Code Names

סמינר בית יעקב "בנות אלישבע"

שם מכללה: בנות אלישבע
שם סטודנט: דסי שפירא
ת.ז. 212318026
שם מנחה: גב' ש. ברלין
תאריך הגשה כ"ז סיון

תוכן

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 שלהם 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 יכוון 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.

הפרויקט שלי הוא פיתוח למשחק "שם קוד" – כפי שבואר בתיאור המשחק. ע"י הטמעות מילים (יפורט בהמשך בחלק אלגוריתם מרכזי). וכך ישנה אפשרות למחשב למצוא רמז במשחק "שם קוד". רמז פרושו מילה אחת הקשורה למילים מסויימות, שתהווה כמין רמז אל המילים ההם לדוג' המילת רמז - שולחן למילים כיסא ומפה – שתי מילים שהמילה שולחן מקשרת בינהם.

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.

משחק "שם קוד" בנוי כולו על מילים והקשרים בין מילים – כל תהליך המשחק זה קבלת רמז ונתינת רמז (מילה אחת הקשורה למילים מסוימות כנ"ל). המחשב צריך לדעת לתת מילת רמז למילים הדומות לה. ולקבל מילת רמז ולבחור מילים הדומות לה.

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".

מטרות ויעדים

מטרות:

מבחינת המתכנת:

- הטמנת מילים בצורה הטובה ביותר.
- 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.

מבחינת המשחק:

- נתינת ה"מילת קוד" היעילה ביותר.
- מציאת המילים המרומזות ע"י ה"מילת קוד" שניתנה.
- אפשרות משחק מהבית עם חברים.

יעדים:

- הכרת ספרית Genism ומודל word2vec
- הכרת המודל "word2vec-GoogleNews-vectors"
- שימוש בפקודות מספרית Gensim לשימוש במודל
- 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.

אתגרים

- הכרת המודל word2vec:
  חיפוש אחר המודל הטוב ביותר שיחזיר לי מילים קשורות בקלות,
  לימוד דרך הפעולה שלו והתנסות עם הפונקציות של ספרית Gensim במודל.
- ללמוד ולהבין להשתמש במודל קיים.
- בניית הממשק למשתמש:
  אפשרות משחק עם חברים ו/או מול מחשב, מהבית והעברת המידע בין שחקנים.
- התמצאות בשפות.

מדדי הצלחה למערכת

הפרמטרים שיבדקו לבחינת ההצלחה הם:

- נתינת רמז ברמת התאמה של 90%
- מציאת מילים דומות מרמז ברמת התאמה של 90%
- ניהול המשחק בצורה הטובה ביותר

תיאור מצב קיים

קיום קיים המשחק, כמובן, במשחק קופסא פופולרי במיוחד, אבל אפשרות זו מחייבת חבר נוסף לפחות. והמשחק האופטימלי ביותר מינימום 4 שחקנים – דבר שמקשה קצת על קיום המשחק בספונטאני.

כאשר חקרתי את הנושא באינטרנט לא מצאתי את המשחק ממוחשב אלא רק כמה קודים כתובים – לא בתור אתר שפתוח לקהל הרחב, ומקיים את כל תהליך המשחק.

רקע תיאורטי, ניתוח חלופות ותיאור החלופה הנבחרת

רקע תיאורטי:

הקושי העיקרי במהלך משחק 'שם קוד' הוא מציאת מילה בעלת קשר לשתים –שלוש מילים נוספות, אך לא תיפול על אף מילה אסורה. (כפי שתואר בהצעת הפרויקט).

אם ננתח את תהליך המשחק מעט יותר לעמוק, נראה כי למעשה השחקן יושב מול קבוצת מילים גדולה, וצריך למצוא קשרים שונים ביניהן.

השחקן בוחר מילה מסוימת, ומרחיב את המשמעות שלה. לדוגמה, שולחן – יושבים לידו על כיסא, שמים עליו מפה, הוא רהיט העשוי מעץ וכו'.

כך, ע"י הרחבת המעגל ניתן להגיע למילים אחרות הנמצאות על הלוח ואינן בהכרח קשורות, במבט ראשון.

פעולה זו, המתבצעת באופן אוטומטי ופשוט במוח אנושי, אינה כה פשוטה לביצוע בעבור מחשב.

למחשב אין את יכולת החשיבה האינטואיטיבית האנושית, ולכן תכנות רגיל, הגורם למחשב לפעול באמצעות דפוסים מוגדרים וקבועים מראש, אינו יעיל בעבור פתרון בעיה זו.

כאן מגיע לעזרתנו תחום הבינה המלאכותית - למידת מכונה.

באמצעות אלגוריתמים של למידת מכונה, ניתן כביכול ללמד את המחשב 'להבין' את משמעות המילים, זאת ע"י אימון ממושך על מסד נתונים המכיל קטעי טקסט רבים מאד.

ניתוח חלופות:

אם רוצים לפתור רק את בעיית השחקנים, שלא תמיד קיימים ארבעה במקום אחד. אפשר ליצור משחק על הרשת, שמחייב ארבעה שחקנים להשתתף במשחק ממקומות שונים. אך פרויקט זה מאבד את רוב העניין, הוא הופך להיות רק תכני ולא חכם בכלל.

ישנה עוד אפשרות ליצור שחקן מחשב רנדומלי, אבל הסיכויים שייתן רמז טוב, הם כמעט אפסיים.

תיאור החלופה הנבחרת:

בפרויקט זה יצרתי את משחק "שם קוד" *גם חכם ובנוסף לכך גם משתף*. בפרויקט אפשר לשחק כמה שחקנים ממקומות שונים על הרשת, ובנוסף להוסיף שחקני מחשב כדי להשלים לארבעה שחקנים.

אפיון המערכת המוצעת

ניתוח דרישות המערכת

- אחוזי הצלחה גבוהים בנתינת ה"רמז" למילים דומות או במציאת מילים דומות.
- אלגוריתמים בעלי סיבוכיות נמוכה ככל הניתן
- כתיבה בסטנדרטיים מקצועיים, סדר ותיעוד
- ממשק נח וידידותי למשתמש
- מהירות תגובה מהירה ככל האפשר

מודל המערכת

- טעינת המודל למערכת.
- כניסת השחקנים למשחק, הוספתם למשחק והתחלת המשחק.
- נתינת "רמז" מילה הקשורה לשלוש או שתי מילים.
- מציאת מילים קשורות ל"רמז" נתון.
- ניהול תור המשחק ומראה הלוח.
- סיום בניצחון אחת הקבוצות.

אפיון פונקציונלי

- Creat game – התחלת משחק
- Join game – הצטרפות למשחק חדש
- Start game – התחלת משחק
- Give clue – נתינת "רמז"
- Gusse - מציאת מילים קשורות מ"רמז"

ביצועים עיקריים

המשתמש יצור/יצטרף למשחק. לפי כמות השחקנים יצטרפו למשחק שחקני מחשב כדי להשלים לארבעה שחקנים מינימום. ויתחיל המשחק.

כל שחקן בתורו ייתן רמז או יקבל רמז לפי התפקיד שלו, מראה הלוח ישתנה מתור לתור אצל כל השחקנים, ולבסוף תוכרז הקבוצה המנצחת.

אילוצים

- לפעמים המחשב נותן רמזים חכמים מדי שקשה לעלות על הקשר.
- זמן העבודה על הפרויקט היה מוגבל. הפרויקט היה צריך להיות מוכן ועובד היטב עד לתאריך ההגשה.
- הפרויקט היה צריך לעמוד בתנאי סיבוכיות זמן ומקום יעילים.
- על הממשק להיות פשוט ומובן לשימוש עבור המשתמש.

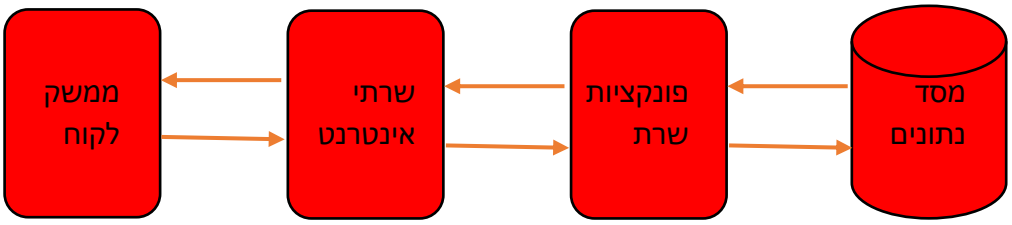
תיאור הארכיקטורה

הארכיטקטורה של הפתרון המוצע בפורמט של Top-Down level Design

התכנון נעשה מן הכלל אל הפרט. השלב הראשון בתכנון הפרויקט היה הסתכלות כללית, והשלבים

הבאים- הם ירידה לעומק, רובד אחר רובד, שלב אחר שלב, לעומק האלגוריתם ולפרטיו.

תיאור הרכיבים בפתרון



תיאור פרוטוקולי התקשורת

HTTP

שרת – לקוח

צד השרת הוא קוד הנכתב בשפת python .

צד הלקוח נכתב בטכנולוגיה החדשנית- React,

הקוד נכתב ב components- שהן קובצי JavaScript והתצוגה ב-html .

התקשורת ביניהם נעשית ע"י קריאות שרת בטכנולוגיית Api Web .

צורת העבודה היא כזאת:

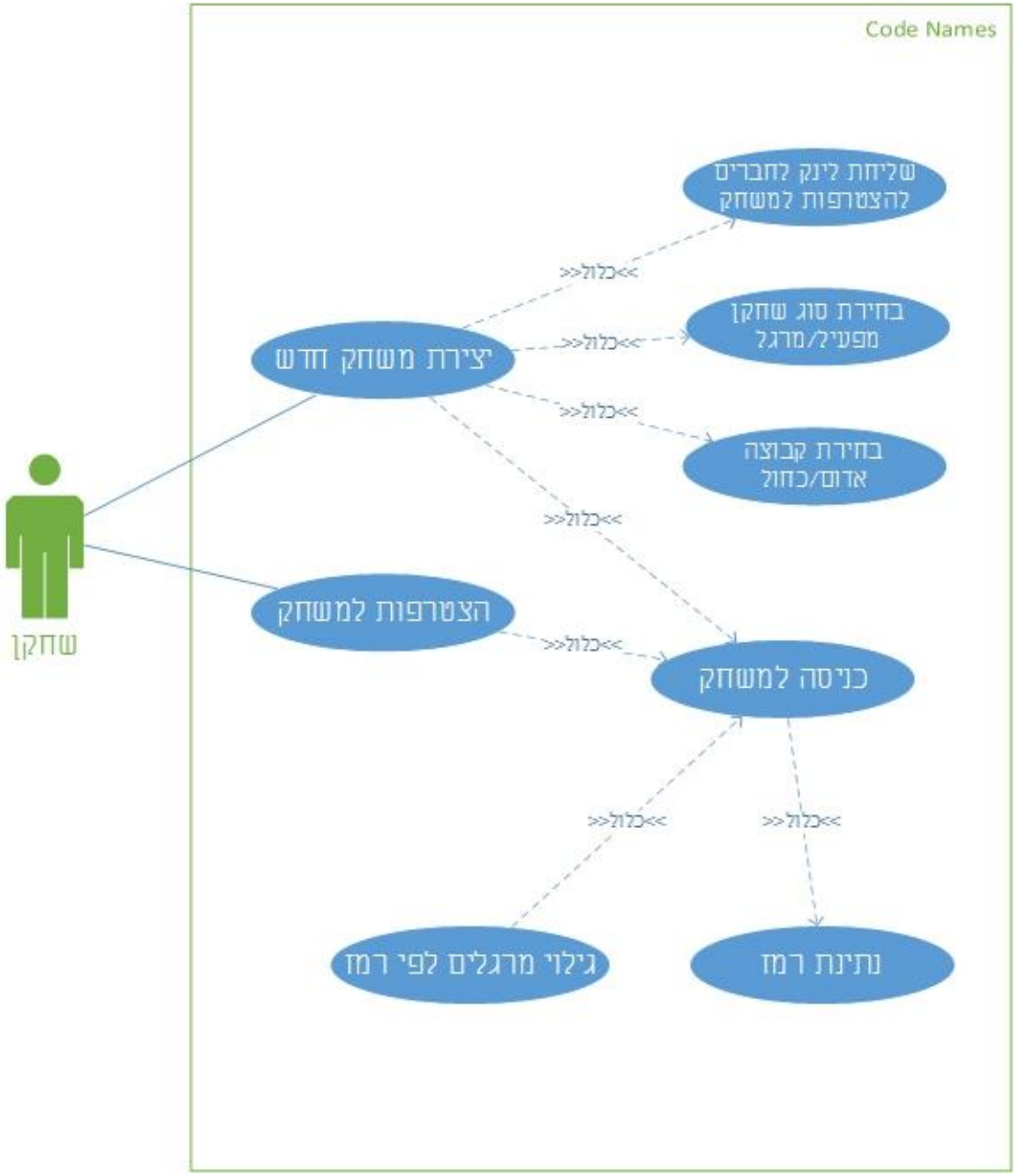
צד הלקוח מעביר נתונים מ/לשרת ע"י קריאה לaction בcontroller.

