# 

תוכן

[הצעת פרוייקט – יד הנדסת תוכנה 3](#_Toc103942058)

[מבוא 6](#_Toc103942059)

[הרקע לפרויקט: 6](#_Toc103942060)

[תהליך המחקר 6](#_Toc103942061)

[סקירת שפות 6](#_Toc103942062)

[מטרות ויעדים 7](#_Toc103942063)

[מטרות: 7](#_Toc103942064)

[יעדים: 7](#_Toc103942065)

[תיאור ה-UC העיקריים של המערכת 8](#_Toc103942066)

[רשימת Use cases: 8](#_Toc103942067)

[תיאור ה-UC העיקריים של המערכת 9](#_Toc103942068)

[מבני נתונים בהם משתמשים בפרויקט 10](#_Toc103942069)

[תרשים מחלקות 10](#_Toc103942070)

[תיאור המחלקות המוצעות 10](#_Toc103942071)

[אלגוריתם מרכזי 11](#_Toc103942072)

[הטמעות אישיות: איך זה עובד? 11](#_Toc103942073)

[הטבעות מילים 15](#_Toc103942074)

[אנלוגיות 16](#_Toc103942075)

[מידול שפה 17](#_Toc103942076)

[הדרכה במודל שפה 20](#_Toc103942077)

[הסתכלות בשני הכיוונים 22](#_Toc103942078)

[- Skipgram דילוג גרם 22](#_Toc103942079)

[בחינה מחודשת של תהליך ההכשרה 25](#_Toc103942080)

[דגימה שלילית 27](#_Toc103942081)

[Skipgram עם דגימה שלילית (SGNS) 31](#_Toc103942082)

[תהליך ההדרכה של Word2vec 31](#_Toc103942083)

[גודל מסגרת ומספר דוגמאות שליליות 34](#_Toc103942084)

[למעשה 35](#_Toc103942085)

[פונקציות עיקריות המשתמשות במודל: 36](#_Toc103942086)

[פונקצית guess 36](#_Toc103942087)

[פונקצית give\_clue 37](#_Toc103942088)

# הצעת פרוייקט – יד הנדסת תוכנה

סמל מוסד: 189084

שם מכללה : בנות אלישבע

שם הסטודנט : הדסה שפירא

ת"ז הסטודנט : 212318026

שם הפרויקט:

**משחק – "שם קוד"**

## תיאור הפרויקט:

משחק "שם קוד" ממוחשב בין כמה שחקנים או מול מחשב.

## תיאור המשחק:

המשחק מתנהל בין שתי קבוצות יריבות של שחקנים. בכל קבוצה אחד השחקנים נבחר להיות רב-המרגלים והוא השחקן שייתן רמזים, כלומר שמות-קוד, לשאר חברי הקבוצה שלו שיצטרכו לנחש את כוונתו.

על גבי לוח המשחק מופיעות 25 מילים שונות שמשתנות בכל משחק, כאשר ישנו "קלף סדר" שגלוי רק לרבי-המרגלים של כל קבוצה והוא מסדיר אילו מהמילים שייכות לאילו קבוצה. מתוך 25 המילים, ישנן 9 מילים של קבוצה אחת, והיא הראשונה שתשחק; 8 מילים של הקבוצה האחרת, ומילה אחת שתסומן באיקס שחור ושאותה אסור להגיד מכיוון שניחושה יביא לסיומו של המשחק ולניצחון הקבוצה היריבה.

**מטרת המשחק:** רב-המרגלים של כל קבוצה צריך לגרום לחברי הקבוצה שלו לנחש את כל המילים המסומנות כשלהם ב"קלף הסדר" במספר המועט ביותר של מהלכים, זאת על ידי שימוש בשמות-קוד מקוריים ככל האפשר שיכללו כמה שיותר מילים ששייכות לקבוצה. רב-המרגלים צריך להיזהר במיוחד בשמות הקוד מכיוון שבלבול של חברי הקבוצה יכול להביא לכך שינחשו מילה של הקבוצה האחרת, והנקודה תיזקף לזכותם. כמו כן, יש להיזהר ששם הקוד לא יכווין את חברי הקבוצה לבחור במילה האסורה המסומנת בשחור, כיוון שהדבר יביא להפסד של אותה הקבוצה.

**סיום המשחק:** המשחק נגמר כאשר קבוצה סיימה את כל המילים שהיו מוקצות לה לנחש ומוגדרת כמנצחת, או כאשר קבוצה ניחשה בטעות את המילה האסורה המסומנת בשחור - מה שמוביל לניצחון מיידי של הקבוצה היריבה.

## הגדרת הבעיה האלגוריתמית:

מציאת "הרמז", המילה הקשורה ביותר למספר רב של פרמטרים.

## רקע תיאורטי בתחום הפרוייקט:

הטמנת מילים עוסקת ביצירת קשרים בין מילים שונות. הפעולה תתבצע באמצעות אלגוריתם word2vec מציאת המילים בלוח המשחק ע"י ה"רמז" שהתקבל.

## תהליכים עיקריים בפרויקט:

* בנית מודל word2vec.
* מציאת "רמז", מילה משותפת, למילים הנדרשות בלוח המשחק.
* מציאת המילים בלוח המשחק ע"י ה"רמז" שהתקבל.
* ניהול משחק

## תיאור הטכנולוגיה:

שפת תכנות בצד השרת : python

שפת תכנות בצד הלקוח: react

## מסד נתונים:

Word2vec

מילים שונות

## פרוטוקולי תקשורת:

Http

## לוחות זמנים:

אוקטובר – גיבוש הרעיון לפרויקט.

נובמבר – לימוד האלגוריתם.

דצמבר-ינואר – כתיבת האלגוריתם.

פברואר – ניסוי ובדיקות.

מרץ – בניית ממשק ופונקציות נוספות.

אפריל – ספר פרויקט.

מאי – הגשת פרויקט

חתימת הסטודנט : דסי שפירא

חתימת רכז המגמה :

אישור משרד החינוך :

# מבוא

## הרקע לפרויקט:

בעולם המודרני, גם השמיים הם לא הגבול. הדהירה הבלתי פוסקת אל הקידמה כובשת שיאים

חדשים בהרחבת יכולת הפיתוח, ורבות מפעולות היומיום שלנו נעשות באמצעות מחשב.

בתוך כל המרוץ הזה חיפשתי פרויקט שגם יאתגר אותי ובנוסף יביא למימוש חלום קטן שיש לי, המשחק "שם קוד", משחק שאהוב עלי מאוד.

כל נושא המילים הוא כים שהסוף איננו נראה, ומבחינתי זה היה אבסורד שאין לו משמעות בחיי הקידמה, חייב להיות מצב שגם מחשב יוכל להבין משמעות של מילה ואת המילים הקשורות אליה.

בפרויקט זה ניסיתי לגלות זאת.

## תהליך המחקר

ישנה עדיפות בולטת ויחידה ליישם את הפרויקט בעזרת AI (בינה מלאכותית) משום שתחום זה הוא התחום המתקדם ביותר בעולם המחשוב. פיתוח בתחום הבינה המלאכותית מהווה את פיתוח המחר של הטכנולוגיה, ופריצת כמה צעדים קדימה בעולם ההייטק.

בשנים האחרונות התפתח התחום בצעדי ענק, על ידי אלגוריתמיקה שמטרתה לדמות את פעולת המוח האנושית ולתת למחשב יכולות עיבוד מידע אנושיות, כביכול. שאיפתו של העולם המודרני היא להגיע למצב בו המחשב יוכל לבצע אף פעולות יצירתיות וכביכול אנושיות- שבן אנוש אינו יכול לבצע בשל מורכבותן, ועל כן נחשב תחום הבינה המלאכותית כנחשק ופורץ דרך.

*והתחלתי לחקור*. גליתי בתוך תחום ה- AI את העולם הענק של המילים. כיום, יש אינסוף ספריות העוסקות בתחום המילים, אחת מהם היא Gensim שעזרה לי להגשים את החלום שלי.

ב- Gensim הכרתי את המודל Word2vec שעוסקת בהטמנת מילים – דבר עוצמתי לעצמו, הרעיון הזה כבש אותי והתחלתי להריץ אותו, בהתחלה אמנתי את המודל בעצמי, אבל היות ו Word2vec חייבת מאגר ענק של טקסט רץ הכרתי את המודלים המאומנים מראש של Word2vec, ובספרית Gensim ישנם פקודות מיוחדות בשביליהם. מודלים אלו מוכרים לרוב ועושים מהם שימושים רבים.

בפרויקט זה, אשתמש ביכולותיו של המחשב ללמוד להבין משמעות של מילה, ונתינת רמז במשחק "שם קוד".

## סקירת שפות

Github, stackOverflow, TensorFlow

# מטרות ויעדים

## מטרות:

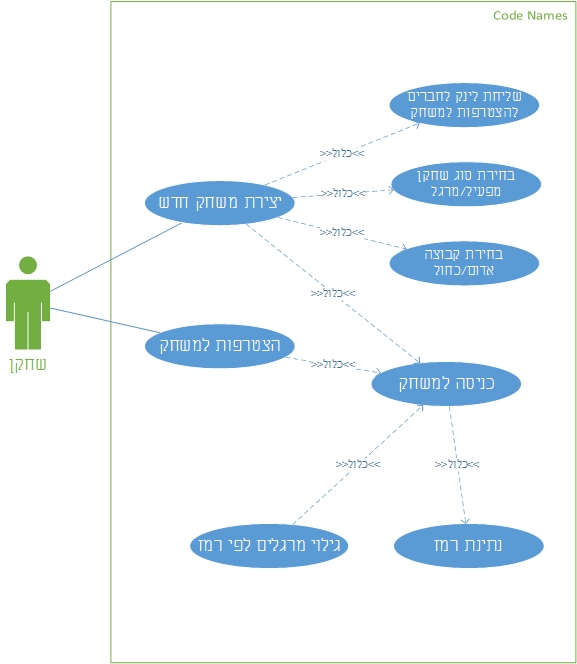
* הטמנת מילים בצורה הטובה ביותר.
* נתינת ה"מילת קוד" היעילה ביותר.
* מציאת המילים המרומזות ע"י ה"מילת קוד" שניתנה.
* אפשרות משחק מהבית עם חברים.
* לימוד והתמקצעות בתחום ה- AI (בינה מלאכותית)
* רכישת מיומנות גבוהה בשפת Python תוך הכרה והתנסות בספריות חדשות ומשמעותיות.
* התנסות בלמידת מכונה.

ממשק נח, נעים וברור למשתמש, תוך הקפדה על נראות בסטנדרטים גבוהים ומקצועיים.

## יעדים:

* הכרת ספרית Genism ומודל word2vec
* תכנון המערכת תוך שימת דגש על אלגוריתם יעיל, כתיבה נכונה, מאורגת ומקצועית של הקוד.
* יצירת ממשק משתמש נח, ברור ומקצועי תוך שמירה על עיצוב ידידותי ונעים לעין.
* שימוש בשפות מתקדמות מעולם הפיתוח.

# תיאור ה-UC העיקריים של המערכת



## רשימת Use cases:

* **יצירת משחק חדש:** השחקן יוצר משחק חדש - מכניס שם, בוחר את סוג השחקן: מפעיל (נותן רמז)/מרגל (מחפש מרגלים-מילים), בחירת קבוצה (אדום/כחול), מקבל לינק לשיתוף חברים ומתחיל המשחק
* **הצטרפות למשחק:** השחקן מקבל לינק ומאשר הצטרפות למשחק
* **כניסה למשחק:** כל אחד בתורו פועל: מפעיל, נותן רמז. מרגל, מאתר מרגלים (מילים דומות) ע"י הרמז ולבסוף ניצחון של אחד הקבוצות
* **נתינת רמז:** כל מפעיל קבוצה בתורו נותן רמז למציאת המרגלים למרגל שלו. נתינת הרמז היא ע"י מציאת מילים דומות מקבוצת המרגלים שלו ושליחת למרגל
* **גילוי מרגלים לפי רמז:** קבלת רמז ממפעיל הקבוצה וניחוש המילים הדומות לרמז – מציאת המרגלים.

