

טרנספורמרים

הקדמה

כחלק מעבודת התזה שלי התעמקתי לאחרונה בארכיטקטורת הטרנספורמרים. כשקראתי את המאמרים הראשוניים בתחום (Attention is all You Need, ViT) התקשיתי לעבד את הרעיונות הקונקרטיים שעליהם מבוססים הטרנספורמרים לרעיונות מופשטים. כתוצאה מכך צללתי לנושא, ניתחתי קוד ממספר מקורות באינטרנט לצד קריאת המאמרים, ולבסוף ניהלתי עם ChatGPT סיבוב שאלות-תשובות כדי לחדד את ההבנה. לאחר שבוע וחצי של מחקר מאומץ, הגעתי למסקנה שאני רוצה לשתף את הידע שצברתי עם העולם. ולכן, החלטתי לכתוב סדרת פוסטים שאוכל להביע דרכה את ההבנה שלי, ואולי לקצר תהליך למי שאין לו הזמן שהיה לי להשקיע בלמידה הזו. כחלק מהכתיבה, אסקור את ההיסטוריה של התחום, מבנה הרשת, יתרונות וחסרונות, וכיצד מאמנים אותה. בשלב השני אסקור מאמרים בתחום הראייה הממוחשבת המציגים את השימוש ברשת למשימות כגון סגנטציה, הדבקת תמונות (Image Matting), העתקת מנח גוף (Gait Transfer), סיווג וכדומה. בכל פוסט אוסיף קישור לקובץ מתעדכן המכיל את כל המידע המתוקצר בפוסטים.

פרט אחרון, במשך זמן מה התלבטתי האם לכתוב את התוכן הזה בעברית או באנגלית. לבסוף בחרתי בעברית מכיוון שישנו המון חומר בנושא באנגלית, ומי שירצה יכול בחיפוש קצר בגוגל למצוא אותו. לעומת זאת בעברית אני מצליח להביא את הקול הייחודי שלי. מאחל לכם קריאה מהנה, ולי הצלחה.

טרנספורמרים הוצעו בהתחלה כפתרון לבעיית ניתוח טקסט. המאמר שבו הטרנספורמרים הופיעו לראשונה נקרא Attention is all You Need, הכותבים השתמשו בארכיטקטורה זו למשימת תרגום מאנגלית לצרפתית. מאמר זה למעשה פתח את הסכר למהפכת ה-NLP שאנחנו רואים היום. טרנספורמרים כללו מספר רעיונות חדשים בתחום ניתוח השפה. השניים המרכזיים, שזורים אחד בשני בארכיטקטורה, ומהווים את אבני הבניין הקונספטואלים של הרשת (בנוסף לעוד חידושים שהמאמר הציג). הראשון היה עיבוד מידע מקבילי, שהוביל ליעילות חישובית באימון המודל ביחס למודלים קודמים (RNN\LSTM\GRU), ואפשר בפעם הראשונה לפרוץ את מחסום הלמידה התלויה בזמן של קלט סדרתי. כלומר, ניתן ללמוד במקביל תלויות קצרות וארוכות טווח בקלט סדרתי. נציין שגם טרנספורמרים מוגבלים ביכולתם לעבד קלט מקבילי, אולם זוהי מגבלה התלויה במשאבי חישוב כדוגמת זיכרון זמין ויחידות עיבוד (כרגע מספר הטוקנים המקסימלי הוא 1024 [1])

בנוסף, הרשת הציגה שני מנגנוני תשומת הלב (attention). הראשון הוא תשומת לב-עצמית (self-attention) שאפשרה למודל להתמקד במידע החשוב ביותר באופן סלקטיבי. המנגנון השני הינו תשומת לב מוצלבת (cross-attention) שאפשרה מידול תלויות וקשרים בין חלקי הקלט והפלט השונים. תכונות אלו הכרחיות במשימות כדוגמת תרגום ומענה על שאלות סיכום טקסט, שדורשות בחירה מודעת בחלקים החשובים ביותר של הקלט והפלט. רעיונות אלו הלהיבו אותי, וגרמו לי לרצות להבין מה הוביל להתפתחותם. על מנת לעשות זאת התחלתי לסקור את הארכיטקטורות שקדמו לטרנספורמרים, כיצד הם עבדו, ומדוע לא צלחו במשימה שהטרנספורמרים כן הצליחו בה.

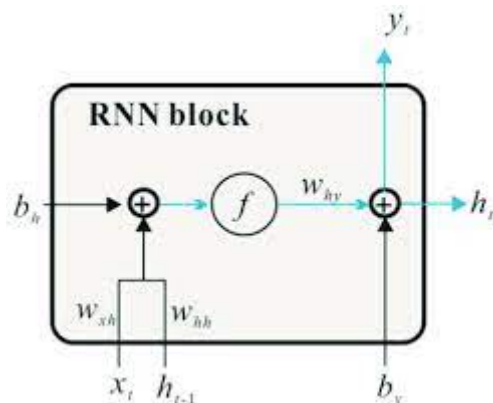
פרק 1 - ארכיטקטורת RNN

לפני שנדון בפתרונות שהוצעו לבעיה, נפתח בהצגה שלה. כאשר אנו מדברים על בעיה, עלינו להתייחס למורכבות (complexity) שלה. מורכבות של משימה הנשענת על ניתוח נתונים יכולה להתבטא בכמות המימדים, גודל הקלט, רעש, דגימות חריגות (outliers), ודפוסים מורכבים החבויים בדאטה. בהקשר זה, שפה טבעית היא בעיה עם מורכבות גבוהה, שכן היא מכילה תלות ארוכה וקצרת טווח בין מרכיבי הקלט השונים, סלנג, כפל משמעות (כדוגמת סרקזם), וסמנטיקה חבויה וגלויה. שפה מקיימת גם כללי תחביר ודקדוק, כאשר משמעות של מילה מסוימת תלויה בהקשר בו היא נאמרת, ועל כל אלו שפה אינה שומרת על

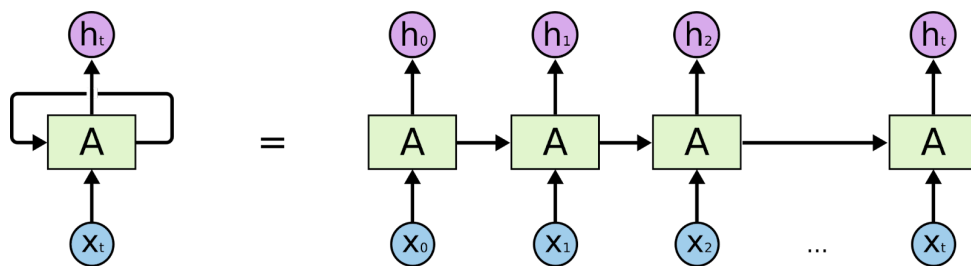
רציפות באופן מוחלט. אלגוריתמים בעלי חוקיות מוגדרת מראש (rule-based) כדוגמת [Cyc](#) שניסה לפרמל פתרון לניתוח שפה באמצעות סט חוקים פנימיים, נכשלו בעקבות כך.

נדגים כעת את מורכבות ניתוח שפה, באמצעות בעיית התרגום, ואת הקושי שהיא מציפה. נניח ואנו מעוניינים לתרגם את המשפט הבא מאנגלית לעברית: "The dog sleeps" - "הכלב ישן". ניתן לתרגם כל מילה במשפט ללא תלות במילים האחרות. הכלל היחיד שעלינו להתחשב בו הינו שהמילה "the" תוסיף תחילית "ה" למילה שבאה אחריה. ולכן, נוכל לשמור מילון המכיל לכל מילה באנגלית את התרגום שלה בתוספת לחוקים שהצענו ולקבל תרגום תקין. אולם, בהינתן המשפט "We're all mad here" (עליסה בארץ הפלאות) תוצאת התרגום שבוצע בהתאם לחוקים אלו תהיה "כולנו את כל מטורף כאן". שורש הבעיה הוא תרגום המילה הנוכחית ללא התחשבות בהקשר (context), כלומר במהלך התרגום אנו חייבים להתחשב בתלויות בין המילה שאנו מתרגמים לבין המילים האחרות במשפט. ניתן להציע פתרון המשכלל את המילון שהצענו, ולשמור בנוסף על המילים והתרגום שלהם, גם צמדי מילים, אך זהו פתרון נקודתי לבעיה מבנית, ודי מהר נצטרך להרחיב את המילון לביטויים בעלי 3, 4 ויותר מילים. ולבסוף, נתכנס לכישלון המשותף לכל האלגוריתמים שניסו לפתור את הבעיה בצורת (rule-based), והוא חוסר היכולת למדל את חוקיות הבעיה בצורה יעילה.

הארכיטקטורה הראשונה שניסתה למדל את התלות בין מילים בקלט מתוארכת לסוף שנות השמונים של המאה הקודמת [2, 3], ונקראת RNN. הרעיון המרכזי מאחוריה הוא ניתוח הקלט הנוכחי, בתוספת למידע שהתקבל מהמילים האחרות במשפט. הרשת מקבלת מילה במשפט ומייצרת שני פלטים, הראשון הוא פלט המתאים למילה ביחס למשימה (תרגום, תקצור טקסט, יצירת טקסט וכו'), והשני הוא ההקשר שנצפה עד עתה ומייצג תמצות של כל המידע שהרשת קיבלה עד אז (בייצוג lossy). פתרון פשוט לבעיה זו הינו שרשור של יחידות חישוב כאלה, כך שכל יחידת עיבוד (שנקרא לה בלוק מטעמי נוחות החל מכאן) מקבלת קלט בזמן t (כלומר את הקלט במיקום t -ה במשפט/קטע טקסט) ואת ההקשר מזמן $t-1$ ומייצרת את הפלט המתאים לו. אולם, משפטים שונים מכילים מספר מילים שונה, ולכן, לא ניתן להחזיק מספר בלתי מוגבל של בלוקים כך שכל בלוק יתאים לקלט מסוימת במשפט. בנוסף בארכיטקטורה שכזו הבלוקים הראשונים יאומנו הרבה יותר מהבלוקים האחרונים, דבר הנובע מכך שבמשפטים, קצרים ממספר הבלוקים המירבי, ירופדו באפסים, והבלוקים האחרונים לא ישתתפו באופן מפורש בתהליך הלמידה. ולכן, הפתרון הינו הפעלה איטרטיבית של הרשת. כלומר שימוש בבלוק יחיד וכתוצאה מכך בסט משקולות משותף, כך שהרשת מקבלת בכל איטרציה קלט חדש ואת ההקשר שהתקבל במוצאה באיטרציה הקודמת. הייצוג הטורי מהווה פריסה (unfolding) של הרשת (ראה איור 2), בייצוג הקומפקטי: הרשת מקבלת קלט X_t ואת המצב הפנימי מהאיטרציה הקודמת שמתואר כ- H_{t-1} , ומוציאה פלט Y_t , בעוד שהמצב הפנימי H_t מוחזר בחזרה לרשת באיטרציה הבאה.



איור 1: מבנה רכיב RNN



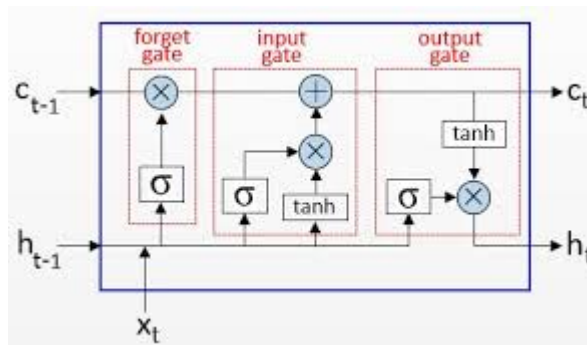
איור 2: RNN בייצוג טורי (מימין) ובייצוג קומפקטי (unfolding)

אז כיצד משתמשים ברשת כזו לטובת ניתוח שפה? טרם השימוש מקודדים את המילים/חלקי מילים (word/subword embedding). פעולת הקידוד היא המרת מילה בשפה טבעית, לוווקטור בעל k ממדים, כאשר הייצוג הוווקטורי מקבץ מילים בעלות משמעות דומה תחת ערכים קרובים, ומרחיק מילים בעלות משמעות שונה אחת מהשנייה. [בקישור](#) הנ"ל ניתן לראות שיטות שונות לבצע זאת, כאשר הנפוצה ביותר היא שימוש ברשת נוירונים (לדוגמה word2vec). הבעיה באלגוריתמים אלו, הינה שקידוד מילים בעלות כפל משמעות יהיה זהה, דבר שמקשה על הלמידה. בהקשר זה, אחד החידושים שארכיטקטורת הטרנספורמרים הציגה היה שילוב שיכון (embedding) המילים כחלק מהארכיטקטורה, דבר שאפשר למידה של שיכון המילה ביחס למילים קרובות למילה הנוכחית, אלא גם מילים רחוקות ממנה (כלומר שיכוני מילים במרחב תלוי הקשר או contextualized embeddings).

על פניו, נראה שהרשת מספקת פתרון לבעיה, היא מסוגלת לאגור מידע ולמדל את התלות בין חתימות הזמן השונות. אז מדוע זה לא עובד? ישנן מספר סיבות האחריות לכך. הראשונה נעוצה במבנה הטורי של הרשת, שהוביל כמעט תמיד לדעיכת הגרדיאנט לאפס (vanishing gradient). כפי שהוזכר בהתחלה, במשימות ניתוח שפה הפלט עבור קלט מסוים תלוי בסדרת קלטים שקדמו לו, ולכן, בעת אימון הרשת המשקולות מעודכנות רק לאחר העברת הסדרה כולה. ברשתות איטרטיביות, יש לחשב את הגרדיאנט של המשקולות גם כן בסדר איטרטיבי אולם הפעם הוא הפוך. כדי לחשב את השפעת המשקולות על השינוי בשגיאה, עלינו לכפול בטור את הנגזרות החלקיות של כל מצב פנימי i ביחס למצב הפנימי $i-1$. מכיוון שהשינוי מחושב עבור אותו סט משקולות, הגרדיאנט יכול לדעוך או להתבדר לאורך האימון. ניתן להמחיש זאת באמצעות הערכים העצמיים של מטריצת המשקולות; במידה והם קטנים מ-1, הגרדיאנט ידעך מכיוון שמטריצת המשקולות מקטינה אותו בכל פעם שמתבצע עדכון, ובמידה והערכים העצמיים גדולים מ-1

הגרדיאנט יתבדר מכיוון שמטריצת המשקולות מגדילה אותו בכל איטרציה ([דעיכת הגרדיאנט ופתרונות ב-RNN](#)). הבעיה השנייה נובעת מהיחס בין גודל המצבים הפנימיים והיכולת שלהם לדחוס מידע, לבין גודל הסדרה שהרשת נדרשת לנתח. המצב הפנימי "דוחס" את כל המידע עד לנקודת הזמן הנוכחית, ולכן, כאשר הסדרה עוברת סף מסוים בגודלה ומתחילות להיווצר תלויות ארוכות טווח בין מרכיביה, המצב הפנימי הופך להיות צוואר-בקבוק ברשת והיא מתחילה "לשכוח" פרטים שרחוקים מהקלט הנוכחי שלה. הבעיה האחרונה צצה במהלך השימוש ברשת לאחר האימון (inference). הרשת אינה מנצלת את מלוא הפוטנציאל החישובי שלה (ושל המכונה שהיא רצה עליה) כתוצאה מאופי קבלת הקלטים באופן טורי, דבר הגורר זמן הסקה איטי (כלומר latency גבוה) של המצבים הפנימיים במגוון משימות NLP. בעיה מהותית זו נפתרת בארכיטקטורת הטרנספורמים שנדבר עליה בהמשך.

פרק 2 - ארכיטקטורת LSTM



איור 2: בלוק LSTM

כפי שראינו, דעיכת הגרדיאנט וכשל בדחיסת מידע מונעים מרשת RNN את היכולת להתרכז בחלקים החשובים ביותר בסדרה. כתוצאה מכך, הרשת מתקשה לאתר את התלות בין מרכיבי השונים שזו כאמור המטרה העיקרית בעיבוד שפה טבעית. כיצד ניתן להתגבר על בעיות אלה ברשתות המקבלות כקלט מידע סדרתי?

