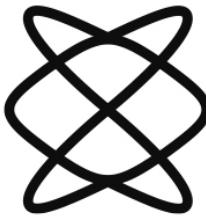


החולג למדעי המחשב (0368)
עיבוד שפה טبעית (3077)
Natural Language Processing
(קורס ארוכה)

מרצה: מאור איבגי
(תשפ"ד, סמסטר ב')

[יונתן ברנט]
(תש"ף, סמסטר א') (2020)

مسכום: רועי מעין



The Raymond and
Beverly Sackler Faculty
of Exact Sciences
Tel Aviv University

**פרק 1 – מודלים בסיסיים**

3.....	סביבת עבודה
14.....	מבוא
14.....	יצוג מיללים
1.....	מודלי שפה

פרק 2 – שיטות מודרניות

31.....	רשתות נוירונים
45.....	טראנספורמרים

1 – אבני היסוד

סביבה לעבודה

Colab and Latex

מטלה 0

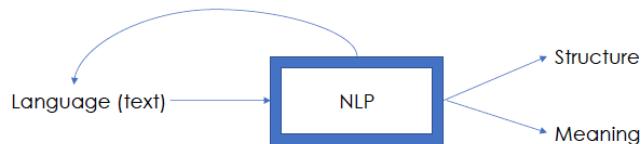
Hugging Face

סרטון הסבר: <https://www.loom.com/share/87a1378f9c674ea5bceebc21b0f90097>

מבוא

מה זה NLP?

NLP (Natural Language Processing): תחום עיבוד השפה הטבעית, עוסק בפיתוח שיטות שמקבלות קלט טקסט בשפה טבעית, ומנסנות להבין את המבנה שלו ואת המשמעות שלו. בשילוב בין השניים, ניתן להסיק מה המידע שהテקסט מעביר, ובתקווה לקחת אותו בחזרה לצורה של שפה טבעית תוך יצירת טקסט חדש.



- לרוב נוח לבצע NLP מול השפה האנגלית (המאמר האקדמיים, המונחים, במות הדוברים, במות הדטאא...), אך יש לתמוך בשילול שפות. גם בשפה נתונה, יש הרבה מקרים שונים לטקסטים (חידושים, רשותות חברתיות, ספרות) המשפיעים על אופי הטקסט ועל העיבוד שלו.

- משמעותות שונות ב-NLP דורשות יחס שונה והגדרות שונות כדי לטפל בהן. נשים לב כי שם הקורס הוא עדין "NLP", כאשר מונח זה משמש כ-term umbrella עבור AI כללי עבור שפה. ה-N נזכר לא מתמקד בשפה טבעית (גם בשפות אחרות, כמו קוד, גנטיקה). ה-P גם נשאר באחוריו 2010, ועודשים גם NLG (understanding), או NLG (generation). אנחנו עדים רוצים להפריד עיבוד שפה טבעית מ-CL (בלשנות חישובית) שפחות מבוססת על ML (כבר לא ממש...) וועשה דברים בצורה יותר מסורתית, החלטנו להישאר עם המונח NLP.

מטרות העל בתחום:

- לגרום למחשבים **להבין** קלט בשפה טבעית שmagiv מבני אדם – בקשות, שאלות ופקודות.
- לגרום למחשבים **לחלץ** מידע רלוונטי משלל המידע שקיים באינטראקטן.
- לגרום למחשבים **לייצר** פלט בשפה טבעית ורحتה, בהתבסס על כל המידע הנגיש שהקיים היום. נשים לב כי אחור זה מעלה שאלות נוספת – עד כמה הטקסט שנוצר הוא שימושי? אמיתי? לא פוגעני?

מודלים מובילים:

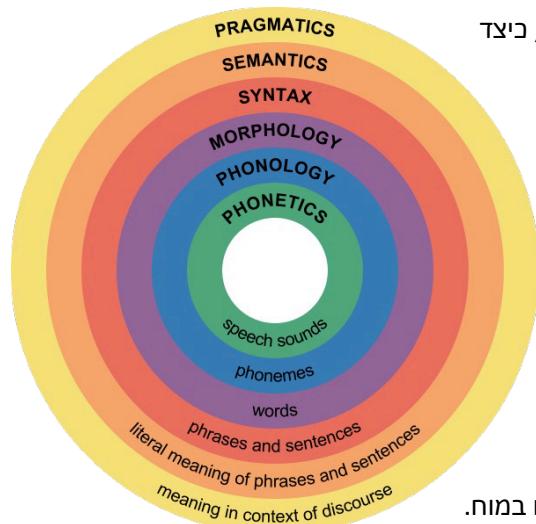
Models - Hugging Face

- התחל בחייבה עבור Pytorch, Transforms, Pytorch, ומשם זה התחיל להתרחב לשירותים שונים.
- משמש בספריה עבור מודלים שונים של Deep Learning, לא רק לשפה, גם ל-visions. בנוסף, מספקים datasets שונים, שיטות לאבלואציה וכו'.

ChatGPT

בלשנות ("אני יודע שזה קצת משעמם (?) , וهم היום (?) , אבל זה חשוב להבין לפחות פעמיים אחד בחיים מאיפה אנחנו מגאים ?) ; וכמה הקורס הזה השתנה ב-3 השנים האחרונות ...") :

NLP שונה מ-CL (בלשנות חישובית), כאשר בלשנות היא המדע הקוגניטיבי **שחוקר שפה**. בניתו בלשוני ישן רמות שונות:



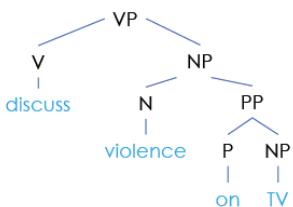
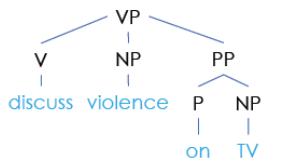
- פונטיקה/פונטיקה** – הצלילים שקיימים בשפה והחוקים השונים שפועלים עליהם, כיצד צלילים מסוימים אחד על השני. מתחבר לתחום עיבוד אודיו (TTS, STT).
- מורפולוגיה** – המבנה הפנימי של מילים, והדרך לייצר אותן.
- תחביר** – המבנה של משפטים, כיצד המילים משפיעות זו על זו, ותלותות זו בזו.
- חלוקת לרכיבים, עצי גזירה (לפי חוקי דקדוק פורמלי כלשהו).
- סמנטיקה** – משמעות של מילים בעולם.
- פרגמטיקה** – השימוש בשפה בהקשר, בחלוקת משיחה, העברת מסרים.
- שיח** – יחסיים בין משפטים בטקסט בכלל, מציאות נרטיב.

תכונות של שפה טבעית:

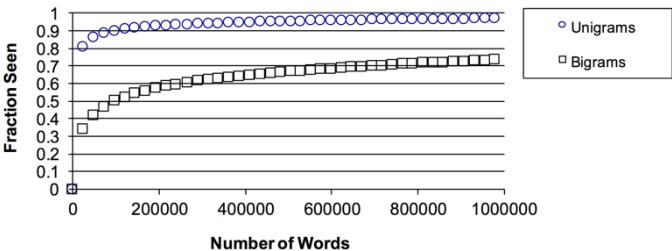
- משמשת לתקשורת ולהעברת מידע, נוצרה על ידי בני אדם למטרה זו.
 - nl마다 מניסיון (על בסיס קלט מהסבירה) – ילדים רוכשים שפה בגיל מוקדם.
 - שפה היא מערכת דיסקרטית – המילים מגיעות מסווג סופי, אחת אחרי השניה.
- אמנם, היא מקודדת באופן של סיגנלים רציפים: צלילים, מחוות וshape גוף, סיגנלים במוח.

אתגרים בשפה טבעית:

discuss violence on TV



(many-to-one).).



בעיית Sparsity – רוב מילים יופיעו פעם אחת בלבד (bigram). מילון אינטראקטיבי (train/test) יכיל מילון אחד בלבד. מילון אינטראקטיבי (train/test) יכיל מילון אחד בלבד.

4. **חיבור לעולם האמתי (Grounding)** – בני אדם חשופים לשפה ממקורות ומחושים רבים – הקלט מהסביבה, השמעת כל ה-features שהמודל אמר לו למצאו בהם כדי למצוות ההיפותזה הנכונה, לעולם לא נראה באימוניו לדוגמה, אין מודל יידע לנתח רגש כמו "not all bad" או אפילו רק "bad all" אבל לא מעבר לה.

1. **עמימות (Ambiguity)** – לאוטו משפט יכולות להיות משמעותיות שונות (one-to-many) – בבחינה תחבירית (עדי גירה) שונות, חיבור רכיבים בצורה שונה). יכולה להיות גם עמימות בפרשנות הפונולוגית שאנו נוהנים לרצף צלילים ("like your" vs. "lie cure"). ישנה גם עמימות סמנטית בהקשרים שונים (קיים, כלל).

2. **שונות (Variability)** – יש לנו הרבה אפשרויות לבטא את אותה משמעות באמצעות מילים שונות (bigram).

3. **דלילות (Sparsity)** – בהינתן משפט "I ate an apple" ניתן להסתבל ברמת מילה בודדת (Unigram):

"I", "ate", "an", "apple" או בrama של זוגות מילים [Bigram]: ("I ate", "ate an", "an apple"). באופן כללי נבנה את הפריטים **n-grams**. לדוגמה, ניקח טקסט מהאינטרנט, נחלק ל-train/test, כאשר נüber על ה-test נבדוק כמה פעמים הפריטים נראו ב-train. רוב המשפטים לא נראו מעולם. ככלمر אנחנו בודקים כאן מה **coverage** של

ה-train שלנו על פני המילים השונות, מה הסיכוי שהן יופיעו. לרבות מילון אינטראקטיבי (train/test) יכיל מילון אחד בלבד. מילון אינטראקטיבי (train/test) יכיל מילון אחד בלבד.

History

ההיסטוריה של התפתחות החישובית עד ל-NLP בימינו:

- Turing – העלה את השאלה "האם מכונות יודעות לחשב?", הגידר את מבחן טירינג: האם מחשב יוכל לשמש באדם ולגרום לו לחשב שהוא אדם ולא מכונה.
- Weaver – בלשן שהציע לחשב על הבעה של תרגום מכונה, במובנים של קידוד ותורת האינפורמציה.
- Chomsky – "אלוחי הבלשנות" שלו, הציע את הדקדוק האוניברסלי (UG) כהסבר לדעת הבלשני. ההסבר זה נמצא עד היום בחלוקת אל מול גישות סטטיסטיות יותר של רכישה. לקריאה נוספת:

[A Critical Discussion of Universal Grammar : r/linguistics \(reddit.com\)](#)

[r/linguistics \(reddit.com\)](#)

- שני מבטים (שונים מכך) על שפה | לא למות טיפש(idanlandau.com)
- מערכות מבוססות חוקים (שנות ה-70 וה-80) – שימוש בדקדוקים, מערכות מבוססות חוקים. המערכות הללו מביאות איתן בעיות גם של overgeneralization (מבלילות מידיו ויצירת משפטים שאינם בשפה), וגם undergeneralization (לא מייצרת מילים שכן בשפה).
- chatbot Eliza IBM יצרה את שנקרא Eliza – מילויים מיידי ויצירת משפטים שאינם בשפה, ופומביים (לא מילויים שכן בשפה).
- שיטות סטטיסטיות מבוססות קורפוסים (שנות ה-90) – במקומם להגדר באופן פורמלי ק-טומ bottom-up או השפה עובדת, עוסקים עם קורפוס גדול ומנתחים את הסטטיסטיות השונות בתוכו.
- פורמליזמים מבניים עשירים (שנות ה-2000) – עוסקים עם עצים תחביריים שלמים במקביל לסטטיסטיות.
- רשותות נוירונים (2010) – בהינתן הרבה DATA, מייצרים יציג של השפה באמצעות קטורום, ומאמנים מודל.
- שימוש בטרנספורמרים (2020) – השינוי הגדול ביותר עד כה.

חומרים נוספים:

הרצאה: [NLP 20A \(JB\) - 1: Intro](#)

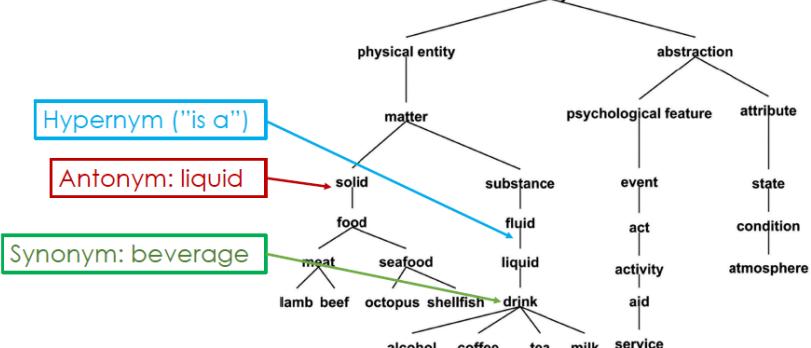
יצוג מילויים

Word Representations

משמעות (meaning): מה שאנו מנסים להבהיר, מה שצליל/מילה מסמלים. ב-sentential semantics מדברים על אובייקט signified (המשתנה, למשל «עז») ואובייקט signifier (הREFERENCE, הדבר, העצם).

- את ההתחאה הזאת ניתן למצוא במילון. הבהה היא שאפשר לשלב בין מילים, המשמעות שלן משתנה בהקשר. בנוסף, כל הגדירה במילון מורכבת ממילים שגם הן מוגדרות במילון (זה מעגלי).

- מאיץ גדול שהיה בעבר נקרא WordNet – מאגר גדול של מילים, כולל הקשרים השונים ביניהם. לשימוש במאגר זה יש מספר חסרונות:



- יקר ליצר עבור שפות נוספות, שם מבונן היחסים בין המילים יכולם להיות שונים.
- סובייקטיבי, צריך מיללים לתוך המאגר הזה.
- סטטי, תמיד מתווסף מילים לשפה והמאגר צריך להתחאים את עצמו.
- חסירה פונקציונלית – באיזו מידת מילים דומות? איך מרכיבים משמעות מכמה מילים?

דוגמה (sentiment analysis): מקבלים טקסט, שמורכב מביקורות לסרטים. علينا להכריע האם הביקורת חיובית או שלילית, מה הרגש שモבע ב ביקורת. נרצה לאמן מודל שידע לבצע את הסיווג הזה.

נwick את המודל הנפוץ: $(x \cdot w) sign = y$, כאשר x יהיה וקטור פיצ'רים שמייצג את הקלט שלנו, ו- w הם משקלים. אנחנו יודעים למצוא את המשקלים המתאימים ביותר לבעה (SVD, least-squares, וכו'). איך נבנה את x ? האורך שלו משתנה ממשפט למשפט כי כמות המילים משתנה! לכן ניצג כל מילה w_i וניעזר בפונקציה כלשהי: $f(w_1, \dots, w_n) = x$.

- נגיד לקסיקון (vocabulary) \mathcal{V} – קבוצה ממוינת של מילים. נסמן $C = |\mathcal{V}|$.
- ניצג כל מילה באמצעות one-hot encoding, ניצור וקטור מממד C שבו הבינה ה- i היא 1 וכל שאר הביניות ה- 0 ,
- באשר i הוא האינדקס של המילה בלקסיקון: $\{0, 1, \dots, C\} \in e_i$.
- ניעזר בשיטת Bag of Words (BoW) וニיצג משפט באמצעות חיבור של הוקטורים, וכך ישאר באותו מימד. אם נתחשב בכמות הפעמים שאוותה מילה מופיעה במשפט, לשם כך נctruct k-hot embedding.

חיסרון – כיצד נבעא את העובדה שיש "קרבה" מסוימת גבואה יותר בין המילים movie/film/meal/motion מאשר בין המילים father/mother. נזכיר את הוקטורים אורותוגונליים, המכפלת הסקלרית (dot product) ביןיהם תמיד מינימום 0: $A \cdot B = \|A\| \|B\| \cos \theta$. המטריה שלנו היא למעשה דרך נוספת לקודד את המילים בוקטורים, כך שנוכל לתפוס את הדמיון בצורה טבעיות.

Distributional similarity

word	context
father	{house: 3, mother: 4, children: 5, kids: 1, ...}
mother	{house: 3, father: 5, children: 6, kids: 1, ...}
dad	{home: 3, mom: 5, children: 1, kids: 6, ...}

במקום לנסתות להבין מה המשמעות של מילה, ננסה להבין באיזה הקשר (context) היא מופיעה, ומבחן נגזר את המשמעות שלה בפועל. הניסיון הראשון שלנו יהיה co-occurrence matrix: כביכול בינה את קורפוס, נספר כמה פעמים מופיעה כל מילה בהקשר של כל מילה אחרת. כל מילה תוצג על ידי השורה המתאימה לה. נוכל לנרטל כל שורה כדי לקבל טווח ערכים בין 0 ל-1. הבעה היא שלມשל המכפלת בין father ל-dad לא תעיד על הדמיון בין המילים, כמו home ו-house.

$$\text{Father} = \begin{bmatrix} 0.278 \\ 0.545 \\ 0.355 \\ 0.002 \\ -0.048 \end{bmatrix} \quad \text{Mother} = \begin{bmatrix} 0.298 \\ -0.615 \\ 0.355 \\ 0.003 \\ -0.051 \end{bmatrix} \quad \text{Dad} = \begin{bmatrix} 0.268 \\ 0.355 \\ 0.555 \\ 0.003 \\ -0.051 \end{bmatrix}$$

לכן, במבנה וקטור דחוס יותר (מממד נמוך) לכל מילה, ברור שהוא דומה יותר לאמה ולבבוריים של מילים ש모ופיעות בהקשרים דומים, כאשר נמדד את הדמיון באמצעות dot product. נלמד אותם ישירות מהדעתה לפי – distributed context. נבנה אותם embeddings, יציגים שהם context שליהם. נבנה אותם context.

Word Vectors

רעיון על למידה מפוקחת: הקלט לבעה הוא training set IID מעל התפלגות בלשיה: $\{(x_i, y_i)\}_{i=1}^N$, זוגות של קלט והטיפול המתאים לו, כאשר הפלט הוא מסוג: $y \rightarrow \mathcal{X}$.
הערה: לרוב התוצאות מגעימים מתייג אונשי, באנו עושים דבר קצת שונה המבונה **self-supervision**, אשר y נלקח מהדעתם עצמו שהוא מלכתחילה unlabeled. אנחנו בהתחלה פורטים proxy task, שבתקווה חלק מהפתרונות שלו נרכוש כלים (יצוגים של מילים) שייעזרו לנו לפחות בעיה אמייתית ומורכבת יותר (כמו sentiment analysis).

(Skipgram) Word2Vec: אלגוריתם זה **מנבא מילים על סמך שכנות** – בהינתן מילה, אילו מילים יכולות להופיע לידה. זו ממשית self-supervision, שמקבלת כ-set train משפט וצמידים (y, x) כאשר x היא מילה, ו- y היא מילה שיכולה להופיע ליד x .

"… that bank holds the mortgage on my home..."

1. $(x, y) = (\text{bank}, \text{that})$
2. $(x, y) = (\text{bank}, \text{holds})$
3. $(x, y) = (\text{holds}, \text{bank})$
4. $(x, y) = (\text{holds}, \text{the})$
5. $(x, y) = (\text{the}, \text{holds})$

נגדיר מודל שמייצר את ההסתברות שפלט מסוים (context) יופיע בהינתן מילת קלט (center) – התוצאה היא אוסף של הסתברויות, עברו כל מילה בנפרד. למשל ("that"|"bank"). בניתוח הקלט "bank" נרצה שההסתברות שהפלט יהיה "that" תהיה כהה שיותר גבוהה (בשאייפה 1-1). פעולה זו מכונה גם **softmax** ומודדרת כך:

$$p_{\theta}(o|c) = \frac{e^{u_o^T v_c}}{\sum_{w \in V} e^{u_w^T v_c}}$$

Dot product compares similarity of o and c .
Exponent makes everything positive
Normalize over entire vocabulary to give probability distribution

באשר:

- $o \in \mathbb{R}^d$ – וקטור עבור מילה "חיצונית" (outside/output/context) – ה- y שלנו. מטריצה U תיצג את הוקטורים הללו.
- $c \in \mathbb{R}^d$ – וקטור עבור מילה "מרכזית/פנימית" (center) – ה- x שלנו. מטריצה V תיצג את הוקטורים הללו.
- הלексיקון שלנו הוא \mathcal{V} , ומספר המחלקות הוא $|\mathcal{V}|$.
- הפרמטרים שלנו הם θ , ומספר הפרמטרים הוא $d \cdot |\mathcal{V}| = 2$.

נשים לב כי אנחנו עובדים עם **שתי מטריות** U, V (אחד לוקטורים של מילים חיצונית, והשנייה לוקטורים של אותן המילים כאשר הן מופיעות במילה פנימית). אם היינו עובדים עם מטריות אחת בלבד, אז לכל מילה תוכאת המכפלה של הוקטור שלה עם עצמו, תמיד יגרום לכך שהיא המילה עם ההסתברות הכי גבוהה.

נגדיר פונקציית מטריה: $\mathcal{L}(\theta) = \prod_{t=1}^T \prod_{j=-m}^m p_{\theta}(w_t | w_{t+j})$. זו הפונקציה שנרצה למינימום. לכל משפט שניינו להרכיב מוקורפוס שלנו (שהוא באורך T), נסתכל לכל word, center word, על כל המילים השכנות בחולון בגודל m .

$$\mathcal{L}(\Theta) = \prod_{c,o} p(o | c)^{\#(c,o)}$$

We can solve separately for each center word \hat{c}

$$\mathcal{L}_{\hat{c}}(\Theta) = \prod_o p(o | \hat{c})^{\#(\hat{c},o)}$$

solve for

$$J_{\hat{c}}(\Theta) = \sum_i \#(\hat{c}, o_i) \log p(o_i | \hat{c})$$

$$\text{s.t. } \sum_i p(o_i | \hat{c}) = 1, p(o_i | \hat{c}) \geq 0$$

Use lagrange multipliers:

$$L(\Theta, \lambda) = \sum_i \#(\hat{c}, o_i) \log p(o_i | \hat{c}) - \lambda \left(\left(\sum_i p(o_i | \hat{c}) \right) - 1 \right)$$

$$\nabla_{p(o_i | \hat{c})} L = \frac{\#(\hat{c}, o_i)}{p(o_i | \hat{c})} - \lambda = 0$$

$$p(o_i | \hat{c}) = \frac{\#(\hat{c}, o_i)}{\lambda}$$

$$\sum_i p(o_i | \hat{c}) = \sum_i \frac{\#(\hat{c}, o_i)}{\lambda} = 1$$

$$\lambda = \sum_i \#(\hat{c}, o_i)$$

$$p(o_i | \hat{c}) = \frac{\#(\hat{c}, o_i)}{\sum_i \#(\hat{c}, o_i)}$$

- מה אם θ יהיה ממש גדול? נלמד נושאים (topics) – כל המילים שפגיעות ממאמר שעוסק בספרות קבלו ממשמעות מאוד קרובה (כפי הן צריכות להיות דומות לכל שאר המילים במאמר).

- מה אם $\lambda = 1$ – בלחומר ממש קטן? (חולון של שתי מילים) נניח שיש לנו את המילים [running, shoes, walking] – running/walking ?shoes? מה סביר שנבנה לצד המילה יקבל משמעות דומה, כי שניהם פעילים, ומופיעים בחולון של שתי מילים ליד שם עצם. לעומת זאת verb-adverb ו-dominim, לא יקבלו ממשמעות דומה.

מה שמקסם את המטריה היא ההסתברות האמפירית:

$$p(o|c) = \frac{\#(c,o)}{\sum_{o'} \#(c,o')}$$


אופטימיזציה (SGD): נגזר פונקציית הפסד:

$$\mathcal{J}(\theta) = -\log(\mathcal{L}(\theta)) = -\sum_{t=1}^T \sum_{j=-m}^m \log(p_\theta(w_{t+j}|w_t))$$

זה negative log-likelihood, כדי שנוכל לבצע **מינימיזציה** ובכך למקסם את פונקציית המטריה שלנו. נבצע אופטימיזציה באמצעות SGD (באופן איטרטיבי) ונעדכן את הוקטוריהם כדי למצער את (\mathcal{J}) .

כל העדכן שלנו הוא: $\theta^{new} = \theta^{old} - \alpha \nabla \mathcal{J}_t(\theta)$.

נסתכל על האופטימיזציה עבור צמד בודד של (c, o) ונרצה לחשב את הנגזרת של הביטוי $\log(p_\theta(o|c))$. אם בן נפתח את הביטוי ונגזר אותו:

$$\begin{aligned} \log(p(o|c)) &= \log\left(\frac{e^{u_o^T v_c}}{\sum_{o_i} e^{u_{o_i}^T v_c}}\right) = u_o^T v_c - \log \sum_{o_i} e^{u_{o_i}^T v_c} \\ \nabla_{v_c} \log(p(o|c)) &= u_o - \frac{1}{\sum_{o_j} e^{u_{o_j}^T v_c}} \cdot \sum_{o_i} e^{u_{o_i}^T v_c} \cdot u_{o_i} = u_o - \sum_{o_i} \frac{e^{u_{o_i}^T v_c}}{\sum_{o_j} e^{u_{o_j}^T v_c}} \cdot u_{o_i} = u_o - \sum_{o_i} p(o_i|c) \cdot u_{o_i} \\ &= u_o - \mathbb{E}_{o_i \sim p(o_i|c)} [u_{o_i}] \end{aligned}$$

מבחן חישובית ההרצה של SGD היא על כל הלקסיקון ולבן מדובר בתהילך בלבד, אמן נבחן שmbחינה תאורטית אנחנו מקבלים כי הנגזרת ביחס ל- v_c היא הפרש בין embedding ברגע זה לבני מה שהוא אמר או היה (התחלתי). אנחנו מנסים לקרב את ההשערות שלנו לגביהם ובין מה שציפינו.

Negative Sampling: הבעיה בביטוי הוא הסכום שיש לנו במבנה, הנגזרת שלו בבדה. אנחנו רוצים להיפטר מהחלוקת זהה. אם נסתכל רק על המכפלת שיש לנו במבנה, לא נוכל ליחס לכך הכרעה ברורה האם זה "טוב" או "רע" באשר זה לא ביחס לכל שאר הדאטא. לכן, עבור מסעוג multi-class לסייע ביארי: בהינתן זוג, נרצה לומר האם הזוג הזה מופיע ביחד (1) או לא (0).

"... that bank holds the mortgage on my home..."

- | | | |
|--|-------------------------------------|--|
| 1. $(x, y) = (\text{bank}, \text{that})$
2. $(x, y) = (\text{bank}, \text{holds})$
3. $(x, y) = (\text{holds}, \text{bank})$
4. $(x, y) = (\text{holds}, \text{the})$
...
... | → | 1. $(x, y) = ((\text{bank}, \text{that}), 1)$
2. $(x, y) = (\text{bank}, \text{table}), 0)$
3. $(x, y) = ((\text{bank}, \text{eat}), 0)$
4. $(x, y) = ((\text{holds}, \text{bank}), 1)$ |
|--|-------------------------------------|--|

למה אנחנו מחזיקים את המידע גם עבור זוגות שלא הופיעו (0)? אנחנו זקוקים לעדות שלילית כדי שהמודל ידע שלא כל זוג יכול להופיע, שלא יזכה פשוט רק את כל הזוגות שמוופיעים עם 1. אם כן, נרצה להגדיל את ההסתברות של זוגות שראינו, ולהפחית הסתברות של כל שאר הזוגות. ניעזר בפונקציית sigmoid כדי לקבל ערכים בין 0 ל-1 ונגיד:

$$\begin{aligned} p_\theta(y = 1|c, o) &= \frac{1}{1 + e^{-u_o^T v_c}} = \sigma(u_o^T v_c) \\ p_\theta(y = 0|c, o) &= 1 - \sigma(u_o^T v_c) = \sigma(-u_o^T v_c) \end{aligned}$$

נקבל את ה-objective המעודכן הבא:

$$\mathcal{L}_2(\theta) = \sum_{t=1}^T \left[\sum_{j=-m}^m \underset{\text{positive}}{\log(\sigma(u_{w_{t+j}}^T v_{w_t}))} + \sum_{k \sim p(w)} \underset{\text{negative}}{\log(\sigma(-u_{w^{(k)}}^T v_{w_t}))} \right]$$

- מילת **center** שלנו היא w_t .
- המחוור הראשון – מייצג את הזוגות שראינו ביחד (positive samples).
- המחוור השני – k מילים רנדומליות, בהנחה שרוב המילים לא מופיעות יחד עם מילים אחרות (negative samples).
- המילה הרandomilit מסומנת $w^{(k)}$.
- ההתפלגות $(w)^k$ שמננה אנחנו דוגמים מוגדרת כך: $= \frac{U(w)^{3/4}}{T} p(w)$. כאשר T הוא אורך הקורפוס, $(w)^k$ הוא כמות הפעמים שהמילה w הופיעה בקורסוס, והפקטור של $\frac{3}{4}$ עוזר (סוג של טריק) להתמודד עם מילים נפוצות באנגלית שאנוanno לא רצחים שיתרמו לנו יותר מדי (...).

