# 10. Natural Language Processing

עיבוד שפה טבעית (NLP) הוא תחום העוסק בפיתוח אלגוריתמים לעיבוד וניתוח של דאטה טקסטואלי. המטרה העיקרית היא לפתח שיטות ומודלים שיאפשרו למכונות "להבין" את התוכן הטקסטואלי, ואת הניואנסים וההקשרים של השפה. עולם ה-NLP מורכב ממספר רב של תתי משימות, כגון ניתוח סנטימנט של טקסט (Sentiment analysis), תמצות/סיכום אוטומטי של טקסט (Text summarization), מציאת תשובה עבור שאלה נתונה (Question answering) ועוד משימות רבות אחרות. פיתוח כלים המסוגלים להתמודד עם משימות אלה יאפשר לנו (בין היתר) לפתח אפליקציות שיעזרו לנו ביום יום כגון עוזרות קוליות, מערכות תרגום, מערכות אוטומטיות לבדיקת דקדוק ועוד. כמה דוגמאות לאפליקציות כאלה שכולנו מכירים הן Siri, העוזרת הקולית של אפל, ההשלמה האוטומטית בתיבת החיפוש ב-Google ו-Grammarly, מערכת לתיקון תחבירי לטקסט באנגלית.

## 10.1 Language Models and Word Representation

בפרקים הקרובים נדון בשניים מהנושאים הבסיסיים ביותר בתחום ה-NLP – מודלי שפה וייצוגי מילים: נראה כיצד מייצרים מודל שפה מדאטה סט טקסטואלי (שנקרא גם "Corpus"), וכיצד מייצגים את הטקסט כך שיהיה אפשר לאמן בעזרתו מודל. כדי להקנות למכונה יכולת הבנה של דאטה טקסטואלי יש לבנות מודל הסתברותי הקובע את ההתפלגות של המילים בשפה. המודל מנסה לכמת את הסיכוי להופעה של סדרות שונות של מילים, ובאופן יותר כללי הוא קובע מה ההסתברות של כל רצף אפשרי להופיע. מודל כזה אפשר לבנות בעזרת אימון על גבי דאטה טקסטואלי, והוא נקרא "מודל שפה" (Language Model). לפני ביצוע האימון, יש לבצע מיפוי של הטקסט לייצוג מסוים (Word Representation), המאפשר למכונה לקחת את הדאטה הקיים, לעבד אותו ולנסות לבנות בעזרתו את מודל השפה.

ראשית נגדיר מהו מודל שפה: מודל שפה הינו מודל סטטיסטי המגדיר התפלגות מעל המרחב המורכב מכל הסדרות האפשריות של מילים (כלומר משפטים, פסקאות וכדומה). מודל זה מקבל כקלט סדרה של מילים ותפקידו הוא לחזות מה היא המילה הבאה שתניב את ההסתברות המרבית לרצף ביחד עם המילה הנוספת.

כעת נתאר בצורה מתמטית מהו מודל שפה. נניח ונתון משפט עם $n$ מילים: $[w_1, \dots, w_n]$, אז ההסתברות לקבל את המשפט הזה הינה:

$$P(w_1, \dots, w_n) = \prod_{i=1}^{n} P(w_i | w_1, \dots, w_{t-1})$$

ניקח למשל את המשפט הבא:

Take a big corpus

ההסתברות של משפט זה ניתנת לחישוב אופן הבא:

$$P(Take, a, big, corpus) = P(Take) \cdot P(a|Take) \cdot P(big|Take, a) \cdot P(corpus|Take, a, big)$$

במילים, נוכל לתאר את המשוואה הזו כך: ההסתברות לקבל את המשפט הנתון שווה להסתברות של המילה Take להופיע כפול ההסתברות של המילה a להופיע אחרי המילה Take, כפול ההסתברות של המילה big להופיע אחרי הצירוף Take a, כפול ההסתברות של המילה corpus להופיע אחרי הצירוף Take a big.

באופן הפשטני ביותר נוכל לשערך את ההסתברויות האלו באופן הבא: ניקח corpus מספיק גדול ועשיר בעל $N$ מילים שונות, כמו למשל כל ערכי ויקיפדיה, ופשוט נספור את כל המילים והצירופים שמופיעים בו. ההסתברות של כל מילה תהיה אחוז הפעמים שהיא מופיעה בטקסט, וההסתברות המותנית תחושב באופן דומה – על ידי מניית מספר המופעים של צירוף כלשהו וחלוקה במספר הפעמים שהמילה עצמה מופיעה. באופן פורמלי נוכל לרשום זאת כך:

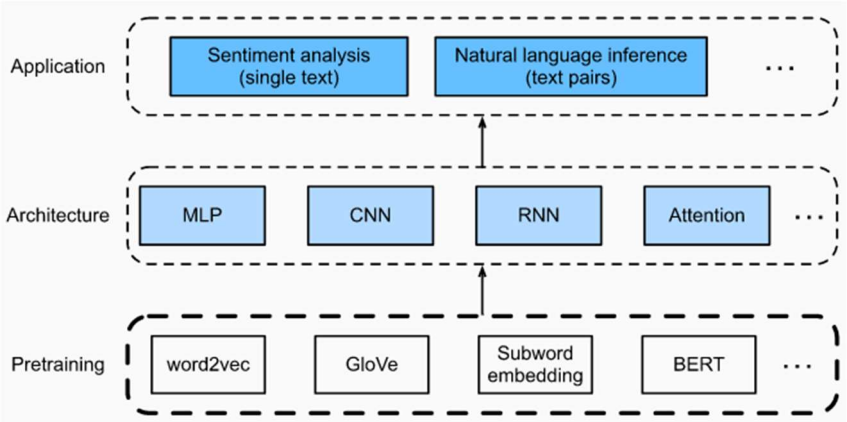
$$P(W_i) = \frac{\#W_i}{N}, P(W_i|W_j) = \frac{\#W_i, W_j}{\#W_i}$$

בין אם נחליט לקחת את מודל השפה הזה או שנייצר מודלים מתוחכמים יותר כפי שנראה בהמשך, המודל הוא אחד הדברים היסודיים ביותר במשימות שפה, כיוון שבעזרתו ניתן לבצע מגוון משימות.

כאמור, בכדי שנוכל לבנות מודל שפה או לאמן כל מודל אחר, נצטרך קודם כל לייצג את הטקסט בצורה כלשהיא. בשיטה שהוצגה לעיל, כל מילה (Token) מיוצגת באמצעות צירוף אותיות. כך למשל המילה הראשונה במשפט

שראינו מורכבת מצירוף של האותיות T, a, k, e. כפי שיפורט בהמשך, נוכל גם לדבר על ייצוג למילים עצמן כך שכל יחידה אטומית תהיה מילה וצירוף של היחידות האלה ירכיבו ייצוג של משפט.

באופן סכמטי, ניתן לתאר את משימת עיבוד השפה מקצה לקצה באופן הבא:




איור 11.1 תהליך פיתוח אלגוריתם של מודל שפה: א. מייצגים את הטקסט בצורה כלשהיא (ניתן כמובן לקחת ייצוג קיים שנבנה על בסיס דאטהסט אחר). ב. מאמנים מודל שמקבל כ-input את הטקסט אותו ייצגנו בדרך כלשהיא בשלב הראשון, והמודל מוציא output מסוים. למודל כזה יכולות להיות ארכיטקטורות שונות. ג. באמצעות המודל המאומן ניתן לבצע משימות קצה שונות.

