

ארכיטקטורת הטרנספורמרים (Transformers)

אז מהו הסוד של מנגנון תשומת הלב בטרנספורמרים?

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 הטרנספורמים?
- מהם תפקידם 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').

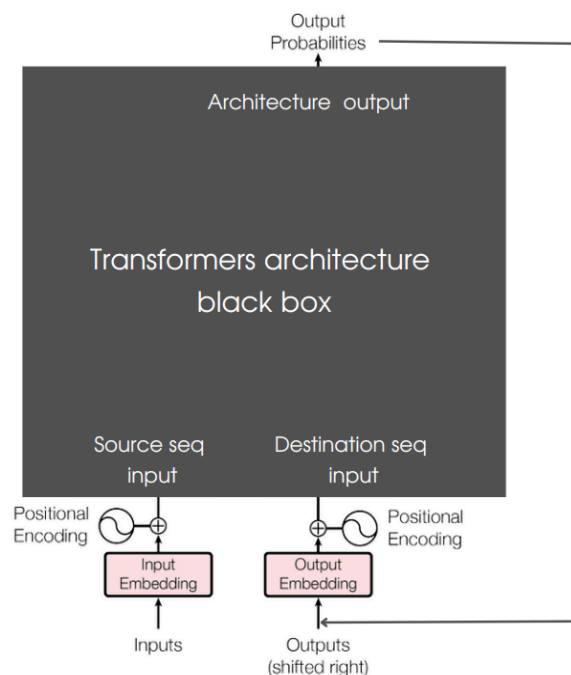
- **קופסא שלישית:** בחלק זה נספק הסבר מקיף על הבלוקים המרכיבים את המקודד והמפענח. נרחיב על מבנה הארכיטקטורה של כל אחד מחלקים אלו ונפתח למעשה את כל הקופסאות השחורות שנותרו לנו מהשלב האחרון. נציג לראשונה את הרעיון המרכזי שהרשת בנויה סביבו,

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', '#t'] (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 ([BytePair Encoding](#) ו-[WordPiece](#)) 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 (Positional Encoding - Input Embedding). **שיבוץ הטוקנים (Tokens Embedding):**

מוצא שכבת השיבוץ נתון על ידי המשוואה הבאה:

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

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 $V \times d_{model}$, 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 $\sqrt{d_{model}}$. אולם, ישנן השערות שהדבר אמור למנוע את דעיכת הגרדיאנטים שכן השיבוץ עלול ליצור ערכים גבוהים מדי שייבילו 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

שמשמששים 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), 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:

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

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 למדוד את חשיבותו ביחס לטוקן במיקום ה-47 אזי המרחק היחסי בינם 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:

$$\begin{aligned} (1) \sin(pos + k) &= \sin(pos) * \cos(k) + \sin(k) * \cos(pos) \\ (2) \cos(pos + k) &= \cos(pos) * \cos(k) - \sin(pos) * \sin(k) \end{aligned}$$

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:

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

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.
 - $2i/d_{\text{model}}$ - מכיוון שוקטור הקידוד מתווסף לוקטור השיבוע, הוא מכיל d_{model} מימדים, ולכן יש צורך לקבוע מה יהיה הערך שיקבל כל מימד.
- הפתרון הנאיבי** הוא להשתמש בערכו של $\sin(\text{pos})$ או $\cos(\text{pos})$ 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:

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

- 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.

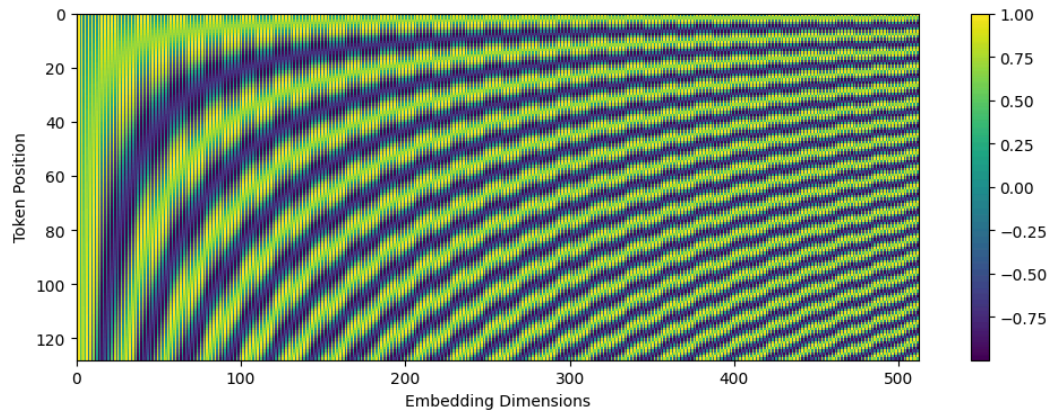
למשל עבור קידוד תלוי מיקום השווה ל:

$$\text{Positional Encoding}(\text{pos}, 2i) = \sin(\text{pos} * (2i/d_{\text{model}}))$$