על מנת שנוכל לענות על שאלה זו, נבחן כיצד בני אדם מתמודדים עם סוג מידע שכזה. נציג הפשטה של הרעיון: בני אנוש משתמשים בשני אלמנטים עיקריים: הראשון הוא זיכרון לטווח ארוך וקצר, והשני הוא היכולת להפריד בין טפל ועיקר. שני מנגנונים אלו קשורים אחד בשני באופן הפעולה שלהם. כלומר, אנחנו מסוגלים להפריד בין טפל לעיקר בזכות ניסיון עבר והשלכתנו על סיטואציה חדשה.

זהו בסיס הרעיון שהרשת הבאה מנסה ליישם. [LSTM](#) (וגרסתה האלטרנטיבית GRU) היא ארכיטקטורה איטרטיבית שבדומה ל-RNN מכילה זיכרון לטווח קצר h_t , (המצב הפנימי ב-RNN) ובנוסף, רכיב זיכרון ארוך c_t . בניגוד ל-RNN שמתמצתת את המידע שהתקבל עד לנקודת הזמן הנוכחית כזיכרון קצר טווח, ההפרדה לזיכרון לטווח קצר וארוך ב-LSTM מאפשרת לרשת לשמר את מאפייני הקלט המהותיים גם בטווח הארוך ופותרת את בעיית צוואר הבקבוק של RNN. מבנה הרשת מאפשר לה לשמור את שני סוגי הזכרונות והמילה החדשה בכל איטרציה, ולנתח את המילה החדשה באמצעותם.

ניהול זיכרון מתבצע באמצעות שלושה שערים: שער השכחה, שער הקלט ושער הפלט (איור 2). **שער השכחה** אחראי על הסרת מידע מהזיכרון לטווח הארוך לאחר שזה התגלה כלא רלוונטי יותר. **שער הקלט** אחראי על הזרמת המידע לתוך הזיכרון. שער זה משתמש במידע המתקבל מהמצב הפנימי הקודם ומקלט חדש, ומחליט מה מהמידע החדש רלוונטי ויש להוסיפו לתא הזיכרון. **שער הפלט** אחראי על חישוב מוצא הרשת - h_t , המהווה למעשה את הזיכרון קצר הטווח.

היתרון של LSTM מתבטא בעיקר במשפטים ארוכים, לדוגמה, במשפט "She left" תרגום המילה "left" על ידי RNN תהייה "הלכה" ולא "הלך" (או "שמאל") מכיון שההקשר הוא קרוב. אולם, אם ניקח את המשפט "The goalie was determined to protect the goal" את המילה האחרונה ניתן לתרגם כ"משימה/מטרה" או "שער (כדורגל)", במקרה זה, מילה ההקשר "goalie" (שוער) מרוחקת מהמילה שאנו רוצים לתרגם, ולכן RNN עלולה לטעות בתרגום, במקרה זה יבוא לידי ביטוי הזיכרון ארוך הטווח, שמתחשב במילה "goalie" בתרגום המילה "goal".

מדוע LSTM סובלת פחות מדעיכת הגרדיאנט? הסיבה נובעת מכך שרכיב הזיכרון אינו מתעדכן בפלט של פונקציית האקטיבציה כמו ב-RNN, אלא בשימוש במוצא שער השכחה ושער הקלט. שער השכחה מסיר חלקים מהקלט על ידי יצירת וקטור של ערכים רציפים במקטע של $[0,1]$, כאשר 0 משמעו שכחה מוחלטת,

ו-1 מהווה שימור מוחלט של המידע. הקלט והמצב הפנימי קובעים כמה מידע יש לשכוח, אולם הם אינם קובעים מה יהיה תוכן תא הזיכרון לאחר העדכון. מן הצד השני, הוספת מידע חדש לזיכרון אינה מתבצעת כדריסה של התוכן על ידי הקלט, אלא כחיבור שלו. שער הקלט מייצר וקטור שמתווסף למידע הקיים בתא הזיכרון בפעולת חיבור (element-wise addition) ובכך מאפשר את העברת האינפורמציה ברשת ומונע את דעיכת הגרדיאנט, מכיוון שפעולה זו פחות רגישה לשינויים בקלט כפי שמוצא פונקציית האקטיבציה רגישה אליהם. כתוצאה מכך, הנגזרות שאינן מושפעות באופן ישיר מפונקציות האקטיבציה ולא מתאפסות, מאפשרות את עדכון המשקולות (בשונה מ-RNN שהנגזרות מתאפסות די מהר). בנוסף, מכיוון ששער השכחה מייצר וקטור הקובע אילו חלקים ברכיב הזיכרון יש לשכוח ואילו לשמר, כאשר ערך הוקטור קרוב ל-1, המידע יכול לזרום ברשת מבלי להינזק באופן משמעותי או במילים אחרות, מתאפשרת שמירת מידע ארוך טווח (מידול מתמטי של נושא זה ניתן למצוא [בקישור](#)). בזכות השימוש ברכיב זיכרון, המצב הפנימי מצטמצם להיות זיכרון לטווח קצר בלבד והשפעת צוואר הבקבוק שנוצרת ב-RNN פוחתת גם היא.

למרות היתרונות המובהקים של הרשת על פני RNN במשימות בהן ישנו צורך לנתח תלויות ארוכות טווח, קיימות לה מגבלות בניתוח שפה טבעית הנובעות ממורכבות הארכיטקטורה ואופן ההפעלה האיטרטיבי שלה. הבעיה הראשונה משותפת לכלל הרשתות האיטרטיביות, והיא חוסר היכולת להפעיל את הרשת במקביל, דבר הגורר הפעלה איטית של הרשת ואי ניצול מספק של משאבי החישוב, בנוסף, לא מאפשרת למדל תלויות בין מקטעים שונים של הקלט.

מכיוון שכל תא LSTM מכיל מספר גדול יותר של פרמטרים בהשוואה ל-RNN, אימון והפעלת הרשת דורשים משאבי חישוב (זיכרון וכוח עיבוד) רבים יותר. בנוסף, על מנת שהרשת תמדל את התלויות ארוכות הטווח באופן אפקטיבי ישנו צורך בכמות משמעותית של דאטה בעלת מספר טוקנים גבוה (=אורך הסדרה). בנוסף, משך האימון הוא גם כן חסרון משמעותי של הרשת. חיפוש תלויות ארוכות טווח בסדרה הינה משימה מורכבת יותר ביחס לחיפוש דפוסים מקומיים (local patterns) בדאטה, ולכן נדרש זמן אימון ארוך יותר.

פרק 3 - מנגנון תשומת הלב לפני עידן הטרנספורמרים

לאחר שראינו כיצד רשתות איטרטיביות כגון RNN/LSTM ממדלות את התלויות בקלט סידרתי, נציג כעת את המנגנון המהווה את ליבה של ארכיטקטורת הטרנספורמרים: מנגנון תשומת הלב (Attention mechanism). במהלך פרק זה נענה על השאלות הבאות:

- מהו מנגנון תשומת הלב?
- מדוע הוא נדרש?
- כיצד מנגנון תשומת הלב התפתח במכוונות לומדות?

מהו מנגנון תשומת הלב ?

לפני שנגדיר מהי תשומת לב במכוונות לומדות, נבחן כיצד היא באה לידי ביטוי בקוגניציה האנושית. תשומת לב הינה התמקדות סלקטיבית בחלקים הרלוונטיים ביותר של מידע שאנו חווים וסינון חלקים בעלי חשיבות פחותה. תשומת הלב מאפשרת לנו למקד את המשאבים העומדים לרשותינו בצורה יעילה בכל סיטואציה, ובכך אנו יכולים להתמצא ולהבין טוב יותר את מלוא האינפורמציה הזמינה לנו. מנגנון תשומת הלב הוא יכולת מולדת של בני אדם, אולם היא גם כן יכולת נלמדת הניתנת לשיפור במהלך חיינו, (לדוגמאה [מיינדפולנס](#)).

האופן המופשט שבו מנגנון תשומת הלב פועל במוחנו הינו :

1. קבלת קלט על ידי הסנסורים החושיים שלנו (כגון מערכת ראייה, מערכת שמע וחוש הריח).
2. עיבוד מקדים וסינון המידע על ידי המוח.
3. בחירת החלקים החשובים ביותר של המידע בהתאם למידע קודם וההקשר הנוכחי.
4. המידע שנבחר כרלוונטי עובר עיבוד, ובסיום נשמר בזיכרון.

מנגנון תשומת הלב למעשה מגן עלינו בתור בני אדם, מכיוון שהוא מאפשר לנו **להתעלם** ממידע שאינו חיוני לנו (המקיף אותנו הרבה יותר ממידע חיוני). דוגמה לכך היא "מסיבת קוקטייל" (cocktail party) שבה אנו נוכחים במפגש חברתי שבו מספר רב של אנשים מדברים בו זמנית, ואנו מעוניינים להתמקד באדם אחד שמדבר. המוח שלנו מסייע לנו למקד את תשומת הלב שלנו באדם זה ולהתעלם משאר הקולות שהופכים לרעש רקע.

הנקודה המזקקת את העיקרון של תשומת הלב הינה היכולת לשערך מידע (information assessment). במילים אחרות אנו **לומדים להעניק משקל** לכל פיסת מידע, ביחס לכל פיסת מידע אחרת, בהתבסס על החשיבות שלה, והקשר שלה עם מידע שאנו כבר יודעים (הנאגר בזכרון), או חווים ברגע נתון.

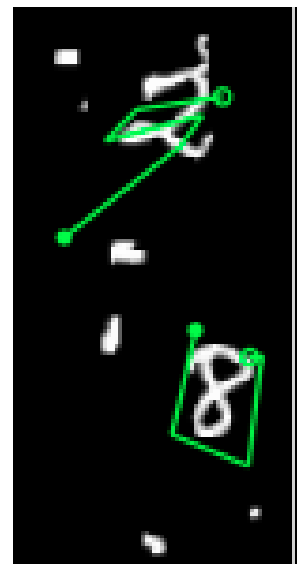
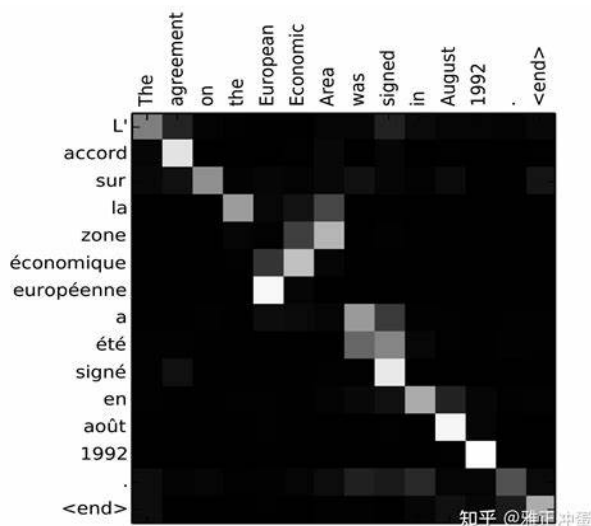
אז מהו החידוש שמנגנון תשומת הלב מביא איתו כאשר הוא משולב ברשתות נוירונים? התשובה לכך היא שמנגנון תשומת הלב הוא **פונקציה הנלמדת** כחלק מתהליך אימון הרשת, שמטרתה לחשב מהו המשקל היחסי של כל יחידת דאטה בהינתן הקשר ומידע שנאגר עד לאותו הרגע. מנגנון תשומת הלב יוצר ייצוג וקטורי רציף תלוי-הקשר (contextualized representation) עבור יחידת המידע (כגון טוקן או מילה), ותוצאתו הינה פונקציה רציפה שניתן לגזור אותה ביחס לפרמטרים של מנגנון תשומת הלב (כלומר soft attention). מנגנון תשומת הלב משכן יחידות דאטה במרחב וקטורי כאשר ייצוג של יחידת דאטה במרחב זה מתחשבת בעוצמת הקשר **הרציפה (לא דיסקרטית)** בינה לבין שאר היחידות הדאטה. מאחר והמשקול

של יחידות דאטה הינה פונקציה גזירה ביחס לפרמטרים של מנגנון תשומת הלב (וגם ביחס לייצוג הקלט של מנגנון זה) אנו יכולים לאמן את הרשת לייצג את הקשרים האמיתיים בין בין החלקים השונים של הטקסט.

איור 1 (שמאל) המציג את מפת תשומת הלב של משפט באנגלית ביחס לתרגומו בצרפתית, ממחיש את עקרון הרציפות של פונקציית תשומת הלב. האיור מציג משקול של הטוקנים בסדרה הראשונה ביחס לכל הטוקנים בסדרה השנייה (מנגנון זה נקרא תשומת הלב המוצלבת אשר נרחיב עליו בהמשך). איור זה לקוח [מהמאמר](#) שהציג את מנגנון תשומת הלב לראשונה עבור משימות שפה טבעית. נדון במאמר זה בפרק העוסק בתשובה לשאלה "כיצד מנגנון תשומת הלב התפתח במכונות לומדות?".

אולם, לא כל מנגנוני תשומת הלב נולדו שווים. [מאמר זה](#), שהציג את השימוש הראשון במנגנון תשומת הלב (במאמר הופיעה לראשונה המונח attention), עבור סיווג תמונות, השתמש במנגנון תשומת הלב דיסקרטי (hard attention). אופן פעולת הרשת דומה למשחק חיבור נקודות באמצעות קווים היוצרות צורה. הרשת מחפשת את הפיקסלים המקיפים את את האזור הרלוונטי ביותר בתמונה (איור 1 מימין), כך שאם נחבר אותם בקווים ישירים, נקבל תמונה חדשה המכילה את המידע החשוב ביותר הדרוש לסיווג התמונה. בכל הפעלה של הרשת, מנגנון תשומת הלב בוחר פיקסל חדש, ומוסיף אותו לפיקסלים שבחר באיטרציות הקודמות, כאשר בחירת הפיקסלים הללו יוצרת בסופו של דבר את האזור התחום. מכיוון שפונקציית תשומת הלב הדיסקרטית איננה גזירה (ומקבלת ערכים דיסקרטיים בלבד) לא ניתן למטב (optimize) אותה עם שיטות ממשפחת מורד הגרדיאנט (gradient descent). עקב כך המאמר עשה שימוש בשיטת אימון משטר מבוסס החלטות (on policy) השאולה מתוך עולם הלמידה מבוססת החיזוקים (reinforcement learning), שלא דורשת גזירות של פונקציית מטרה, על מנת לאמן את מנגנון תשומת הלב.

בעולם ניתן השפה, מנגנון תשומת הלב הדיסקרטי, ילמד למצוא את הטוקנים החשובים ביותר (בהתאם למשימה) לבנייה של וקטור הייצוג עבור טוקן נתון. לדוגמא, נניח והמטרה של הרשת היא למצוא את האובייקט החשוב ביותר במשפט "The cat is playing with the toy, it is soft". הרשת תיתן את המשקל המקסימלי למילה "cat" ומשקל נמוך לשאר המילים.



איור 1 -בחירת הפיקסלים החשובים ביותר בתמונה (ימין) מפת תשומת הלב רציפה בתרגום משפט מאנגלית לצרפתית (שמאל)

תשומת לב "לא מפורשת" לעומת תשומת לב "מפורשת"

המנגנון שדנו בו עד עתה נקרא תשומת לב מפורשת (explicit attention), שבו אנו משערכים קשרים בין יחידות מידע שונות באופן יזום. לעומת זאת, ישנו מנגנון תשומת הלב נוסף הנקרא תשומת לב "לא מפורשת" (implicit attention). מנגנון זה הינו "תוצר לא מכוון" של רשתות עמוקות. רשתות אלו נוטות להתמקד בחלקים מסוימים של המידע ולהתעלם מאחרים. לדוגמא, בסיווג של מנח גוף מתוך סרטונים (pose estimation), תהייה לרשת הנטייה להתמקד באזורים שבהם מופיעים חלקי גוף ולהתעלם מאזורים בהם לא מופיע אדם בכלל (רקע או אובייקטים דוממים). ניתן להמחיש באופן ויזואלי את תשומת לב המרומזת על ידי איור 2, המגיע [מהרצאה הבאה](#). האיור ממחיש את מיקוד הרשת בעת ניתוח תמונה עבור אימון סוכן במשחק. הסוכן לומד לנסוע בתוך השביל ולהימנע ממפגעים באמצעות למידה מבוססת חיזוקים. החלקים הבוהקים באדום מייצגים את האזורים בהם הרשת מתמקדת על מנת לקבל את ההחלטה הבאה. מכיוון שהתקדמות בשביל מובילה לעלייה בנקודות הרשת מתמקדת באופן, ובלוח התוצאה שמציג את הניקוד. **מענה, בכל מקום בטקסט שנתייחס לתשומת לב, נתכוון לתשומת לב מפורשת.**



איור 2 - תשומת לב מרומזת ברשת מבוססת חיזוקים

מדוע מנגנון תשומת הלב נדרש?