בהמשך פרק זה נתמקד בשכבה התחתונה של התרשים: נתאר מספר שיטות מרכזיות לייצוג טקסטים, ונראה כיצד ניתן לאמן מודלי שפה שונים היכולים לבצע כל מיני משימות.

### 10.1.1 Basic Language Models

מודל השפה הראשון אותו נציג הינו N-Grams, שהינו מודל סטטיסטי המניח שההסתברות למילה הבאה תלויה אך ורק ב-$N-1$ המילים שקדמו לה בסדרה. הנחה זאת נקראת "הנחת מרקוב" (Markov assumption), ובאופן כללי יותר, מודלי מרקוב (או שרשראות מרקוב במקרה הדיסקרטי) מסדר $n$ הם מודלי הסתברות המניחים שניתן לחזות הסתברות של אירוע עתידי $T$ בהתבסס על האירועים שהתרחשו ב-$n$ נקודות הזמן שקדמו למאורע $T$ מבלי להתחשב באירועי עבר רחוקים מדי.

מודל ה-N-Gram הפשוט ביותר נקרא unigram. במודל זה אנחנו חוזים את המילה הבאה לפי התדירות של המילה עצמה ב-corpus מבלי להתחשב במה שקדם לה. כמובן שחיזוי כזה הינו בעייתי, מכיוון שהמילה הבאה **חייבת להיות תלויה במילים שקדמו לה**, וכמו כן יהיו מילים כמו the שמופיעות באופן תדיר בטקסט שאינן בהכרח משפיעות על ההקשר. לכן, נסתכל על מודל קצת יותר מורכב הנקרא bigram, המתייחס למילה האחרונה הקודמת למילה הנחזית. במודל bigram אנחנו מניחים את המשוואה הבאה:

$$P(w_n|w_{n-1}, w_{n-2}, \dots, w_1) = P(w_n|w_{n-1})$$

כלומר, רק למילה האחרונה יש השפעה על החיזוי של המילה הבאה, וכל המילים שלפניה הן חסרות השפעה על ההתפלגות של המילה הנחזית (וממילא גם על המשך המשפט). באופן כללי, מודל N-grams מניח את המשוואה הבאה:

$$P(w_n|w_{n-1}, w_{n-2}, \dots, w_1) = P(w_n|w_{n-1}, w_{n-2}, \dots, w_{n-N}), N \leq n$$

ניקח לדוגמא מספר משפטים וננתח אותם בשיטת bigram:

- I know you
- I am happy
- I do not know Jonathan

נניח ונרצה לבחון את ההסתברות שהמילה Jonathan היא המילה הבאה אחרי הסדרה I do not know. ראשית נגדיר את המילון, המכיל את כל המילים האפשריות בשפה:

$$V = \{I, know, you, am, happy, do, not, Jonathan\}$$

כעת נוכל להעריך את ההסתברות לכך על ידי ספירת כמות הפעמים שהצמד $\langle know|Jonathan\rangle$ מופיע בטקסט, ולנרמל בכמות הפעמים ש-know מופיע בטקסט עם מילה כלשהי (כולל הפעמים שמופיע עם Jonathan). באופן פורמלי נגדיר את המשוואה הבאה:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_{w\in V} C(w_{n-1}w_n)}$$

כאשר האות $C$ מסמנת את מספר הפעמים שצמד מסוים בטקסט. ניתן לשים לב שהביטוי במכנה למעשה סופר את כמות הפעמים ש-$w_{n-1}$ קיים בטקסט, ולכן נוכל לפשט את המשוואה האחרונה ולרשום במקומה:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

אם כך, ההסתברות לכך שהמילה Jonathan תופיע אחרי המילה know היא:

$$P(Jonathan|know) = \frac{1}{2}$$

באופן דומה ניתן לנסות לשערך את ההסתברות של המילה הבאה על סמך יותר ממילה אחת אחורה, למשל לקחת 2 מילים אחורה. מודל זה נקרא trigram, וכפי שנאמר לעיל, באופן כללי מודל המסתמך על $N-1$ מילים לצורך חישוב ההסתברות של המילה הבאה בטקסט נקרא N-gram.

המודל המרקובי סובל ממספר בעיות:

1. טבלת ההסתברויות המתקבלת מאוד דלילה. כיוון שה-corpus (שהוא סט האימון) הינו בגודל מוגבל, לעולם לא נוכל לראות את כל הקומבינציות הקיימות, מה שיוביל להערכה של הסתברות 0 עבור צמדים שלא מופיעים בטקסט האימון.

2. כאשר נרצה לחזות בעזרת מודל זה את ההסתברויות על טקסט חדש, כנראה שנתקל במילים שלא נתקלנו בהן בטקסט האימון ולכן לא נוכל להגדיר את ההסתברות עבור ה-N-Grams המכילים מילים אלו.

בשביל להתמודד עם בעיות אלו – באופן מלא או חלקי – נוכל להפעיל שיטות החלקה (Smoothing) על ההסתברויות של צמדים שהופיעו מעט/כלל לא הופיעו בטקסט האימון. שיטות החלקה במהותן לוקחות קצת מסת הסתברות מהאירועים השכיחים (כלומר, ה-N-Grams המופיעים הכי הרבה) ו"תורמות" לאירועים פחות שכיחים. ישנן מגוון שיטות להחלקת הסתברות ונסקור כעת חלק מהן.

### Laplace Smoothing

הדרך הפשוטה ביותר לבצע החלקה היא להוסיף מספר קבוע $\mathcal{K}$ לכל האירועים. באופן הזה אנו דואגים לתת משמעות גם לצירופים נדירים שמופיעים מספר מועט של פעמים (או אפילו לא מופיעים בכלל). אם למשל יש צירוף שמופיע רק פעם אחת ולעומתו יש צירוף שמופיע 1000 פעמים, אז הוספת הקבוע $\mathcal{K}$ משמעותה שעכשיו נמנה את הצירוף הראשון ככזה המופיע $(1+\mathcal{K})$ פעמים וביחס לצירוף השני נתייחס ככזה שהופיע $(1000+\mathcal{K})$ פעמים. הוספה זו כמעט ואינה משפיעה על הצירוף השני, אך היא יכולה להכפיל פי כמה את ההסתברות של הצירוף הראשון. כמובן שכאשר אנחנו מתעסקים עם הסתברויות נרצה לאחר הוספת הקבוע במונה לתקן גם את מקדם הנרמול. באופן פורמלי, החלקת לפסל מוגדרת באופן הבא:

$$P_{Laplace}(w_n|w_{n-1}) = P_{Add-\mathcal{K}}(w_n, w_{n-1}) = \frac{C(w_{n-1}w_n)+\mathcal{K}}{\sum_w C(w_{n-1}w_n)+\mathcal{K}} = \frac{C(w_{n-1}w_n)+\mathcal{K}}{C(w_{n-1})+\mathcal{K}\cdot|V|}$$

כאשר $|V|$ הוא גודל המילון (כמות המילים המופיעה ב-Corpus). היתרון בשיטה זאת היא הפשטות שלה, אך עם זאת יש לה חסרון בולט הנובע מכך שהיא משנה באופן משמעותי את ההסתברויות של מאורעות תדירים ובכך בעצם משבשת את ההסתברויות שנלמדות מהטקסט. כפועל יוצא יותר מדי מסת הסתברות עוברת מהמאורעות השכיחים למאורעות עם הסתברות נמוכה. השיטה הבאה מנסה לתקן בדיוק את זה.

