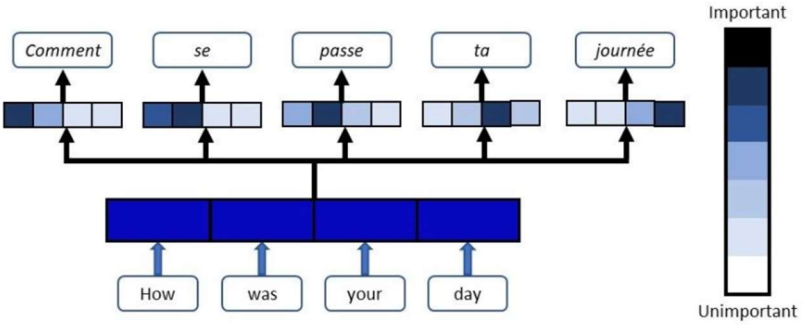
8. Attention Mechanism

8.1 Sequence to Sequence Learning and Attention

8.1.1 Attention in Seq2Seq Models

ניתוח סדרות בהן יש קשר בין האיברים יכול להיעשות בעזרת רשתות עם רכיבי זיכרון, כפי שהוסבר באריכות בפרק 6. ברשתות אלו הסדרה הנכנסת לרשת עוברת דרך encoder היוצר וקטור בגודל ידוע מראש המייצג את הסדרה המקורית תוך התחשבות בסדר של איברי הסדרה ובקשר ביניהם. לאחר מכן וקטור זה עובר ב-decoder שיכול לפענח את המידע שיש בוקטור ולהציג אותו בצורה אחרת. למשל בתרגום משפה לשפה – מודל של seq2seq מקודד משפט בשפה אחת לוקטור מסוים ולאחר מכן מפענח את הוקטור למשפט בשפה השנייה.

הדרך המקובלת ליצור את הוקטור ולפענח אותו הייתה שימוש בארכיטקטורות שונות של RNN, כמו למשל רשת עמוקה המכילה רכיבי זיכרון מסוג LSTM או GRU. מודלים אלו נתקלו בבעיה בסדרות ארוכות, כיוון שהווקטור מוגבל ביכולת שלו להכיל קשרים בין המון איברים. כדי להתמודד עם בעיה זו ניתן לנקוט בגישה שונה – במקום ליצור וקטור במוצא ה-encoder שלמעשה מפריד בין איברי הכניסה לאיברי המוצא, ניתן להשתמש במצבים החבויים של ה-encoder בשילוב המצבים החבויים של ה-decoder, וכך למצוא תלויות בין איברי הכניסה לאיברי המוצא (general attention) וקשרים בין איברי הכניסה עצמם (self-attention). ניקח לדוגמא תרגום של המשפט "How was your day" מאנגלית לשפה אחרת– מנגנון ה-attention מייצר וקטור חדש עבור כל מילה במוצא, כאשר הוקטור אומר עד כמה המילה הנוכחית במוצא קשורה לכל אחת מהמילים במשפט המקורי. באופן הזה כל איבר במוצא נותן תשומת לב שונה לכל אחד מאיברי הכניסה, ולכן המנגנון נקרא attention.



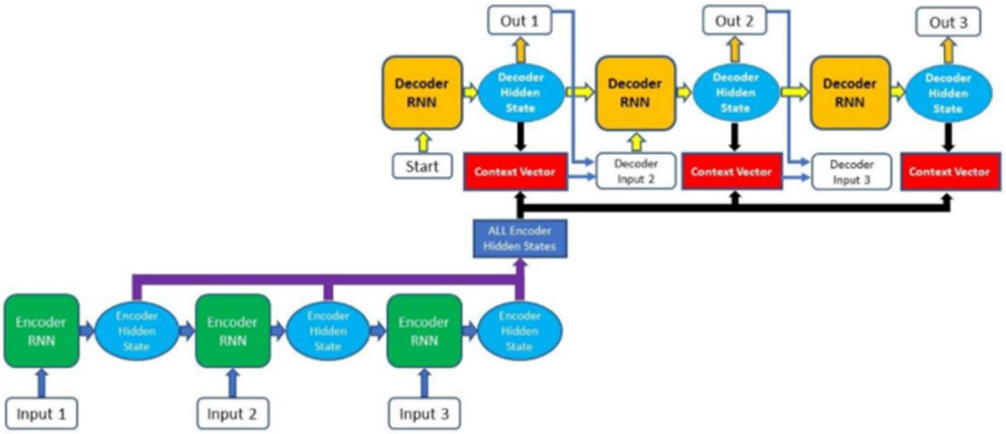
איור 8.1 מנגנון attention – נתינת משקל לכל אחת ממילות הקלט ביחס לכל אחת ממילות הפלט.