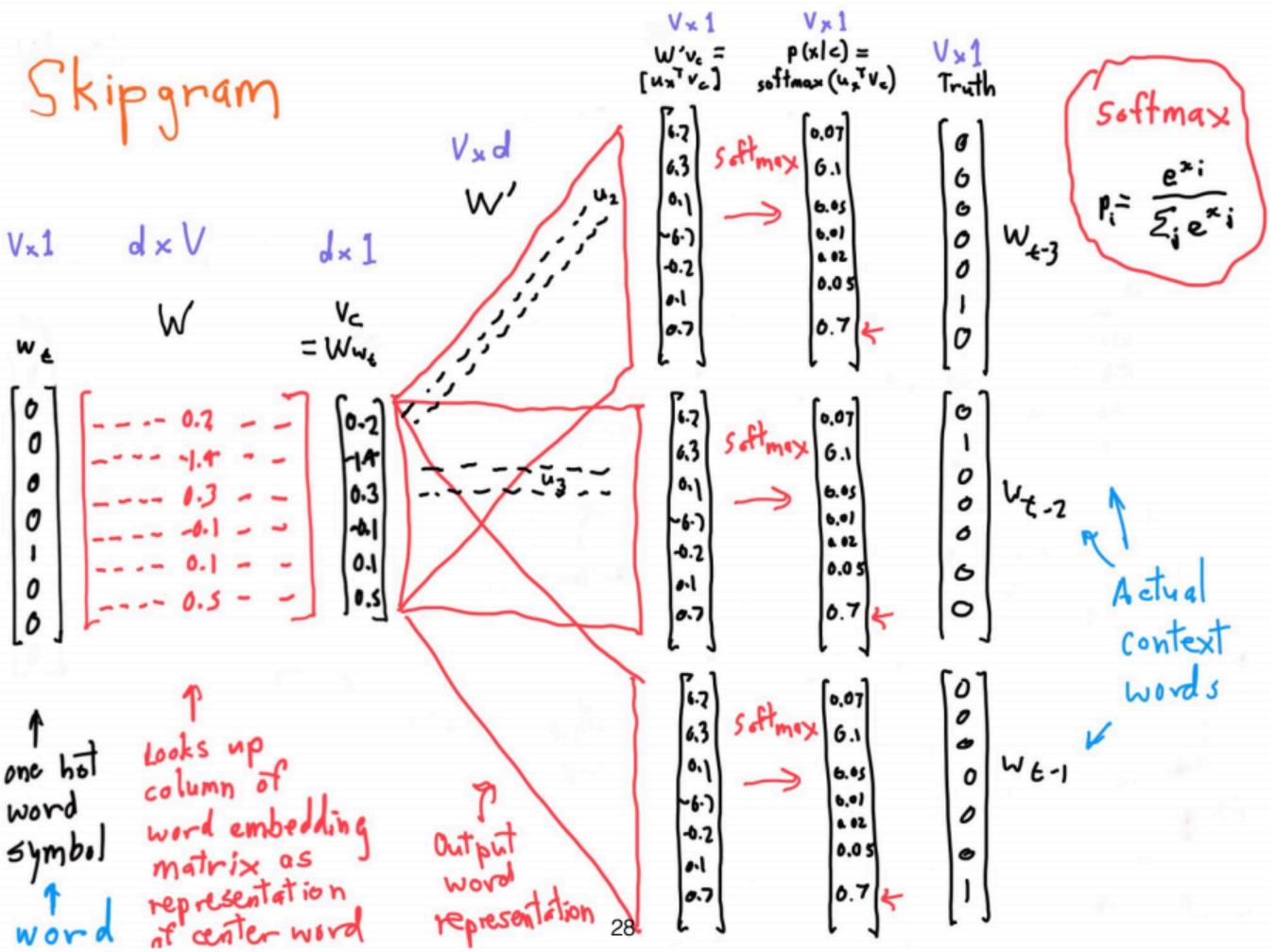


הגישה הזאת היא דוגמה ל-**contrastive learning**: הדעתה הוא זוגות (y, x) שאנו יודעים כי האיברים בו צריכים להיות **זהים** זה לה. אפשר להניח כי זוגות רנדומליים אחרים הם **לא** זהים. נמצא בשימוש היום ב-CLIP (vision) וב-DPR (חילוץ מידע).

תרשים skipgram המקורי (ללא negative sampling):

- אנחנו מתחילהים מ- m -center one hot vector w_t שהוא מילת center שלינו.
- מכפילים אותו עם מטריצת embedding המסמננת W ושולפים את העמודה המתאימה לפיה האינדקס i בוקטור w שבו הערך הוא 1. כך נקבל את ה- i -word embedding center word.
- בעצם נרצה לבדוק מה תוצאות המכפלה של המילה c עם כל שאר המילים החיצונית n , באמצעות המטריצה W .
- נפעיל את פונקציית softmax כדי לקבל את התוצאה של המודול.

עבור ה-objective אנחנו עוברים לסקום של \log על פני המילים שבאמת הופיעו ב-context של המילה שלנו (ה-Truth), ועל כן נדרש להוסיף את 0.7 (עבור w_{t-1}), את 0.1 ואת 0.05. המודל ינסה למקסם את ההסתברויות הללו.



סיכום עד כה:

- המטרה: ייצוג מילים בעזרת וקטורים ממימד נמוך.
- גישה: self-supervision.
- הרכיבים הדורשים עבור skipgram:
 - מודל – סיווג בינהי.
 - פונקציית מטרה – שימוש ב-negative sampling.
 - אופטימיזציה – SGD.



Matrix Factorization

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

ניתן לבנות מטריצת co-occurrence עבור הדוגמה הבאה:

- הקורסוס: "I like deep learning", "I enjoy flying", "I like NLP"
- הלקסיקון: $\mathcal{V} = \{I, like, enjoy, deep, learning, NLP, flying, \dots\}$
- כל שורה מייצגת וקטור של "word count", במא פעים מילה Z בלשוני מהלקסיקון הופיעעה לצה המילה X שאנו ממתכוים עליה.

הערות:

- למה אי אפשר להשתמש בו בטור הווקטור המיצג של המילה? סיבה לכך היא שהווקטור הוא sparse – יש הרבה אפסים.
- בנוסף, הוא גדול בהתאם לגודל הלексיקון (המיןדי שלו הוא בגודל הלексיקון).
- ניתן להשתמש בפרק SVD ולכטוב את המטריצה שלנו A בטור UDV^T , כדי לקבל מטריצה שהיא יחסית דומה למטריצה המקורית. לכל מילה יהיה וקטור מגודל k (יותר קטן מ- d) שמייצג את המילה.

הקשר ל-word2vec: [skipgram](#) – נסתכל על הפלט של word2vec

- המטריצה V היא אוסף של וקטורים ממימד d , כל אחד מייצג center word בלקסיקון שלנו שהוא מגודל $|V|$.
- המטריצה U^T היא אוסף של וקטורים ממימד d בייצוג context word.
- אם נכפיל את המטריצות הללו נקבל את המטריצה M שהיא מגודל $|V| \times |V|$. התא במקומות (j, i) במטריצה M הוא המכפלה של השורה V_i עם העמודה U_j^T .

$$M \in \mathbb{R}^{|V| \times |V|} = V \in \mathbb{R}^{|V| \times d} \times U^T \in \mathbb{R}^{d \times |V|}$$

מהי המטריצה שאנו עושים לה decompose? ראשית נגדיר:

- D: אוסף הזוגות של מילים (center, output). $(center, output)$
- $\#_c(w) = \sum_o \#(w, o)$: כמות הפעמים ש-center מופיע, היא סכום של כל הפעמים שהוא הופיעו לצד כלשהו.
- $\#_o(w) = \sum_c \#(c, w)$: כמות הפעמים ש-output מופיע באופן דומה.
- $T = \sum_{(c,o)} \#(c, o) = |D|$: כמות הפעמים שזוג של מילים בלחן (center, output) מופיעות ביחד.
- $P_D(w) = \frac{\#_c(w)}{N} = \frac{\#_c(w) \cdot m}{N \cdot m} = \frac{\#_o(w)}{T}$: התפלגות unigram של מילות output.

נשבט את objective שלנו עם ההגדרות החדשות:

$$\mathcal{L}(\theta) = \sum_{c,o} \left[\#(c, o) \left(\log(\sigma(u_o^T v_c)) + k \cdot E_{o' \sim P_D} [\log(\sigma(-u_{o'}^T v_c))] \right) \right]$$

Instead of iterating over center words and their context windows, group appearances together

Positive term

Summing k random log dot products according to the unigram distribution is like multiplying k by the expectation of the log dot product (in expectation)



המשך החישוב:

$$\begin{aligned}
 \mathcal{L}(\theta) &= \sum_{c,o} \left[\#(c,o) \left(\log(\sigma(u_o^T v_c)) + k \cdot E_{o' \sim P_D} [\log(\sigma(-u_{o'}^T v_c))] \right) \right] \\
 \text{Distribute} &= \sum_{c,o} \#(c,o) (\log(\sigma(u_o^T v_c))) + \sum_{c,o} \#(c,o) \cdot k \cdot E_{o' \sim P_D} [\log(\sigma(-u_{o'}^T v_c))] \\
 \text{Expectation is} &= \sum_{c,o} \#(c,o) (\log(\sigma(u_o^T v_c))) + \sum_c \#_c(c) \cdot k \cdot E_{o' \sim P_D} [\log(\sigma(-u_o^T v_c))] \\
 \text{constant for } o &= \sum_{c,o} \#(c,o) (\log(\sigma(u_o^T v_c))) + \sum_c \#_c(c) \cdot k \cdot \sum_o \left[\frac{\#_o(o)}{T} \cdot \log(\sigma(-u_o^T v_c)) \right] \\
 \text{Open} &= \sum_{c,o} \left[\#(c,o) (\log(\sigma(u_o^T v_c))) + \#_c(c) \cdot k \cdot \frac{\#_o(o)}{T} \cdot \log(\sigma(-u_o^T v_c)) \right] \\
 \text{expectation} & \\
 \text{Gather terms} & \\
 \end{aligned}$$

Terms in tension

נניח כי המכפלות בלתי תלויות, שכן $\sum_o n_o = x$ עבור זוג בלשנו (c, o). נסמן:

$$l(x) = \#(c,o) \log(\sigma(x)) + \#_c(c) \cdot k \cdot \frac{\#_o(o)}{T} \log(\sigma(-x))$$

וכך נקבל $(\Theta) \mathcal{L}$. אם גוזרים ומשווים ל-0 עבור זוג מסוים מקבלים:

$$\begin{aligned}
 \frac{\partial l(x)}{\partial x} &= \#(c,o) \sigma(-x) - \#_c(c) \cdot k \cdot \frac{\#_o(o)}{T} \log(\sigma(-x)) = 0 \Leftrightarrow x = \log \left(\frac{\#(c,o) \cdot T}{\#_c(c) \cdot \#_o(o)} \cdot \frac{1}{k} \right) \\
 &= \log \left(\frac{p(c,o)}{p(c) \cdot p(o)} \right) - \log k = PMI(c,o) - \log k
 \end{aligned}$$

התוצאה הנחמדה שאנו מקבלים באן היא: $PMI(c,o) - \log k$. באשר PMI הוא point-wise mutual information [P_{X,Y}] / [P_XP_Y]. הוא משמש באן מzd להערכתה שהן יופיעו זו לצד זו. נוכל לקבל באן 1 במקרה שה משתנים הם ב"ת. נוכל לקבל באן 0 במקרה לא מופיעות בכלל זו לצד זו. לעומת זאת נקבל 1.

המסקנה היא שאלגוריתם word2vec עם negative sampling בוצע מבחן מטריצת PMI שעבירה shift. שיטות רבות ב-NLP נזירות במטריצת PMI לחישובים.

Evaluation

אנו רוצים לדעת האם האמ-h-embeddings שקיבלו הם שימושיים עבורנו. יש שני סוגים של אובלואציה:

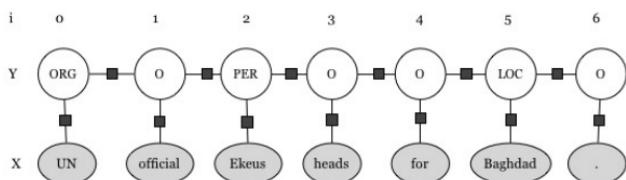
- Intrinsic**: מגדירים מילאכיות שמנסה באופן ישיר לבדוק את האיכות של אלגוריתם הלמידה שלנו. אנו לא מסתכלים על ה-downstream task.
- Extrinsic**: בודקים האם הפלט של המודול שלנו אכן שימoshi בבעית NLP Amityut (downstream).

אפשר להסתכל על אנלוגיות בין מילים. נרצה למצוא את המילה המתאימה ל-king, כמו sh-woman מתאים ל-man: **Intrinsic**: cosme similarity $a :: b \Leftrightarrow c :: d$.

$$d = \arg \max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}$$

- אם נקבל *meenq*, זה מוכיח את התהוושה שלנו שהמודול עובד בצורה נכונה.
- ממצאים הורדת ממדים (לא באמצעות PCA) ומטיילים את הייצוגים על מערכת צירים דו-ממדית.
- התוצאות מראות כי GloVe מציג תוצאות גבוהות במיוחד, גם על טסט-טסט שתויג על ידי בני אדם (דרוג קורלציה בין מילים).

עובדת שיצאה לא מזמן מנסה לנתח את הגיאומטריה של הייצוגים במודל Gemma.



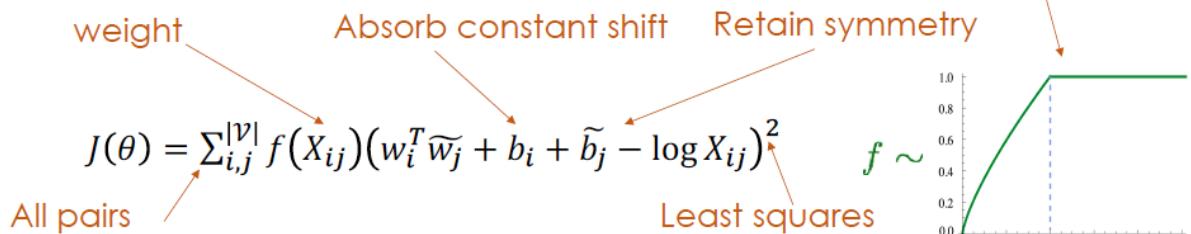
Extrinsic: מסתכל על המשימה של NER (זהוי ישויות) – מציאת אובייקטים לאנושיים, מקומות, ארגונים. גם כאןמצא Sh-**GloVe** מציג את הביצועים הגבויים ביותר.

GloVe: אלגוריתם נוסף לייצוג מילים, שבו אנחנו מוקודים ל-objective את העבודה שאנו מבנה לינארו. למשל, הרואں עבר המילים ice ו-steam **שהיחסים** בין הסתבריות יותר חשובים מהסתבריות עצמן.

אם כל מה שאנו עושים זה להכפיל מטריצה PMI, אבל علينا לעבור על כל הקורפוס שלו, לבצע אופטימיזציה (SGD), כדי לקבל ייצוגים שמאפשרים את המטריצה M שמעניינת אותנו – נרצה באופן ישיר למצא את M תוך שמירת האינפורמציה שמעניינת אותנו.

X - co-occurrence matrix $P_{ij} = P(j|i) = \frac{X_{ij}}{X_i}, \quad X_i = \sum_k X_{ik}$

Recast as a regression task $w_i^T \tilde{w}_j = \log(P_{ij}) = \log(X_{ij}) - \log(X_i)$



לסייע:

- מילים הם אבני בניין בכל שפה, ולכן ב כמעט כל מערכת NLP נוצר רישום למילים.
- יוצרים גרפים קשים למניפולציה והם יקרים, ו**one-hot vectors** עם שימושיים עם מספיק דאטא אבל מאבדים הכללה (לא מבטאים דמיון בין מילים).
- ביקורת אפשרית על **word embeddings** היא שהמלים דומות מאוד good-excellent, אך גם good-bad כי לרוב מילים אלו מופיעות באותו קונטקסט תחבירו. כדי להבחין בין מילים נרדפות למילים הופכות, לא תמיד המודל הזה מצליח.
- אלגוריתם skipgram עם negative sampling הוא דרך פופולרית ללמידה של word embeddings, ויש לאלגוריתם זה קשר חזק למגוון קלסיות של matrix decomposition.
- עדדים להמשך:
- **contextualized word representations**: הייצוג למילה מחושב כפונקציה של המילים שופיעו ב-contex, ולא רק על בסיס המילה עצמה. עם זאת בסיס, יש ייצוג non-contextualized context.
- **sentence representations**: ייצוג משפטים, ויצוגים אחרים כמו sub-words, תווים, מorfומות וכו'.

חומרנים נוספים:

- הרצאה: [NLP 20A \(JB\) - 2: Word embeddings](#)
- סרטון: [Word Embedding and Word2Vec, Clearly Explained!!!](#)
- סרטון: [Understanding Word2Vec](#)
- סרטון: [A Simple Introduction to Softmax](#)
- מאמר: [GloVe: Global Vectors for Word Representation - ACL Anthology](#)
- מאמר: [\[2406.01506\] The Geometry of Categorical and Hierarchical Concepts in LLM](#)
- מאמר: [\[1402.3722\] word2vec Explained: deriving Mikolov et al.'s negative-sampling](#)
- מאמר: [\[1301.3781\] Efficient Estimation of Word Representations in Vector Space](#)
- מאמר: [Neural Word Embedding as Implicit Matrix Factorization](#)
- מאמר: [Machine Learning — SVD & PCA | by Jonathan Hui | Medium](#)

מודלי שפה

בחלק זה נדבר על מודל שפה, ממשימה יסודית ב-NLP אשר נמצאת בשימוש בתרגום, תיקון אוטומטי, זיהוי דיבור וכו'. מודלי שפה מסווגים מושתמשים ב-*n-grams*.

Language Models

מושג central: LMs are all the buzz! אבל מה זה בכלל LM? **מודל שפה** מגדיר התפלגות מעל משפטים. שימושים:

- **likelihood**: איך מבין שני משפטים סביר יותר? למשל בתרגום מוכנה, את הביטוי "רוחות עדות/חזקות" נרצה לתרגם באופן מילולי ל- "strong winds" כאשר בפועל באנגלית נאמר "high winds" ויש לכך סבירות גבוהה יותר.
- **generation**: אפשר לייצר טקסט על ידי בר שמייצר כל פעם מילה אחת, ונחזה את המילה הבאה שהיאabic סבירה בהינתן המילה הקודמת.
- **TST**: נשתמש בכלל בייס ונקבל $P[\text{text}|\text{speech}] = P[\text{speech}|\text{text}] \cdot P[\text{text}]$, כלומר אנחנו מכניםים מה גם פקטורי של ההסתברות שהtekst עצמו יופיע, מעבר לתמול השמע לטקסט.
- **next word prediction**: גם Classification/Question Answering.

סוגיה פילוסופית/בין אדם למוכנה: האם הגדרת התפלגות מעל משפטים משמע הבנת שפה?

- יש שכבות רבות בבלשנות שהידע שלהן הכרחי **להבנה של שפה טבעית** – למשל מבנים תחביריים וקשר בין מילים (לפחות פועל, מה יכול להשלים אותו), פרגמטיקה ושימוש בשפה בהקשר. בנוסף, **ידע על העולם** והקשר, עוזר לנו לבני אדם לחזות מה המילה הבאה ברכץ דיבור/טקסט כלשהו. האינטואיציה הזאת מנחלה בכיתה מודלים של **word prediction** ואיך ליצג את המילים בשפה.
- ניסויים שהוציאו כדי לבחון את הקשר הזה בין אדם למוכנה – מבחן טיוורינג (האם אדם ידע להבחין בין שיח מול אדם אמיתי לבין שיח מול מוכנה), מבחן החדר הסיני (האם אדם שעומד באמצע ומקלט קלט בסינית מצד אחד, מסתכל על משפטים נפוצים בסינית והמשבים אפשריים, וכן פולט סביר בעינו בצד השני – מבין סיניות? בכמה $\% \text{ gpt}$ עשו את זה..).

הגדרת הבעה: בהינתן **לקסיקון סופי** $\{ \text{the}, \text{a}, \text{man}, \text{telescope}, \text{one}, \dots \}$, נגדיר את **השפה האינסופית** $STOP \circ \mathcal{N}^*$.

לכל רצף מילים שאפשר לחשב עלייו, יש הסתברות בלשיה. לא בעניין הסתברות 0, בלבד משפט הוא "אפשרי".

- קלט: train set של משפטים לדוגמה.
- פלטו: התפלגות p מעל L : $0 \leq p(x) \leq 1, \forall x \in L$.

ניסוי ראשון – התפלגות אמפירית: נניח שיש לנו N משפטים אימון. יהיו x_1, x_2, \dots, x_n ממשפט, $-x_1, \dots, -x_n$ כמספר הפעמים שהוא הופיע במסט האימון. נגדיר מודל שפה נאיבי: $\frac{c(x_1, \dots, x_n)}{N} = p(x_1, \dots, x_n)$. לאור ענייני sparsity, נקבל 0 עבור הרבה מילים שלא ראיינו עדין (מצב שאינו רצוי עבורהנו, נרצה להעניק להם הסתברות חיה/ית בלשיה).

ניסוי שני – bigram: אם הסט שלנו גדול מספיק, נוכל לקבל מידע טוב באמצעות bigrams. נסתכל על פרויקט של משפטים שנאספו מזהמנויות במסעדה ב-Berkeley, נספור את כמות הפעמים שבל מילה הופיעה ליד כל מילה ונקבל הסתברויות מתאימות. נוכל ללמידה מכך מספר דברים:

- $P[\text{English}|\text{want}] < P[\text{Chinese}|\text{want}]$ (אנשיים מעדיפים אוכל סיני).
- $P[\text{to}|\text{want}] = 0.66$ כהתנהוג מסויימת של השפה האנגלית.

זה עדין לא מספיק. חלון המילים שאנו מסתכלים עליו (או זכרים בראש) הוא יותר מ-2.

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0



תחליבים מركוביים (Markov): בהינתן רצף של $100 = a$ משתנים מקראיים אקראיים $\mathcal{V} \in X_i$. יש $i|\mathcal{V}$ רצפים אפשריים. משתמש בכל השרשרת כדי לעבור להסתברות מותנית (המילה הראשונה, השנייה מותנית בראשונה, השלישי מותנית בראשונה והשנייה..):

$$P[X_1 = x_1, \dots, X_n = x_n] = P[X_1 = x_1] \cdot \prod_{i=2}^n P[X_i = x_i | X_1 = x_1, \dots, X_{i-1} = x_{i-1}]$$

עדין צריך לאחסן טבלה גדולה עבור כל ההסתברויות, פשוט יסחנו מחדש שזונה אבל לא צמצמו משמעותית את הבעייה.
נציג הנחות מרכוביות – הערך של משתנה מקרי X_i תלוי רק בסביבה מסוימת שקרובה אליו – נניח רק במילה שלפנינו. בולמר, ההסתברות שלמילה תופיע לא תליה בכל רצף המילים שהופיע לפנייה, אלא רק במילים האחרונים שלפניה (1 או 2 ..). נניח שיש לנו את המשפט "I ate a pizza", אם נסתכל רק על מילה אחרת: $P[\text{pizza}|a]$ התשווה היא שההסתברות יותר נמוכה מאשר אם נסתכל על מספר מילים אחרות: $P[\text{pizza}|I \text{ ate } a]$.

- נניח סדר ראשון (bigram):

$$P[X_i = x_i | X_1 = x_1, \dots, X_{i-1} = x_{i-1}] = P[X_i = x_i | X_{i-1} = x_{i-1}]$$

- אפשר להניח סדר שני (trigram):

$$P[X_i = x_i | X_1 = x_1, \dots, X_{i-1} = x_{i-1}] = P[X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}]$$

כדי לעבוד בצורה נוחה נגדיר סמלים מיוחדים להתחלה: $* := x_0 := x_{-1}$. קיבלנו בסדר שני כי מילה תליה ב-2 המילים לפניה:

$$P[X_1 = x_1, \dots, X_n = x_n] = P[X_1 = x_1] \cdot P[X_2 = x_2 | X_1 = x_1] \cdot \prod_{i=3}^n P[X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}]$$

התאמה לאור משותנה: האם סביר להגיד $100 = n$? למה לא 10? או 1024? כדי להיפטר מהמגבלה הזאת, נגדיר $X_n = STOP$ וכן נקבל התפלגות מעלה כל הרצפים האפשריים. יש כאן תהליך גנרטיבי. מהי ההסתברות של משפט מסויים?

- ראשית, $* | * | STOP$ היא ההסתברות שהמודול נותן למשפט הריק. ההסתברות המשלימה היא כל המשפטים מאורך 1 או יותר.

- ניציר את המילה הבאה: $\sum_{w \in \mathcal{V}} P[STOP| * w]$ ואם נסכם עם הקודם נקבל את ההסתברות למשפט ריק ועוד משפטים מאורך 1.

כלומר בכל שלב יש לנו הסתברות עבור **עציירה באורך מסוים - α_h** (α_h על בסיס המילים עד כה), והסתברות $(\alpha_h - 1)$ עבור המשך התהליך.

מודל שפה Trigram

$$\begin{aligned} p(\text{The dog barks STOP}) &= q(\text{The} | *, *) \\ &\quad \times q(\text{dog} | *, \text{The}) \\ &\quad \times q(\text{barks} | \text{The}, \text{dog}) \\ &\quad \times q(\text{STOP} | \text{dog}, \text{barks}) \end{aligned}$$

- פרמטרים: $w | u, v \in [0,1]$ כאשר $w \in \mathcal{V}$ ו- $u \in \mathcal{V}$ ו- $v \in \mathcal{V}$ ו- $\{*\}$
- ההסתברות של משפט שלשה $x_1, x_2, \dots, x_n = STOP$ תהיה:

$$P[x_1, x_2, \dots, x_n] = \prod_{i=1}^n q(x_i | x_{i-2}, x_{i-1})$$

הנחה המרכיבית זו בעייתית, יש לנו משפטים עם תלויות ארוכות יותר בהן צריך להסתכל הרבה מילים כדי לחזות את המילה הבאה בצורה מדוייקת: "He is from France, so it makes sense that his first language is _____.

נניח שנרצה לשערق את q באמצעות maximum likelihood, נגדיר $(w_n, w_{n-1}, \dots, w_1) c$ כמספר הפעם שהוא-gram מופיע בקורסוס, ונחשב:

$$q(w_i, w_{i-2}, w_{i-1}) = \frac{c(w_{i-2}, w_{i-1}, w_i)}{c(w_{i-2}, w_{i-1})}$$

נניח שבלקסיקון שלנו יש $20,000 = |\mathcal{V}|$ מילים נציג $10^{12} \sim 8^3 |\mathcal{V}|^3$ פרמטרים. علينا לחשב את q עבור כל שלשה אפשרית מעלה הלקסיקון. בנוסף, לחוב המשפטים יהיו הסתברויות 0, כי יהיו שלשות של מילים (trigrams) שלא ראיינו מעולם, או שלא ראיינו את ה-prefix של שתי מילים שראינו וגם שם זה לא מוגדר.

Evaluation

נניח שיש לנו set test נתון (s_1, \dots, s_M) כאשר M הוא מספר המילים בלקסיקון. ההנחה היא ש-LM טוב יעניק הסתברות $P[S]$ גבוהה יותר לטקסט שלא נראה בעבר. נניח כי $P[S] = \prod_{i=1}^M P[s_i | s_1, \dots, s_{i-1}]$. אם ניקח \log_2 נקבל:

$$\log_2 P[S] = \sum_{i=1}^M \log_2(P[s_i | s_1, \dots, s_{i-1}])$$

מטריקת perplexity זו היא S הינה l ונקבל: $perplexity = \frac{1}{M} \log_2 P[S]$.

$$PPL = 2^{-l} = 2^{-\frac{\log_2 P[S]}{M}}$$

PPL נמוך יותר מצביע על הסתברות גבוהה יותר.