כמובן שניתן לבחור כל ערך $\mathcal{K}$ שרוצים ואותו להוסיף למכנה. באופן זה כן ניתן לשלוט ברמה מסוימת עד כמה להגדיל את ההסתברויות לקומבינציות שאינן מופיעות בסט האימון על חשבון הקומבינציות השכיחות (כתלות ב-$\mathcal{K}$). בחירה למשל של $\mathcal{K}=0.5$ מאפשרת למזער את העיוות היכול להתקבל משינוי הספירה.

### Backoff and Interpolation

שיטת ההחלקה בסעיף הקודם נותנת פתרון לקביעת ההסתברות של מאורעות המקבלים הסתברות 0, וישנן גישות נוספת שנותנת מענה למגבלות האלה. נניח ואנחנו משתמשים במודל trigram, מודל המניח שההסתברות למילה הבאה תלויה בשתי המילים שבאות לפניה. אם נבצע את השערוך של כל שלישיה לפי הגדרה (=ספירת המופעים שלהם בטקסט), כל שלישית מילים שאינה מופיעה כרצף בטקסט תקבל הסתברות 0, ובעצם תקיים את המשוואה:

$$P(w_n|w_{n-1}, w_{n-2}) = 0$$

בכדי להימנע מלתת הסתברות 0 בכזה מצב, ניתן להיעזר גם בהסתברויות של הbigram וה-unigram:

$$P(w_n|w_{n-1}), P(w_n)$$

שיטת ה-backoff מציעה לקחת את כל המקרים בהם קיבלנו 0 ולשערך אותם מחדש באמצעות מודל bigram. לאחר מכן ניקח את כל המילים שעדיין נותרו עם הסתברות 0 (כלומר, צמדי מילים שאינן מופיעים בטקסט) ונשערך אותם באמצעות unigram. באופן הזה מקווים להגיע למצב בו מספר המילים בעלי הסתברות 0 היא קטנה (עד אפסית), כיוון ששימוש במודלים יותר פשוטים מאפשר שימוש במגוון רחב יותר של קומבינציות הקיימות בטקסט. שיטה דומה נקראת Interpolation, ובה במקום לקחת את הרצפים בעלי הסתברות 0 ולשערך אותם במודל הנמצא בדרגה אחת מתחת (למשל – לשערך בעזרת bigram במקום trigram), המודל מראש מתייחס לקומבינציה כלשהי של ה-unigram, bigram and trigram (למשל קומבינציה לינארית). בשיטה זו גם מקרים שאינם בעלי הסתברות 0 נעזרים ב-unigram and bigram לשערוך ההסתברות ובניית מודל השפה.

### 10.1.2 Word representation (Vectors) and Word Embeddings

עד כה, המילים השונות היו מיוצגות בעזרת אותיות. כך לדוגמא, המילה כלב תיוצג על ידי צירוף האותיות DOG בעוד שהמילה חתול תיוצג על ידי הצירוף CAT. ייצוג זה מכיל מאפיינים סינטקטיים (=תחביריים) של השפה, קרי איך המילה נכתבת. עם זאת, ייצוג זה חסר מאוד, כיוון שהוא יתקשה בלמידת ייצוג של מאפיינים סמנטיים. דוגמא להבנה סמנטית של שפה היא ההבנה שכלב וחתול הן לא מילים נרדפות, אך הן כן קשורות אחת לשנייה באופן כלשהו – שתיהן מייצגות חיית מחמד שאנשים מגדלים בביתם. מודל המבוסס על ייצוג טקסטואלי של השפה לא יצליח להגיע להבנה סמנטית שלה, ולכן נרצה שהייצוג שלנו יהיה מספיק עשיר ויכיל הן מאפיינים תחביריים והן מאפיינים סמנטיים של השפה.

בפועל, נוכל למפות את הייצוג הטקסטואלי לייצוג נומרי בצורה פשוטה בעזרת One-Hot vectors – מערך בגודל המילון שלנו, המייצג כל מילה במילון בעזרת 1 באיבר המתאים במערך. לדוגמא נתון המילון הבא:

| Index | Word |
|---|---|
| 0 | Dog |
| 1 | Cat |
| 2 | Lion |

נוכל לייצג את המילים השונות בעזרת וקטורים באורך 3, באופן הבא:

$$\text{Dog} \rightarrow [1, 0, 0]$$

$$\text{Cat} \rightarrow [0, 1, 0]$$

$$\text{Lion} \rightarrow [0, 0, 1]$$

כך, לכל מילה יהיה ייצוג וקטורי ייחודי. עם זאת, ייצוג פשטני זה הינו בעייתי מכיוון שיהיה קשה ללמוד ממנו מאפיינים סמנטיים. כדי להבין את הסיבה לכך ראשית יש להגדיר את מושג הדמיון בעולם של וקטורים.

**Cosine similarity**

מעבר לייצוג וקטורי של מילים דורש מאיתנו להגדיר דמיון בין וקטורים במרחב שנוצר. אחת מההגדרות הפופולריות לדמיון בין וקטורים היא ה-Cosine similarity. כמו רוב השיטות לחישוב דמיון בין וקטורים, גם פונקציית דמיון זו מבוססת על מכפלה פנימית של וקטורים. נניח ונתונים 2 וקטורים $v, w$, שניהם בעלי אותו הממד $N$. המכפלה הפנימית (dot product) בין הווקטורים האלה מוגדרת באופן הבא:

$$v \cdot w = v^T w = \sum_{i=1}^{N} v_i w_i$$

באמצעות הגדרה זו ננסה לאמוד את הדמיון בין הייצוג של המילים כלב וחתול במרחב הווקטורי שנוצר בעקבות ייצוג בעזרת One-Hot vectors. כאמור, הייצוג של המילה חתול במרחב זה הוא: $\text{Cat} \to [0,1,0]$, בעוד שהייצוג של המילה כלב באותו מרחב הינה: $\text{Dog} \to [1,0,0]$. לכן, לכן המכפלה הפנימית במרחב זה תהיה:

$$[0,1,0] \cdot [1,0,0] = 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 0 = 0$$

תוצאה זו מדגימה את הבעייתיות בייצוג פשטני זה, מכיוון שהינו רוצים שווקטורים אלה כן יהיו דומים במובן מסוים, ולא שהתוצאה תהיה 0, המלמדת על כך שכלל אין קשר בין שתי המילים. למעשה בשיטה זו הדמיון בין ייצוגים של כל זוג מילים יהיה אפס.

**Bag-of-words**

דרך אחת לייצר קשר בין מילים בעלות קשר סמנטי היא להתייחס לא רק למילים בודדות אלא גם להקשרים בתוך המשפט עצמו. באופן הזה נוכל להגדיר ייצוג וקטורי של מילה על ידי ספירה של כמות הפעמים שמילה אחרת נמצאת איתה באותו ההקשר. שיטה זאת נקראת Bag-of-Words, ולפני שנוכל להדגים אותה, נצטרך להגדיר מהו הקשר של מילה. באופן פשוט הקשר של מילה זה המשפט בו היא מופיעה, אך ניתן גם לקחת רק חלון של מספר מילים מתוך המשפט (לרוב אורך החלון קטן מאורך המשפט). לדוגמא, נניח ונתונים לנו הטקסט והמילון הבאים:

טקסט:

- [The dog is a domestic mammal, not wild mammal], is a domesticated descendant of the wolf, characterized by an upturning tail.
- [The cat is a domestic species of small mammal], It is the only domesticated species in the family.