כפי שהסברנו בפרקים הקודמים, ארכיטקטורות איטרטיביות סבלו מבעיה מרכזית המשותפת לכולן, והיא רכיבי זיכרון הקבועים בגודלם, ומאידך, קלט בעל אורך משתנה. כתוצאה מכך אנו נאלצים לקודד משפטים באורכים שונים לוקטור בגודל קבוע. ולכן, בעת קידוד קלטים העולים על גודל מסוים, נתכנס לבעיית מידול של תלויות ארוכות הטווח. עיקר הבעיה בא לידי ביטוי בכך שלא ניתן להשתמש בכל יחידות הקלט באופן מפורש לבניה של וקטור ההקשר h_t עבור יחידת דאטה i . במילים פשוטות, מטרנו לאפשר לוקטור המקודד את הקלט לגשת לכל חלקיו (של הקלט) במקביל בעת בניית הייצוג.

על מנת להמחיש את הנושא, נסתכל על הפסקה הבאה:

"צח, מהנדס תוכנה, עובד מהבית בשנתיים האחרונות. הוא מתגורר ביישוב קטן בצפון הארץ עם אשתו ושני ילדיו. היישוב שקט ורגוע ויש בו תחושת קהילתיות. צח נהנה לבלות עם משפחתו ולצאת לטיולים ארוכים ביער הסמוך. הוא מעריך את הגמישות שעבודה מרחוק מציעה לו. בשבוע שעבר גילה צח כי החברה שלו מתכננת

ליישם **מדיניות חדשה** שתחייב את כל העובדים לעבוד מהמשרד. צח שוקל כעת האם לעבור לגור בקרבת המשרד שנמצא במרכז הארץ או **לחפש עבודה חדשה**."

מודל ניתוח וסיכום טקסט המבוסס RNN או LSTM עשוי להתקשות להבין מהו המידע החיוני בטקסט זה וליצור סיכום תמציתי איכותי. מכיוון שאופן עיבוד הקלט הוא טוקן-אחרי-טוקן, תוצאה אפשרית של מודל איטרטיבי יכולה להיות:

"צח, מהנדס תוכנה, שעובד מרחוק, עובר לגור ליד המשרד, או מחפש עבודה חדשה, בגלל שינוי במדיניות החברה"

למרות שהרשת אכן "דחסה" את כל המידע החשוב, עדיין חסרה נהירות (קוהרנטיות) בפלט.

לעומת זאת, רשת המקיימת את התנאים הבאים:

- בעלת מנגנון תשומת הלב.
- מייצרת וקטור הקשר גדול מספיק.

יכולה לספק את התמצות הבא:

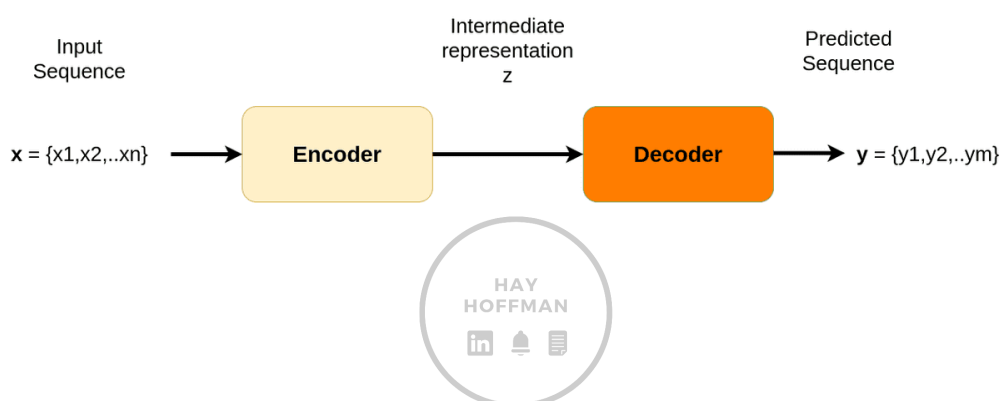
"צח, מהנדס תוכנה שעובד מרחוק, שוקל האם לעבור לעיר הקרובה לעבודתו במשרד או למצוא משרה חדשה המאפשרת עבודה מרחוק בשל מדיניות החדשה בחברה"

נקודות לסיכום הפרק:

- מנגנון תשומת הלב במכונות לומדות הינו פונקציה נלמדת, השואבת השראה מתשומת הלב בקוגניציה האנושית, וממשקלת חשיבות של קלט ביחס לקלט אחר.
- מנגנון תשומת הלב שאנו דנים בו נקרא soft attention שמהווה פונקציה רציפה. תכונה זו נובעת מהעובדה שמשקול של יחידת דאטה נמדד ביחס לכל יחידות הדאטה האחרות.
- ישנן שני סוגים של מנגנוני תשומת לב. תשומת לב מפורשת, הינה פונקציית תשומת לב הממומשת באופן יזום כחלק מארכיטקטות המודל. לעומת זאת, תשומת לב לא מפורשת הינה תוצר של עיבוד דאטה על ידי רשתות עמוקות, הלומדות חשיבות של אזורים מסוימים בקלט ללא הכוונה יזומה.
- רשתות בעלות מנגנון תשומת לב יכולות ללמוד קשרים מורכבים יותר בקלט ולייצר פלט קוהרנטי יותר ביחס לרשתות איטרטיביות.

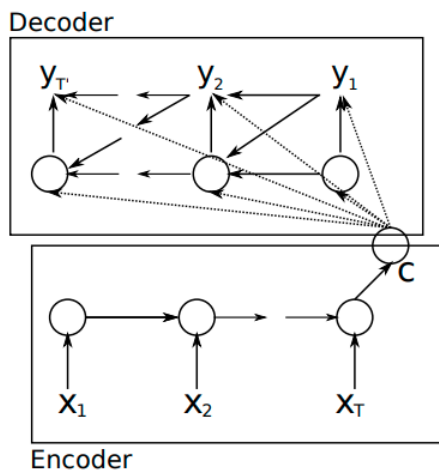
כיצד מנגנון תשומת הלב התפתח במכונות לומדות ?

[המאמר הראשון](#) שהציג שימוש במנגנון תשומת הלב עבור משימות של ניתוח שפה טבעית השתמש בארכיטקטורת מקודד-מפענח (encoder-decoder architecture), המתוארת באיור 3, לטובת תרגום מאנגלית לצרפתית. על מנת להסביר את הצורך במנגנון תשומת הלב, נציג כעת מושג חשוב בתחום עיבוד השפה הטבעית הנקרא "יישור" (alignment). מושג זה מתאר בהקשר של תרגום, את התאימות בין מילה/מילים משפת המקור לבין מילים בשפת היעד (איור 1 משמאל מדגים את היישור בין מאנגלית לצרפתית). במילים אחרות, זהו ייצוג של "עוצמת הקשר" בין קבוצות של מילים בשפת היעד לבין מילים בשפת המקור.

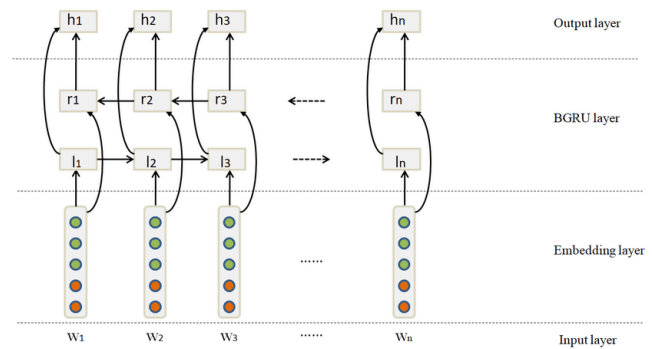


איור 3 - הפשטה של ארכיטקטורת מקודד-מפענח

כאן המקודד מקבל משפט (או קטע של טקסט) בשפה א' כסדרת טוקנים, ומייצג את המידע במשפט כוקטור במימד חבוי (latent representation). מן הצד השני, המפענח מחליץ מתוך הוקטור שהתקבל את המידע הרלוונטי ומפיק מתוכו את המשפט בשפה ב'. הרשת מאומנת כמקשה אחת, ולומדת לקודד ולפענח באותו הזמן. במאמר גם המפענח וגם המקודד מומשו על ידי יחידות GRU משורשרות (אולם ניתן להשתמש גם ב-RNN או LSTM), כאשר המקודד מורכב מיחידות GRU דו-כיווניות ([bidirectional GRU](#)). ארכיטקטורת GRU דו-כיוונית מורכבת משני שרשרים של יחידות GRU כאשר השרשור הראשון משמש להעברת הקלט מתחילתו לסופו, והשרשור השני של יחידות ה-GRU משמש להעברת הקלט מהסוף להתחלה (ראה איור 4). הסיבה לשימוש בארכיטקטורה דו-כיוונית נובע מכך שאנו מעוניינים לקבל מידע מקיף על הקלט, כלומר גם מתחילתו וגם מסופו, שכן כל המידע (משפט המקור) נתון לנו בזמן הפעלת המודל לאחר האימון (inference). לעומת זאת הפלט במפענח נוצר באופן אוטוגרסיבי (מילה אחרי מילה, כאשר פלט נוכחי הופך לקלט עתידי לאחר שנוצר) בזמן ההסקה, שהופך את השימוש ברשת דו-כיוונית במפענח למהלך משולל היגיון.



איור 5 - ארכיטקטורת מקודד מפענח



איור 4 - GRU דו-כיוונית

בארכיטקטורות מסוג מקודד-מפענח שקדמו למאמר, הקלט שהמפענח מקבל בכל איטרציה (יצירת יחידת דאטה חדשה) הינו המצב הפנימי h_i והפלט y_{i-1} מהאיטרציה הקודמת (של המפענח). על מנת לחבר את המידע שהמקודד למד מהקלט, המפענח מקבל בנוסף את וקטור המוצא של המקודד שנקרא לו מעתה C . בארכיטקטורות מקודד-מפענח בסיסית, C הינו שרשור המצבים הפנימיים שחושבו באיטרציה האחרונה מכל אחת מהרשתות המרכיבות את הרשת הדו-כיוונית במקודד ($h_i = [\rightarrow h_i; \leftarrow h_i]$) שכן הוא מכיל מידע על כל הקלט. איור 5 מציג את הארכיטקטורה שהסברנו בפסקה זאת.

כיצד בא לידי ביטוי מנגנון תשומת הלב במאמר?

כפי שהזכרנו קודם לכן, השימוש בתשומת הלב מיועד לפתור את בעיית היישור בין הקלט לפלט. עקב אכילס של הארכיטקטורות שקדמו לזו של המאמר, הייתה שימוש במצבים הפנימיים האחרונים של המקודד (הפלט של האיטרציה האחרונה של משני הכיוונים של BGRU). מכיוון שמצבים אלו הכילו מידע דחוס על כל הקלט, לא ניתן היה למדל את התלויות המקומיות בין המצבים הפנימיים של המפענח, לאלו של המקודד. כלומר, וקטורי ההקשר המתקבלים בכניסת המפענח דוחסים את כל המידע מסדרת הקלט של המקודד, ללא התחשבות בקשר שבין יחידות מידע בסדרה א' ליחידת מידע הנבנת בסדרה ב'. לעומת זאת, בשיטה המוצעת במאמר, וקטור ההקשר שהמפענח מייצר בעת בניית יחידת פלט i , מקבל את המידע על

כל יחידות הדאטה של הקלט שנבנו על ידי המקודד. וקטור ההקשר נוצר כסכום משוקלל של כל המצבים הפנימיים של המקודד, כאשר המשקלים ממדלים את הקשרים בין כל יחידות הקלט ליחידת פלט i .

מנגנון תשומת הלב: המידול המתמטי

המושג הראשון שהמאמר מגדיר הינו **עוצמת היישור** (alignment score), שמייצג את הקשר בין המצב הפנימי $i-1$ במפענח לבין מצב פנימי j כלשהו במקודד. על מנת למנוע בלבול, נגדיר את המצבים הפנימיים של המפענח כ- s (כפי שהוא מובא במאמר) ואת המצבים הפנימיים במקודד נשאיר כ- h . האינדקסים i, j מייצגים את המספר הסידורי של יחידות הקלט והפלט ה- j, i . כעת עבור יחידת פלט i נגדיר וקטור e_{ij} באופן הבא:

$$(1) e_{ij} = \text{attention}(s_{i-1}, h_j) = v_a^T * \tanh(W * [s_{i-1}; h_j]), j = 1, \dots, T$$

משוואה 1 - חישוב עוצמת היישור (מנגנון תשומת הלב)

כאשר:

- e_{ij} - עוצמת יישור לא מנורמלת
- h_j - המצב הפנימי של יחידה j של המקודד.
- s_{i-1} - המצב הפנימי מהיחידה $i-1$ של המפענח.
- W - מטריצת המשקלות של מנגנון תשומת הלב.
- v_a - וקטור המשקל של פונקציית תשומת הלב.
- T - מספר יחידות הדאטה במקודד.

כאמור, מנגנון עוצמת היישור הינו פונקציה נלמדת המחשבת את עוצמת הקשר שבין המצב הפנימי של המפענח למצבים הפנימיים של המקודד. מכיוון שהמכפלות בתוך פונקציית הטנגנס ההיפרבולית \tanh יוצרים וקטור בגודל $1 \times a$, וערך תשומת הלב בין שתי יחידות דאטה צריך להיות סקלר, המכפלה בוקטור v_a יוצרת סקלר במוצא. נשים לב כי המטריצה W והוקטור v_a הינם פרמטרים הנלמדים (מאומנים) של המודל.

המושג השני שהמאמר מגדיר הינו **משקולת תשומת הלב** (attention weight). מטרת מנגנון תשומת הלב הינה ליצור משקול החשיבות של טוקן אל מול כל טוקן אחר, כפונקצייה רציפה וגזירה. המשמעות של רציפות בהקשר שאנו מדברים עליו, הינו משקול של עוצמת הקשר e_{ij} (המקושרת למצב הפנימי h_i מהמקודד) ביחס לכל עוצמות הקשר האחרות. עוצמות אלו מייצגות את הקשר שבין כל שאר המצבים הפנימיים של המקודד ביחס למצב פנימי הנתון של המפענח. על מנת לעשות זאת, אנו משתמשים בפונקציית softmax המופעלת על עוצמות הקשר. חישוב זה למעשה פותר את בעיית היישור שפתחנו איתה את הפרק, מכיוון שפונקציית ה-softmax תהפוך את תוצאת היישור, שעמדה בפני עצמה, להיות פונקציית צפיפות הסתברות התלויה בכל המצבים הפנימיים של המקודד. אנו מבצעים פעולה זו עבור כל מצב פנימי של המקודד אל מול אותו מצב פנימי של המפענח, ובכך מקבלים את עוצמת הקשר הרציפה שהזכרנו.

$$\alpha_{ij} = \exp(e_{ij}) / \left(\sum_{k=1}^{T_x} \exp(e_{ik}) \right) \quad (2)$$

משוואה 2 - חישוב משקל תשומת הלב עבור זוג יחידות דאטה i, j



כאן T_x הוא מספר יחידות חישוב במקודד כלומר אורך מקסימלי של סדרת דאטה שניתן להכניס בו כמקשה אחת.