נניח שיש לנו לקסיקון \mathcal{N} באשר $1 + |\mathcal{N}| = N$ (עבור המילה STOP) ומודל trigram עם התפלגות יוניפורמית: $q(w|u, v) = \frac{1}{N}$. נקבל כי PPL הוא אפקטיבית גודל הלקסיקון שמננו המודל צריך לבצע בכל פעם את המילה הבאה:

$$PPL = 2^{-\frac{\log_2 P[S]}{M}} = 2^{-\frac{\sum_{i=1}^M \log_2(P[s_i | s_1, \dots, s_{i-1}])}{M}} = 2^{-\frac{\sum_{i=1}^M \log_2(\frac{1}{N})}{M}} = 2^{\log_2(N)} = N$$

ערבים נפוצים – עברו $50,000$ $= |\mathcal{N}|$:

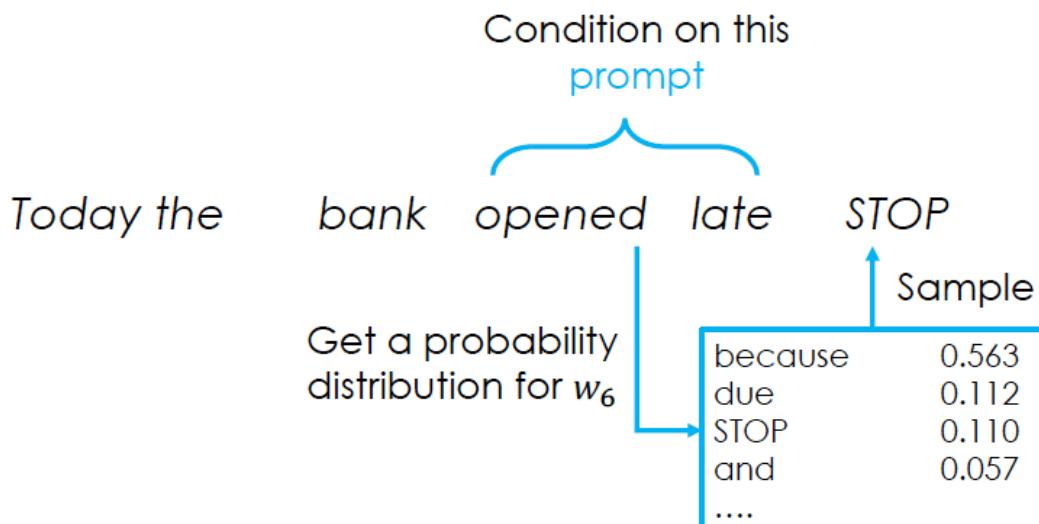
- $PPL_{trigram} = 74$
- $PPL_{bigram} = 137$
- $PPL_{unigram} = 955$

Seq2Seq: קיבלת רצף בלשונו המקורי, והזרת רצף אחר בפלט. אנחנו מכירים כבר מספר סוגי מודלים:

- קלאסיפיקציה בינארית $f: \mathcal{X} \rightarrow \{0,1\}$
- קלאסיפיקציה multi-class $f: \mathcal{X} \rightarrow \{c_1, \dots, c_m\}$
- גראסיה: $f: \mathcal{X} \rightarrow \mathbb{R}$

בעת אנחנו מדברים על structured prediction: הפלט הוא רצף מילים מתוך סט נתון (עצים, משפטים). הקלט יכול להיות אובייקט בווד (למשל תמונה – נרצה לתת לה תיאור). נבובון על הפלט ייצור טקסט עם LM n -gram:

- ניקח את "Today the" בתור prompt ונקבל התפלגות עבור המילה w_3 .
- בגישה חמדנית ניקח את המילה עם הציון המקסימלי, או שנחליט לדגום רנדומלית מתוך התפלגות הזאת.
- בכל שלב ה-prompt שלנו (חלון של 2 מילימ) זו קידמה, ואנו ממשיכים לדגום מילים עד שנdagם את STOP.



יש tradeoff בין גודל ה-gram ו- n . אם מסתכלים על ההיסטוריה קצרה מידי (unigram) אנחנו מאבדים קורנליות והתקסט לא נראה דקוטרי, לא תופסים את המשמעות והקשרים בין מילים (underfit). מהצד שני, אם לוקחים את זה לקיצון (quadrigram) אנחנו מפספסים את היכולת להכיל בזמן אמת (overfit).

בשלב מסוים יכול להיות שנגעה ל- n -gram שנראה רק פעם אחת ב-set, train-set, אז ברגע שמייצר אותו, בוודאות ניציר את המשך שלו, וכך נמשיך בעצם לפולוט מידע מה-set train-set וזה הפוך פשוט לשזהר מידע שהלומד ראה ב-set ביל' שום היגיון. אין ממש spot sweet, או שזה חסר היגיון (unigram) או שהוא פשוט מעתיק מילים (quadrigram).

Unigram

To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have
Every enter now severally so, let
Hill he late speaks; or! a more to leg less first you enter
Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like

Bigram

What means, sir. I confess she? then all sorts, he is trim, captain.
Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.
What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?

Trigram

Sweet prince, Falstaff shall die. Harry of Monmouth's grave.
This shall forbid it should be branded, if renown made it empty.
Indeed the duke; and had a very good friend.
Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

Quadrigram

King Henry.What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;
Will you not tell me who I am?
It cannot be but so.
Indeed the short and the long. Marry, 'tis a noble Lepidus.

Improvements

כדי לתת מענה לביעיות על ציר trade-off זהה, נשלב בין כל שלושת המודלים עם משקלים שונים:

$$q_{LI}(w_i|w_{i-2}, w_{i-1}) = \lambda_1 q(w_i|w_{i-2}, w_{i-1}) + \lambda_2 q(w_i|w_{i-1}) + \lambda_3 q(w_i)$$

$$\lambda_i \geq 0, \lambda_1 + \lambda_2 + \lambda_3 = 1$$

- במקרה שניתקל בזמן test ברכף שלא ראיינו מעריכם, לפחות מסת הסתרות כלשהי בגל מודל ה-unigram (כל עוד ראיינו את המילה הזאת מתייחסו). במקביל, ישנים גם רכיבי bi/tri-grams שמאפשרים להתמודד עם תלויות רחוקות יותר, ולתרום לאקספרסיביות של המודל.
- כדי למצוא את הפרמטרים המתאימים, מפצלים ל-dev set-train ו-set-dev ובוחרים את הערכים λ_2, λ_1 שנוטנים את ה-PPL הנמוך על dev.

$$\Pi(w_{i-2}, w_{i-1}) = \begin{cases} 1 & c(w_{i-2}, w_{i-1}) = 0 \\ 2 & 1 \leq c(w_{i-2}, w_{i-1}) \leq 10 \\ 3 & \text{else} \end{cases}$$

אפשר להפוך את המודל למורכב יותר אם נגדיר שהוא תלוי ב-context: סביר שנרצה להשתמש בערכים אחרים של פרמטרים כתלות בהקשר בו אנחנו נמצא.

במקרים שהמילה נדירה מאוד נרצה לחזק את λ_3 ונסמן על ה-unigram (טוב בשימוש מעט DATA), ואם שתי המילים האחרונות מאוד נפוצות נרצה לחזק את λ_1 ונסמן על מודל ה-trigram (טוב בשימוש הרבה DATA).

יש לנו 3 אופציות למשקלים שונים של הפרמטרים:

$j = \Pi(w_{i-2}, w_{i-1})$	$j = 1$	$j = 2$	$j = 3$
λ_1^j	0.05	0.2	0.8
λ_2^j	0.2	0.5	0.15
λ_3^j	0.75	0.3	0.05



מילים לא ידועות: איך נתמוך עם מילה חדשה לחוטין בזמן $?test$? $P[S] = 2^{-\log(0)} = \infty$. נשתמש ב-token $<UNK>$. מיוחד שישוון $<UNK>$. ניקח את ה-set train, ונשים לב למילים שהם נדירות מאוד ונחליף עם $<UNK>$.

- הלексיקון יהיה כל המילים ב-set train ללא החליפו עם $<UNK>$.

בזמן test ישש מילה שאינה מוכרת מה-set train נחליף אותה עם $<UNK>$ ונחשב את ההסתברות שלו.

לסיכום:

- מודל שפה (LM): בהינתן טקסט מעניק ציון הסתברות לטקסט, או באופן שקול, בהינתן prefix כלשהו מעניק ציון הסתברות למילה הבאה (מעל הלексיקון).
- linear interpolation ו-smoothing. אפשר לבצע על אמן מודלי n-gram. אפשר לשבב בין ניסיונות לשפר מודלי שפה באמצעות הוספת תחביר (לא פשוט).

חומרים נוספים:

הרצאה: [NLP 20A \(JB\) - 3: Language Models](#)

פרויקט עבר: [A Simple and Effective Model for Answering Multi-span Questions - ACL Anthology](#)

מאמרם:

[A Neural Probabilistic Language Model](#)

[Exploring the Limits of Language Modeling](#)

[Infini-gram: Scaling Unbounded n-gram Language Models to a Trillion Tokens](#)

[\[1211.5063\] On the difficulty of training Recurrent Neural Networks](#)

[Language Models are Unsupervised Multitask Learners](#) :GPT-2

[lm-spring2013.pdf \(columbia.edu\)](#)

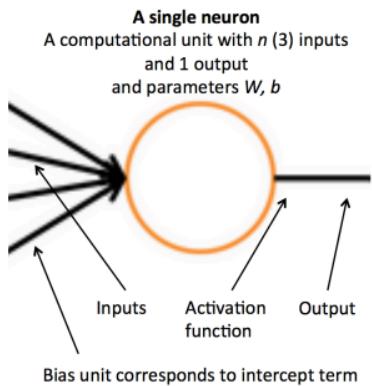
[planspace.org/perplexity/](#)

2 – למידה عمוקה

רשתות נוירונים

נסקרו רשתות נוירונים לטובת מודל שפה, ונראה כיצד FFNN לוקח בחשבון דמיון בין מילים כדי להכליל טוב יותר. *RNN* פוטנציאלית יכולה ללמידה תלויה ארכוקט-טוווח. מודלים נוירוניים הורידו דרמטית ביצוע perplexity.

Neural Networks Recap



נוירון: יחידה חישובית מהצורה $f_{w,b}(x) = f(w^T x + b)$ כאשר \mathbf{x} הוא וקטור קלט, \mathbf{w} וקטור משקלים, b הוא bias, f -היא פונקציה אקטיבציה כלשהי שאינה בהכרח לינארית – למשל ReLU/sigmoid. לעומת זאת הוקטור \mathbf{x} , מחשבים ממוצע ממושקל שלו לפי w (זה נקרא pre-activation) $= w_1 x_1 + \dots + w_d x_d$. אם זה גדול מ- b אז קורה המשחה (activation) שתלויה בפונקציה f .

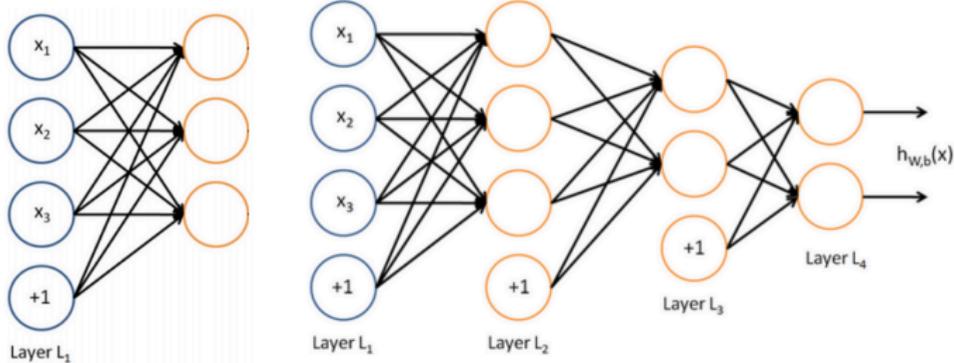
למשל, אם ניקח את הפונקציה sigmoid,נוירון הוא למעשה logistic regression. עברו דוגמת אימון (y, x) נקבל:

$$P[y = 1|x] = \frac{1}{1 + e^{-w^T x - b}} = \sigma(w^T x + b)$$

רשת נוירונים: אפשר לפתוח את הרעיון הזה עוד, ולבצע את **אותו החישוב** מספר פעמים במקביל – נקבל **מספר נוירונים שונים**, כאשר לכל אחד המשקלים שלו: $\sigma(w^T \hat{x} + b) = \sigma(\hat{w}^T \hat{x} + b)$.

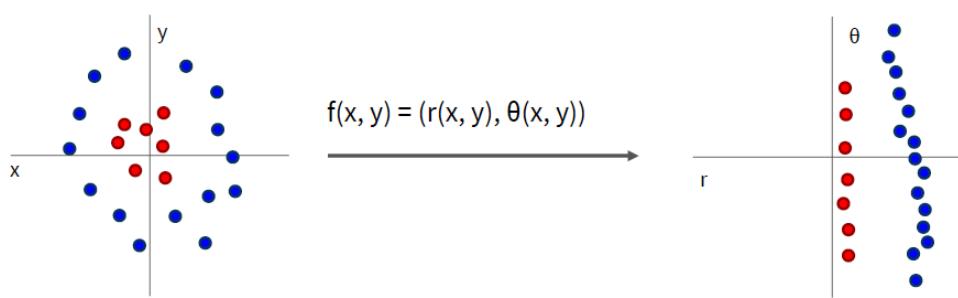
כדי לייצג את ה-*bias* נוסיף עוד צומת: $x_{d+1} = 1, w_{d+1} = b, w \in \mathbb{R}^{d+1}, x \in \mathbb{R}^{d+1}$.

אם חוזרים על התהליך מספר פעמים, נקבל רשת נוירונים הבנויה ממספר שכבות. נניח שכבה ראשונה של *outpus*, ושכבה חビיה שנמצאת בינהן (hidden). השכבה החビיה נקראת **learned representation** – אותן מספרים שהמודל החליט לחשב על מנת לבצע משימה מסוימת, מה שמיתרגם בסופו של דבר לפולט הרצוי (נניח 1 או 0 בסיווג ביןארי).

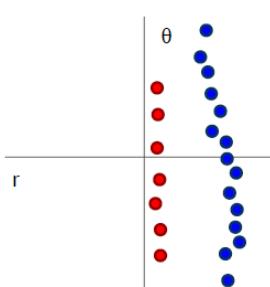


בכתב מתמטי: איך נגידיר את הטרנספורמציה בין שתי שכבות? נגידיר את $b = z = Wx + a$ שמופיע elsewhere על הוקטור המתkowski. במקום לבסוף 3 מכפלות של וקטור, כתוב מכפלה אחת של מטריצה בוקטור.

בנייה רשתות נוירונים:

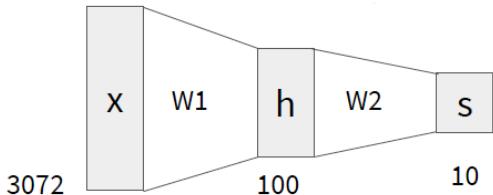


Cannot separate red and blue points with linear classifier



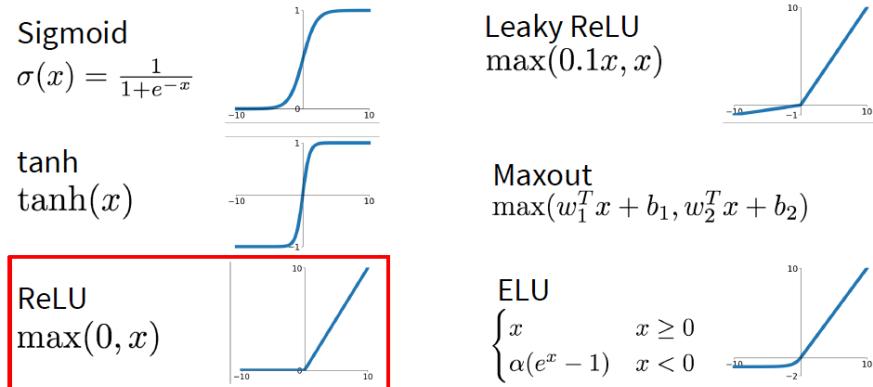
After applying feature transform, points can be separated by linear classifier

נתחיל מפונקציית linear score פשוטה: $f = Wx$. בעת רצח להוסיף שכבה נוספת: $f = W_2 \max(0, W_1 x + b)$. מה שונצח לעשות את זה? נניח כי אנחנו רוצים לסווג נקודות $+/$ – עם יכולת להפריד במרקם ההתחaltı. נטיל את הנקודות באופן צבאי לאחר מכן. נוכל להעביר בינהן קו ישר ולהפריד ביניהן.

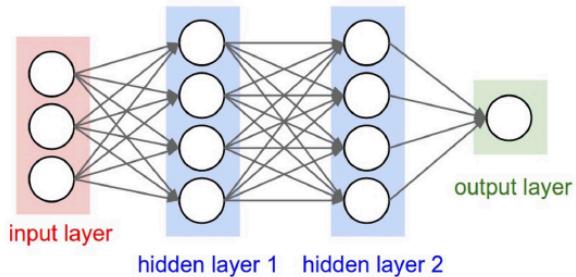


נשים לב כי $x \in \mathbb{R}^D$, ו- $W_1 \in \mathbb{R}^{H \times D}$. כך שלאחר הפעלה בעצם עברנו למימד $x' \in \mathbb{R}^H$ לאחר מכן אナンחנו מבצעים את המבוקשomin בין התוצאות לקטור ה-0 element-wise activation function להעיף את הקואורדינטות השיליות (ReLU) – פונקציה זו נקראת **ReLU**.

בעת $W_2 \in \mathbb{R}^{C \times H}$ ולאחר המכפלה קיבל לבסוף וקטור y . במובן הרחב זו **רשת מירוכים**, נקראת גם **fully-connected network** או **multi-layer perceptron** (MLP).



דוגמא ל-**feed-forward** עם שכבות חיוביות:



```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

דוגמא לaimon של רשת עם שתי שכבות: מחשבים את הגרדיינט בצורה אנליטית, ה-loss היה $2(\hat{y} - y)^2$ אז הנגזרת תהיה $(\hat{y} - y)$

