

Wettir nettih eual IA M-itluA negS ysmet

ייצוג טקסט

מטוקנים לוקטורים

Text Representation

From Tokens to Vectors

פרק מתוך ספר ארכיטקטורת Transformer

Chapter from the Transformer Architecture Book

נכתב באמצעות מערכת כתיבה רב־סוכנית

Written using a Multi-Agent Authoring System

2025

ייצוג טקסט: מטוקנים לוקטורים

Text Representation: From Tokens to Vectors

פרק 2 מתוך:

ארכיטקטורת Transformer ומודלי שפה גדולים

Transformer Architecture and Large Language Models

מאת: ד"ר יורם סגל

By: Dr. Yoram Segal

מערכת הכתיבה הרב-סוכנית:

Source Research	תורוקמ שופיח
Content Drafting	ןכות תביתכ
Code Implementation	דוק סושיי
Math Review	טיטמתמ הקידב
Citations	סיטוטיצ
Hebrew Editing	תינושל הכירע
Architecture Review	תינוטקטיכרא הריקס

זכויות יוצרים

Copyright © 2025

כל הזכויות שמורות. אין לשכפל, להעתיק, לצלם, להקליט, לתרגם, לאחסן במאגר מידע, לשדר או לקלוט בכל דרך או אמצעי אלקטרוני, אופטי, מכני או אחר - כל חלק שהוא מהחומר בספר זה.

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the author.

הערת תוכנה:

ספר זה נכתב באמצעות מערכת כתיבה רב-סוכנית מבוססת Claude Sonnet 4.5. שבעה סוכני AI ייעודיים שיתפו פעולה ליצירת תוכן איכותי: סוכן מחקר מקורות, סוכן כתיבה, סוכן קוד, סוכן בדיקה מתמטית, סוכן ציטוטים אקדמיים, סוכן עריכה לשונית, וסוכן סקירה ארכיטקטונית.

Software Notice: This book was created using a multi-agent authoring system based on Claude Sonnet 4.5. Seven specialized AI agents collaborated to produce high-quality content: source research agent, content drafting agent, code implementation agent, math review agent, academic citation agent, Hebrew language editor, and architecture review agent.

תקציר

פרק זה עוסק בשאלה יסודית: כיצד מחשבים הופכים טקסט אנושי למשהו שהם יכולים לעבד? אנו עוקבים אחר המסע המלא מהמשפט "החתול ישן על הספה" ועד לייצוג מתמטי מלא במרחב וקטורי רב-ממדי.

הפרק מכסה שלושה שלבים מרכזיים: **טוקניזציה** - פירוק הטקסט ליחידות בסיסיות (טוקנים) באמצעות אלגוריתמים כמו BPE, WordPiece, SentencePiece; **שכבות Embedding** - המרת טוקנים לווקטורים צפופים במרחב \mathbb{R}^d שמקודדים משמעות סמנטית; ו**קידוד מיקום** - הוספת מידע על מיקום הטוקן ברצף באמצעות פונקציות סינוס וקוסינוס. התוצאה הסופית היא מטריצה של ווקטורים שמכילה את כל המידע הדרוש למודל ה-Transformer - גם מה כל מילה אומרת (מהembedding), וגם איפה היא נמצאת (מקידוד המיקום). הבנת תהליך זה חיונית להבנת ארכיטקטורת ה-Transformer כולה. הפרק כולל דוגמאות קוד ב-Python, ויזואליזציות של מרחבי embedding ודפוסי קידוד מיקום, והפניות למקורות אקדמיים מובילים בתחום.

הערה: ניתוח מעמיק של קידוד מיקום מופיע בספר נלווה ייעודי.

Abstract

(snekoT) stinu cisab otni txet gnikaerb - **noitazinekoT** :segats niam eerht srevoc retpahc ehT
gnitrevnoc - **sreyal gniddebme** ;eeciPecnetneS dna ,eeciPdroW ,EPB ekil smhtirogla gnisu
gnidocne lanoitisoP dna ;gninaem citnames edocne taht ecaps \mathbb{R}^d ni srotcev esned ot snekot
enisoc dna enis gnisu ecneuqes eht ni noitisop s'nekoT eht tuoba noitamrofni gnidda -
.snoitcnuf

eht yb dedeen noitamrofni eht lla gniniatnoc srotcev fo xirtam a si tluser lanif ehT
detacol si ti erehw dna (gniddebme morf) syas drow hcae tahw htob - ledom remrofsnarT
eht gnidnatsrednu rof laitnesse si ssecorp siht gnidnatsrednU .(gnidocne lanoitisop morf)
.erutcetihcra remrofsnarT eritne

dna secaps gniddebme fo snoitazilausiv ,selpmaxe edoc nohtyP sedulcni retpahc ehT
.dleif eht ni secruos cimedaca gnidael ot secnerefer dna ,snrettap gnidocne lanoitisop
noinapmoc detacided a ni sraepa gnidocne lanoitisop fo sisylana htped-ni nA :etoN
.koob