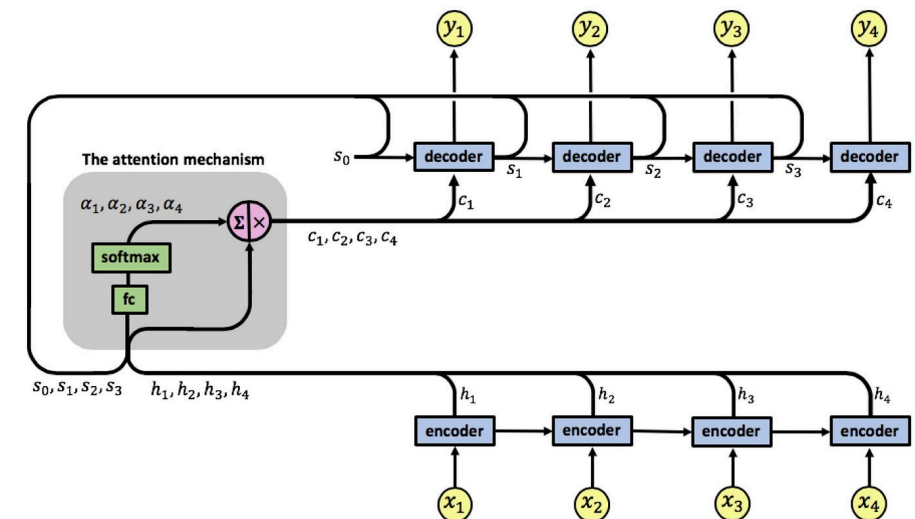
החלק האחרון בהסבר שלנו הוא **בניית וקטור הקשר דינמי C**. אנו משתמשים במשקולת תשומת הלב שחישבנו, ומרכיבים את הוקטור כמכפלה של משקולת זו במצב הפנימי של המקודד המקושר אליה. בנייה זו של הוקטור ההקשר מאפשרת לנו לתת למפענח את המידע הרלוונטי ביותר ביחס למצב הפנימי הנוכחי שלו. מצבים פנימיים של המקודד בעלי עוצמת קשר נמוכה ביחס למצב הפנימי של המפענח, לא ישפיעו על הוקטור C (שכן אם ערך משקולת תשומת הלב שלהם נמוך, חלקם היחסי בוקטור C יהיה נמוך גם כן).

$$C_i = \sum_{j=1}^{T_x} \alpha_{ij} * h_j \quad (3)$$

משוואה 3 - בניית וקטור הקשר הדינמי C_i

הקלט של המפענח הינו שקלול של המצב הפנימי הקודם שלו, וקטור ההקשר, ומוצא הפלט הקודם המוזנים לתוך הרשת בדומה לרשתות איטרטיביות קודמות שראינו (LSTM/RNN). פונקציית השקלול מורכבת מפונקציות האקטיבציה הפנימיות של בלוק ה-GRU (בדומה ל-LSTM ישנם שערי שכחה ועדכון). בניית פלט המפענח מתוארת באיור 6.

אם נחזור לאיור 1, נוכל לראות כי לאורך רוב האלכסון, רק מילה אחת מהפלט מיושרת (קשורה) באופן מובהק למילה בקלט, ולכן המצב הפנימי מהמקודד המתאים יועבר כמעט בשלמותו לווקטור ההקשר C_i (נוסחה 3). לעומת זאת, בחלקים שבהם מילה בפלט תלויה (בהתאם למשקולות תשומת הלב) בכמה מילים מהקלט, וקטור ההקשר יהיה מורכב מסכום משוקלל של מצבים פנימיים של המקודד. איור 6 מציג את הארכיטקטורה כפי שהיא מובאת במאמר בשלמותה.



איור 6 - תיאור הארכיטקטורה בשלמותה.

אז במה מנגנון תשומת הלב מסייע לנו?

מלבד היכולת לגרום למפענח להתרכז בקלט מסוים בעת החיזוי של המילה הנוכחית, מנגנון תשומת הלב עובד באופן דומה למנגנון skip connection ברשתות עמוקות. אנו מספקים גישה ישירה בין המצבים הפנימיים של המקודד למפענח באמצעות הוקטור הדינמי שאנו יוצרים, ובכך מאפשרים למידע הקיים בהם "לזרום" כמעט ללא שינוי בדומה לאופן פעולת skip connection. וזאת בשונה מארכיטקטורות מקודד-מפענח ללא מנגנון זה שחלק מהמידע אובד בעת יצירת הוקטור ביציאת המקודד. מנגנון זה נותן

מענה לשתי החסרונות המרכזיים של הרשתות האיטרטיביות: צוואר הבקבוק של ייצוג סדרות דאטה ארוכות ודעיכת הגרדיאנט. בנוסף, מנגנון תשומת הלב מאפשר "פרשנות" (Interpretability) טובה יותר לרשת. פרשנות היא מושג המתאר את היכולת שלנו בתור בני אדם להבין את התהליכים המתרחשים בתוך רשתות לומדות. בהקשר של הארכיטקטורות שמשתמשות במנגנון תשומת לב, ניתן להבין כיצד המודל מייצר את הפלט באמצעות המשקל שכל יחידת מידע קיבלה, ומתוך כך ללמוד את מגבלות הרשת ולמצוא דרכים לשפר אותה.

מה היו החסרונות של הארכיטקטורה שראינו עד כה?

אף על פי שהארכיטקטורה שהצגנו היוותה התקדמות עצומה בתחום ניתוח שפה טבעית, היו לה מספר מגבלות.

המגבלה הראשונה נבעה מכמות המשאבים שנדרשו לחישוב וקטור ההקשר C , שהוסיף על עומס החישוב הקיים גם כך ברשתות איטרטיביות. כעת נדרשות $O(n * m)$ הפעולות של פונקציית היישור (alignment score) כאשר m מייצג את מספר הטוקנים בקלט ו- n הוא מספר הטוקנים בפלט. דבר זה גרם לזמני אימון והסקה (inference) ארוכים במקודד.

המגבלה השנייה של הרשת הינה ייצוג ההקשר המוגבל שלה (limited context representation). הרשת לומדת את הקשר בין קלט לפלט. אולם, היא אינה לומדת את ההקשרים שיחידות המידע יוצרות אחת עם השנייה (תלויות פנים) בקלט ובפלט. לכן, היא לא יכולה "להבין" סמנטיקה מורכבת כגון סלנג, סרקזם, כפל משמעות, ומערכות יחסים עקיפות החבויות בקלט (שבתור בני אדם אנו מבינים בקלות). דבר זה מוביל לכך שביצועי הרשת פוחתים ביחס ישיר לאורך הסדרה.

כיצד ניתן לפתור בעיה זו?

מנגנון תשומת הלב אותו תיארנו עד כה נקרא תשומת לב מוצלבת (cross attention). שכן, היא (תשומת הלב) מצליבה את הידע מהמקודד עם זה שהתקבל מהמפענח. עתה, נציג מנגנון תשומת לב נוסף, **תשומת לב עצמית (self attention)** שהוצג לראשונה [במאמר זה](#). תשומת לב עצמית מחפשת את עוצמת הקשר בין כל טוקן עם כל טוקן אחר באותה סדרה. נזכיר, שבארכיטקטורות מקודד-מפענח, גם המקודד וגם המפענח מקבלים סדרה במהלך האימון, ולכן ניתן לשבץ את מנגנון תשומת הלב העצמית בכל אחד מהם. בהקשר זה **אחד החידושים המרכזיים של ארכיטקטורת הטרנספורמרים היה השילוב של שני מנגנונים אלו.**

אז מדוע תשומת לב עצמית נדרשת מלכתחילה?

כאמור, אחת המגבלות של הארכיטקטורה הקודמת שהצגנו הייתה יכולת מוגבלת לעבד טקסטים מורכבים. מכיוון שהרשת חיפשה **רק את עוצמת הקשר** שבין מילה בשפה א' לקבוצת המילים בשפה ב' הדרושות לתרגום שלה (או במילים אחרות בנתה את וקטור ההקשר רק מתוך המצבים הפנימיים של המקודד). אולם, כאשר אנו ניגשים למשימת תרגום, אנו צריכים למפות את התלויות המורכבות במשפט המקור על מנת לתרגם בצורה נכונה. על ידי שילוב של מנגנון תשומת הלב העצמית, ניתן לשקול את הקשר שבין מצב פנימי אחד לאחר, בנוסף על בחינת הקשר שבין מצב פנימי במקודד למפענח. נמחיש בעיה זו באמצעות דוגמא המציגה תרגום מאנגלית לעברית, ונראה כיצד ארכיטקטורה המשלבת את שני מנגנוני תשומת הלב תתרגם את המשפט לעומת ארכיטקטורה עם מנגנון תשומת לב מוצלבת בלבד. נניח והמשפט אותו אנו מעוניינים לתרגם הוא:

"Despite the stormy weather causing some delays, the couple, who were accompanied by their close friends, managed to reach the mountaintop and enjoy the breathtaking view."

ארכיטקטורה המשתמשת בשני המנגנונים תתרגם את המשפט באופן הבא:



"למרות האיחורים שגרם מזג האוויר הסוער, הזוג, שליוו אותו חברים קרובים, הצליח להגיע לפסגת ההר ולהנות מהנוף עוצר הנשימה"

לעומת זאת ארכיטקטורה המשתמשת רק במנגנון תשומת הלב המוצלבת עלולה לתרגם את המשפט כך:

"הזוג, שליוו אותו חברים קרובים, הצליח להגיע לפסגת ההר, למרות האיחורים ומזג האוויר הסוער, ולהנות מהנוף עוצר הנשימה."

למרות ששני התרגומים קוהרנטיים ושמרו על כללי תחביר ודקדוק, התרגום השני נכשל בהבנת הקשר שבין האיחור למזג האוויר. לעומת זאת, הארכיטקטורה שכן משתמשת בתשומת לב עצמית הצליחה למצוא קשר זה, ולהביא אותו לידי ביטוי בתרגום.

פרק 4 - ארכיטקטורת הטרנספורמרים

כפי שראינו בפרק הקודם, הארכיטקטורות שעשו שימוש במנגנוני תשומת הלב לבדם לא שרדו את מבחן הזמן, והוחלפו על ידי טרנספורמרים בכל המשימות הקשורות לניתוח שפה טבעית.

בפרק זה אנו נבצע ניתוח מעמיק של ארכיטקטורת הטרנספורמרים, ונענה על השאלות הבאות:

- כיצד הטקסט מוזן למודל הטרנספורמר?
- מהו קידוד תלוי מיקום, ומדוע הוא משחק תפקיד חשוב בארכיטקטורת הטרנספורמרים?
- מהם תפקידים של המקודד והמפענח בטרנספורמרים, וכיצד הם עובדים יחד?
- כיצד מנגנון תשומת הלב בא לידי ביטוי בטרנספורמרים?
- מהו הייחוד של מנגנון תשומת הלב בטרנספורמרים ביחס לחישוב תשומת הלב ברשתות איטרטיביות, ולמה תשומת לב היא אכן כל **מה שאנו זקוקים לו?**
- כיצד טרנספורמרים פתרו את המגבלה העיקרית של הארכיטקטורות שקדמו להן, העיבוד הטורי של הדאטה?

אנו נבנה את המאמר בצורה של בבושקה (או מטריושקה ברוסית תקנית). המאמר ייבנה כסדרה של קופסאות שחורות, שכל אחת מהן תזכה לתת-פרק שבו נסביר מה היא טומנת בתוכה. צורת העבודה תהייה "מלמעלה-למטה" (top -> down), בכל שלב אנו נחשוף אבן בניין נוספת בארכיטקטורה, וחלקים אחרים שלה נותרו כקופסאות שחורות. אף על פי שמבנה המודל פשוט באופן יחסי, הוא סובל מתדמית של נושא מורכב ומאתגר להבנה. זאת מכיוון שהוא בנוי ממספר חלקים המבוססים על עקרונות מופשטים כך שאם הם אינם מובנים לעומק, קשה לחברם יחד לכדי רעיון כולל. מסיבה זו החלטנו לבנות את ההסבר, כך שבכל שלב נוכל להתמקד ברעיונות ספציפיים מאחורי אבן בניין מסוימת, ואחרות להשאיר בתור "קופסא שחורה" שניתן יהיה להגדיר את הקלט והפלט שלה, מבלי לצלול להסבר על אופן הפעולה שלה. אנו מקווים שדבר זה יקל על הקורא.

- **קופסא ראשונה:** בחלק זה כל המודל הוא קופסה שחורה, מלבד חלקי השיבוץ והקידוד תלוי מיקום של הטוקנים (token embedding and positional encoding) שזוכים לפרקים משלהם. אנו נדבר על הקלט והפלט של המודל. נסביר כיצד הקלט מאורגן כך שניתן יהיה לעבדו באופן מקבילי, וכיצד הפלט נבנה באופן אוטורגרסיבי. בנוסף, נדבר על "שרשרת החיול" של הקלט, החל מקטע טקסט בשפה טבעית דרך קידוד למרחב חבוי והפיכתו בחזרה לטקסט בשפה טבעית במוצא הרשת.

- **קופסא שנייה:** בחלק זה נתעמק במבנה הפנימי של המקודד והמפענח. נסביר מהו התפקיד שלהם, וכיצד הם עובדים יחד. כאן נשאיר את החלקים הפנימיים של המקודד והמפענח כקופסאות שחורות (מנגנון תשומת הלב, שכבת ה-feed-forward וכו').

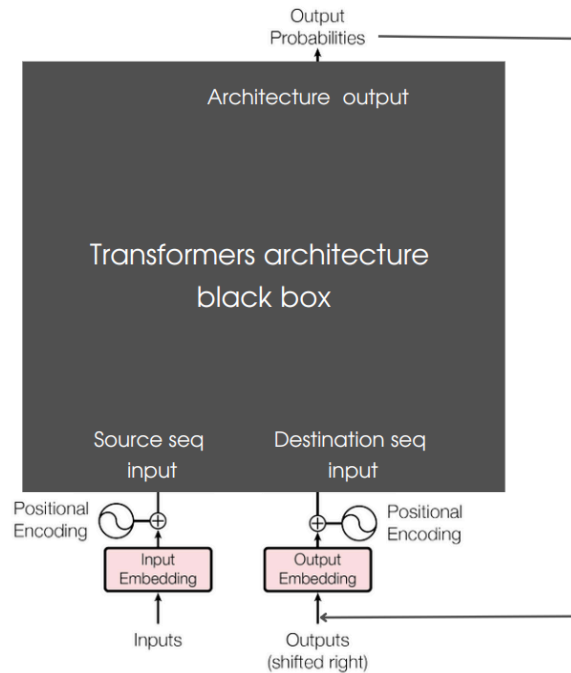
- **קופסא שלישית:** בחלק זה נספק הסבר מקיף על הבלוקים המרכיבים את המקודד והמפענח. נרחיב על מבנה הארכיטקטורה של כל אחד מחלקים אלו ונפתח למעשה את כל הקופסאות השחורות שנותרו לנו מהשלב האחרון. נציג לראשונה את הרעיון המרכזי שהרשת בנויה סביבו, והוא ייצוג הקלט בתור מפתח, שאילתה, וערך (key, query, value), שעליהם מבוסס מנגנון תשומת הלב של טרנספורמרים.

במאמר זה אנו נדון בנושאים הבאים:



- תשומת לב עצמית ותשומת לב מוצלבת.
- פלטיי הביניים של המקודד והמפענח ובפרט מהו הפלט אחריי שכבת תשומת הלב, והפלט הסופי של המקודד.
- מהו הקלט למנגנון תשומת הלב המוצלבת ומנגנון תשומת הלב העצמית.
- מהו מנגנון תשומת הלב הרב ראשית.
- מדוע נדרשת שכבת הנרמול (layer normalization) ?
- נראה כיצד skip-connections ושכבות ה feed forward ממלאות תפקיד אינטגרלי בארכיטקטורה איך הם משפיעים על ביצועי המודל.

קופסה ראשונה: הארכיטקטורה, מבט מלמעלה



איור 1 - ייצוג הארכיטקטורה כקופסה שחורה

אנו נחלק פרק זה לשני חלקים עיקריים. הנושא הראשון יעסוק בקלט של המודל ויכיל בתוכו את התהליך שעובר הקלט מקבלתו כטקסט בשפה טבעית ועד לשלב שניתן להזינו לתוך הרשת לצורך האימון. בחלק השני אנו נעסוק בפלט של המודל עד להפיכתו למילה בשפה טבעית שוב.