```

1 import numpy as np
2 from numpy.random import randn
3
4 N, D_in, H, D_out = 64, 1000, 100, 10
5 x, y = randn(N, D_in), randn(N, D_out)
6 w1, w2 = randn(D_in, H), randn(H, D_out)
7
8 for t in range(2000):
9     h = 1 / (1 + np.exp(-x.dot(w1)))
10    y_pred = h.dot(w2)
11    loss = np.square(y_pred - y).sum()
12    print(t, loss)
13
14    grad_y_pred = 2.0 * (y_pred - y)
15    grad_w2 = h.T.dot(grad_y_pred)
16    grad_h = grad_y_pred.dot(w2.T)
17    grad_w1 = x.T.dot(grad_h * h * (1 - h))
18
19    w1 -= 1e-4 * grad_w1
20    w2 -= 1e-4 * grad_w2

```

Define the network

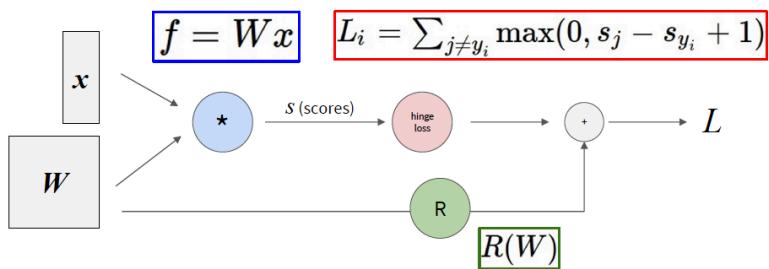
Forward pass

Calculate the analytical gradients

Gradient descent



הקוושי באן הוא בחישוב האנלטי, כאשר נוסף עוד ועוד שכבות. נרצה לחשב את הגרדיאנט – על הניר זה יותר מורכב. לכן משתמש בגרפים חישוביים -[hession](#). backpropagation. כפי שלמדונו על SVM multi-class hinge loss loss על margin של 1 לפחות (איזה כיף היה מבוא למידה חישובית אה..).



נסמן ב- q משתנה שיוביל את תוצאה החיבור $y + x$, לאחר מכן נרצה להכפיל אותו ב- z . מzin ערכים לדוגמה (בירוק) ונחשב ממשמאלי לMIN (forward pass). נחשב את הנגזרת של q ביחס ל- x ו- y . כדי לחשב את $(x')f$ נצטרך את כל השרשנות: $(x'q) \cdot (q')f$. באופן דומה עבור $(y'q) \cdot (q')f$. מוצאים propagation ביחסוב הנגזרות לאחר (back).

Backpropagation: a simple example

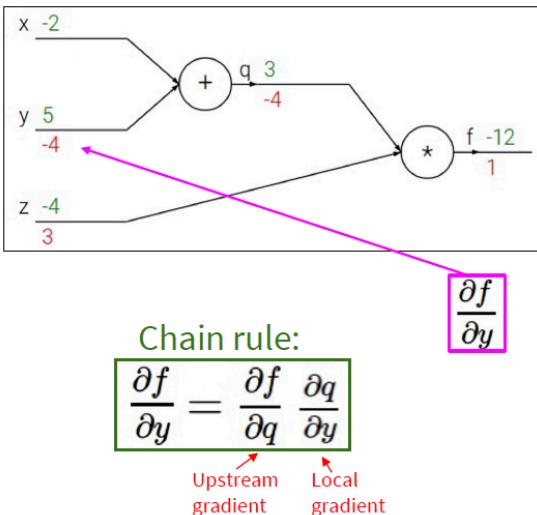
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



עד כה הסתכלנו רק על פונקציות שבهن הקלט והפלט היו סקלרים, לרוב אנחנו מבון עובדים עם וקטורים ומטריצות.

FFNN

מודולציה ורקיון

השעורור שלנו ($n, u | w$) q מבוסס על one-hot representation. אין קרבה/יחס בין מילים, כל מילה נחשבת ב"ת באחרת. נרצה שאם יש הסתברות גבוהה ל-[*w*]

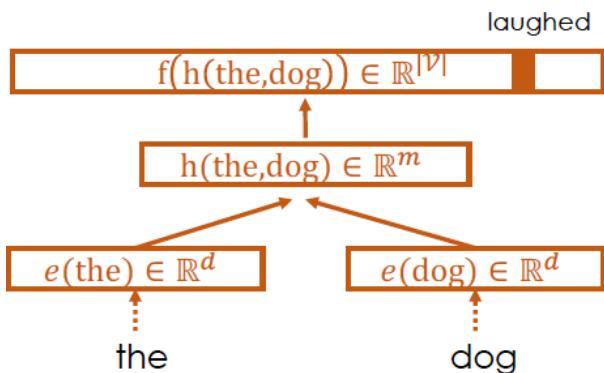
נרצה להשתמש ברשתות נוירונים (NN): מודל חישובי שמקווים בדרך הפעולה של המוח, הוצע באמצעות המאה ה-20. בשנות ה-80 עלה מחדש בעניין בנזין אלגוריתם back-propagation עם אלגוריתם SVM. בזמן האחרון אפשר לראות שהשימוש ב-NN נרחב ומשרת מגוון תחומיים. שתי סיבות עיקריות הובילו ל-NN:

- [learn representations from raw data](#): כדי לבצע משימת ML יש ליציג טקסט באמצעות מספרים כלשהם – על בסיס ידע מסוים ב-domain יש למצוא מיפוי שזכה. זו עבודה מורכבת שצריך לעשות עבור כל domain בנפרד, על מה לתה יותר משקל, והמאמץ שדרוש כדי להביא raw data למצוב שאפשר לעבוד עליו גודל.
- [learn non-linear decision boundaries](#): לבדוק בעיקר עם מודלים לינאריים, וה-NN שהקשה לקבל תוצאות טובות עבור מודלים שהם אינטראטיביים – [linearly separable](#).

דברינו על ML, ועל הבעיה שהייצוגים אינטראטיביים לא יכולים אינפומרציה בין "the actor" ו- "the actress" למשל. נרצה לפתור את זה. [נשמר את ההנחות המרקבויות](#) – ההנחות של מילה ש תהיה תליה ארוך ורק במילים שלפניה, אבל הייצוג של context יהיה [משותף](#) בין הרבה מילים.



הweeney: נתאר NN שמקבלת שתי מילים "the", "dog", ופולט התפלגות על כל המילים האפשריות בלקסיקון בהינתן שה-word הוא "the dog". בעת נפרט כיצד זה עובד.



- המילים מיוצגות כ-one-hot vectors ב-
- נפעיל פונקציה e שסמן את שני הווקטורים לייצוג ממימד נמוך.
- יותר, באמצעות **embedding matrix** W_h נשרhar את המילים (ליצות וקטור ממימד d), ככפוף במטריצה W_h .
- נפעיל פונקציה s לקבלת שכבה חביה. בתגובה, זה הייצוג של ה-word "the dog" בעבר. נסמן את התוצאה בווקטור $z = s(w_{i-2}, w_{i-1})$.
- השכבה האחורונה מבצעת **softmax** שיתן לנו את ההתפלגות על פני הלקסיקון. הערך שמתתקבל מהרשת לפני שהופכים אותו לסתובנות נקרא **logit** והוא הטללה על המטריצה W_o . התוצאה של הפעלת softmax לאחר מכן נשמרת בתוך וקטור ממימד הלקסיקון.

- Learn a probability $p(w_i | w_{i-2}, w_{i-1}) = f(h(w_{i-2}, w_{i-1}))$ with distributed representations

$$\begin{array}{ll} e(w) = W_e w, \quad W_e \in \mathbb{R}^{d \times |\mathcal{V}|}, \quad w \in \mathbb{R}^{|\mathcal{V}|} & \text{Embed} \\ h(w_{i-2}, w_{i-1}) = \sigma(W_h [e(w_{i-2}); e(w_{i-1})]), \quad W_h \in \mathbb{R}^{m \times 2d} & \text{Proj.} \\ f(z) = \text{softmax}(W_o z), \quad W_o \in \mathbb{R}^{|\mathcal{V}| \times m}, \quad z \in \mathbb{R}^m & \text{Prob.} \end{array}$$

.word2vec-n-gram U, V . המטריצות W_e, W_h, W_o מזכירות לנו את המטריצות V . המטריצה W_e מגדירה לנו את המטריצות U . המטריצות W_h מגדירה לנו את המטריצות V . המטריצות W_o מגדירה לנו את המטריצות U . בנוסף, עלינו להגדיר פונקציית loss. נגיד **negative log-likelihood**: $L(\theta) = -\sum_{i=1}^T \log p_\theta(w_i | w_{i-2}, w_{i-1})$

הערות:

- אם ראיינו בסט האימון שלנו: "The cat is walking in the bedroom", אנחנו מוקווים ללמידה של משפט הבא יש סבירות דומה: "A dog was running through a room". זה נובע מכך שאנו מוכדים עם word embeddings, שמחזקים מידע על הדמיון בין המילים.
- נניח שנרצה להתנסת על יותר מ-2 מילים אחריה, נניח 3 מילים אחרת. המטריצה $W_h \in \mathbb{R}^{m \times 3d}$ יוגדר עם $d=3$ כדי לשדרר 3 מילים. במודול n-gram-ה מס' הפרמטרים גדל **אקספוננציאלית** עם גודל החולון (כי זה חזקתו מ'). באן מס' הפרמטרים גדל **לינארית** עם גודל החולון.
- נשים לב שה-word embeddings של שתי המילים לא מושפעים זה מזה, בלבד מהם מוכפל במטריצה אחרת ("חצ'י") מהמטריצה המקורית W_h . לעומת זאת, אין שיתוף של מידע בין המילים.

Backpropagation: אנחנו מאמינים עם SGD, ונרצה לחשב באופן יעיל את הגראדיינט. לכן, ניעזר ב-

אלגוריתם תכונות דינמי שעדיר לנו לבצע את החישובים ביחס לפרמטרים של רשותנו. היום בשימושם רשותנו נוירונים יש חבילות **auto-differentiation** שעשויות את זה עבורנו (TensorFlow, PyTorch וכו'), אך לא צריך לחשב ידנית. נרצה לחשב את הגראדיינט של loss function ביחס לכל אחד מהפרמטרים שלו. סימונים חשובים:

- W_t : מטריצת המשקלים שלוקחת את הפלט של שכבה 1 – t ומחשבת את שכבה t .
- z_t : וקטור הפלט של שכבה t .
- $z_0 = x$: וקטור הקלט.
- y : וקטור הפלט הרצוי (gold).
- $\hat{y} = z_L$: וקטור הפלט של המודול (prediction).
- $\ell(y, \hat{y})$: פונקציית loss.
- $\delta_t = \frac{\partial \ell(y, \hat{y})}{\partial z_t}$: וקטור הגראדיינט.

חישוב: נירץ את הרשות באמצעות forward-pass (מ- z_0 נחשב את z_1, z_2, \dots , עד שמחשבים את z_L), נוכל לחשב את loss, וכן בעבשו לחשב backward-pass ולקבל את הגראדיינטם לכל פרמטר שכבה-שכבה.



הבסיס – תחילה נחשב את הגרדיינט ביחס לקטור האחרון: $\delta_L = \ell'(y, z_L)$

ברקורסיה – מחשבים בעורת הגרדיינט מהשכבה הבאה δ_{t+1} ומכפילים אותו בגרדיינט של ה-pre-activation באותו שכבה, ואז מבצעים טרנספורמציה לינארית כדי לקבל את הגרדיינט בשכבה הקודמת:

$$\delta_t = W_{t+1}^T(\sigma'(\nu_{t+1}) \circ \delta_{t+1})$$

נחשב את הגרדיינטים ביחס ל-activation באופן הבא:

$$\frac{\partial \ell}{\partial W_t} = (\delta_t \circ \sigma'(\nu_t)) z_{t-1}^T$$

דוגמאות לחישוב עברו forward/backward pass

- Forward pass:

$z_0 \in \mathbb{R}^{|\mathcal{V}| \times 1}$: one-hot vector input

$z_1 = W_1 \cdot z_0, W_1 \in \mathbb{R}^{d_1 \times |\mathcal{V}|}, z_1 \in \mathbb{R}^{d_1 \times 1}$

$z_2 = \sigma(W_2 \cdot z_1), W_2 \in \mathbb{R}^{d_2 \times d_1}, z_2 \in \mathbb{R}^{d_2 \times 1}$

$z_3 = \text{softmax}(W_3 \cdot z_2), W_3 \in \mathbb{R}^{|\mathcal{V}| \times d_2}, z_3 \in \mathbb{R}^{|\mathcal{V}| \times 1}$

$l(y, z_3) = \sum_i y^{(i)} \log z_3^{(i)}$

$$\delta_2 = W_3^\top(\sigma'(v_3) \circ \delta_3) = W_3^\top(z_3 - y)$$

$$\delta_1 = W_2^\top(\sigma'(v_2) \circ \delta_2) = W_2^\top(z_2 \circ (1 - z_2) \circ \delta_2)$$

$$\frac{\partial l}{\partial W_3} = (\delta_3 \circ \sigma'(v_3)) z_2^\top = (z_3 - y) z_2^\top$$

$$\frac{\partial l}{\partial W_2} = (\delta_2 \circ \sigma'(v_2)) z_1^\top = (\delta_2 \circ z_2 \circ (1 - z_2)) z_1^\top$$

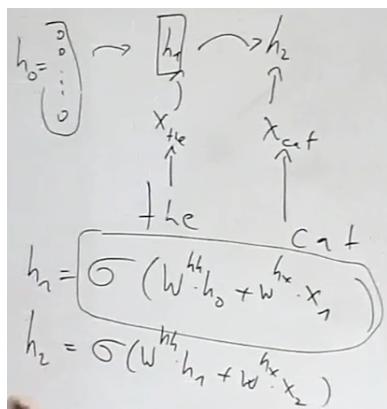
$$\frac{\partial l}{\partial W_1} = (\delta_1 \circ \sigma'(v_1)) z_0^\top = \delta_1 z_0^\top$$

לסיכום: קיבל טוב יותר בכל ש-N גדול, קיבל גם שימוש בייצוגים מימייד נמור בעלי משמעות סמנטית, ובנוסף אנחנו יכולים ליצור decision boundaries מורכבים יותר (לא לינאריים). עם זאת, **עדין יש לנו הנחה מרקובית** שמוטיבית אך ורק על 2 המילים האחרונות, ובאשר המידע נמצא בחולן הרובה יותר גודל זה לא יתפס:

"He is from France, so it makes sense that his first language is..."

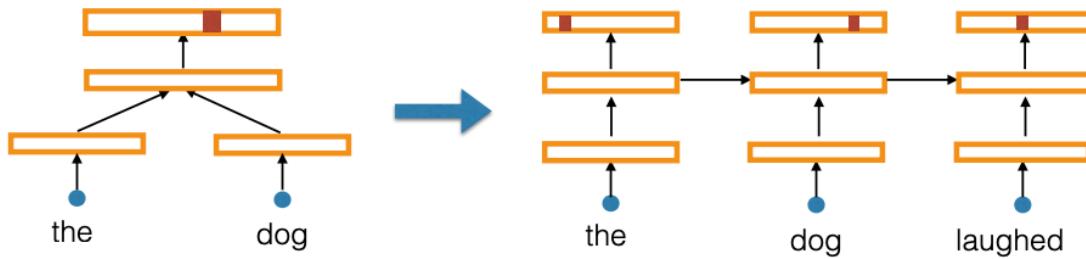
השוואת FFNN ו-LM מושפע word2vec

	Skip-gram	Nerual LMs
Supervision	Self-supervised	Self-supervised
Negative samples	Required	Irrelevant
Embedding matrix(es)	Center & context matrices	Global embedding matrix
Objective (intuition)	Co-occurrence prediction	Next word prediction
Context utilization	Bi-directional	Auto-regressive
Order of words	Permutation invariant	Sensitive to permutations
Scalability (data & params)	Sparsity issues, no additional params	Benefits from data & size
Role of the embedding	Crucial – the only trainable part	Secondary
Embeddings' interactions	Direct interaction (dot-product)	Indirect via projections
Generation	Impossible (why?)	Possible
Horizon	Fixed	Fixed



מוטיבציה: מודל של RNN (recurrent neural network) יכול להימצא בחילון גדול יותר בעבר. נניח שיש לנו קורפוס w_1, w_2, \dots, w_T כאשר $w_i \in \mathbb{R}^n$, והמטרה שלנו היא לקבל התפלגות על פני הלקסיקון עבור המילה הבאה, בכל מקום.

- המעבר מ-RNN ל-FFNN נועז בכך שהוא פונקציה יחידה, כך שלא משנה אילו מילים ראיינו בעבר, בהינתן מילה חדשה נקבל ייצוג חדש. הצעד הראשון זה FFNN: $x_t = W^{(e)} \cdot w_t \in \mathbb{R}^{d_e \times n}$.
- נחשיב ייצוג עבור המילה הנוכחית: $x_t = W^{(e)} \cdot w_t \in \mathbb{R}^{d_e \times n}$.
- בעת נרצה לחשב ייצוג חבוי עבור כל המילים שראיינו מההתחלה עד כה x_1, \dots, x_{t-1}, x_t באופן רקורסיבי. ניקח את ה-ing embedding של המילה (x_t) , ונוסף את **היצוג החבוי הקודם** שמייצג לנו את כל ההיסטורייה שראיינו עד כה (h_{t-1}) : $h_t = \sigma(W^{(hh)} \cdot h_{t-1} + W^{(hx)} \cdot x_t)$.



Input: $w_1, \dots, w_{t-1}, w_t, w_{t+1}, \dots, w_T, w_i \in \mathbb{R}^n$

Model: $x_t = W^{(e)} \cdot w_t, W^{(e)} \in \mathbb{R}^{d_e \times n}$

$h_t = \sigma(W^{(hh)} \cdot h_{t-1} + W^{(hx)} \cdot x_t), W^{(hh)} \in \mathbb{R}^{D_h \times D_h}, W^{(hx)} \in \mathbb{R}^{D_h \times d_e}$

$\hat{y}_t = \text{softmax}(W^{(s)} \cdot h_t), W^{(s)} \in \mathbb{R}^{V \times D_h}$

- נשים לב שהחישוב עבר הקלט ומעבר הפלט זהים למה שבירצנו ב-FFNN. ההבדל היחיד הוא בחישוב עבור השכבה החבוייה. בעת-B-RNN אנחנו מוצאים את אותו החישוב כל פעם, אנחנו מפעלים את אותה הפונקציה שוב ושוב (recurrent) – יש לה את אותן פרמטרים, גם当我们คำ산ים את h_3 בהינתן h_2 וגם当我们คำ산ים את h_2 בהינתן h_1 . ב-RNN יהיה לנו זיכרון חסום ואנחנו משתמשים רק 2 מילימ אחורה, שם הייתה לנו "פונקציה" (embedding matrix) שונה עבור המילה במיקום 2 – i ועבור המילה במיקום 1 – i .

A Simple RNN Language Model

output distribution

$$\hat{y}^{(t)} = \text{softmax}(U h^{(t)} + b_2) \in \mathbb{R}^{|V|}$$

hidden states

$$h^{(t)} = \sigma(W_h h^{(t-1)} + W_e e^{(t)} + b_1)$$

$h^{(0)}$ is the initial hidden state

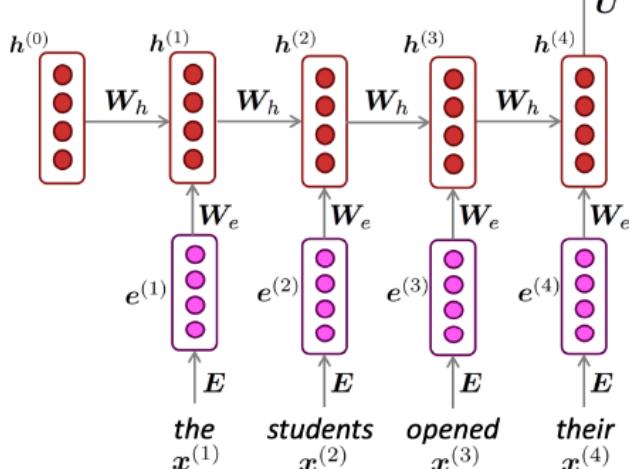
word embeddings

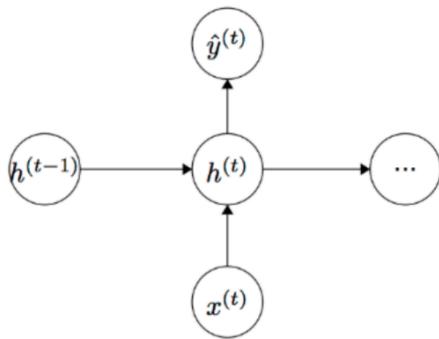
$$e^{(t)} = E x^{(t)}$$

words / one-hot vectors

$$x^{(t)} \in \mathbb{R}^{|V|}$$

Note: this input sequence could be much longer now!

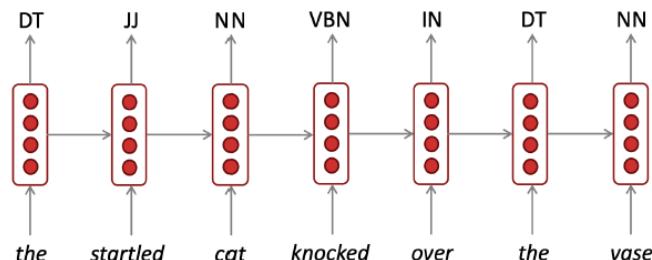


תכונות נוספות:

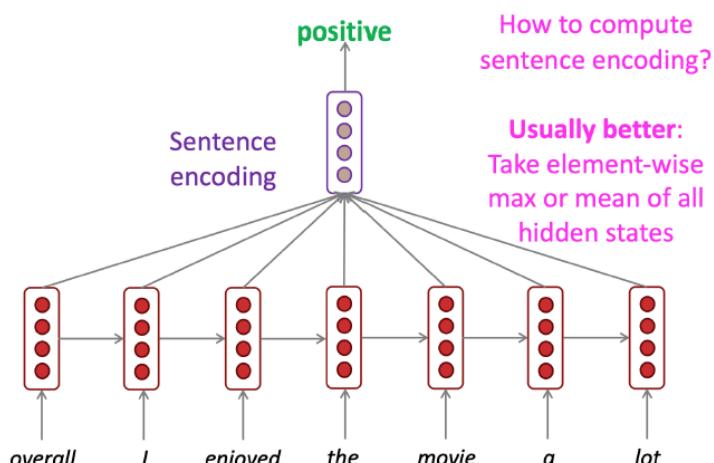
- גם כאן משתמש ב-**backpropagation** (עבור RNN מוכנה גם loss function $CE(y_t, \hat{y}_t)$)
- יתרון: אנחנו משיגים את המטרה שלנו – במיוקם **תלויים בכל המילים** שראינו מzd תחילת המשפט. זה נובע מכך ש- h_t תלוי ב- h_{t-1} ובר רקורסיבית עד שmaguius למילה ראשונה. אנחנו יכולים לעבוד עם קלט **מכל אורך**.
- יתרון: אותן משקלות מופעלות בכל זמן נתון, ולכן יש **סימטריה** בדרך שבה הקלטים עובדים עיבוד.
- חיסרון: החישוב ה-recurrent הוא איטי, גם בזמן אימון. למרות שיש לנו גישה להיסטוריה, זה לא עובד טוב בפועל.

שימושים:

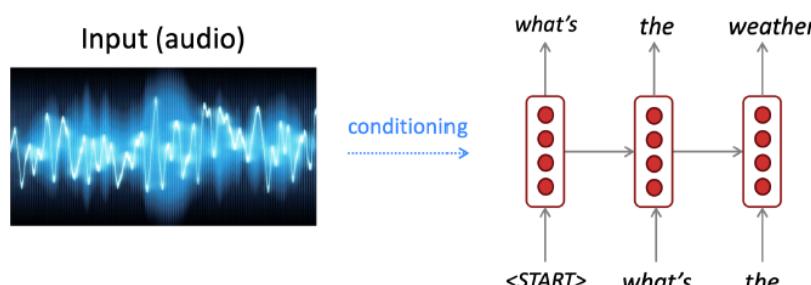
- **תוווגים: PoS tagging (tagging chunk of words)**



- **קלסיפיקציה של משפטים:** פה משתמשים לעתים ב-pool max (לקיחת הקואורדינטה הגדולה יותר בין וקטורים) או ב-pool mean (לקחים ממוצע).



- **יצירת טקסט: summarization ,machine translation ,speech recognition**

RNN-LM

One RNN can “**encode**”
the sequence

A second RNN can “**decode**”
the given representation and
generate a new sequence

A conditional language model

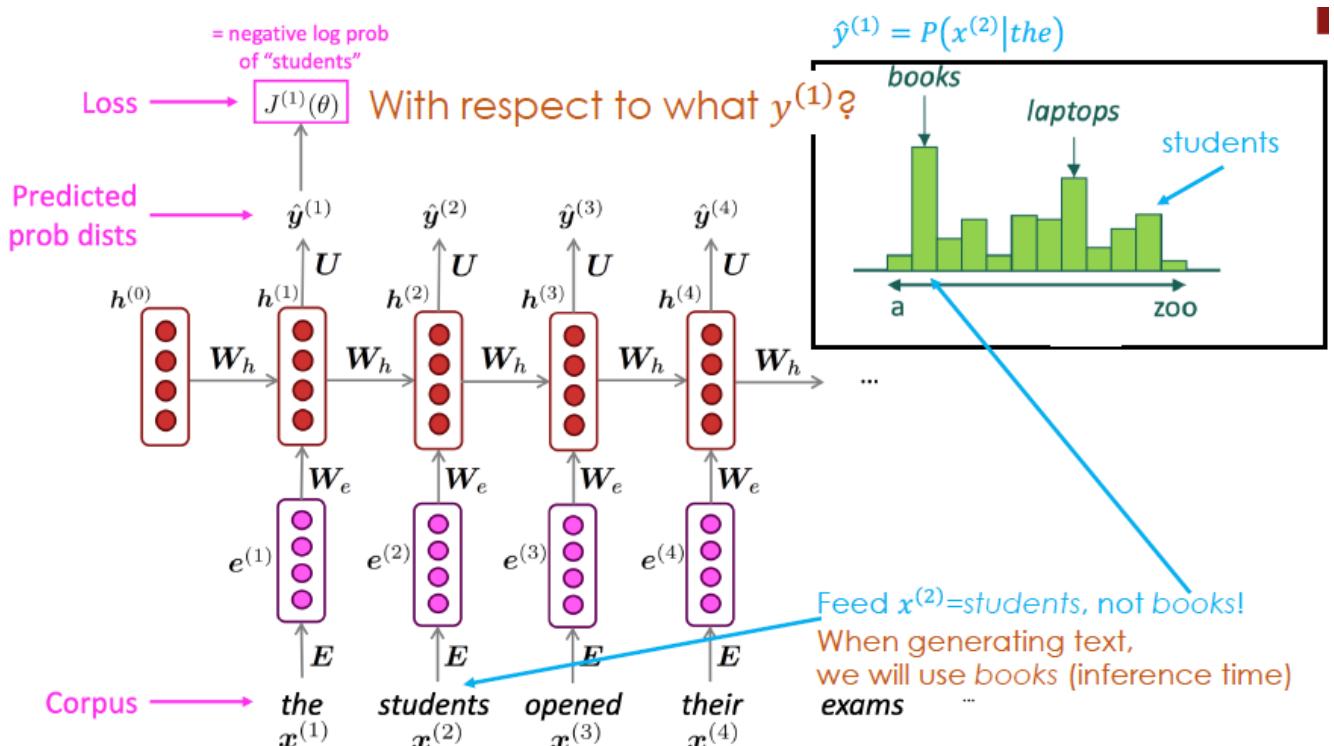
Training RNNs