כאשר עלה הרעיון של attention שילבו אותו יחד עם רכיבי הזיכרון הקלאסיים (LSTM, GRU), אך לאחר מכן הייתה פריצת דרך במאמר שנקרא "Attention is All You Need", בו זנחו את רכיבי הזיכרון והשתמשו ב-attention בלבד.

בחלק זה יוצגו הגישות המשלבות בין רכיבי זיכרון לבין attention ולאחר מכן יוסבר על ה-transformer שמשתמש ב-self-attention וב-positional encoding בנוסף ל-attention הרגיל, ובכך מייתר את הצורך ברכיבי זיכרון.

8.1.2 Bahdanau Attention and Luong Attention

הגישה הראשונה שהוצעה נקראת Bahdanau Attention על שם הממציא שלה – Dzmitry Bahdanau.



איור 8.2 ארכיטקטורת Bahdanau Attention.

הרעיון של גישה זו היא לבנות ארכיטקטורה בה זוכרים את כל המצבים החבויים של רכיבי הזיכרון ב-encoder ומעבירים אותם ל-decoder. כתוצאה מכך ה-decoder מחשב את המוצא לא רק על סמך המצבים הקודמים, אלא משקלל את המצבים הקודמים יחד עם המצבים החבויים של ה-encoder. עבור כל אחד מאיברי המוצא מבצעים alignment score בין המצב החבוי של רכיב הזיכרון הקודם לבין כל המצבים החבויים של ה-encoder, וכך יוצרים context vector שבעזרתו מחשבים את הפלט של האיבר הנוכחי ב-decoder. ביצוע הפעולה הזו הוא הלב של מנגנון ה-attention, כיוון שהוא קושר בין הקלט לפלט ואומר בכל נקודה כמה משקל יש לתת לכל אחד מאיברי הקלט.

ביצוע פעולה זו יוצרת לכל אחד מאיברי הפלט context vector ייחודי משלו הנבנה גם מהמצב הקודם וגם מאיברי ה-encoder, בשונה מהארכיטקטורות הקודמות של seq2seq בהן לא היה ניתן להעביר מידע באופן ישיר מהמצבים החבויים של ה-encoder ל-decoder. את ה-context vector מחברים למוצא של האיבר הקודם ב-decoder, ויחד עם המצב החבוי הקודם יוצרים את המצב החבוי הבא, שבעזרתו מוצאים את הפלט של האיבר הנוכחי.

באופן פורמלי, אם נסמן ב-H_e, H_d את המצבים החבויים של ה-encoder וה-decoder ה-alignment score יתקבל על ידי:

$$\text{alignment score} = w_{alignment} \times \tanh\left(w_d H_d + w_e H_e\right)$$

כאשר $w_{alignment}, w_d, w_e$ הם המשקלים הנלמדים של ה-encoder, ה-decoder והחיבור ביניהם. את התוצאה מעבירים דרך SoftMax, מכפילים בH_e ומקבלים את ה-context vector:

$$\text{context vector} = H_e \times \text{SoftMax}(\text{alignment score})$$

הוקטור המתקבל אומר כמה משקל יש לתת לכל אחד מאיברי הקלט ביחס לאיבר הפלט הנוכחי. את התוצאה כאמור מחברים לפלט של האיבר הקודם, ובעזרת המצב החבוי הקודם מחשבים את המצב החבוי הנוכחי, שממנו מחלצים את הפלט של האיבר הנוכחי.