ה-controller מחזיר נתונים לצד לקוח.

ה-component מעבדת את הנתונים ומציגה אותם למשתמש.

ה-controllers כתובים ע"י שימוש בספריית flask אשר מחברת בין 2 הצדדים.

תיאור ה UC-העיקריים של המערכת



רשימת Use cases:

- **יצירת משחק חדש:** השחקן יוצר משחק חדש - מכניס שם, בוחר את סוג השחקן: רב-מרגלים (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: רב-מרגלים, נותן רמז. מרגל, מאתר מרגלים (מילים דומות) ע"י הרמז ולבסוף ניצחון של אחד הקבוצות

- **נתינת רמז:** כל רב-מרגלים 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 מרב-המרגלים של הקבוצה וניחוש המילים הדומות לרמז – מציאת המרגלים.

תיאור ה UC-העיקריים של המערכת

שם: יצירת משחק חדש.

מזהה: UC1.

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: UC2.

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: UC2.

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.

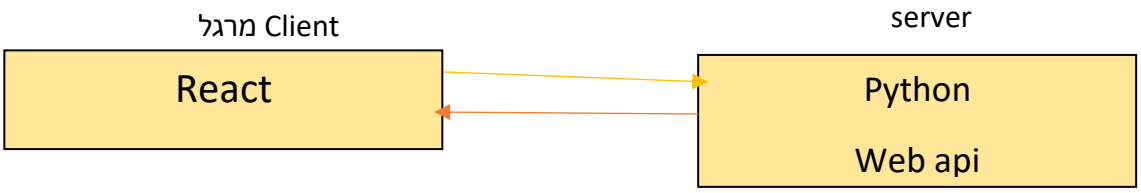
מבני נתונים בהם משתמשים בפרויקט

מילון tuple:

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.

רשימה: ליצור את לוח המשחק ע"י רשימת מילים לכל צבע (אדום, כחול נטרלי ומתנקש)

תרשים מודולים



תרשים מחלקות

תיאור המחלקות המוצעות

מחלקת Game:

תפקיד: מחלקה הבונה משחק "שם קוד"

פונקציות:

- addPlayer:
  תפקיד: מוסיפה שחקן למשחק
  קלט: שם, תפקיד, צבע קבוצה, האם בן-אדם של שחקן
  פלט: מחזיר true/false האם אפשרי להוסיף את המשחק ואת תפקיד השחקן
- __init__
  תפקיד: יוצרת משחק חדש
- Orderly_queue:
  תפקיד: מסדרת את התור של המשחק
- Next_player:
  תפקיד: מעבירה את התור לשחקן הבא
- Press_word:
  תפקיד: כאשר נלחצת מילה
  קלט: מילה ושחקן שלחץ על המילה

מחלקת Gruop:

תפקיד: אחראית על הקבוצה של השחקנים במשחק

פונקציות:

- __init__
  תפקיד: יוצרת קבוצה חדשה
- Guss:
  תפקיד: קבלת רמז ומציאת מילים קשורות
  קלט: רמז, רשימת מילים של הקבוצה שלה רוצים לתת את הרמז ומספר ניחושים
  פלט: המילים שנמצאו קשורות
- Clean_clue:
  תפקיד: בודקת אם הראשונה אינה נופלת על המילה השניה
  קלט: 2 מילים
  פלט: True/False
- Give_clue:
  תפקיד: נותנת מילת רמז
  קלט: רשימת מילים לניחוש ורשימת מילים אסורות
  פלט: רמז לכמה מילים (2/3)

פונקצית Player:

תפקידה: שחקן במשחק

פונקציות:

- __init__
  תפקיד: יוצרת שחקן חדש

פונקצית board:

תפקידה: לוח המשחק

פונקציות:

- __init__
  תפקיד: מפעילה את פונקצית board
- Generat_board:
  תפקיד: יוצרת לוח משחק חדש
  קלט: רשימת מילים

תיאור התוכנה

סביבות עבודה

צד שרת- Pycharm

צד לקוח - Visual Studio Code

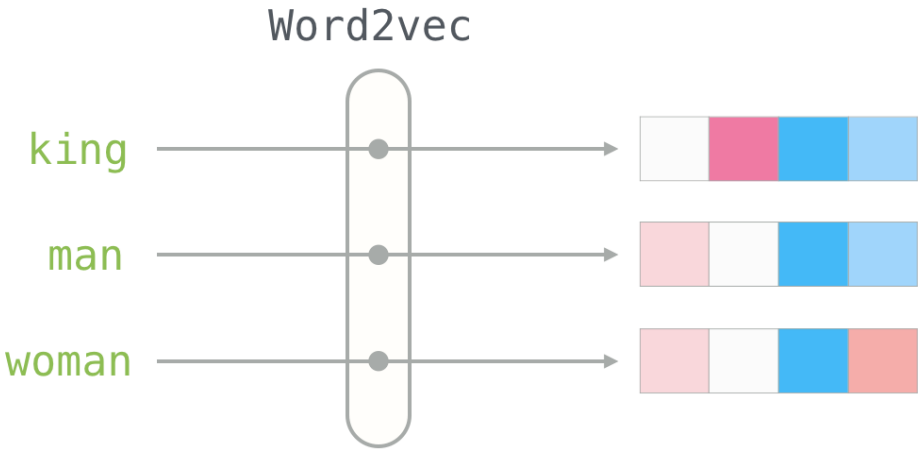
שפות תכנות

צד שרת- Python

צד לקוח –React, HTML, JavaScript

אלגוריתם מרכזי

בפרויקט הזה השתמשתי במודל מאומן מראש של word2vec בספרית Gensim על הDataBase – "GoogleNews ", כדי להשתמש במודל זה, הייתי צריכה מראש ללמוד את הנושא לעומק והרי הוא לפניכן בקצרה.



הטמעת מילים (לקיחת מילה ולמפות אותה לוקטור מספרי מימדים רבים) 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, Spotify, Anghami, Alibaba 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?

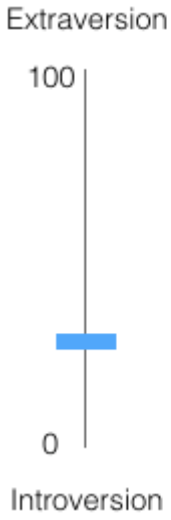
הטמעת/ חיזוי אישיות: איך זה עובד?

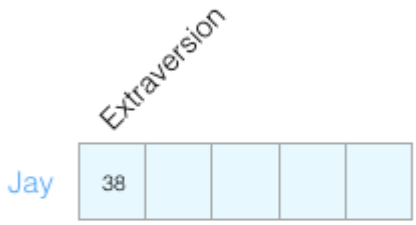
בסולם של 0 עד 100, כמה מופנם / מוחצן את/ה (באשר 0 הוא המופנם ביותר, ו 100 הוא המוחצן ביותר)? האם אי פעם עשית/ה מבחן אישיות כמו MBTI – או אפילו טוב יותר, מבחן " Big Five Personality Traits"? אם לא, אלה מבחנים ששואלים אותך רשימה של שאלות, ואז מציינים לך על מספר צירים את האישיות שלך, מופנמות / החצנה אחד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

Openness to experience	79 out of 100
Agreeableness	75 out of 100
Conscientiousness	42 out of 100
Negative emotionality	50 out of 100
Extraversion	58 out of 100

דוגמה לתוצאה של מבחן Big Five Personality Traits. זה באמת יכול לספר לך הרבה על עצמך והוא הוכיח שיש לו יכולת ניבוי בהצלחה אקדמית, אישית ומקצועית.

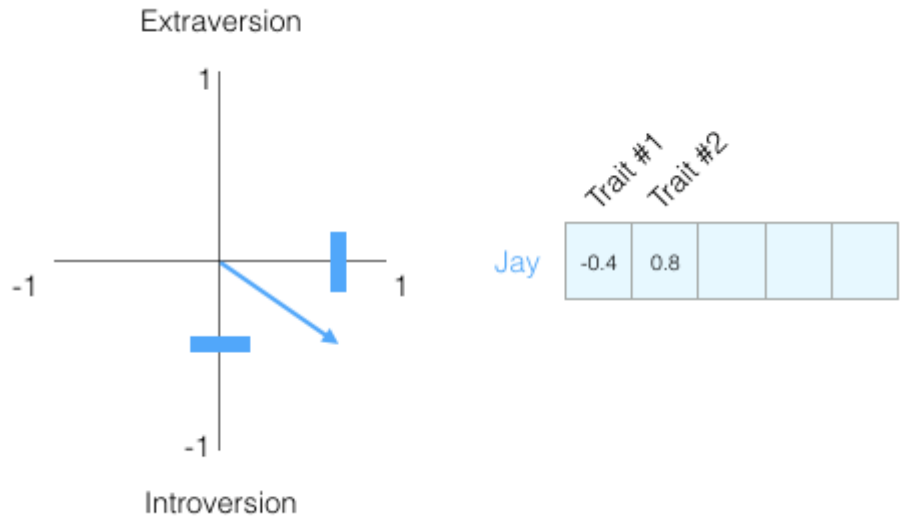
תארו לעצמכם שjay קיבל 38/100 כציון ההפנמה/ ההחצנה שלו. אנחנו יכולים לשרטט את זה בדרך זו:





בוא נחליף את הטווח כך שיהיה מ-1 ל-1:

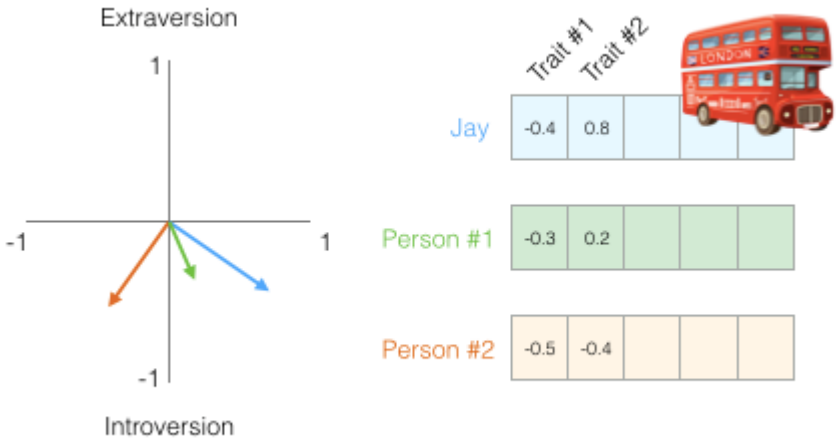
עד כמה את/ה מכיר אדם שיודע רק את פיסת המידע האחת הזו עליו? לא הרבה. אנשים מורכבים. אז בואו נוסיף עוד מימד - ציון של תכונה אחת נוספת מהמבחן.