תוכן העניינים

vi

תקציר

v

Abstract

1	ייצוג טקסט: מטוקנים לווקטורים	1
1	1.1 מבוא: האתגר של ייצוג שפה למחשבים	1.1
1	1.2 טוקניזציה: פירוק הטקסט ליחידות בסיסיות	1.2
1	1.2.1 מה זה טוקן ולמה הוא חשוב?	1.2.1
2	1.2.2 שלוש גישות לטוקניזציה	1.2.2
2	1.2.3 אלגוריתם BPE: הפתרון האלגנטי לבעיית הטוקניזציה	1.2.3
3	1.2.4 חלופות ל-BPE: WordPiece ו-SentencePiece	1.2.4
4	1.3 שכבות Embedding: מטוקנים לווקטורים	1.3
4	1.3.1 מה זה Embedding ולמה הוא כל כך חזק?	1.3.1
4	1.3.2 מטריצת ה-Embedding: הטבלה המרכזית	1.3.2
5	1.3.3 איך לומדים את ה-Embeddings?	1.3.3
5	1.3.4 ויזואליזציה של מרחב ה-Embedding	1.3.4
6	1.4 קידוד מיקום: הוספת המימד הרביעי - זמן	1.4
6	1.4.1 הבעיה: Transformers אינם מודעים לסדר	1.4.1
6	1.4.2 הפתרון: קידוד מיקום Sinusoidal	1.4.2
7	1.4.3 למה סינוס וקוסינוס? התובנה המתמטית	1.4.3
7	1.4.4 ויזואליזציה של דפוסי קידוד המיקום	1.4.4
7	1.4.5 חלופות לקידוד סינוסואידי	1.4.5
8	1.5 הייצוג הסופי: חיבור Embedding וקידוד מיקום	1.5
8	1.5.1 פעולת החיבור הפשוטה	1.5.1
9	1.5.2 נורמליזציה ושיקולים נוספים	1.5.2
9	1.6 סיכום: מטקסט לייצוג מתמטי מלא	1.6
1.7	English References	11

1 ייצוג טקסט: מטוקנים לווקטורים

תקציר

פרק זה עוקב אחר המסע המרתק של טקסט מהצורה האנושית הטבעית שלו - מילים ומשפטים - אל הצורה המתמטית שמחשבים יכולים לעבד. נחקור כיצד משפט פשוט כמו "החתול ישן על הספה" הופך לרצף של וקטורים במרחב רב-ממדי, תוך שימוש בטכניקות טוקניזציה, שכבות embedding, וקידוד מיקום. הבנת תהליך זה חיונית להבנת ארכיטקטורת ה-Transformer כולה.

1.1 מבוא: האתגר של ייצוג שפה למחשבים

בני אדם מבינים שפה באופן טבעי. כאשר אנו קוראים את המילה "חתול", מוחנו מיד מעלה תמונה של יצור פרוותי בעל ארבע רגליים, יחד עם רשת עשירה של אסוציאציות: חמימות, עצמאות, ציפורניים חדות. מחשבים, לעומת זאת, אינם "מבינים" מילים. הם עובדים עם מספרים בלבד - ספרות בינאריות, מטריצות, וקטורים.

האתגר המרכזי בעיבוד שפה טבעית (Natural Language Processing - NLP) הוא לגשר על הפער הזה. כיצד נוכל להמיר טקסט אנושי למשהו שמחשב יכול לעבד, תוך שמירה על המשמעות והקשרים הסמנטיים? [1] השאלה הזו עומדת בלב ארכיטקטורת ה-Transformer, והתשובה טמונה בשלושה שלבים מרכזיים: טוקניזציה, embedding, וקידוד מיקום.