חלק ראשון: קלט הרשת

כאשר אנו מדברים על מודלים לעיבוד שפה טבעית, אנו מגדירים את המילון המשמש את מודל. מילון זה מכיל את כל המילים שהמודל מכיר, כאשר לא ניתן יהיה להשתמש במילים שלא נמצאות במילון זה או שלא ניתן להרכיב ממילים אחרות המוכרות על ידי המילון.

לפני שנתאר את מבנה הארכיטקטורה, נרחיב על תהליך העיבוד של הקלט שבסופו הוא מיוצג במרחב וקטורי חדש. תהליך זה כולל שני שלבים עיקריים:

1. טוקניזציה של הטקסט.
2. שיבוץ הטוקנים שהתקבלו במרחב וקטורי.

נדגים כיצד שני שלבים אלו באים לידי ביטוי באמצעות הקלט הבא:

"The stars danced across the velvet sky, painting the night with their celestial beauty."

טוקניזציה (tokenization):

השלב הראשון בעיבוד הקלט הוא טוקניזציה. פעולה זו מחלקת את הקלט למילים (בתוך קטגוריית זו אנו כוללים גם סימני פיסוק, וחלקי מילים). ישנם טוקניזרים שונים שיחלקו את אותה המילה באופן שונה. לדוגמא, המילה don't יכולה להתחלק באופנים הבאים: ['don't'] או ['don', '#'] (הסבר מפורט על הטוקניזרים השונים ניתן למצוא כאן). לאחר הפעלת טוקניזציה בסיסית (כל טוקן הינו מילה או סימן פסוק) על המשפט שנתנו כדוגמא, נקבל את הפלט הבא:

```
tokens = ["The", "stars", "danced", "across", "the", "velvet", "sky", ",", "painting", "the", "night",  
"with", "their", "celestial", "beauty", "."]
```

מודלי שפה מודרניים (למשל אלו מ- "סדרות" BERT או GPTs למיניהם) משתמשים בשיטות טוקניזציה מתקדמות, בהם חלק מהטוקנים הינם תת-מילים לא מילים שלמות. העיקרון המוביל בשיטות טוקניזציה אלו ([WordPiece](#) ו-[BytePair Encoding](#)) הוא הקניית טוקנים למילים או תת-מילים השכיחים ביותר בסט הנתונים (dataset) עליו הוא מאומן (בדרך כלל סט זה הינו מגוון ועצום בגודלו). בנוסף, [BERT](#) מגדיר שני טוקנים מיוחדים לתוצר הטוקניזציה: SEP, CLS. התפקיד של SEP הוא להפריד בין משפטים. טוקן - CLS משמש ליצירת ייצוג וקטורי של מקטעי טקסט (הסבר מעמיק יותר ניתן למצוא [בלינק](#)). נציין כי מודלים אחרים מגדירים טוקנים מיוחדים בצורה אחרת.

השלב הבא בפעולת הטוקניזציה הוא מתן מספר מזהה ייחודי לכל טוקן שקיבלנו (כלומר כל טוקן מקבל מספר סידורי). במידה וישנן מילים במשפט שלא נמצאות במילון שלנו, הן יחולקו לתתי טוקנים, עד שכל מילה בקלט תקבל מספר מזהה ייחודי (או קבוצת מספרים במידה ופעולת הטוקניזציה חילקה אותה למספר טוקנים). בדוגמא שלנו, הפלט של שכבה זו עבור קטע טקסט מסוים יכולה להיות:

```
token_ids = [101, 1996, 3340, 5228, 101, 1996, 16441, 3712, 1010, 101, 1996, 2305, 2007,  
2037, 12631, 5053, 1012, 102]
```

השלב השני בעיבוד הקלט הוא שיבוץ טוקנים נלמד (Learnable Token Embedding) במרחב וקטורי. המימדים השונים בוקטור מייצגים מאפיינים סמנטיים ותחביריים שונים שלו. כתוצאה מכך ניתן למדל את מערכת היחסים בין הטוקנים באמצעות פעולות אריתמטיות. בארכיטקטורות שקדמו לטרנספורמרים, נעשה שימוש במודלים שאומנו במיוחד בשביל לבנות שיבוץ זה (לדוגמא [word2vec](#)). לעומת זאת, בארכיטקטורת הטרנספורמרים שיבוץ הטוקנים משולב בתהליך האימון.

כעת נתאר את שני השלבים האחרונים שהקלט עובר עד שניתן להזינו למקודד ולמפענח: תהליך השיבוץ של הטוקנים וקידוד תלוי מיקום (positional encoding). אנו נסביר את החלקים שאינם מוסתרים על ידי קופסא שחורה באיור 1 כלומר Input Embedding - Positional Encoding).

שיבוץ הטוקנים (Tokens Embedding):

מוצא שכבת השיבוץ נתון על ידי המשוואה הבאה:

$$\text{Embedded vector} = E * X[i] * \sqrt{d_{\text{model}}}$$

משוואה 1 - שיבוץ הטוקנים למרחב נסתר

- X - מטריצה הבנויה מ [one hot encoded vectors](#) שגודלה $V \times V$, כאשר V מסמן את מספר הטוקנים במילון. גודל המטריצה X נובע מכך שגודלו של כל וקטור ה-one-hot הוא V וישנם V וקטורים כאלו.
- i - אינדקס הטוקן המשובץ.
- d_{model} - מימד מרחב הקידוד (שיבוץ).
- E - מטריצת שיבוץ נלמדת שגודלה $V \times d_{\text{model}}$, מתפקדת כ-LUT המכילה את השיבוצים של כל הטוקנים, כלומר אנו משבצים V טוקנים למרחב בגודל d_{model} .

בפועל, המכפלה $E * X[i]$ היא גישה למיקום i בטבלה E . צורת ייצוג זו היא יעילה מבחינה חישובית, כך ניתן לנצל את הטבלה לחישוב שיבוץ הטוקנים של הקלט בו זמנית (במקביל). כותבי המאמר "[Attention is All You Need](#)" לא מציינים מהי הסיבה להכפלת מוצא השכבה הלינארית בגודל $\sqrt{d_{\text{model}}}$. אולם, ישנן השערות שהדבר אמור למנוע את דעיכת הגרדיאנטים שכן השיבוץ עלול ליצור ערכים גבוהים מדי שיובילו לרוויה של פונקציית ה-softmax במנגנון תשומת הלב (אשר נרחיב עליו בהמשך). השערה אחרת טוענת כי המטרה הינה למנוע מערך קידוד תלוי המקום (שנסבירו בהמשך) להיות דומיננטי. אולם, אלו הן רק השערות ויש להתייחס אליהם בהתאם.

כעת נשאלת השאלה מדוע אנו משקיעים בקידוד עצמאי של הקלט ולא משתמשים ברשתות שאומנו מראש (כגון word2vec) במיוחד עבור מטרה זו?

הדבר נובע משלוש סיבות מרכזיות:

1. במשימות הכרוכות ניתוח שפה כל מודל מגדיר מהי שיטת השיבוץ והמילון המשמשים לצרכי הטוקניזציה של הטקסט. מתוך כך נובעת ההבנה מדוע לא ניתן להשתמש במודל שיבוץ טוקנים שאומן עם גישה אחרת לטוקניזציה (כלומר אי אפשר להשתמש ב-word2vec לטוקניזציה של טרנספורמרים המשתמשים בשיטת טוקניזציה כמו wordpiece או byte pair encoding).
2. מכיוון שניתן להשתמש באותה המילה בהקשרים שונים או בכפל משמעות, עלינו למצוא ייצוג ממוצע עבור הטוקנים המייצגים את המילה במרחב וקטורי. דוגמא לכך יכולה להיות המילה bat, שיכולה להופיע בהקשר של משחק כדור (bat = מחבט) או בהקשר של זואולוגיה (bat = עטלף). מכיוון ששפה טבעית הינה בעיה מורכבת לאפיון, נדרשת כמות עצומה של דאטה כדי לבנות מודלים שמסוגלים ללמוד את מגוון הקשרים החבויים בתוכה. ייצוגי טוקנים המופקים באמצעות מודלים כגון word2vec שאומנו על כמות דאטה קטנה הרבה יותר מזו של הדאטהסטים העצומים שמשתמשים בהם לאימון טרנספורמרים, אינם מסוגלים להכיל את כל הקשרים המורכבים בין הטוקנים.
3. כפי שאמרנו בעבר, מנגנון תשומת הלב לומד את חשיבותה של יחידת קלט אחת אל מול יחידת קלט אחרת כתלות בערכה. ולכן, הדבר ההגיוני ביותר לעשות הוא לתת לרשת ללמוד את השיבוץ שהכי מתאים לה באופן עצמאי.

קידוד תלוי מיקום (Positional Encoding)

החדשנות של ארכיטקטורת הטרנספורמרים נבעה מהניסיון לענות על השאלה הבאה: **כיצד ניתן לוותר על ההפעלה האיטרטיבית של הרשת בעיבוד סדרות?** הפתרון ההגיוני ביותר הוא ניתוח כל הקלט במקביל. אבל כיצד ניתן לבצע זאת? אחת הדרכים לעשות זאת היא ייצוג הקלט כסט (מבנה נתונים שבו הסדר איננו רלוונטי) דבר המאפשר לנו להזין אותו כמקשה אחת. אולם, הקלט למודל הינו סדרתי. אז כיצד ניתן להתגבר על סתירה זו? הפתרון הוא לספק למודל מידע על הסדר של יחידות הדאטה, כלומר, מידע על מיקום המילה בטקסט שיאפשר למודל להבין את המשמעות של מרחק בין מילים. במידה ולא נספק למודל מידע זה מנגנון תשומת הלב שארכיטקטורת הטרנספורמרים מבוססת עליו עלול לנתח באופן זהה את שני המשפטים הבאים:

"Tom bit a dog." | "A dog bit Tom."

המידע שאנו מוסיפים לקלט מאפשר למודל ללמוד את המיקום של כל יחידת דאטה בטקסט ואת מרחקה היחסי מכל יחידת דאטה אחרת. הוספת מידע זה מאפשרת לרשת ובפרט לפונקציית תשומת הלב להתחשב במיקומם של החלקים השונים של הקלט כאשר היא שוקלת את חשיבותה של מילה בקלט ביחס למילה אחרת. האופן שבו הרשת מוסיפה מידע זה נקרא **קידוד תלוי מיקום** (Positional encoding) או (Positional embedding).

קידוד תלוי מיקום בטרנספורמרים

כיצד מבצעים קידוד תלוי מיקום? נזכיר שמטרתנו היא להעניק למודל יכולת ללמוד מה המרחק בין יחידות הקלט. אילוץ נוסף שאנו מעוניינים בו הוא שהקידוד יהיה ייחודי עבור כל מיקום בקלט, אחרת לא ניתן יהיה להבחין בין מילים במיקומים שונים. הפתרון הנאיבי הוא להשתמש ב-"one-hot encoding". בקידוד זה אנו יוצרים ווקטור שאורכו שווה לאורך סדרת הקלט עבור כל טוקן, ומאתחלים את ערכיו לאפסים, במקום בו מופיעה המילה אנו מציבים 1.

ניקח לדוגמא את המשפט הבא:

"The quick brown fox jumps over the lazy dog near the blue river."

דוגמא זו מייצגת את הבעיה שקידוד זה מעמיד בפנינו, והיא ש-one-hot encoding הוא קידוד שווה-מרחק (equidistant), כלומר, כל וקטור מרוחק מכל וקטור אחר ב- $\sqrt{2}$. בדוגמא שלנו המילה "the" מופיעה 3 פעמים, ולכן הרשת תתקשה לשייך כל "the" למילה המקושרת אליה (dog, fox, river). מתוך כך עולה השאלה: **כיצד ניתן לקודד מילה כך שמיקומה במשפט יקנה לה ערך ייחודי ובנוסף שהמרחק בין הקידודים ישקף את מרחק בין מיקומי המילים בקלט?** במילים אחרות אנו רוצים קידוד המקרב את ערכם של כל זוג וקטורים המייצגים טוקנים קרובים (מבחינת מרחקם בקלט) ומרחיק כל שני וקטורים המייצגים מילים רחוקות.

אז מהו הפתרון?

קידוד תלוי מיקום באמצעות פונקציות מחזוריות:

$$\begin{aligned} \text{Positional Encoding}(pos, 2i) &= \sin(pos / (10000^{2i/d_{\text{model}}})) \\ \text{Positional Encoding}(pos, 2i + 1) &= \cos(pos / (10000^{2i/d_{\text{model}}})) \end{aligned}$$

משוואה 2- קידוד תלוי מיקום



כאשר:

- pos - מיקום הטוקן בסדרה המקורית.
- i - האינדקס בתוך המרחב השיבוץ d_{model} כאשר מתקיים: $i \in \{d_{model}\}$.

כעת נסביר את המשוואה עבור $d_{model} = 512$ (כפי שהוא מוגדר במאמר המקורי). אנו מחשבים את הוקטור עבור מיקום (pos) של כל טוקן בסדרה, כאשר החישוב הוא לפי פונקציית הסינוס עבור מימדים זוגיים של וקטור הקידוד ופונקציית הקוסינוס למימדים האי-זוגיים. מכיוון שערכים אלו קבועים לאורך כל השימוש במודל אנו מחשבים אותם באתחול ומשתמשים בהם באמצעות LUT (Look Up Table). לבסוף אנו מחברים את הוקטור הקידוד שהתקבל עם וקטור השיבוץ. החוקרים לא מסבירים באופן מפורש מדוע הם מחברים את וקטור הקידוד עם וקטור השיבוץ, ולא משרשרים אותו אילו.

אז מדוע השימוש בפונקציות מחזוריות מספק את הדרישות שניסחנו בתחילת הפרק הקודם?

נזכיר את הדרישות שאנו מבקשים עבור וקטור הקידוד ונראה כיצד הפונקציות המחזוריות מספקות מענה לכל אחת מהן.

• למידת מיקום יחסי (relative position):

כפי שתיארנו בפתיחה, על המודל ללמוד כיצד לאמוד מרחק. אולם, אנו לא מעוניינים ללמוד את המרחק כיחידה אבסולוטית (כלומר מרחק של טוקן כלשהו מהטוקן הראשון בסדרת הקלט), אלא את **המרחק בין מיקומי הטוקנים בסדרה**. המשמעות הינה, שאנו מקבעים טוקן מסוים בסדרה, ומבקשים לאמוד את המרחק בינו לכל הטוקנים בקלט. לדוגמא, אם המיקום האבסולוטי של הטוקן הוא 67, ואנו רוצים ללמוד את חשיבותו ביחס לטוקן במיקום ה-47 אזי המרחק היחסי ביניהם יהיה 20-.

במילים אחרות, אנו בוחנים את המרחקים בין $pos1$ ו- $pos2$ המקיימים:

$$pos2 = pos1 + k$$

בפשטות, אנו מספקים לרשת את ייצוג של $pos1$ ו- $pos2$ ומבקשים מהמודל "להפיק" ייצוג של המרחק k ביניהם.

כיצד פונקציות טריגונומטריות עוזרות לנו במקרה זה? התשובה לכך טמונה בעובדה שפונקציות אלו מקיימות את הזהויות הטריגונומטריות הבאות:

$$(1) \sin(pos + k) = \sin(pos) * \cos(k) + \sin(k) * \cos(pos)$$

$$(2) \cos(pos + k) = \cos(pos) * \cos(k) - \sin(pos) * \sin(k)$$

משוואה 3 - זהויות טריגונומטריות עבור סכום בתוך פונקציית סינוס וקוסינוס

ולכן, חישוב המרחק בין שיבוצי טוקנים מרוחקים אחד מהשני ב- k טוקנים באמצעות מנגנון תשומת הלב, יוליד ביטויים התלויים במיקום האבסולוטי (pos) והמרחק היחסי (k).