ישנו שיפור של מנגנון ה-attention שהוצע על ידי Bahdanau והוא נקרא Loung attention. שני הבדלים עיקריים יש בין שני המנגנונים: חישוב ה-alignment score מתבצע באופן שונה, ובנוסף בכל שלב לא משתמשים במצב החבוי הקודם של ה-decoder כמו שהוא אלא יוצרים מצב חבוי חדש ובעזרתו מחשבים את ה-alignment score.

8.2 Transformer

לאחר שמנגנון ה-attention התחיל לצבור תאוצה, הומצאה ארכיטקטורה המבוססת על attention בלבד ללא שום רכיבי זיכרון. ארכיטקטורה זו הנקראת transformer מציעה שני אלמנטים חדשים על מנת למצוא קשרים בין איברים בסדרה מסוימת – positional encoding ו-self-attention.

8.2.1 Positional Encoding

ארכיטקטורות מבוססות RNN משתמשות ברכיבי זיכרון בשביל לקחת בחשבון את הסדר של האיברים בסדרה. גישה אחרת לייצוג הסדר בין איברי הסדרה נקראת positional encoding, בה מוסיפים לכל אחד מאיברי הקלט פיסת מידע לגבי המיקום שלה בסדרה, והוספה זו כאמור באה כתחליף לרכיבי הזיכרון ברשתות RNN. באופן פורמלי, עבור סדרת קלט $x \in \mathbb{R}^d$, מחשבים וקטור במימד $d \times 1$ באופן הבא:

$$p_t(i) = \begin{cases} \sin(\omega_k, t), & i \text{ is even} \\ \cos(\omega_k t), & i \text{ is odd} \end{cases}, \omega_k = \frac{1}{10000^{\frac{2k}{d}}} \rightarrow p_t = \begin{bmatrix} \sin(\omega_1 t) \\ \cos(\omega_1 t) \\ \sin(\omega_2 t) \\ \cos(\omega_2 t) \\ \vdots \\ \sin\left(\omega_{\frac{d}{2}} t\right) \\ \cos\left(\omega_{\frac{d}{2}} t\right) \end{bmatrix}_{d \times 1}$$

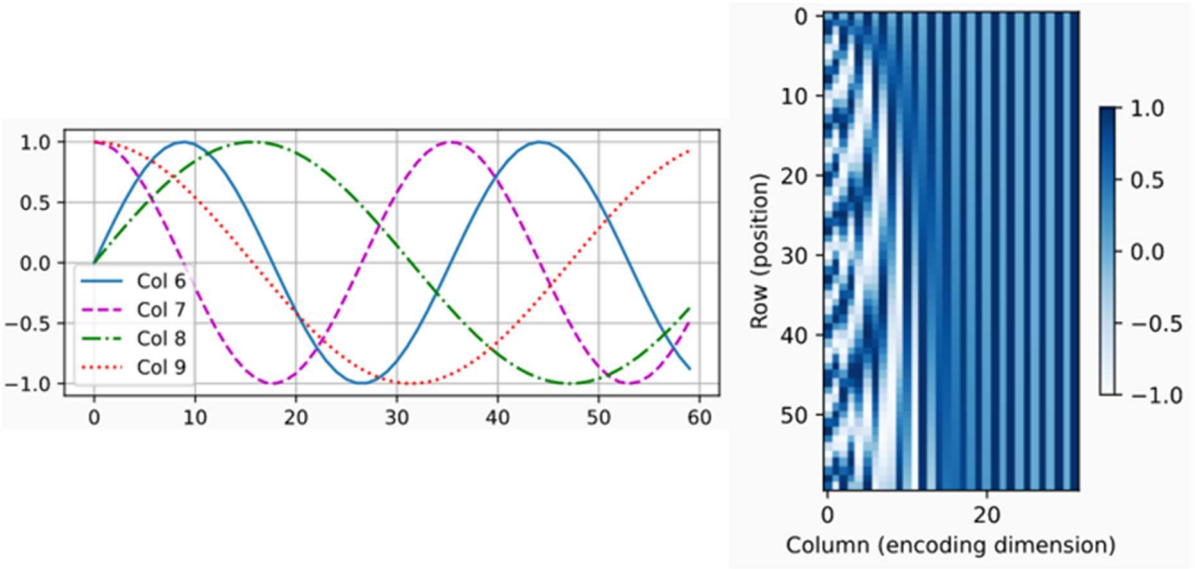
בכדי להבין כיצד וקטור זה מכיל משמעות של סדר בין דברים, נציג את הרעיון שהוא מייצג בצורה יותר פשוטה. אם נרצה לקחת רצף של מספרים ולייצג אותם בצורה בינארית, נוכל לראות שככל שלביט יש יותר משקל, כך הוא משתנה בתדירות נמוכה יותר, ולמעשה תדירות שינוי הביט היא אינדיקציה למיקום שלו.