מילון:

| | | | |
|---|---|---|---|
| 1. The | 5. Mammal | 9. Animal | 13. Tail |
| 2. Is | 6. Not | 10. Descendant | 14. Cat |
| 3. A | 7. Natural | 11. Dog | 15. Species |
| 4. Domestic | 8. Wild | 12. Wolf | 16. Small |

כעת נרצה לייצג את המילים Dog ו-Cat בשיטת Bag-of-words באמצעות חלון באורך 7 (=כמות המילים לפני ואחרי המילה שנרצה לייצג. חלון זה מסומן בטקסט בסוגריים מרובעות בצבע אדום). נבנה מטריצה עם מספר עמודות כאורך המילון ומספר שורות כמספר המילים אותן נרצה לייצג (לרוב מספר השורות יהיה כמספר המילים במילון, אך לשם הדוגמא נציג כאן טבלה קטנה יותר). עבור כל מילה, נבדוק כמה פעמים היא נמצאת בטקסט באותו חלון יחד עם מילים אחרות:

| Index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dog | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Cat | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

כעת נוכל לראות שהמכפלה הפנימית בין שני הווקטורים המייצגים את המילים Dog ו-Cat אינה 0. עם זאת, ישנן שתי בעיות נוספות הנובעות מפשטות פתרון זה:

1. מילים פחות משמעותיות כמו is, the, a נכללות בספירה למרות שהן לא בהכרח מוסיפות מידע משמעותי לייצוג.
2. מילים המופיעות בטקסט לעיתים רחוקות יהיו בעלות ייצוג וקטורי מאוד דליל, מה שמגדיל שוב את הסיכוי למכפלה פנימית בעלת ערך קטן מאוד (עד אפס) עם רוב הווקטורים האחרים.

**TF-IDF**

הדרך הטבעית לפתרון של הבעיה הראשונה – רעש שנוצר ממילים בעלות תדירות גבוהה שאינן תורמות בייצוג, היא סינון של המילים האלה מהמילון. אולם, פתרון זה אינו ישים, כיוון שהתדירות של מילים משתנה בין תחומים/דומיינים שמהם נלקח הטקסט. לכן פותחה שיטת ייצוג הנקראת TF-IDF, ומטרתה להפחית את רעש זה באופן אוטומטי.

בשיטת TF-IDF, מגדירים פונקציה ניקוד אחרת בעלת שני רכיבים. במקום להסתכל על חלון יחיד מסביב לכל מילה, ניתן להסתכל על כל המילים בחלונות הנוצרים מסביב למילה המיוצגת, ולתת לה ניקוד לפי התדירות של אותה מילה. באופן פורמלי, $tf$ מוגדר בצורה הבאה:

$$tf = \log(C(w,c)+1)$$

משמעות הביטוי היא שאנו סופרים כמה המילה $c$ מופיעה בקונטקסט (=בחלון) של המילה המיוצגת $w$ (על התוצאה מפעילים $log$ בשביל rescaling של ערכים גבוהים מאוד).

עד כה השיטה אינה שונה במהותה מספירה כמו שראינו בBag-of-words, אך מה שעושה את TF-IDF שונה הוא הביטוי השני. הביטוי $idf$ מוגדר בצורה הבאה:

$$df = \text{count}(w \in \text{Context})$$

$$idf = \log\left(\frac{N}{df}\right)$$

המונח $df$ מייצג את כמות הפעמים שהמילה $w$ מופיעה בקונטקסטים אחרים, בעוד ש-N מייצג את כמות המילים במילון. נשים לב שאם מילה מסוימת, למשל the, מופיעה בכל הקונטקסטים של כל המילים במילון, אז הביטוי בתוך הלוג יהיה 1 ולכן ה-$idf$ יהיה 0. ולעומת זאת, אם מילה מסוימת מופיעה אך ורק בקונטקסט יחיד, אז הערך שיהיה בתוך הלוג יהיה $N$.

לבסוף, TF-IDF מוגדר באופן הבא:

$$TF-IDF = tf \cdot idf$$

מדד משוקלל זה מצליח עבור ייצוג של מילה מסוימת לתת משקל גדול למילים אחרות הנמצאות איתה בקונטקסט באופן תדיר אך אינן נמצאות בקונטקסט של מילים אחרות.

### PPMI - Positive Pointwise Mutual Information

כעת נרצה לפתור את הבעיה השנייה – בעיית הייצוג הדליל למילים שאינן תדירות בטקסט. שיטת PPMI מגדירה פונקציית ניקוד המחשבת את היחס בין הסיכוי של שתי המילים להמצא יחד לעומת הסיכוי לראותן בנפרד – $\frac{P(x,y)}{P(x)P(y)}$.
כעת בשביל לחשב את הערך של התא המייצג את המילה $y$ בוקטור הייצוג של המילה $x$ נשתמש בהסתברות הנ"ל ונפעיל לוג. אם ההסתברות לראות את המילים $x, y$ ביחד שווה לכפל ההסתברויות לראות כל אחת לחוד, נקבל שערך הביטוי הוא $\log(1)$ כלומר 0. לעומת זאת, אם הסיכוי לראות את המילים האלו ביחד גדול מהסיכוי שיראו אותם לחוד אז נקבל ערך הגדול מ-1. ישנו מקרה נוסף, בו הסיכוי לראות את הביטויים ביחד קטן מהסיכוי לראות אותם לחוד. במקרה זה הביטוי שנקבל יהיה קטן מ-1 ולכן הלוג יהיה שלילי, אך מכיוון שהערכים השלילים נוטים להיות לא אמינים (אלא אם הטקסט שלנו גדול מספיק), נוסיף עוד אלמנט קטן לפונקציית החישוב:

$$PPMI = \max\left(\log\frac{P(x,y)}{P(x)P(y)}, 0\right)$$

באופן זה נוכל לנרמל את הערך הנמוך עבור מילים נדירות בטקסט.

### Word2Vec

השיטות שראינו עד כה לחישוב וקטורי הייצוג של המילים מאפשרות לנו לקודד מאפיינים סמנטיים בייצוג המילים. עם זאת, יש כמה חסרונות לשיטות אלו: ראשית, הן יוצרות וקטורים מאוד דלילים, ובנוסף לכך גודל הווקטורים תלוי בכמות המילים שיש לנו במילון, מה שיוצר וקטורים גדולים שמכבידים על החישובים במשימות השפה השונות. למשל – ראינו קודם שניתן לייצג מילה באמצעות וקטור שכולו אפסים למעט תא אחד עם הערך 1 במיקום ייחודי לכל מילה. עבור שפה עם אלפי מילים ואף יותר מכך, כל וקטור המייצג מילה הוא באורך עצום, ועם זאת הוא מאוד דליל כיוון

שיש בו רק מספר תאים מועט שערכם שונה מ-0. לכן, נרצה לפתח שיטה שתיצור וקטורי ייצוג דחוסים (dense) בעלי ממד קטן יותר.