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 ע"י פעולות חשבוניות על וקטורים (פלוס, מינוס, כפל וכו'):

Jay Person #1

cosine_similarity(-0.4 0.8 , -0.3 0.2) = 0.87

Jay Person #2

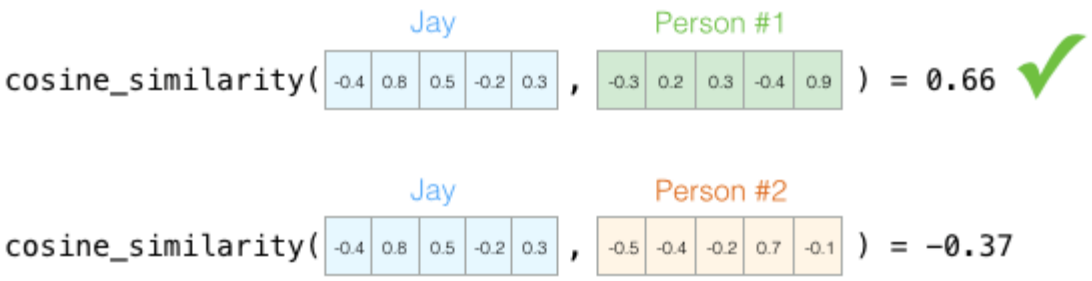
cosine_similarity(-0.4 0.8 , -0.5 -0.4) = -0.20

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:

	Trait #1	Trait #2	Trait #3	Trait #4	Trait #5
Jay	-0.4	0.8	0.5	-0.2	0.3
Person #1	-0.3	0.2	0.3	-0.4	0.9
Person #2	-0.5	-0.4	-0.2	0.7	-0.1

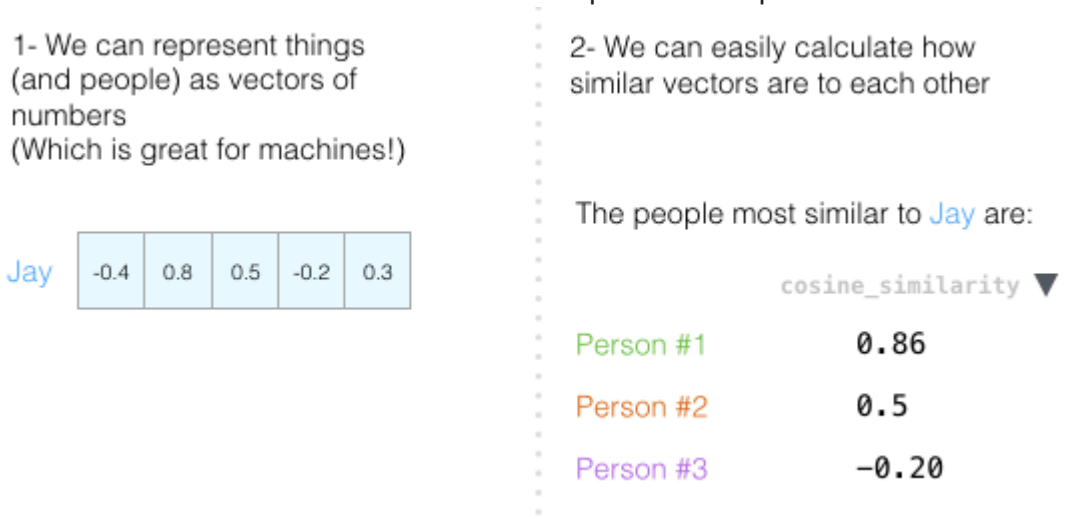
הבעיה עם חמישה ממדים היא שאנחנו מאבדים את היכולת לצייר חיצים קטנים ומסודרים בדו מימד. זהו אתגר נפוץ בלמידת מכונה שבה אנחנו צריכים לעתים קרובות לחשוב במרחב ממדי גבוה יותר. עם זאת, הדבר הטוב הוא ש-cosine_similarity עדיין עובד. זה עובד עם כל מספר של ממדים:



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

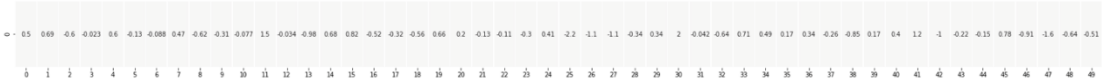
"מתנת המילים היא מתנת ההונאה והאשליה" ~ ילדי חולית

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

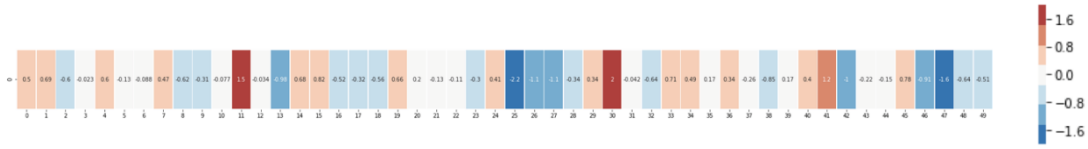
וקטור/הטמעות מילים עבור המילה "מלך" (מוקוצר לצורך הדוגמא) ממודל Word2vec:

[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]

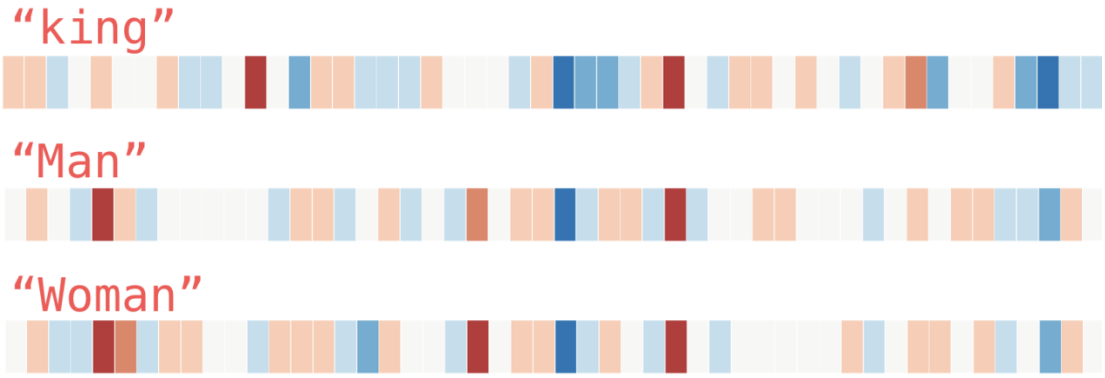
זו רשימה של 50 מספרים. אנחנו לא יודעים הרבה רק על ידי התבוננות בערכים. אבל בואו נדמיין את זה קצת כדי שנוכל להשוות את זה עם וקטורים אחרים של מילים. בואו נשים את כל המספרים האלה בשורה אחת:



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-):

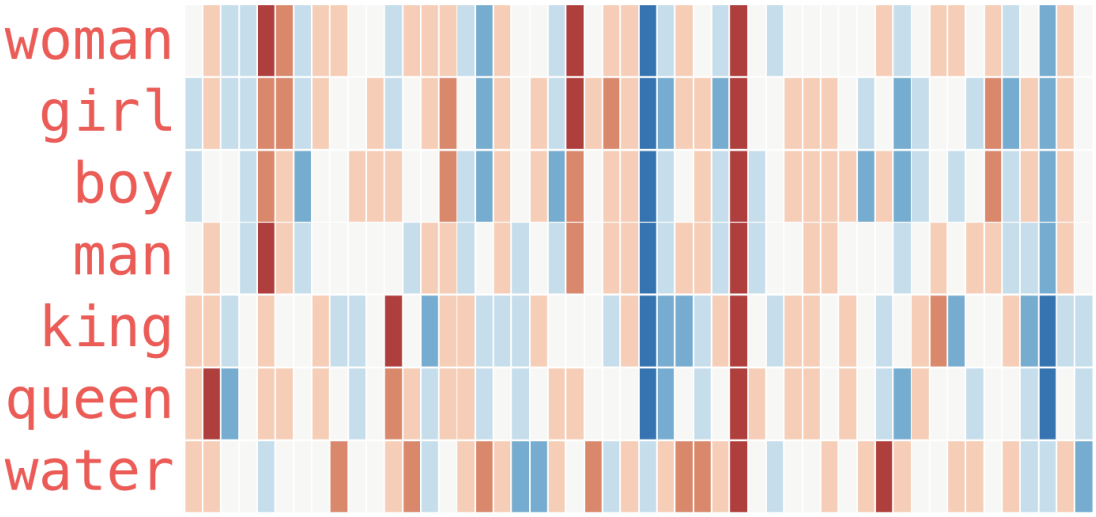


נמשיך על ידי התעלמות מהמספרים והסתכלות רק על הצבעים כדי לציין את ערכי התאים. בואו נשווה עכשיו "מלך" למילים אחרות:



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":

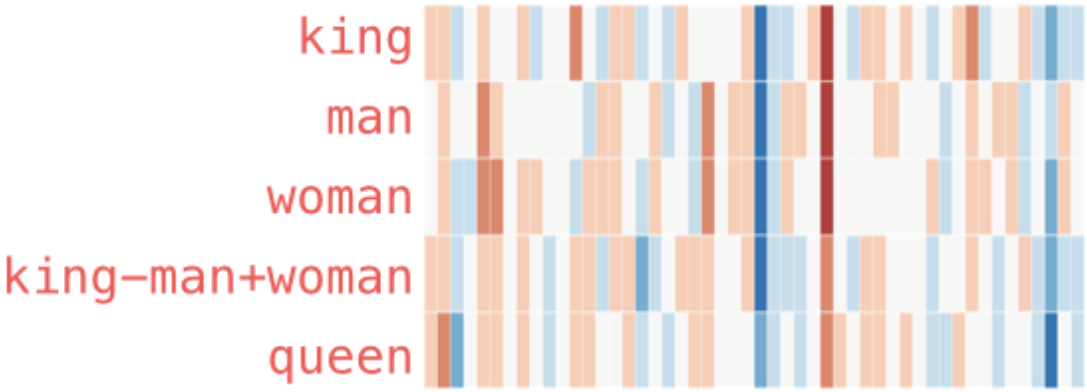
```
model.most_similar(positive=["king","woman"], negative=["man"])
```

```
[('queen', 0.8523603677749634),
 ('throne', 0.7664333581924438),
 ('prince', 0.7592144012451172),
 ('daughter', 0.7473883032798767),
 ('elizabeth', 0.7460219860076904),
 ('princess', 0.7424570322036743),
 ('kingdom', 0.7337411642074585),
 ('monarch', 0.721449077129364),
 ('eldest', 0.7184862494468689),
 ('widow', 0.7099430561065674)]
```

באמצעות ספריית 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:

king − man + woman ~= queen



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 עיבוד השפה.

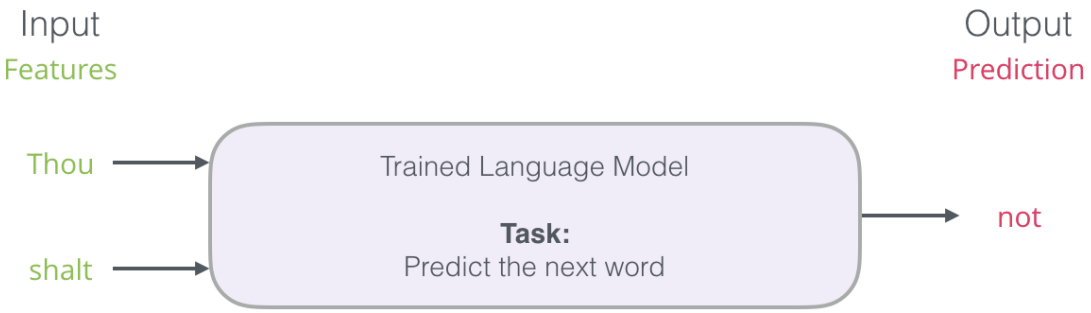
איך רשת הנוירונים של word2vec עובדת. איך נוצרים לנו הוקטורים לכל מילה בשפה שלנו. אז נתחיל

עיבוד שפה - NLP

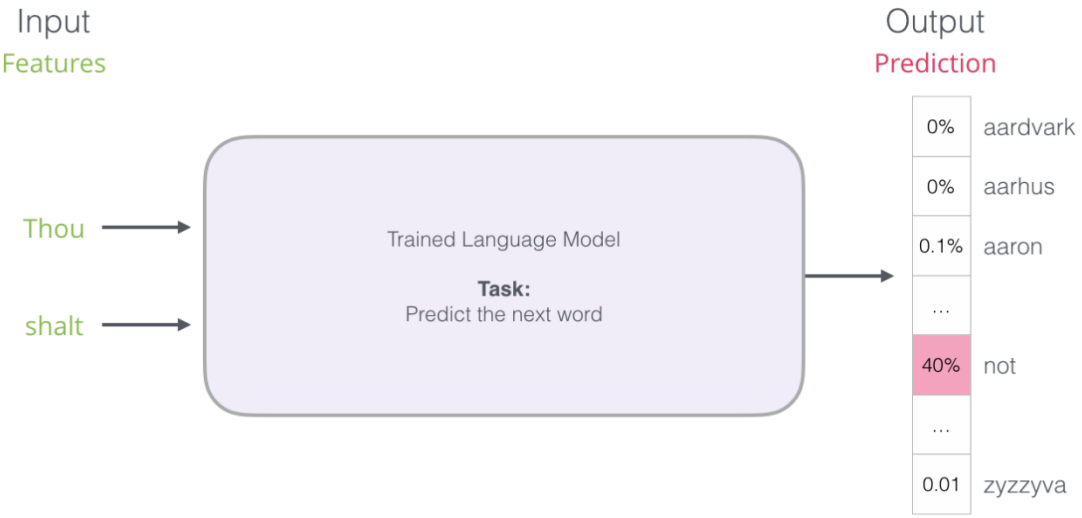
דוגמה ליישום NLP, אחת הדוגמאות הטובות ביותר תהיה תכונת החיזוי של המילה הבאה של מקלדת טלפון חכם. זו תכונה שמיליארדי אנשים משתמשים בה מאות פעמים בכל יום.