כפי שכבר רأינו, עבור קורפוס של מילים x_1, \dots, x_t , נחשב את ההסתברות המתקבלת \hat{y} לכל צעד t . נגדיר את פונקציית הפסד להיות CE כמו קודם, בין ההסתברות החזיה \hat{y}_t והמילה האמיתית הבאה (one-hot vector) y_t של x_{t+1} . כפי שראינו במלטה 1, אפשר להסתכל על CE בתור negative log loss:

$$J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = - \sum_{w \in \mathcal{V}} y_w^{(t)} \log \hat{y}_w^{(t)} = -\log \hat{y}_{x_{t+1}}^{(t)}$$

כך נקבל שהפסד עבור המילה הראשונה יהיה בעצם **הסתברות שהרשota מעניקה למילה האמיתית שאחריה – students**. שיטה זו מכונה "Teacher forcing", אנחנו מcriחים את המודל להסתכל רק על המילים שאנו מאמנים אותו עליהם. נמצע את זה על פניו כל סט האימון כדי לקבל את הפסד הכללי:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^T -\log \hat{y}_{x_{t+1}}^{(t)}$$



Model	Uses word-similarities	Unlimited horizon	Training over sequences
-------	------------------------	-------------------	-------------------------

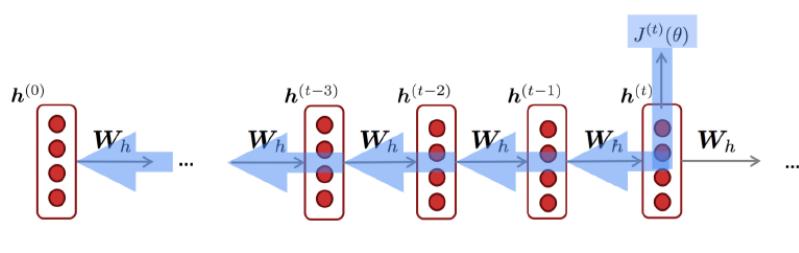
מודל שאומן בצורה זו, בזמן inference הוא ירצה לחזות משווה אחר והוא יסעה מההתפלגות של האימון, מה שיוציאו אותנו לחולוטן מההתפלגות. קונספטואלית הרעיון של RNN טוב, אבל בפועל מאד קשה לאמן את הרשותות האלו. יש שתי תופעות שיכולות לקרות:

- **N-gram**: **vanishing gradients** – הערך של h_t לא משתנה בכלל כפונקציה של x_1 . ב UIControl, המודל אמרו לתמוך בתליות רוחקות, אבל בפועל זה לא ממש קורה. שינויים מ-10 עדים אחריה לא משפיעים על החנן הנוכחי – האם יוכל לתפוס תלויות ארכובות באלו? מאמנים את המודל וזה יכול לקרות מתחת לשיטה.
- **FFNN-LM**: **exploding gradients** – כאן זה דבר יותר מובהן. משנים ערך מסוים בעבר, והוא משנה את הערכים בהווה באופן קיצוני. רואים שהגראדיינטים 10 צעדים אחורה ענקיים, וזה משפיע על הגראדיינט הנוכחי. כאן נגלה שה-loss training לא קтен.
- **RNN-LM**: **Sequential** – לא הצלחו למזער את loss בשינויו לאמן RNN עבור LM, ולכן בוצעו שינויים במודל. עבור h_t מבצעים הרבה מכפלות של מטריצות, ובקרוב נראה את ההצעה שגורם ל-RNN עבור LM לעבוד.

הרחבה - אימון RNN באמצעות SGD

אינטואיציה: נניח שיש לנו RNN פשוט ללא קלט, אשר מתחילה מוקטור h_0 . נגדיר $W = Wh_{t-1} = Wh_t = W^t h_0 = W^t h_0 = Q \Lambda Q^{-1}$ ובר לקבלת $h_t = Q \Lambda Q^{-1} h_0 = Q \Lambda Q^{-1} (Q \Lambda Q^{-1})^t h_0 = Q \Lambda Q^{-1} h_0$. כלומר לאחר t צעדים נקבל את אותה מטריצה רק שהערבים העצמיים הם בחזקת t . לעומת זאת, אם נסתכל על הביטוי שלו בצל' לפני בסיס ה- 0 , נראה כי ביל"ו מוכפל ב- λ_i^t , אבל הראשון הוא גדול מאוד בהשוואה לאחרים (השאר הולכים וקטנים):

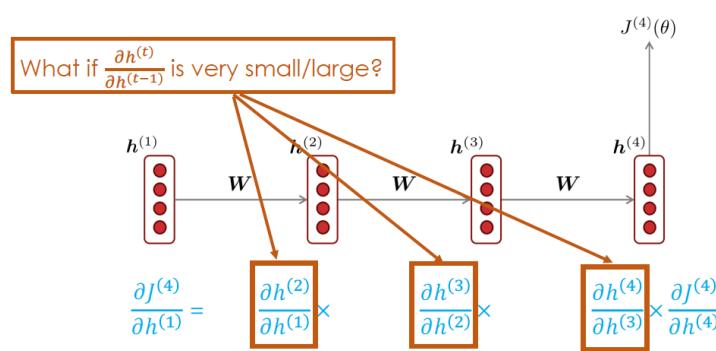
אם $a_n q_n \dots a_0 q_0 = a_0 + a_1 \lambda_1^t q_1 + \dots + a_n \lambda_n^t q_n = h_t$. כלומר, h_t יכול להיות שרירותי, **אחרי מספר צעדים מסוימים הוא יציב בעקבו** q שמתאים לע"מ ה- h_t נעלמת (vanishing gradient) עם הזמן, כאשר אנחנו מכפילים אותה מטריצה W שוב ושוב.



גע למן t , יש לנו את loss שהגדכנו ($J(\theta)$). נרצה לעדכן את W_h בהתאם. הגדריאנט הוא הסכום של הגדריאנטים ביחס לכל פעם שהפעלו את W_h (במעבר בין hidden states) הושנו. נחשב בצורה הבאה:

$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^t \left. \frac{\partial J^{(t)}}{\partial W_h} \right|_{(i)} \cdot \frac{\partial W_h|_{(i)}}{\partial W_h} = \sum_{i=1}^t \left. \frac{\partial J^{(t)}}{\partial W_h} \right|_{(i)}$$

הчисלוב של הגדריאנט באופן זה נקרא **backpropagation through time**.



נסתכל בעת על RNN מתחכם יותר

$$\begin{aligned} \mathbf{h}_t &= W^{(hh)} \sigma(\mathbf{h}_{t-1}) + W^{(hx)} \mathbf{x}_t + \mathbf{b} \\ \text{fonkciyit hahepsod: } \mathcal{L} &= \sum_{t=1}^T \mathcal{L}(\mathbf{h}_t) = \sum_{t=1}^T \mathcal{L}(\mathbf{h}_t) \end{aligned}$$

$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \sum_t \mathcal{L}(h^{(t)})}{\partial \theta}$

$\frac{\partial \mathcal{L}(h^{(t)})}{\partial \theta} = \sum_{k=1}^t \frac{\partial \mathcal{L}_t}{\partial h^{(t)}} \cdot \frac{\partial h^{(t)}}{\partial h^{(k)}} \cdot \frac{\partial h^{(k)}}{\partial \theta}$

$$\frac{\partial h^{(t)}}{\partial h^{(k)}} = \prod_{i=k+1}^t \frac{\partial h^{(i)}}{\partial h^{(i-1)}} = \prod_{i=k+1}^t W_{hh}^T \cdot \text{diag}(\sigma'(h^{(i-1)}))$$

- מפעילים את פונקציית האקטיבציה ואז מכפילים במטריצה.
- פונקציית ההפסד: $\mathcal{L}(\mathbf{h}_t) = \sum_{t=1}^T \mathcal{L}(\mathbf{h}_t)$, שראינו כי הולות בפועל היא negative log loss על ה- h .
- הרעיון הוא להסתכל על loss ביחס ל- θ , ולראות איך התלויות ארכות הטוויה משיפיעות.
- נסתכל על $t = 4$ למשל. הנגזרת שם תהיה הסכום של כל הנגזרות בכל נקודות זמן עד t , תוך הפעלת כל השרשרת. באשר k קרוב ל- t זו תרומה בטוחה ארוכה (long-term contribution).

במאמר המקורי מוכחים גם עבור המקרה של

exploding gradients ו- vanishing gradients. נסתכל ברגע על **exploding gradients**: אם נסתכל על נורמה 2 של מטריצת הגדריאנט, היא חסומה מלמעלה על ידי מכפלת הנורמות. עבור המטריצה של המטריצה $(W^{(hh)})^t$: נניח כי $1/\gamma \leq \lambda_1 \leq 1/\gamma$ כאשר זהה הע"מ הגדול ביותר במטריצה. עבור המטריצה האלבונית: מדובר בגדריאנט של פונקציית אקטיבציה כלשהי (sigmoid, relu, tanh) ולכן זה חסום ע"מ קבוע כלשהו ($1 < \gamma < \frac{1}{4}$). בתנאים אלו – **הנורמה קטנה מ-1**. לכן כל פעם שנכפיל במטריצות הגדריאנט האלו, שוב ושוב, נבטיח שהוקטור שלנו ידוע ונקבל loss.

פתרונות:

- עבור exploding gradients נוכל לבצע **gradient clipping**: מביצעים נורמה של הגדריאント ואם זה גדול מדי, מנורמים אותו: $\frac{g}{\|g\|}$ וזה מפחית מקצב הלמידה. לא ניקח צעד גדול מדי, אלא ננסה **צעד קטן יותר** (באוטו כיוון של הגדריאנט).
- עבור vanishing gradients מכפלת מטריצות הייתה בעיה, אך נסיף גם פעולה של הוספה, וזה יפתר לנו דברים.

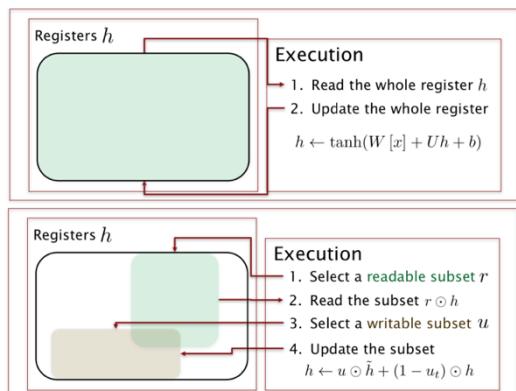
Memory RNNs

מודל GRU:

התובנה המרכזית: הרעיון של **gating**, נלמד שער מסוים שאומר לנו כמה מידע **ידוע להיכנס** כאשר אנחנו **מבצעים חישוב**. נחשב שני שערים, בדומה ל-RNN אבל פונקציית האקטיבציה חיבת להיות sigmoid (1 – המידע זורם, 0 – המידע לא זורם, ויכול לקבל כל ערך ביןיהם). השערים האלה הם וקטורים שנוצרים בתוצאה מהפעלת הפונקציה של המ痴 החבוי האחרון והמילה הנוכחית:

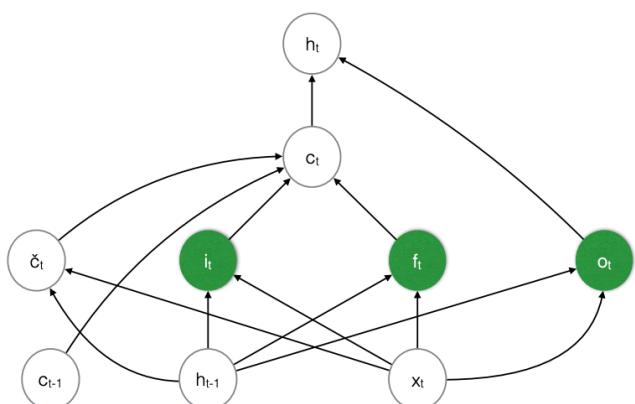
- $z_t = \sigma(W^{(z)} h_{t-1} + U^{(z)} x_t)$: update/copy
- $r_t = \sigma(W^{(r)} h_{t-1} + U^{(r)} x_t)$: reset

נקבל את המ痴 החבוי הבא: $\bar{h}_t = \tanh(Wx_t + r_t \circ Uh_{t-1})$, כאשר נגדיר: $\bar{h}_t = z_t \circ h_{t-1} + (1 - z_t) \circ \bar{h}_t$



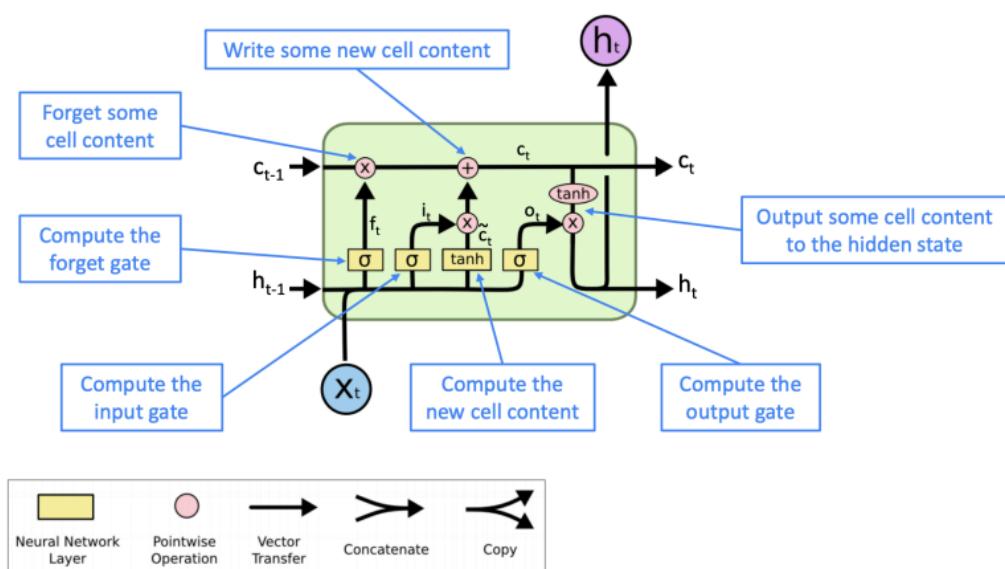
- אם $z_t = 1$, פשוט נעתק את המ痴 הקיים, הגדיאנטים כולם עוברים להלאה. הכל ממשיך כפי שהוא. כדי לקבל קצט אינטואיציה, ב-sentiment analysis הראשון מצביע על ביקורת סרטים, יכול להיות שהמשפט הראשון מצביע על ביקורת חיובית: "this is the best movie that I've ever seen", נרצה לתת לחץ להמשיך לפעוף הלאה.
- אם $z_t = 0$ אנחנו חוזרים ל-vanilla RNN בתוספת r_t . אם הוא 1, זה חלוטין RNN רגיל. אם הוא 0, נשכח מכל ההיסטוריה (מה הייתה לו ב- h_{t-1}), ונחשב רק ב-token הנוכחי x_t .

מודל LSTM:

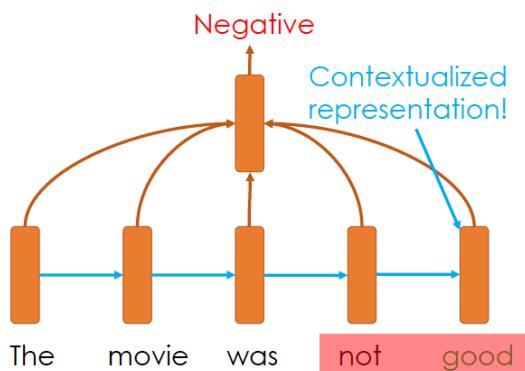


גם כאן יש שימוש בשערים, אך יש חלוקה של שער z בין שער f (forget) לשער i (input). אין את שער z . יש שער חדש o (output) שמחiou בין **הזכרון** (מה נרצה לזכור להמשך באופן כללי, לשימוש עתידי) לבין הפלט (הчисוב עצמו שנעשה עבור הפלט בזמן הנוכחי, השכבה החבוי). כאן נגידיר את **c** בתור **cell state** (בדומה ל- \bar{h} -ב-GRU), ו- h -ב-**hidden state**.

LSTM קצת יותר מורכבים מ-GRU. בפועל, GRU לא תמיד יותר טוב מ-LSTM. יש תיאוריה שאומרת כי יש פועלות ש-GRU לא יודע לעשות (למשל, לא מסוגל למספר עד אינסוף בהינתן מערכת של floats (finite-precision floats).



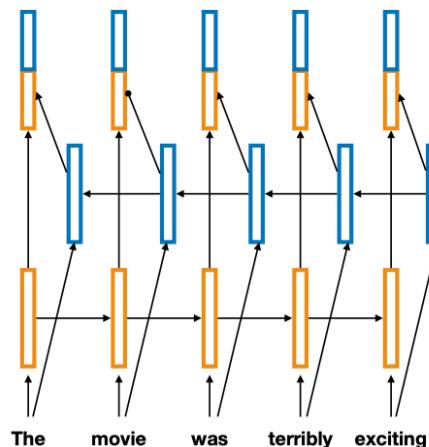
More RNNs



Contextualized Word Representation: נניח שיש לנו את המשפט "The movie was not good" וඅנו רצים לבצע sentiment analysis. באופן כללי, ניקח את הטקסט, ובתהליך כלשהו נגיע לתוצאה של וקטור שמייצג את המשפט, ושבבה נוספת תסוג אותו למחלקה כלשהי – זה דבר מאד נפוץ למשימות NLP (textual entailment, והם בין שני משפטי יש גיראה, סטירה, או אף אחד מהם. מבוטן גם textual similarity).

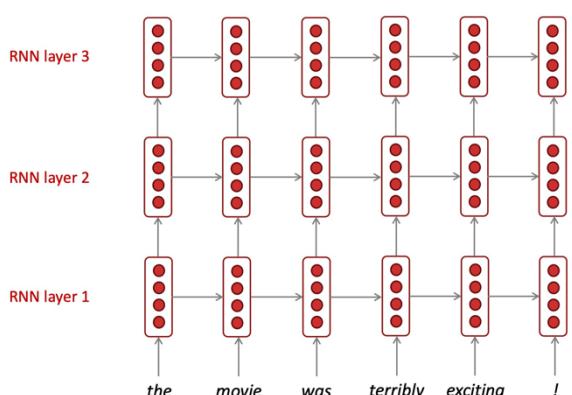
אם נשתמש בskipgram, רק ל-"good" יש משקל **חיובי** שמקשור אליו, שאר המילים די נייטרליות, אין תלות בין המילים. אם נשתמש ב-RNN, נשתמש ב-hidden state שטומן בחובו מידע על מילים קודמות. עד שנגע למילה "good" נזכיר גם את "not" ונ��ואה שכן החיזוי יהיה לרשות **שלילי**. היצוג של כל מילה כאן נקרא **icontextualized**.

נשים לב כי מדובר אך ורק בהיסטוריה מצד שמאל! (כל המילים הקודמות). אם נסתכל על משפט קצת שונה באשר הביטוי בסוף במקום "not good" היה "terribly exciting" ("exciting" ("exciting") שכן היא משפיעה על הקונוטציה של הנאמר (היא משמשת ב-modifier).



$$\begin{aligned}\vec{h}_t &= \text{RNN}_{\text{fw}}(\vec{h}_{t-1}, x_t) \\ \underline{h}_t &= \text{RNN}_{\text{bw}}(\underline{h}_{t+1}, x_t) \\ h_t &= [\vec{h}_t; \underline{h}_t]\end{aligned}$$

BiLSTM/Bidirectional RNNs: נוכל להריץ RNN משמאלי למין, עד RNN מימין לשמאלי, והיצוג בהקשר של מילה במקומות קדומים יהיה שרשו של שני ה-RNN-ים, כך שאנוחנו מתבססים גם על העתיד וגם על העבר. ה-RNN יכול להיות כל דבר (LSTM, GRU וכו'). כל hidden state בעצם מקבל הקשר גם מימין וגם משמאלי. BRNN הפרמטרים לכל RNN הם נפרדים. אפשר להשתמש ב-BRNN ל-sentence classification אבל לא למודל שפה גנרטיבי שמייצר רק על בסיס ההיסטוריה. היה פופולרי מאוד ב-NLP בשנת 2017.



Multi-Layer RNNs: נוסיף עוד RNN שמקבל תוצאה של hidden state מ-RNN קודם, ומעביר את התוצאה לעוד RNN נוספת. כך אנחנו הופכים את הרשת שלemo לעומקה יותר. היצוג של מילה הוא היצוג בשכבה האחורונה (output), או שרשור של היצזוגים בכל השכבות. בפועל נקבל LSTM אחד קדימה ואחד אחורה, עם מספר שכבות.

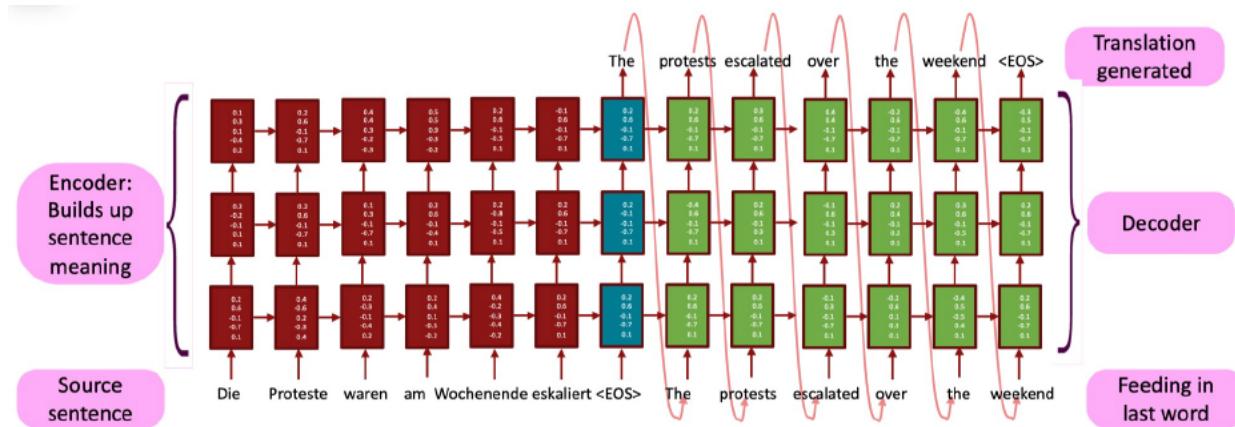
חומרים נוספים:

- הרצתה (חלק שני): [NLP 20A \(JB\) - 3: Language Models](#)
- הרצתה (חלק ראשון): [NLP 20A \(JB\) - 4: NN and Transformers](#)
- פרוייקט עבר: [\[1810.12686\] Evaluating Text GANs as Language Models](#)
- מאמרים:
 - [\[1406.1078\] Phrase Representations using RNN Encoder-Decoder for Statistical MT](#)
 - [On the difficulty of training recurrent neural networks](#)
 - [\[2212.00768\] Simplifying and Understanding State Space Models with Diagonal Linear RNNs \(arxiv.org\)](#)
 - [:LSTM](#)
 - [אתר: The illustrated LSTM](#)
 - [מאמר: LSTM](#)
 - [מאמר: \[1412.3555\] Empirical Evaluation of Gated Recurrent NNs on Sequence Modeling](#)
 - [מאמר: \[1503.04069\] LSTM: A Search Space Odyssey \(arxiv.org\)](#)
 - [מאמר: \[1805.04908\] On the Practical Computational Power of Finite Precision RNNs for Language Recognition \(arxiv.org\)](#)
 - [פרק מטור ספר: 9.pdf \(stanford.edu\)](#)
 - הערות נוספת/טכניות: [Log Loss Function Explained by Experts | Dasha.AI](#)
 - [UMass CS685 \(Advanced NLP\) F20: Implementing a neural LM in PyTorch](#)
 - [סרטון: ResNet \(actually\) explained in under 10 minutes \(youtube.com\)](#)
 - [סרטון: Log Loss Function Explained by Experts | Dasha.AI](#)

טרנספורמרים

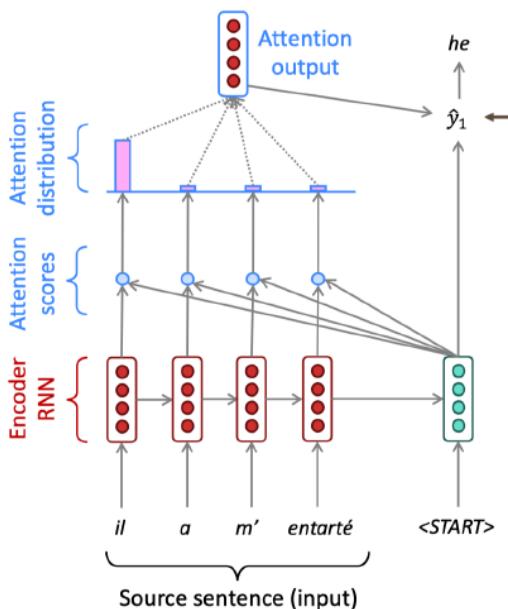
Attention

מוטיבציה: נניח שנרצה להשתמש בארכיטקטורה של Multi-Layer RNN לטובת תרגום מוכנה. יש לנו רצף מילים בגרמנית בקלט, ונרצה לקבל רצף מילים באנגלית בפלט. ראשית, נקודד את הקלט באמצעות RNN, ונקבל ייצוג בהקשר של רצף המילים שלנו. שלב זה הוא ה-**encoder**. לאחר מכן, נרצה לקבל תרגום של המשפט. שלב זה הוא ה-**decoder**.



ה-**hidden state** האחרון שבסוף הקלט, יוצר לנו bottleneck שצורך לתפוס את כל המידע על משפט הקלט שלנו. נרצה של-
decoder שלנו תהיה גישה ישרה גם לעבר, לרכיבים במקום מילים במשפט הקלט.

- במקום שה-**decoder** קיבל את המידע רק מהמילה האחרונה, אפשר להציג שהוא יחוור גם למכבים קודמים. לא ברור איך להשתמש בהা בצורה נכונה, וגם מבחינת סיבוכיות הרשות יש עלות גבוהה לכך.
- דרך אחרת היא לאפשר לו גישה לעבר, אבל עם תנאים מסוימים, ולהחליט אילו מילים רלוונטיות עברו, לאילו מילים הוא רוצה להתייחס, להיות קשוב אליהן – מנגנון זה נקרא **attention**.



ביצוע seq2seq עם attention: הרעיון המרכזי הוא שבלול צעד של ה-**decoder**, נשתמש בחיבור ישיר ל-**encoder** על מנת לתת קשב לחלק מסוים במשפט הקלט.

- נסתכל על כל ה-**tokens** שיש במשפט הקלט, וכל אחד ניתן **attention score**. לרוב נחשב את זה באמצעות מכפלה סקלרית.
- כדי להחליט איפה לחתת יותר קשב (באייה hidden state נוסף אנחנו רצים להתחשב), בוצע softmax כדי לקבל מטמון מוסים ב-**hidden state** הראשון ("he").
- נשתמש בתפלגות כדי לחתת סכום משוקל של כל ה-**hidden states** של **encoder**. הוא יוכל בעיקר מידע על מביבים שקיבלו קשב גבוה. זה מכונה **attention output**.