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

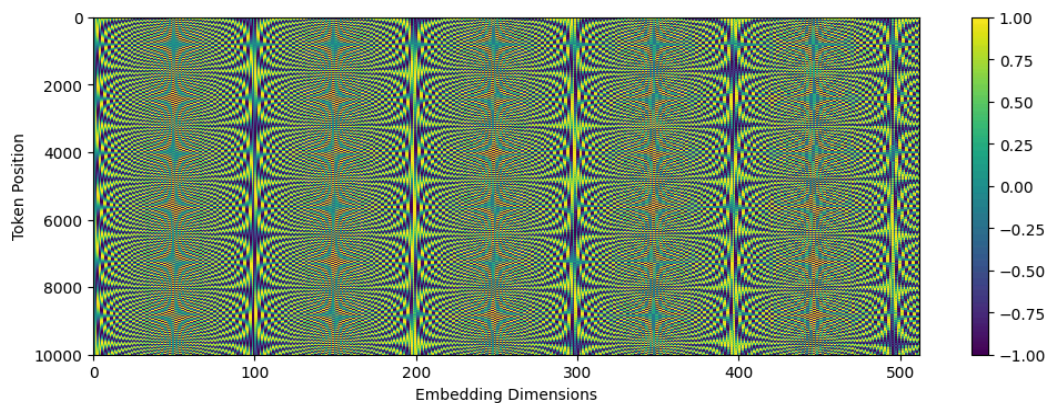
משוואה 5 - קידוד תלוי מיקום ללא המרכיב המעריכי

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



איור 2 - קידוד תלוי מיקום אלטרנטיבי, עבור 128 טוקנים

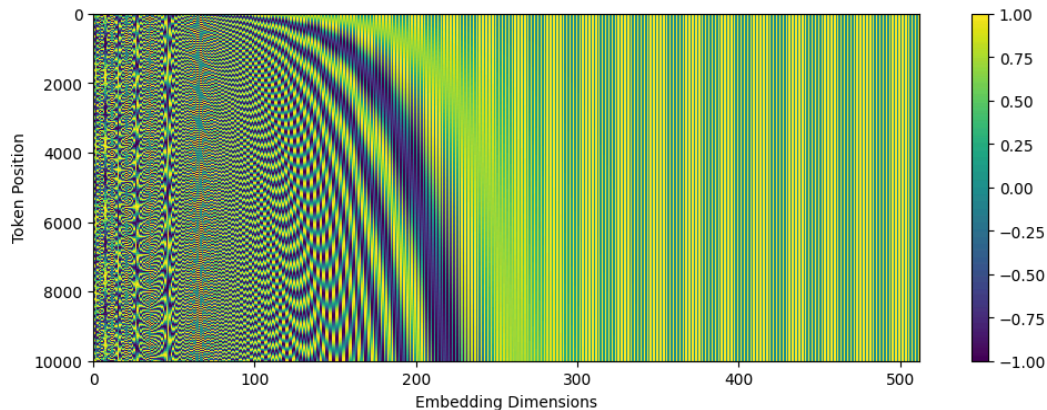
לכאורה לא נראית בעיה גלויה לעין. אולם, אם נגדיל את מספר הטוקנים ל-10,000 כאשר $d_{model} = 512$ (אורך הקשר ריאלי עבור טרנספורמרים בני ימינו) נקבל את הגרף הבא:



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

כפי שניתן לראות, כאשר מספר הטוקנים בסדרה, גדול מ- d_{model} מתחילות להיווצר 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

להרחבת הקריאה על הניסויים שביצענו על מנת להגיע למסקנות אלו ניתן להיכנס למחברת זו.

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.

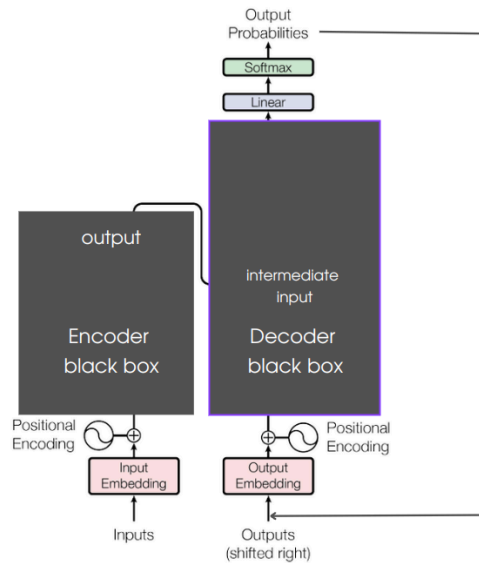
חלק שני: פלט הרשת

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)

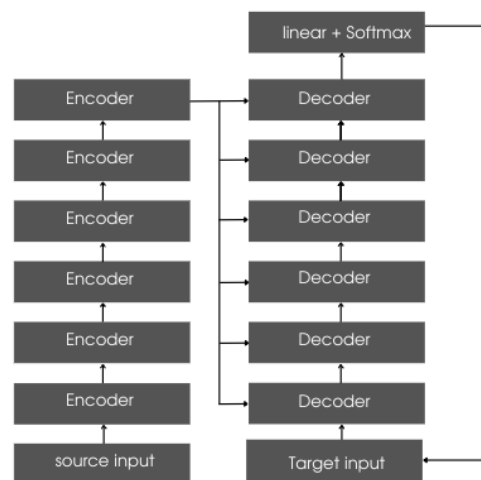
וביצירת הפלט שלו. אם נחזור לדוגמא שפתחנו איתה, המקודד מעביר למפענח את **תוכן הרעיון** שהמשפטים מייצגים, ללא תלות במרחבי הקלט (השפות השונות).

