTM - long short-term memory. רשת זו לומדת "לשכוח" אינפורמציה לא רלוונטית, ולהוסיף מידע אשר יהיה נדרש בהסתברות גבוהה בעתיד. ישנם שני שינויים מרכזיים ב-LSTM, אחד הוא הוספה של "שערים" (gates). שערים אלה אחראים על "זיכרון" ו"שיכחה" של המודל. פרקטית, אלו שכבות מהצורה הבאה:  
הסיגמואיד מאלץ את המוצא של השכבה להיות בין אפס לאחד. כאשר מכפילים מוצא זה בוקטור אחר, כל האברים הקרובים לאפס בשער יתאפסו בווקטור האחר, וכל הוקטורים הקרובים לאחד בשער ישארו ללא שינוי בוקטור האחר. כלומר, חלק מהאינפורמציה "נשכח" וחלק ממנה נשאר. כל השערים הם מאותה הצורה:  
השינוי השני הוא הוספת "וקטור קונטקסט", אשר מועבר משלב לשלב בדומה ל-. וקטור זה עובר עיבוד (כלומר עובר שער) והתוצר של העיבוד הופך ל-. דבר זה מאפשר לבצע הבחנה בין וקטור ה"זיכרון" לוקטור שבו נעשה שימוש לפרדיקציה בזמן t.*

*כעת נצלול לארכיטקטורה. וקטור (הקונטקסט) מורכב מסכום של שני איברים:  
האיבר הוא הרכיב של שעובר מהשלב הקודם . כלומר, עובר ב"שער" (forget gate) אשר שולט באיזה אינפורמציה* ***נמחקת*** *מ-, על ידי הכפלה באפס. האיבר שולט באיזו אינפורמציה* ***מתווספת*** *ל-, כלומר מה אנחנו מעוניינים "לזכור". דבר זה מתבצע על ידי הכפלה של :  
בשער . היא האינפורמציה שמופקת בשלב t, ו- שולט באיזה חלק מהאינפורמציה אנחנו מעוניינים לאגור לשלבים הבאים. לאחר ה"מעבר בשער" (ההכפלה ב-), הוקטור מתווסף ל-. לאחר שמחושב ניתן לחשב את על ידי העברה בשער נוסף (על מנת להנגיש "החוצה" את האינפורמציה הרלוונטית מתוך ):*

*בצורה הזאת, ארכיטקטורת ה-LSTM מסוגלת להתבסס על וקטור זיכרון משוכלל יותר, וכך גם להתמודד בהצלחה עם רצפים ארוכים וגם לבצע את ההפרדה בין הזיכרון לפרדיקציה.*

Encoder-Decoder RNN

*בנוסף ל-sequence labeling, sequence classification ו-text generation, קיימת מטלות מסוג sequence to sequence. במטלות אלה הרשת מקבלת רצף של מילים ונדרשת להוציא רצף של מילים. הדוגמה הקלאסית לכך היא תרגום (machine translation), ובנוסף בסיכום (summarization), דיאלוג (כמו chatGPT) ו-question answering. ניתן לבצע מטלות אלה על ידי RNN על ידי שימוש ב-encoder ו-decoder. ה-encoder יקבל את רצף הכניסה ויקודד אותו לוקטור בגודל קבוע (למשל לתוך ). לאחר מכן ה-decoder יקבל את הקידוד כ-input וישתמש בו על מנת ליצור את רצף המוצא מילה אחרי מילה (בדומה ל-text generation שהזכרנו לעיל).*

Attention

*הבעיה המרכזית הקיימת בארכיטקטורת ה-encoder-decoder ב-RNN היא ה-information bottleneck – "צוואר הבקבוק". מכיוון שכל המידע מה-encoder מקודד לווקטור יחיד () באורך קבוע, כמות המידע שיכולה להיות מקודדת בו היא מוגבלת. עבור משפטים ארוכים, האינפורמציה המצויה בתחילת המשפט עלולה "להימחק" ולא להגיע על לווקטור הקונטקסט. מכיוון של-decoder יש גישה אך ורק לווקטור הקונטקסט, המידע שהוא יכול לפענח מוגבל. מנגנון ה-attention בא לפתור בעיה זו. הרעיון מאחוריו הוא שהמפענח יקבל בכל שלב וקטור* ***שונה*** *המורכב מסכום משוקלל של האינפורמציה מהמקודד בנקודות הזמן השונות. פונקציית ה-attention תבחר את המשקל עבור כל hidden state של המקודד כך שהמידע שיגיע למפענח בנקודת הזמן הנוכחית יורכב מהמידע הרלוונטי עבורו. חישוב ה-hidden state של המפענח בנקודת זמן תושפע מהפרדיקציה הקודמת, המצב הקודם וכן – וקטור האינפורמציה המשוקללת מהמקודד:  
כאשר מחושב בצורה הבאה. בשלב הראשון מתבצע חישוב של המשקלים של כל אחד מה-hidden states של המקודד, במילים אחרות, אילו חלקים מתוך דגימת ה-input רלוונטיים לפענוח בנקודת הזמן הזו. הצורה הפשוטה ביותר למדל משקל זה היא על ידי מכפלת וקטורים:  
נעשה חישוב של ה-score עבור כל אחד מ- ולאחר מכן מתבצע softmax על מנת לנרמל את הערכים:  
לאחר מכן מחושב באמצעות המישקול של :  
בצורה כזאת, מחושב ממוצע משוקלל של כל ה-hidden states של ה-encoder () והוא נלקח בחשבון בפענוח.*