:

0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

איור 8.3 ייצוג בינארי של מספרים. ה-MSB משתנה בתדירות הכי נמוכה, ואילו ה-LSB משתנה בתדירות הכי גבוהה.

כיוון שמתעסקים במספרים שאינם בהכרח שלמים, הייצוג הבינארי של מספרים שלמים הוא יחסית בזבזני, ולכן כדאי לקחת גרסה רציפה של אותו רעיון – פונקציות טריגונומטריות עם תדירות דועכת. זהו בעצם הוקטור p – הוא מכיל הרבה פונקציות טריגונומטריות בעלות תדירות הולכת וקטנה, ולפי התדירות שמתווספת לכל איבר בסדרה המקורית ניתן לקבל אינדיקציה על מיקומו.



איור 8.4 Positional encoding. דוגמא למספר פונקציות בעלות תדירות הולכת וקטנה, בהתאם לאיבר אותן הן מייצגות (שמאל). המחשה לקצב השינוי של כל פונקציה בהתאם למיקום של האיבר אותו היא מייצגת – מעין גרסה רציפה לקצב שינוי הביטים בייצוג בינארי של מספרים שלמים (ימין).

להוסיף Relative Positional Information

https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

8.2.2 Self-Attention Layer

בנוסף ל-positional encoding, עלה הרעיון לבצע attention לא רק בין איברי הקלט לאיברי הפלט, אלא גם בין איברי הקלט בעצמם. הרעיון הוא לייצר ייצוג חדש של סדרת הקלט באותו אורך כמו הסדרה המקורית, כאשר כל איבר בסדרה החדשה ייצג איבר בסדרה המקורית בתוספת מידע על הקשר שלו לשאר האיברים. בפועל בכל פעם לוקחים איבר אחד ומבצעים עליו מכפלה פנימית עם שאר האיברים בסדרה, כאשר איברים דומים בסדרה יתנו ערכים גבוהים ואיברים שונים בסדרה יתנו ערכים נמוכים. דמיון בין איברים נמדד על פי הקשר שיש ביניהם – ב-NLP זה יכול להיות מילים שסביר שיופיעו בסמיכות, ובתמונה זה יכול להיות פיקסלים דומים. כל מכפלה פנימית בין שני איברים נותנת מקדם שהוא מספר ממשי, וכך ניתן לסכם את מכפלת כל המקדמים באיברים המקוריים, ולקבל ייצוג חדש לאיבר המקורי המכיל גם קשר בין האיבר הנוכחי לאיברים דומים בסדרה.