בפרק זה נעקוב אחר המסע המלא - מהמשפט המקורי, דרך פירוקו לחתיכות קטנות יותר (טוקנים), המרתן לווקטורים מתמטיים, וכלה בהוספת מידע על המיקום בטקסט. בסוף התהליך, נקבל ייצוג עשיר שמאפשר למודל ה-Transformer לעבד ולהבין טקסט בצורה שלא הייתה אפשרית קודם לכן.

1.2 טוקניזציה: פירוק הטקסט ליחידות בסיסיות

הצעד הראשון בעיבוד טקסט הוא **טוקניזציה** - תהליך פירוק המשפט ליחידות בסיסיות שנקראות "טוקנים". נראה לכאורה שמדובר בפעולה פשוטה: לקחת משפט ולפצל אותו למילים. אבל בפועל, הדבר הרבה יותר מורכב ומעניין.

1.2.1 מה זה טוקן ולמה הוא חשוב?

טוקן הוא היחידה הבסיסית שהמודל רואה. זה יכול להיות מילה שלמה, חלק ממילה, או אפילו תו בודד - תלוי באסטרטגיית הטוקניזציה שנבחרה. בחירה נכונה של אסטרטגיית טוקניזציה משפיעה באופן דרמטי על ביצועי המודל, גודלו, ומהירות העיבוד. נסתכל על דוגמה פשוטה בעברית:

המשפט המקורי:

"החתול ישן על הספה"

איך נפרק את המשפט הזה? האם נפצל לפי מילים? לפי תווים? או אולי משהו באמצע?

1.2.2 שלוש גישות לטוקניזציה

קיימות שלוש גישות עיקריות לטוקניזציה, כל אחת עם היתרונות והחסרונות שלה:

גישה 1: טוקניזציה ברמת המילה (Word-level Tokenization)

בגישה זו, כל מילה הופכת לטוקן נפרד. המשפט שלנו יהפוך ל: ["החתול", "ישן", "על", "הספה"]

היתרון: פשוט ואינטואיטיבי. כל מילה שומרת על המשמעות השלמה שלה. החיסרון: אוצר המילים (vocabulary) יכול להיות ענקי. בשפה העברית יש מאות אלפי מילים, ובאנגלית אפילו יותר. מילים נדירות או חדשות (out-of-vocabulary - OOV) פשוט לא יהיו במילון, ולא נוכל לייצג אותן.

גישה 2: טוקניזציה ברמת התו (Character-level Tokenization)

כאן, כל תו (אות) הופך לטוקן. המשפט שלנו יהפוך ל: ["ה", "ח", "ת", "ו", "ל", " ", "י", "ש", "נ", " ", "ע", "ל", " ", "ה", "ס", "פ", "ה"]

היתרון: אוצר מילים זעיר! בעברית יש רק בערך 30 תווים, כולל רווחים וסימני פיסוק. החיסרון: רצפים ארוכים מאוד. משפט קצר הופך לעשרות טוקנים, מה שמקשה על המודל ללמוד דפוסים סמנטיים ברמת המילה. [2]

גישה 3: טוקניזציה ברמת תת-המילה (Subword Tokenization)

זו הגישה המודרנית והנפוצה ביותר, המאזנת בין שתי הקצוות. המשפט שלנו עשוי להיות מפורק ל: ["ה", "חתול", "ישן", "על", "ה", "ספה"] שימו לב: הטוקניזציה הפרידה את התחיליות "ה" (ה"א הידיעה) מהמילים עצמן. זה מאפשר למודל לזהות דפוסים משותפים (כמו "ה" כתחילית) תוך שמירה על יכולת לייצג מילים נדירות באמצעות חלקי מילים קטנים יותר.

1.2.3 אלגוריתם BPE: הפתרון האלגנטי לבעיית הטוקניזציה

אחד האלגוריתמים הפופולריים ביותר לטוקניזציה ברמת תת-המילה הוא Byte Pair Encoding (BPE) [3]. האלגוריתם פשוט אך יעיל להפליא:

שלב א': התחלה ברמת התו

מתחילים עם כל תו כטוקן נפרד. לדוגמה, אם יש לנו את המילה "gninrael" שמופיעה 10 פעמים בקורפוס, נייצג אותה כ: g n i n r a e l (01□)

שלב ב': מיזוג זוגות תכופים

מוצאים את זוג התווים הכי תכוף (נניח "re") וממזגים אותו לטוקן יחיד: n i n r a e l g n i n r a e l □ g

שלב ג': חזרה

ממשיכים את התהליך: מוצאים את הזוג הכי תכוף הבא, ממזגים, וחוזר חלילה. אחרי מספר איטרציות, נקבל משהו כמו: (nekot elgnis a sa) gninrael התוצאה: אוצר מילים בגודל סביר (בדרך כלל 30000–50000 טוקנים) שמכיל גם מילים שלמות נפוצות וגם חלקי מילים שניתן לשלב לייצוג מילים נדירות. זו הסיבה ש-GPT-2 ו-GPT-3 משתמשים ב-BPE [4].

