**עיבוד שפה טבעית - תרגיל בית רטוב 2**

מגישים: אור קרופניק 302629027

רואי סינוף 201542982

**אימון**

כללי

1. הן עבור המודל הבסיסי והן עבור המודל המתקדם, עבור כל המאפיינים השתמשנו במבנה נתונים מסוג מילון מקונן (או מילון של מילונים). לדוגמה:

* p\_word נשמר כמילון שבו המפתחות הן המילים שמשמשות כ"אבות", או מקורות הקשתות, והערך שלהן הוא מס' תת-המאפיין
* p\_word, c\_word, c\_pos נשמר כמילון שבו המפתחות הן המילים שמשות כ"אבות". עבור כל מפתח כזה, הערך הוא מילון נוסף שבו המפתחות הן המילים שנראו כ-"בנים" של "אב" זה (כלומר יש קשת מ-p\_word ל-c\_word). לכל מפתח שבתת המילון הזה העכים הם מילונים נוספים אשר המפתחות שלהם הם ה-POS של ה"בן" וערכם הוא מספר הפיצ'רים.

1. במסגרת האימון השתמשנו בחבילות הקוד הבאות:
   1. numpy - עבור וקטור המשקולות.
   2. random – על מנת להכניס אקראיות (פונקציית shuffle) על המשפטים אותם לומד הפרספטרון.
   3. edmonds – על מנת לקרוא לפונקציית ה-mst של Chu-Liu-Edmonds.
   4. pickle – כדי לשמור את הפיצ'רים שיצרנו על מנת שנוכל לטעון אותם בעתיד מבלי לבצע parsing מחדש.
2. במהלך קריאת קובץ הקלט אנחנו יוצרים אובייקטים מסוג משפט, הכוללים בין היתר רשימה של אובייקטי קשתות וכן היסטוגרמה של הפיצ'רים שנדלקים בעץ עבור משפט מסוים – דבר זה שימש אותנו בהמשך באלגוריתם הפרספטרון (ראה הערה 4).
3. באלגוריתם הפרספטרון, עבור כל משפט שקיבלנו יצרנו 2 גרפים שלמים: אחד הכולל את הפיצ'רים של קשת ספציפית והשני הכולל את סכום משקלי הפיצ'רים הללו. את שלב עדכון המשקלים ביצענו **תמיד** (גם אם לא טעינו בעץ שפרשנו), כאשר בפועל אם צדקנו בפרסור הרי שחיסרנו וחיברנו אותם ערכים (ולכן לא נעשה שינוי במשקלים, כפי שהוגדר באלגוריתם המקורי). חיבור הערכים בוצע ע"י שימוש בהיסטוגרמת הפיצ'רים שצויינה בסעיף 3, בעוד שחיסור הערכים בוצע ע"י חיסור 1 בוקטור המשקלים עבור כל הפיצ'רים המתאימים לקשת זו בגרף הפיצ'רים.
4. כאמור, אחד השינויים שביצענו באלגוריתם הפרספטרון הנתון הוא הכנסת הקלט בצורה אקראית, על מנת "לשבור" רצפים קבועים שיכולים ליצור הטייה בערכי המשקלים שנחשב.

מודל בסיסי

להלן סיכום כמויות הפיצ'רים במודל הבסיסי (מספרי הפיצ'רים כפי שהוגדרו בתרגיל):

|  |
| --- |
| 1 : 9993  2 : 8876  3 : 37  4 : 15908  5 : 14162  6 : 45  8 : 31314  10 : 33936  13 : 749  total : 115020 |

הזמן שלקח להריץ את הלימוד עבור המודל הבסיסי:

|  |  |  |
| --- | --- | --- |
| **מס' האיטרציות** | **זמן הלימוד** | **המחשב עליו בוצעה הריצה** |
| 20 | 0:20:31 | 4 i7 CPUs, 16 GB RAM |
| 50 | 1:08:00 |
| 80 | 1:44:00 |
| 100 | 2:05:00 |