שיטת Word2Vec הינה שיטת Self-Supervised שפותחה למטרת יצירה של וקטורי ייצוג דחוסים של מילים. הפרדיגמה של למידת Self-Supervised "דומה" ללמידה מונחית (supervised learning) רגילה, אך התיוגים אינם נתונים אלא נוצרים באופן אוטומטי מתוך הדאטה הלא מתויג. באופן זה ניתן לאמן מודלים עם כמות גדולה של דאטה לא מתויג בצורה יעילה וללא צורך בתיוג (שעלול להיות מאוד יקר). בהקשר זה, אלגוריתם Word2Vec משתמש ב-Self-supervision באופן של Skip-Grams-With-Negative-Sampling, או בקיצור –SGNS. הרעיון מאחורי גישה זו הוא להגדיר בעיית סיווג שמטרתה לחזות מה ההסתברות של מילים שונות להיות בקונטקסט של מילה נתונה. בסוף האימון לוקחים את המשקלים שנוצרו בעקבות תהליך אימון המשימה הראשית, והם יהיו הייצוג של המילה. בכדי להבין זאת לעומק, נבחן טקסט פשוט יחסית בעזרת אלגוריתם Word2Vec. נניח ונתון המשפט הבא כחלק מהטקסט האימון שלנו:

Folklore, legends, myths, and fairy tales have followed childhood through the ages.

ראשית נקבע את אורך החלון (=מספר המילים עליהן מסתכלים בסביבות כל מילה) – 3. כעת נעבור על הטקסט וניצור תיוגים בין כל מילה במילון ליתר המילים. למשל עבור המילה tales נוסיף לדאטה סט שלנו דוגמאות חיוביות של המילים שנמצאות בקונטקסט עם tales. בנוסף, בכדי למנוע התנוונות של כל הייצוגים לוקטור בודד, נצטרך "להראות" למודל איך נראות דוגמאות שליליות ולכן נשתמש בשיטה הנקראת negative sampling. בשיטה זו דוגמים מהמילון בהסתברות פרופורציונלית לתדירות המילה (עם תיקון קטן שנותן קצת יותר סיכוי למילים נדירות) את המילים שישמשו אותנו **כדוגמאות שליליות**. הרעיון מאחורי תהליך זה הוא שכאשר יש לנו מילון גדול המילים שנגריל לא יהיו קשורות למילה שעבורה אנחנו יוצרים את הדוגמאות השליליות.

Folklore, legends, [myths and fairy [tales] have followed childhood] through the ages.

| word | context | Label |
|---|---|---|
| tales | myths | + |
| tales | and | + |
| tales | fairy | + |
| tales | have | + |
| tales | followed | + |
| tales | childhood | + |
| tales | great | - |
| tales | April | - |
| tales | the | - |
| tales | young | - |
| tales | orphan | - |
| tales | dishes | - |

כך נוכל ליצור דאטה מתויג עבור משימת הסיווג, ונוכל להשתמש בתיוגים אלה בשביל לאמן את המודל. הכוונה במשימת סיווג בהקשר זה מעט שונה מסיווג במובן הפשוט של המילה: מטרת המודל היא שבהינתן מילה $w$ (במקרה שלנו tales), נרצה שהייצוג של מילים המופיעות באותו קונטקסט עם כל מילה (במקרה שלנו – אלו שמופיעות עם

tales באותו חלון) יהיה קרוב לייצוג של tales בעוד שמילים שאינן מופיעות באותו קונטקסט יהיו בעלות ייצוג "שונה" (מבחינת מכפלה פנימית או cosine similarity). נניח שבחרנו שהייצוג של כל מילה יהיה וקטור בגודל 100. נתחיל את התהליך כך שכל מילה מקבלת וקטור רנדומלי. נסמן את וקטור הייצוג של המילה $w$ ב-$e_w$ ואת הווקטור של מילת קונטקסט כ-$e_c$. השאיפה היא שהמכפלה הפנימית של וקטורי הייצוג של המילים בקונטקסט של $w$ יהיה גבוה יחסית, בעוד שהמכפלה הפנימית של וקטורי יצוג של מילים שאינן מופיעות באותו קונטקסט יהיה נמוך. כאמור לעיל, Cosine similarity (המטריקה המגדירה דמיון בין וקטורים) היא בעצם מכפלת פנימית של הווקטורים (=מכפלת dot עם נרמול). כעת נוכל להגדיר בעיית logistic regression באופן הבא:

$$p(+|w,c) = \sigma(e_w, e_c) = \frac{1}{1+e^{-e_w e_c}}$$

$$p(+|w,c) = 1 - p(+|w,c)$$

ובהתאם לכך פונקציית המטרה (Loss) תוגדר באופן הבא:

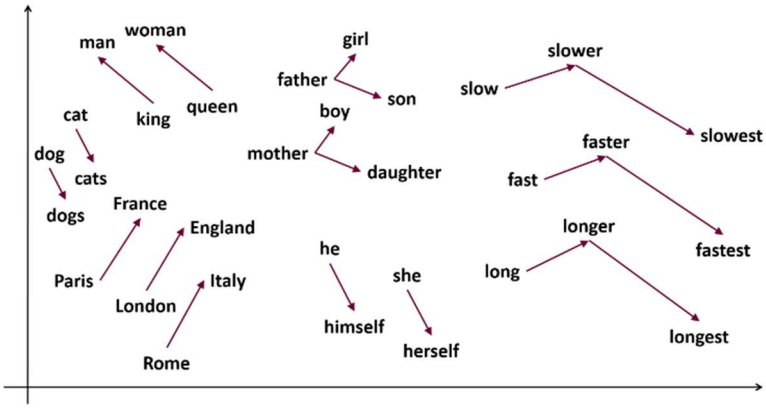
$$L = -\log\left[p\left(+\middle|w, c_{pos}\right)\Pi_{i=1}^{k} p\left(-\middle|w, c_{neg_i}\right)\right]$$

$$= -\left[\log\left(p\left(+\middle|w, c_{pos}\right)\right) + \log\left(\Pi_{i=1}^{k} p\left(-\middle|w, c_{neg_i}\right)\right)\right]$$

$$= -\left[\log\left(\sigma(e_w \cdot e_{c_{pos}}\right) + \sum_{i=1}^{k} \log\left(1 - \sigma\left(e_w \cdot e_{c_{neg_i}}\right)\right)\right]$$

נשים לב שאנחנו מניחים אי תלות בייצוג של הדוגמאות השליליות. בכך שנבצע מינימיזציה לפונקציית מטרה זו נגרום למכפלה הפנימית בין הייצוג של מילה לבין מילת הקונטקסט להיות גבוהה ובו בזמן למכפלה הפנימית בין וקטורים שאינם בקונטקסט להיות נמוכה. כך הייצוג של המילה tales יהיה "דומה" (=קרוב במונחים של cosine similarity) לייצוג של המילים בקונטקסט ושונה מייצוגן של המילים שאינן בקונטקסט. את תהליך המינימיזציה במשך האימון נוכל לבצע באמצעות stochastic gradient descent.

אחת התוצאות היפות והחשובות של שיטת Word2Vec ניתנת להמחשה על ידי פריסת וקטורי הייצוג בממד נמוך. בעזרת שיטות מתקדמות להורדת ממד (כפי שהוסבר בהרחבה בפרק 2), ניתן לצייר בדו-ממד או תלת ממד את וקטורי הייצוג של המילים לאחר האימון.