חיזוי המילים הבאות הוא משימה שמודל עיבוד שפה יכול לטפל בה. מודל עיבוד 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:

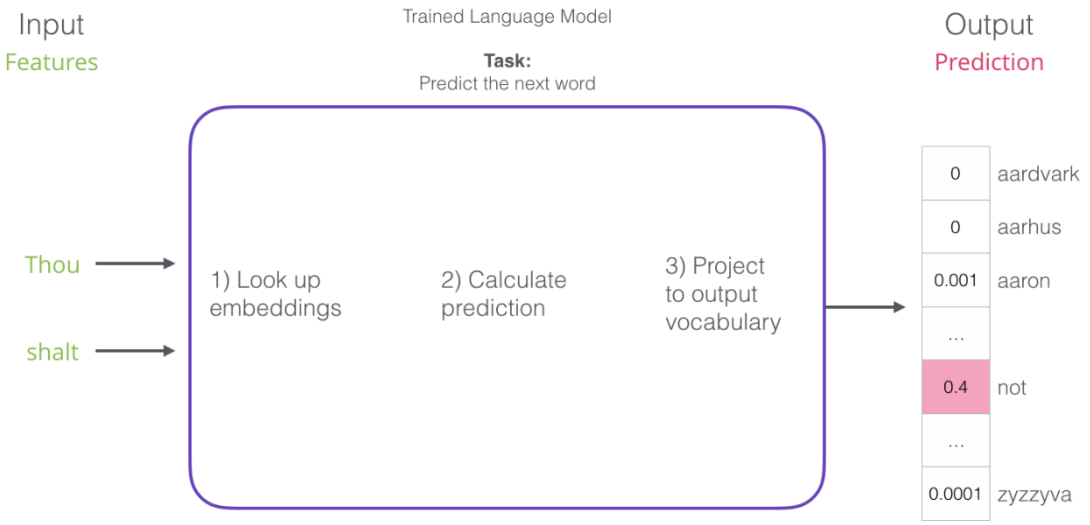


אבל בפועל, המודל לא פלט רק מילה אחת. הוא בעצם פלט ציון הסתברות עבור כל המילים שהוא יודע ("אוצר המילים" של המודל, אשר יכול לנוע בין כמה אלפים ליותר ממיליון מילים). יישום המקלדת צריך למצוא את המילים עם הציונים הגבוהים ביותר, ולהציג אותם למשתמש.

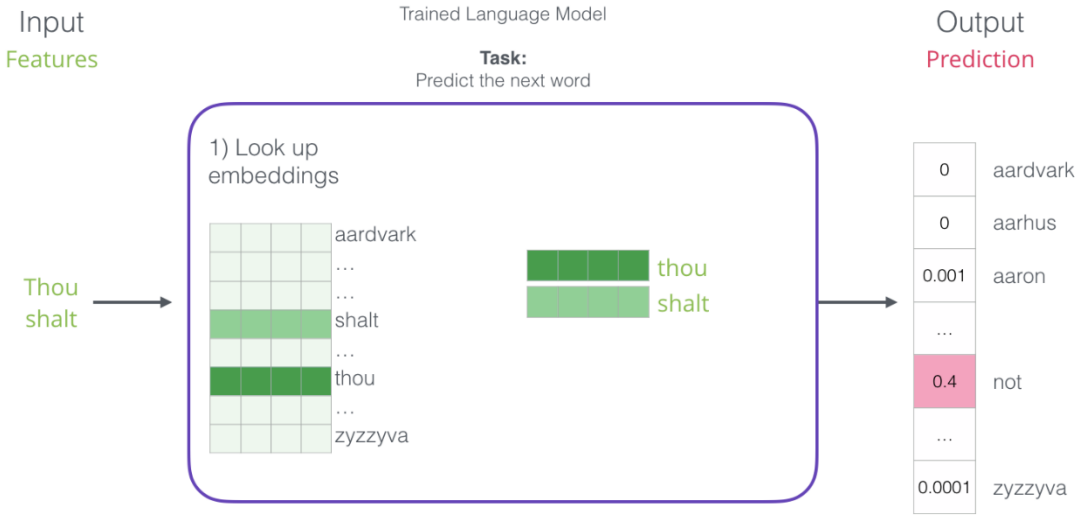


הפלט של מודל עיבוד השפה הוא ציון הסתברות עבור כל המילים שהמודל יודע. אנחנו מתייחסים להסתברות כאחוז כאן, אבל 40% יוצגו למעשה כ-0.4 בווקטור הפלט.

לאחר ההכשרה, מודלים מוקדמים של עיבוד שפה (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 בעיבוד 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'

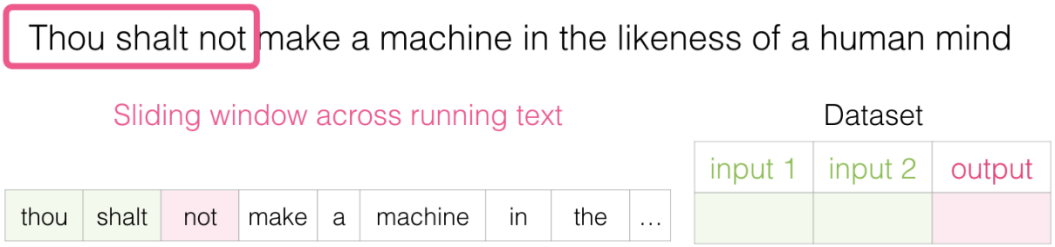
מילים מקבלות את ההטבעות שלהן על ידי כך שאנחנו מסתכלים על אילו מילים אחרות הן נוטות להופיע ליד. הטכניקה של זה היא:

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.

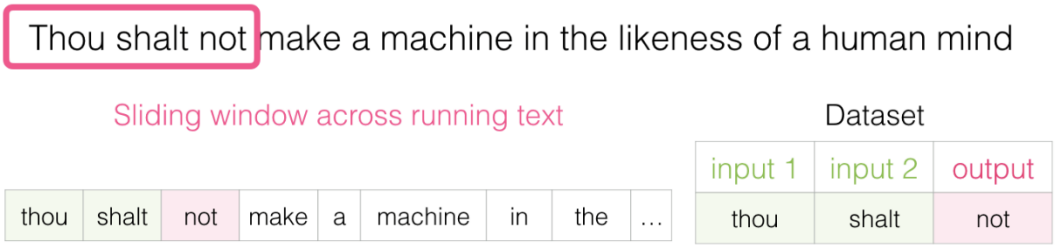
כאשר מסגרת זו מועברת בטקסט, יוצרים ערכת נתונים בה משתמשים כדי לאמן מודל. כדי לבחון בדיוק כיצד זה נעשה, בוא נראה כיצד מסגרת ההזזה מעבדת את הביטוי הזה:

"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 לעיבוד שפה.

לאחר מכן מעבירים את המסגרת למיקום הבא ויוצרים מדגם שני:

Thou shalt not make a machine in the likeness of a human mind

Sliding window across running text

thou	shalt	not	make	a	machine	in	the	…
thou	shalt	not	make	a	machine	in	the	

Dataset

input 1	input 2	output
thou	shalt	not
shalt	not	make

דוגמה שנייה נוצרת כעת.

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

Thou shalt not make a machine in the likeness of a human mind

Sliding window across running text

thou	shalt	not	make	a	machine	in	the	…
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	

Dataset

input 1	input 2	output
thou	shalt	not
shalt	not	make
not	make	a
make	a	machine
a	machine	in

נתונים משני הכיוונים

לפי איך שראינו, מלא את החסר:

Jay was hit by a _____

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?

Jay was hit by a _____ bus

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.

Jay was hit by a _____ bus in...

by	a	red	bus	in

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

input 1	input 2	input 3	input 4	output
by	a	bus	in	red

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:

Jay was hit by a red bus in...

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:

Jay was hit by a red bus in...

by	a	red	bus	in

input	output
red	by
red	a
red	bus
red	in

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:

Thou shalt not make a machine in the likeness of a human mind

thou	shalt	not	make	a	machine	in	the	...

input word	target word
not	thou
not	shalt
not	make
not	a

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

Thou shalt not make a machine in the likeness of a human mind

thou	shalt	not	make	a	machine	in	the	...

thou	shalt	not	make	a	machine	in	the	...

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

כמה עמדות מאוחר יותר, יש לנו הרבה יותר דוגמאות:

Thou shalt not make a machine in the likeness of a human mind

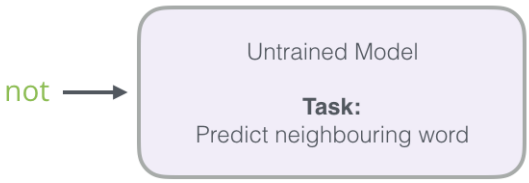
thou	shalt	not	make	a	machine	in	the	…
thou	shalt	not	make	a	machine	in	the	…
thou	shalt	not	make	a	machine	in	the	…
thou	shalt	not	make	a	machine	in	the	…
thou	shalt	not	make	a	machine	in	the	…

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness

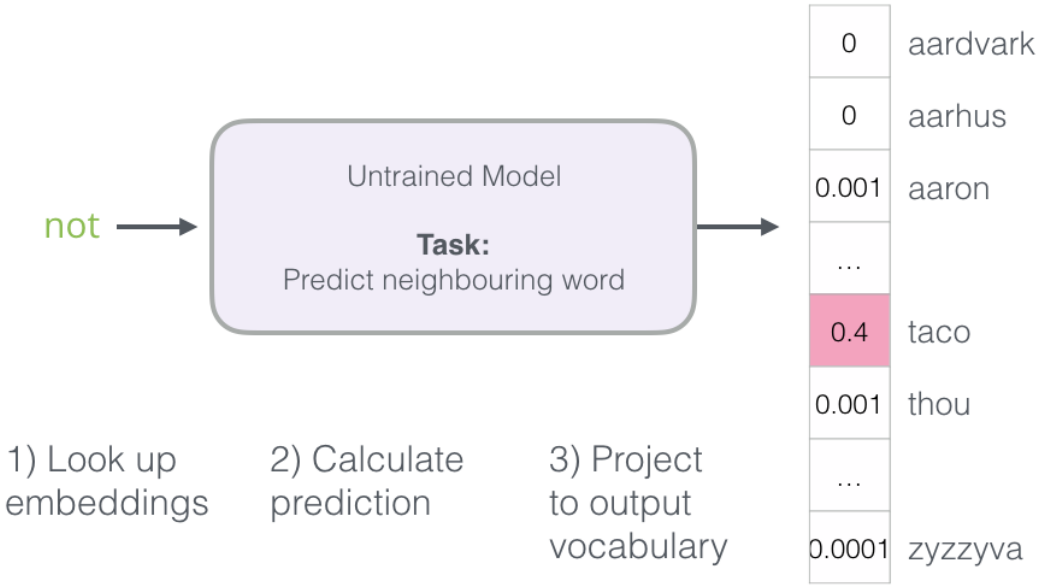
בחינה מחודשת של תהליך ההכשרה

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 לעיבוד שפה בסיסי שחוזה את המילה השכנה.