• קידוד ייחודי לכל וקטור (unique embedding):

השאלה הראשונה שנשאלת, היא מדוע אנו משלבים את הפונקציות הטריגונומטריות \sin ו- \cos ? (כלומר מדוע המודל לא משתמש רק בפונקציית \sin או רק בפונקציית \cos). המאמר לא מספק תשובה חד משמעית עבור שאלה זו, אולם אחת ההשערות הינה ששילוב של שתי פונקציות אלו יוצר ייצוג עשיר יותר עבור המידע המיקומי מאשר שימוש בפונקציה יחידה.



השאלה השנייה שנשאלת הינה מדוע המחברים בחרו להשתמש דווקא בביטוי:

$$pos / (10000^{2i/d_{model}})$$

משוואה 4 - הביטוי המעריכי בתוך פונקציית הקידוד

על מנת להבין מדוע אנו משתמשים דווקא בביטוי זה עלינו להבין את מרכיביו ואת התפקיד שלהם:

- pos - הדרישה הבסיסית של קידוד תלוי מיקום היא ליצור וקטור ייחודי עבור כל טוקן בקלט. ולכן, הדבר ההגיוני ביותר לעשות הוא להשתמש במיקום שלו בקלט (pos), המגדיר אותו באופן ייחודי.

- $2i/d_{model}$ - מכיוון שוקטור הקידוד מתווסף לוקטור השיבוץ, הוא מכיל d_{model} מימדים, ולכן יש צורך לקבוע מה יהיה הערך שיקבל כל מימד.

הפתרון הנאיבי הוא להשתמש בערכו של $\sin(pos)$ או $\cos(pos)$ עבור כל המרכיבים של וקטורי הקידוד. אולם, הבעיה בנישה זו הינה קיום מחזוריות במרחקים בין וקטורים. כלומר, עבור ערך m כלשהו, התלוי במורכבות הפונקציה, המרחקים בין וקטורים המקודדים את מיקומם של טוקנים הנמצאים במרחקים m ו- $2m$ למשל עלולים להיות כמעט שווים. זהו מצב לא רצוי מכיוון שהקידוד אמור לשקף את המרחק בין הטוקנים.

במידה והדבר לא מתקיים המודל יתקשה להפיק מוקטורים אלו את מלוא המידע על המרחק בין הטוקנים בסדרת הקלט. ככל שנשתמש בפונקציה מורכבת יותר, כך נקטין את הסיכוי למחזוריות שכזו (ערכו של m גדל ככל שהפונקציה מורכבת יותר).

הפתרון המוצע הוא לתת לכל מימד בוקטור ערך שונה. דבר זה מתבצע באמצעות מכפלה של pos במימד i וחלוקתו ב- d_{model} .

כלומר אנו יוצרים את המערך הבא:

$$[pos/d_{model}, pos * 2/d_{model}, \dots, pos * i/d_{model}, \dots, pos], \forall i \in [d_{model}]$$

- הוספת בסיס החזקה של 10000 יוצרת מורכבות שאינה מאפשרת חזרתיות גם כאשר מספר הטוקנים גדל בסדרי גודל של עשרות אלפים.

על מנת להוכיח את התזה שהצענו כאן, נבחן מה הייתה יכולה להיות האלטרנטיבה הפשוטה ביותר וכיצד הייתה נראית תוצאתה.

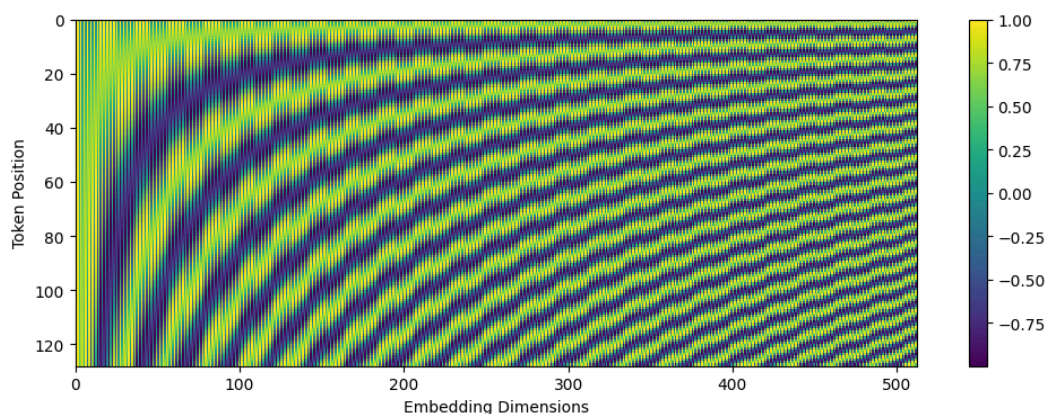
למשל עבור קידוד תלוי מיקום השווה ל:

$$\begin{aligned} \text{Positional Encoding}(pos, 2i) &= \sin(pos * (2i/d_{model})) \\ \text{Positional Encoding}(pos, 2i + 1) &= \cos(pos * (2i/d_{model})) \end{aligned}$$

משוואה 5 - קידוד תלוי מיקום ללא המרכיב המעריכי

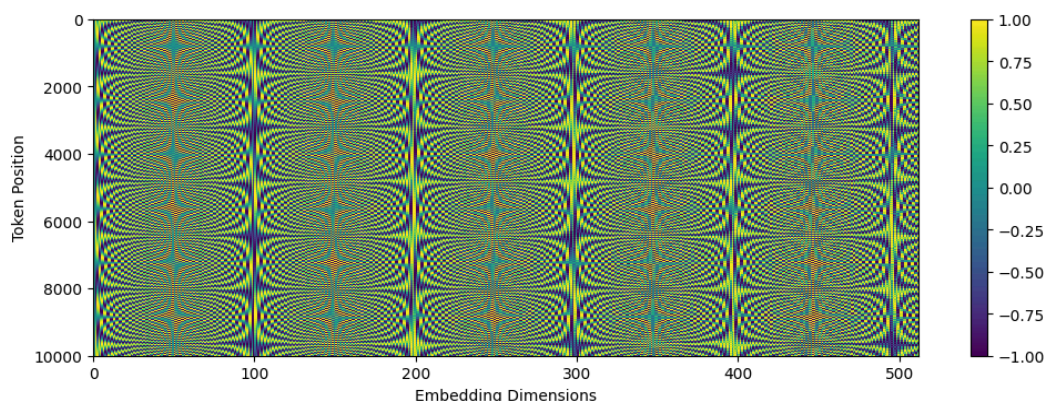
כאשר $d_{model} = 512$ - מספר הטוקנים הוא 128, נקבל את הגרף הבא:





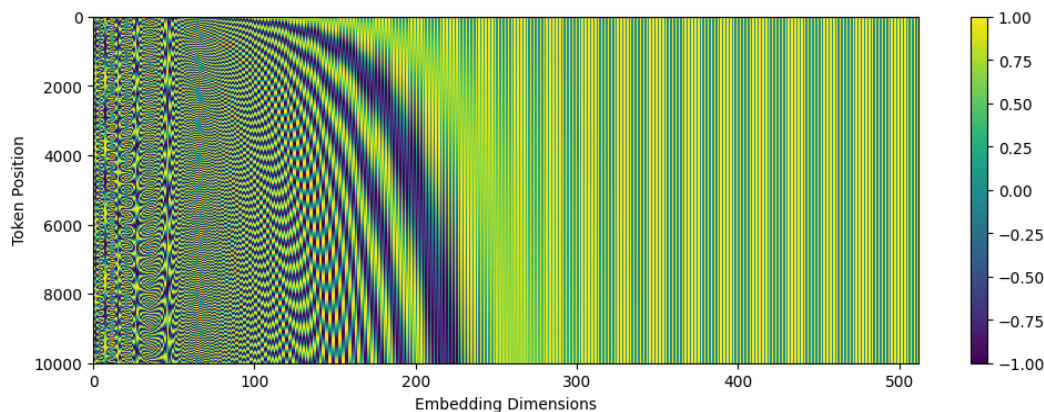
איור 2 - קידוד תלוי מיקום אלטרנטיבי, עבור 128 טוקנים

לכאורה לא נראית בעיה גלויה לעין. אולם, אם נגדיל את מספר הטוקנים ל-10,000 כאשר $d_{model} = 512$ (אורך הקשר ריאלי עבור טרנספורמרים בני ימינו) נקבל את הגרף הבא:



איור 3 - קידוד תלוי מיקום אלטרנטיבי, עבור 10,000 טוקנים

כפי שניתן לראות, כאשר מספר הטוקנים בסדרה, גדול מ- d_{model} נוצרת חזרתיות בדפוסי קידוד תלוי המיקום. למרות שניתן לפתור את זה באמצעות הגדלת ערכו של d_{model} , זה לא פתרון ריאלי. כותבי המאמר מציעים את הקידוד המעריכי. הסיבה שהקידוד המעריכי עובד בצורה כל כך טובה, נובעת מהעובדה שהוא מדכא חזרתיות עבור כל גודל סדרת קלט בכניסה, כפי שניתן לראות באיור 4 עבור סדרה בעלת 10,000 טוקנים.



איור 4 - קידוד תלוי מקום מעריכי עבור 10,000 טוקנים

שיטות נוספות לביצוע קידוד תלוי מקום:

קידוד תלוי מיקום אינו מחויב לאופן בו הוא מוצג במאמר המקורי, וישנן שיטות נוספות. אנו לא נסקור שיטות אלו במאמר זה (אך ניתן למצוא אותן ב[1], [2], ו-[3] או בסקירה עתידית שנבצע) אולם נתעכב על נקודה אחת: מדוע החוקרים משתמשים דווקא בייצוגים באמצעות פונקציות מחזוריות, ולא לומדים את הקידוד כחלק מתהליך האימון. החוקרים מציינים במאמר המקורי שהם ביצעו ניסויים עם קידוד נלמד, וראו כי התוצאות לא השתנו לעומת קידוד קבוע מראש. מכיוון שאין הבדל בביצועים אנו נעדיף לבצע קידוד ידוע מראש ולא נלמד, מכיוון שהדבר יעיל יותר מבחינה חישובית.

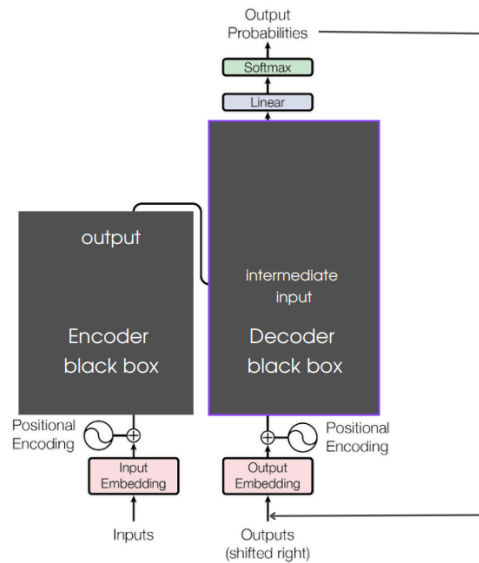
חלק שני : פלט הרשת

פלט המודל הוא תוצר של שכבת softmax על פני כל מילון הטוקנים שאנו משתמשים בו, כאשר הרשת מייצרת בכל הפעלה את הטוקן החזוי הבא ביחס לקלט. במהלך האימון אנו בונים את הפלט של המודל טוקן אחרי טוקן עד שגודל הסדרה שווה לגודל הסדרה המקורית, ואז מבצעים את פונקציית ה-loss ביחס לפלט המקורי של הרשת.

במהלך השימוש במודל (inference) אנו לא יודעים מה אמור להיות אורך משפט המוצא. לכן עלינו להפסיק את הפעלת המודל בנקודה כלשהי. ישנן שיטות שונות לקביעת נקודת העצירה:

- קביעת סף למספר הטוקנים שהמודל יכול לייצר עבור קלט מסוים.
- שימוש בטוקן מיוחד הנקרא EOS שתפקידו להצביע כי המודל סיים יצירת המשפט.
- בחינת וודאות (confidence) של המודל בטוקן אותו הוא יצר, והפסקת הריצה במידה וערך זה לא חוצה סף מסוים.

קופסא שנייה: מקודד ומפענח



איור 5 - המקודד והמפענח כקופסאות שחורות

המבנה הכללי:

ארכיטקטורת הטרנספורמרים שהוצגה במאמר המקורי (Attention is All You Need) יועדה להמרת סדרת קלט (source) לסדרת פלט אחרת (target). הארכיטקטורה, בדומה לארכיטקטורות שקדמו לה בנויה כמקודד ומפענח (ישנם טרנספורמרים, הבנויים ממקודד בלבד, או מפענח בלבד ואנו נדון בהם באחד החלקים הבאים של סדרת מאמרים זו).

חשוב להבין כי במשימת תרגום שתי הסדרות המתקבלות בכניסת המקודד והמפענח, מקושרות אחת לשנייה, אך כל אחת מהן מתוארת באמצעות ייצוג שונה (בדמות שפה). לדוגמא, משפטים בעברית ובאנגלית המתארים סיפור על ילד שמשחק בכדור, מכילים את אותו התוכן, אך שונים באופן שבו הם מביעים אותו. **הבעיה הניצבת בפנינו במשימה זו הינה להבין כיצד ממירים מייצוג אחד לייצוג אחר, במיוחד במשימות בהן חוקי המיפוי מורכבים.**

זוהי תמצית השימוש בטרנספורמרים כפי שהיא מופיעה במאמר המקורי שיועד למשימות תרגום וסיכום טקסט. בסופו של דבר, תכלית המודל **שהוצע במאמר** הינה ללמוד כיצד לבצע את הטרנספורמציה (המרה) בין שני מרחבי ייצוג שונים המכילים את אותו המידע. יש לזכור שכיום משתמשים בטרנספורמרים למשימות מגוונות שבהן אנו לא לומדים טרנספורמציה בין מרחבי ייצוג של "אותו הדאטה" ולכן פסקה זו תקפה רק למאמר שאנו מנתחים.

מטרת המודל הינה לחזות את ייצוג הקלט במרחב היעד (השפה שמתרגמים אליה). על מנת לעשות זאת, המפענח מרכיב את משפט היעד טוקן אחריו טוקן. אנו מתחילים את חיזוי המשפט כסדרה ריקה, ובכל שלב של הפעלת המודל אנו מוסיפים טוקן נוסף שחזינו להיות חלק מסדרת הקלט של המפענח. על מנת ללמוד כיצד לעשות זאת, אנו מזינים למקודד את סדרת המקור (לדוגמא, הטקסט שאנו מעוניינים לתרגם) כדי שיבנה ייצוג המכיל את הקשרים בטקסט במרחב וקטורי המשותף לו ולמפענח (בהמשך נסביר מהו מרחב זה וכיצד בונים אותו). המפענח נעזר במידע בניתוח הקלט שלו (שנבנה טוקן אחריו טוקן)

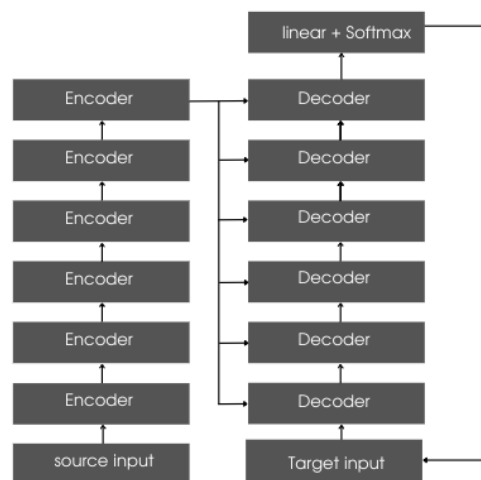
וביצירת הפלט שלו. אם נחזור לדוגמא שפתחנו איתה, המקודד מעביר למפענח את **תוכן הרעיון** שהמשפטים מייצגים, ללא תלות במרחבי הקלט (השפות השונות).

בדומה לארכיטקטורות קודמות שסקרנו, גם ארכיטקטורות הטרנספורמרים משתמשות במנגנון תשומת לב. אולם בשונה מהן, מנגנון זה הינו לב ליבה של הארכיטקטורה ולא מתפקד כמרכיב עזר שתפקידו לבנות ייצוג ביניים למידע המופק מהמקודד או מהמפענח. מכאן מגיע גם שם המאמר, כי תשומת לב היא הדבר היחידי שהמודל צריך.