1.2.4 חלופות ל-BPE: WordPiece ו-SentencePiece

אלגוריתמים נוספים מציעים וריאציות על הרעיון:
WordPiece (משמש ב-BERT) [5] דומה ל-BPE, אבל במקום למזג את הזוג הכי תכוף, הוא בוחר את הזוג שמשפר הכי הרבה את הסבירות של הקורפוס על פי מודל שפה. זה אופטימיזציה קצת יותר מתוחכמת.
SentencePiece (משמש ב-T5 ו-LLaMA) [6] מתייחס לטקסט כזרם גולמי של Unicode, כולל רווחים. זה הופך אותו לבלתי-תלוי בשפה - הוא עובד באותה מידה טובה על עברית, אנגלית, סינית, או כל שפה אחרת, ללא צורך בעיבוד מוקדם ספציפי לשפה.
כפי שמציג איור 1, תהליך הטוקניזציה הוא הצעד הקריטי הראשון שהופך טקסט אנושי למשהו שמודל יכול לעבד.



איור 1: תהליך הטוקניזציה: המרת משפט לרצף טוקנים

איור 1 ממחיש את הזרימה המלאה מטקסט מקורי לטוקנים. שימו לב כיצד הטוקניזר מפצל את "החתול" ל"ה" + "חתול" - זיהוי אינטליגנטי של תחילית נפוצה שחוסך מקום באוצר המילים.

1.3 שכבות Embedding: מטוקנים לווקטורים

כעת, אחרי שפירקנו את הטקסט לטוקנים, אנחנו מתמודדים עם שאלה חדשה: איך נייצג כל טוקן במספרים? התשובה טמונה במושג של embeddings - ייצוגים וקטוריים צפופים של טוקנים.

1.3.1 מה זה Embedding ולמה הוא כל כך חזק?

Embedding הוא פשוט וקטור של מספרים ממשיים. לדוגמה, הטוקן "חתול" עשוי להיות מיוצג על ידי וקטור בגודל $d = 512$:

$$(1) \quad \text{embedding}(\text{"לוחח"}) = [-0.7, 1.2, 0.13, -0.45, \dots, 0.04] \in \mathbb{R}^{512}$$

למה 512 מימדים? זו החלטת עיצוב. Transformer המקורי [1] השתמש ב- $d_{\text{model}} = 512$, אבל מודלים מודרניים כמו GPT-3 משתמשים ב- $d = 12288$ למודל הגדול ביותר. הרעיון המרכזי: **מילים עם משמעות דומה יקבלו וקטורים קרובים במרחב הווקטורי**. זה אומר שהמרחק (למשל, מרחק אוקלידי או קוסינוס) בין gniddebme ("חתול") ל- gniddebme ("כלב") יהיה קטן יחסית, כי שתי המילים מתייחסות לחיות בית. לעומת זאת, המרחק בין gniddebme ("חתול") ל- gniddebme ("מחשב") יהיה גדול יותר.

1.3.2 מטריצת Embedding: הטבלה המרכזית

בפועל, כל ה-embeddings של הטוקנים שמורים במטריצה גדולה אחת שנקראת **מטריצת Embedding**:

$$(2) \quad E \in \mathbb{R}^{V \times d}$$

כאשר:

V = גודל אוצר המילים (vocabulary size) - למשל 50000 טוקנים

d = מימד ה-embedding - למשל 512

כל שורה במטריצה E מייצגת את ה-embedding של טוקן ספציפי. כאשר המודל רוצה לקבל את ה-embedding של הטוקן "חתול" (נניח שהוא מופה ל-1523 token ID), הוא פשוט מחלץ את שורה 1523 מהמטריצה:

$$(3) \quad x_{\text{חתול}} = E[1523, :] \in \mathbb{R}^{512}$$

פעולה זו נקראת **embedding lookup** - חיפוש בטבלה. היא מהירה ויעילה מאוד.