איור 11.2 וקטורי הייצוג של מילים לאחר ביצוע embedding באמצעות word2vec. צמדי מילים בעלי משמעות דומה מיוצגים על ידי וקטורים באותו כיוון.

נוכל להבחין שהמרחב הווקטורי מקודד מאפיינים סמנטיים. לדוגמא, ניתן לראות שהווקטור המחבר בין וקטורי הייצוג של המילים $King, Man$ מקביל ובעל אורך דומה לווקטור המחבר בין הייצוג של $Queen, Woman$. דוגמה נוספת – הווקטור בין שם של ארץ לעיר הבירה שלה מקביל ובעל אורך דומה לקטור שבין ארץ אחרת ועיר הבירה המתאימה. כמובן שניתן ללמוד מכך על קשרים סמנטיים, כמו למשל שהיחס בין $King, Man$ זהה ליחס שבין $Queen, Woman$.

### 10.1.3 Contextual Embeddings

מנגנוני בניית ייצוגי מילים (embedding) שראינו עד כה למדו ייצוג **סטטי** עבור כל מילה. אך דבר זה יכול להיות בעייתי מכיוון ששפה טבעית היא דינמית ותלוית הקשר, ולאותה מילים יכולה להיות כמה פירושים. בשביל להבין את הבעייתיות נסתכל על המשפטים הבאים:

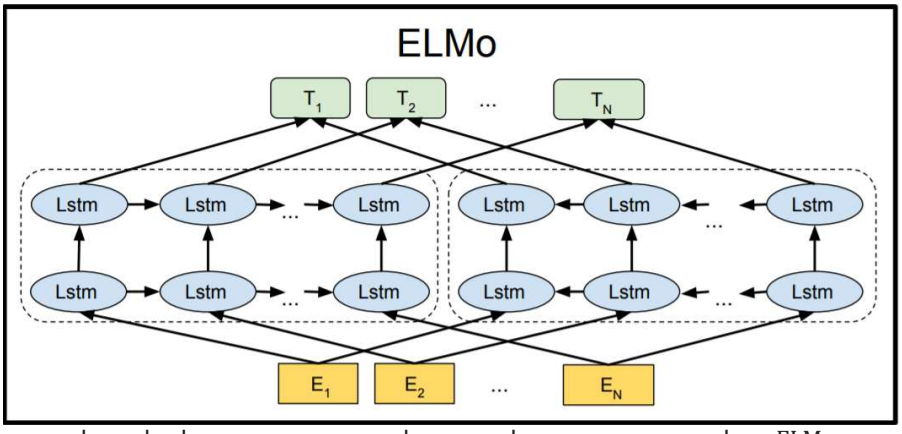
1. We need to **book** the flight as soon as possible
2. I read the **book** already

למילה book יש כפל משמעות במשפטים האלו. במשפט הראשון המילה משמשת כפועל ובמשפט השני כשם עצם עם תפקיד סמנטי שונה במשפט. אם כך ברור שההקשר שבו המילה מופיעה משפיע על המשמעות שלה אך מנגנוני embedding כמו word2vec ייצגו את המילה באותו וקטור ייצוג עבור שני המופעים.

לכן נרצה לפתח מנגנון embedding עבור המילים, שבאמצעותו וקטור המייצג מילה יהיה תלוי בהקשר בו היא מופיעה.

**Embeddings from Language Models (ELMo)**

אחת השיטות הראשונות שהציגה טכניקה ללמידת ייצוג תלוי הקשר למילים הינה Embeddings from Language Models, או בקיצור ELMo – ארכיטקטורה הבונה לכל מילה ייצוג שתלוי בהקשר שלה בתוך המשפט. הרעיון במודל זה הוא לקחת ייצוג של מילה, להוסיף לו מידע נוסף מההקשר של המילה במשפט ולקבל ייצוג חדש התלוי גם בהקשר שלה. בניסוח אחר ניתן לומר ש-ELMo הינה פונקציה המקבלת משפט שבו כל מילה מיוצגת בדרך כלשהיא (למשל – Word2Vec), ומוסיפה לייצוג זה גם את ההקשר של המילה בתוך כלל המשפט. בפועל זה נעשה על ידי משימת אימון מודל שפה דו-כיווני – המודל לומד לחזות גם את המילה הבאה בטקסט וגם את המילה הקודמת, ובכך הוא לומד לתת למילה גם את ההקשר שלה. ארכיטקטורת הרשת נראית כך:




איור 11.3 ארכיטקטורת ELMo. הקלט הינו משפט המיוצג כלשהוא, והפלט הוא אותו משפט אך כל מילה קיבלה מידע נוספת על ההקשר שלה וכעת מיוצגת באופן חדש. תהליך האימון והוספת ההקשר בין המילים נעשה באמצעות שכבות של רכיבי LSTM.

כפי שמתואר בפרק 6.2.1, כל בלוק של LSTM מקבל כקלט שני רכיבים המייצגים את ההיסטוריה של המשפט עד הנקודה בה מופיעה המילה של ה-timestamp הנוכחי $(c_t, h_t)$, וקלט נוסף של האיבר הנוכחי בסדרה, שבמקרה שלנו זה המילה הנוכחית $(x_t)$. המוצא של ה-LSTM הינו ייצוג חדש המשקלל את רכיבי ההיסטוריה יחד עם הייצוג הנוכחי של המילה.

בדומה לשיטות אחרות ליצירת ייצוג וקטורי למילים, אנחנו מאמנים את המודל בעזרת משימת מידול שפה וחוזים את המילה הבאה בהינתן המילים הקודמות. אך בשונה מאלגוריתמים אחרים, ELMo משתמש בארכיטקטורה דו כיוונית, כך שבתהליך האימון משולבת משימת שפה נוספת המנסה לחזות את המילה הקודמת בהינתן הסוף של המשפט. הארכיטקטורה של ELMo בנויה ממספר שכבות של LSTM שמורכבות זו על גבי זו, ולפי כותבי המאמר השכבות התחתונות מצליחות ללמוד פיצ'רים פשוטים (למשל מאפיינים סינטקטיים למיניהם), בעוד שהשכבות העליונות לומדות פיצ'רים מורכבים (למשל מאפיינים סמנטיים, כמו משמעות המילה בהקשר).

לאחר תהליך האימון ניתן להקפיא את הפרמטרים של המודל ולהשתמש בו עבור משימות אחרות. הכותבים מציעים לשרשר את הייצוג הווקטורי של LSTM בכל שכבה ככה שיכיל אינפורמציה גם מתחילת המשפט עד המילה הנבדקת וגם מסוף המשפט עד המילה הנבדקת. מה שקורה בפועל זה שהשכבות החבויות (hidden layers) של LSTM הם עצמם מהווים את ייצוגי המילים בשיטת ייצוג כזו, כלומר כל מילה במשפט מיוצגת על ידי התא המקביל בשכבת ה-LSTM שמעליה. בנוסף הם מוסיפים מספר פרמטרים קטן שמאפשר כיול (Fine tune) עבור משימה ספציפית. כך לדוגמא נוכל להתאים את הייצוג של המילים למשימת סיווג של משפט לעומת משימת תיוג של ישויות במשפט.