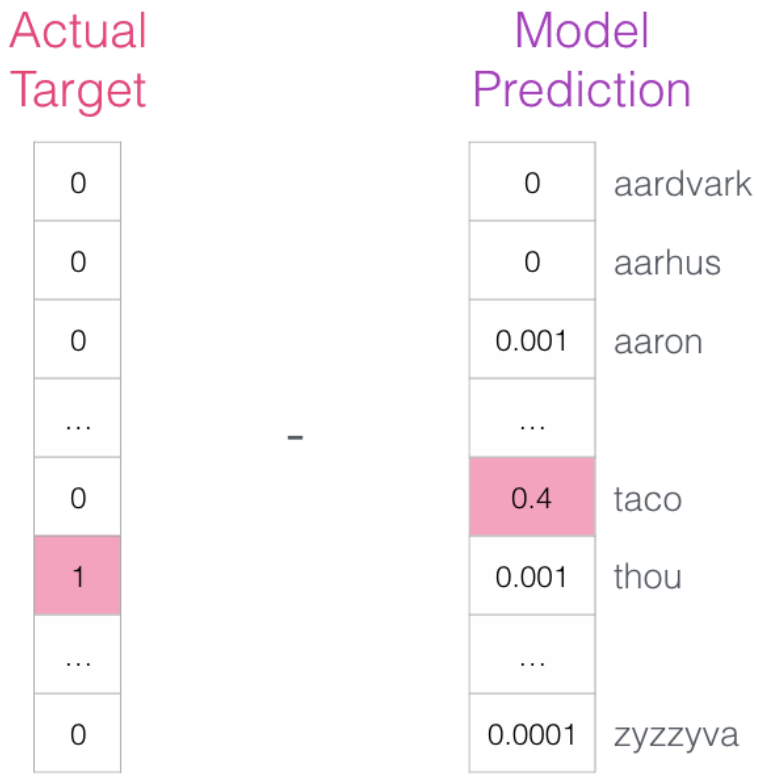
input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness



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

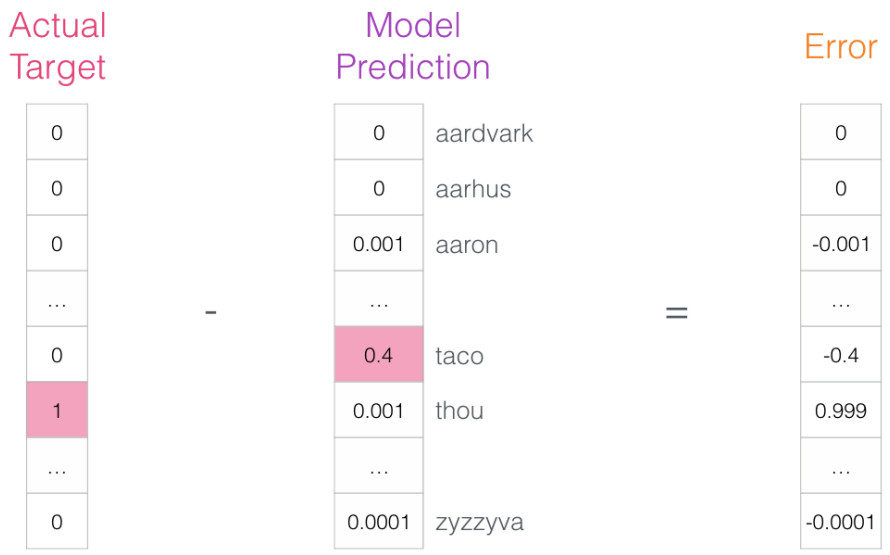


המודל מבצע את שלושת השלבים ומפיק וקטור חיזוי (עם הסתברות המוקצית לכל מילה באוצר המילים שלה). מכיוון שהמודל אינו מאומן, התחזית שלו בטוח תהיה שגויה בשלב זה. אבל זה בסדר. אנחנו יודעים איזו מילה הוא היה צריך לנחש – תא התווית/פלט בשורה שבה אנו משתמשים כרגע כדי לאמן את המודל:

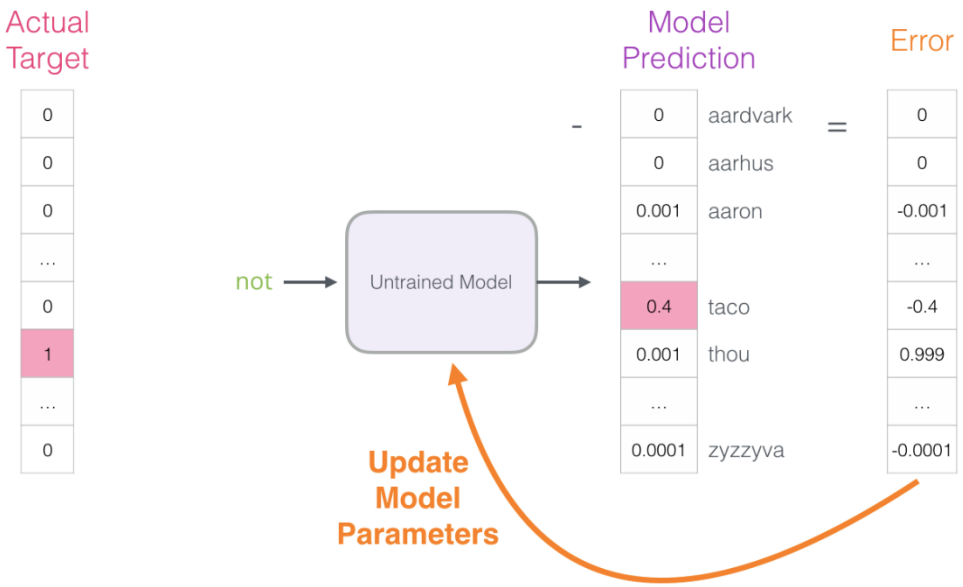


'וקטור היעד' הוא וקטור שבו מילת היעד יש את ההסתברות 1, וכל המילים האחרות יש את ההסתברות 0.

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 בקלט.



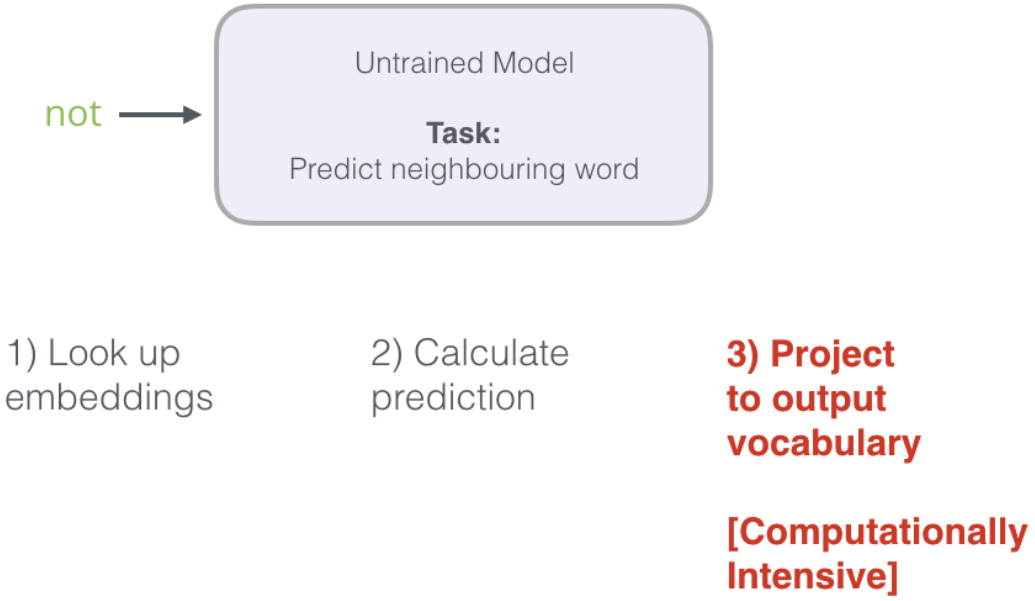
וזה מסכם את הצעד הראשון של האימונים. ממשיכים לעשות את אותו תהליך עם המדגם הבא בערכת הנתונים, וחוזר חלילה, עד שמכסים את כל הדגימות בערכת הנתונים. זה מסכם *תקופה* אחת של אימונים. עושים את זה שוב במשך מספר *תקופות*, ואז המודל מאומן ויכולים לחלץ את מטריצת ההטבעה ממנו ולהשתמש בו לכל יישום אחר.

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 עיבוד שפה זה מחשב את התחזית שלו:



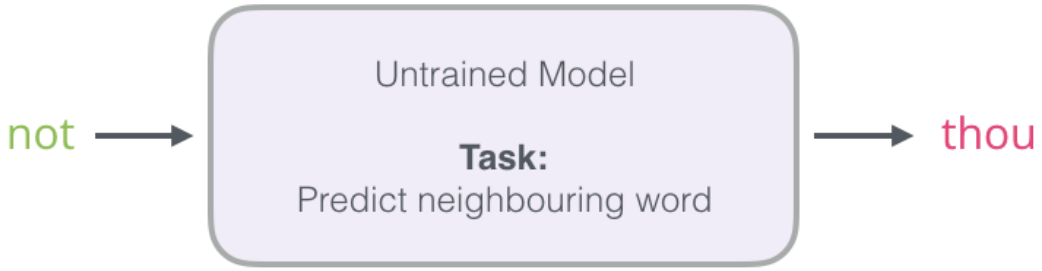
השלב השלישי יקר מאוד מבחינה חישובית – במיוחד בידיעה שהוא נעשה פעם אחת עבור כל מדגם אימון בערכת הנתונים (בקלות עשרות מיליוני פעמים). צריכים לעשות משהו כדי לשפר את הביצועים.

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 לעיבוד שפה (ביצוע חיזוי של המילה הבאה).

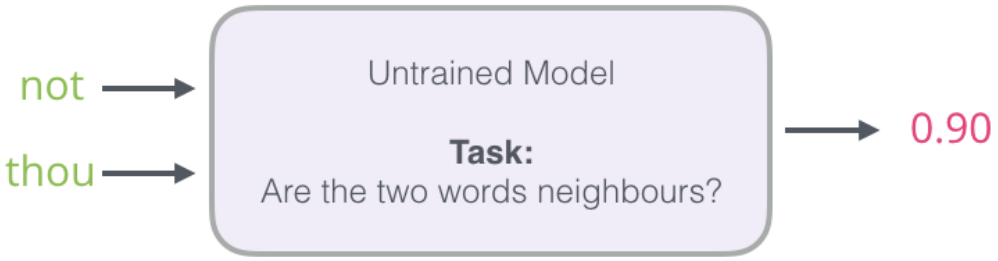
נתמקד בשלב 1. כאשר אנו מתמקדים בהטבעות. כדי ליצור הטמעות באיכות גבוהה באמצעות מודל בעל ביצועים גבוהים, אנו יכולים להחליף את משימת המודל מחיזוי מילה שכנה:

Change Task from



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

To:



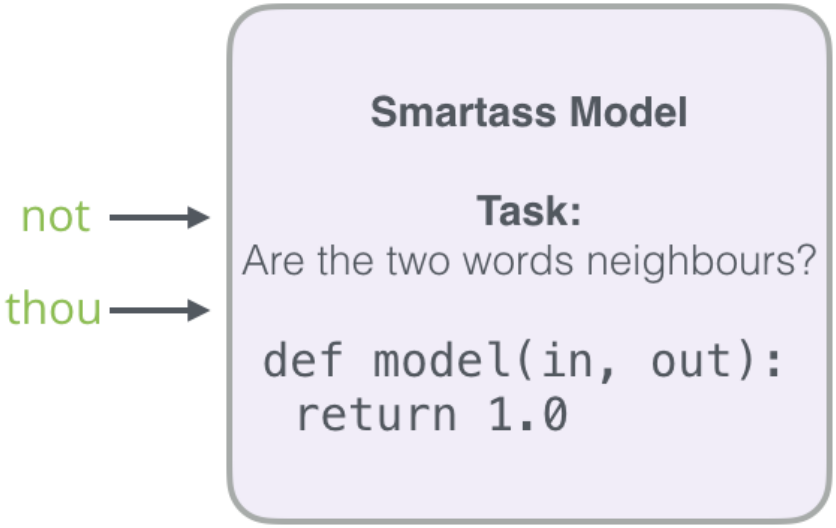
שינוי פשוט זה משנה את המודל שאנחנו צריכים מרשת עצבית, למודל רגרסיה לוגיסטית - ובכך הוא הופך להיות הרבה יותר פשוט והרבה יותר מהר לחישוב.

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.

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

input word	output word	target
not	thou	1
not	shalt	1
not	make	1
not	a	1
make	shalt	1
make	not	1
make	a	1
make	machine	1

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.

input word	output word	target
not	thou	1
not		0
not		0
not	shalt	1
not	make	1

Negative examples

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.

Pick randomly from vocabulary
(random sampling)

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	make	1

Word	Count	Probability
aardvark		
aarhus		
aaron		
taco		
thou		
zyzzyva		

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.

