

metsyS tnegA-itluM IA edualC htiw nettirW

ייצוג טקסט

מטוקנים לווקטוריים

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

יצוג טקסט: מטוקנים לווקטורים

Text Representation: From Tokens to Vectors

פרק 2 מתוד:

ארQUITקטורת 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	תורוקם שופיה
Content Drafting	וכות תביתכ
Code Implementation	דוק סושי
Math Review	תיתמתם הקידב
Citations	סיטוטיז
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 ייעודים שיתפו פעולה ליצור תוכן אינטואיטיבי: 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 בשאלת 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 ו; **שכבות Embedding** - המרת טוקנים לוקטורים צפופים במרחב \mathbb{R}^d שמקודדים משמעות סמנטיות; **և קידוד מיקום**

- הוספת מידע על מיקום הטוקן ברכף באמצעות פונקציות סינוס וקוסינוס.

התוצאה הסופית היא מטריצה של ווקטורים שמכילה את כל המידע הדרוש למודל Transformer - גם כל מילה אומرت (מהembedding), וגם איפה היא נמצאת (מקידוד המיקום). הבנת תהליך זה חיונית להבנת ארכיטקטורת Transformer כולה.

הפרק כולל דוגמאות קוד בPython, ויזואלייזציות של מרחבי embedding ודףusi קידוד מיקום, והפניות למקורות אקדמיים מוביילים בתחום.

הערה: ניתוח מעמיק של קידוד מיקום מופיע בספר נלווה "יעודי".

Abstract

(snekot) stinu cisab otni txet gnikaerb - **noitazinekoT** :segats niam eerht srevoc retpahc ehT gnitrevnoc - **sreyal gniddebmE** ;eceiPecnetneS dna ,eceiPdroW ,EPB ekil smhtirogl 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	1. ייצוג טקסט: מטוקנים לווקטורים
1	1.1 מבוא: האתגר של ייצוג שפה למחשבים
1	1.2 טוקניזציה: פירוק הטקסט ליחידות בסיסיות
1	1.2.1 מה זה טוקן ולמה הוא חשוב?
2	1.2.2 שלוש גישות לטוקניזציה
2	1.2.3 אלגוריתם BPE: הפתרון האלגנטiy לביעית הטוקניזציה
3	1.2.4 חלופות LE-BPE:- SentencePiece ו WordPiece
4	1.3 שכבות Embedding: מטוקנים לווקטורים
4	1.3.1 מה זה Embedding ולמה הוא כל כך חזק?
4	1.3.2 מטריצת Embedding:- הטללה המרכזית
5	1.3.3 איך לומדים את Embeddings?-?
5	1.3.4 ויזואלייזציה של מרחב ה-Embedding
6	1.4 קידוד מיקום: הוספת המימד הרביעי - זמן
6	1.4.1 הבעיה: Transformers אינס מודעים לסדר
6	1.4.2 הפתרון: קידוד מיקום Sinusoidal
7	1.4.3 למה סינוס וקוסינוס? התובנה המתמטית
7	1.4.4 ויזואלייזציה של דפוסי קידוד המיקום
7	1.4.5 חלופות לקידוד סינוסואידי
8	1.5 הייצוג הסופי: חיבור Embedding וקידוד מיקום
8	1.5.1 פועלות החיבור הפשוטה
9	1.5.2 נורמליזציה ושיקולים נוספים
9	1.6 סיכום: מטקסט לייצוג מתמטי מלא
1.7 English References	11

1 ייצוג טקסט: מוטקנים לווקטורים

תקציר

פרק זה עוקב אחר המשע המרתך של טקסט מהצורה האנושית הטבעית שלו - מילים ומשפטים - אל הצורה המתמטית שמחשובים יכולים לעבוד. נחקרו כיצד משפט פשוט כמו "החתול ישן על הספה" הופך לרץ' של וקטורים במרחב רב-ממדי, תוך שימוש בטכניקות טוקנייזציה, שכבות embedding, וקידוד מיקום. הבנת תהליך זה חיונית להבנת ארכיטקטורת 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 צור פרוטי בעל ארבע רגליים, יחד עם רשת עשרה של אסוציאציות: חמיימות, עצמאות, ציפורניים חזות. 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 לעבוד, תוך שמירה על המשמעות והקשרים הסמנטיים? [1] השאלה הזאת עומדת בלב ארכיטקטורת Transformer, והתשובה טמונה בשלושה שלבים מרכזיים: טוקנייזציה, embedding, וקידוד מיקום. בפרק זה נעקوب אחר המשע המלא - מהמשפט המקורי, דרך פירוקו לחתיכות קטנות יותר (טוקנים), המרתן לווקטורים מתמטיים, וכלה בהוספת מידע על המיקום בטקסט. בסוף התהליך, קיבל ייצוג עשיר שמאפשר למודל Transformer לעבוד ולהבין טקסט בצורה שלא הייתה אפשרית קודם לכן.