## תיאור ה-UC העיקריים של המערכת

**שם:** יצירת משחק חדש.

**מזהה:** 1UC.

**תיאור:** המשתמש יוצר משחק חדש.

**משתמשים:** ישות חיצונית.

**תנאים מוקדמים:** מחשב מחובר לאינטרנט.

**תנאים מאוחרים:** כניסה למשחק.

**UC מוכללים:** שליחת לינק לחברים להצטרפות למשחק, בחירת סוג שחקן, בחירת צבע קבוצה, כניסה למשחק.

**הנחות:** המשתמש יודע את כללי המשחק.

**דרך פעולה בסיסית:** המערכת יוצרת שחקן ומוסיפה אותו לקבוצת השחקנים.

**גרסה:** ראשונה, דסי שפירא.

**שם:** הצטרפות למשחק.

**מזהה:** 2UC.

**תיאור:** המשתמש מצטרף למשחק חדש.

**משתמשים:** ישות חיצונית.

**תנאים מוקדמים:** מחשב מחובר לאינטרנט ולינק הצטרפות.

**תנאים מאוחרים:** כניסה למשחק.

**UC מוכללים:** כניסה למשחק.

**הנחות:** המשתמש יודע את כללי המשחק.

**דרך פעולה בסיסית:** המערכת יוצרת שחקן ומוסיפה אותו לקבוצת השחקנים.

**גרסה:** ראשונה, דסי שפירא**.**

**שם:** כניסה למשחק.

**מזהה:** 2UC.

**תיאור:** התחלת המשחק.

**משתמשים:** השחקנים.

**תנאים מוקדמים:** מינימום של שחקן אחד אנושי.

**תנאים מאוחרים:** ניצחון אחד הקבוצות.

**UC מוכללים:**

**הנחות:** השחקנים יודעים את כללי המשחק.

**דרך פעולה בסיסית:** התנהלות המשחק.

**גרסה:** ראשונה, דסי שפירא**.**

## מבני נתונים בהם משתמשים בפרויקט

מילון: יצירת הרמז

Tuple: בנתינת הרמז

**עץ מודולים**

Client מרגל

React

server

Python

Web api

## תרשים מחלקות

## תיאור המחלקות המוצעות

**צד שרת:**

הוספת שחקן

יצירת לוח

נתינת רמז

קבלת רמז והחזרת מילים דומות לרמז

ניהול תור המשחק

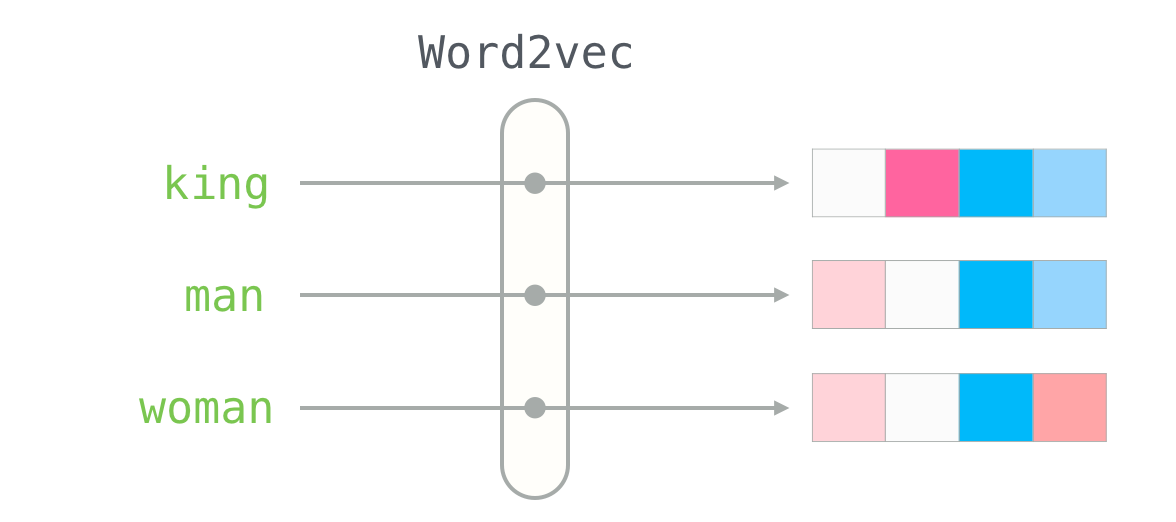
ניצחון

**צד לקוח:**

תצוגת המשחק ותקשורת בין השחקנים

# אלגוריתם מרכזי

בפרויקט הזה השתמשתי במודל מאומן מראש של word2vec בספרית [Gensim](https://radimrehurek.com/gensim/) על הDataBase –GoogleNews" ", כדי להשתמש במודל זה, הייתי צריכה מראש ללמוד את הנושא לעומק והרי הוא לפניכן בקצרה.



הטמעת מילים הוא אחד הרעיונות המרתקים ביותר בלמידת מכונה. אם אי פעם השתמשתם ב-Siri, Google Assistant, Alexa, Google Translate או אפילו במקלדת של סמארטפון עם חיזוי של המילה הבאה, רוב הסיכויים שהפקתם תועלת מהרעיון הזה, שהפך למרכזי במודלים של עיבוד שפה טבעית. בשני העשורים האחרונים חלה התפתחות לא קטנה בשימוש בהטבעות עבור מודלים עצביים (ההתפתחויות האחרונות כוללות הטמעות מילים קונטקסטואליות המובילות למודלים חדשניים כמו BERTו- GPT2).

**Word2vec** היא שיטה ליצירת הטבעות מילים ביעילות והיא קיימת מאז 2013. אבל בנוסף לשירות שלה כשיטה להטמעת מילים, חלק מהמושגים שלה הוכחו כיעילים ביצירת מנועי המלצות ובהבנת נתונים רציפים גם במשימות מסחריות, שאינן שפות.

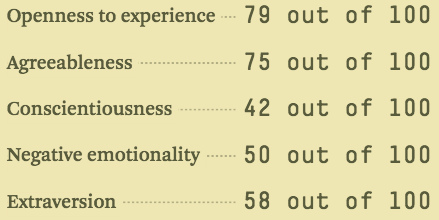
חברות כמו [Airbnb](https://www.kdd.org/kdd2018/accepted-papers/view/real-time-personalization-using-embeddings-for-search-ranking-at-airbnb),   [Alibaba](https://www.kdd.org/kdd2018/accepted-papers/view/billion-scale-commodity-embedding-for-e-commerce-recommendation-in-alibaba) ,[Spotify](https://www.slideshare.net/AndySloane/machine-learning-spotify-madison-big-data-meetup), [Anghami](https://towardsdatascience.com/using-word2vec-for-music-recommendations-bb9649ac2484) נהנו כולן משימוש של המכונה המבריקה הזו מעולם ה-NLP בייצור, כדי להעצים זן חדש של מנועי המלצות.

ועכשיו נעבור על הרעיון של הטמעה, ואת המכניקה של יצירת הטבעות עם word2vec. אבל נתחיל עם דוגמה כדי להכיר את השימוש בווקטורים לייצוג דברים.

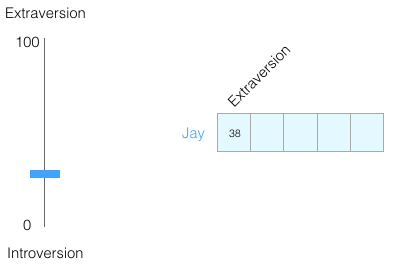
האם ידעת/ה שרשימה של חמישה מספרים (וקטור) יכולה לייצג כל כך הרבה על האישיות שלך?

## הטמעות אישיות: איך זה עובד?

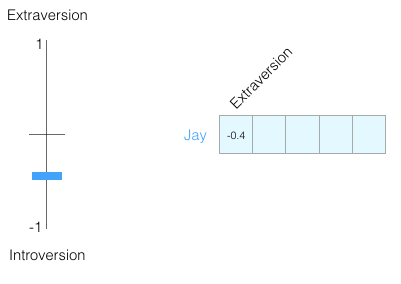
בסולם של 0 עד 100, כמה מופנם / מוחצן את/ה (כאשר 0 הוא המופנם ביותר, ו 100 הוא המוחצן ביותר)? האם אי פעם עשית/ה מבחן אישיות כמו MBTI – או אפילו טוב יותר, מבחן " [Big Five Personality Traits](https://en.wikipedia.org/wiki/Big_Five_personality_traits)"? אם לא, אלה מבחנים ששואלים אותך רשימה של שאלות, ואז מציינים לך על מספר צירים את האישיות שלך, מופנמות / החצנה אחד מהצירים הנ"ל.

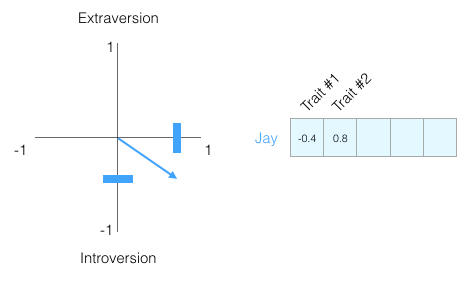
  
דוגמה לתוצאה של מבחן [Big Five Personality Traits](https://en.wikipedia.org/wiki/Big_Five_personality_traits). זה באמת יכול לספר לך הרבה על עצמך והוא הוכיח שיש לו יכולת ניבוי בהצלחה [אקדמית](http://psychology.okstate.edu/faculty/jgrice/psyc4333/FiveFactor_GPAPaper.pdf), [אישית](https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1744-6570.1999.tb00174.x) [ומקצועית](https://www.massgeneral.org/psychiatry/assets/published_papers/soldz-1999.pdf).

תארו לעצמכם שjay קיבל 38/100 כציון ההפנמה/ ההחצנה שלו. אנחנו יכולים לשרטט את זה בדרך זו:



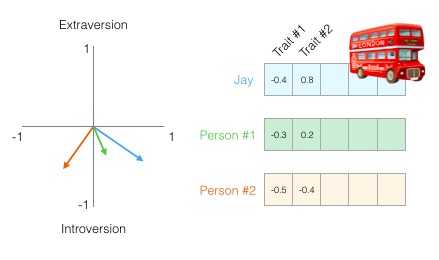
בוא נחליף את הטווח כך שיהיה מ-1 ל-1:



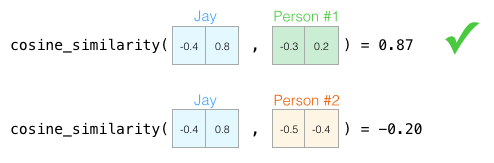
עד כמה את/ה מכיר אדם שיודע רק את פיסת המידע האחת הזו עליו? לא הרבה. אנשים מורכבים. אז בואו נוסיף עוד מימד - ציון של תכונה אחת נוספת מהמבחן.  
אנו יכולים לייצג את שני הממדים כנקודה בגרף, או יותר טוב, כווקטור מהמקור לאותה נקודה. יש לנו כלים מדהימים להתמודד עם וקטורים שיהיו שימושיים בקרוב מאוד.

הסתרתי את התכונות שאנחנו מייצגים, רק כדי שתתרגלו לא לדעת מה כל ממד מייצג – אבל עדיין מקבלים הרבה ערך מהייצוג הווקטורי של אישיותו של האדם.

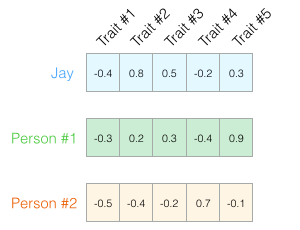
כעת אנו יכולים לומר שהווקטור הזה מייצג חלקית את האישיות של jay. התועלת של ייצוג כזה מגיעה כאשר את/ה רוצה להשוות שני אנשים אחרים לjay. נניח שjay נפגע מאוטובוס וצריך להיות מוחלף על ידי מישהו עם אישיות דומה. באיור הבא, מי משני האנשים דומה יותר לjay?

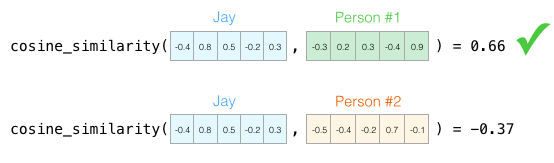


בעת התמודדות עם וקטורים, דרך נפוצה לחשב ציון דמיון היא [cosine\_similarity](https://en.wikipedia.org/wiki/Cosine_similarity):

  
אדם מספר 1 דומה לjay יותר באישיות. הווקטורים מצביעים על אותו כיוון (גם האורך משחק תפקיד) יש ביניהם ציון, דמיון קוסינוס, גבוה יותר.

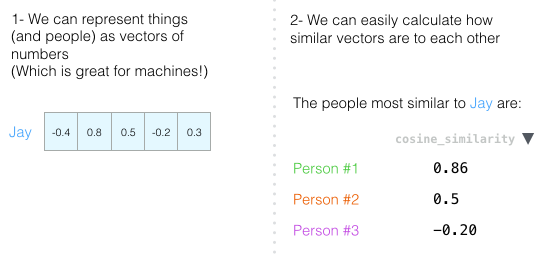
שוב, שני ממדים אינם מספיקים כדי ללכוד מספיק מידע על כמה אנשים שונים. עשרות שנים של מחקר פסיכולוגי הובילו לחמש תכונות עיקריות (והרבה תת-תכונות). אז בואו נשתמש בכל חמשת הממדים בהשוואה שלנו:



הבעיה עם חמישה ממדים היא שאנחנו מאבדים את היכולת לצייר חיצים קטנים ומסודרים בדו מימד. זהו אתגר נפוץ בלמידת מכונה שבה אנחנו צריכים לעתים קרובות לחשוב במרחב ממדי גבוה יותר. עם זאת, הדבר הטוב הוא ש-cosine\_similarity עדיין עובד. זה עובד עם כל מספר של ממדים:  
cosine\_similarity עובד עבור כל מספר של ממדים. אלו ציונים טובים בהרבה, מכיוון שהם מחושבים על סמך ייצוג ברזולוציה גבוהה יותר של הדברים שמשווים.

בסוף החלק הזה, אני רוצה שנצא עם שני רעיונות מרכזיים:

1. אנחנו יכולים לייצג אנשים (ודברים) כוואקטורים של מספרים (וזה נהדר למכונות!).
2. אנו יכולים לחשב בקלות עד כמה וקטורים דומים זה לזה.



## הטבעות מילים

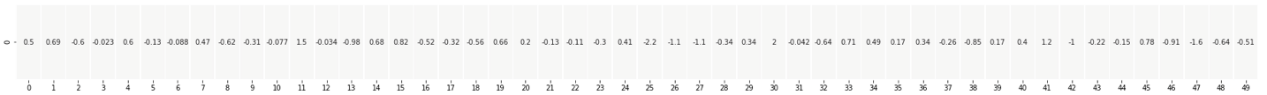
*"מתנת המילים היא מתנת ההונאה והאשליה" ~ ילדי חולית*

עם הבנה זו, אנו יכולים להמשיך להסתכל על דוגמאות מאומנות של מילים וקטוריות (הנקראות גם הטמעות מילים) ולהתחיל להסתכל על כמה מהמאפיינים המעניינים שלהם.