Skipgram

shalt	not	make	a	machine

input	output
make	shalt
make	not
make	a
make	machine

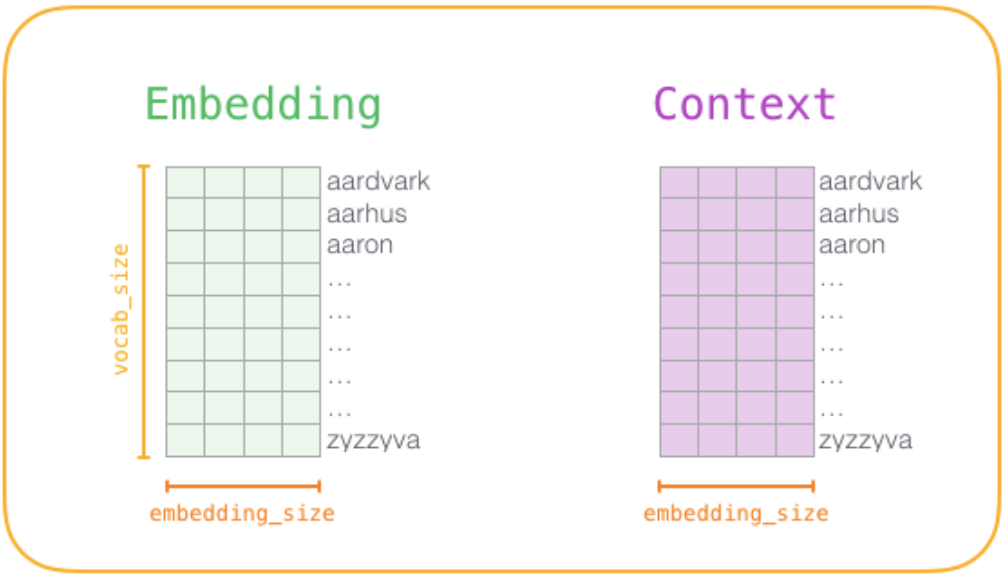
Negative Sampling

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0

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).

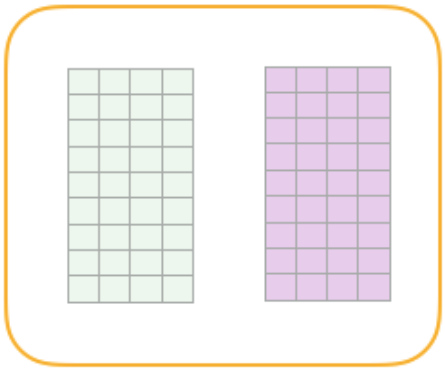


בתחילת תהליך ההכשרה, מאתחלים מטריצות אלו עם ערכים אקראיים. ואז מתחיל תהליך ההכשרה. בכל שלב אימון, לוקחים דוגמה חיובית אחת ואת הדוגמאות השליליות הקשורות אליה. ניקח את הקבוצה הראשונה:

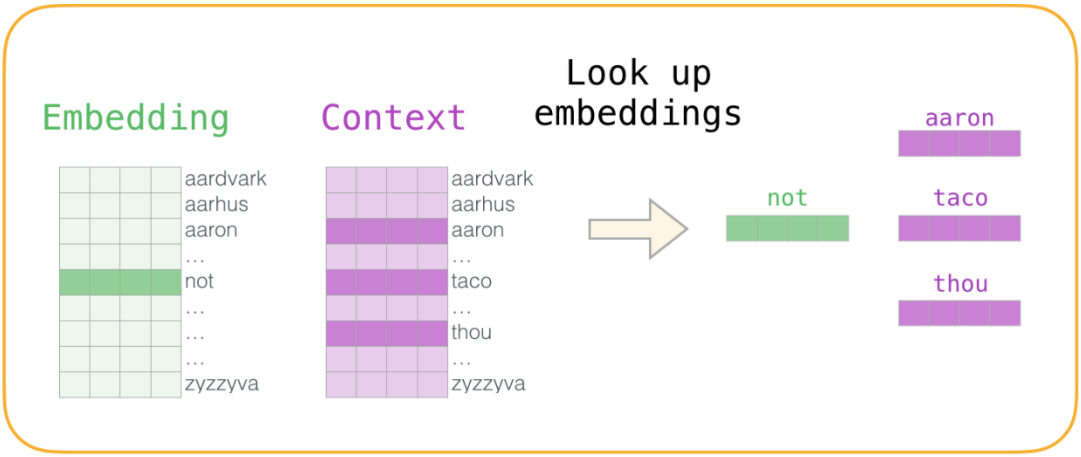
dataset

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...

model



עכשיו יש ארבע מילים: מילת הקלט not ומילות הפלט/ההקשר: thou (השכן בפועל), taco ו aaron (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

input word	output word	target	input • output
not	thou	1	0.2
not	aaron	0	-1.11
not	taco	0	0.74

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, המבצע הלוגיסטי.

input word	output word	target	input • output	sigmoid()
not	thou	1	0.2	0.55
not	aaron	0	-1.11	0.25
not	taco	0	0.74	0.68

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.

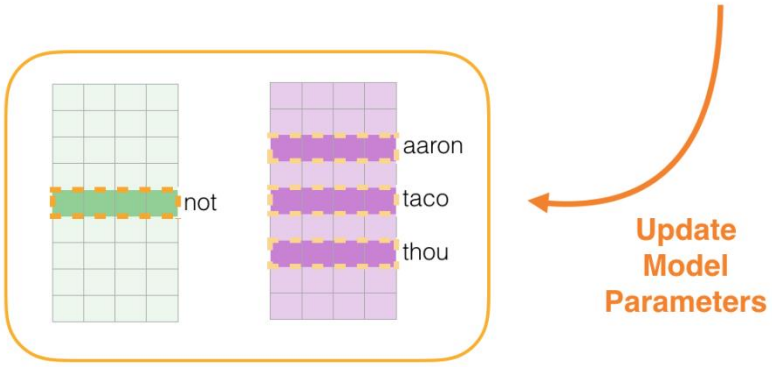
כעת, כשהמודל הלא מאומן ביצע חיזוי, ומאחר ויש תווית יעד ממשית להשוות מולה, מחשבים כמה שגיאה יש בחיזוי המודל. כדי לעשות זאת, פשוט להפחית את הציונים sigmoid מתוויות היעד.

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68

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.

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68

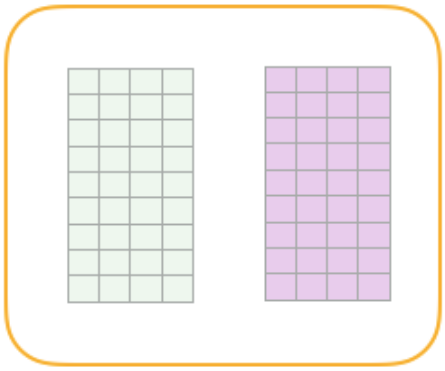


זה מסכם את שלב האימונים. יוצאים ממנו עם הטמעות מעט טובות יותר, עבור המילים המעורבות בשלב זה (not, thou, aaron ו taco). כעת אנו ממשיכים לשלב הבא (המדגם החיובי הבא והדגימות השליליות המשויכות אליו) ועושים את אותו תהליך שוב.

dataset

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...

model



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.

Window size: 5

Window size: 15

משימות שונות מוגשות טוב יותר על-ידי גדלי מסגרות שונים. היוריסטיקה 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).

Negative samples: 2

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0

Negative samples: 5

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0
make	finglonger	0
make	plumbus	0
make	mango	0

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:

<u>פונקצית guess</u>

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

<u>הפונקיה מקבלת:</u>

param clue: רמז נתון

param words: רשימה נתונה של מילים לנחש מהן

param n: מספר המילים המקסימלי לניחוש

הפונקציה מחזירה: 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]
```

פוקצית 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

<u>הפונקציה מקבלת:</u>

param words: רשימת מילים ליצירת רמז עבורן

param bad_words: רשימת מילים שיש להימנע ממתן רמזים עבורן

<u>הפונקציה מחזירה:</u>

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

<u>הקוד:</u>

```python
def give_clue(words, bad_words):
    # מחבר כל שתי צמדים אפשריות למילון ערך ה- key הוא tuple (מילה 1, מילה 2): וערך ה- value שלהן הוא ה-similarity המשותף
    similarities = {}
    if len(words) >= 2:
        for i in range(len(words)):
            for j in range(i + 1, len(words)):
                similarities[(words[i], words[j])] = model.similarity(words[i], words[j])

    # מחבר כל שלשות אפשרים למילון ערך ה- key הוא set (מילה 1, מילה 2, מילה 3): וערך ה- value שלהן הוא ה-similarity המשותף
    triple_similarities = {}
    if len(words) >= 3:
        seen = set()
        for w in words:
            for key in similarities.keys():
                z = key + (w,)
                if w not in key and tuple(sorted(z)) not in seen:
                    triple_similarities[z] = model.n_similarity([w], list(key))
                    seen.add(tuple(sorted(z)))

    # ריצת לולאה עד מציאת הרמז האופטימלי.
    while True:
        # אם נשארה מילה אחת לנחש או שנגמרו לנו קווי הדמיון הזוגיים אז:
        # נגדיר את max_correlated_n למילה שנותרה
        if len(words) == 1 or not similarities:
            max_correlated_n = (words[0],)

        # אם האורך של רשימת המילים היא 2, הגדר את max_correlated_n ל-max_correlated_pair
        elif len(words) >= 2:
            max_correlated_pair = max(similarities, key=similarities.get)
            max_correlated_n = max_correlated_pair

        # אם האורך הוא 3, הגדר את max_correlated_n ל:
        # אם הדמיון המשולש * 0.9 =< מהדמיון הזוגי הגדר ל-max_correlated_triple
        # אחרת הגדר ל-max_correlated_pair
        if len(words) >= 3:
            max_correlated_triple = max(triple_similarities, key=triple_similarities.get)
            if triple_similarities[max_correlated_triple] * 0.9 >= similarities[max_correlated_pair]:
                max_correlated_n = max_correlated_triple
            else:
                max_correlated_n = max_correlated_pair
        # הדפסת הרמז האפשרי בלוח הבקרה למעקב
        print("Giving clue for:", max_correlated_n)
        c_words = list(max_correlated_n)

        # מצא את המילים הדומות ביותר למילים ב-max_correlated_n
        clues = model.most_similar(positive=c_words, topn=10, restrict_vocab=10000)

        # ניקוי המילים הבעייתיות שנמצאו דומות
        clues_dict = dict(clues)
        cleaned_clues = [c[0] for c in clues if all([clean_clue(w, c[0]) for w in c_words])]

        # ריצה בלולאה כל עוד ש-cleaned_clues לא ריק או שנמצא רמז אופטימלי
        while cleaned_clues:
            # מצא את הרמז העדכני והטוב ביותר
            possible_clue = max(cleaned_clues, key=lambda x: clues_dict[x])

            # אם המשחק מתקרב לסיום דלג על סינון הרמזים
            if len(words) == len(max_correlated_n):
                return possible_clue, tuple(max_correlated_n)

            # מצא את המילה (הדומה ביותר) לרמז הנוכחי (הטוב ביותר) מ-bad_word
            enemy_match = model.most_similar_to_given(possible_clue, bad_words)

            # חשב את הדמיון בין השתיים
            enemy_sim = model.similarity(enemy_match, possible_clue)
```

```
    # ,max_correlated_pair-ב גדולה יותר בדמיון מכל המילים (enemy_match) אם המילה של האויב
    # הסרת הרמז הנוכחי הטוב ביותר מ-cleaned_clues והמשך באיטרציה
    # אם לא, החזר את הרמז הנוכחי, מכיוון שהוא אופטימלי
    optimal = True
    for n in max_correlated_n:
        if enemy_sim >= model.similarity(n, possible_clue):
            # הדפסת המילה שנפלה על איזה מילה בעיתית, ללוח הבקרה
            print("Foreign word " + enemy_match + " was too similar. Removing clue: " + possible_clue)
            cleaned_clues.remove(possible_clue)
            optimal = False
            break

    if optimal:
        return possible_clue, tuple(max_correlated_n)