פה חשוב להדגיש נקודה מרכזית – בסופו של דבר התוצר של ELMo הינו **מודל שפה הלוקח ייצוג של טקסט והופך אותו לייצוג תלוי הקשר**. שכבות ה-LSTM השונות מאמנות מודל שפה על מנת ליצור ייצוג חדש עבור המילים,

המתייחס גם להקשר. לאחר סיום האימון של מודל השפה (pre-training), ניתן לקחת אותו ולבצע transfer learning, כלומר להשתמש בייצוגים שהוא מפיק גם למשימות אחרות על ידי הוספת שכבות בקצה. לאחר פרסום המאמר, Sebastian Ruder (חוקר NLP מפורסם) טען כי:

"It is very likely that in a year's time NLP practitioners will download pretrained language models rather than pre-trained word embeddings"

כלומר, כעת מי שירצה לבצע משימת שפה כבר לא יתבסס רק על ייצוג סטטי של המילים אלא הוא יסתמך על מודל שפה מאומן שיודע לקחת ייצוג התחלתי של מילים ולהפוך אותם לייצוגים קונטקסטואליים. ELMo ועוד מאמרים רבים אחריו אימנו מודלי שפה מאומנים שניתם לקחת אותם ולהשתמש בהם עבור משימות קצה שונות על ידי הוספה של כמה שכבות וכיול המודל.

**Bidirectional Encoder Representations from Transformers (BERT)**

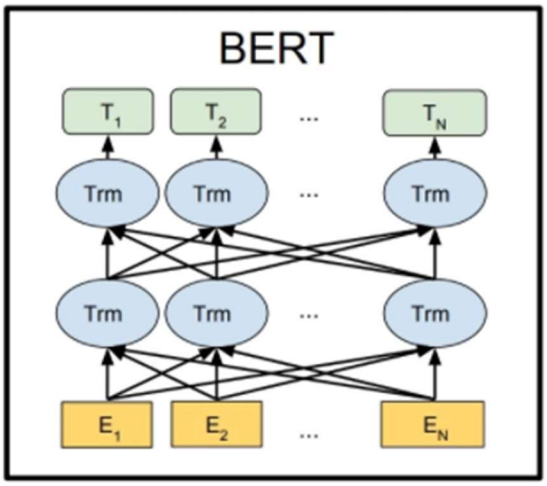
כאמור, הרעיון של ELMo הוא לייצר contextual embedding, ובכך לקבל מודל שפה המאפשר לקחת ייצוג וקטורי של מילים ולהעשיר אותו במידע על ההקשר של כל מילה בטקסט. למרות ש-ELMo משתמש בשני הכיוונים של המשפט (חוזה את המשך המשפט מתחילתו ואת תחילת המשפט מסופו) הוא אינו לומד משני הכיוונים בתהליך אחד, אלא צריך לחלק את הלמידה לשני חלקים שונים. בנוסף, כפי שהוסבר בהקדמה לפרק 8, כאשר מתעסקים עם סדרות ארוכות של מילים, וקטור הייצוג של כל איבר נהיה בעייתי כיוון שהוא מוגבל ביכולת שלו להכיל קשרים בין מספר רב של איברים. במילים אחרות, כאשר רוצים להוסיף לווקטור הייצוג של מילה מסוימת קשר למילים רחוקות, אנו מאלצים את הייצוג "לזכור" מידע רב, אך הייצוג הווקטורי של המילים ב-ELMo אינו מצליח לעשות זאת בצורה מספיק טובה. לכן, על אף הצלחת גישה זו במשימות שונות, היא עדיין התקשה במשימות בהן נדרשת יכולת לנתח טקסטים ארוכים (כמו למשל משימה של summarization). בנוסף לכך, האלגוריתם יחסית איטי, כיוון שכל פעם הוא מסתכל על מילה אחת בלבד.

בכדי להתמודד עם בעיות אלו וליצור ייצוג המסוגל להכיל מידע איכותי גם ברצפים ארוכים, ניתן להשתמש ב-attention (-מוסבר בהרחבה בפרק 8). אחד השימושים הראשונים במנגנון ה-attention עבור משימת עיבוד שפה היה בטרנספורמרים, ובפרט בארכיטקטורת רשת הנקראת BERT, המבוססת על ה-encoder של הטרנספורמר המקורי. שימוש זה היווה פריצת דרך בתחום, וכיום ברוב המוחלט של המחקר והפיתוח בתחום ה-NLP משתמשים בארכיטקטורת רשת מבוססת attention. למעשה, BERT מציע שיטה לבניית ייצוג קונטקסטואלי של מילים הבא להתמודד עם החולשות הקיימות ב-ELMo. נתאר בקצרה את העקרונות של מנגנון ה-attention, שהוא הלב של BERT:

באופן הכי פשוט, בהקשר של עיבוד שפה self-attention הוא מנגנון שמשערך את הקשרים של כל מילה בטקסט כלשהו ביחס לשאר המילים באותו טקסט. כאשר מבצעים self-attention על קטע טקסט, מקבלים ייצוגים חדשים של המילים הלוקחים בחשבון גם את הקשרים בין המילים השונות באותו טקסט. בזכות אופיו של מנגנון ה-attention, ניתן לבנות ייצוג של מילה שתלוי בקשרים שלה עם מילים הנמצאות רחוק ממנה בקטע טקסט, כלומר ההקשרים המתקבלים בין המילים יכולים להיות מיוצגים בצורה טובה גם עבור רצפים ארוכים ומילים שאינן נמצאות בסמיכות יחסית (שכאמור זה היה אחד החסרונות הגדולים של ELMo). בנוסף, מנגנון זה מייתר את הצורך לעבור מילה אחר מילה בקטע טקסט לצורך בניית ייצוג המילים שבו. במקום מעבר זה, ה-encoder מקבל הקלט את כל קטע הטקסט כמקשה אחת, מה שעשוי להקטין את הזמן הנדרש עבור בניית הייצוג של המילים. לכן ה-encoder בטרנספורמר יכול לשמש מודל שפה, אם מאמנים אותו בצורה מתאימה.

בשונה ממודל LSTM ששומר את המצב בכל נקודת זמן ובעצם מקודד את המיקום של כל מילה בכך שהקלט מתקבל כמילה בודדת בכל פעם, מודל הטרנספורמר מקבל את כל הקלט בבת אחת. לכן בשביל לקחת בחשבון את המיקום של כל מילה במשפט אנחנו משתמשים באלמנט נוסף שנקרא Positional Embedding. אלמנט זה מקודד וקטור ייחודי לכל מיקום במשפט ובסוף מבצעים חיבור של הווקטור שנוצר מהקלט והווקטור שנוצר מהמיקום.

המפתחים של BERT אימצו מארכיטקטורת הטרנספורמר המקורית את ה-encoder, והגדירו משימת אימון חדשה בכדי להפוך אותו למודל שפה. בכדי לבנות מודל מוצלח, תהליך האימון של BERT כלל שתי משימות: 1. Masked Language Model (MLM) – באופן רנדומלי עושים masking למילים מסוימות, ומטרת המודל הוא לחזות את המילים החסרות. 2. Next Sentence Prediction (NSP) – המודל מקבל כקלט זוגות של משפטים מקטע טקסט, ומטרת המודל היא לחזות לחזות האם המשפט השני הוא המשכו של המשפט הראשון במסמך המקורי. ארכיטקטורת הרשת נראית כך:




איור 11.4 ארכיטקטורת BERT. הקלט הינו משפט המיוצג כלשהוא, והפלט הוא אותו משפט אך כל מילה קיבלה מידע נוסף על ההקשר שלה וכעת מיוצגת באופן חדש. תהליך האימון והוספת ההקשר בין המילים נעשה באמצעות self-attention.