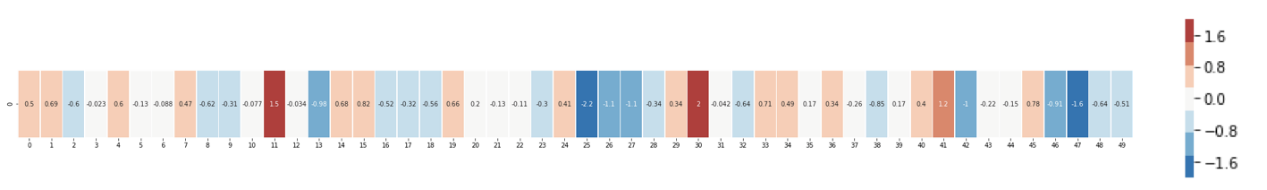
זוהי מילה המטביעה עבור המילה "מלך":

[ 0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 , -0.13498 , -0.08813 , 0.47377 , -0.61798 , -0.31012 , -0.076666, 1.493 , -0.034189, -0.98173 , 0.68229 , 0.81722 , -0.51874 , -0.31503 , -0.55809 , 0.66421 , 0.1961 , -0.13495 , -0.11476 , -0.30344 , 0.41177 , -2.223 , -1.0756 , -1.0783 , -0.34354 , 0.33505 , 1.9927 , -0.04234 , -0.64319 , 0.71125 , 0.49159 , 0.16754 , 0.34344 , -0.25663 , -0.8523 , 0.1661 , 0.40102 , 1.1685 , -1.0137 , -0.21585 , -0.15155 , 0.78321 , -0.91241 , -1.6106 , -0.64426 , -0.51042 ]

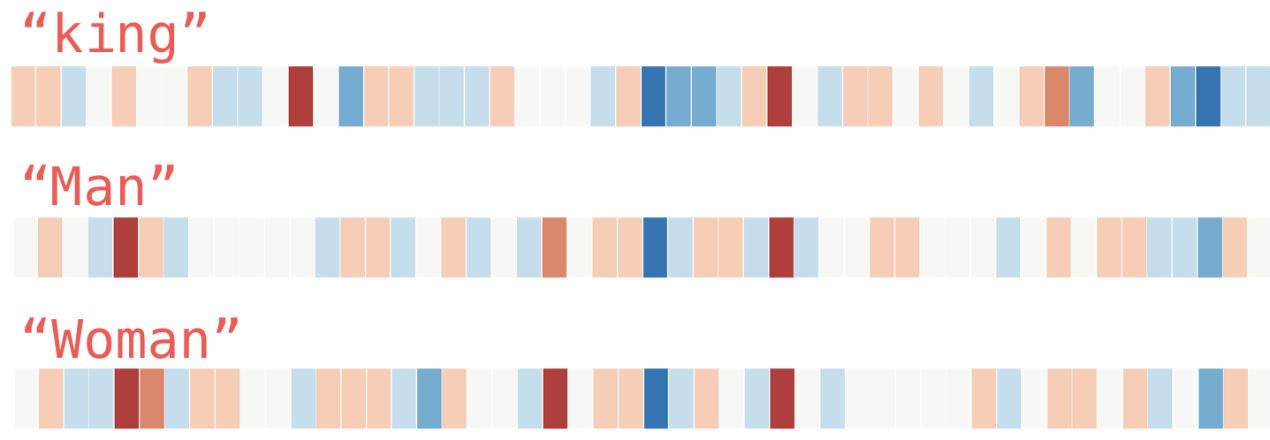
זו רשימה של 50 מספרים. אנחנו לא יודעים הרבה רק על ידי התבוננות בערכים. אבל בואו נדמיין את זה קצת כדי שנוכל להשוות את זה עם וקטורים אחרים של מילים. בואו נשים את כל המספרים האלה בשורה אחת:



בואו נצבע את התאים בהתבסס על הערכים שלהם (אדום אם הם קרובים ל- 2, לבן אם הם קרובים ל- 0, כחול אם הם קרובים ל- 2-):

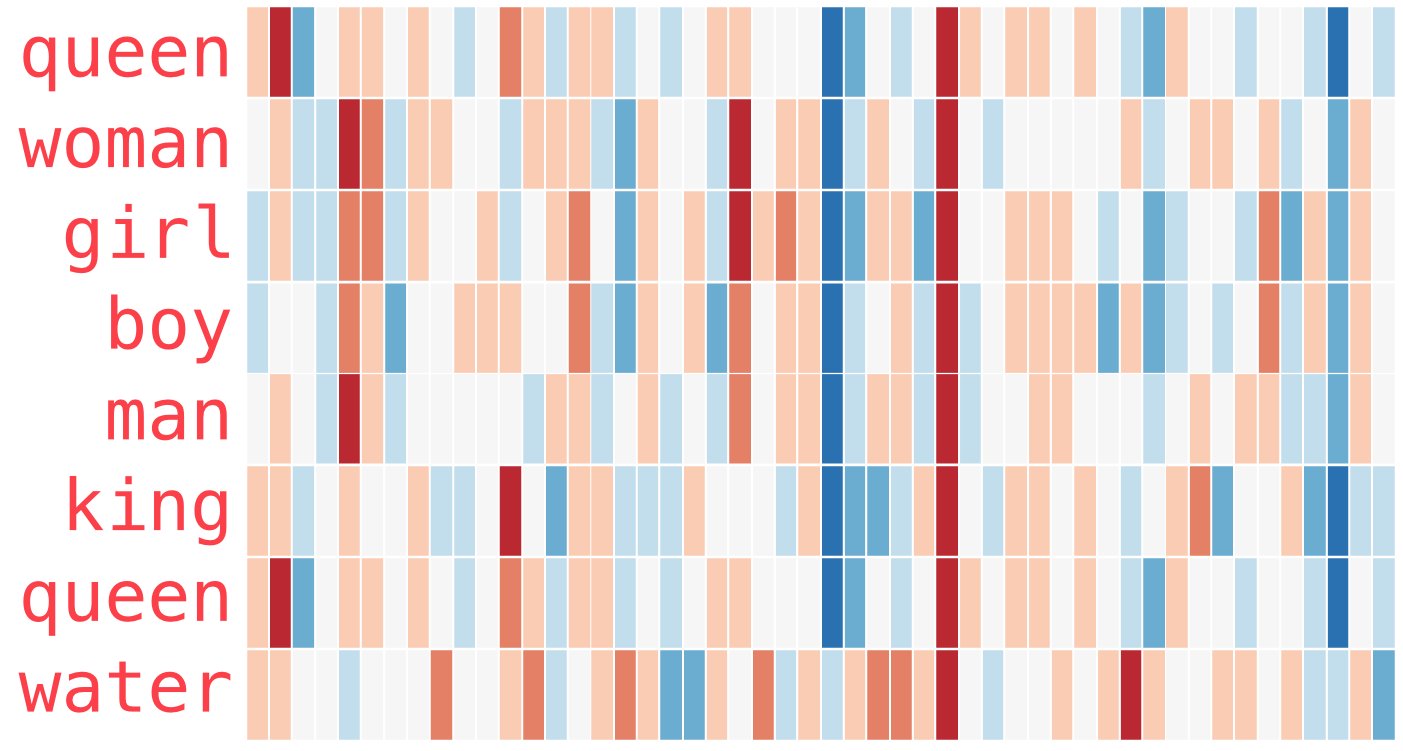


נמשיך על ידי התעלמות מהמספרים והסתכלות רק על הצבעים כדי לציין את ערכי התאים. בואו נשווה עכשיו "מלך" למילים אחרות:



רואים איך "גבר" ו"אישה" דומים הרבה יותר זה לזה מאשר כל אחד מהם עם "מלך"? זה אומר משהו. ייצוגים וקטוריים אלו לוכדים לא מעט מידע/ משמעות/ אסוציאציות של מילים אלה.

להלן רשימה נוספת של דוגמאות (ההשווה על-ידי סריקה אנכית של העמודות המחפשות עמודות עם צבעים דומים):



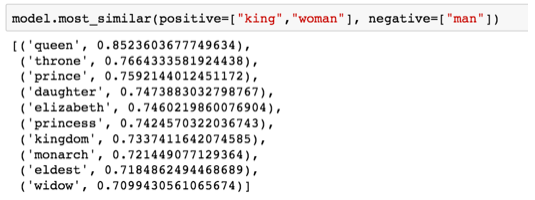
כמה דברים שיש לציין:

1. יש עמודה אדומה ישרה, בכל המילים השונות האלה. הם דומים לאורך הממד הזה (אינו ידוע קידוד של כל מימד)
2. אפשר לראות איך "אישה" ו "ילדה" דומים זה לזה בהרבה מקומות. אותו הדבר עם "גבר" ו "ילד"
3. "ילד" ו "ילדה" יש גם מקומות שבהם הם דומים זה לזה, אבל שונים מ "אישה" או "גבר". האם אלה יכולים להיות קידוד לתפיסה מעורפלת של הנוער? אפשרי.
4. כולם מלבד המילה האחרונה הם מילים המייצגות אנשים. הוספתי אובייקט (מים) כדי להראות את ההבדלים בין קטגוריות. אפשר לראות את הטור הכחול עובר בין כל המילים עד המילה האחרונה "מים".
5. ישנם מקומות ברורים שבהם "מלך" ו "מלכה" דומים זה לזה ונבדלים מכל האחרים. האם אלה יכולים להיות קידוד לתפיסה מעורפלת של מלוכה?

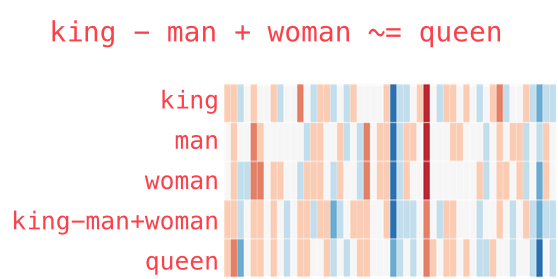
## אנלוגיות

*"מילים יכולות לשאת בכל נטל שנרצה. כל מה שנדרש הוא הסכמה ומסורת שעליה ניתן לבנות." ~ קיסר חולית*

הדוגמאות המפורסמות המציגות מאפיין מדהים של הטמעות הוא הרעיון של אנלוגיות. אנחנו יכולים להוסיף ולהחסיר הטבעות מילים ולהגיע לתוצאות מעניינות. הדוגמה המפורסמת ביותר היא הנוסחה: "מלך" - "גבר" + "אישה":

  
באמצעות ספריית [Gensim](https://radimrehurek.com/gensim/) בפיתון, אנו יכולים להוסיף ולהחסיר וקטורים של מילים, והיא תמצא את המילים הדומות ביותר לווקטור המתקבל. התמונה מציגה רשימה של המילים הדומות ביותר, כל אחת עם הדמיון הקוסינוס שלה.

אנו יכולים לראות אנלוגיה זו כפי שעשינו בעבר:

  
הווקטור המתקבל מ"מלך"-"גבר"+"אישה" לא בדיוק שווה ל"מלכה", אבל "מלכה" היא המילה הקרובה ביותר אליו מ-400,000 הטבעות המילים שיש לנו באוסף הזה.

כעת, לאחר שבדקנו הטמעות מילים מאומנות, בואו נכיר את תהליך ההכשרה שלהן. אבל לפני שנגיע ל- word2vec, אנחנו צריכים להסתכל על המושג "הטמעות מילים" - מודל השפה העצבית.

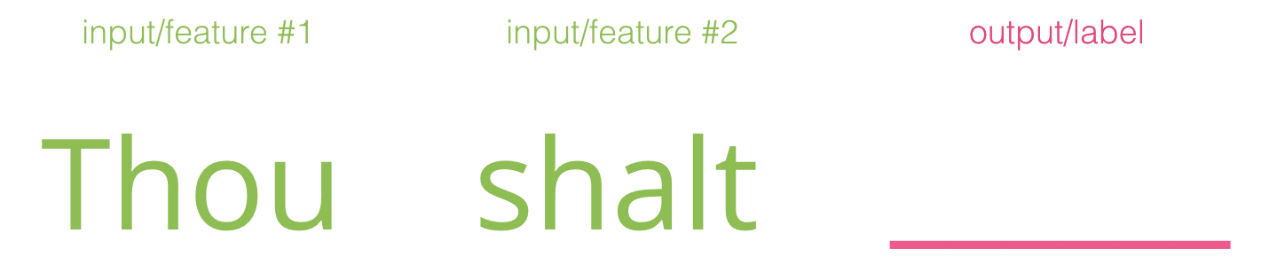
## מידול שפה

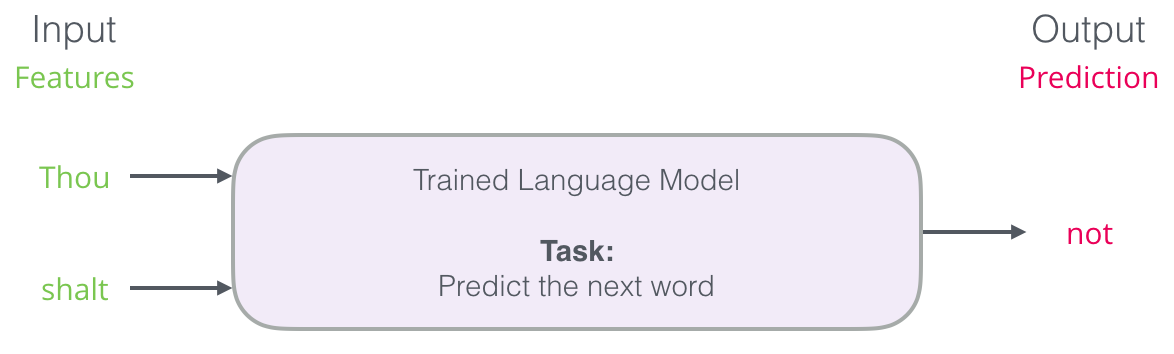
דוגמה ליישום NLP, אחת הדוגמאות הטובות ביותר תהיה תכונת החיזוי של המילה הבאה של מקלדת טלפון חכם. זו תכונה שמיליארדי אנשים משתמשים בה מאות פעמים בכל יום.



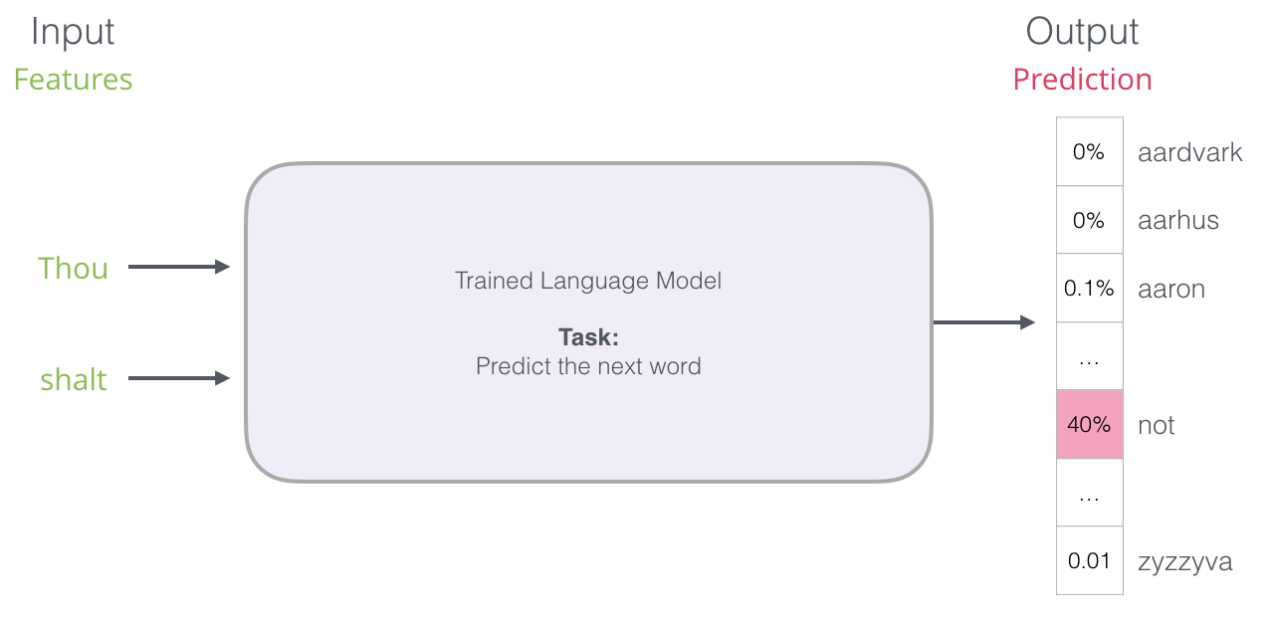
חיזוי המילים הבאות הוא משימה *שמודל שפה* יכול לטפל בה. מודל שפה יכול לקחת רשימה של מילים (נניח שתי מילים) ולנסות לחזות את המילה הבאה אחריהן.