מודל מתקדם

במסגרת המודל הוספנו פיצ'רים המתייחסים לאינדקס המילים בקשת (הן ה"אבא" והן ה"בן"), למרחק בין המילים בקשת, לגודל המשפט הכולל, למילים המופיעות לפני / אחרי המילים שבקשת וכן ל"אחים" של ה"בן". חלק מהפיצ'רים ממש מחקנו מאחר שפגעו בתוצאות בצורה ברורה (לדוגמה כמות ה"אחים"), ועל אחרים בסופו של דבר החלטנו לוותר, אך השארנו כתובים (מאחר שהם לא פגעו בביצועים, אך גם לא שיפרו). להלן הפיצ'רים אשר בסופו של דבר השתמשנו במסגרת מודל זה (בסוגריים הראשונים מספר המאפיין, ואז לפי הסדר התגים לפיהם מאונדקסים המילונים המקוננים. למשל, עבור פיצ'ר מס' 1, החיפוש הוא לפי מילת ה"אב", או המילה ממנה יוצאת הקשת, ואז לפי חלק הדיבר של ה"אב"):

[1][p\_word][p\_pos]

[2][p\_word]

[3][p\_pos]

[4][c\_word][c\_pos]

[5][c\_word]

[6][c\_pos]

[7][p\_word][c\_word][p\_pos][c\_pos]

[8][c\_word][p\_pos][c\_pos]

[10][p\_word][p\_pos][c\_pos]

[12][p\_word][c\_word]

[13][p\_pos][c\_pos]

[14][p\_word][p\_ind]

[15][p\_pos][p\_ind]

[16][c\_word][c\_ind]

[17][c\_pos][c\_ind]

[18][p\_word][p\_pos][p\_ind]

[19][c\_word][c\_pos][c\_ind]

[20][p\_pos][c\_pos][p\_ind]

[21][p\_pos][c\_pos][c\_ind]

[22][p\_pos][c\_pos][p\_ind][c\_ind]

[23][p\_word][c\_word][dist]

[24][p\_pos][c\_pos][dist]

[25][p\_word][p\_pos][c\_pos][dist]

[26][c\_word][p\_pos][c\_pos][dist]

[28][p\_pos][c\_pos][dist][sen\_len]

[29][p\_word][p\_pos][dist][sen\_len]

[30][c\_word][c\_pos][dist][sen\_len]

[31][p\_word][p\_pos][c\_pos][dist][sen\_len]

[32][c\_word][p\_pos][c\_pos][dist][sen\_len]

[33][p\_ind][c\_ind][sen\_len]

[34][p\_pos][p\_ind][c\_ind][sen\_len]

[35][c\_pos][p\_ind][c\_ind][sen\_len]

[36][p\_pos][c\_pos][p\_next\_pos][c\_next\_pos] [41][p\_pos][c\_pos][p\_next\_pos][c\_pre\_pos][dist]

[42][p\_pos][c\_pos][p\_pre\_pos][c\_next\_pos][dist]

[43][p\_pos][c\_pos][p\_pre\_pos][c\_pre\_pos][dist]

[44][p\_word][c\_pos][dist]

[45][p\_pos][c\_word][dist]

[46][p\_word][dist]

[47][p\_pos][dist]

[48][p\_word][p\_pos][dist]

[49][c\_word][dist]

[50][c\_pos][dist]

[51][p\_word][c\_word][p\_pos][c\_pos][dist]

[52][p\_word][c\_word][c\_pos][dist]

[53][p\_word][c\_word][p\_pos][dist]

מקרא:

p – "אב", או המילה ממנה יוצאת הקשת

C – "בן", המילה אליה מצביעה הקשת

Word – המילה עצמה

Pos – חלק דיבר

Ind – אינדקס במשפט (ROOT מקבל 0 ואז לפי הסדר)