1.2 טוקנייזציה: פירוק הטקסט ליחידות בסיסיות

הצעד הראשון בעיבוד טקסט הוא טוקנייזציה - תהליך פירוק המשפט ליחידות בסיסיות שנkirאות "טוקנים". נראה לכאהר שמדובר בפעולה פשוטה: לחת 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 אפילותו בודד - תלוי באסטרטגיית הטוקנייזציה שנבחרה. בחירה נכונה של אסטרטגיית טוקנייזציה משפיעה באופן דרמטי על ביצועי המודל, גודלו, ומהירות העבודה. 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 לTokenizerיזציה

קיימות שלוש גישות עיקריות לTokenizerיזציה, כל אחת עם יתרונות וחסרונות שלה:
גישה 1:Tokenizerיזציה ברמת המילה (Word-level Tokenization)

בגישה זו, כל מילה הופכת לTOKEN נפרד. המשפט שלנו יופיע ל: [”חחות”, ”ישן”, ”על”, ”הספה”]

היתרון: פשוט ואינטואיטיבי. כל מילה שומרת על המשמעות השלמה שלה.

החסרון: אוצר המילים (vocabulary) יכול להיות ענק. בשפה העברית יש מאות אלפי מילים, ובאנגלית אף יותר. מילים חדשות או OOV (out-of-vocabulary) פשוט לא יהיו במילון, ולא יוכל לייצג אותן.

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

כאן, כל תו (אות) הופך לTOKEN. המשפט שלנו יופיע 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.

החסרון: רצפים ארוכים מאוד. משפט קצר הופך לעשרות TOKENים, מה שמקשה על המודל ללמידה דפוסים סמנטיים ברמת המילה. [2]

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

זו הגישה המודרנית והנפוצה ביותר, המאוזנת בין שתי הקצויות. המשפט שלנו עשוי להיות מפרק ל: [”ה”, ”חות”, ”ישׁו”, ”על”, ”ה”, ”ספה”]

משמעותו לב: tokenizeriza את התחיליות ”ה” (ה”א הידעה) מהמילים עצמן. זה מאפשר למודל לזהות דפוסים מסוימים (כמו ”ה” כתחילית) תוך שמירה על יכולת לייצג מילים חדשות באמצעות חלקים קטנים יותר.

1.2.3 אלגוריתם BPE: הפתרון האלגנטיבי לבעיית tokenizerיזציה

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

שלב א': התחלת ברמת התו

מתחילה עם כל תו כTOKEN נפרד. לדוגמה, אם יש לנו את המילה ”gninrael” שמופיעה 10 פעמים בקורסוס, נציג אותה כ: e l i n r a e 1 (010)

שלב ב': מיזוג זוגות תכופים

מוצאים את זוג התווים הכי תכוף (נניח ”re”) ומזגים אותו לTOKEN יחיד: g n i n r a e 1 g n i n r a e 1