בצילום המסך לעיל, אנו יכולים לחשוב על המודל כעל אחד שלקח את שתי המילים הירוקות האלה (thou shalt) והחזיר רשימה של הצעות ("not" אחד עם ההסתברות הגבוהה ביותר):

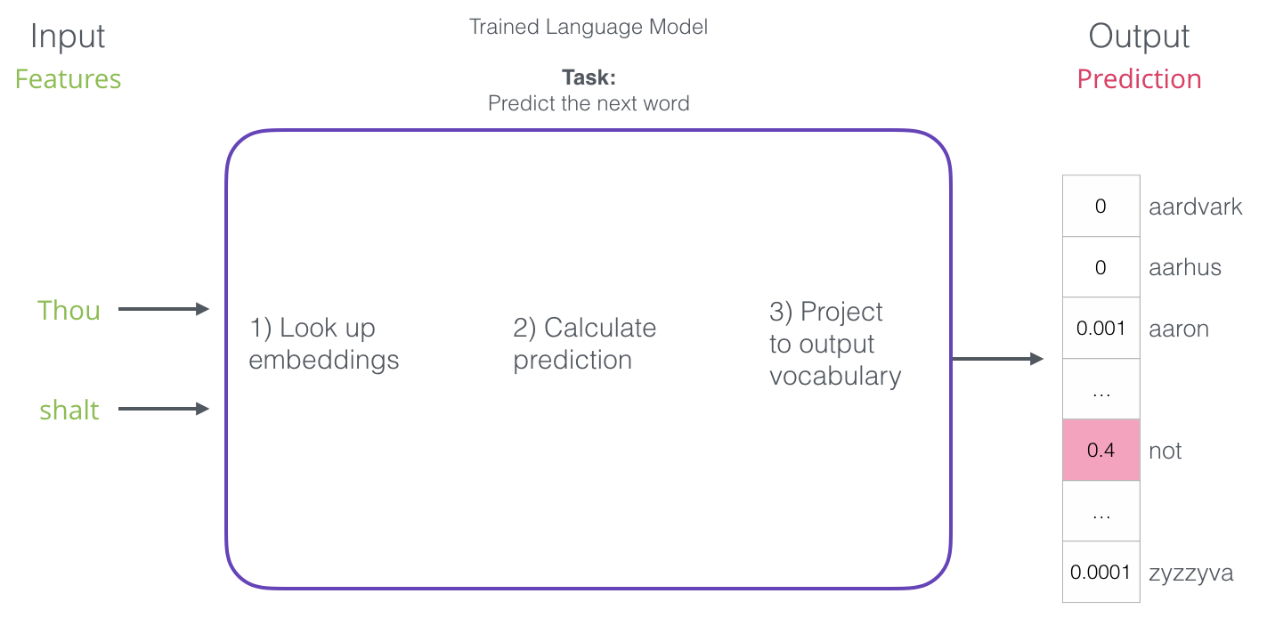
אנחנו יכולים לראות את המודל כמו הקופסה השחורה הזאת:



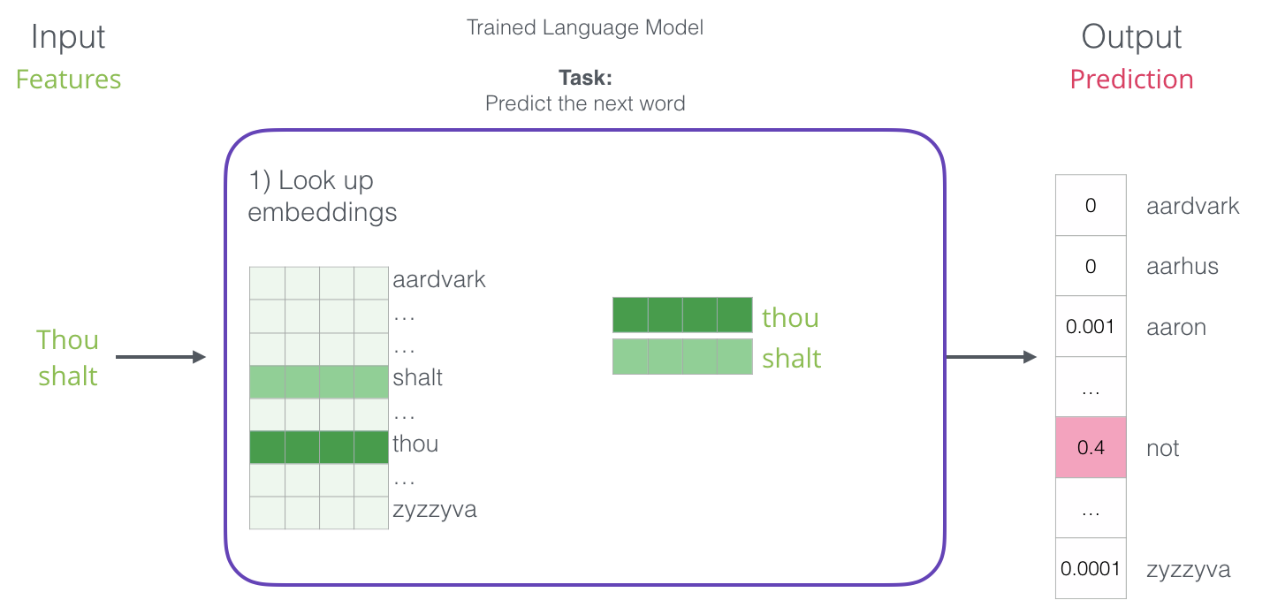
אבל בפועל, המודל לא פלט רק מילה אחת. הוא בעצם פלט ציון הסתברות עבור כל המילים שהוא יודע ("אוצר המילים" של המודל, אשר יכול לנוע בין כמה אלפים ליותר ממיליון מילים). יישום המקלדת צריך למצוא את המילים עם הציונים הגבוהים ביותר, ולהציג אותם למשתמש.

הפלט של מודל השפה העצבית הוא ציון הסתברות עבור כל המילים שהמודל יודע. אנחנו מתייחסים להסתברות כאחוז כאן, אבל 40% יוצגו למעשה כ-0.4 בווקטור הפלט.

לאחר ההכשרה, מודלים מוקדמים של שפה עצבית (Bengio 2003) יחשבו תחזית בשלושה שלבים:



הצעד הראשון הוא הרלוונטי ביותר עבורנו, כאשר אנו דנים בהטבעות. אחת התוצאות של תהליך האימון הייתה מטריצה זו המכילה הטמעה לכל מילה באוצר המילים. במהלך זמן החיזוי, אנו פשוט מחפשים את ההטבעות של מילת הקלט, ומשתמשים בהן כדי לחשב את החזוי:



כעת נפנה לתהליך ההכשרה כדי ללמוד עוד על אופן פיתוח מטריצת ההטבעה הזו.

## הדרכה במודל שפה

*"אי אפשר להבין תהליך על ידי עצירתו. ההבנה חייבת לנוע עם זרימת התהליך, חייבים להצטרף אליו ולזרום איתו." ~ חולית*

למודלים של שפות יש יתרון עצום על פני רוב המודלים האחרים של למידת מכונה. היתרון הוא שהאימון הוא על טקסט רץ – אשר יש לנו בשפע. כל הספרים, המאמרים, תוכן ויקיפדיה וצורות אחרות של נתוני טקסט שיש לנו בסביבה. למול מודלים אחרים של למידת מכונה הזקוקים לתכונות בעבודת יד ונתונים שנאספו במיוחד.

*"אתה תדע מילה על ידי המילה הקודמת או העוקבת שלה" ג'יי.אר פירת'*

מילים מקבלות את ההטבעות שלהן על ידי כך שאנחנו מסתכלים על אילו מילים אחרות הן נוטות להופיע ליד. הטכניקה של זה היא:

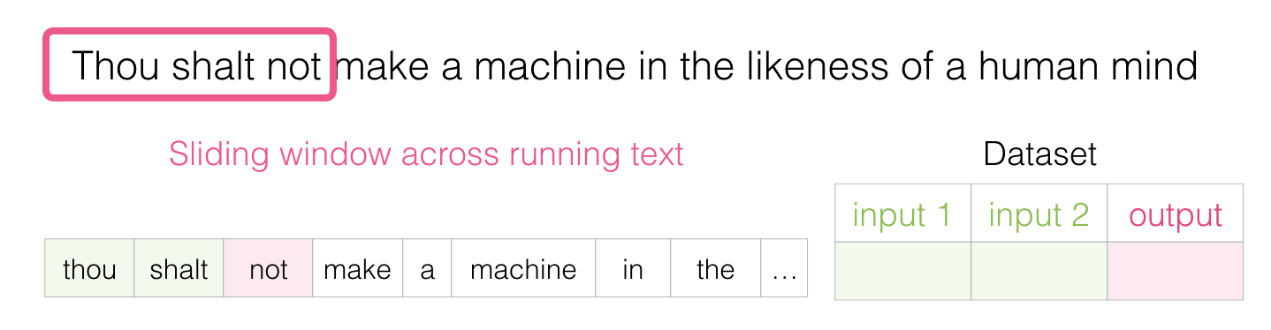
1. מקבלים הרבה נתוני טקסט (למשל, כל המאמרים בוויקיפדיה). ולאחר מכן,
2. יש מסגרת (נגיד, של שלוש מילים) שמועברת על כל הטקסט.
3. מסגרת הזזה זו מייצרת דגימות אימון עבור המודל.

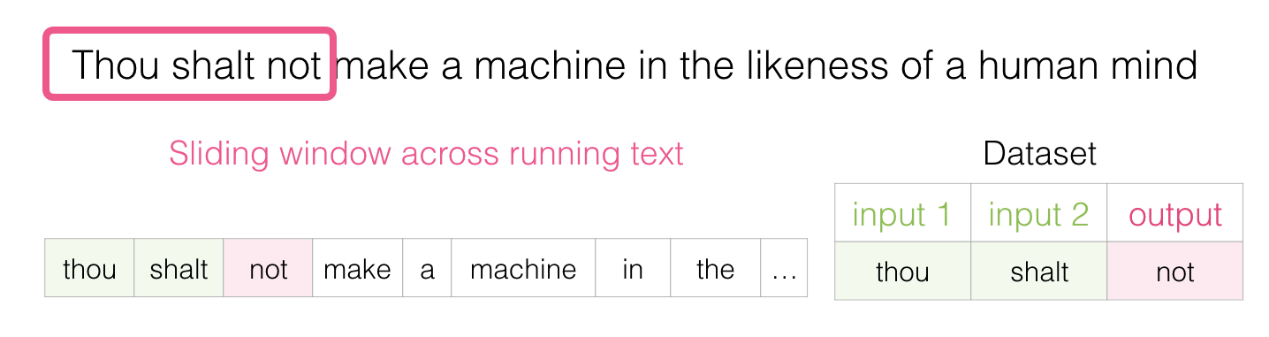


כאשר מסגרת זו מועברת בטקסט, יוצרים ערכת נתונים בה משתמשים כדי לאמן מודל. כדי לבחון בדיוק כיצד זה נעשה, בוא נראה כיצד מסגרת ההזזה מעבדת את הביטוי הזה:

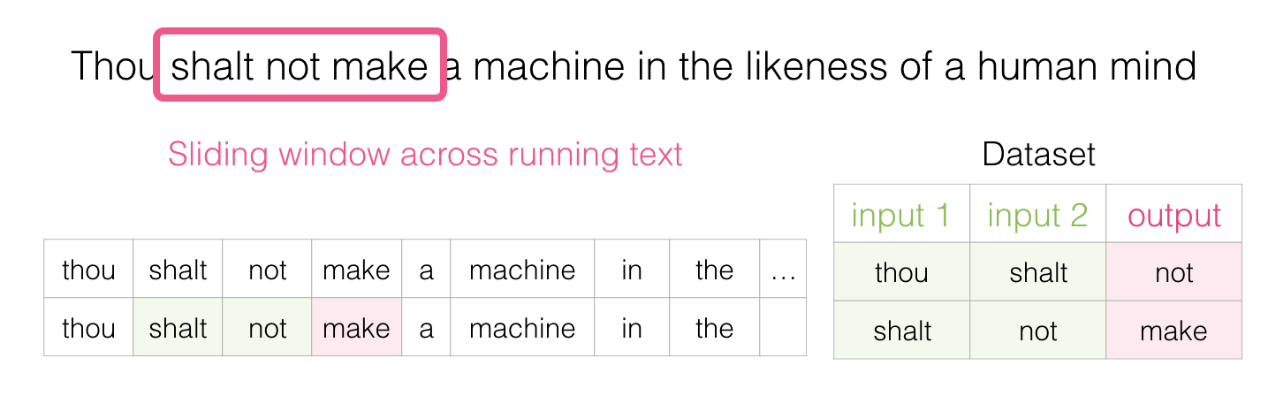
*"אתה לא תעשה מכונה בדמותו של מוח אנושי" ~ חולית*

בהתחלה המסגרת היא על שלוש המילים הראשונות של המשפט:

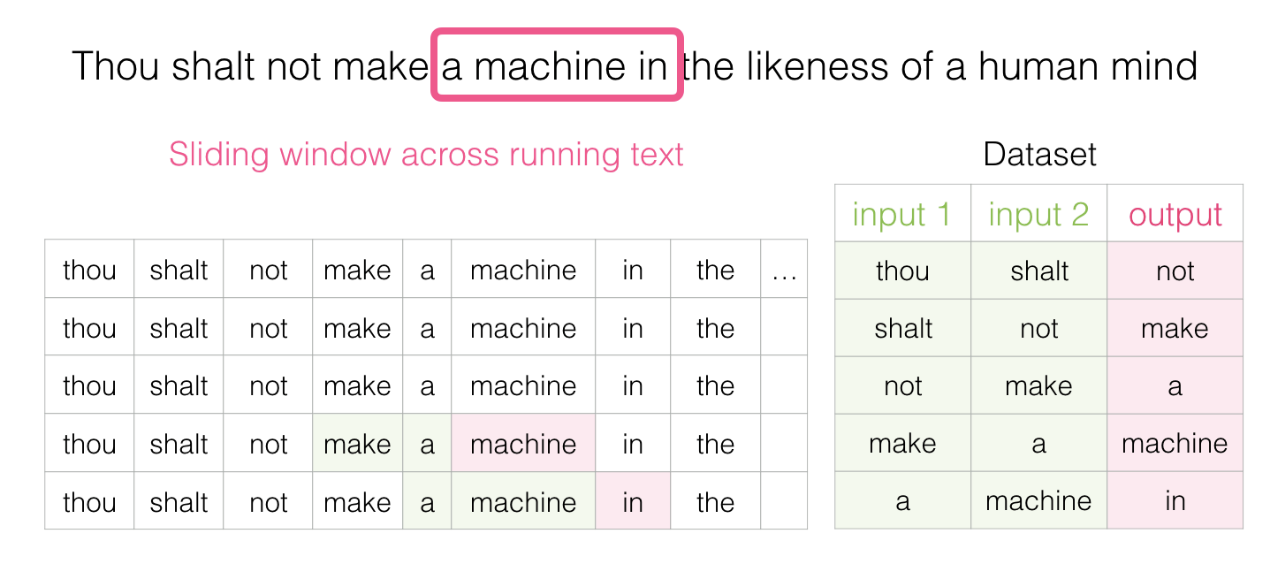
לוקחים את שתי המילים הראשונות להיות התכונות, והמילה השלישית להיות התווית:

  
כעת יצרנו את המדגם הראשון בערכת הנתונים שבה נוכל להשתמש מאוחר יותר כדי לאמן מודל שפה.