באופן פורמלי, בשביל לחשב את ה-self-attention יוצרים שלוש מטריצות עבור כל אחד מאיברי הכניסה. המטריצות נקראות Query, Key, Value וכל אחת מהן נוצרת על ידי הכפלה של מטריצת משקלים באיבר הקלט. בעזרת מטריצות אלו מחשבים את ה-attention score:

$$\text{Attention}(\text{Query}, \text{Key}, \text{Value}) = \text{SoftMax}\left(\frac{\text{Q} \cdot \text{K}}{\sqrt{d_k}}\right) \cdot \text{V}$$

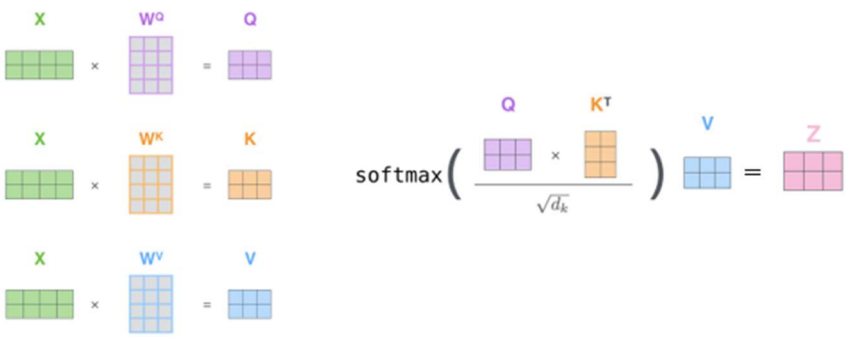
כדי להבין כיצד הנוסחה הזו מסייעת במציאת קשר בין איברים, נבחן כל איבר בנפרד. עבור סדרת קלט x מקבלים שלוש מטריצות, כאשר כל איבר בסדרה המקורית x_i יוצר שורה בכל אחת מהמטריצות. כאשר לוקחים את השורה $q_i = \text{Q} \cdot x_i$, ומכפילים אותה בכל אחת מהשורות במטריצה K, מקבלים וקטור חדש, שכל איבר j בוקטור אומר עד כמה יש קשר בין האיברים i, j בסדרה המקורית. ביצוע ההכפלה הזו עבור כל סדרת הקלט יוצר מטריצה חדשה בה כל שורה מייצגת את הקשר בין איבר מסוים לשאר איברי הסדרה. ההכפלה הזו היא בעצם $\text{Q} \cdot \text{K}$, כאשר כל מכפלה $q_i^T k_j$ מייצגת את הקשר בין האיבר i לאיבר j. את התוצאה מחלקים בשורש של מימד ה-embedding כדי לשמור על יציבות הגרדיאנט, ולאחר מכן מנרמלים על ידי SoftMax. באופן הזה מקבלים מטריצה של מספרים בטווח $[0, 1]$, המייצגים כאמור את הקשר בין כל שני איברים בסדרה המקורית. נסמן כל איבר במטריצה ב-w_{ij}, ונוכל לקבל אותו ישירות על ידי הנוסחה:

$$w_{ij} = \text{SoftMax}\left(\frac{q_i \cdot k_j}{\sqrt{d_k}}\right) = \frac{\exp\left(\frac{q_i^T k_j}{\sqrt{d_k}}\right)}{\sum_{s=1}^{n} \exp\left(\frac{q_i^T k_s}{\sqrt{d_k}}\right)}$$

כעת בעזרת משקלים אלו בונים ייצוג חדש לסדרה המקורית, על ידי הכפלתם בוקטור V:

$$z_i = \sum_{j=1}^{n} w_{ij} v_j = \frac{\sum_{j=1}^{n} \exp(q_i^T k_j)}{\sum_{s=1}^{n} \exp(q_i^T k_s)} v_j$$