Sen\_len – אורך המשפט במילים

Dist – המרחק בין ה"אב" ל-"בן", למעשה p\_ind – c\_ind. יכול להיות כמובן שלילי

Pre\_pos – חלק הדיבר של המילה שלפני המילה המדוברת

Next\_pos – חלק הדיבר של המילה שאחרי המילה המדוברת

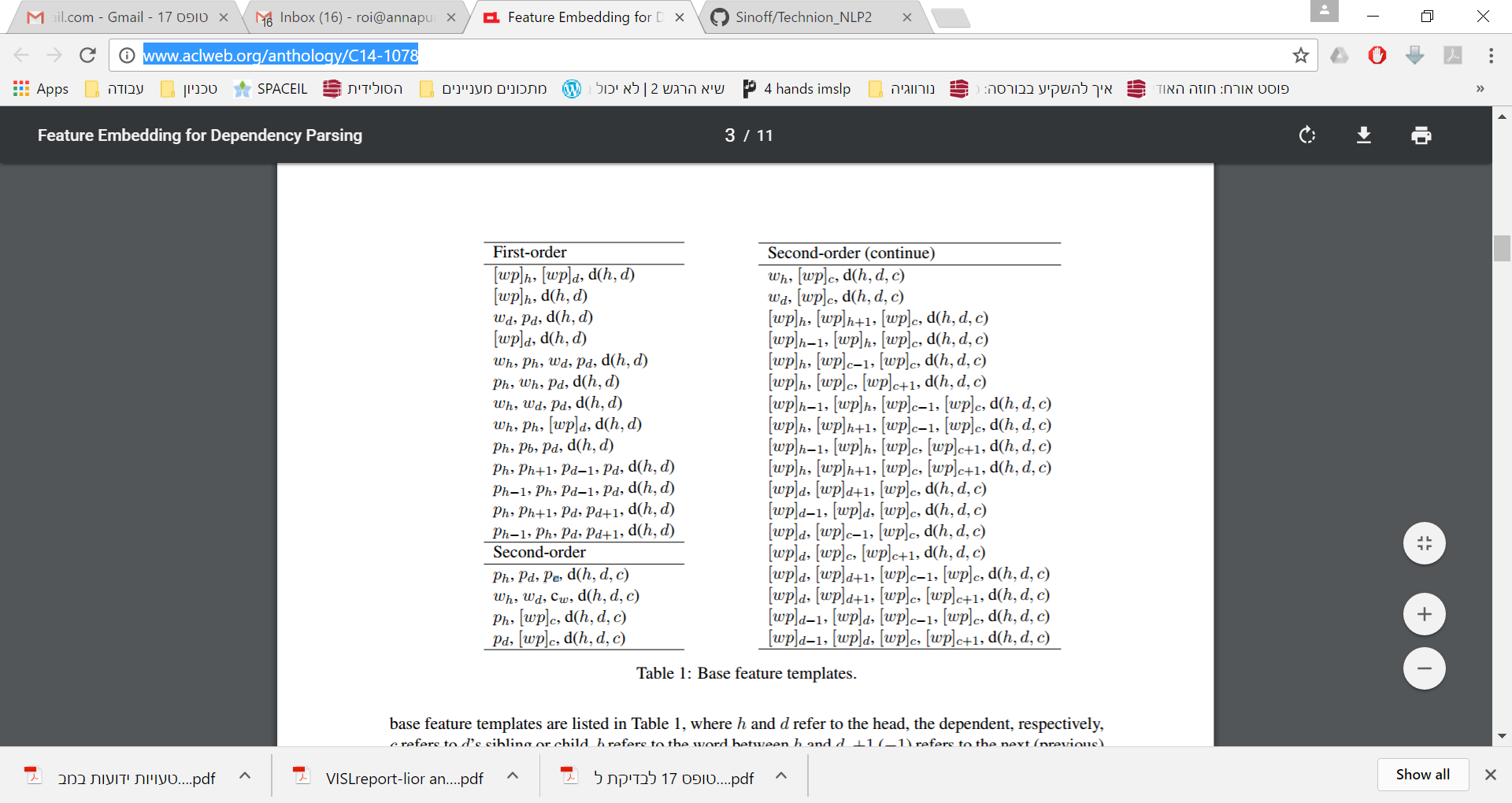
להלן כמויות הפיצ'רים האלה:

|  |
| --- |
| 1 : 9993  2 : 8876  3 : 37  4 : 15908  5 : 14162  6 : 45  7 : 71232  8 : 31314  10 : 33936  12 : 69819  13 : 749  14 : 29600  15 : 1366  16 : 50409  17 : 2459  18 : 30402  19 : 52172  20 : 12776  21 : 13501  22 : 33732  23 : 82999  24 : 6196  25 : 52139  26 : 46235  28 : 40317  29 : 81835  30 : 75089  31 : 95517  32 : 88765  33 : 21128  34 : 57869  35 : 61530  36 : 16447  41 : 28329  42 : 29393  43 : 28118  44 : 51120  45 : 45525  47 : 986  48 : 35506  50 : 1387  51 : 83822  52 : 83357  53 : 83485 |

בסך הכל: 1,679,582 פיצ'רים.

להלן הסברים לשינויים שביצענו:

1. ראשית, נציין שבחרנו את הפיצ'רים על סמך אינטואיציה ועל סמך מה שלמדנו בכיתה, אך בהמשך חיפשנו גם בספרות ומצאנו לדוגמה את הטבלה הבאה[[1]](#footnote-1). לשמחתינו הפיצ'רים שבחרנו תאמו במרביתם לאלו שהוצעו במאמר, אך הוא גם נתן לנו מוטיבציה לוותר על פיצ'רים מסויימים.



1. כפי שצוין, ניסינו מספר רב של פיצ'רים אשר בסוף לא השתמשנו בהם (ואת חלקם אף מחקנו – כגון פיצ'רים המתייחסים לכמות ה"אחים" של מילת "בן" בקשת, אשר פגעו בביצועים). עם זאת, ייתכן ששימוש בחלוקה כלשהי של הערכים למחלקות יותר מוצלחות (ע"י הגדרת סף, לדוגמה עבור אורך משפט –"קטן מ-10", "בין 10 ל-15" וכו'), היה גורם לפרמטר זה להפיק מידע יותר אינדיקטיבי ובהתאם לשיפור יותר גדול בתוצאות.  
   נציין שהיו פיצ'רים נוספים שיכולנו לוותר עליהם מאחר שלא תרמו מידע, אך מפאת קוצר הזמן לא הספקנו לבודדם בצורה מיטבית, ולכן החלטנו להישאר עם המודל שנתן את התוצאות המיטביות, גם אם הוא "בזבזני" בכמות הפיצ'רים.

הזמן שלקח להריץ את הלימוד עבור המודל המתקדם:

|  |  |  |
| --- | --- | --- |
| **מס' האיטרציות** | **זמן הלימוד** | **המחשב עליו בוצעה הריצה** |
| 20 | 1:15:12 | 4 i7 CPUs, 16 GB RAM |
| 50 | 3:05:00 |
| 80 | 4:55:00 |
| 100 | 6:08:00 |