לאחר מכן מעבירים את המסגרת למיקום הבא ויוצרים מדגם שני:

  
דוגמה שנייה נוצרת כעת.

ועד מהרה יש ערכת נתונים גדולה יותר של מילים הנוטות להופיע לאחר זוגות שונים של מילים:



## הסתכלות בשני הכיוונים

לפי איך שראינו, מלא את החסר:



המשפט הנתון הוא חמש מילים לפני המילה החסרה (ואזכור מוקדם יותר של "אוטובוס"). אני בטוח שרוב האנשים היו מנחשים שהמילה החסרה היא bus. אבל מה אם היה עוד פיסת מידע אחת – מילה אחרי המילה החסרה, זה ישנה את התשובה?



זה משנה לחלוטין את מה שצריכה להיות המילה החסרה. המילה red היא עכשיו בעלת הסבירות הגבוהה ביותר למילה החסרה. מה שאנו לומדים מכך הוא, המילים לפני ואחרי מילה מסוימת נושאות ערך אינפורמטיבי. מסתבר שחשבונאות בשני הכיוונים (מילים משמאל ומימין למילה שאנחנו מנחשים) מובילה להטמעות מילים טובות יותר. בואו נראה איך אנחנו יכולים להתאים את האופן שבו אנחנו מאמנים את המודל כדי להסביר את זה.

## - Skipgram דילוג גרם

*"המודיעין לוקח סיכון עם נתונים מוגבלים בזירה שבה טעויות הן לא רק אפשריות אלא גם הכרחיות." ~: חולית*

במקום להסתכל רק על שתי מילים לפני מילת היעד, אנחנו יכולים גם להסתכל על שתי מילים אחרי מילת היעד.



אם נעשה זאת, ערכת הנתונים שבונים ומאמנים את המודל ממנה, תיראה כך:

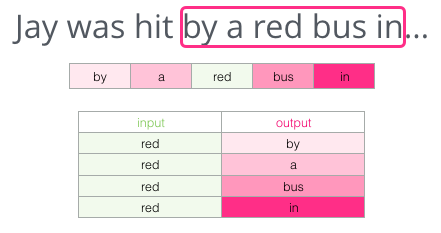


זה נקרא ארכיטקטורת **שק מילים רציף**. ארכיטקטורה נוספת שגם נוטה להראות תוצאות נהדרות עושה את הדברים קצת אחרת.

במקום לנחש מילה המבוססת על ההקשר שלה (המילים לפניה ואחריה), ארכיטקטורה זו מנסה לנחש מילים שכנות באמצעות המילה הנוכחית. אנחנו יכולים לראות זאת על המסגרת שמעבירים על טקסט האימון וזה נראה כך:

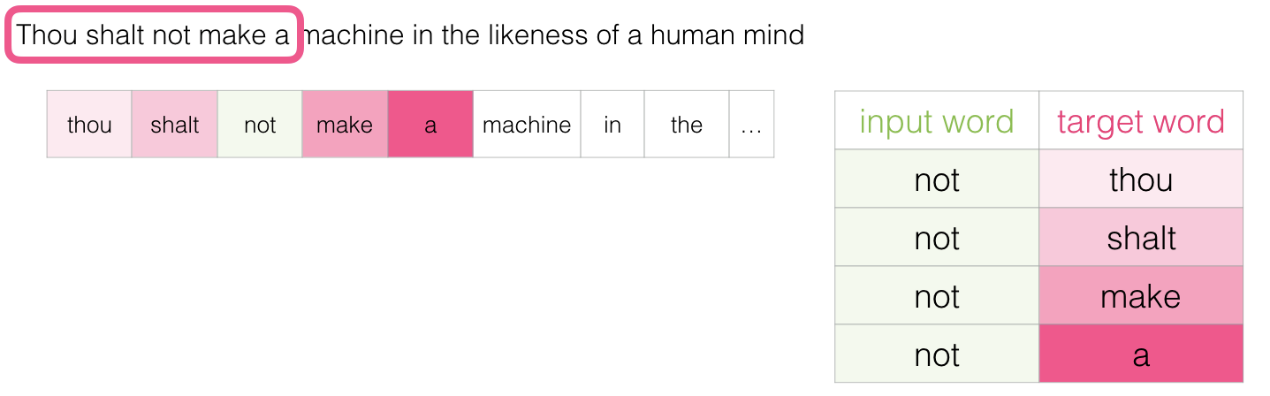
  
המילה בתא הירוק (כרגע לבן) יהיה מילת הקלט, כל תא ורוד יהיה פלט אפשרי.

התאים הוורודים נמצאים בגוונים שונים מכיוון שמסגרת הזזה זו יוצרת למעשה ארבע דוגמאות נפרדות בערכת נתוני האימון:

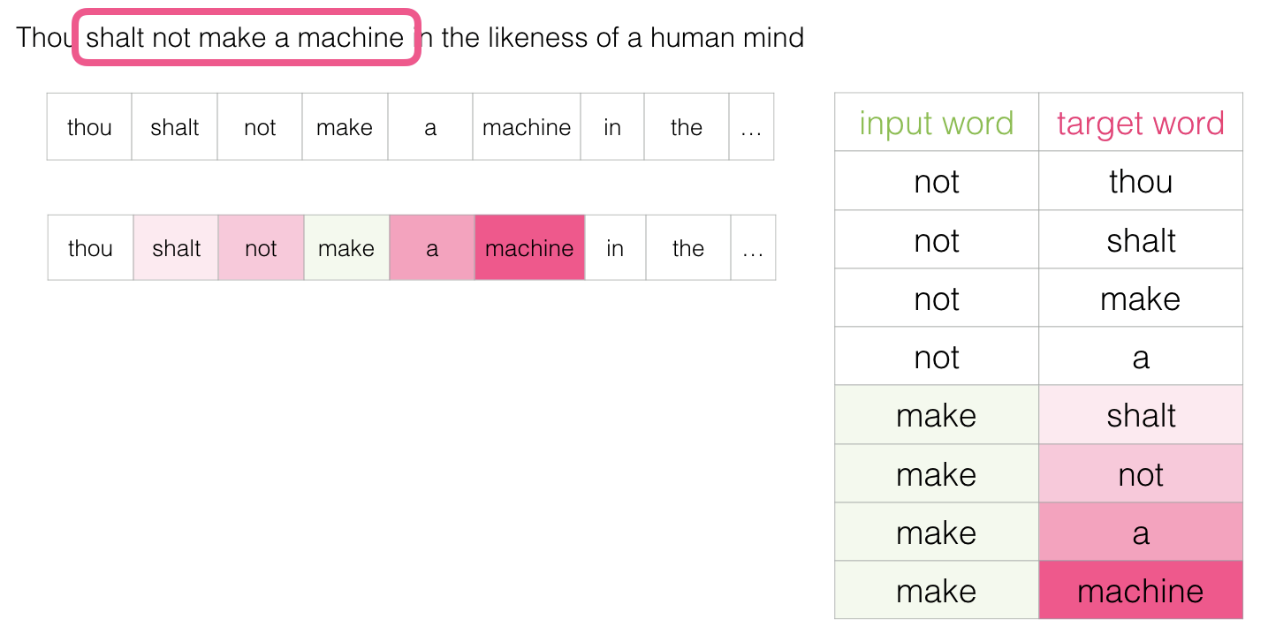


שיטה זו נקראת ארכיטקטורת **הסקילוגרמה**. אנו יכולים לדמיין את מסגרת ההזזה בביצוע הפעולות הבאות:

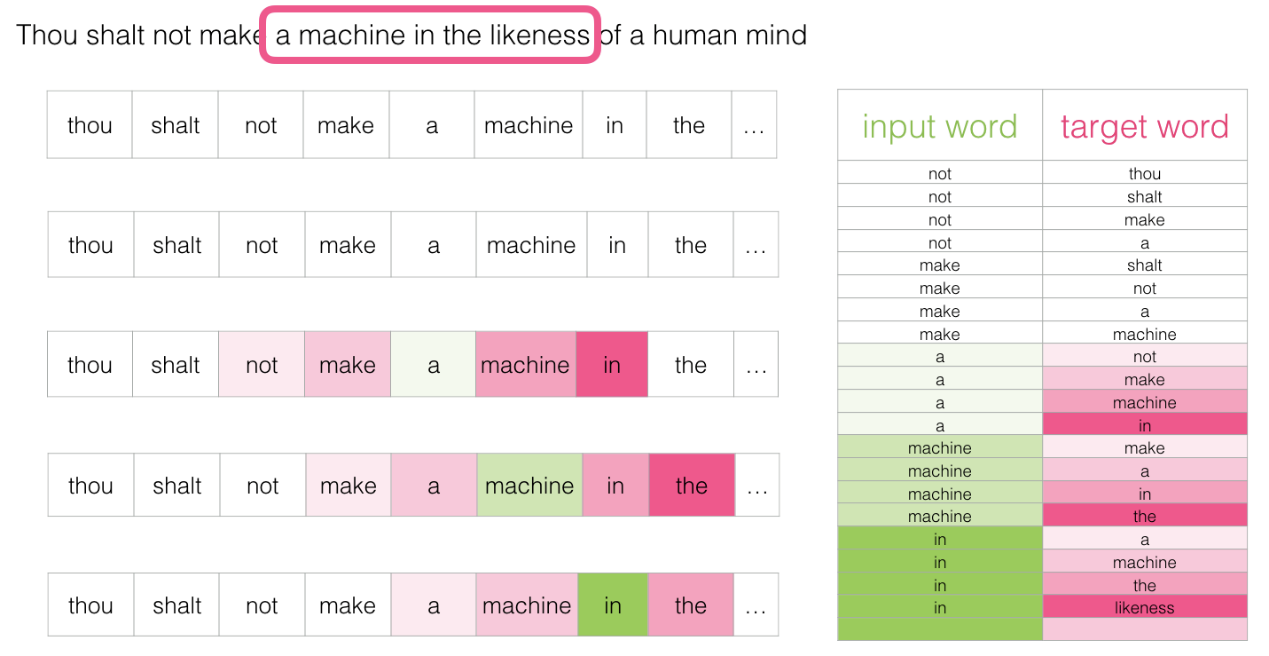
פעולה זו תוסיף את ארבע הדגימות האלה לערכת נתוני האימון שלנו:



לאחר מכן אנו מעבירים את המסגרת שלנו למיקום הבא. מה שמייצר את ארבע הדוגמאות הבאות שלנו:

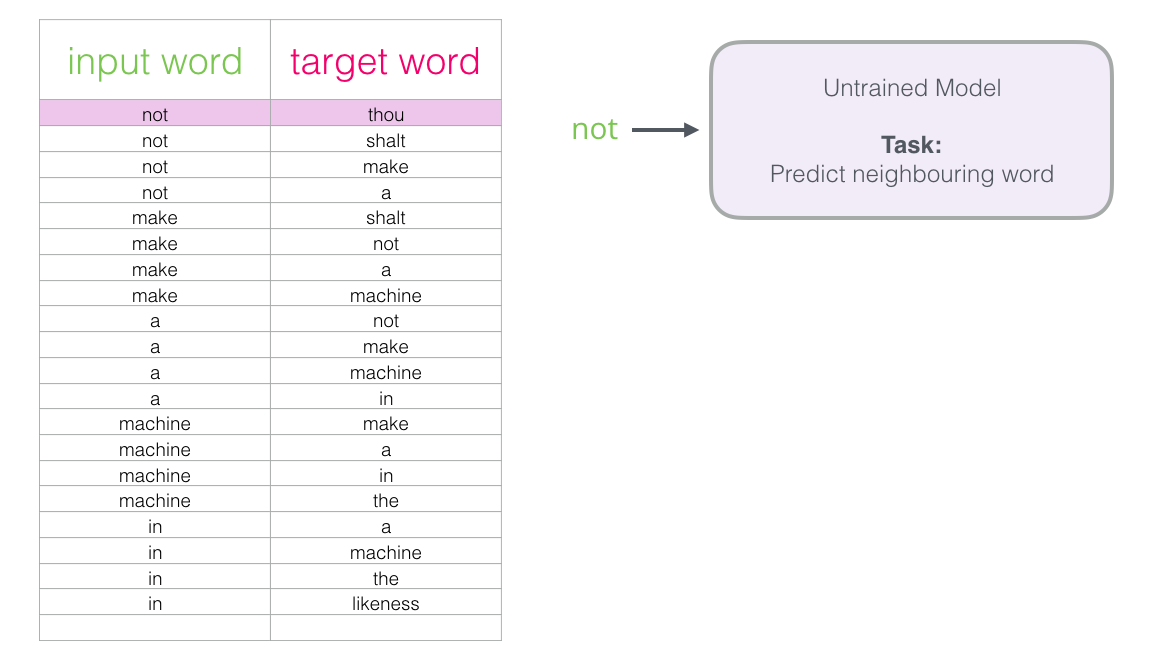


כמה עמדות מאוחר יותר, יש לנו הרבה יותר דוגמאות:

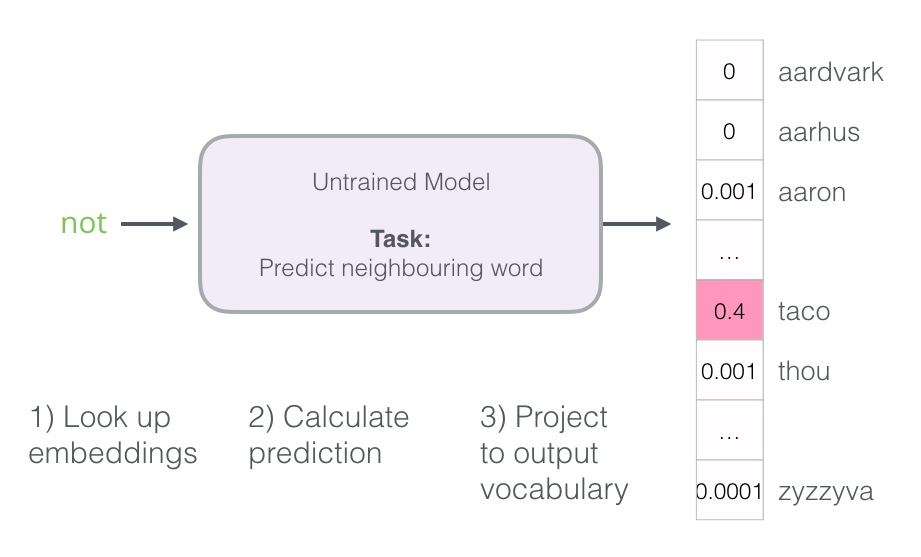


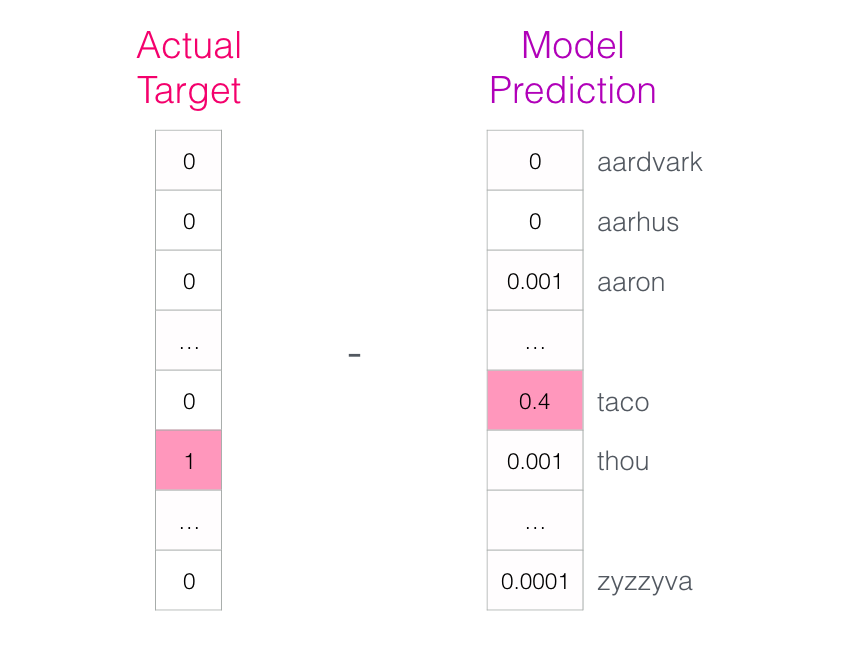
## בחינה מחודשת של תהליך ההכשרה

כעת, לאחר שיש לנו את ערכת נתוני האימון של דילוגרמה שחילצנו מטקסט רץ קיים, בואו נסתכל על האופן שבו אנו משתמשים בו כדי לאמן מודל שפה עצבית בסיסי שחוזה את המילה השכנה.

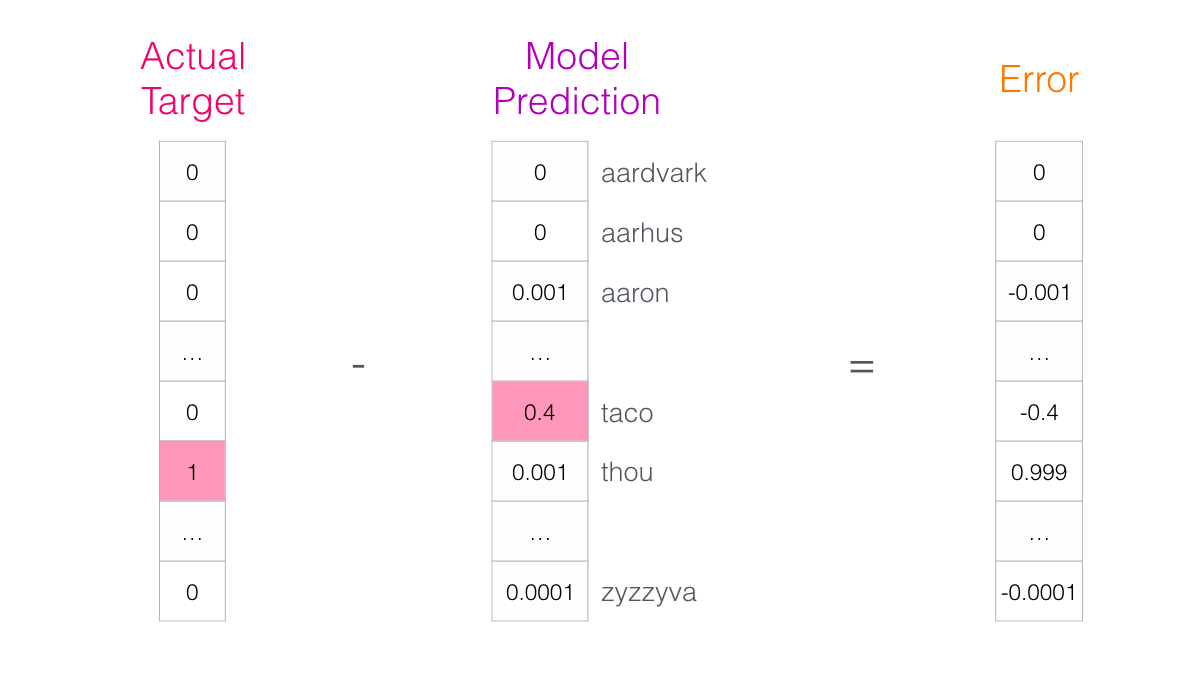


נתחיל עם המדגם הראשון בערכת הנתונים שלנו. אנחנו לוקחים את התכונה ומזינים את המודל הלא מאומן, ומבקשים ממנו לחזות מילה שכנה מתאימה.

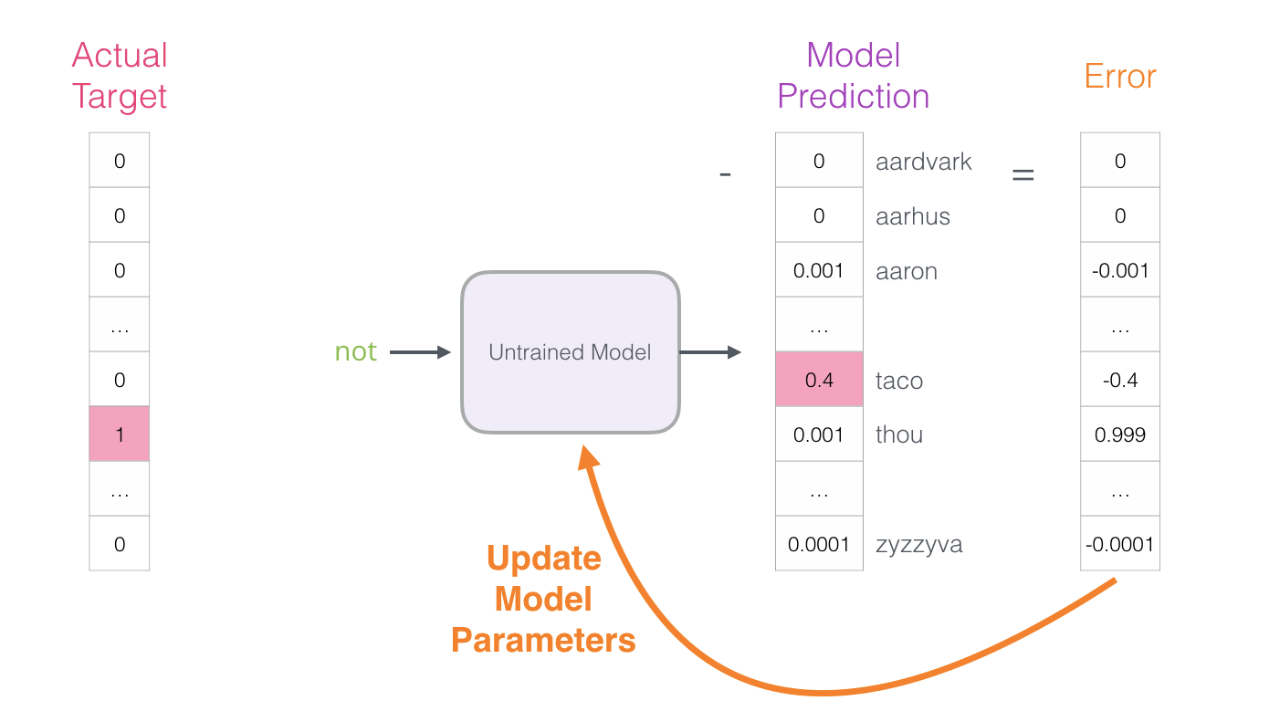
המודל מבצע את שלושת השלבים ומפיק וקטור חיזוי (עם הסתברות המוקצית לכל מילה באוצר המילים שלה). מכיוון שהמודל אינו מאומן, התחזית שלו בטוח תהיה שגויה בשלב זה. אבל זה בסדר. אנחנו יודעים איזו מילה הוא היה צריך לנחש – תא התווית/פלט בשורה שבה אנו משתמשים כרגע כדי לאמן את המודל:

  
'וקטור היעד' הוא וקטור שבו מילת היעד יש את ההסתברות 1, וכל המילים האחרות יש את ההסתברות 0.

כמה רחוק היה המודל? אנו מחסירים את שני הווקטורים וכתוצאה מכך וקטור שגיאה:



כעת ניתן להשתמש בווקטור השגיאה זה כדי לעדכן את המודל כך שבפעם הבאה, סביר יותר להניח שהוא ינחש thou מתי שהוא מקבל not כקלט.



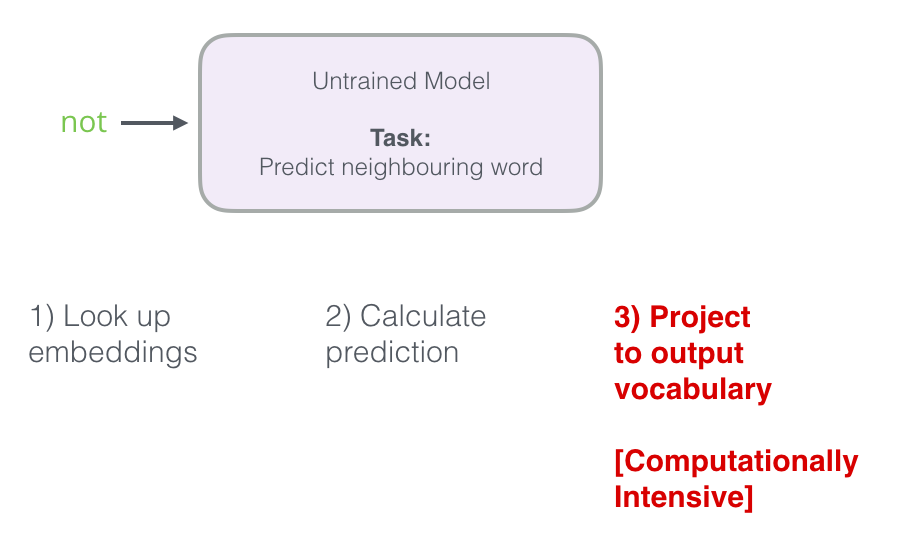
וזה מסכם את הצעד הראשון של האימונים. ממשיכים לעשות את אותו תהליך עם המדגם הבא בערכת הנתונים, וחוזר חלילה, עד שמכסים את כל הדגימות בערכת הנתונים. זה מסכם *תקופה* אחת של אימונים. עושים את זה שוב במשך מספר *תקופות*, ואז המודל מאומן ויכולים לחלץ את מטריצת ההטבעה ממנו ולהשתמש בו לכל יישום אחר.

אמנם זה מרחיב את ההבנה של התהליך, אך זה עדיין לא איך word2vec למעשה מאומן. חסרים לנו כמה רעיונות מרכזיים.

## דגימה שלילית

*"לנסות להבין אדם בלי לדעת את האויבים שלו, זה לנסות לראות את האמת מבלי לדעת שקר. זהו הניסיון לראות את האור מבלי לדעת את החושך. זה לא יכול להיות." ~ חולית*

זכור את שלושת השלבים של האופן שבו מודל שפה עצבית זה מחשב את התחזית שלו:

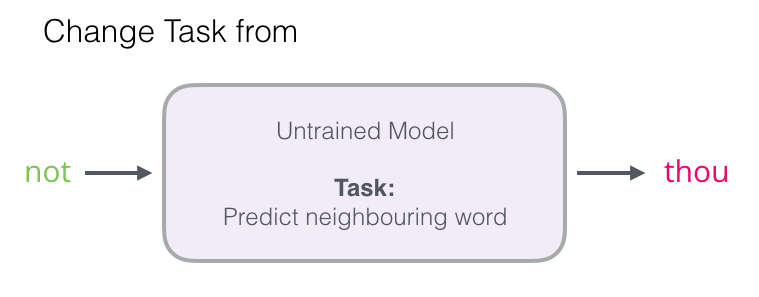


השלב השלישי יקר מאוד מבחינה חישובית – במיוחד בידיעה שהוא נעשה פעם אחת עבור כל מדגם אימון בערכת הנתונים (בקלות עשרות מיליוני פעמים). צריכים לעשות משהו כדי לשפר את הביצועים.