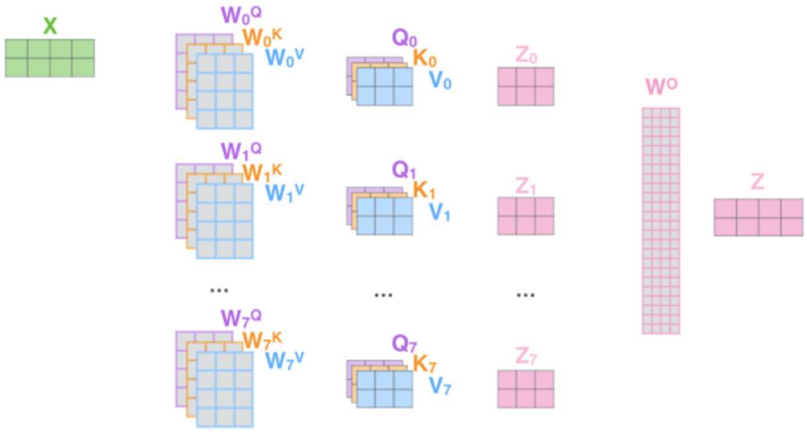
הסדרה המתקבלת z היא למעשה ייצוג חדש של הסדרה המקורית, כאשר כל איבר z_i מייצג איבר בסדרה המקורית יחד עם מידע על הקשרים בינו לבין שאר איברי הסדרה. את הסדרה המתקבלת ניתן להעביר ב-decoder ובכך לבצע כל מיני משימות, כפי שיוסבר בהמשך.



איור 8.5 ביצוע Self-attention – יצירת מטריצות Query, Key, Value (שמאל) וחישוב ה-attention score (ימין).

8.2.3 Multi Head Attention

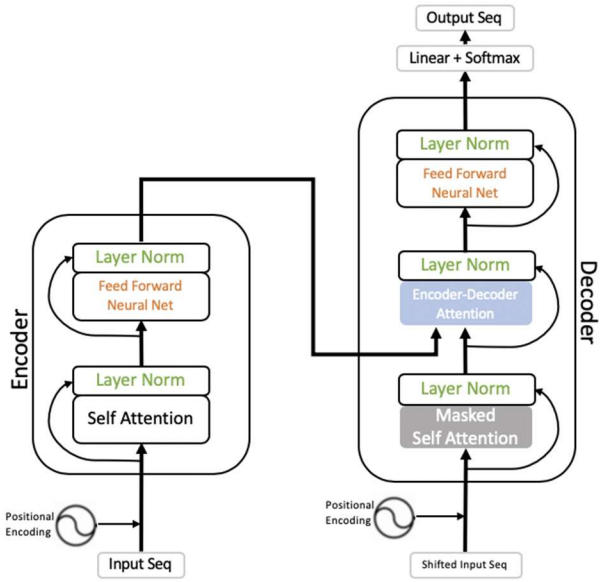
ניתן לבצע את פעולת ה-self-attention מספר פעמים במקביל. כל פעם מקבלים שלשה $(\text{Q}_\text{r}, \text{K}_\text{r}, \text{V}_\text{r})$ ובעזרתה מחשבים את ה-attention score. כל שלשה כזו נקראת attention head, וחיבור במקביל של כמה heads נקרא Multi-head attention. באופן הזה לכל איבר כניסה x_i יש כמה ייצוגים שונים z_{ir}, אותם ניתן להכפיל במטריצת משקלים w_o ולקבל את הייצוג המשוקלל של סדרת הכניסה.



איור 8.5 Self-attention with 8 heads.

8.2.4 Transformer End to End

בעזרת multi head attention ו-positional encoding ניתן לבנות Transformer – ארכיטקטורה עבור סדרות המבוססת רק על attention ללא רכיבי זיכרון.



איור 8.6 Transformer.

כפי שניתן לראות באיור ה-transformer מורכב משני חלקים – encoder ו-decoder. ה-encoder מקבל סדרה מסוימת x (לרוב אחרי שעברה embedding מסוים) ומבצע עליה positional encoding. לאחר מכן הסדרה עוברת דרך residual block של self-attention, ומתקבלת התוצאה $x + \text{attention}(x)$. על תוצאה זו, כמו גם על שאר המוצאים של השכבות השונות ב-transformer, מבצעים layer normalization (כפי שהוסבר בפרק 5.1.4). לאחר מכן יש residual block נוסף, המכיל שכבת fully connected, ומשם יוצא הפלט לכיוון ה-encoder.