# max_correlated_pair-כל הרמזים של האויב היו דומים יותר מאחת המילים ב
# אז הוצא את max_correlated_n מנקודות הדמיון התואמות והמשיכו באיטרציה
print("Too many enemy correlations. Removing ", max_correlated_n)
if len(max_correlated_n) == 2:
    similarities.pop(max_correlated_n)
elif len(max_correlated_n) == 3:
    triple_similarities.pop(max_correlated_n)
```

קוד התוכנית

בנית לוח משחק

```
def generate_board(word_list):
        used = set()
        red = []
        blue = []
        neutral = []
        assassin = []

        # Generate 9 random words for red team.
        while len(red) < 9:
            index = random.choice(range(len(word_list)))
            word = word_list[index]
            if index not in used:
                red.append(word)
                used.add(index)

        # Generate 8 random words for blue team.
        while len(blue) < 8:
            index = random.choice(range(len(word_list)))
            word = word_list[index]
            if index not in used:
                blue.append(word)
                used.add(index)

        # Generate 7 random neutral words.
        while len(neutral) < 7:
            index = random.choice(range(len(word_list)))
            word = word_list[index]
            if index not in used:
                neutral.append(word)
                used.add(index)

        # Generate assassin word.
        while not assassin:
            index = random.choice(range(len(word_list)))
            word = word_list[index]
            if index not in used:
                assassin.append(word)
                used.add(index)
        board = red + blue + neutral + assassin
        random.shuffle(board)
        board = np.reshape(board, (5, 5))
        print(red)
        return board, red, blue, neutral, assassin
```

```
class Game():
    def __init__(self):
        self.idGame=1
        self.queue = []
        self.groupRed=Group("red")
        self.groupBlue=Group("blue")
        self.bourd=bourd()
        self.platerNow=0
        self.grupNow="red"
        self.Humans=0
    def addPlayer(self,role,name,color,isHuman):
        if isHuman == True:
            self.Humans = self.Humans + 1
        if role=="multi-spy":
            if color=="red":
                if (self.groupRed.players and
self.groupRed.players[0].role == "multi-spy") or
len(self.groupRed.players)>1:
                        return False
            else:
                if (self.groupBlue.players and
self.groupBlue.players[0].role == "multi-spy")or
len(self.groupBlue.players)>1:
                        return False
        else:
            if color == "red":
                if self.groupRed.players and
self.groupRed.players[0].role == "spy"or
len(self.groupRed.players)>1:
                    return False
            else:
                if self.groupBlue.players and
self.groupBlue.players[0].role == "spy"or
len(self.groupBlue.players)>1:
                    return False

        player=Player(role,name,color,isHuman)
        self.queue.append(player)
        if player.color=="blue":
            self.groupBlue.players.append(player)
            self.groupBlue.words=self.bourd.blue

self.groupBlue.bad_words=self.bourd.red+self.bourd.neutral+self.bourd
.assassin
        else:
            self.groupRed.players.append(player)
            self.groupRed.words = self.bourd.red
            self.groupRed.bad_words = self.bourd.blue +
self.bourd.neutral + self.bourd.assassin
        return True
    def orderly_queue(self):
        global temparr
        temparr = []
        for i in range(len(self.queue)):
            temparr.append(self.queue[i])
        for i in range(len(self.queue)):
            if self.queue[i].role == "multi-spy":
                if self.queue[i].color == "red":
                    temparr[0] = self.queue[i]
                elif self.queue[i].color == "blue":
                   temparr[2] = self.queue[i]
```

```
            elif self.queue[i].role == "spy":
                if self.queue[i].color == "red":
                    temparr[1] = self.queue[i]
                elif self.queue[i].color == "blue":
                    temparr[3] = self.queue[i]
        for i in range(len(self.queue)):
            self.queue[i]=temparr[i]
    def Next_pleyer(self):
        if self.platerNow<3:
            self.platerNow+=1
        elif self.platerNow==3:
            self.platerNow=0

    def Pressed_word(self,word,player):
        if word in self.bourd.red:
            self.bourd.red.remove(word)
            if len(self.bourd.red):
               return "red"
            else:
                return "The red team wins"

        if word in self.bourd.blue:
            self.bourd.blue.remove(word)
            if len(self.bourd.blue):
               return "blue"
            else:
                return "The blue team wins"

        if word in self.bourd.neutral:
            self.bourd.neutral.remove(word)
            return "neutral"
        if word in self.bourd.assassin:
            if player in self.groupBlue.players:
                return "The blue team wins"
            elif player in self.groupRed.players:
                return "The red team wins"
```

פונקצית עזר לפונקצית Give_Clue