גם BERT, בדומה ל-ELMo, מציע בסופו של דבר מודל שפה מאומן היודע לקחת טקסט המיוצג באופן מסוים ולהוסיף לו מידע על היחס בין המילים השונות שבטקסט. תהליך יצירת המודל היה אמנם יקר, אך כעת ניתן לקחת אותו ויחסית בקלות לכייל אותו ואף להוסיף שכבות בקצה עבור משימות שפה שונות.

### GPT: Generative Pre-trained Transformer

עם הכניסה של מנגנון ה-attention וטרנספורמרים לעולם ה-NLP, הוצעו יותר ויותר מודלי שפה מבוססי attention. לצורך ההמחשה ניתן לציין שבשנים הבודדות שעברו מאז יצא BERT, הוא צוטט כבר בעשרות אלפי מאמרים. אחד המודלים היותר מפורסמים הינו Generative Pre-Training (GPT). מודל ה-GPT הינו מודל שעובד בשיטת auto-regression, כלומר, כאשר המודל חוזה את המילה הבאה הוא מוסיף את המילה לקלט עבור האיטרציה הבאה. כך הוא יכול בעצם ליצר משפטים מהתחלה של מילה בודדת. אם נרצה לדייק, המודלים הללו לא תמיד משתמשים במילים כיחידה האטומית, לפעמים אנחנו נעבוד עם חלקי מילים ואפילו אותיות להם נקרא טוקנים או אסימונים. דבר זה יכול לעזור לנו בהכללה ולהקטין את הסיכוי לטוקן שלא נמצא במילון (Out of Vocabulary).

הארכיטקטורה של GPT בנויה מ-Transformers מה שמאפשר לבנות ארכיטקטורה עמוקה שמתחשבת בקונטקסט של המשפט עבור כל מילה (Contextual embeddings). ארכיטקטורת הTransformer הינה היחידה המרכזית של GPT, כאשר בשונה מ-BERT ה-GPT משתמש רק ב-decoder (מנגנון ה-self-attention) שמקודד את הפיצ'רים, והפלט שלו הינו הטוקן הבא.

השכבה הראשונה בארכיטקטורה של GPT היא שכבה הנקראת Input encoding והיא הופכת את המילים (או ליתר דיוק הטוקנים) לווקטורים, כלומר היא מבצעת word embedding.

לאחר קידוד הקלט נשתמש במודל ה-Transformer בכדי לקודד פיצ'רים שמהם נסיק את הטוקן הבא. התהליך הזה מתבצע בעזרת רכיב הנקרא Masked Self attention. בשונה ממנגנון self-attention רגיל שמקודד כל טוקן בעזרת הקונטקסט של כל שאר הטקסט, GPT צריך לקודד כל טוקן רק בעזרת הטוקנים שקדמו לו, כיוון שבשלב זה המידע היחיד שקיים זה הטוקנים שנוצרו עד כה (וכמובן שאין גישה לטוקנים שעדיין לא נוצרו). כאשר מקודדים את הייצוג עבור טוקן מסוים, רכיב Masked Self attention מאפס כל וקטור של טוקן שבא אחריו, כך שהמודל לא יכול ללמוד ייצוג התלוי מילים שבאות לאחר הטוקן המיוצג, אלא עליו להפיק את המירב מהטוקנים הקודמים לו.

כיוון ש-GPT פועל בצורה של auto-regressive, ניתן לאחר האימון ליצור טקסט באמצעותו – ניתן למודל התחלה קצרה של טקסט, ונבקש ממנו ליצור את המילים הבאות. כך בכל שלב ניתן לו כקלט את הטקסט הראשוני ואת הטוקנים שיצר בשלבים הקודמים, והוא ימשיך וייצר עוד ועוד טקסט.

### Perplexity

לאחר בניית מודל שפה, נרצה "למדוד" עד כמה הוא מוצלח. לצורך זה יש להגדיר מטריקה מתאימה. המטריקה הכי נפוצה למדידת "עוצמה" של מודל שפה הינה perplexity, שזהו מושג הלקוח מתורת האינפורמציה והוא מודד כמה טוב מודל השפה חוזה את השפה ב-Corpus שאותו ניסינו למדל.

לפני שנסביר את המושג באופן פורמלי ניתן אינטואיציה למה אנו מצפים לקבל מהמטריקה שנבחר. נניח ואנו מבצעים את הפעולה הבאה: ראשית לוקחים משפט שלם וחותכים ממנו את ההתחלה, ואז לוקחים את אותה התחלה ומכניסים כקלט למודל שפה ומבקשים מהמודל לחזות את המשך המשפט. כמובן שנרצה לקבל חיזוי שדומה ככל האפשר למשפט המקורי, ונוכל למדוד הצלחה של מודל על ידי השוואת הפלט שלו למשפט האמיתי. באופן יותר כללי ניתן לקחת טקסט המכיל כמות משפטים כרצוננו, להכניס חלקים ממנו למודל השפה, ולהשוות את הפלט המתקבל לטקסט המקורי. כיוון שמודל שפה הינו הסתברותי, השוואת הפלט למשפט המקורי באופן מילולי בלבד אינה מספיקה, כיוון שהיא אינה משקפת בצורה מספיק טובה את מידת ההצלחה שלו. אם למשל במשפט המקורי הייתה כתובה המילה "לבנה" ואילו המודל חזה את המילה "ירח", השוואת שתי המילים כשלעצמן מראה לכאורה שהמודל שגה לחלוטין, אך בפועל אנו יודעים שמילים אלו נרדפות ולכן הפלט של המודל במקרה זה הוא דווקא כן טוב. לכן, נרצה לבחור מדד המסוגל לבחון עד כמה סביר לקבל את הפלט של המודל בהינתן חלק מהמשפט המקורי.

מדד perplexity בא להתמודד עם אתגר זה, והוא אכן פועל **בצורה שונה** מהאופן בו תיארנו את ההשוואה הפשוטה בין טקסט המקור לבין הפלט של מודל השפה. מדד זה מסתכל רק על הטקסט המקורי, והוא עובר מילה-מילה בטקסט זה ובודק **מה ההסתברות שמודל השפה ינבא את המילה הבאה בטקסט** בהינתן כל המילים שלפניה. ככל שההסתברות יותר גבוהה, כך המודל יותר מוצלח. באופן פורמלי, perplexity מוגדר באופן הבא:

$$PP(W) = P(w_1, w_2, \dots, w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1, w_2, \dots, w_N)}}$$

ככל שהמודל מנבא בהסתברות גבוהה יותר את המילים של המשפט המקורי, כך המונה שבתוך השורש יהיה יותר גדול, וממילא כל הביטוי עצמו של ה-perplexity נהיה קטן יותר. כלומר, ככל שערך ה-perplexity קטן יותר, כך המודל מוצלח יותר. נפתח מעט את הביטוי האחרון בעזרת כלל השרשרת:

$$= \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(w_i | w_{i-1}, w_{i-2}, \dots, w_1)}}$$

למשל עבור מודל מבוסס bigram, המדד יהיה פשוט יותר ויראה כך:

$$= \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(w_i | w_{i-1})}}$$

כאמור לעיל, ככל שערך מדד perplexity **נמוך יותר**, כך מודל השפה איכותי יותר.

## 10. References

http://d2l.ai/

ELMo, BERT:

https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html