שלב ג': חזרה

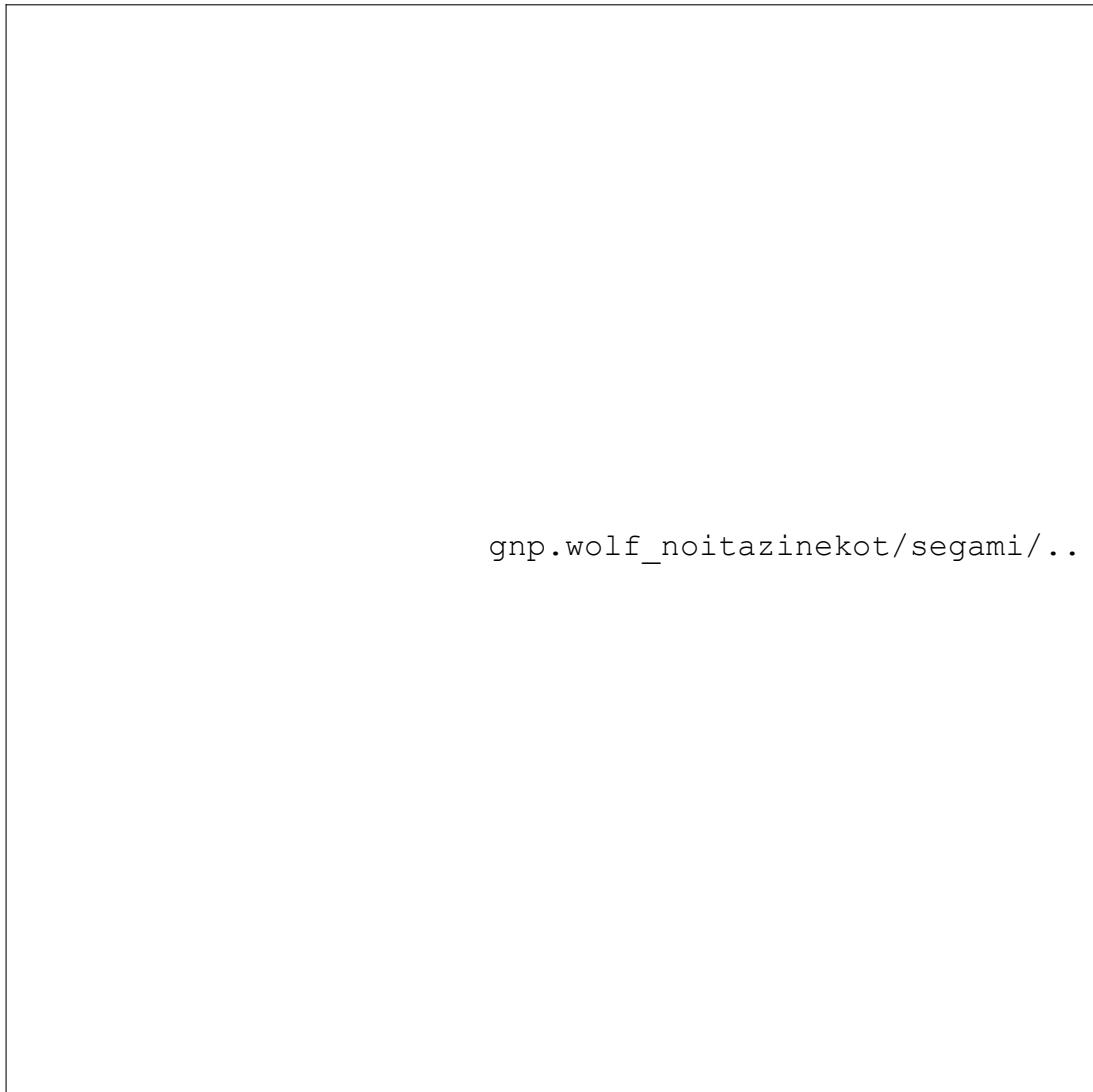
ממשיכים את התהליך: מוצאים את הזוג הכי תכוף הבא, ממזגים, וחזור חלילה. אחרי מספר איטרציות, נקבל משהו כמו: e g n i n r a e 1 (nekot elgnis a sa)

התוצאה: אוצר מילים בגודל סביר (בדרך כלל 30000–50000 TOKENים) שמכיל גם מילים שלמות נפוצות וגם חלקים שנייה לשלב לייצוג מילים חדשות. זו הסיבה ש-2 GPT-31 GPT-3 BPE משתמשים ב-[4]

חלופות ל WordPiece : -BPE 1.2.4 -SentencePiece

אלגוריתמים נוספים מעריכים וריאציות על הרעיון: **WordPiece** (משמש ב-BERT-) [5] דומה לBPE, אבל במקום למזג את הזוג הכי תכוף, הוא בוחן את הזוג שמשפר הכי הרבה את הסבירות של הקורפוס על פי מודל שפה. זה אופטימיזיה קצר יותר מתוחכמת.

SentencePiece (משמש בBERT ו-LLM) [6] מתייחס לטקסט כזרם גולמי של Unicode, כולל רוחים. זה הופך אותו לבלתי-תלוי בשפה - הוא עובד באמצעות מידת טוביה על עברית, אנגלית, סינית, או כל שפה אחרת, ללא צורך בעיבוד מוקדם ספציפי לשפה. כפי שמציג איור 1, תהליך הטוקניזציה הוא הצד הקרייתי הראשון שהופך טקסט אנושי למשהו שמודל יכול לעבד.