**הסקה**

1. במסגרת ההסקה ניסינו שתי אופציות למציאת ה- MST. הסיבה לכך היא שקוד ה- Chu-Liu-Edmonds כלל באגים ולכן לא תמיד יצר עצים פורשים (לעיתים היו מעגלים, לעיתים הגרף שהתקבל חזרה לא היה קשיר וכו'), וניסינו לפתור זאת באמצעות השיטה השנייה. עם זאת, בסופו של דבר התוצאות שהתקבלו עבור השיטה הראשונה היו טובות יותר, על אף המקרים בהם הגרפים שהתקבלו לא היו עצים תקינים. להלן שתי השיטות:
   1. Onetree – במסגרת שיטה זו, חיברנו את צומת ה-"root" לכל הצמתים (כמובן רק בכיוון אחד), ופונקציית ה-MST היא שבחרה את הקשת המיטבית.
   2. Best Tree – במסגרת שיטה זו, חיברנו את צומת ה"root" בכל פעם רק לצומת אחת והרצנו את אלגוריתם ה-MST על עץ זה. בסוף בחרנו את העץ המיטבי מבין הריצות השונות. היתרון בשיטה זו הוא שאנו מכריחים את האלגוריתם לאפשר לכל היותר קשת אחת מצומת ה-“root”.
2. נציין שהשיטה הראשונה מהירה יותר משמעותית – בערך פי 25, דבר המתאים לגודל המשפטים הממוצע (ובהתאם לכמות הקריאות ל- MST בשיטה 2).

**מבחן**

מודל בסיסי

להלן זמני הריצה והתוצאות עבור המודל הבסיסי:

|  |  |  |  |
| --- | --- | --- | --- |
| **מס' איטרציות** | **זמן הריצה** | **אחוז הדיוק** | **המחשב עליו בוצעה הריצה** |
| 20 | 0:07 | 29.8% | 4 i7 CPUs, 16 GB RAM |
| 50 | 0:08 | 30.2% |
| 80 | 0:08 | 26.9% |
| 100 | 0:08 | 30.85% |

מודל מתקדם

נציין שהמימוש שלנו יכול להיות מהיר יותר אם נמחק פיצ'רים שאינם בשימוש, אך מאחר שממילא זמני הריצה סבירים, לא טרחנו לנסות ולשפר זאת, למקרה שנרצה בסופו של דבר להשתמש בפיצ'רים אלה.

להלן זמני הריצה והתוצאות עבור המודל המתקדם (כפי שתואר בחלק ה"לימוד"):

|  |  |  |  |
| --- | --- | --- | --- |
| **מס' איטרציות** | **זמן הריצה** | **אחוז הדיוק** | **המחשב עליו בוצעה הריצה** |
| 20 | 0:44 | 80.62% | 4 i7 CPUs, 16 GB RAM |
| 50 | 0:43 | 77.6% |
| 80 | 0:44 | 76.4% |
| 100 | 0:41 | 75% |

כללי

1. כפי שניתן לראות, התוצאות שהתקבלו במודל המתקדם טובות יותר משמעותית מהתוצאות שהתקבלו במודל הבסיסי. הדבר סביר מאחר שהמודל הבסיסי לא כלל מידע מהותי לגבי הקשתות (לדוגמה אורכן), ושהמודל המתקדם נבחר בתור האופציה המיטבית מתוך סט הרצות ואופציות מאפיינים רחב יותר בצורה משמעותית.
2. נשים לב שהתוצאות שהתקבלו עבור 20 איטרציות הן המיטביות, ככה"נ בשל overfitting לקובץ הלימוד. ניתן היה למנוע זאת ע"י הוספת תנאי עצירה שמבוסס על התוצאות במהלך הריצה על מנת להקטין את הסיכוי שדבר כזה יקרה. תנאי עצירה אפשריים לדוגמה – כמות המחזורים הרצופה בה לא טעינו, גודל הטעות (במשקל או בקשתות שלא סיווגנו נכון) וכו'.