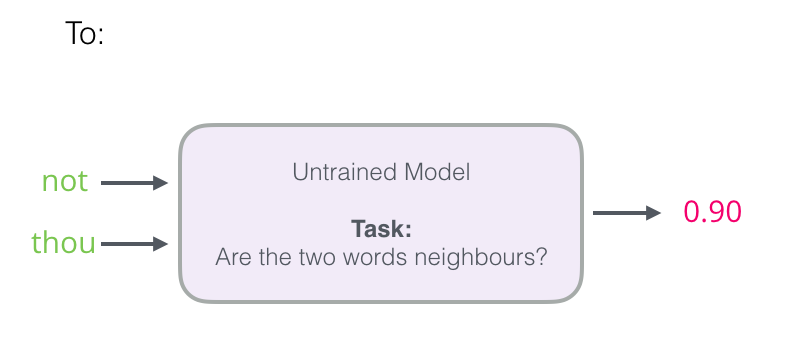
דרך אחת היא לפצל את המטרה לשני שלבים:

1. ליצור הטבעות מילים באיכות גבוהה (ללא דאגה לגבי חיזוי המילים הבאות).
2. שימוש בהטבעות באיכות גבוהה אלה כדי לאמן מודל שפה (ביצוע חיזוי של המילה הבאה).

נתמקד בשלב 1. כאשר אנו מתמקדים בהטבעות. כדי ליצור הטמעות באיכות גבוהה באמצעות מודל בעל ביצועים גבוהים, אנו יכולים להחליף את משימת המודל מחיזוי מילה שכנה:

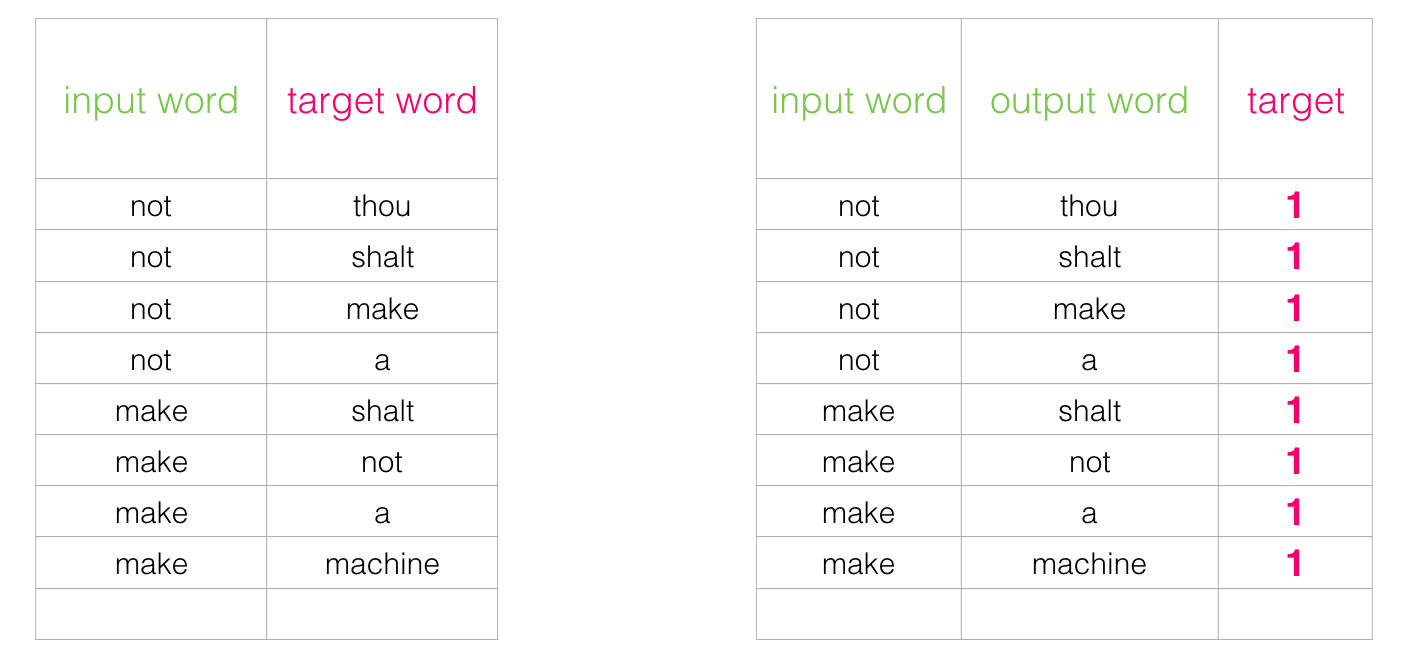


במודל שלוקח את מילת הקלט והפלט, ומפיק ציון המציין אם הם שכנים או לא (0 עבור "לא שכנים", 1 עבור "שכנים").

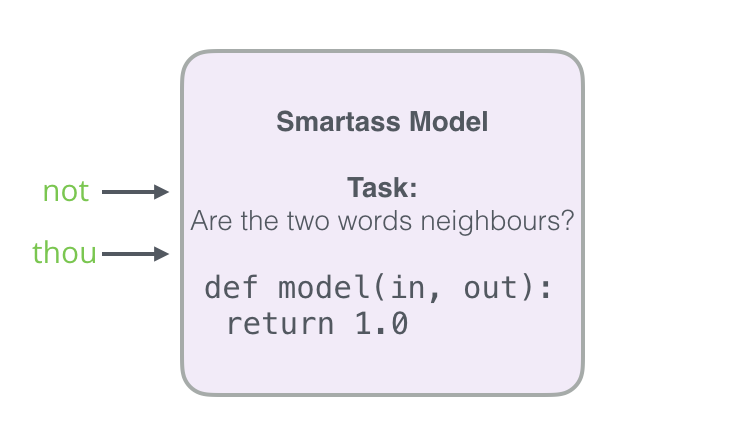


שינוי פשוט זה משנה את המודל שאנחנו צריכים מרשת עצבית, למודל רגרסיה לוגיסטית - ובכך הוא הופך להיות הרבה יותר פשוט והרבה יותר מהר לחישוב.

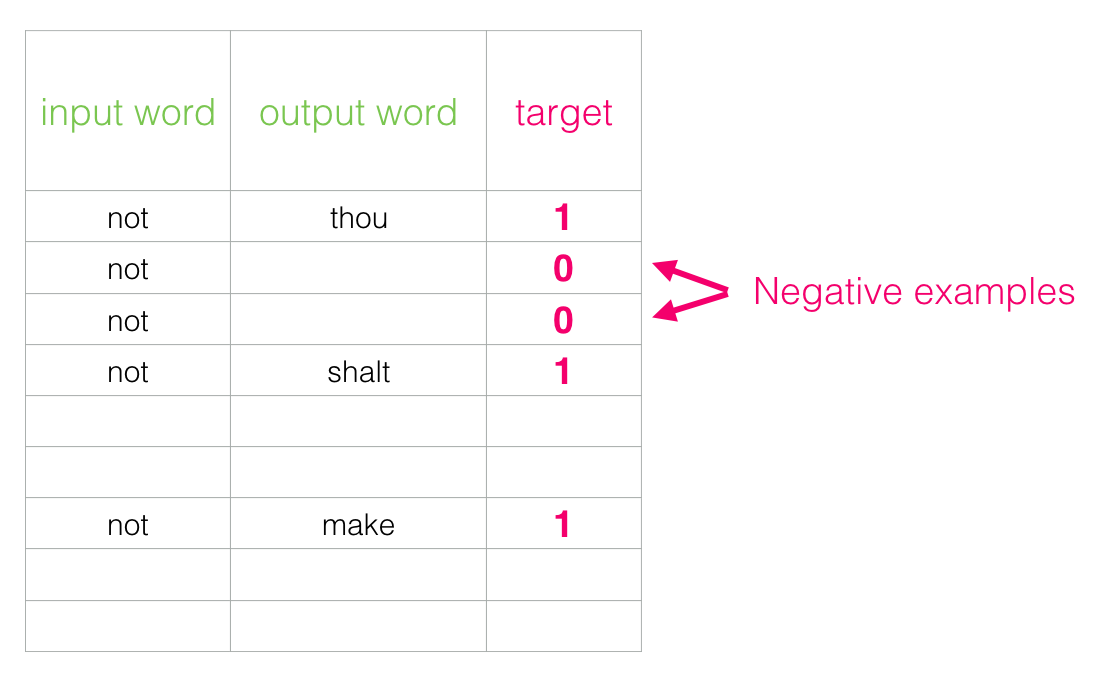
שינוי זה דורש החלפה של מבנה ערכת הנתונים – התווית היא כעת עמודה חדשה עם ערכים 0 או 1. אם הכל 1 אז כל המילים נוספו הם שכנים.



כעת ניתן לחשב זאת במהירות מסחררת – עיבוד מיליוני דוגמאות תוך דקות. אבל יש פרצה אחת שצריך לסגור. אם כל הדוגמאות חיוביות (יעד: 1), אנו פותחים את עצמנו לאפשרות של מודל smartass שתמיד מחזיר 1 - השגת דיוק של 100%, אבל לא לומדים כלום ומייצרים הטמעות אשפה.



כדי לטפל בכך, מוסיפים *דגימות שליליות* לערכת הנתונים – דוגמאות של מילים שאינן שכנות. המודל צריך להחזיר- 0 עבור הדגימות האלה. עכשיו זה אתגר שהמודל צריך לעבוד קשה כדי לפתור – אבל עדיין במהירות גבוהה.

  
עבור כל מדגם בערכת הנתונים, **מוסיפים דוגמאות שליליות**. לאלה יש את אותה מילת קלט, ותווית 0.

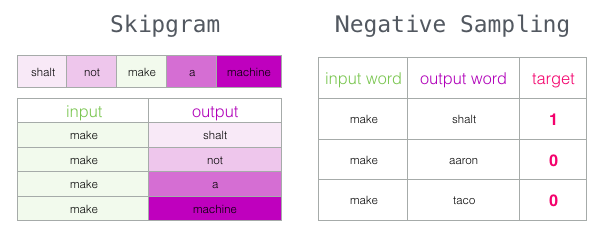
אבל מה ממלאים כמילות פלט? דוגמים באופן אקראי מילים מאוצר המילים.



רעיון זה מוביל להחלפה גדולה של יעילות חישובית וסטטיסטית.

## Skipgram עם דגימה שלילית (SGNS)

כעת כוסו שניים מהרעיונות המרכזיים ב- word2vec: כזוג, הם נקראים skipgram עם דגימה שלילית.

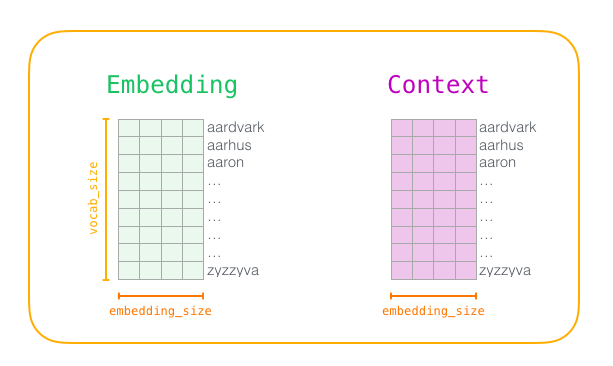


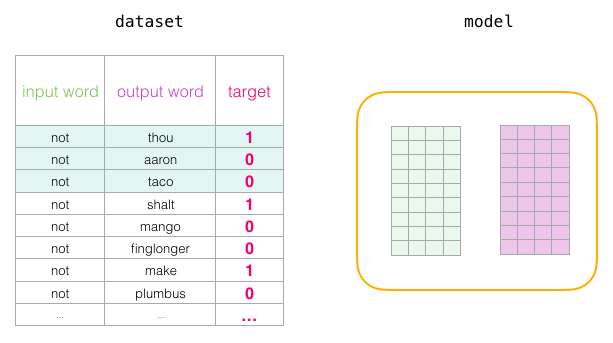
## תהליך ההדרכה של Word2vec

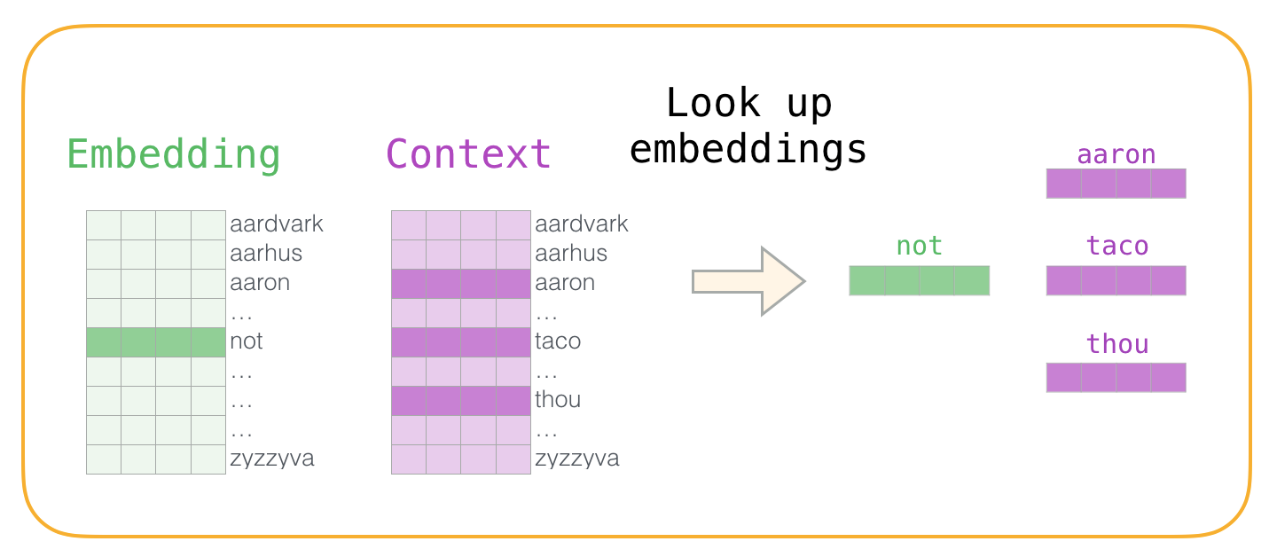
כעת, לאחר שביססנו את שני הרעיונות המרכזיים של סקיפגרם ודגימה שלילית, אנו יכולים להמשיך להסתכל מקרוב על תהליך האימון של word2vec.

לפני שתהליך ההכשרה מתחיל, מעבדים מראש את הטקסט שמאמנים לפיו את המודל. בשלב זה, קובעים את גודל אוצר המילים (נקרא לזה vocab\_size, למשל 10,000) ואילו מילים שייכות לו.