המקודד והמפענח בנויים מ-"לבנים" (blocks) שכל אחד מהם נקרא **בלוק טרנספורמר** או בקצרה **טרנספורמר**. בלוקי טרנספורמר של המקודד והמפענח שונים אחד מהשני, כאשר המרכיב החשוב ביותר הוא מנגנון תשומת הלב הזהה בשניהם (נרחיב על כך בפרק הבא).

חשוב להבין כי מנגנון תשומת הלב עובד באותה הדרך גם במקודד וגם במפענח. אולם **האופן בו המידע מהמקודד מוזן למנגנון זה שונה מהאופן בו הוא מוזן במפענח**. הקלט של המפענח נבנה באופן אוטוגרסיבי, כלומר הפלט שלו הופך להיות חלק מהקלט הבא שלו. עקב כך אנו משתמשים בתשומת לב ממוסכת, ה"מסתירה" ממנגנון תשומת הלב את המידע על הטוקנים הבאים בסדרה אחרי הטוקן הנחזה.

המאמר מציין כי הבלוקים של המפענח והמקודד מוערמים (stacked) אחד על השני. איור 6 מציג כיצד חיבור זה מתבצע בפועל.



איור 6 - מערום (stacking) של בלוקי הטרנספורמר

תפקיד המקודד והמפענח

כפי שניתן לראות באיור 6, כל בלוקי המקודד מחוברים אחד לשני בטור, כאשר הפלט של כל בלוק מזין את הבלוק אחריו. בניגוד למקודד, המפענח מוזן בפלט המקודד המתקבל מהבלוק האחרון שלו, בתוספת לפלט מבלוק המפענח הקודם. הסיבה שהארכיטקטורה בנויה כך, טמונה בעובדה שבלוק המקודד האחרון מפיק את ייצוג המידע המופשט ביותר ומכיל את האינפורמציה המלאה ביותר עליו. לכן, הזנתו תסייע למפענח בהבנת ההקשר של הקלט שלו.

קלט ופלט:

כעת נגדיר מהו הפלט והקלט לחלקים השונים של הטרנספורמר. כאשר אנו מדברים על הקלט של המקודד או המפענח אנו תמיד נתייחס לסדרה שהתקבלה בכניסתם.

כעת נרחיב על הקלט והפלט של המקודד והמפענח:

המקודד:

- **קלט המקודד (encoder input)** - סדרת הקלט המקורית (source) אותה אנו רוצים לעבד.
- **פלט המקודד (encoder output)** - ייצוג וקטורי של הקלט המופק על ידי המקודד, כלומר הפלט של בלוק הטרנספורמר האחרון שלו. פלט זה מכיל מידע על הקשרים הקיימים בקלט.

המפענח:

- **קלט המפענח (decoder input)** - סדרת היעד (target) אותה אנו לומדים לחזות בצורה אוטורגרסיבית (טוקן לאחר טוקן). חשוב להבין כי חלק מסדרת היעד ממוסך (החלקים שבאים לאחר הטוקן הנחזה מוסתרים על מנת שהרשת לא תשתמש במידע זה). כאמור **הקלט הנוסף** של המפענח, המתקבל מהמקודד, הינו פלט הבלוק האחרון שלו.
- **פלט המפענח (decoder output)** - קלט לשכבה לינארית שמטרתה לחשב התפלגות של הטוקן הבא בסדרת הפלט הסופית (כגון טקסט). כמו במקודד, הפלט של המפענח נוצר על ידי בלוק הטרנספורמר האחרון שלו.

עקרונות הקלט/פלט של מקודד-מפענח

המקודד מעבד את הקלט בכניסתו באופן מקבילי ומייצר את הפלט בפעולה יחידה (single forward pass) ללא צורך בהפעלה איטרטיבית. בשונה ממנו, המפענח מייצר את הפלט שלו באופן אוטורגרסיבי. המשמעות של אוטורגרסיביות הינה יצירת פלט בהסתמך **רק** על יחידות הפלט שכבר נוצרו. כלומר, בכל איטרציה אנו מזינים לתוך הרשת יחידות מידע נוספות שחושבו באיטרציות קודמות כקלט, או איברי סדרת היעד עצמה עד הטוקן הנחזה (teacher forcing).

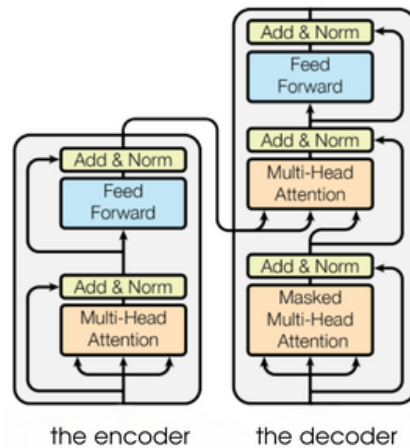
אז מדוע אנו בונים את הקלט של המפענח באופן אוטורגרסיבי? הסיבה לכך נובעת מכך שהמפענח הוא החלק במערכת המפיק את משפט היעד, ומטרתו לחקות את תהליך יצירת המילה האנושי.

אולי ישנה דרך אחרת לבצע זאת. נבחן את האפשרויות העומדות לפנינו:



- **הראשונה** היא לא להשתמש במיסוך כלל. כלומר, נזין את כל סדרת היעד למפענח, ואת כל סדרת המקור למקודד. לאחר מכן נבצע הרצה בודדת (single forward pass) למפענח ונשווה את פלט שהתקבל עם סדרת היעד. אפשרות זו מקבילה ללמידת פונקציית הזהות, אנחנו מקבלים את התוצאה הרצויה (הרשת מוציאה פלט נכון) אבל הרשת לא לומדת דבר מכיוון שהיא תלמד להעתיק את הקלט ללא שינוי.
- **השנייה** היא להזין למקודד את סדרת המקור, ואילו למפענח לא להזין את סדרת היעד כלל, ואת פלט המפענח להשוות מול כל סדרת היעד כמקשה אחת. פתרון כזה לא יעבוד גם כן, מאחר ולשפה טבעית מבנה מורכב, והמפענח יתקשה להסתמך רק על הקלט מהמקודד, אלא יצטרך להיות מודע לחלקים הקודמים בסדרה שכבר ייצר (או לקבלם כקלט).
- **האפשרות האחרונה ההגיונית ביותר** הינה לבנות את הקלט באופן אוטורגרסיבי. כלומר, בכל איטרציה אנו חוזים אך ורק את הטוקן הבא (שטרם נחזה) ומוסיפים אותו לקלט המפענח המשמש לחיזוי הטוקן הבא. לפעמים לאחר סיום חיזוי מילה (כלומר כאשר נחזה הטוקן "[SEP]" המסמן מרווח בין מילים) המילה שנחזתה מוחלפת במילה מתאימה מסדרת היעד ([teacher forcing](#)). תהליך זה מחקה את אופן יצירת מילה חדשה על ידי בני אדם. אנו יכולים לחשוב על מילה חדשה רק בהינתן מילים שנאמרו עד כה מכיוון שהמילה החדשה תלויה בהם.

קופסא שלישית: צוללים לתוך המקודד ומפענח



איור 7 - המבנה הפנימי של המקודד והמפענח

אבני הבניין המרכזיות של המקודד והמפענח.

המקודד והמפענח בנויים משלושה אבני בניין מרכזיות שבאמצעותן הארכיטקטורה מקבלת את עוצמתה הייחודית.

- **מנגנון תשומת הלב** - מנגנון תשומת הלב שהוצע לראשונה ב- [4] הכיל חישוב של וקטור דינמי המכיל את הקשרים החשובים ביותר בין המקודד למפענח בעת יצירתו של טוקני הפלט בעת יצירתו של טוקני הפלט (דומה לתשומת הלב המוצלבת בטרנספורמרים). [5] השתמש במנגנון תשומת הלב כדי לחשב את הקשרים בין חלקי הקלט השונים במקודד (דומה לתשומת הלב העצמית בטרנספורמרים).
- רעיון דומה יושם גם כן במנגנון תשומת הלב של טרנספורמרים שממשקל את עוצמת הקשר בין ייצוגי יחידות דאטה בתוך אותה הסדרה (תשומת לב עצמית), או את הקשר בין ייצוגי יחידות דאטה הנבנים על ידי המקודד למפענח (תשומת לב מוצלבת). אולם ההבדל העיקרי בין המנגנונים הוא שבטרנספורמרים תשומת הלב (העצמית) מחושבת במקביל עבור כל הטוקנים בתוך הקלט, ולכן אין צורך בזיכרון. החידוש הנוסף הוא שימוש בשני המנגנונים אלו (עצמית ומוצלבת) יחד.
- **רשת feed-forward** - שתי שכבות fully-connected (השכבה השנייה הינה לינארית ללא פונקציית אקטיבציה). לכאורה אין טעם לדון בה, אולם יש לרשת זו תפקיד חשוב בבנייתו של תוצר מנגנון תשומת הלב. כאשר נדבר עליה נציג מאמרים שמראים כיצד השמטת חלק זה מובילה לירידה משמעותית בביצועי המודל.
- **שכבת Add and Norm** - שכבה שתפקידה לבצע residual connection ונרמול (layer normalization). מוצא פונקציית תשומת הלב היא מטריצה בגודל $N \times d_{model}$ כאשר N מייצג את מספר הטוקנים. שכבת זו מנרמלת כל מימד i של הקלט (אנחנו מנרמלים את מימד i של כל וקטורי ייצוג של כל הטוקנים יחד - כלומר וקטור בגודל N "מנורמל" כל פעם).

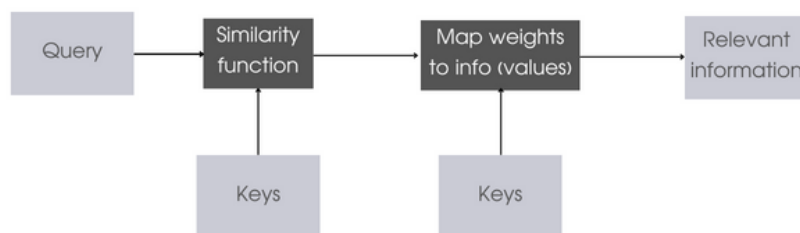
מנגנון תשומת הלב

ארכיטקטורת הטרנספורמרים שינתה את האופן בו אנו מסתכלים עיבוד קלט סדרתי. עד לאותו הרגע, מנגנוני תשומת הלב השתמשו במידע המתקבל מהמצבים הפנימיים של רשתות איטרטיביות, שאילצו הזנת קלט בודד (מילה או חלק מילה) בכל הפעלה. תשומת הלב חושבה באמצעות בניית וקטור זיכרון דינמי, (שדרש הקצאת זיכרון) על מנת לייצג את הקשרים במידע בכל איטרציה. שתי תכונות אלו היוו את עקב אכילס של מודלים אלו. אולם, עם הופעת הטרנספורמרים, התאפשר בפעם הראשונה עיבוד מקבילי של הקלט כמקשה אחת. כתוצאה מכך, לא היה צורך יותר בוקטור זיכרון עבור כל איטרציה, מה שאפשר בפעם הראשונה מידול איכותי של תלויות ארוכות וקצרות טווח בקלט. חשוב לזכור שמנגנון תשומת הלב אינו חוסך במשאבי חישוב לעומת ארכיטקטורת שקדמו לו, סיבוכיות הריצה וסיבוכיות המקום (time and space complexity) של מנגנון תשומת הלב עבור הרצה בודדת הינם $O(n^2 * d_{model})$, אולם הוא אופן ניסוחו הינו יעיל יתר מבחינה רעיונית כי הוא בנוי אינהרנטית לעיבוד מקבילי של כל המידע.

על מנת לבצע זאת הוגדרו 3 אובייקטים הנקראים שאילתה, מפתח וערך שהרעיון מאחוריהם הושאל מאלגוריתמי אחזור נתונים (information retrieval) ומנועי חיפוש:

- **שאילתה (query)** היא הבקשה שאנו משתמשים בה על מנת לקבל את הערך/ים.
- **מפתח (key)** זוהי האינפורמציה (ייצוג) המקושרת אל הערך, ומשמשת לזיהוי שלו.
- **ערך (value)** הוא הפריט אותו אנו מחפשים, השמור במסד הנתונים.

דוגמא הממחישה את הרעיון של שימוש בשאילתה, מפתח וערך, הינה חיפוש סרטונים ביוטיוב (YouTube). השאילתה מייצגת את הטקסט שאנו כותבים בשורת החיפוש, המפתח מייצג את המידע על הסרטון השמור במסד הנתונים המאחסן את הסרטונים, (שם מלא של הסרטון, תיאור שלו, אורך, יוצרים וכו.) והערך הוא הסרטון בו אנו מעוניינים לצפות. אם נקביל את דוגמת חיפוש סרטונים להפעלת מנגנון תשומת הלב של טרנספורמרים, הסרטון שיבחר יהיה זה שהמפתח שלו יהיה בעל הקורלציה הגבוהה ביותר לשאילתה שלנו.



איור 8 - המחשת קבלת ערך לפי שאילתה וציון

דרך נוספת לחשוב על אובייקטים אלו היא באמצעות גרף לא מכוון שלם וממושקל. כלומר גרף שבו כל קודקוד מחובר עם כל קודקוד אחר (כולל עצמו) ולקשת בינם יש משקל מוגדר. בכל פעם אנו בוחרים קודקוד אחד ומקבעים אותו בתור השאילתה, כל קודקוד בגרף (כולל הקודקוד שקבענו) מוגדר כמפתח. הערך מוגדר כמשקל הקשת המחברת בין קודקוד השאילתה לקודקוד המפתח. תוצאת פונקציית תשומת הלב תהייה מכפלה פנימית של ייצוג וקטורי של שני הקודקודים במשקל הקשת וביצוע softmax כדי לנרמל את המשקל של קודקוד.

מדוע אנחנו משתמשים במפתח, ערך ושאלתה?

הסיבה שאנו משתמשים במפתח, ערך, ושאלתה במנגנוני תשומת הלב נובעת מיכולותם בייצוג קשרים בין חלקים שונים של הדאטה הסדרתי. בניגוד למאמרו הקודם, שבו מנגנון תשומת הלב עשה שימוש במצבים הפנימיים של המקודד והמפענח, ארכיטקטורת הטרנספורמרים לא מכילה מידע פנימי שמה ולכן אנו זקוקים לדרך בה אנו יכולים לייצג קשרים אלו. מכיוון שמנגנון תשומת הלב של טרנספורמרים מתפקד כמנגנון אוניברסלי גם עבור תשומת הלב העצמית ותשומת הלב המוצלבת, עלינו להגדיר כלים מופשטים שיכולים לייצג את הרעיון עצמו.

הדוגמא שבה הצגנו את מנגנון תשומת הלב באמצעות גרף שלם ממושקל, מייצגת בדיוק את הקשרים הללו. בדוגמא זו, לא משנה לנו מהיכן מגיעים השאלות או המפתחות, אנחנו עדיין יכולים ליצור את הגרף השלם המחבר בינם. הייצוג של מנגנון תשומת הלב בצורה שכזו מאפשר לנו להשתמש בו גם בתשומת הלב המוצלבת בה המפתחות והערכים מגיעים מהמקודד בעוד שהשאלות מגיעות מהמפענח.

אז כיצד משתמשים במנגנון זה בארכיטקטורה?

כפי שניתן לראות באיור 7, גם המקודד וגם המפענח משתמשים תחילה במנגנון תשומת הלב עצמית, שבה אנו מאתרים את הקשרים בתוך הקלט. מטרת מנגנון תשומת הלב העצמית הינה יצירת ייצוג תלוי הקשר עבור כל טוקן. לאחר מכן, המידע שחושב באמצעות מנגנון תשומת הלב העצמית של המקודד והמפענח משולב באמצעות תשומת לב מוצלבת. בשלב זה אנו מגדירים את השאלות להיות מוצא פונקציית תשומת הלב של המפענח, ואת המפתחות והערכים אנו מגדירים להיות מוצא בלוק המקודד האחרון.