Transformers

*ארכיטקטורה דומה אך אחרת היא ה-transformer או ה-self-attention. הארכיטקטורה ממפה רצף של וקטורים לרצף של ווקטורים באותו האורך: . ה-transformer מורכב משכבות transformer blocks שכל בלוק מורכב ממספר שכבות במבנה קבוע. שכבת ה-self-attention מאפשרת לרשת גישה לכל ה-inputs ללא צורך ב-RNN. כל בלוק כזה מקבל וקטורים כ-input ומוציא וקטורים.*

*בשכבת זו, ה-attention מוגדר בצורה הבאה:  
כאשר:  
כלומר הם טרנספורמציה לינארית של (שלוש טרנספורמציות שונות). הוא ה-****query****, כלומר הערך שה-token הנוכחי שאנו מעבדים "מתשאל" את שאר ה-tokens, ו- הם ה-****keys****, ערכים שמייצגים את שאר ה-tokens. לאחר אימון ה-transformer, אם ה-token ה- אינפורמטיבי לטובת הפרדיקציה בשלב ה-, הדמיון בין ה-query ל-key יהיה גדול, ו- יהיה גדול. הוא ה****ערך*** *עצמו, כלומר הערכים שאותם "נעביר הלאה". אם יהיה גבוה, אזי יהיה ייצוג משמעותי לווקטור בווקטור המשוקלל . הסימון בסכום הוא כאשר מאמנים מודל שחוזה את המילה הבאה, ולכן הוא יקבל אך ורק את ה-tokens הקודמים.*

*על מנת להימנע מערכים גבוהים במכפלה , וכתוצאה מכך אימון לא יציב, מתבצע נירמול של ה-score בממדי הוקטור:  
בנוסף, ניתן לבצע את כל החישובים הנ"ל בצורה מקבילית על כל ה-tokens. ניתן לייצג את כל הכניסות כמטריצה , ולבצע את החישובים באמצעות כפלי מטריצות:  
ולחשב את כל ה-scores באמצעות . לאחר מכן נרצה לבצע softmax על פני השורות של . עם זאת, אם נבצע softmax רגיל, יתקבלו ערכים גם "מהעתיד", כלומר עבור ה-token ה-, היינו רוצים כי כל הערכים הגדולים מ- יתאפסו בשורה ה-. לכן לפני ביצוע ה-softmax ניתן להציב במשולש העליון של המטריצה כך שה-softmax יתאפס במקומות הללו.*

*ה-transformer block מורכב מהשלבים הבאים. בשלב הראשון מתבצע ה-self-attention שתואר לעיל. לאחר שלב זה מתקבלים וקטורים . לאחר מכן מתבצעת נורמליזציה – layer normalization, אשר מנרמלת את הממוצע וסטיית התקן של כל הוקטורים (אותו הרעיון של batch normalization). הנורמליזציה מתבצעת עבור כל אחד מהווקטורים בנפרד. עבור כל token, מחושב הממוצע וסטיית התקן של הערכים שלו ומתבצע נרמול. בדומה ל-batch normalization, קיימים שני פרמטרים נלמדים אשר שולטים על הממוצע וסטיית התקן. לסיכום, עבור כל וקטור מתבצעים השלבים הבאים:  
לאחר מכן, כל וקטור (מנורמל) עובר בשכבת FC ולאחר מכן בעוד נירמול. מתבצע גם residual connection אשר "עוקף" את ה-self-attention ואת ה-fully connected (אבל לא את הנורמליזציה). לסיכום, משוואת ה-attention block:  
כאשר היא שכבת fully connected (feed forward).*

*שיפור שנהוג לבצע לשכבת self attention הוא ה-multi head attention. שכבה זו מכילה N שכבות attention שפועלות במקביל, עם סט פרמטרים שונה לכל שכבה, כאשר בסופה של השכבה מתבצע שרשור (concatenation) של כל ה-outputs של השכבה והטלה לינארית (linear projection) חזרה לממד . הרציונל במספר שכבות attention הפועלות במקביל הוא שכל שכבה תתמקד בפן אחר של קשרים בין inputs, ויחד שכבות אלו יוכלו לתפוס את היחס השלם שבין ה-tokens.*

*שיפור נוסף שנהוג לבצע במשימות NLP הוא הוספת ה-positional encoding. הבעיה שה-positional encoding בא לפתור היא שבשונה מ-RNN, ה-transformers הוא אינווריאנטי לסדר, כלומר אין שום דרך למודל לדעת באיזה מקום כל token נמצא במשפט. אך בשפה טבעית המיקום של המילים הוא קריטי להבנת המשפט, ולכן הרשת מוכרחת לקבל בנוסף מידע לגבי מיקום ה-token במשפט. אחד מהפתרונות הוא הוספה של שכבת embeddings שתקודד את המיקומים. כלומר, הרשת תלמד ייצוג שונה עבור כל מיקום, וייצוג זה יתווסף לייצוג ה-embedding של המילה עצמה. למשל המשפט “The dog eat meat” יקודד ל:  
בצורה כזאת, המודל יקבל את האינפורמציה לגבי המיקום במשפט, למרות שהארכיטקטורה אינווריאנטית לסדר.*