בצעד הבא, נחשב **attention scores** של מיקומנו הנוכחי ('he') שוב פעם אל מול כל שבות ה-**encoder**, ונקבל התפלגות חדשה. בר נסיק את המילה הבאה ('hit').

הערות נוספת לגבי קשב:

- קשב פוטר את בעיית bottleneck.
- קשב עוזר לנו עם vanishing gradients – יש קיצור דרך למכבים רחוקים.
- קשב מאפשר לנו interpretability – לפי התפלגות הקשב אפשר לראות מה שה-**decoder** הסתכל עליו על פני ההיסטוריה. אנחנו מקבלים באן alignment.

Transformer Architecture

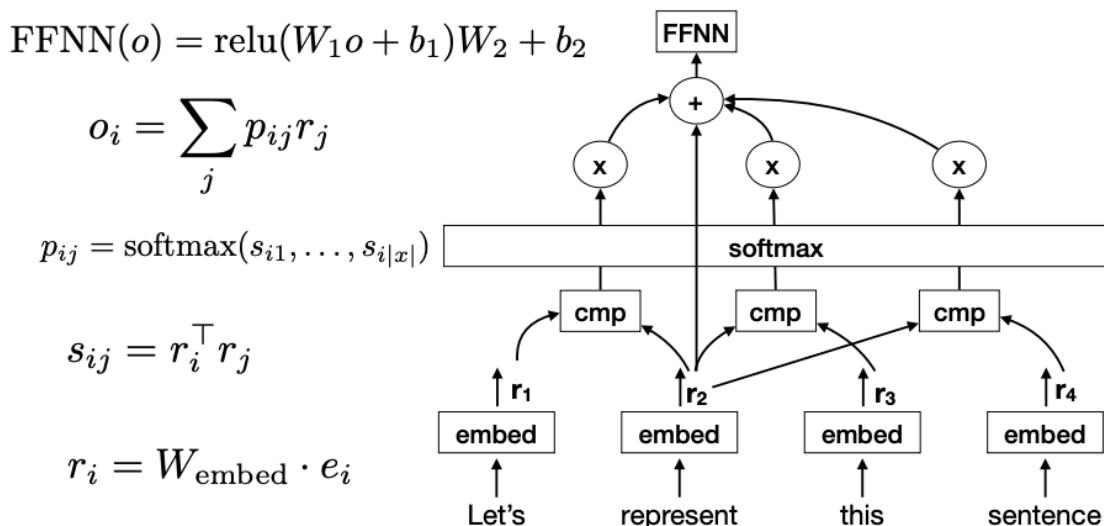
בשנת 2017 יצא מאמר (שלא קיבל הרבה תשומת לב..) בשם "Attention is all you need" (AIAYN), שבו לקחו מושג שהוא קיים (attention), והשתמשו בו כדי ליצור ארכיטקטורת Transformer. המטריה שלנו: להחת משפט, ולקבל ייצוג בהקשר לכל מילה במשפט.

המושג ביצעה:

- בשובדים עם הרבה DATA (כמו ב-Google למשל), RNN-ים קשים מאוד למקובל כיון שככל צורת זרימת המידע היא סדרתי (מצב h_t תליי ב- h_{t-1}), מה שהגיג מאינטואיציה מאוד ברורה של שבירת היסטוריית המשפט עד כה ב-context).
- חיפשו דרך מקובלת לחילוטiano שונה שアイון בה חישוב סדרתי שכזה. זה אפשר להוסיף עוד DATA ואם מודלים מוקובל (בעזרת GPU).
- באשר לטוווח שבו אנחנו שומרים על הקשר, ב-RNN יש לנו את המצב הקודם h_{t-1} , אין לנו הרבה שליטה על המיקום שבו אנחנו רוצים להסתכל, צריך לשמר מידע עד נקודת זמן מסוימת, אבל אי אפשר להציגו על נקודת זמן מיוחדת בעבר.
- המטריה היא לקבל ייצוג מילויים בהקשר (כל מילה תלויה בכל מילה אחרת במובן מסוים), בצורה **שaina סדרתית**.

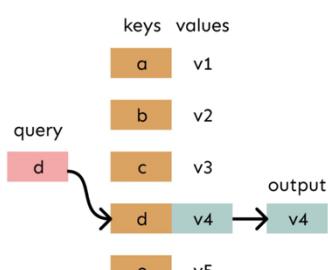
מבנה בסיסי של Transformer Block:

- צעד ראשון הוא ברגיל לבצע $r_i = W_{\text{embed}} \cdot e_i$: embedding.
- הבסיס כאן הוא **self-attention**: נתמך ברגע ביצוג בהקשר של x (זה קורה במקביל עבור כל מילה אחרת). נסתכל על כל אחת מהמילות האחרות, באמצעות פונקציית **compare** כלשהי, נניח ברגע שמדובר פשוט במכפלה סקלרית, ויש מגד שאומר כמה המילים משפיעות זו על זו: $s_{ij} = r_i^T r_j$. הפונקציה צריכה להיות מהירה לחישוב, ומראה במובן מסוים האם r_i אכפת מ- r_j (כמה קשב היא צריכה לתת לה).
- נróż שכבת softmax ונקבל וקטור של הסתבותיות, כמה מילה i נותנת קשב למילה j .
- ניחוס את כל המידע הזה לתוך וקטור יחיד o_i . יש בכך ממוצע ממושך על פני כל המילים, בהתבסס על הציון של כמה קשב כל מילה נותנת לכל מילה אחרת. מקבלים את ה-**attention output**.

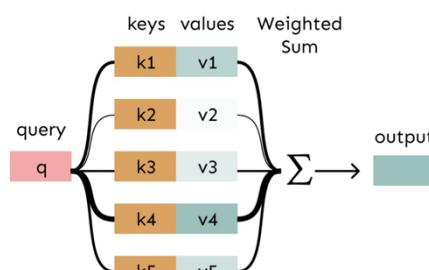


(a) This is fast and parallelizable! (b) words attend to entire context!

In a **lookup table**, we have a table of **keys** that map to **values**. The **query** matches one of the keys, returning its value.



In **attention**, the **query** matches all **keys softly**, to a weight between 0 and 1. The keys' **values** are multiplied by the weights and summed.

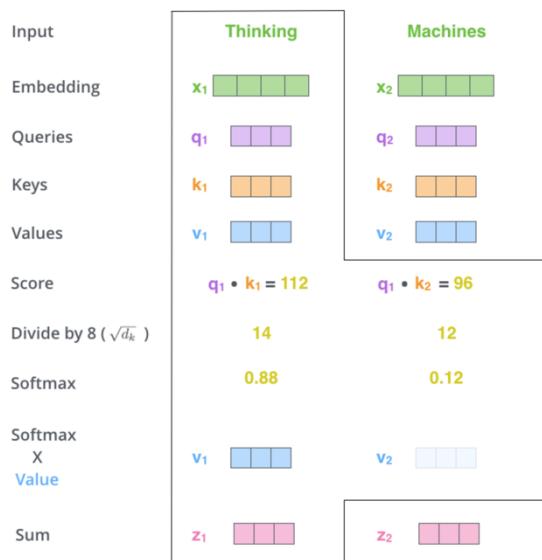
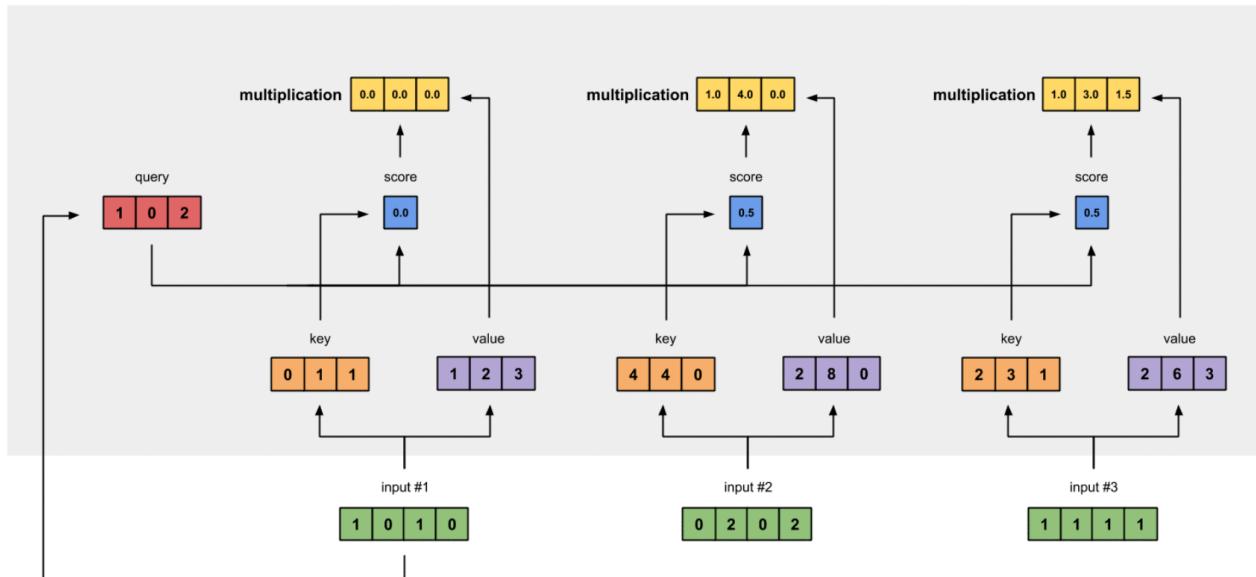


דמיון ל-table **lookup**: נסתכל על dict קלאסי בפייטון, יש לנו צמדים של key:value ו-

query שלנו (הדבר שאנו חפשים) תואם ל-key מסויים. כאן יש לנו fuzzy lookup. בראמה צדו או אחרית (משקל בין 0 ל-1). הפלט שלמו יהיה סכום משקל של ה-values. בחישוב attention, query ה-table נמצא בההתאמה לכל ה-keys. key:query הוא הקלט הנוכחי שלמו, keys הם כל שאר המילים (בודקים התאמה לכל שאר המילים), וה-values הם אותן המילים, כאשר ממושכים לפי ה-softmax ואז סכומים לקבלת ה-output הסופי.

הערות:

- הדגש בכך הוא על **mahirot**. אפשר להשוות את כל המילים (הוקטורים) לכל שאר המילים באמצעות מכפלת מטריצה אחת, במקביל. גם ה-softmax מוחזק במקביל, וגם ה-FFNN פועל במקביל על כל נקודה.
 - יש בפועל **אינטרקציה בין כל שתי מילים** במושפט (pairwise).
 - בכל זה הוא **transformer block** אחד שוב ושוב, עד שנגיע ליצוג סופי.
 - הארQUITקטורה עובדת, אבל תלויה מאוד בפרטים הקטנים.
- עובדת עם מטריצות:** לא לוקחים אממת את i , מבצעים הטלה לニアיריות שלהם לקבלת המטריצות V, K, Q , ומשם אין עוד פרמטרים. כך ניתן לבטא את חישוב ה-attention עבור כל הקלטים במקביל לקבלת המוצע המשוקל. כל מילה יכולה להיראות אחרת כאשר היא משמשת בתפקיד query, key, value. לכן נבצע: $o_i = \sum_j p_{ij} v_j$.



- כעת נוכל לקבל מטריצות לטובת חישוב עבור כל מילوت הקלט (ה-queries) במקביל: $X \in \mathbb{R}^{n \times d}$ מביאה את כל וקטורי הקלט בשורות, $W^Q, W^K, W^V \in \mathbb{R}^{d \times d_k}$ יש לנו את המטריצות לטובת הטלה: $Q = XW^Q, K = XW^K, V = XW^V$. לבסוף תוצאה ה- $A(Q, K, V) = \text{softmax}(QK^T)V$.
- אם נרצה לבצע residual connection למשל, נצטרך לדאוג שההמודדים יתאימו. יש פקטור נורמליזציה שהוא שורש המימד שלו!
- אנחנו מתייחסים באשר לעוברים למטריצות:
$$A(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad n \times d_k \times n \times d_n$$
- כדי לחזור למימד d נוכל להיעזר במטריצה $W^O \in \mathbb{R}^{d_k \times d}$ כדי לחזור למטריצה $\mathbb{R}^{n \times d}$.
- הסיבות לכך קובלות ייצוג בהקשר של כל אחד מהקלטים: עבור קלט אחד צריך להשוות אותו לכל שאר הקלטים - $O(n)$. סה"כ נבצע בכל הקלטים ונקבל $O(n^2)$.



הכוונים (של המאמר AIAYN) הבחינו שימושה אחת (למשל r_2) תיתן קשב בפועל רק למילה בזוזת אחרת (למשל r_4) כי softmax נועה להערכת יחסית כמו max, ומסתכל על **נקודה מסוימת**. הבעה היא שביל מילא מתחשבת:attention (במקרה ספציפי אחד בשכבה הקודמת. לכן הגדרו **multi-head attention**, שבו לכל מילה יש **h** ראשי-attention (attends)

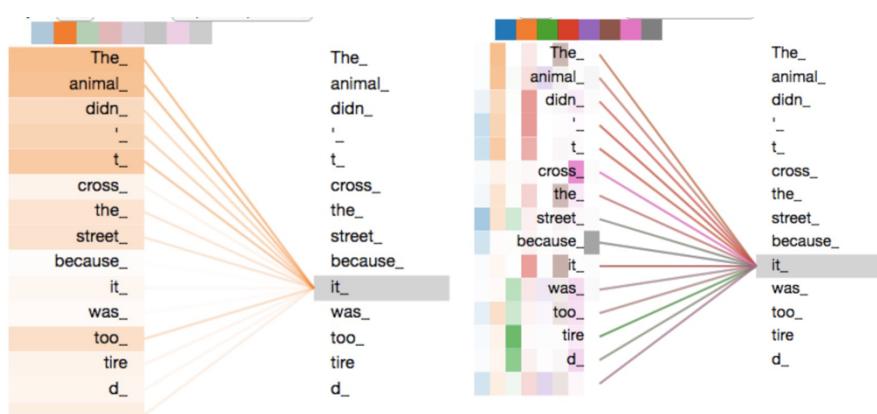
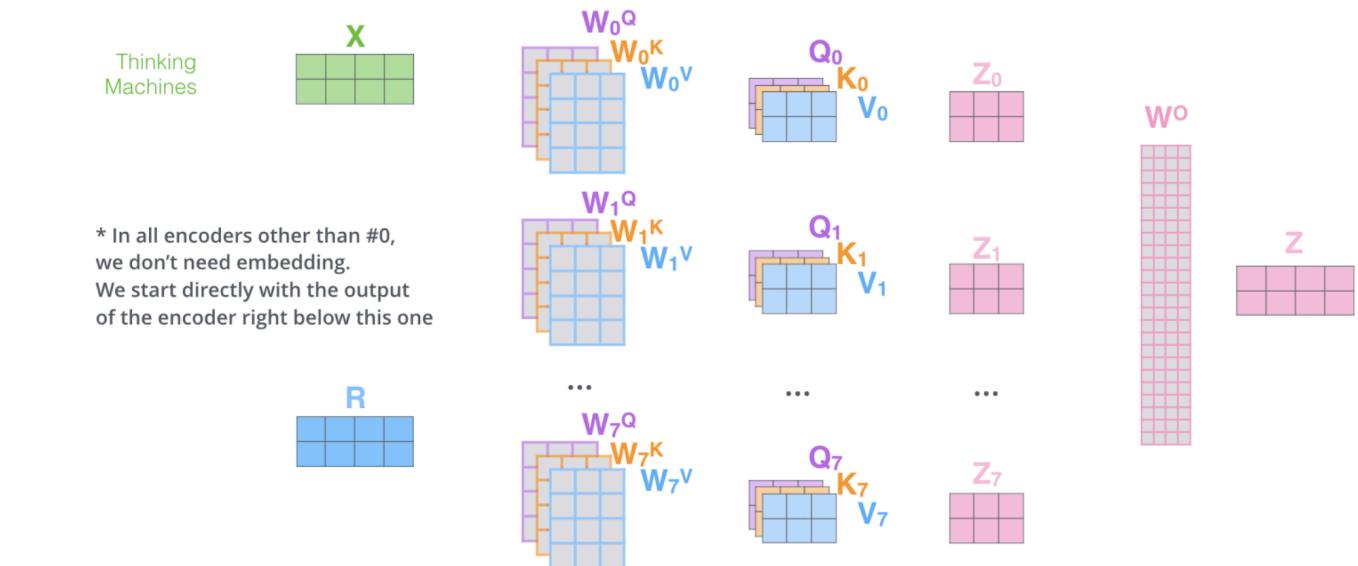
- כל תהליך attention הוא head נפרד. נגדיר $d_k = \frac{d}{h}$
- ככלומר יהיו לנו h מטריצות שונות, אחת לכל head.
- הייצוג הסופי הוא שרשור של תוצאות ביצוע הפעולה בכל head.
- לבסוף נבצע ברגיל את ההטלה בחזרה לממדים המקוריים על ידי מכפלה ב- W^o .

$$\text{MultiHeadAttention}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^o$$

$$\text{head}_i = \text{Attention}(XW_i^Q, XW_i^K, XW_i^V)$$

קודם לכן, קיבלנו את הייצוג z מהקלט x , ובהתאם איה היצוג z עברו הקלט x . בעת יהי z מחראש הראשון, z_1 מחראש השני, וכך הלאה עד שנתקבל שרשור שליהם את z_1 הסופי עברו x_1 .

- 1) This is our input sentence*
- 2) We embed each word*
- 3) Split into 8 heads. We multiply X or R with weight matrices
- 4) Calculate attention using the resulting $Q/K/V$ matrices
- 5) Concatenate the resulting Z matrices, then multiply with weight matrix W^o to produce the output of the layer



ראינו שכבר single-head attention נותנת לנו צורה כלשהי של interpretability. בעצם, זו לא פונקציה אחת, לכל head יש את h -attention map שלו. זה קצת מבלגן את יכולת שלנו לפרש את מה שהמודול עשו, אבל אפשר לו להיות יותר אקספרסיבי ביצוג המילים בהקשר.

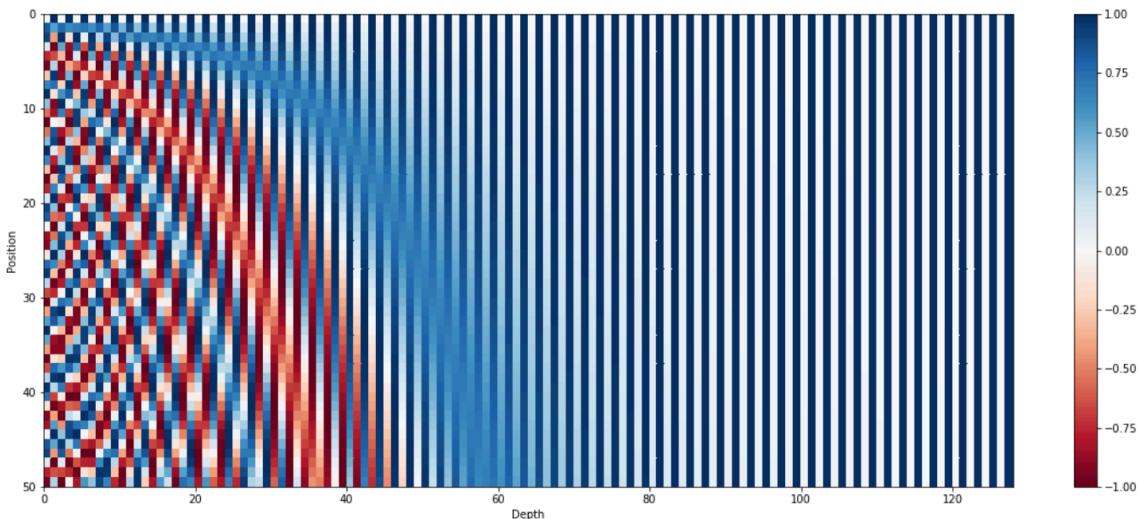
בהרבה מקרים (דוגמה שלא מופיעה כאן) ניתן לראות בשכבות שונות עדות לכך שהמודול לומד "the" co-reference (וידע ש-its" מתיחס ל- "the law" שהופיע לפני), וגם משלימים של פועל מתיחס לנושא שmagע בהמשך ("making" more difficult).

Positional Embeddings: בארכיטקטורה זו אין שום סימון למיקום המילה במשפט, היא permutation-invariant (אפשר להפוך את כל המשפטים בקורסוסים ולקבל תוצאה זהה, לרבות את השורות במטריצה X שלנו). זה שונה מאוד מ-LSTM, שם יש timestamp ואנחנו יודעים מתי כל מילה עברה עיבוד. בשפות רבות יש חשיבות לסדר המילים.

כפתרון, הקלט הוא לא $z_i + p_i$ אלא $z_i + p_i + r_i$ כאשר r_i מייצג את המיקום שלו במשפט (encoding בלשונו של מספר).

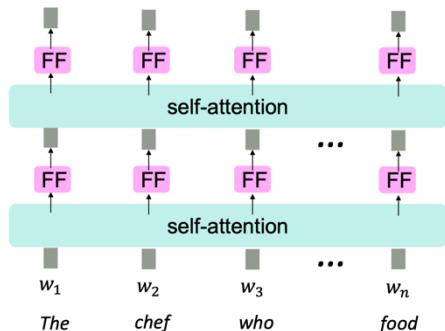
$$\vec{p}_t^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k + 1 \end{cases} \quad \omega_k = \frac{1}{10000^{2k/d}}$$

ניתן לראות שככל position וכל קואורדינטה, הערך הוא שונה (גם באותה שורה, וגם באותה עמודה). כאן המימד הוא 128, וב모ת המילים במשפט היא 50, אז יש לנו 50 וקטורים עבור כל מיקום במשפט.



גישהות עדכניות:

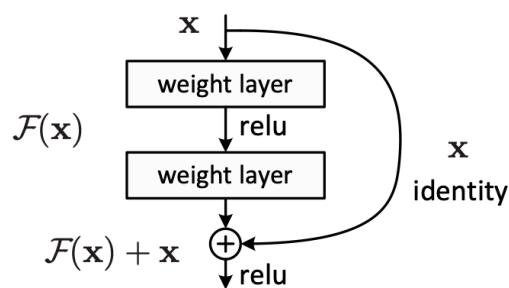
- מה שראינו.cn הוא סטטי, אנחנו תומכים במשפט שמקיף עד 50 מילים, ומימד 128. זו המטריצה שיש לנו לעבוד איתה. לא חייבים לעבוד עם מטריצה נתונה, אלא שהמודול גם ילמד את הערכים שלו, נסיף.
- אוטם כפרמטרים למודול. עדין מדובר ב-fixed length: אם נקבל משפט באורך 51 מילים, המודול לא יידע להכבל.
- אנחנו מביצעים כאן absolute embedding, אך שמיוקם 30 (ביחס לתחילת המשפט) תמיד יקבל את אותו הייצוג. ב-relative embedding מושגים ל-attention score עד רכיב bias שנובע מהמיקום היחסי למילה שאליה אנחנו משווים, ככלمر בהתאם למרחק בין ה-key ל-value מחשבים את attention-alignment (מילה קרובת מימין, מילה רחוקה משמאלה).
- דומה ל-relative embedding אבל משתמשים בסיבוב במרחב ה-d מימדי, כך שלשלוש וקטורים יישתמשו תמיד דזיות שמייצגת את המרחק ביניהם ונשמרת גם לאחר סיבוב.



Adding non-linearities: פרט ל-softmax שקובע לנו את ההתפלגות הסופית, כל הפעולות שביצענו היו הטלות לינאריות באמצעות מטריצה בלשוני. אז היכן מסתתר כל הקסם של רשותות נוירונים? ניקח את הייצוג שקבעו בקשר לשקבוטן באמצעות-h-MLP כמו קודם, ואז נסיף חישוב של ReLU – זו שכבת-h FF שאנו מושגים. שלב-h MLP הוא זהה לכל המילים וקורא במקביל.

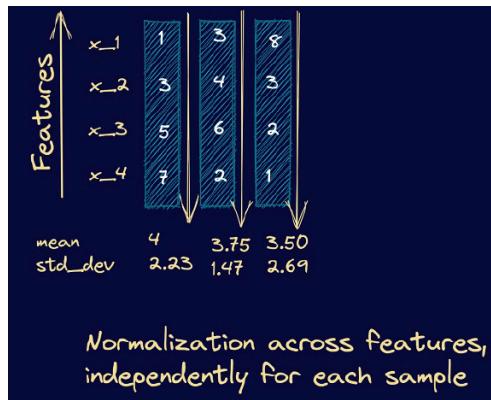
$$m_i = \text{MLP}(o_i) = W_2 \cdot \text{ReLU}(W_1 o_i + b_1) + b_2$$

נשים לב שאנו שומרים על המימד של הפלט שלנו, כך שנוכל לבצע stacking של הפעולות שוב ושוב.



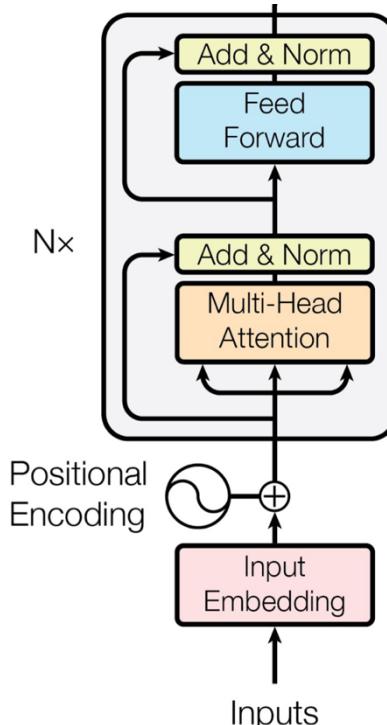
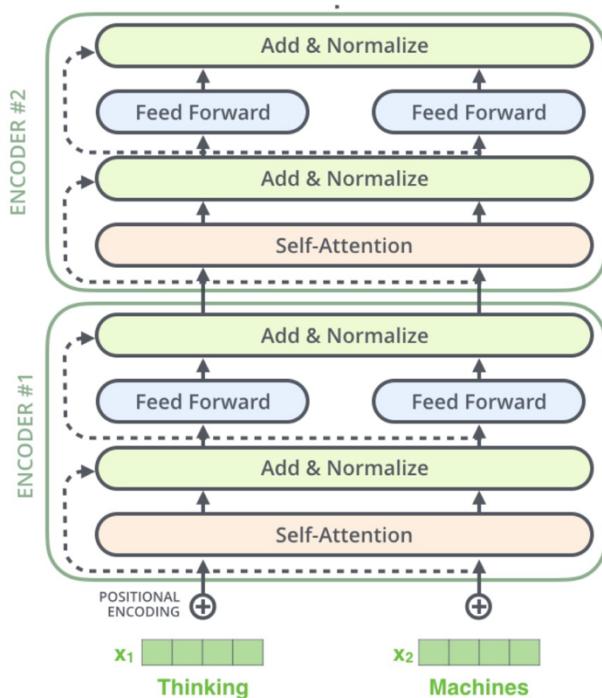
Residual Connections: נזכיר בבעיה של vanishing/exploding gradients ברגע אנחנו לוקחים את z_i ופולטים את o_i , שלאחר מה שנרצה לעשות: בפלט הראשון בוצע $z_i + o_i = o'_i$, ובהמשך קיבל את $o_i + o'_i = o''_i$. יש כאן שימוש בסכימה של הפלט יחד עם הקלט המקורי. במקרים למדוד ישירות את הפונקציה $F(x)$ אנחנו לומדים את $x - H(x)$.

בקשר זה, כל שכבה נקראת תחת-שכבה: FFNN - I - attention sublayer - skip connections. חיבורם אלא נקראים גם sublayer.



Layer Normalization: בשאנחנו מאמנים NN, שימושינו עבורנו לאות כל המשקלים עם ממוצע קרוב ל-0, וסטיית תקן קטנה. יש מקרים שבהם לא ניתן לנורמל כל את הקלט שלנו. מה קורה בשבות הבאות של הרשות? לquo רעיון מתחום ה-mosaics: נחלק את הקלט שלנו ל- b בatches ורק שונרמל רק עבור כל batch בנפרד ולא עבור כל הדאטה. איך אנחנו מגדירים ?batch?

בשיטת normalization layer, נבצע את הנורמליזציה בציר הפיצ'רים שלנו (לאורך הקואורדינטה הראשונה של הוקטו, השניה, וכו').



אולי כל השיטות שעשינו להכיר על מנת להרכיב את הבלוק הבסיסי של הטרנספורמר, והן כולן משתמשות ייחודה. הכל כאן הוא seq2seq, הכל מיוצג בהקשר (bi-directional), הכל ניתן לפחות ללחוט לתהיליך. אין צורך לשיסים, הריצה אינה סדרתית. אגנוסטי לאורך הקלט, מבון בהינתן יציג position מותאים. מה המחיר? $O(n^2)$, בגלל המנגנון של self-attention מילימוט שzierיות כל אחת להכיר n מילים אחרות.

כמות הפרמטרים במודל

ראשית יש לנו את ה-embedding שתורם לנו $d \cdot |\mathcal{V}|$. בעת בכל שכבה:

- בשלב חישוב attention יש לנו h ראשים, משתמשים ב-3 מטריצות הטלה שונות. כל מטריצה ממימד $d_k \times d$.
- לבסוף יש לנו הטלה נוספת כדי לחזור למימד המקורי.סה"כ $d_k \cdot d_h \cdot h = 4d \cdot d \cdot h$.
- בעת בשכבה ה-MLP יש לנו 2 מטריצות הטלה ממימד $d \times d_{ff}$ כאשר לרובה $4d = d_{ff}$. נקבל $8d^2$.

$$\text{סה"כ, כיוון שהגדירנו } \frac{d}{h} = \frac{d}{|\mathcal{V}|}, \text{ אז } L(4hdd_k + 8d^2) = |\mathcal{V}|d + 12Ld^2 : d_k = \frac{d}{h}$$

הערות:

- התלוות ב-L (כמות השכבות, עומק המודול) היא לנארית. עדין מחייבים שהחישוב בשכבה הקודמת יסת沆ים כדי שנוכל לחשב את השכבה הבאה.
- התלוות ב-d (כמות הפיצ'רים בכל וקטור) היא ריבועית. כאן אנחנו מגדילים את כמות המידע שצורך לחשב במקביל בכל בלוק, וכך נדרש יותר GPU.

Masked Language Modeling

רקע: כל מה שראינו עד כה מזכיר matrix embedding, על מנת לייצג מילים בצורה מסכנית באמצעות וקטורים. ישנו מספר מקורי קצה שעילנו להתמודד איתם שעולים להרחב את אוצר המילים שאנו חומכים בו – אותיות גדולות (capitalization), מילים ניקוד וכו'. החלטנו להחליף כל token שלא נמצא באוצר המילים שלנו, בסימן מיוחד (UNK).

גם אם אנחנו מוכנים לשלים בגודל אוצר המילים שלנו [7] – יתפרק בעיית sparsity – אם המילים שאנו מדברים עליהן נדירות, לא יהיו לנו ייצוגים טובים עבורן כי לא נראה אותן מספיק במהלך האימון.

Word	Token(s)
surf	['surf']
surfing	['surf', '##ing']
surfboarding	['surf', '##board', '##ing']
surfboard	['surf', '##board']
snowboard	['snow', '##board']
snowboarding	['snow', '##board', '##ing']
snow	['snow']
snowing	['snow', '##ing']

subword tokenization: במקום לייצג רקה מילה כ-token, ניצג גם חלקו מילה (subwords) (subwords) וכך אפשר מבט יותר מדויק ועדיין על מבנה המילים, חלקים משותפים להן וודרך היצירה שלהם (מורפולוגיה מישוה?). יש הרבה tokenizers בגישה זו, והנפוץ ביותר הוא WordPiece:

- לוחכים את הקורפוס שלם, ומתחילה לחלק אותם לפי character. נדרש כי נרצה שוגול אוצר המילים יהיה קטן מ-N במשהו, ושלכל token תהיה תדירות של F הופעות באוצר המילים.
- נסתכל על זוגות של tokens ונספר כמה פעמים הם מופיעים ביחד. הזוג שהופיעו הכי הרבה יכולים להיליך לשלב הבא בטור token חדש (אוורך של 2 TOKENS). נשמר את הtokens מהשלב הקודם (ing למשל תמיד יישאר, אבל יהיה גם sleeping בהמשך).

בר אין לנו יותר יותר בסימון של UNK. לכל מילה חדשה שאנו רואים, ואין לנו מספיק מידע כדי ללמידה אותה ישירות (את כל המילה), אנחנו מרכיבים אותה מחלקים קטנים יותר שכן למדנו.

מודלים: בשנת 2014-2013 רצוי להשתמש בהרבה DATA כדי לאמן word embeddings, ורק ב-2018 רצוי להשתמש בהרבה DATA כדי לאמן מודלי שפה. ברור שאימון word embeddings הוא טוב, כי אפשר להשתמש בהרבה DATA ולקבל ייצוגים טובים אחרי הריצה של word2vec. אבל – הם לא ייצוגים בהקשר. מצד שני – יש לנו מודלים כמו LSTM שבן מתחשבים בהקשר. הגעה ההבנה שאפשר לאמן מודל שפה גדול על הרבה מאוד טקסט (באמצעות Transformer או LSTM), וכל פעם שמדובר את המודל על קלט מסוים, נקבל ייצוגים מילים בהקשר שיכולים להיות טובים לכל tasks downstream.

$$R_k = \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} | j = 1, \dots, L\}$$

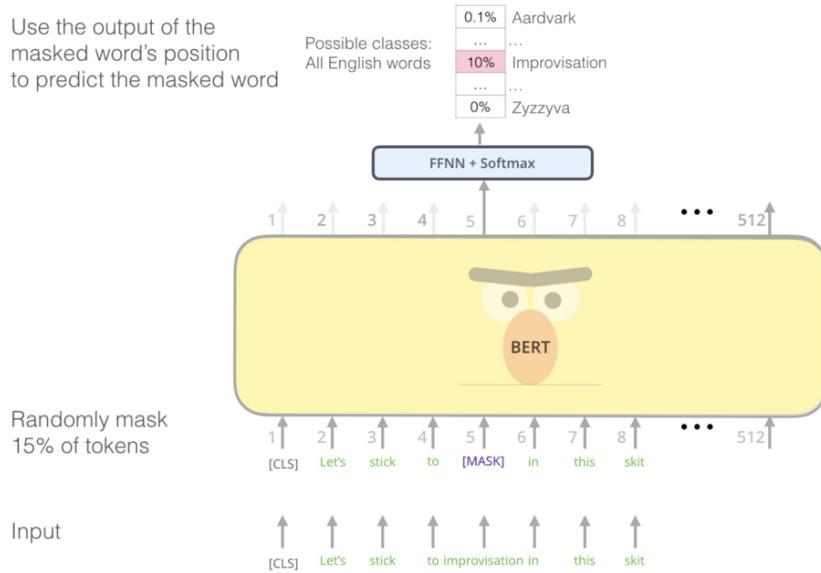
$$= \{\mathbf{h}_{k,j}^{LM} | j = 0, \dots, L\},$$

ELMO: אימנו מודל שפה גדול מבוסס FFNN, בשני כיוונים (forward, backward), בעל L שבבות. בר כל מילה באינדקס k נקבל ייצוג מממד $2L + 1$: אחד עבר המילה שאינה בהקשר, ועוד L מכל מודל שפה. הייצוג הסופי עבר מילה יורכב מביצוע ממוצע משוקל של:

$$\text{ELMo}_k^{\text{task}} = E(R_k; \Theta^{\text{task}}) = \gamma^{\text{task}} \sum_{j=0}^L s_j^{\text{task}} \mathbf{h}_{k,j}^{LM}.$$

- R_k : כל $1 + 2L$ הייצוגים מכל השבבות.
- s^{task} : פרמטר של משקל שמתקיים באוצרות softmax.
- γ^{task} : עוד scaling factor.

BERT (Bidirectional Encoder Representations from Transformers): אנחנו מקבלים ת-קלטים (tokens), ופלטיהם בצד השמי הפלטים. נרצה לעבוד בצורה BERT self-supervised מבודס על encoder transformer שהואencoder? גם על העתיד (בשונה מ-LSTM).



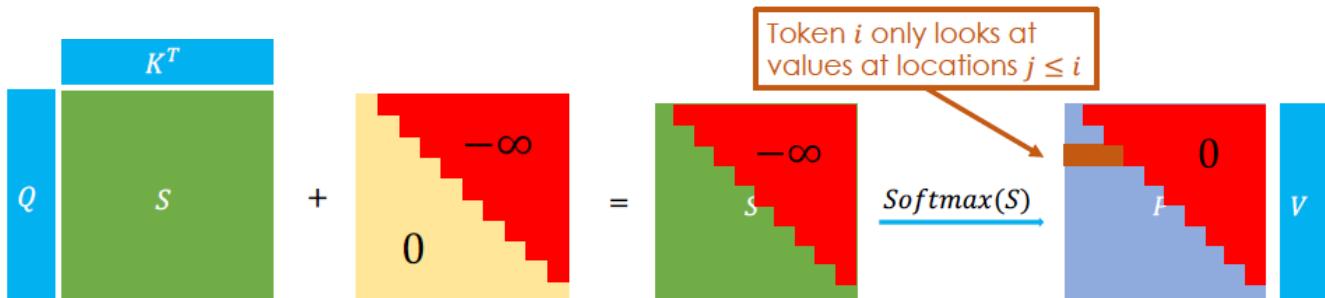
הרעין שנעשה בו שימוש עבור pre-training הוא MLM (masked language modeling): לוקחים token רנדומלי ומסתירים אותו, באמצעות החלפה בסימן המיעוד <MASK>. נספק את הקלט הזה למודל BERT, ונשתמש בפלט כדי לחזות את המילה המקורית שהסתירנו. כדי לחשב את פונקציית הפסד, ניעזר במו קודם ב-CE (cross-entropy) מחושב מול המילה שאנו מוצאים לה (לפניהם ביצעו את ה-MASK).

בעת, נוכל ללקח את המודל שהוא pre-trained ולשימוש בו למשימות שונות: QA, או sentiment analysis וכו'. עבר בול, לא צריך ללמוד את המודל הcoil, והוא צריך רק עוד מספר דוגמאות כדי להבין טוב יותר את עולם הבעיה. שלב זה נקרא fine-tuning והוא מתבסס על דוגמאות מתוצאות.

Transformer: ב-ELMO מודל השפה הינו LSTM. כאשר החליפו את LSTM ב-Transformer, לא הצליחו את המילים בעתיד (ולא לאפשר attentionлокטורים בעתיד), ולבסוף עלייתם במעין לולאת for. נרצה לפטור את זה בדרך אחרת, שבה אנחנו עודין מקבלים את היתרונות של Transformer עם (ריצה במקביל), בתוספת האילוץ של **auto-regressive generation**, וגישה ארך ווקטורי.

causal masking: קודם לקחנו את וקטורי הקלט במטריצה X , צרכנו את המטריצות Q - K , הכפלנו וביצענו softmax, ולבסוף את הסתברויות לקחנו כמשקלים במכפלה עם המטריצה V . ברגע שיחסבנו attention. נשים לב של token יש גישה לערכים שהופיעו אחריו (בעתיד). נרצה להפוך אותו להיות causal, להיות תלוי ווקטורי.

נרצה שככל הערכים שקשורים לעתיד יהיו 0 במכפלה וכן לא יתרמו לנו. לא נוכל רק להעביר אלכソン במטריצה הסתברויות ולשים משקלים בערך 0 מעל האלכסון. לכן, מטריך להציג בברל מטריצה שיעש 0 מעל האלכסון, אבל הכל מנורמל בראו (הסתברות סכמת ל-1). נסיף למטריצה הראשתונה מטריצה אחרת שלא משנה את הערכים שאנו רוצים להתייחס אליהם (בעבר), ומתקינה מאוד את הערכים שאנו רוצים להתעלם מהם (בעתיד).

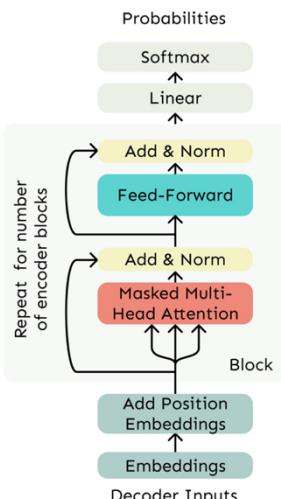


נקבל את אותו transformer block כמו קודם (לו קראו encoder block), רק שכאן אנחנו נוקטים ב-causal masking וכך במקורה זה decoder block.

ב-BERT מביצעים fine-tuning כדי להכין אותו לשימה ספציפית, אבל ב-GPT הכל עבר רדוקציה לשאלת שפה טبيعית ומצפים ממנו למשיך את התשובה, חיזוי המילה הבאה (הוא מודל גנרטיבי – generative).

GPT2 יצא ב-2020, שם רואו את הרעיון של zero-shot performance. עד אז, כל דבר היה צריך לעבור training/fine-tuning על דוגמאות מתוצאות עברו המשימה הספציפית שאנו רוצים לבצע. ב-GPT2, ראיינו את היכולת לשאול את המודל שאלה, ולקבל את התשובה הנכונה, בלי להראות לו אפילו לא דוגמה אחת נספתחה עבור המשימה הזאת, הוא מצליח בניסוי הראשון (zero-shot).

נושא נוסף שעלה הוא in-context learning. המודל מסוגל לפתור משימות שלא היו צפויות לו בזמן האימון.

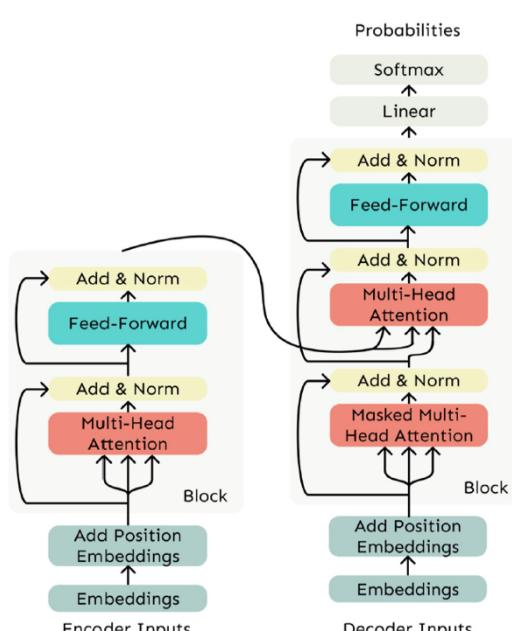


Encoder-Decoder

BERT הוא encoder שנutan לנו ייצוג מילים בהקשר, ו-GPT הוא decoder שיודע לייצר מילים. אמנם, משימות כמו תרגום מכונה, וסיבום, דורשות ארכיטקטורה **שמבצעת שילוב בין השניים**, כמוencoder-decoder. קודם עליינו למצאו ייצוג מתאים לקלט שקיבלנו, ואז לייצר פלט חדש.

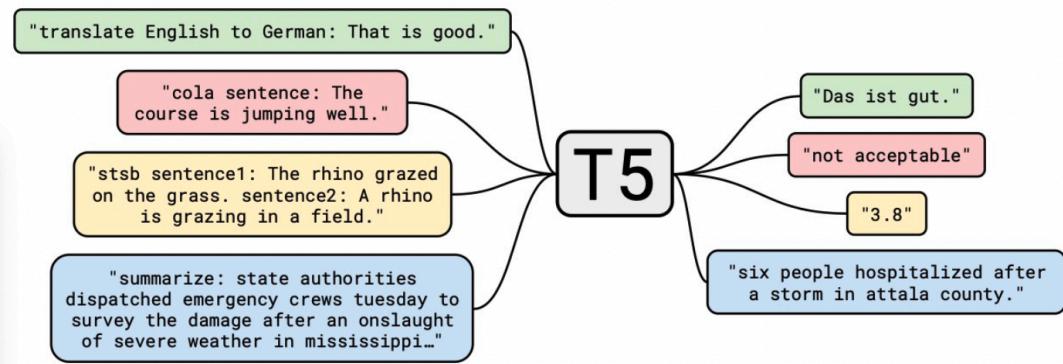
בעור ה-encoder ראיינו self-attention, ובעור ה-decoder cross attention ביצענו masked self-attention decoder block. הרחבה טבעיות היא להרשות ל-encoder-decoder **conditional generation**. לגשת למידע מה-encoder, self-attention cross attention. ב-GPT, הערכים של V , K , Q , מгиינים מאותה המקור (וקטורי הקלט X שעוברים הטלה בלהשי). ב-cross attention, נראים קצת אחרת – K , V מгиינים מה-encoder, Q מгиינים מה-decoder, אין צורך בבייצוע masking כי מלכתחילה כל token ב-decoder ימצא קדימה שאנו רוצים לצריכם למן.

למשל בתרגום מכונה (זמן inference), ניקח משפט קלט בצרפתית (שיעבור דרך encoder), ואשר אנחנו רוצים שהמודול ייצור את התרגום של המשפט לאנגלית, הוא בברל לא יוכל לגשת לעתיד, רק ליזוג של המשפט בצרפתית (חלק זה בבר עבר דרך ה-decoder).





T5: מבצעים אימון מקדים, ולאחר מכן עוד שלב של fine-tuning שמתבסס על דוגמאות מהוות בשמהטריה היא.langauge modeling. ה-prefix של הקלט מצביע על המשימה הנדרשת לביצוע על ידי המודל.



את האימון המקדים מבצעים באמצעות **span corruption**. לוקחים את הקלט, דוגמים כמה רצפים מהקלט (אפשר גם יותר ממילה אחת) ומוצאים להם חסר באלעתות sentinel tokens (אין חשיבות אם הייתה מילה אחת, או שתי מילים) כמו <Y>, <X>. המודל צריך למודד לחזות את החסר באלעתה כמה מילים יש שם מלכתחילה. את המשפט בירוק ניתן בקלט ל-encoder, ואת המשפט באדום ניתן בקלט ל-decoder.

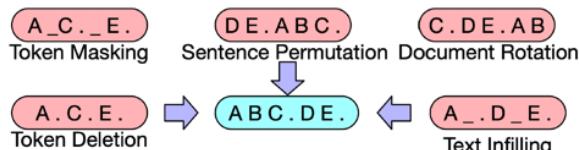
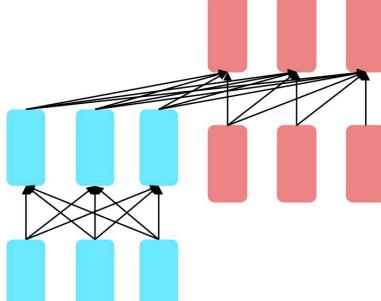
Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!

Original text
Thank you for inviting me to your party last week.

This is implemented in text preprocessing: it's still an objective that looks like **language modeling** at the decoder side.

Targets
<X> for inviting <Y> last <Z>

Inputs
Thank you <X> me to your party <Y> week.



BART: בכל פעם שלוקחים encoder-decoder, אנחנו צריכים למצאו דרך ללקחת טקסט לא מתואג בקלט, לבצע עליו מניפולציות (noising), ואז לבצע חיזוי לקלט המוקורי (denoising) באמצעות denoising-decoder. ב-T5 ביצעו span corruption לא צריך לחזות את כל הקלט, רק למצוא מה חסר. **ב-BART דריש שפהפלט יהיה תמיד הקלט המקורי.** ישן כל מיני מניפולציות שכנית בוציע: מחיקת token, פרMOVטציה, וכו'. המודל שנתקבל יוצר טקסט קוהרנטי בצורה נפלאה.

ואמנם, **נותים להשתמש יותר ב-T5 בהשוואה ל-BART**. העולות לאימון T5: העולות לאימון ה-encoder ה-*decoder*, ניתן להחליט על אורך קבוע S עבור הפלט, נחליט בהתאם איזה span לנקחת מהקלט. העולות לאימון BART: העולות של ה-encoder ה- $O(n^2)$, העולות גם ב-*decoder* וגם ב-cross attention ה- $O(n^2)$.

בנוסף, אין הרבה היגיון לבצע scale לקלטים גדולים יותר, שם הרבה מהמניפולציות בבר לא רלוונטיות (פרMOVטציה על קלט מאוד גדול בבר חסרת משמעות, ולמודל לא תהיה שום אפשרות להבין מה היה הרצף המקורי). יש אורך קלט מסוים שמאנו ואילך BART בבר לא יעיל. את T5 היה קל להרחיב לקלטים גדולים יותר.

מודלים pretrained לומדים מגוון של סטטיסטיות על השפה שעלייה הן אומנו. דוגמאות: שימושות שכוללות מידע טריוויה (עובדות), מידע על מבנה תחבירי, coreference, sentiment וכו'.

ישנן עוד דרכים רבות לבצע אימון מקדים:

- [1911.02116] Unsupervised Cross-lingual Representation Learning at Scale
- [1901.02860] Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context
- [2112.07916] LongT5: Efficient Text-To-Text Transformer for Long Sequences

-
-
-

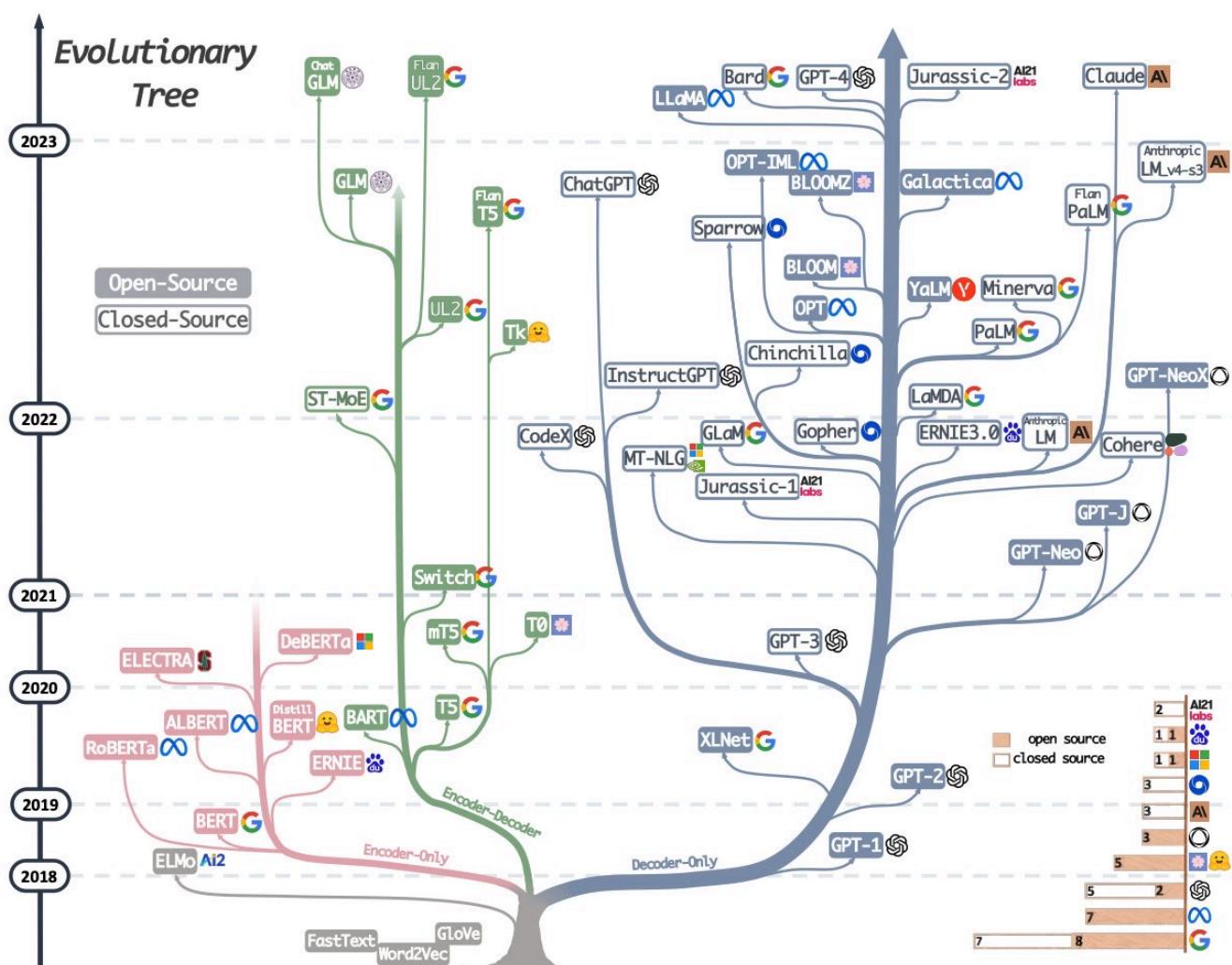
התפתחות המודלים:

לטיכום, **encoder-decoder** היא ארכיטקטורה מאוד טובה כאשר מאמנים אותה כראוי. המודל הגדול ביותר שאומן נכון הוא 2UL. מדוע אנחנו לא רואים עוד מודלים כאלה בשנתיים האחרונות? ובעיקר models decoder-only (כמו GPT?) encode-decoder models היו מיעולים. היום המשימה הנפוצה היא אחרת, אנחנו עברו היי בערך ממשימות קלסיפיקציה, בהן encode-decoder models היו מיעולים. היום המשימה הנפוצה היא אחרת, אנחנו נעצרים ב-AI, אנחנו מנהלים "שיחה" מול המודל:

- אם היינו עובדים מול encoder-decoder, היינו נוטנים קלט ראשון ומקבלים בחזרה פלט ראשון. אם ניתן בעת קלט שני בשיחה, והרי אנחנו ממעוניינים שהמודל ישמר על ה-context של מה שכתבנו לו כבר קודם – המודל יצטרך לבצע שוב תהיליך של encoding על ההיסטוריה – הקלט הראשון, הפלט הראשון, והקלט השני. העולות של תהיליך זה היא גובהה.
- אנחנו חווים לבצע את השרשור הזה כל פעם מחדש ולא להשתמש במה שהוא לנו עד כה, כי **המודל מסתכל לעתיד**, והוא bi-directional contextualized representation של כל הרצף חדש, המילה הראשונה שעוברת את הה-encoding צריכה לדעת מה קורה בעתיד.
- לעומת זאת, במודל שהוא decoder-only, מיציר את הפלט השני, הייצוג של tokens הקודמים (ההיסטוריה) לא צריך להשתנות, כי אין הסתבלות לעתיד. אפשר לבצע caching לכל מה שהוא לנו עד כה, אין צורך לחשב מחדש. העולות לנו לשיחת שכחנו נמוכה בהרבה מ-encoder-decoder.

עץ המודלים:

- בשורש העץ אנחנו רואים **מודלים סטטיסטיים** כמו Word2Vec ו-ELMo כמו RNN.
- ישנו ענף של **encoder-only models**, מודלים שעוזרים לנו למשימות דומות לקלסיפיקציה. אנחנו רצים לייצג את הקלט בצורה מסוימת כך שנוכל בהמשך לענות על משימות סיווג. מודלים כמו BERT.
- התפתח ענף של **encoder-decoder** בוון שימוש ב-generation. מודלים כמו BART ו-UL2.
- הענף המוכר ביותר הוא של **decoder-only models** שבשנים 2022-2023 התפתח משמעותית. כאן יש את כל המודלים המוכרים כמו GPT, LLaMA, Claude, transformer וכו'. הערה: מודלים אלה משתמשים בחלק ה-decoder של ה-encoder-decoder. הם לא בא מת decoders, כי הם עדיין משתמשים ב-conditioned input-causal attention כאשר שישי בו. לפני שואלים שאלת decoding על השאלה שלנו, לפני שהוא מיציר (decoding) מידע חדש.



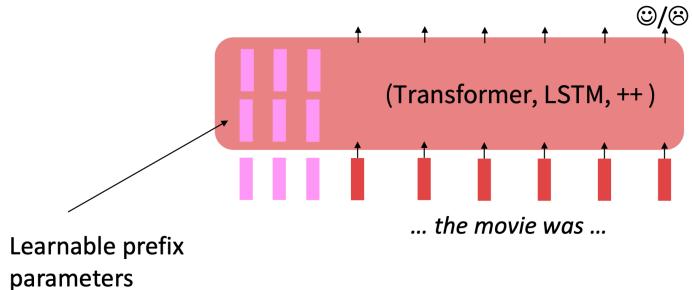


Fine-tuning

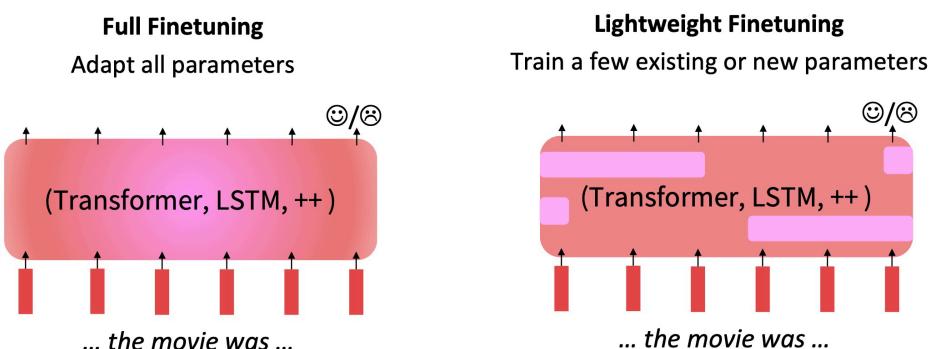
לאחר שיש לנו מודל pre-trained, נרצה להפוך אותו לשימושי עבור task downstream שמשמעותו – לא MLM אלא sentiment analysis. התריליך הזה עובד טוב אבל העולות שלו לא מבוטלות: עדין דוחשים מושגים לחישוב הנ"ל, דרוש תיאוג אנושי של נתונים (דבר שלא קל להציג), וצריך לאחסן הרבה עותקים נוספים כאשר כל אחד מהם מיועד למשימה ספציפית. כל מודל זהה בפני עצמו הוא בודד.

ישנן דרכים יותר יעילות לבצע את התהיליך הזה:

- **Prompt-tuning**: עדין מבצעים tuning אבל "מקפאים" את כל הפרמטרים של המודל עצמו, ולא מאמנים אותם. אנחנו מאמנים initial prompt שימוש ב-prefix ומיצג את המשימה שאנו חוננו ורוצים לבצע. ניקח צעד גרדיאנט רק ביחס ל-prefix שהוכנסנו. עלות האחסון נמוכה יותר, וצריך לשמר רק עותק אחד של המודל המקורי בכל פעם.
Prefix-Tuning adds a **prefix** of parameters, and **freezes all pretrained parameters**.
The prefix is processed by the model just like real words would be.
Advantage: each element of a batch at inference could run a different tuned model.



- **(Parameter-Efficient) PEFT**: לא משנים את הקלט, אבל מוסיפים עוד שכבות עם פרמטרים חדשים ומאמנים רק אותם. (Low-Rank Adaptation) LoRA. טכניקה מוכרת היא adapters. Finetuning every parameter in a pretrained model works well, but is memory-intensive. But **lightweight** finetuning methods adapt pretrained models in a constrained way. Leads to **less overfitting** and/or **more efficient finetuning and inference**.



- **In-Context Learning**: המתודת הבci נפוצה היום. כדי ללמד את GPT את הפורמט והמבנה של המשימה שמשמעותו אותנו, אין לנו גישה למודל עצמו, אלא רק לשיח מולו בזמן inference. ניתן למודל דוגמאות של קלט ופלט רצוי עבור המשימה שלנו, כך שהוא "ילמד" כיצד אנחנו מצפים ממנו להגיב.

The in-context examples seem to specify the task to be performed, and the conditional distribution mocks performing the task to a certain extent.

Input (prefix within a single Transformer decoder context):