**תחרות**

1. תחזית של אחוזי הדיוק שאנו צופים לקבל – דומה לאלו שקיבלנו עבור המבחנים שביצענו לעיל, או מעט נמוכה יותר. עם זאת יכולות להיות מספר סיבות יכולות להשפיע על ההבדל בין בחינת הקבצים השונים:
   1. התאמות מסוימות שביצענו בעקבות הטעויות שראינו בעת הבחינה בקובץ המקורי עשויות להתאים ספציפית לקובץ זה, ולא להיות כלליות מספיק כדי לשפר תוצאות גם על קובץ התחרות (אולי אף overfitting, על אף שניסינו להימנע מכך). נציין שלהערכתינו הסיכוי שהדבר קרה לנו במקרה הנוכחי נמוכה יחסית לתרגיל הקודם (שגם בו זה לא קרה...), מאחר שהפעם "האינטואיציה הבלשנית" שלנו פחות מפותחת, שהרי "dependency trees" הם פחות אינטואיטיביים לעומת POS והקשרים ביניהם (כפי שהיה בתרגול הקודם).
   2. באופן דומה למה שכתבנו בתרגיל הקודם – יכול להיות קיים הבדל ב-domain ממנו הגיעו ה-test לעומת ה-comp (על אף שבמבט חטוף על הקבצים שניתנו לנו במקרה זה, נראה שזה לא המקרה).
2. על מנת לוודא שההטייה שנובעת מביצוע שיפורים מקסימליים על קובץ ה-test תהייה נמוכה ביצענו מעין "Cross Validation" בסיסי מאוד –יצרנו קבצי train ו- test נוספים ע"י חיבור הקבצים שקיבלנו ופיצולם מחדש לסטים שונים, על מנת לראות שהתוצאות שאנו מקבלים עדיין טובות – ואכן קיבלנו כ- 80.5% (כמו התוצאות עבור הסטים המקוריים). על כן כאמור אנו מעריכים שאחוז הדיוק יהיה דומה מאוד לזה לערך זה.
3. כפי שצויין בשלב הלימוד, הכנסנו את הקלט לפרספטרון בצורה אקראית.
4. אנו מעריכים שניתן לשפר את אחוז הדיוק על קובץ התחרות אם נגדיל את הדאטה, לדוגמה ע"י שימוש גם ב-test בתור דאטה ללמידה (כמובן אחרי שסיימנו עם שלב הולידציה). על מנת לראות אם הערכה זו נכונה – הוספנו קובץ שלישי בשם comp\_m3\_201542982.wtag.

**סיכום**

נציין שבדומה לכלל הקורס, גם תרגיל זה היה מעניין ומעורר מחשבה. על אף שבתרגיל הקודם אחת המסקנות היתה שכוח חישובי יותר גדול היה יכול לעזור לנו מאוד, החלטנו לוותר על שימוש בשרת בענן, מאחר שהכוח החישובי הנדרש הפעם היה נמוך יותר משמעותית (הפרספטרון מהיר יותר משמעותית ביחס ל Gradient Descent אותו מימשנו בתרגיל הקודם).

מסקנה נוספת מהתרגיל הקודם היא שעל אף שיש יתרונות בקידוד בזוג, במסגרת הזמן של התרגיל זו שיטה בעייתית. לכן אמנם הגדרת מבני הנתונים (ובהתאם המחשבה על מימוש האלגוריתמים) נעשתה במשותף, אך הקידוד נעשה בנפרד (גם אם בעת ישיבה משותפת זה לצד זה). נציין כי למרות הנ"ל, דגלנו גם הפעם באחריות משותפת על הקוד וביצענו שיפורים ותיקונים בכל הקוד – מבלי להתייחס למי כתב את הקוד במקור.

בחלוקה גסה ניתן לומר שאור כתב את המודולים classes, depparser והבסיס של features, תיעד, וכן חקר לעומק, דיבג וניסה למצוא קוד נכון יותר של אלגוריתם edmonds. רואי (שכתב פייתון לראשונה במסגרת התרגילים בקורס זה) כתב את main, learning, inference ומרבית הפיצ'רים המתקדמים (אשר האחריות לחשיבה עליהם הופקדה עליו, אך נעתשה כמובן בהתייעצות ובסיכום עם אור) וכן את הבסיס של דו"ח זה.

1. From: Chen, Wenliang, Yue Zhang, and Min Zhang. "Feature Embedding for Dependency Parsing." COLING. 2014. [↑](#footnote-ref-1)