הקלט למנגנון תשומת הלב של המקודד הינו הפלט של בלוק הטרנספורמרים ששורשר לפניו (כאשר מדובר בבלוק הטרנספורמרים הראשון הקלט הוא סדרת הקלט שעברה טוקניזציה והוספת קידוד תלוי מיקום). לפני שקלט כלשהו מוכנס למנגנון תשומת הלב הוא עובר הטלה למרחב חדש באמצעות שלושה שכבות לינאריות (בפשוטות, הכפלות במטריצה). במודל המוצע במאמר הטלות אלו הן שיוצרות את המפתח, שאלתה והערכים (אולם אין זה מחייב, ניתן להגדיר את המפתח השאלתה והערך כרצוננו). זוהי נקודה חשובה, שכן **אם נשתמש בקלט בצורתו המקורית ללא הטלה זו, הרשת תאבד את הגמישות**

(flexibility) שלה ללמוד קשרים אלו.

אבל מדוע זה נכון? אם נחזור לדוגמא של הגרף הממושקל, נראה שלא רק משקל הקשת הוא זה שקובע את עוצמת הקשר, אלא גם תכונות (ייצוג) הקודקודים. הדרך להקנות ייצוג וקטורי לקודקודים היא באמצעות הטלה זו. במילים אחרות, המודל לומד לשייך משקל שונה למילים שונות כתלות במשמעות הסמנטית והתחבירית שלהן, ומתוך כך משערך את עוצמת הקשר באופן מדויק. נקודה זו מאפשרת להבין מדוע הטרנספורמרים מסוגלים לפתור בעיות שרשתות שקדמו להן לא הצליחו.

נקודת מבט נוספת על העניין הינה ההבדל בין ארכיטקטורת הטרנספורמרים למודלים שקדמו להן, באופן שבו המודל לומד את התלות בין מרכיבי הקלט. במאמר [4] עוצמת הקשר בין מרכיבי הקלט למנגנון חושבה באופן הבא:

$$e_{ij} = \text{attention}(s_{i-1}, h_j) = v_a^T * \tanh(W * [s_{i-1}; h_j]), j = 1, \dots, T$$

משוואה 6 - תשומת הלב כפי שהוצגה במאמר [4]

כאשר s_{i-1} ו h_j הם המצבים הפנימיים מהמפענח והמקודד בהתאמה (להרחבת הקריאה, ניתן למצוא את הפירוט המלא במאמר הקודם שלנו על מנגנון תשומת הלב).



זהו ההבדל העיקרי בין שני מנגנוני תשומת הלב. במאמר [4] מנגנון תשומת הלב הוא זה שלומד למשקל את חשיבות קלט מסוים באמצעות המכפלה במטריצה W , דבר שגורם לחוסר יכולת לייצג קשרים מורכבים בדאטה. לעומת זאת, בארכיטקטורת הטרנספורמים אנו כבר לא זקוקים לסט משקולות בודד (המכיל W ו- v_a) שילמד את הקשרים בין מרכיבי המידע השונים בתוך המנגנון, אלא דורשים זאת מהרשת בשלב מוקדם יותר באמצעות ההטלה עליה דיברנו. פעולה זו מצמצמת את מנגנון תשומת הלב בטרנספורמרים להיות חישוב קורלציה בין משתנים שונים אולם היא הופכת את המודל להיות עוצמתי בהרבה.

לפעולת ההטלה ישנן מספר משמעויות חשובות:

- היא מאפשרת למודל ללמוד ייצוג יעיל של המידע בהתאם למשימה הנדרשת. כל אחד מהאובייקטים הללו מייצג התבוננות שונה על הקלט, דבר המאפשר למודל ללמוד אספקטים שונים עליו.
- מכיוון שהייצוג מתחשב בקשרים בתוך הדאטה, הוא מאפשר למנגנון תשומת הלב להתאים דרגות חשיבות משתנות לחלקים שונים בקלט בהתאם לתוכן שלהם.

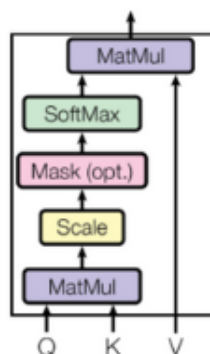
הפלט של מנגנון תשומת הלב מוגדר כמטריצה בגודל $N \times d_{model}$, המהווה את קבוצת הוקטורים המייצגים ייצוגים תלויי הקשר של כל הטוקנים בסדרה.

המידול המתמטי של מנגנון תשומת הלב

- כאמור הקלט שנשמנו כעת ב- X הוא מטריצה בגודל $N \times d_{model}$ כאשר N מייצג את מספר הטוקנים המתקבלים בכניסה של פונקציית תשומת הלב.
- 3 המטריצות המשמשות להעברת הקלט לשלושת האובייקטים החדשים מסומנות בתור W_Q, W_V, W_K (שאינן, מפתח וערך בהתאמה) והן כאמור מעבירות את הקלט לייצוג החדש שלו. בפרק הבא נראה כי מטריצות אלו משמשות אותנו להטלת הקלט ל h מרחבים שונים כל אחד בגודל d_{model}/h עבור מנגנון תשומת הלב הרב-ראשית.

$$Attention(Q, K, V) = softmax(QK^T / \sqrt{d_k})V$$

משוואה 7 - תשומת הלב של טרנספורמרים



איור 9 - תרשים הזרימה של מנגנון תשומת הלב העצמית

נפרק את המשוואה לגורמים:

- מטריצות W_K, W_V, W_Q הן בעלות מימד $d_{model} \times d_k$ כל אחת (המאמר המקורי מגדיר אותן באותו הגודל):

$$K = X * W_K, Q = X * W_Q, V = X * W_V$$

- X הוא מטריצה בגודל $N \times d_{model}$ המייצגת את הטוקנים לאחר שיבוץ וקידוד תלוי מיקום. גודל כל אחת מהמטריצות הינה $d_{model} \times d_k$, כאשר בחלק זה נגדיר $d_{model} = d_k$ כמו במאמר המקורי.
- $Q * K^T$ - תפקידה לחשב את עוצמת התאימות (compatibility) שבין כל השאילות לבין כל המפתחות האפשריים. התוצאה הינה מטריצה בגודל $N \times N$.
- $QK^T / \sqrt{d_k}$ - מטרת החלוקה היא למנוע מפונקציית ה-softmax להגיע לרוויה, דבר שמוביל לדעיכת גרדיאנטים ובכך עלולה לעכב את הלמידה. על מנת להבין מדוע הקלט של פונקציית ה-softmax יכול לגדול/לקטון מעבר לרצוי, ניתן להרחיב את הקריאה בלינק [הבא](#).
- $softmax(QK^T / \sqrt{d_k})$ - כפי שראינו בארכיטקטורות קודמות, פונקציית ה-softmax מאפשרת לקבל פונקציית תשומת לב רציפה, הממשקלת את עוצמות התאימות בין שאילותה נתונה למפתח ביחס לכל צמדי שאילותה-מפתח אחרים.
- $softmax(QK^T / \sqrt{d_k}) * V$ - מכיוון שהפעלת פונקציית ה-softmax מחזירה את המשקל של צמדי שאילותה-מפתח, המכפלה במטריצת הערכים משרתת את אותה המטרה כמו בארכיטקטורות קודמות, ומאפשרת לבנות את ייצוג הקלט באופן רציף.

תשומת לב ממוסכת

כפי שהזכרנו בפרקים הקודמים, אנו ממסכים חלק מהמידע המתקבל בכניסת המפענח על מנת שלא יוכל לראות חלקים עתידיים של הטקסט הנבנה ובכך לפגום בתהליך הלמידה. על מנת לעשות זאת אנו מוסיפים מיסוך במנגנון תשומת הלב. לכן, משוואה 6 משתנה למשוואה 7 כאשר M מכילה $-\infty$ (בפועל מדובר במספר שלילי מאוד גדול) במקומות בהם לא נרצה לתת גישה למפענח, ואפס (= אין השפעה) במקומות שכן נרצה לתת גישה. ערך זה (מינוס אינסוף) מתרגם בפונקציית ה-softmax לאפס.

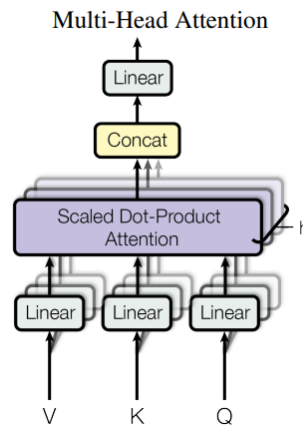
$$Attention(Q, K, V, M) = softmax((QK^T + M) / \sqrt{d_k})V$$

משוואה 8 - תשומת הלב עם מיסוך

מנגנון תשומת לב הרב-ראשית

ארכיטקטורת הטרנספורמרים מציגה הרחבה למנגנון תשומת הלב בדמות מנגנון תשומת לב רב ראשית (multi head attention). אופן פעולת המנגנון הינה הטלת הטוקנים המשובצים ל- h מרחבים שונים, כולם בגודל $\mathbb{R}^{N \times d_k}$, במקום להטיל אותם למרחב אחד בעל מימד d_{model} כאשר $d_k = d_{model} / h$. במילים פשוטות, ישנם h שלישיות שונות של מטריצות שאילותה מפתח וערך. כל שלישיה שכזו, הנלמדת באופן עצמאי, מייצגת קשרים שונים בקלט, ומעובדת על ידי מנגנון תשומת לב עצמאי כפי שמתואר בפסקה הקודמת. בסוף החישוב מוצאי כל הראשים משורשרים אחד לשני, ומוטלים למישור $\mathbb{R}^{N \times d_{model}}$ באמצעות מטריצה W^O .





איור 10 - מנגנון תשומת הלב הרב ראשית

משוואה 9 מציגה את המידול המתמטי של הפעולה שתיארנו כעת:

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h) * W^O$$

$$where head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

משוואה 9 - מנגנון תשומת הלב הרב-ראשית

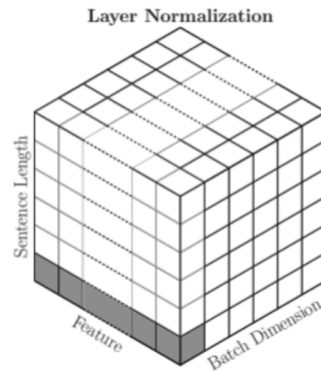
אז מדוע אנו זקוקים לתשומת לב רב ראשית?

החוקרים במאמר המקורי טוענים כי השימוש במנגנון תשומת הלב הרב-ראשית מאפשר למודל ללמוד הקשרים שונים הקיימים בטקסט המתבטאים כאינטראקציות שונות בין המילים. אבל מדוע זה מתאפשר למעשה? הסיבה לכך נובעת מהעובדה שאנו לא רק מטילים את המידע לראשים שונים אלא גם מפעילים מנגנון תשומת הלב עליהם. דבר זה מאפשר לכל ראש להתמקד בתכונות שונות של הקלט וכתוצאה מכך ניתן למדל תלויות שונות בתוכו.

מאמר [6] שבחן את תפקיד הראשים השונים במנגנון, הגיע למסקנה כי ראשים שונים מבצעים תפקידים שונים. לדוגמא, תפקיד אחד הראשים היה לגשת למילים במיקומים קרובים, תפקיד של ראש אחר היה לבצע זיהוי ומידול של מילים שהופיעו בתדירות מאוד נמוכה, ותפקיד ראש אחר הינו למדל מילים בעלות קשר ספציפי. המאמר ביצע אנליזה זו על ידי גיזום (pruning) של ראשי תשומת הלב במהלך השימוש ברשת (inference).

שכבת הנרמול (Layer Normalization)

כפי שציינו בפתיח, כותבי המאמר המקורי בחרו להשתמש ב-LN (layer normalization). כותבי המאמר לא מציינים מדוע הם בחרו להשתמש דווקא בסוג נרמול זה על פני שיטות נרמול אחרות, אולם כפי שנראה בהמשך, נרמול על פני כל השכבה, מונע בעיות הצצות במהלך השימוש בנרמול על פני קבוצה (batch).



איור 11 - layer normalization של שכבת הפיצ'רים

מאמר [7] (שמציע בין היתר שיטת נרמול חדשה עבור יישומי שפה בטרנספורמרים) מנתח את המאפיינים הסטטיסטיים של mini-batch (באץ') במשימות של ניתוח שפה טבעית, ומציג פרשנות משלו מדוע אנו משתמשים דווקא ב-LN במקום שיטת הנרמול BN (batch normalization). כותבי המאמר מציינים כי הסיבה המרכזית לכך ש-BN לא מתאימה למשימת ניתוח שפה נובעת מכך שסטטיסטיקת הבאצ'ים של הטקסט, אינה אחידה. כלומר ערך הטוקנים הממוצע והשונות שלהם משתנים באופן בלתי צפוי, באץ' אחד לאחר, דבר הבא לידי ביטוי בשונות של הגרדיאנטים, והניסיון להשתמש בממוצע ושונות נלמדים של באץ' פוגע בביצועים.

שכבות ה- feed forward - skip connection :

הנושא האחרון שנדון בו הינו שכבת ה-feed forward ושכבות ה-skip connection. שכבות אלו זכו לפופולריות רבה בלמידה עמוקה והן מככבות במגוון מודלים בדומיינים שונים עבור משימות רבות. מכיוון שאלו אבני בניין בסיסיות, עלולה להתעורר השאלה מדוע אנו מקדישים פרק שלם רק עבורן. הסיבה לכך טמונה בעובדה ששני שכבות אלו חיוניות על מנת למנוע מהמודל להתכנס ייצוגי פלט מנוונים במהלך האימון.

המאמר המקורי אינו מספק את הסיבות לשימוש בשכבות אלו, אולם, המאמרים המאוחרים יותר [8, 9] ביצעו ניתוחים מעמיקים של המודל והגיעו למסקנות הבאות:

ללא שכבת FF - skip connections הפלט של מנגנון תשומת הלב נוטה להתכנס לשיבוץ טוקנים בעלי תלות לינארית. כלומר, הם ניתנים לייצוג כ- $k_i v$ עבור וקטור v קבוע וסקאלרים $k_i, i = 1, \dots, N_{token}$. תופעה זו מחמירה כאשר אנו עורמים (stack) את בלוקי הטרנספורמרים. הבעיה בתופעה זו הינה שהתלות הלינארית של ייצוגי הטוקנים אינה מייצגת את הקשרים החבויים בטקסט ואת הסמנטיקה שלו. הסיבות לתופעה זו טרם התבררו, אך ידוע כי הן נובעות ממבנה מנגנון תשומת הלב והתפלגות הדאטה.

המחברים של [8] מציינים כי הוספת שכבות skip connections ו-FF מאפשרת להתגבר על סוגיה זו. [8] מראה כי שכבת ה-skip connection הינה הגורם הקריטי למניעת התלות הלינארית בין שיבוצי הטוקנים. בנוסף שכבת ה-FF מאטה את קצב "היוניפורמיזציה" (uniformization) של ייצוגי הטוקנים במוצא מנגנון תשומת הלב אך תפקידה אינה בולט כמו שכבת ה-skip-connections.

נציין כי FF בטרנספורמרים בנוי משכבה לינארית עם אקטיבצית ReLU ושכבה לינארית נוספת (נתונה על ידי משוואה 9):

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

משוואה 10 - שכבת ה-feed forward

רוצים לדבר איתנו? -> לתגובות, חידודים, הצעות לשיתופי פעולה ושאר ירקות אתם מוזמנים לבוא ללינקדין שלנו: [מיכאל חי](#), [מיכאל](#)

[חי הופמן](#) הוא מסטרנט בשלבי סיום באוניברסיטת אריאל. חוקר בתחום הראייה הממוחשבת והלמידה העמוקה בדגש על אלגוריתמים גנרטיביים. בנוסף כותב תוכן בעברית בתחומים אלו.

[מיכאל \(מייק\) ארליכסון](#), [PhD](#), Michael Erlihson. עובד בחברת הסייבר [Salt Security](#) בתור Principal Data Scientist. מיכאל חוקר ופועל בתחום הלמידה העמוקה, ולצד זאת מרצה ומנגיש את החומרים המדעיים לקהל הרחב