איור 1: תחיליך הטוקניזציה: המרת משפט לרצף טוקנים

איור 1 ממחיש את הזרימה המלאה מטקסט מקורית לTOKENים. שימו לב כיצד הTOKENים מפצל את "חתול" ל"ה" + "חתול" - זיהוי אינטיגנטי של תחילית נפוצה שחושך מקום באותן המילים.

1.3 שכבות Embedding: מטוקנים לווקטורים

כעת, אחרי שפירקנו את הטקסט לטוקנים, אנחנו מתמודדים עם שאלה חדשה: איך נציג 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 $d = 512$:

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

למה 512 ממדים? זו החלטת עיצוב Transformer המקורי [1] השתמש ב512 אבל מודלים מודרניים כמו GPT-3 משתמשים ב12288 ממדים d למודל הגדל יותר. הרעיון המרכזי: **מילים עם שימושות דומות מקבלו וקטורים קרובים במרחב הווקטורי**. זה אומר שהמרחב (למשל, מרחק אוקלידי או קוסינוס) בין gniddebme ("כלב") יהיה קטן יחסית, כי שתי המילים מתיחסות לחיות בית. לעומת זאת, המרחק בין gniddebme ("חתול") ל- gniddebme ("מחשב") יהיה גדול יותר.

1.3.2 מטריצת Embedding-: הטבלה המרכזית

בפועל, כל הטוקנים של embeddings 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 = ממד embedding - למשל 512

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

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

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

1.3.3 איך לומדים את Embeddings?

קיימות שתי גישות עיקריות:

גישה 1: Embeddings נלמדים מאפס

המודל מתחילה עם אקרים embeddings ולומד אותם במהלך האימון, באמצעות back-propagation. זו הגישה הנפוצה במודלי Transformer מודרניים. embeddings מותאמים ספציפית למשימה - למשל, תרגום, סיוכום, או ייצרת טקסט.

גישה 2: Embeddings מאומנים מראש

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

1.3.4 ויזואлизציה של מרחב Embedding

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



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

באיור 2 ניתן לראות כיצד מילים מסוימת קטגוריה סמנטית (חיות, טכנולוגיה, פעלים) מתבקשות יחדיו במרחב. זו הדוגמה היפה לכך של embeddings - הם לומדים לייצג משמעות באופן אוטומטי, שירות מהדעתה.

1.4 קידוד מיקום: הוספת המימד הרביעי - זמן

עכשו הגענו לנקודה קריטית. יש לנו טוקנים, יש לנו embeddings שמייצגים את המשמעות של כל טוקן. אבל... חסר לנו מהותי סדר.

1.4.1 הבעיה: Transformers אינם מודעים לסדר

להבדיל ממודלים רקורסיביים כמו LSTM, שמעבדים טוקנים אחד אחרי השני ולכן "יודעים" מה בא לפני מה, ארכיטקטורת Transformer מסתכלת על כל הטוקנים **במקביל**. זה נותן לה יתרון עצום במדויקות 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 בראץ'**.

1.4.2 הפתרון: קידוד מיקום Sinusoidal

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

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

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

כasher:

- pos = מיקום הטוקן בראץ' ($0, 1, 2, \dots, 3$) -

- i = אינדקס המימד בווקטור ($0, 1, 2, \dots, d/2 - 1$) -

- d = מימד ההספטי embedding (למשל 215)

נוסחאות 4 ו 5 מגדרות קידוד ייחודי לכל מיקום. לממדים הזוגיים ($0, 2, 4, \dots$) משתמשים בסינוס, ולממדים הא-זוגיים ($1, 3, 5, \dots$) משתמשים בקוסינוס. כל מימד מתנדנד בתדרות שונה - ממדים נמוכים יותר מתנדנדים לאט, וממדים גבויים יותר מהר.

1.4.3 למה סינוס וкосינוס? התובנה המתמטית

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

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

יתרון 3: אקסטרפולציה - המודל יכול להתמודד עם רצפים ארוכים יותר מאשר שראה באימון, כי הfonקציות מוגדרות לכל מקום חיובי.

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

1.4.4 ויזואлизציה של דפוסי קידוד המיקום

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



איור 3: דפוס קידוד המיקום: מפת חום המציגת ערכי PE עבור 50 מיקומים ו-128 ממדים

באיור 3 ניתן לראות בבירור את הדפוס הגלי - ממדים נמוכים יותר (צד שמאל) משתנים לאט על פני מיקומים, בעוד ממדים גבוהים יותר (צד ימין) משתנים מהר יותר. זה בדוק התכוונה שמאפשרת למודל ללמידה גם מרחקים קצרים וגם מרחקים ארוכים בין מילימטרים.