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).

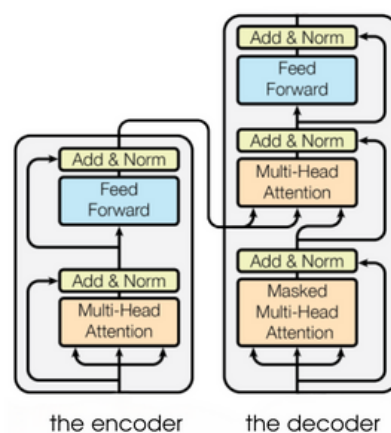
אז מדוע אנו בונים את הקלט של המפענח באופן אוטורגרסיבי? הסיבה לכך נובעת מכך שהמפענח הוא החלק במערכת המפיק את משפט היעד, ומטרתו לחקות את תהליך יצירת המילה האנושי.

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 עלייה 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).

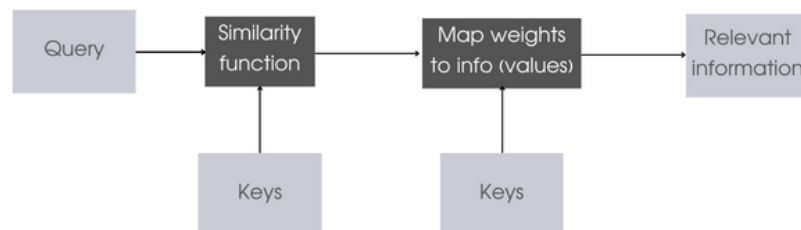
מנגנון תשומת הלב

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 השאלית, כל קודקוד בגרף (כולל הקודקוד שקבענו) מוגדר כמפתח. הערך מוגדר כמשקל הקשת המחברת בין קודקוד השאלית 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 ושאלית?

הסיבה שאנו משתמשים במפתח, ערך, ושאלית במנגנוני תשומת הלב נובעת מיעילותם 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 השאליות 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 שהשאליות 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 השאליות 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, שאלית והערכים (אולם אין זה מחייב, ניתן להגדיר את המפתח השאלית 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 מהמהפענח 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.

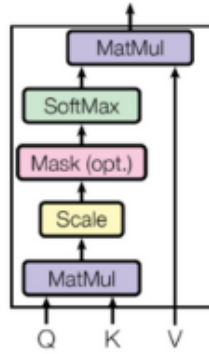
הפלט של מנגנון תשומת הלב מוגדר כמטריצה בגודל $N \times d_{model}$, המהווה את קבוצת הוקטורים המייצגים ייצוגים תלויי הקשר של כל הטוקנים בסדרה.

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_K, W_V, W_Q (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.

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

משוואה 7 - תשומת הלב של טרנספורמרים



איור 9 - תרשים הזרימה של מנגנון תשומת הלב העצמית

נפרק את המשוואה לגורמים:

- מטריצות W_Q, W_V, W_K 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

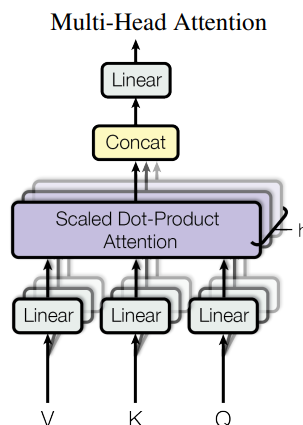
כפי שהזכרנו בפרקים הקודמים, אנו ממסכים חלק מהמידע המתקבל בכניסת המפענח על מנת שלא יוכל לראות חלקים עתידיים של הטקסט הנבנה ובכך לפגום בתהליך הלמידה. על מנת לעשות זאת אנו מוסיפים מיסוך במנגנון תשומת הלב. לכן, משוואה 6 משתנה למשוואה 7 כאשר M מכילה $-\infty$ (בפועל מדובר במספר שלילי מאוד גדול) במקומות בהם 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 שאילתה 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 8 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

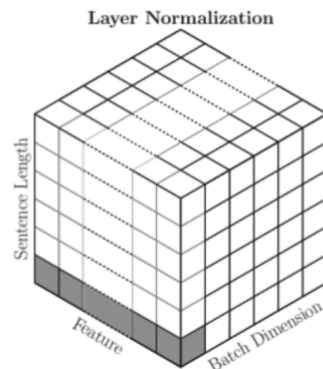
אז מדוע אנו זקוקים לתשומת לב רב ראשית?

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. לדוגמה, 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 הפיצורים

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 ו- 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 ו-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 הטרנספורמים. 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