```
def clean_clue(self,word1, word2):
    engine = inflect.engine()
    word1 = word1.lower()
    word2 = word2.lower()
    return not (word1 in word2 or word2 in word1 or "_" in word2 or
word2 == engine.plural(word1) or len(
        word2) <= 2)
```

תיאור מסד הנתונים

מסד נתונים השמור בקובץ word.txt המכיל 200 מלים ללוח המשחק.

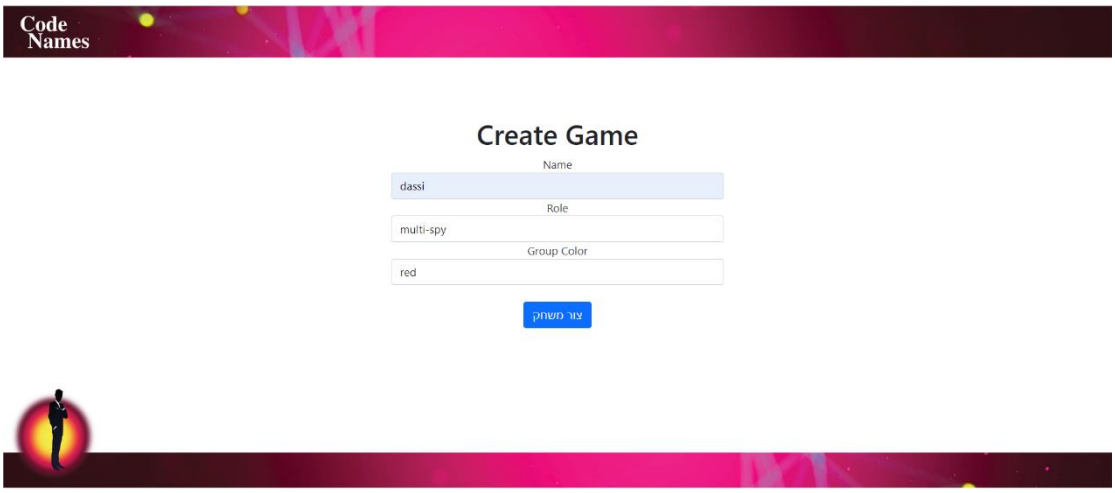
תיאור מסכים

פרוט מסכים

Create game

התחלת משחק חדש. הכנסת נתונים, שם, תפקיד וצבע קבוצה.

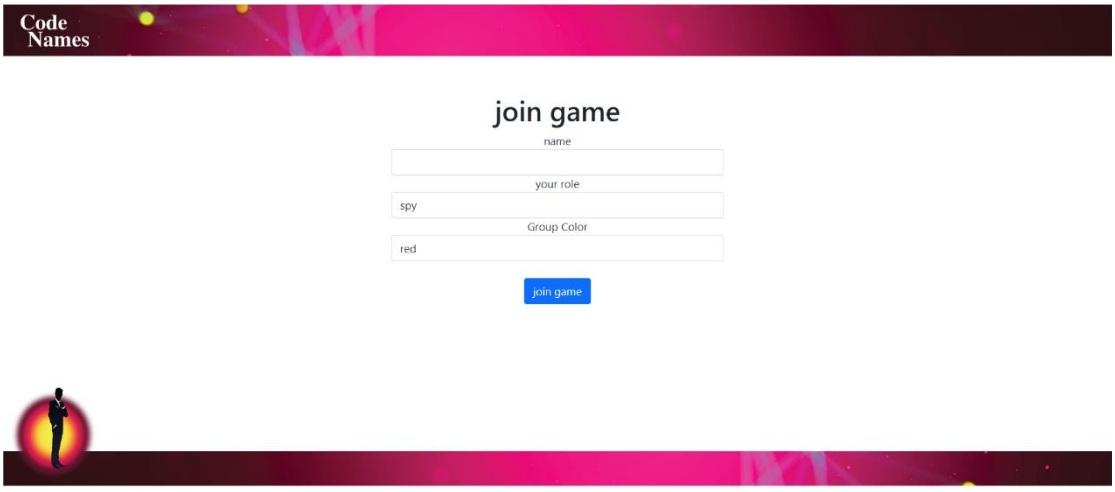
לחיצה על "צור משחק" מעבירה למסך השחקנים.



Join game

הצטרפות למשחק. הכנסת נתונים, שם, תפקיד, סיסמה (guid) וצבע קבוצה.

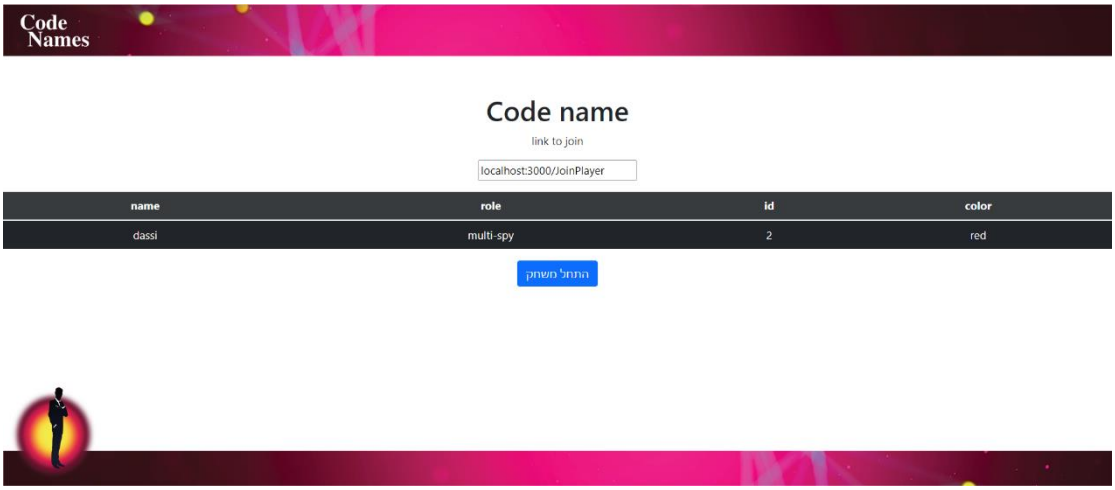
לחיצה על "הצטרף למשחק" מעבירה למסך השחקנים.



Player

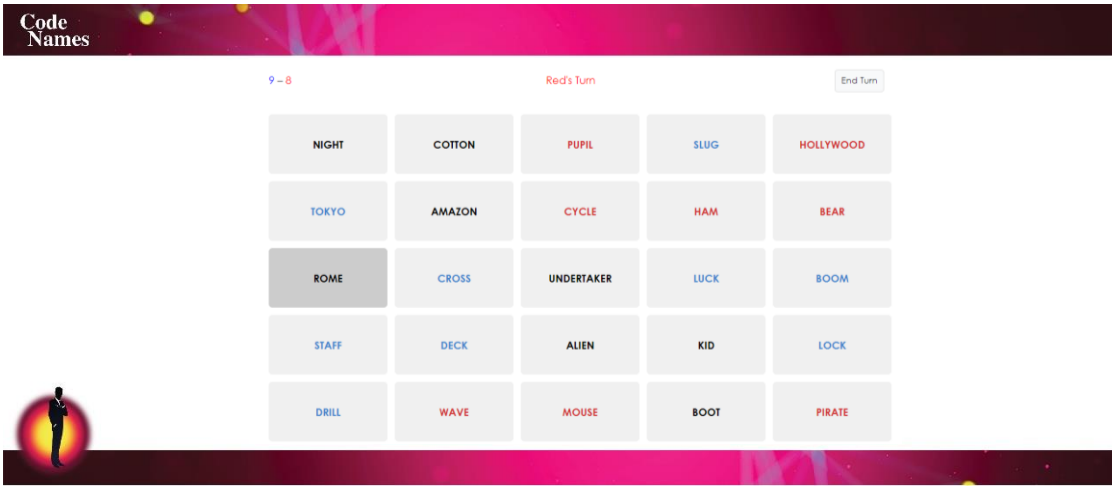
נתוני השחקנים המשתתפים במשחק.

לחיצה על "התחל משחק" מעבירה ללוח המחשק.

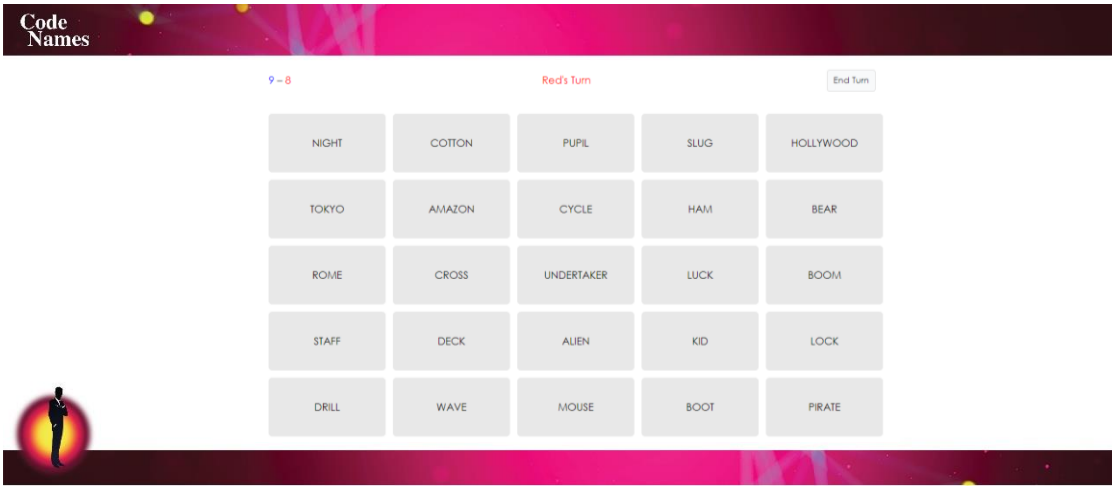


Board

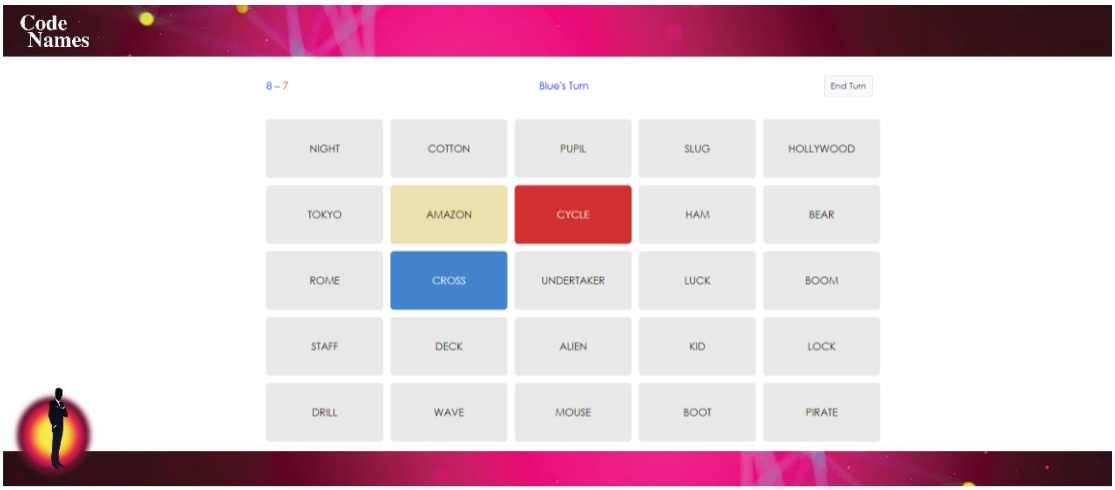
לוח משחק - מפעיל



לוח משחק מרגל



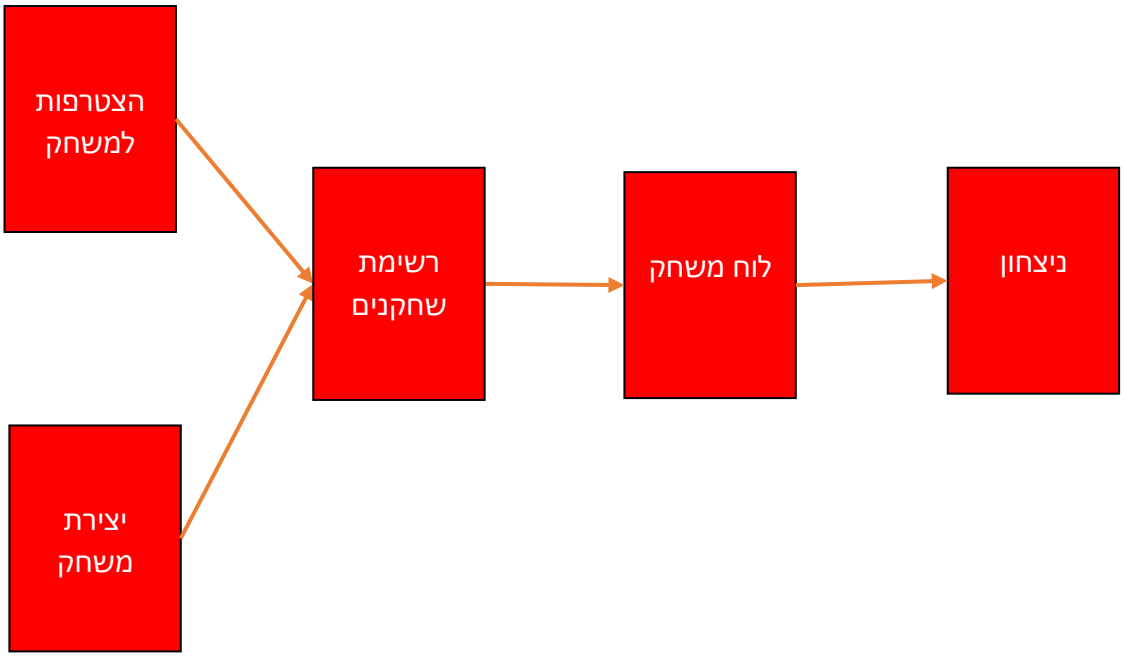
משתנה במהלך המשחק, כל מילה ש"הוצבעה" – לחצו עליה, נצבעת בצבע המתאים ומספור הנקודות.



winner

החרזת הקבוצה המנצחת וסיום המשחק.

תרשים מסכים



מדריך למשתמש

המשתמש יוצר משחק חדש ומזמין חברים למשחק, בכמות שבוחר מ-0 חברים עד 3 חברים.

והמשחק "שם – קוד" מתחיל...

המשחק מתקיים בין 2 קבוצות, אדומה וכחולה. בכל קבוצה אחד השחקנים משמש כרב-מרגלים, ויתר השחקנים הם מרגלים.

לוח המשחק מכיל 25 מילים המהוות את רשת הסוכנים. רבי המרגלים מקבלים מסך גלוי של רשת הסוכנים, כשכל מילה יכולה להיות כחולה, אדומה, אזרח תמים או מתנקש. מטרת המשחק היא לזהות את כל הסוכנים בצוות שלך, כשיש 9 לקבוצה המתחילה ו8 לשנייה.

בתחילת התור רב-המרגלים נותן רמז למציאת הסוכנים בצוות שלך המכיל, מילה ומספר, זה ה"שם קוד". המילה לא יכולה להיות זהה למילה שמופיעה על אחד הקלפים שבלוח המשחק – שעדיין לא גילו אותם. המספר מתאר כמה סוכנים מתאימים ל"שם קוד" – הרמז, והוא שווה לכמות הקלפים שהמרגלים יכולים לחשוף.

צריך להיזהר בבחירת שם הקוד, מכיוון שהוא יכול להתאים ליותר קלפים ממה שרב-המרגלים התכוון אליהם. והמרגלים שלו יכולים להתבלבל ולבחור בקלפים לא נכונים. אם המרגלים בוחרים בקלף שהוא סוכן של הקבוצה השניה, הקבוצה השניה מקבלת אותו והתור של הצוות הנוכחי מסתיים. אם זה עובר-אורח, התור של הצוות גם מסתיים ולבסוף אם זהו המתנקש, הצוות מפסיד מידית. וניצחון לצוות השני.

המשחק מסתיים בקבוצה שגילתה ראשונה את כל חברי הצוות שלה, או שצוות כלשהו גילה את המתנקש והניצחון בידי הצוות הנגדי.

לדוג' הקבוצה האדומה הצליחה לזהות את כל המילים - חברי הצוות שלה וזכתה בניצחון. או שהקבוצה הכחולה זיהתה את המתנקש, והקבוצה האדומה נצחה (אפילו שלא זיהתה את כל חברי הצוות שלה).

בדיקות והערכה

הרצתי את פונקציית מציאת הרמז והתוצאות היו מדהימות

המודל החזיר תשובות נכונות והכי יפה היה לראות את הדרך שעשה עד מציאת הרמז הטוב ביותר, איך לא נפל על מילים אחרות.

הינה דוגמה קטנה

Giving clue for: ('horseshoe', 'park')

Foreign word beach was too similar. Removing clue: lake

Foreign word beach was too similar. Removing clue: recreation

Foreign word beach was too similar. Removing clue: recreational

Foreign word beach was too similar. Removing clue: beach

Foreign word compound was too similar. Removing clue: tent

Clue is grass

בתרגום חופשי המערכת רוצה לתת "רמז" למילים 'פרסה' , 'פארק'

המערכת נפלה מהרמז – אגם, בילוי, פנאי, חוף, מהמילה האסורה – חוף

בסוף ניתן הרמז – דשא.

ניתוח יעילות

הפונקציה guess רצה על רשימת המילים בלוח – משתנה קבוע (25). – O(1)

הפונקציה give clue רצה עד שהיא מוצאת רמז על מקסימום 25 מילים. – O(n)

מסקנות

מסקנה ראשונה ועיקרית שניתן להסיק מהעבודה על הפרויקט, היא שהטקטיקה של מציאת קוד קיים במאגרי האינטרנט, והתאמתו לצרכים מסויימים- היא טקטיקה נצרכת וחשובה לא פחות מאשר פיתוח קוד באופן עצמוני.

אם אנו מנסים לייעל את העבודה כמה שיותר, אין סיבה שלא נשתמש בקודים, מסקנות או תובנות של אנשים שכבר עברו את אותה דרך שאנו מתכננים לעבור, או דרך דומה, וכבר הגיעו לתוצאה. לכן, חיפוש נכון של חומר, והבנת הקוד על מנת להתאים אותו לצרכים מסויימים- הן מיומנויות חשובות ונצרכות, וכתיבת הפרויקט בהחלט תרמה לפיתוחן.

מסקנה נוספת, היא חשיבות השמירה על הסדר במהלך העבודה. עבודה מסודרת, עם שמות משמעותיים לקבצים ותיקיות, וחלוקה לוגית בין התיקיות- מקלה לעין ערוך על מהלך העבודה.

והמסקנה העיקרית: מטרת הפרויקט היא התהליך, הלמידה וההתנסות, ולא התוצאה. היו שורות פשוטות, של טכניקה בלבד, שניצב הצורך רק למצוא את השורה המתאימה - והחיפוש אחריה ארך שבועות ארוכים, והיו קודים שדרשו חשיבה מעמיקה. בשני המקרים, הדרך בעצמה- היא זו שתרמה לי צבירת ניסיון. ההתמודדות מול עבודה רבה, עם חומר לא מוכר - לימדה אותי ותרמה לי רבות. בכללי כל עולם ה-AI גדול ומרתק ושואב, ההתנסות בפועל בו תרמה לי הרבה בהשמת הידע שרכשתי, במהלך לימודיי.

פיתוחים עתידיים

מודל word2vec בעברית

בבליוגרפיה

- ויקיפדיה
- YouTube
- GitHub
- https://www.tensorflow.org/tutorials/text/word2vec
- https://machinelearningmastery.com/develop-word-embeddings-python-/gensim
- https://www.analyticsvidhya.com/blog/2021/06/practical-guide-to-word-/embedding-system
- https://www.analyticsvidhya.com/blog/2021/07/word2vec-for-word-/embeddings-a-beginners-guide
- Word Embeddings in Python with Spacy and Gensim | Shane Lynn
- https://www.hamichlol.org.il/%D7%A2%D7%99%D7%91%D7%95%D7%93_%D7%A9%D7%A4%D7%94_%D7%98%D7%91%D7%A2%D7%99%D7%AA
- https://arxiv.org/abs/2010.01309
-