1.3.3 איך לומדים את ה-Embeddings?

קיימות שתי גישות עיקריות:

גישה 1: Embeddings נלמדים מאפס

המודל מתחיל עם embeddings אקראיים ולומד אותם במהלך האימון, באמצעות back-propagation. זו הגישה הנפוצה במודלי Transformer מודרניים. ה-embeddings מותאמים ספציפית למשימה - למשל, תרגום, סיכום, או יצירת טקסט.

גישה 2: Embeddings מאומנים מראש

אפשר להשתמש ב-embeddings שאומנו מראש על קורפוס ענק, כמו Word2Vec [7] או GloVe [8]. גישה זו הייתה פופולרית לפני עידן ה-Transformer, אבל כיום פחות נפוצה - מודלים מעדיפים ללמוד embeddings מותאמים למשימה.

1.3.4 ויזואליזציה של מרחב ה-Embedding

מרחב של 512 מימדים קשה לדמיין. אבל נוכל להשתמש בטכניקות הורדת מימד כמו t-SNE או PCA כדי להקרין את הווקטורים למישור דו-ממדי, כפי שמוצג באיור 2.



איור 2: ויזואליזציה דו-ממדית של מרחב embeddings - מילים קרובות סמנטית מקובצות יחדיו

באיור 2 ניתן לראות כיצד מילים מאותה קטגוריה סמנטית (חיות, טכנולוגיה, פעלים) מתקבצות יחדיו במרחב. זו הדוגמה היפה לכוח של embeddings - הם לומדים לייצג משמעות באופן אוטומטי, ישירות מהדאטה.

1.4 קידוד מיקום: הוספת המימד הרביעי - זמן

עכשיו הגענו לנקודה קריטית. יש לנו טוקנים, יש לנו embeddings שמייצגים את המשמעות של כל טוקן. אבל... חסר לנו משהו מהותי: **סדר**.

1.4.1 הבעיה: Transformers אינם מודעים לסדר

להבדיל ממודלים רקורנטיים כמו LSTM, שמעבדים טוקנים אחד אחרי השני ולכן "יודעים" מה בא לפני מה, ארכיטקטורת ה-Transformer מסתכלת על כל הטוקנים **במקביל**. זה נותן לה יתרון עצום במהירות ויכולת מקבול, אבל יוצר בעיה: המודל אינו יודע אם "החתול רודף אחרי העכבר" או "העכבר רודף אחרי החתול". בלי מידע על מיקום, שני המשפטים נראים זהים! זו בעיה קריטית. סדר המילים נושא משמעות עצומה. "לא טוב" זה ההפך מ"טוב לא רע". אנחנו חייבים לספק למודל מידע על **מיקום כל טוקן ברצף**.

1.4.2 הפתרון: קידוד מיקום Sinusoidal

המאמר המקורי של Transformer [1] הציע פתרון אלגנטי: להוסיף לכל embedding ווקטור נוסף שמקודד את המיקום שלו ברצף, באמצעות פונקציות סינוס וקוסינוס:

$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right) \quad (4)$$

$$PE_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right) \quad (5)$$

כאשר:

pos - מיקום הטוקן ברצף (0, 1, 2, 3, ...)

i - אינדקס המימד בווקטור (0, 1, 2, ..., $d/2 - 1$)

d - מימד ה-embedding (למשל 215)

נוסחאות 4 ו-5 מגדירות קידוד ייחודי לכל מיקום. למימדים הזוגיים (0, 2, 4, ...) נוסחאות 4 ו-5 משתמשות בסינוס, ולמימדים האי-זוגיים (1, 3, 5, ...) משתמשות בקוסינוס. כל מימד מתנדנד בתדירות שונה - מימדים נמוכים יותר מתנדנדים לאט, ומימדים גבוהים יותר מהר.

1.4.3 למה סינוס וקוסינוס? התובנה המתמטית

הבחירה בפונקציות טריגונומטריות אינה אקראית. יש לה כמה יתרונות מרתקים: **יתרון 1: ייחודיות** - כל מיקום מקבל קידוד ייחודי. אין שני מיקומים שונים עם אותו ווקטור קידוד.

יתרון 2: מרחקים יחסיים - המודל יכול ללמוד בקלות מרחקים יחסיים. הקשר בין מיקום k למיקום $k + 3$ הוא קבוע, ללא קשר למיקום המוחלט.