הפניה בספר הנלווה: לפרטים מתקיפים על קידוד מיקום sin/cos, כולל הצגה דו-ממדית, סקלינג לא-ממדים, הוכחת ייחודיות, יכולת אינטראקטיבית, 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 הקלاسي, קיימות גישות חלופיות: Learned Positional Embeddings (משמש בBERT) [5] - מקום מסוימת קבועה, המודל לומד טבלה של וקטורי מיקום במהלך האימון. יתרון: גמישות. חיסרון: לא ניתן לאקסטרפלציה לאורכים לא-נראים.

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

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

1.5 הייצוג הסופי: חיבור Embedding וקידוד מיקום

עכשו מגיע השלב האחרון: איחוד embedding של הטוקן עם קידוד המיקום שלו.

1.5.1 פועלות החיבור הפשוטה

בארכיטקטורת Transformer המקורי, הפתרון פשוט להפעיל: **חיבור וקטורי רגיל**:

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

כאשר:

- $x_{embed}[pos]$ = הembedding של הטוקן במיקום pos

- $PE[pos]$ = וקטור קידוד המיקום

- $x_{final}[pos]$ = הייצוג הסופי של הטוקן, שմשלב גם משמעות ו גם מיקום

נוסחה 6 מגדרה את הקלט הסופי למודל Transformer. כל טוקן עכשו מיוצג על ידי וקטור שמכיל גם את המשמעות הסמנטית שלו (embedding) וגם את המיקום שלו ברכז (קידוד 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 מידע הווקטור ומאפשר למודל ללמידה איך "לערबב" את שני סוגי המידע בצורה אופטימלית.

1.5.2 נורמליזציה ושיקולים נוספים

במודלים מודרניים, לעיתים מוסיפים גם צעדים נוספים:

סקלינג - מכפילים את embedding ב- $\sqrt{d_{model}}$ לפני החיבור עם קידוד המיקום [1]. זה נועד לאזן בין שני הרכיבים ולמנוע מקידוד המיקום "לבלו" את embedding .

Dropout - מוסיפים רעש אקראי כל (מאפסים חלק מהערבים) כדי למנוע overfitting ולשפר את יכולת ההכללה של המודל.

התוצאה הסופית היא רצף של וקטוריים $[x_{final}[0], x_{final}[1], \dots, x_{final}[n-1]]$, כאשר כל וקטור מייצג טוקן אחד עם המידע המלא שלו: מה הוא ואיפה הוא.

1.6 סיכום: מטකסט לייצוג מתמטי מלא

באו נעקוב שוב אחורי המסע המלא, מהתחלת ועד הסוף:

שלב 1: טוקניזציה

"החתול ישן על הספה" \rightarrow ["הפט", "ה", "לע", "ושי", "לוטח", "ה"]

שלב 2: המרת מילים

[5, 1523, 892, 71, 5, 2103] \rightarrow ["הפט", "ה", "לע", "ושי", "לוטח", "ה"]

שלב 3: Embedding Lookup

כל מילה \rightarrow וקטור ב- \mathbb{R}^{512}

שלב 4: קידוד מיקום

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

שלב 5: חיבור

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

התוצאה: מטריצה $X \in \mathbb{R}^{6 \times 512}$ (6 טוקנים \rightarrow 512 ממדים) שמכילה את כל המידע הדרוש למודל Transformer כדי להבין את המשפט - גם מה כל מילה אומרת, וגם באיזה סדר.

הצעד הבא: המטריצה הזו תועבר עכשו לשכבות תשומת הלב (attention) של ה-Transformer, שם המודל יחשב את הקשרים בין המילים, בין תלויות סמנטיות, ויצור "יצוגים"前者,后者 עוד יותר. אבל זה כבר נושא לפרק הבא.

יצוג הטקסט - התהליך שעקבנו אחריו בפרק זה - הוא הבסיס לכל מה שהTransformer יכול להציג. בלי טוקניזציה נכונה, embeddings, וקידוד מיקום מדויק, אף אחד מהיכולות המדדיימות של מודלים כמו GPT או BERT לא יהיה 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, *תריצת טרנספורם: מיקם חווינית כראב סוקים דו-מימדיים*, 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, 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.