```
"      thanks -> merci
      hello -> bonjour
      mint -> menthe
      otter ->      "
```

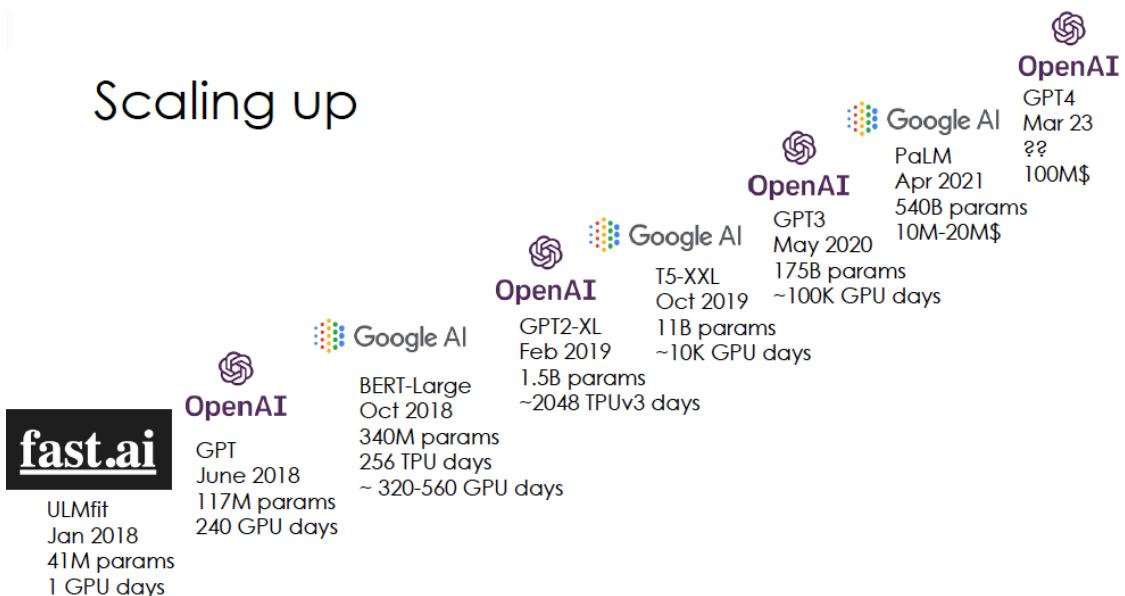
Output (conditional generations):

loutre..."



Improving LLMs

ביכד ניתן להשתמש ב-large pre-trained transformers ולהמשיך לשפר אותם? מנסים היום למצות מהמודל ביצועים כמו שיווט טוביים, גם כאשר משתמשים במודל קטן יותר (עם מספר פרמטרים כזה שמאפשר להתקין את המודל על טלפון נייד). הניסיונות כוללים טיפול מكيف בDATA, אימון מקדים על קטעי קוד, alignment.



בעיות גדולות שבתרו ב-NLP:

- self-attention – הבעיות שלהם עברו transformer היא $O(n^2)$ כפי שראינו במנגנון ה-self-attention
- Interpretability – מה המודלים באמת עושים? האם הם דובריאמת?
- אמינות – LLM-ים נוטים לבצע hallucinations, שגיאות בזוויג לעובדות ידועות וכו'
- Benchmarks – יש חוסר התאמה בין ביצועים על benchmark-ים קיימים לבין ביצועי המודל בשיטה.
- Multi-linguality – תמיינה בשפות נוספות לאנגלית.
- עלויות המשאבים והחישוב...

חומרים נוספים:

- הרצאה (חלק שני): [NLP 20A \(JB\) - 4: NN and Transformers](#)
- פרויקט עבר: [MenakBERT](#) (הוספה ניקוד בעברית).
- המאמר על [\[1706.03762\] Attention Is All You Need](#):transformers [\(ai-blog.co.il.o.i\)](#)
- הסבר על [וניה זה עובד?](#) – בלוג בינה מלאכותית([ai-blog.co.il.o.i](#))
- כניסה לנושא:
 - [The Annotated Transformer \(harvard.edu\)](#) ○
 - [The illustrated attention](#) ○
 - [The illustrated transformer](#) ○
- [\[1902.06006\] Contextual Word Representations: A Contextual Introduction](#): מאמר: ○
- [\[1911.11423\] Single Headed Attention RNN: Stop Thinking With Your Head](#): מאמר: ○
- [\[2402.19427\] Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models \(arxiv.org\)](#): מאמר: ○
- [\[1902.06006\] Contextual Word Representations: A Contextual Introduction](#): מאמר: ○
[\(arxiv.org\)](#)
- [\[2305.13048\] RWKV: Reinventing RNNs for the Transformer Era \(arxiv.org\)](#): מאמר: ○
[BlinkDL/RWKV-LM](#): פרויקט: ○
- [\[1802.05365\] Deep contextualized word representations \(arxiv.org\)](#): מאמר: ○
[AllenNLP - ELMo — Allen Institute for AI \(allenai.org\)](#): אחר: ○
- [\[1810.04805\] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding \(arxiv.org\)](#): מאמר: ○
[The illustrated BERT \(and ELMo\)](#) ○
[nlp - Uni-directional Transformer VS Bi-directional BERT - Stack Overflow](#) ○
[How Transformer is Bidirectional - Machine Learning - Data Science Stack Exchange](#) ○
[BERT , a Bidirectional Transformer | Artificial Intelligence in Plain English](#) ○
[\[1910.10683\] Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer](#): מאמר: ○
[How much memory do you need to train a transformer](#) ○
[\[1907.11692\] RoBERTa: A Robustly Optimized BERT Pretraining Approach \(arxiv.org\)](#) ○
[\[1910.13461\] BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension \(arxiv.org\)](#) ○
[\[2009.06732\] Efficient Transformers: A Survey \(arxiv.org\)](#) ○
[\[2203.15556\] Training Compute-Optimal Large Language Models \(arxiv.org\)](#) ○
[\[2001.08361\] Scaling Laws for Neural Language Models \(arxiv.org\)](#) ○
[GitHub Copilot jailbreak and instructions](#) ○

חומרים נוספים:

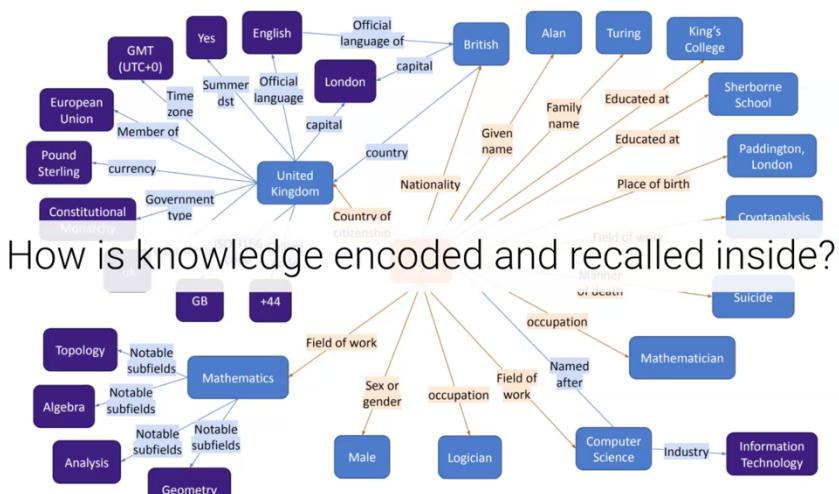
העשרה

(הרצת אורך – מור גבע)

רקע ומשמעות: נסתכל בעת מזויה אחרת על LLM – תחום המחקר של **interpretability**. ספציפית, דבר על ידע בתוך מודול שפה. אין ספק שמודול שפה הנחכו לחלק אינטגרלי מהעברית היומיומית שלו, והם גם נמצאים מאחורי הרבה מוצרים שאנחנו משתמשים בהם. יש שני גורמים עיקריים שהובילו למצב זה:

1. מודלי שפה מבינים ומיצרים שפה טבעית בצורה טובה מאוד, ואנחנו **מקבלים מהם תשובה הגיונית, כאילו היו אדם**. בחברת AI21 AI ערכו ניסוי (social Turing test) שבו נתנו למשתמשים לדבר עם בויטים כמו ChatGPT והבוט שהם פיתחו, ולאחר זמן מסוים של שיחה שאלו את המשתמשים האם דיברו עם בוט או עם אדם אמיתי. ב-40% מהמקרים שבHAM דיברו עם בוט, חשבו שדיבר עם אדם אמיתי. זו דוגמה יפה שמראה את ההתקדמות שנעשתה בתחום בשנים האחרונות.
2. המודלים האלה מכילים מידע עצום על העולם (על בסיס הדאטה מרחב האינטרנט שנונצנים להם לאימון). דבר זה הופך להיות מאוד אפקטיבי בשיחות אitem, ומאפשר לשאלות שאלות מתוחמים שונים ומגוונים. ניתן לראות במודל השפה **בمعنى knowledge interface**, אין צורך יותר ללבת למנוע חיפוש.

עם זאת, **מודלי השפה אינם אמינים מספיק**, קשה להסתמך על הפלט שלהם (unreliable knowledge interface). מחקר הראה שכאשר שואלים את המודל לגבי נושאים פחות פופולריים, הדיווק ממשוערת נמור יותר. תופעה נוספת שניתן לראות במודלים היא **hallucinations**, כאשר המודל מידע עובדתי שגוי. זו בעיה גדולה ויסודית במודלי השפה.

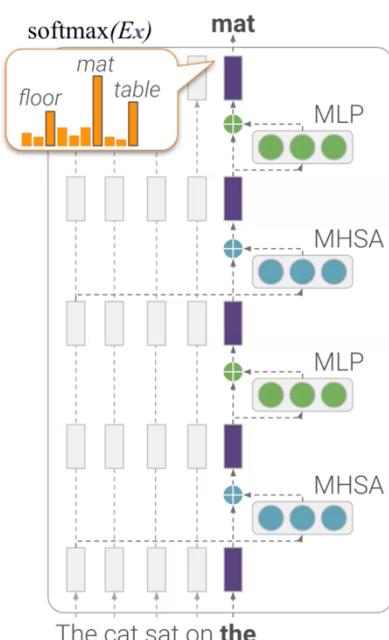


לרוב, כדי לגשת לבעה זו מתייחסים למודל בתרו **black box**, נתונים לו prompts ואז בוחנים את התשובות שלו. אפשר לנסות להתחקות אחרי הידע של המודל (crawl the model's knowledge) "מבחן". אפשר להתחיל מישות התחלה בלחשי כמו **Alan Turing**, ולשאול שאלות שמצביעות על קשרים מסוימים לתשובות (relations), ולהבין כיצד נראה ה-**knowledge base**. אפשר לחזור ולהרחיב את הגרף של הקשרים השונים ולהבין בר את הידע של המודל. האם המודל אכן מקודד את כל התוצאות האלה בראויים? ואיך בצד זה?

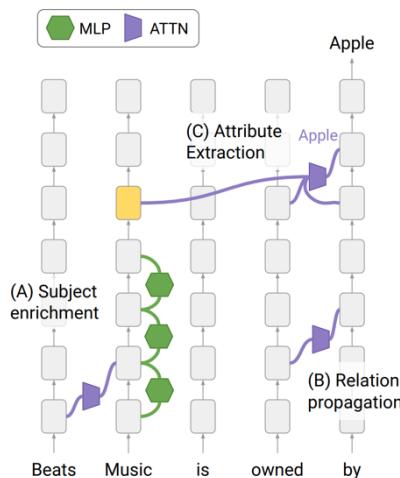
השאלה שמנחה אותנו היא – איך ידע מקודד ומהולץ מהמודל? איך ידע מקודד בתוך הפורמטרים ותחלין החישוב של המודל?

נתמקד ב-**auto-regressive LMs**, ונציג בהקשר זה שתי עבדות:

1. ניתוח של האופן בו מודל מחלץ מידע בזמן inference לפי query מסוים.
2. בהתבסס העבודה הראשונה, ניסיין לעירק ידע נתון בתוך LLM, כמה הוא "חכם" בנושא מסוים (בל' לייצר פלט בפועל).



רעיון טרנספורמרים: דבר על inference-**self-attention** עצמו, לאחר שאימנו מודל טרנספורמר. בהינתן מחרוזת קלט, אנחנו מחלקים אותו לרצף של tokens. כל token מיוצג על ידי embedding vector, ומושם יש לנו שתי שכבות. הראשונה (MHSA), מבצעת עדכון גלובליים בהינתן כל hidden representations previous hidden representations מהשכבה הקודמת. השניה (MLP), מבצעת עדכון חיבויים לokens כל אחד בפועל ב��תי תליי במצבים האחרים. בשכבה الأخيرة, לוקחים את היעציג lokalim אתו כדי לקבל התפלגות מעלה הלקסיקון שלנו. בר' נוכל לדגם את ה- token ההפוך, מティים אותו כדי להסביר למה קורה בתוך החישוב, וכייזד המודל מייצר פרדיקציות מסוימות.



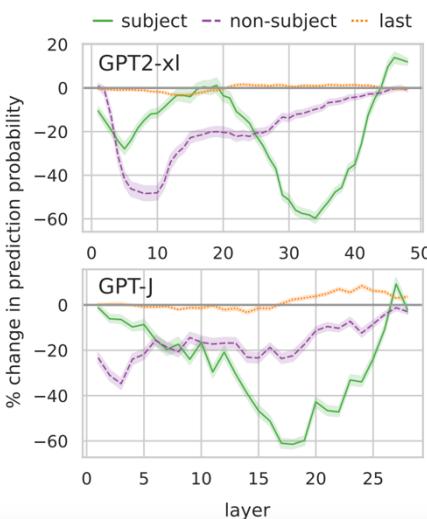
(נקמה): Dissecting factual knowledge recall in LLMs

ה-question שננתח הוא מהסוג: "Alan Turing's nationality is ____". ננסה להבין כיצד המודל מחלץ את התשובה.

mbut על התהילן שנסקרו - three-step attribute recall

- ה-MLP מייציר **ההילן subject enrichment** שמתחליל בבר בשכבה הראשונה. הוא לוקח את הנושא של המשפט (כמו Alan Turing) ויציר לו ייצוג עשיר במידע. חלק מהקטורים במקומ האחורי הנושא (המצב הצחוב למשל), מוקדים מידע רב על הנושא.
- ה-MHSA מפעפע מידע על ה-**relation** (למשל nationality) למיקום האחורי בקלט, ממנו אנחנו לבסוף מחלצים את התשובה.
- בשבות העליונות, המודל מחלץ את התשובה מאותו **subject representation**.

בצד המידע הקרייטי מפעפע: בתחום המחקר שלנו, אנחנו יודעים כיצד החישוב עובד באופן כללי, וסבירו בלשטי בשכבה האחורונה אנחנו מקבלים את החיזוי של Apple. היפותטיות המודול יכולה להגיע לכך בהרבה דרכים: למשל, המודול יכול להתבסס רק על השכבה הראשונה, לבצע על ארגזיה על הייצוגים החובים, לבצע את החישוב בסוף ולהגיע לתשובה. אפשרות אחרות, הוא יכול לשלב את ה-**subject** ומספר פעמים לטור ה-**relation**, מחשב שם את התשובה ומפעפע אותה לשכבה האחורונה.



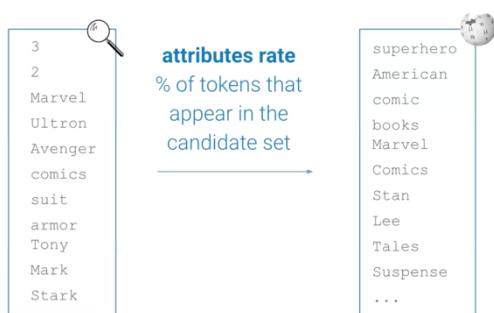
ביצוע attention knockouts להבנת זרם המידע: ניקח השראה מבiology שם מביצעים gene knockouts, ונרצה לבצע את הדבר הבא: נזהה נקודות קרייטיות בחישוב, אם נחסם את ה-**attention** וונמנע מידע לעבור בנקודות מסוימות. **לא ניתן לנתקות מסוימות בחישוב להיות נגויות בחישוב ה-**attention****, לא ניתן למידע לפעוף ממש. אם המידע הקרייטי נמצא שם, נקווה באופן זהה לגלות את זה.

בוצעו ניסויים על שני מודלים: J-P, GPT2-xl, GPT-J. הקו הסגול מראה חסימה של גישה מהמייקום האחרון ל-**relation**. הקו הירוק מראה חסימה של גישה ל-**subject**. אנחנו רואים כי בשכבות המוקדמות (0-10) יש דעיכה בהסתברות לחיזוי התשובה כאשר לא נתונים למודול לתת קשב ל-**relation**. בשכבות המאוחרות יותר, יש ורידה דרסטיבית בהסתברות כאשר מונעים גישה ל-**subject**. מכאן אנחנו לומדים את אותן נקודות קרייטיות שבhan המידע מפעפע לחיזוי התשובה. הגף הזה מתקיים גם אם אנחנו מספקים קלט במבנה קצת שונה: "The owner of beats music is ____".

כלומר, **ה-**subject** מפעפע בשכבות המאוחרות, וה-**relation** בשכבות המוקדמות**.

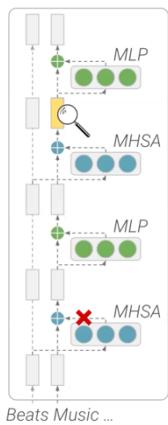
אנו יודעים כי מידע קרייטי מגיע מה-**subject** ומה-**relation**: איזה מידע בדיק נמצא שם? לטובת זה נבצע **projection to the vocabulary**. ניקח את ה-**hidden state** וונסה לפענה אותן בדיק decoding בשכבה האחורונה. במקום לבצע את זה בשכבה האחורונה, נעשה זאת זה לייצוגים בשכבות מוקדמות יותר כדי להבין מה נמצא בהן.

אנו יודעים כי ה-**object** קרייטי (מנענו גישה אליו וזה השפיע על התשובה), ולכן מעוניין לבחון את הייצוג שלו. מה שמשמעותו לואות כאן, הוא שהמודול בכוון הפעולה שלו יודע לחזות את המילה הבאה. לכן בהינתן subject בלבד, נצפה שלאחר מכן יופיע מילים מתאים (VP או PP), ואנחנו מקבלים כאן דזוקה הרבה NP לצד עוד מועמדים אחרים כמו המילה הבאה שהיא מצפים לואות אחרי ה-**subject**, אלא פשוט מילים הקשורות אליו בצורה מסוימת – נקרא להן **attributes**.



נרצה לנסוט לבמת כמה מה-**tokens** שאנחנו מקבלים כ-**subject-related** tokens אלי. זו משימה לא פשוטה ומאוד סובייקטיבית. התחלנו מפסקה בוויקיפדיה לגבי ה-**object**, והזאננו ממנה את tokens הרלוונטיים (content words). נבדוק כמה מהמילים שקיבלו ב-**projection** מופיעות בטקסט שהזאננו, מה שמכונה **attributes rate**.

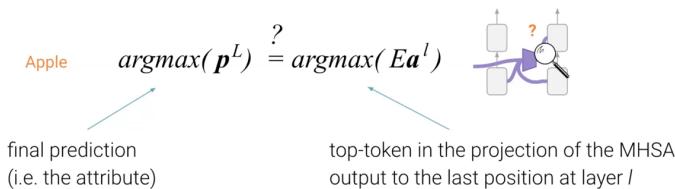
התהילן בוצע בכל השכבות, על לייצוגים שונים מהחישוב, ועובד כל אחד מהם חושב המדד הזה (**attributes rate**). לייצוגים נלקחו עבר וקטורים שונים במיקומים שונים שמעניינים אותנו בקלט, למשל subject-last (המקום של music בביטוי "beats music") או subject-first (music, subject-first) וכו'. הגרפים שהתקבלו מראים שככל שמתקדם בשכבות, **ה-**rate** גדל**. היצוג של ה-**subject** אוסף יותר ויותר מידע רלוונטי (מידע שמוופיע בעמוד הוויקיפדיה של ה-**subject**) ככל שהחישוב מתקדם.



אנו מנסים להבין מהו attribute **magiun**? הם יכולים להגיד מ-3 סוגים פרמטרים: היצוגים של הקלט בשכבה הראונה (embeddings), שכבת ה-MLP, או שכבת ה-MHSA. תחילה מה-MHSA כי זה הכל פשוט, ושם חישבו את attributes rate כמו קודם. התוצאות הראו שאכן הם מזקדים מידע רב על ה-subject, אבל בוודאי לא את כלו. נראה חלק מהתוכנות מגיעות מה-embeddings.

כדי לבדוק האם שאר התוכנות מגיעות מה-MHSA או מה-MLP ביצעו ניסוי נוספת של knockouts. בניסוי חסמו את הפלט משכבה מסוימת (אחד מהשתיים), ובדקו כמה זה משנה על attributes rate בהמשך. המשקנה הכללית מההתוצאות שהתקבלו, היא שאם חוסמים את שכבת ה-MLP רואים הפחתה של 88% ב-attributes rate, בהשוואה להפחיתה של 30% אם חוסמים את A. **מכאן ניתן להסיק כי שכבת ה-MLP ממלאת תפקיד מפתח במבנה ה-factual attributes וחילוץ subject representation**.

חילוץ התשובה בהינתן info:s: הסתכלו על הפלט של ה-MHSA והתהאם שם נמצא כבר את התשובה. נניח כי הוקטור \mathbf{r} הוא וקטור ההסתבריות המתתקבל בשכבה האחרון (L). התשובה תתקבל על ידי קיצית המילה בעלת ההסתברות הגבוהה ביותר. ניקח את הפלט של ה-attention ונסמן א' בשכבה l (בלשוני intermediate), לא الآخرונה), ונוטל את הוקטור על הלקסיקון (באותה גישה כמו קודם) על ידי מכפלה במטריצה E . נקבל וקטור של logits מעלה כל הלקסיקון, ערך עבור כל token. ניקח גם כאן argmax ונסווה למה שאנו מוצאים לקבל מ- \mathbf{r} .



התוצאות הראו כי MHSA מחלץ את attribute לעיטים יותר קרובות (כמעט 70%) מאשר ה-MLP (במעט 31%). וכך **ה-MHSA הרבה יותר קריטי בחילוץ התשובה**.

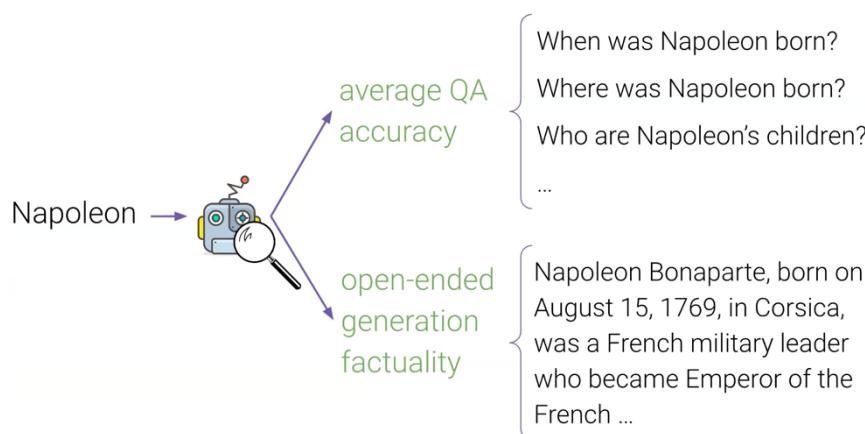
ביצעו הרבה ניסויים כדי לוודא שגם הגינו לתוצאות ברות ממשמעות. אחד הניסויים בא לבדוק האם יכול להיות שבתא ה策, apple, בבר מדורג במועדם הטוב ביותר. אם מסתכלים על הטבלה של ייצוג subject ובודקים כמה נמוך המועדן מדורג (0 זה ה-top, המעודן המוביל ביותר לתשובה), התקבל rank נמוך מאוד. אך, ה-MHSA לוקחת את ייצוג subject וודוחפת את התשובה הנכונות להיות יותר ל-top. איך הוא עושים את זה?

השתמשו גם בשיטה של קיצית הפרמטרים של head וניסוון להטיל אותם על הלקסיקון ולהבין מה קורה שם. מה שראו, הוא שאפשר לזהות attention heads שמקודדים factual knowledge (למשל ב-GPT-2 דיזהו ראש שמאמה בין מוצא למדינה כמו Finland → Chinese Yuan).

[מאמר] Estimating knowledge from internal representations

רקע ומוטיבציה: נזכר בבעית *slogans*: ניסוחים מהמודול לייצר ביוגרפיה של אדם מסוים, נוכל למצוא בכל מין מקומות בטקסט שנוצר אמריות שגויות עבדתיות. כדי להתמודד עם זה, יש ניסיונות לוודא את הפלט של המודול מול מקורות חיצוניים (כמו ויקיפדיה). גישות יותר עדכניות נותנות למודל לייצר מספר פעמים את התשובה, ולבדוק **consistency** בין התשובות. בהינתן כל מה שראינו במחקר הקודם, האם יש לנו דרך במודול "ודע" דברים בנושא מסוים, כמה *hallucinations* יהיו לו, ורק על בסיס החישוב הפנימי שלו – בלי לייצר שם טקסט?

נניח שיש לנו נושא – "נאפוליאון" שנרצה להכניס למודל, ולהבטיח על החישוב בפנים. אנחנו רוצים להעיר שני דברים:



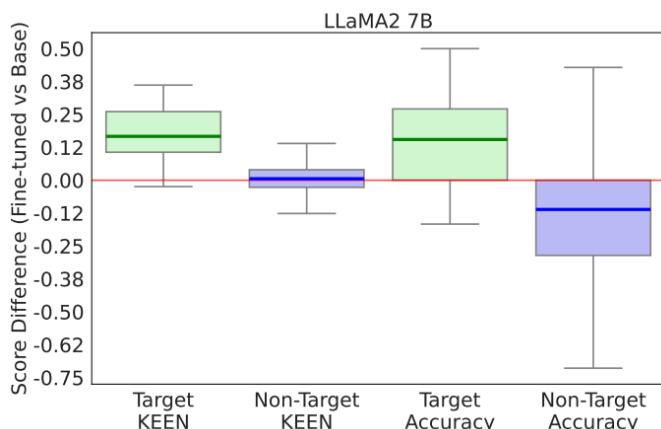
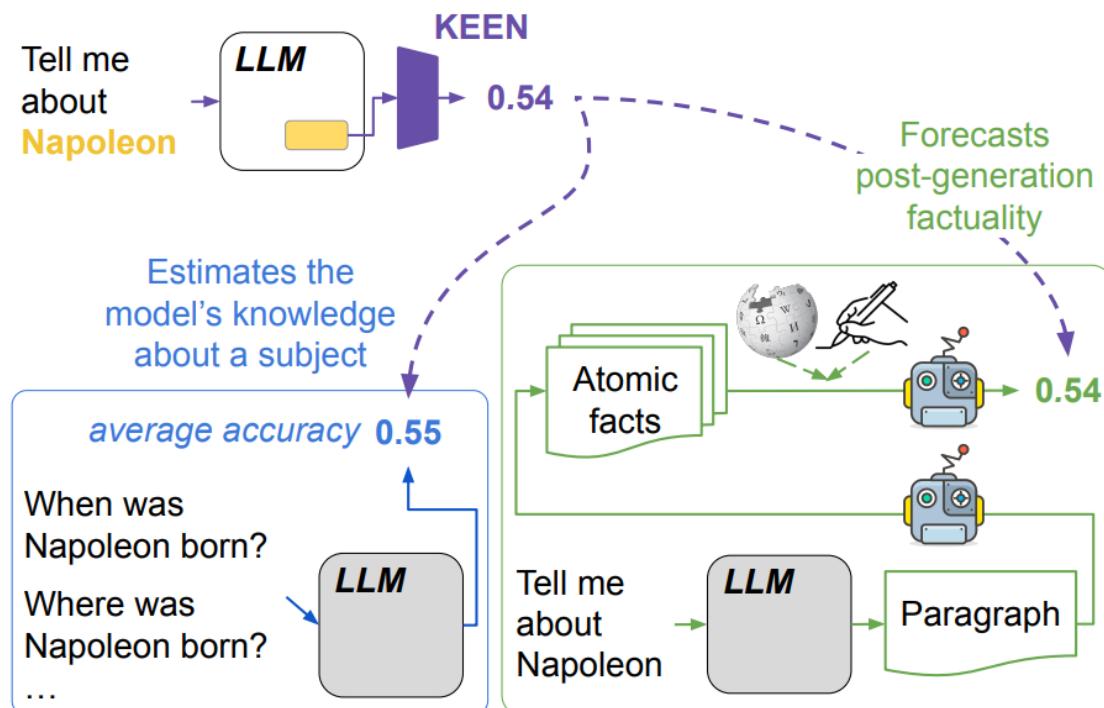
1. **QA accuracy:** אם היינו שואלים את המודול שאלות לגבי הנושא הזה, על כמה מהן הוא היה עונה נכון? כאמור מודדים את הדיקוק של המודול (בלי לייצר שום פלט בפועל).

2. **factuality:** אם היינו נתונים למודול לייצר פסקה שלמה על הנושא, האם נוכל לדעת מראש מה מהטענות שיכתנות שם באמת נכוןות (שוב בלי לייצר את הפסקה בפועל).

ראינו כבר קודם של-project subjects נתון המודל בונה ייצוג מסויים, ומחזיק מידע לגבי. אנחנו משתמשים על כך שההkontext זהה מקיים הרבה **attributes** של-the-subject, אך בנוסף אנחנו יודעים **קל להלץ את attributes** הלו באמצעות פונקציה לינארית – אין צורך בפונקציות מסובכות מידי.

בעבודה זו, אימנו probe (מודל קטן מבוסס נוירונים), שבහינת subject representation חוצה את 2 התכונות שהגדרכנו קודם (accuracy, factuality). בנו דאטא לאימון עבור ה-probe (מושאים ותוצאות ה-factuality), ואימנו את המודל בשם KEEN. ביצינו אבלציה של KEEN מול מודל FActScore ומול QA Accuracy. נראה הקורלציה יחסית גבוהה, ונראה ש-KEEN מצליח לשערק את התכונות האלה שרצינו בצורה טובה.

דבר נוסף שראו, שיש קורלציה עם התנהלות נספנות של מודלים. למשל, מודלים מאומנים יותר להימנע מתשובה (hedge) כשהם לא מסוגלים לענות. זה מצופה לקרות כאשר המודל יודע פחות (less knowledgeable).



ניסוי נוסף שבוצע הוא לגבי causality. אם ה-probe שלנו תופס מידע מהמודל יודע לגבי נושא מסוים, מה יקרה אם נשנה את המידע על הנושא הזה, האם KEEN ישקוף את זה. בתוצאות ניתן לראות את הקטגוריות השונות בציר ה-*x*: Target (לעומת הנושא שאינו עליית המודל (לעומת האחרים, Non-Target). נראה שגם KEEN וגם Accuracy גודלים בהתאם לכך שהמודל לומד יותר לגבי הנושא הספציפי (KEEN משקף גדייה ב-accuracy).

לGBTX נושאים אחרים, ה-accuracy יורד (! catastrophic forgetting). אידיאלית נרצה שגם KEEN יקט. ההיפותזה הנוכחית לגבי זה, היא שגם אם נאמן מודל בר-sha-accuracy יורד (המודל שוכב מידע לגבי נושא מסוים), המידע לא באמת נעלם מהתמודל, רק יותר מtgtגר עברו להלץ אותו.

3 – מושימות עדכניות

Natural Language Generation

מוטיבציה

עד כה התמקדנו במשימה של הבנה (**understanding**) בחלק מעיבוד שפה טבעית – לקיית טקסט (אוסף של סמלים דיסקרטיים), ייצוג שלו והבנתו במודל ML בשמהטרה היא לגרום למודל **להבין את הטקסט** ולהוציא בפלט תשובה/ביקורת כלשהי. בעוד, בדבר על מושימת הייצור (**generation**) – איך נגרום למודל **להפיק טקסט** שימושי עבורנו.

נזכיר לא מעט דרישות בפני מערכת זאת שעבדת עם שפה טבעית מבון – הטקסט שהמודול מוציא צריך להיות>Dקדוקי, Kohonen ושותף, וכן עובדתי, בסגנון מסוים, לא פוגעני... איך ניתן לה?

אם ניקח צעד מספר שנים אחורה (לפני עידן GPT), rob המשימות היו מבוססות סיווג (**classification**). המשימות שבהן נתקלנו ב-GNL היו תרגום מכונה, סיכום, עזריים דיגיטליים, מעבה על שאלות פתוחות. בהמשך, התחלנו לראות משימות יותר יצירתיות שככלו כתיבת סיפורים ושירים, נתינת כותרות עבר תמונות וכו'. כאמור, NLG אינו דבר טריוניyal, עד GPT מודלים לא יכולים לכתוב סיפור שאנחנו בבני אדם נשפטו בקריא והגוני.

ນמקם על סקירה את המשימות השונות בתחום:

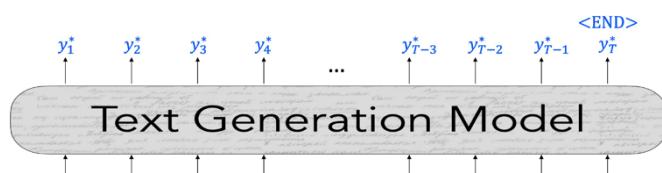
- בצד שמאל נמקם משימות שבהן יש לנו מספר יחסית תחום ומוגדר היטב של מחלקות שנוצרות לבחור בינהן. גם בתרגום, הדומין הוא יחסית סגור על אף שניתן לכתוב תרגומים שונים עברו אותו המשפט. גם בסיכום יש דרגות חופש מסוימות.
- בצד ימין נמקם משימות שהפלט שלהן הוא הרבה יותר פתוח ורחיב, טקסט שהוא יצירתי וgemäßיש יוכל ללבת להרבה כיונים שונים.

Less Open-ended

More Open-ended



Decoding Methods



התהילר הבסיסי**:** יש לנו מודל כלשהו שמייצר טקסט. בזמן אימון נרצה למקסם את ההסתברות לקבל את token האמתי הבא t בהינתן כל המילים שהופיעו לפני t $\{y_i\}_{i=1}^{t-1}$.

- נגידו פונקציית הפסד של negative log likelihood אותה נרצה לזרע: $\mathcal{L} = - \sum_{t=1}^T \log P[y_t | \{y_i\}_{i=1}^{t-1}]$.

בזמן ייצור, בכל נקודה זמן t המודל מחשב וקטוור של ציוני token (scores) עבור כל token שקיים אצלנו בלקסיקון \mathcal{V} , נקבל $\pi_i \in \mathbb{R} \in \{y_i\}_{i=1}^T$ כאשר f הוא המודל שלנו.

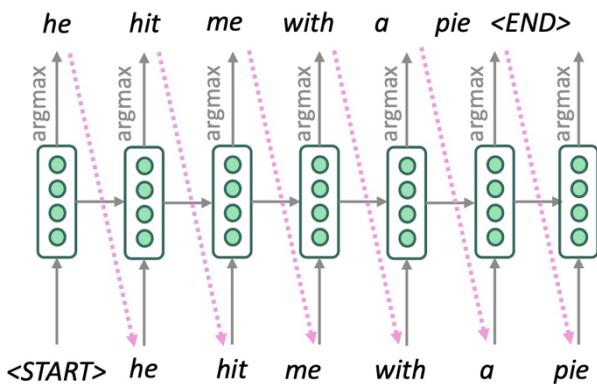
$$\text{כדי לקבל התפלגות } \mathcal{P} \text{ מעל הציוניים הללו ניעזר ב-softmax כפי שביר ראיינו: } P[y_t] = \frac{e^{s_w}}{\sum_{w' \in \mathcal{V}} e^{s_{w'}}}.$$

אלגוריתם **the-g-greedy** שלנו יגידו פונקציה g שתבחר token מעל התפלגות זה, נציגו: $(y_t) = g(P[y_t | \{y_i\}_{i=1}^{t-1}])$. זה החלק שבו אנחנו מפסיקים ליצור פלט שהוא רציף, עליו לבחור אחד בודד token אחד שהוא הפלט הבא.

Exposure bias: בנויגוד לאימון שבו אנחנו נתונים למודול GT-teacher-forcing, כאן בזמן הריצה המודול חוזה את token הבא ואז הוא משתמש בו כדי לחזות את זה לאחריו וכו', אין כאן GT. במצב זה, המודול יכול לחזות token שגוי ובכך לסלול לעצמו דרך שגואה. אנחנו קוראים לה **exposure bias**.

על מנת לפתור את הבעיה, ניתן להסתבל על ה-loss בצורה גלובלית ולא מקומי – לספק למודל ליצור את כל הפלט, ואז ללמד את המודל כמה טוב היה הפלט הכלול שהוא יצר, בלי לחשב באופן מקומי את ה-loss אחריו לtoken. נניח שיש לנו פונקציה פלאים שעוזרת לנו לחשב עד כמה הפלט הכלול של המודול הוא טוב (נתונת ציון בין 0 ל-1). האם נוכל להשתמש בפונקציה זו כדי לאמן את המודול ולשפר את הביצועים שלו? יש כאן בעיה אם הפונקציה אינה דיפרנציאבילית, דבר על בר עוד בהמשך בהקשר של Reinforcement Learning.

בום אנחנו פחות רואים את הבעיה הזאת. כיון שהמודול מאומן על במודות שימושית יותר של>Data, הוא נחשף לאחוז רחיב יותר מההתפלגות של המילים האפשרות.



שיטת Greedy decoding

הדרך הנאייה לבחור את token הבא היא בצורה חמדנית, נסתכל על ההתפלגות וניקח את המילה עם הרסຕירות הגבואה ביותר .(argmax).

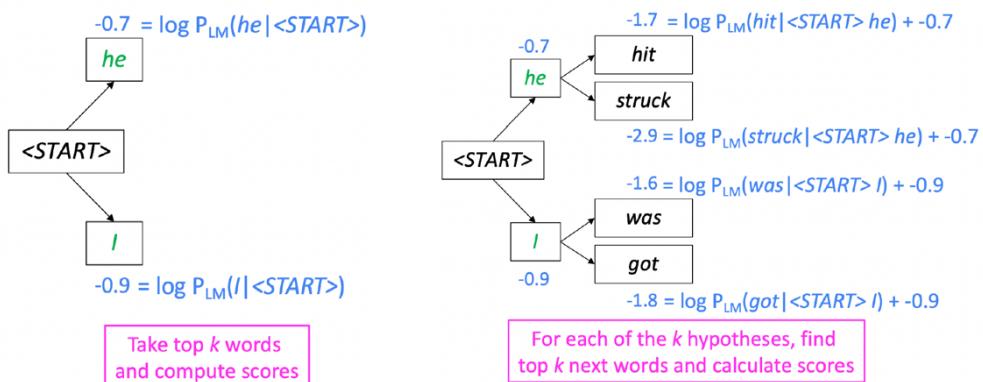
שיטת זו היא דטרמיניסטיבית – יש לביר יתרכנות וחסרונות. מצד אחד אנחנו יודעים לבדוק מה נקבל, אם המודל צודק פעם אחת הוא תמייד יהיה צודק גם בהמשך. מצד שני, עברו סיורים יצירתיים לא נרצה לקבל את אותו הפלט כל פעם.

Beam Search: הבעה הגדולה ביותר בשיטה זו היא בהקשר ל-exposure bias, כשהמודול עושה טעות, היא הופכת לטעות נגררת. נרצה לתקן זאת באמצעות **beam search**. אנחנו רוצים למצוא רצף עם maximum likelihood, אבל אי אפשר לבצע חיפוש נאיבי (שייה בעלות של $\approx 2^{n-1}$ כאשר n הוא אורך הרצף שלנו).

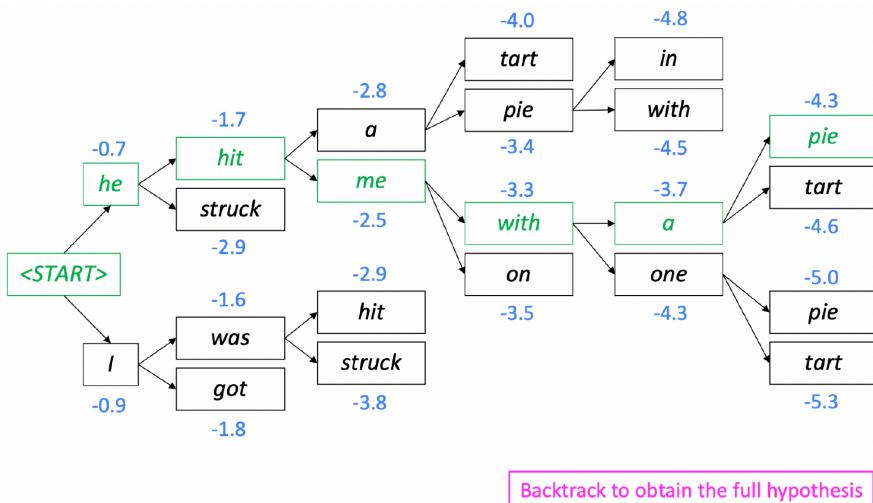
הרעון המרכזי הוא במקומ להסתכל על כל הרצפים האפשריים, להסתכל על קבוצה של k היפותזות להמשיך את הרצף בעלות ההסתברות הגבואה ביותר, באשר **דרג אותן לפי log probability**. כל התוצאות הן שליליות ($\log(x) < 0 \iff x \in (0,1)$), ותצא גבואה יותר היא טובת יותר.

זה לא מבטיח למצוא את הפתרון האופטימלי, אבל הרבה יותר יעיל מחיפוש exhaustive.

לדוגמא, נגיד את גודל-beam שלו 2 . נסתכל על 2 המילים על לוג ההסתברות הגבואה ביותר אחרי ה-token הראשון. לאחר מכן נעשה אותו דבר עבור כל אחת מהמילים שקיבלנו, ונסיף את ההסתברות שלה להסתברות של המילה אחרת. אנחנו מבצעים סכום כי likelihood הוא מכפלת ההסתברויות, ו-log של המכפלה הוא סכום -log-log-ים:



עבשו קיבלו בשכבה השנייה $4 = k^2$ אפשרויות, ונשמרו רק את $2 = k$ בעלות הצוין הגבוה ביותר. כך נמשיך עד שנקבל את המילה الأخيرة בעלת הצוין הגבוה ביותר, ונקבל את המסלול השם שהוביל אליה:



שיטת זו טוביה לכל הფחות כמו ביצוע אלגוריתם greedy פשוט שבו $1 = k$, כאן אפשר לחקור פוטנציאלית יותר אפשרויות. מילה שביחס נאיבי נראהות פחות טוביה (בעל צוין נמוך יותר), עדין יכולה להוביל למסלול טוב יותר שבו הצוין הכללי שיתקבל יהיה נמוך.

קריטריון עצירה:

- המודל מייצר את ה-token המယוחד <END>.

שיטת Sampling decoding

ו

Evaluation

ו

Ethics and Pitfalls

ו



Information Retrieval

ToDo

א

Alignment

ToDo

א