יתרון 3: אקסטרפולציה - המודל יכול להתמודד עם רצפים ארוכים יותר ממה שראה באימון, כי הפונקציות מוגדרות לכל מיקום חיובי.

הערה חשובה: ניתוח מעמיק ומקיף של קידוד המיקום, כולל הוכחת הייחודיות, ניתוח אורכי הגל, ויכולות האינטרפולציה, מופיע בספר הנלווה [9]. כאן נסתפק בהבנה הבסיסית של המושג.

1.4.4 ויזואליזציה של דפוסי קידוד המיקום

איור 3 מציג מפת חום של קידוד מיקום עבור 50 מיקומים ו-128 מימדים. כל שורה מייצגת מיקום, וכל עמודה מייצגת מימד. הצבע מייצג את הערך של הקידוד (כחול = ערך נמוך, אדום = ערך גבוה).



איור 3: דפוס קידוד המיקום: מפת חום המציגה ערכי PE עבור 50 מיקומים ו-128 מימדים באיור 3 ניתן לראות בבירור את הדפוס הגלי - מימדים נמוכים יותר (צד שמאל) משתנים לאט על פני מיקומים, בעוד מימדים גבוהים יותר (צד ימין) משתנים מהר יותר. זה בדיוק התכונה שמאפשרת למודל ללמוד גם מרחקים קצרים וגם מרחקים ארוכים בין מילים.

הפניה לספר הנלווה: לפרטים מקיפים על קידוד מיקום sin/cos, כולל הצגה דו-ממדית, סקלינג ל-N-ממדים, הוכחת ייחודיות, יכולת אינטרפולציה, וניתוח אורכי גל, ראו את הספר הנלווה:

../Sin-Cos-Encoding-Book/main.pdf

הספר הנלווה מכסה בפירוט מלא את כל ההיבטים המתמטיים והגיאומטריים של קידוד המיקום בארכיטקטורת Transformer.

1.4.5 חלופות לקידוד סינוסואידי

בעוד הקידוד הסינוסואידי הוא הפתרון הקלאסי, קיימות גישות חלופיות: **Learned Positional Embeddings** (משמש ב-BERT) [5] - במקום נוסחה קבועה, המודל לומד טבלה של ווקטורי מיקום במהלך האימון. יתרון: גמישות. חיסרון: לא ניתן לאקסטרפולציה לאורכים לא-נראים.

Rotary Position Embedding (RoPE) [10] - מקודד מיקום באמצעות מטריצות סיבוב, מה שמאפשר לתשומת הלב להיות תלוי-מיקום באופן יעיל יותר. משמש במודלים כמו LLaMA ו-GPT-Neo.

ALiBi (Attention with Linear Biases) [11] - מוסיף הטיה ליניארית ישירות לציוני תשומת הלב, ללא קידוד מיקום מפורש. מאפשר אקסטרפולציה מצוינת לרצפים ארוכים.

1.5 הייצוג הסופי: חיבור Embedding וקידוד מיקום

עכשיו מגיע השלב האחרון: איחוד הembedding של הטוקן עם קידוד המיקום שלו.

1.5.1 פעולת החיבור הפשוטה

בארכיטקטורת Transformer המקורית, הפתרון פשוט להפתיע: **חיבור וקטורי רגיל:**

$$(6) \quad x_{final}[pos] = x_{embed}[pos] + PE[pos]$$

כאשר:

$$pos = x_{embed}[pos] \text{ - embedding של הטוקן במיקום } pos$$

$$PE[pos] = \text{ווקטור קידוד המיקום}$$

$$x_{final}[pos] = \text{הייצוג הסופי של הטוקן, שמשלב גם משמעות וגם מיקום}$$

נוסחה 6 מגדירה את הקלט הסופי למודל הTransformer. כל טוקן עכשיו מיוצג על ידי ווקטור שמכיל גם את המשמעות הסמנטית שלו (מהembedding) וגם את המיקום שלו ברצף (מקידוד המיקום).

למה חיבור ולא פעולה אחרת? זו שאלה מצוינת. החוקרים ניסו גם שרשור (noitanetacnoc) וגם פעולות אחרות, אבל חיבור פשוט התברר כיעיל ביותר - הוא שומר על מימד הווקטור ומאפשר למודל ללמוד איך "לערבב" את שני סוגי המידע בצורה אופטימלית.