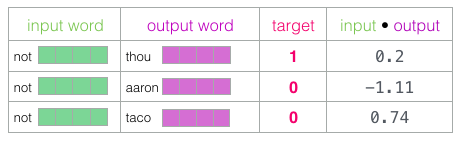
בתחילת שלב האימונים יוצרים שתי מטריצות – מטריצה Embeddin ומטריצה Context. לשתי המטריצות האלה יש הטמעה לכל מילה באוצר המילים (כך גם vocab\_size אחד הממדים שלהן). הממד השני הוא כמה זמן אנחנו רוצים שכל הטמעה תהיה (embedding\_size – 300 הוא ערך נפוץ, אבל הדוגמה הקודמת של 50 ).

בתחילת תהליך ההכשרה, מאתחלים מטריצות אלו עם ערכים אקראיים. ואז מתחיל תהליך ההכשרה. בכל שלב אימון, לוקחים דוגמה חיובית אחת ואת הדוגמאות השליליות הקשורות אליה. ניקח את הקבוצה הראשונה:

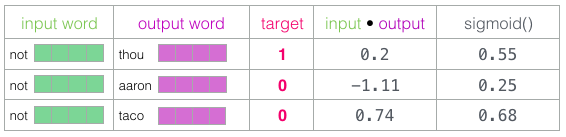
עכשיו יש ארבע מילים: מילת הקלט not ומילות הפלט/ההקשר: thou (השכן בפועל), taco וaaron (הדוגמאות השליליות). ממשיכים לחפש את ההטבעות שלהם - עבור מילת הקלט, מסתכלים במטריצה Embedding . עבור מילות ההקשר, מסתכלים במטריצה Context (למרות שלשתי המטריצות יש הטמעה לכל מילה באוצר המילים).



לאחר מכן, לוקחים את מכפלה הנקודות של הטבעת הקלט עם כל אחת מהטבעות ההקשר. בכל מקרה, זה יגרום למספר, מספר זה מציין את הדמיון של הטבעות הקלט וההקשר

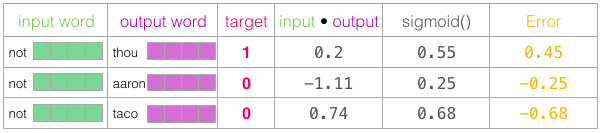


עכשיו צריך דרך להפוך את הציונים האלה למשהו שנראה כמו הסתברויות – צריך שכולם יהיו חיוביים ויהיו להם ערכים בין אפס לאחד. זוהי משימה נהדרת עבור [סיגמואיד](https://jalammar.github.io/feedforward-neural-networks-visual-interactive/" \l "sigmoid-visualization), [המבצע הלוגיסטי](https://en.wikipedia.org/wiki/Logistic_function).

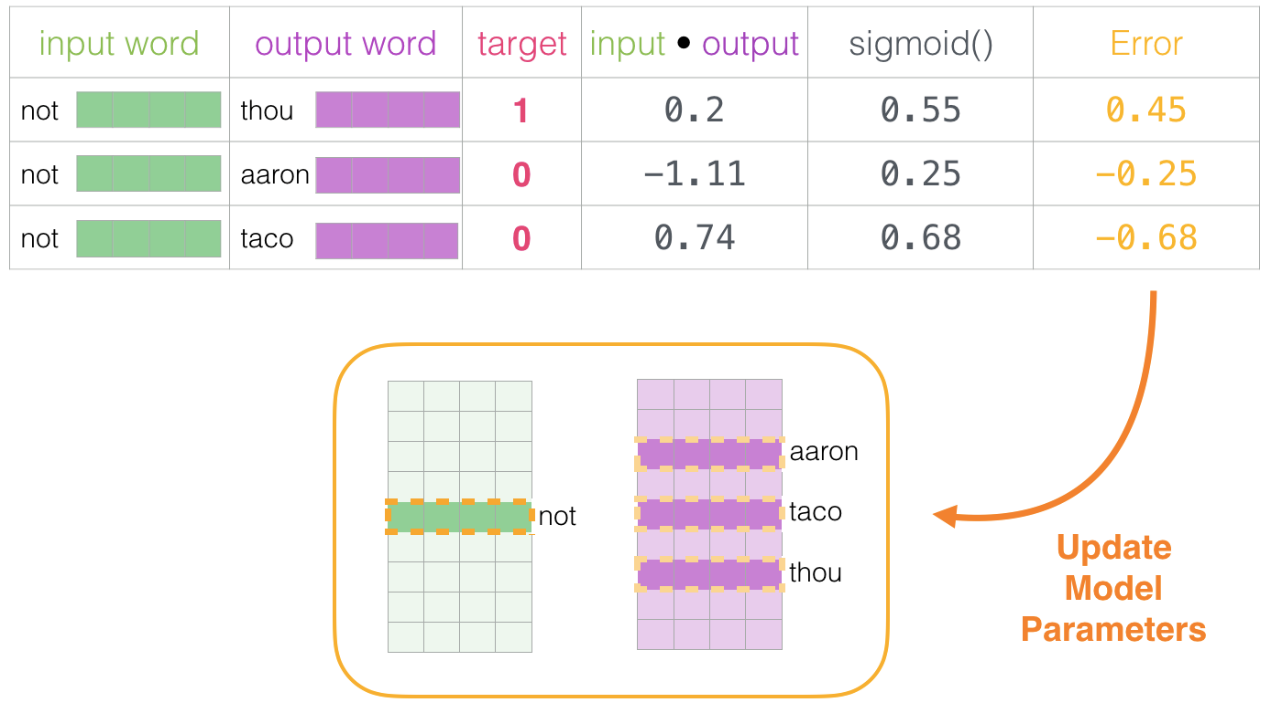


וכעת יכולים להתייחס לפלט של פעולות הסיגמואיד כפלט של המודל עבור דוגמאות אלה. אפשר לראות שיש לו taco את הציון הגבוה ביותר ו aaron עדיין יש את הציון הנמוך ביותר הן לפני ואחרי פעולות sigmoid.

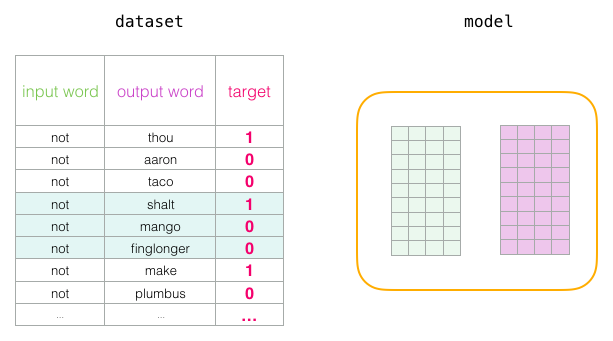
כעת, כשהמודל הלא מאומן ביצע חיזוי, ומאחר ויש תווית יעד ממשית להשוות מולה, מחשבים כמה שגיאה יש בחיזוי המודל. כדי לעשות זאת, פשוט להפחית את הציונים sigmoid מתוויות היעד.

  
error = target - sigmoid\_scores

הנה מגיע החלק "למידה" של "למידת מכונה". כעת מתאפשר להשתמש בניקוד שגיאה זה כדי להתאים את ההטבעות של not, thou, aaron ו taco כך שבפעם הבאה שנבצע חישוב זה, התוצאה תהיה קרובה יותר לציוני היעד.



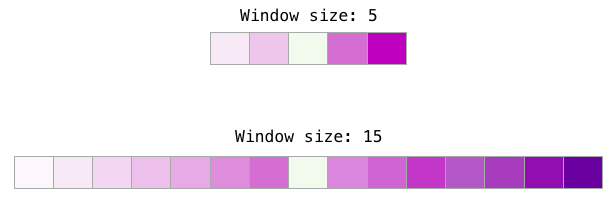
זה מסכם את שלב האימונים. יוצאים ממנו עם הטמעות מעט טובות יותר, עבור המילים המעורבות בשלב זה (not, thou, aaron ו taco ). כעת אנו ממשיכים לשלב הבא (המדגם החיובי הבא והדגימות השליליות המשויכות אליו) ועושים את אותו תהליך שוב.



ההטבעות ממשיכות להשתפר בזמן שעוברים על כל ערכת הנתונים במשך מספר פעמים. לאחר מכן עוצרים את תהליך ההכשרה, להעיף את המטריצה Context ולהשתמש במטריצה Embeddings כהטבעות המאומנות מראש למשימה הבאה.

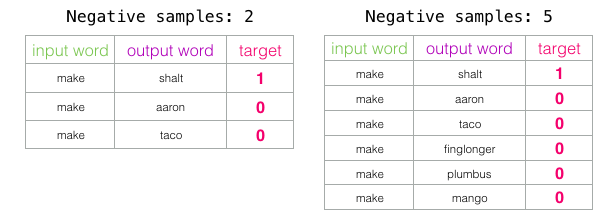
## גודל מסגרת ומספר דוגמאות שליליות

שני פרמטרים מרכזיים בתהליך האימון של word2vec הם גודל המסגרת ומספר הדגימות השליליות.



משימות שונות מוגשות טוב יותר על-ידי גדלי מסגרות שונים. [היוריסטיקה](https://youtu.be/tAxrlAVw-Tk?t=648) אחת היא שגודלי מסגרות קטניות (2-15) מוביליות להטמעות שבהן ציוני דמיון גבוהים בין שתי הטבעות מצביעים על כך שהמילים *ניתנות להחלפה* (שימו לב כי לעתים קרובות ניתן להחליף את הפכים אם אנו מסתכלים רק על המילים הסובבות אותם - למשל *טוב* *ורע* מופיעים לעתים קרובות בהקשרים דומים). גדלי חלונות גדולים יותר (15-50, או אפילו יותר) מובילים להטמעות שבהן הדמיון מעיד יותר על *הקשר בין* המילים. בפועל, לעתים קרובות יהיה צורך לספק הערות המנחות את תהליך ההטבעה המובילים לתחושת דמיון שימושית עבור המשימה הנוכחית.

גודל החלון המוגדר כברירת מחדל Gensim הוא 5 (שתי מילים לפני ושתי מילים לאחר מילת הקלט, בנוסף למילת הקלט עצמה).



מספר הדגימות השליליות הוא גורם נוסף לתהליך ההכשרה. המאמר המקורי קובע 5-20 כלהיות מספר טוב של דגימות שליליות. זה גם קובע כי 2-5 מספיק כאשר יש לך ערכת נתונים גדולה. ברירת המחדל Gensim היא 5 דוגמאות שליליות.

## למעשה

*"אם זה נופל מחוץ לאמצעים שלך, אז אתה עוסק עם אינטליגנציה, לא עם אוטומציה" ~הקיסר של חולית*

לאחר שראינו איך מודל word2vec מאומן ואיך משתמשים בו נוכל לעבור לאלגוריתם המרכזי בפרויקט:

האלגוריתם משתמש במודל מאומן מראש של word2vec הנקרא בשם:

word2vec-GoogleNews-vectors""

ספרית Gensim מכירה מודל מאומן זה, ויש לה פקודות מיוחדות שהופכות את המודל word2vec למודל שאפשר לעבוד איתו...

model = gensim.models.KeyedVectors.load\_word2vec\_format(  
 **'./GoogleNews-vectors-negative300.bin'**, binary=**True**,  
 limit=200000  
)

קבלת המודל וטעינתו ע"י הפקודה genism.models.KeyedVectors.load\_word2vec\_format

שמקבלת את המודל word2vec-GoogleNews-vectors"", האם בינארי (המודל בינארי כדי שבקלות נוכל לעבוד איתו וכמות הוקטורים שרוצים מהמודל)

## פונקציות עיקריות המשתמשות במודל:

### פונקצית guess

הפונקציה מנחשת את n המילים הדומות ביותר מתוך רשימת מילים נתונה (words) בהתבסס על רמז נתון (clue)

הפונקציה משתמשת בפקודת model.similarity המחזירה מספר בין 1- ל 1 המבטא את רמת הקשר בין שתי מילים

דמיון הסף לניחוש המילים הוא 0.2

הפונקיה מקבלת:

:param clue רמז נתון

:param wordsרשימה נתונה של מילים לנחש מהן

param n: מספר המילים המקסימלי לניחוש

הפונקציה מחזירה:

:return רשימה, באורך לכל היותר ,n של ניחושים אופטימלים

**הקוד:**

**def** guess(clue, words, n):  
 poss = {}  
 **for** w **in** words:  
 poss[w] = model.similarity(clue, w)  
 poss\_lst = sorted(poss, key=poss.\_\_getitem\_\_, reverse=**True**)  
 top\_n = poss\_lst[:n]  
 **return** [w **for** w **in** top\_n **if** poss[w] > 0.2]

### פונקצית give\_clue

נותן רמז אופטימלי המבוסס על מצב הלוח הנוכחי.

פונקציה זו מחשבת את מדד הדמיון בין כל הזוגות ושלשות המילים.

לאחר מכן, הוא מחשב באופן איטרטיבי רמז אופטימלי על סמך מספר המילים שנותרו

והמקסימום בין דמיון הזוגות הגבוה ביותר לדמיון השלישייה הגבוה ביותר.

אחרי מציאת הרמז. בדיקת האם הרמז תקין ע"י מעבר על bad\_words ושלילת הרמז האופטימלי שנופל על מילה מרשימת המילים bad\_words.

חיפוש אחר רמז אחר עד מציאת רמז אופטימלי שלא נופל על מילה מ- bad\_words.

והחזרת הרמז הנבחר

הפונקציה מקבלת:

param words: רשימת מילים ליצירת רמז עבורן

param bad\_words: רשימת מילים שיש להימנע ממתן רמזים עבורן

הפונקציה מחזירה:

possible\_clue: רמז אופטימלי,

: tuple(max\_correlated\_n) המילים שנועדו לנחש

הקוד:

מחבר כל שלשות מילים אפשריות (לפי כללי המשחק) למילון. ערך ה- key הוא tuple :(מילה 1, מילה 2, מילה 3) וערך ה- value שלהן הוא similarity המשותף. אם לא מצליח בשלשות מחבר זוגות באותו אופן.

משתנים:

max\_correlated\_n המילים שהרמז מתאים להן ולא נופל על מילה אסורה/ לא יעילה

max\_correlated\_pair זוגות מילים

max\_correlated\_triple שלשות מילים

cleaned\_clues - רשימת הרמזים לאחר ניקוי

possible\_clue – הרמז העדכני הטוב ביותר

**הקוד:**