ה-decoder בנוי בצורה מאוד דומה, עם שני הבדלים עיקריים: הקלט שלו הוא איברי הפלט שהיו עד כה, ובנוסף יש בתחילת ה-decoder שכבה של Masked self-attention. שכבה זו מקבלת את כל איברי הפלט שהיו עד כה, ומטרת ה-decoder היא ללמוד בעצמו מה האיבר הבא של הפלט. בשלב הראשון ה-decoder מבצע self-attention על איברי הסדרה שהתקבלו עד כה, וכך לומד ייצוג חדש שלהם, המכיל גם את הקשר בין איברי סדרה זו.

לאחר השכבה הראשונה יש שכבת multi head attention נוספת הנקראת Encoder-Decoder Attention, כיוון שהיא שילוב של ה-encoder וה-decoder: המטריצות Key, Value נלקחות מה-encoder וה-Query מגיע מה-decoder. כעת כשמבצעים את המכפלה $Q \cdot K$, לא מחפשים דמיון בין איברים של אותה סדרה אלא בין האיברים של סדרת הפלט (בייצוג שלהם לאחר ה-encoder) לבין איברי סדרת הקלט (בייצוג שלהם לאחר שכבת ה-masked). שלב זה דומה מאוד ל-attention המקורי, רק שהייצוגים שהתקבלו לא נעזרו ברכיבי זיכרון. כאמור, המכפלה $Q \cdot K$ מייצרת מטריצת משקלים שכל איבר בה אומר מה היחס בין איבר בסדרה המקורית לבין איבר בסדרת הפלט. את

המטריצה הזו מכפילים ב-V וכך מקבלים וקטור מסוים שהוא ייצוג חדש של איבר הפלט הבא. וקטור זה עובר בשכבת FC וב-SoftMax, וכך מתקבל איבר הפלט.

ניקח דוגמא ממאמר שנקרא DETR המראה כיצד ניתן להשתמש ב-transformer בשביל זיהוי אובייקטים בתמונה. בשלב הראשון לוקחים כל פיקסל בתמונה ומשווים אותו לשאר הפיקסלים (זוהי בעצם המכפלה $Q \cdot K$). באופן הזה ניתן למצוא אזורים דומים ושונים בתמונה, כאשר דמיון ושוני זה לאו דווקא פיקסלים עם ערכים קרובים, אלא זה יכול להיות למשל שני אזורים שונים בפנים של אדם. לאחר מכן מייצרים ייצוג חדש לתמונה, בעזרת המשקלים והכפלתם ב-V. שלב זה למעשה מאפשר לבצע זיהוי של אובייקטים, בלי לדעת מה הם אותם אובייקטים. בשביל לבצע סיווג לכל אובייקט שזוהה, מעבירים את הייצוג החדש של התמונה ב-decoder, כאשר ה-Query שמכניסים זה כל מיני לייבלים אפשריים, ומחפשים מבין כל ה-Query את הפלט של ה-decoder שמצליח לייצר תמונה שהכי דומה ל-Query.

אם למשל יש תמונה גדולה ויש אזור מסוים בו יש חתול, אז ה-encoder מוצא איפה החתול בתמונה, וה-decoder משווה את האזור הזה לכל מיני חיות אפשריות. כל Query שלא יהיה חתול, המכפלה $Q \cdot K$ תהיה קרובה ל-0, וה-decoder יזהה שה-Query הנוכחי לא תואם לאובייקט שזוהה. אך כאשר ה-Query יהיה חתול, אז כיוון ש-Q, K דומים אחד לשני, המכפלה $Q \cdot K$ תביא לכך שהייצוג החדש $z_i = \sum_{j=1}^{n} w_{ij} v_j$ כן יהיה דומה לחתול. ייצוג זה עובר בשכבת FC, ולאחר מכן ה-SoftMax יסווג את התמונה הזו כחתול.