1.5.2 נורמליזציה ושיקולים נוספים

במודלים מודרניים, לעיתים מוסיפים גם צעדים נוספים:
סקלינג - מכפילים את ה embedding ב $\sqrt{d_{model}}$ לפני החיבור עם קידוד המיקום [1]. זה נועד לאזן בין שני הרכיבים ולמנוע מקידוד המיקום "לבלוע" את ה embedding.
Dropout - מוסיפים רעש אקראי קל (מאפסים חלק מהערכים) כדי למנוע overfitting ולשפר את יכולת ההכללה של המודל.
התוצאה הסופית היא רצף של ווקטורים $[x_{final}[0], x_{final}[1], \dots, x_{final}[n-1]]$, כאשר כל ווקטור מייצג טוקן אחד עם המידע המלא שלו: מה הוא ואיפה הוא.

1.6 סיכום: מטקסט לייצוג מתמטי מלא

בואו נעקוב שוב אחרי המסע המלא, מההתחלה ועד הסוף:
שלב 1: טוקניזציה

"החתול ישן על הספה" \square ["הפס", "ה", "לע", "ישי", "לותח", "ה"]

שלב 2: המרה למזהים

["הפס", "ה", "לע", "ישי", "לותח", "ה"] \square [5, 1523, 892, 71, 5, 2103]

שלב 3: Embedding Lookup

כל מזהה \square ווקטור ב \mathbb{R}^{512}

שלב 4: קידוד מיקום

חישוב $PE[0], PE[1], \dots, PE[5]$ לפי נוסחאות הסינוס/קוסינוס

שלב 5: חיבור

$$x_{final}[i] = x_{embed}[i] + PE[i] \text{ לכל } i$$

התוצאה: מטריצה $X \in \mathbb{R}^{6 \times 512}$ (6 טוקנים \square 512 מימדים) שמכילה את כל המידע הדרוש למודל ה Transformer כדי להבין את המשפט - גם מה כל מילה אומרת, וגם באיזה סדר.
הצעד הבא: המטריצה הזו תועבר עכשיו לשכבות תשומת הלב (attention) של ה-Trans former, שם המודל יחשב את הקשרים בין המילים, יבין תלויות סמנטיות, ויצור ייצוגים עשירים עוד יותר. אבל זה כבר נושא לפרק הבא.
ייצוג הטקסט - התהליך שעקבנו אחריו בפרק זה - הוא הבסיס לכל מה שה Transformer יכול להשיג. בלי טוקניזציה נכונה, embeddings עשירים, וקידוד מיקום מדויק, אף אחד מהיכולות המדהימות של מודלים כמו GPT או BERT לא היה אפשרי. זהו שלב ראשון, אבל קריטי, במסע של הבנת שפה על ידי מכונות.

1.7 English References

- 1 A. Vaswani et al., “Attention is all you need,” in *Advances in neural information processing systems*, 2017, 5998–6008.
- 2 Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, “Character-aware neural language models,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 30, 2016.
- 3 R. Sennrich, B. Haddow, and A. Birch, “Neural machine translation of rare words with subword units,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, 1715–1725. DOI: [10.18653/v1/P16-1162](https://doi.org/10.18653/v1/P16-1162)
- 4 A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al., *Language models are unsupervised multitask learners*, 2019.
- 5 J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, 4171–4186. DOI: [10.18653/v1/N19-1423](https://doi.org/10.18653/v1/N19-1423)
- 6 T. Kudo and J. Richardson, “Sentencepiece: A simple and language independent approach to subword tokenization,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2018, 66–71. DOI: [10.18653/v1/D18-2012](https://doi.org/10.18653/v1/D18-2012)
- 7 T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013.
- 8 J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, 1532–1543. DOI: [10.3115/v1/D14-1162](https://doi.org/10.3115/v1/D14-1162)
- 9 Y. Segal, תרומת טיכראב מוקים דודיק, *Transformer: חותני*, PDF Document, Companion book covering 2D/N-dimensional sin/cos positional encoding, uniqueness properties, and wavelength analysis. Located at: [../Sin-Cos-Encoding-Book/main.pdf](https://Sin-Cos-Encoding-Book/main.pdf), 2025.
- 10 J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, “Roformer: Enhanced transformer with rotary position embedding,” *arXiv preprint arXiv:2104.09864*, 2021.
- 11 O. Press, N. A. Smith, and M. Lewis, “Train short, test long: Attention with linear biases enables input length extrapolation,” *arXiv preprint arXiv:2108.12409*, 2022.