Stanford CS224N – NLP with Deep Learning

Nitzan Barzilay

based on lectures from winter 2021

תוכן עניינים

קישורים להרצאות 5
Intro & Word Vectors 6
ייצוג משמעות של מילים 6
מבוא ל-Word2Vec 6
Neural Classifiers 8
מבוא לאופטימיזציה 8
Negative Sampling Skip-Gram 8
מטריצת הופעות משותפות Co-occurrence Matrix 9
מודל GloVe ל-Word Vectors 10
אוולוואציה של Word Vectors 11
Word Senses 11
Backpropagation 12
בעיית NER – Named Entity Recognition 12
דוגמא לחישוב Backpropegation 12
אלגוריתם ה-Backpropagation 14
דוגמא לחישוב האלגוריתם לכל אורכו 15
סכמה כללית של אלגוריתם ה-backpropagation 16
Syntactic Structure and Dependency Parsing 17
Syntactic Structure וסוגי עמימות 17
עמימות Ambiguity 17
Dependency Grammar and Treebanks 17
מקורות המידע שמשמשים לניתוח תלויות Dependency parsing 18
פרוג'קטיביות 18
Transition Based Dependency Parsing 19
שיטות לבנית עצי תלויות 19
אלגוריתם Transition Based Dependency Parsing 19
דוגמא לריצת אלגוריתם Transition Based Dependency Parsing 19
MaltParser 20
אוולוואציה של עצי תלויות 20
Neural Dependency Parsing 21
קלט לפרסר תלויות מבוסס רשת נוירונים 21
ארכיטקטורה בסיסית של פרסר תלויות מבוסס רשת נוירונים 21
Graph Based Dependency Parsers 22
רשתות נוירונים, RNN 23
רשתות נוירונים 23
היתרון של רשתות נוירונים על פני למידת מכונה קלאסית 23
רשת נוירונים פשוטה מסוג feed forward 23

רגולריזציה 23
Dropout 24
שכבות לא לינאריות ברשתות נוירונים 24
אתחול פרמטרים 24
אימון רשתות נוירונים באמצעות Optimizers 25
בחירת learning rate 25
מידול שפה 25
מודלי n-gram 25
מודלי שפה מבוססי רשתות נוירונים 26
RNN 27
אימון רשתות RNN 28
Backpropagation ברשתות RNN 28
מודלי שפה מבוססי RNN 29
אוולואציה של מודלי שפה 29
משימות נוספות שמשתמשים בהן ב-RNN 30
Exploding & Vanishing grdients 30
Vanishing Gradients 30
Exploding gradients 32
פתרונות לגרדיאנטים מתפוצצים ונעלמים בארכיטקטורות שונות 32
LSTM 32
מבנה רשתות LSTM 32
Bi-directional and multi-layer RNNs 34
RNN דו כיווני 34
תרגום מכונה 35
תרגום לפני עידן רשתות הנוירונים 35
תרגום מכונה באמצעות רשתות נוירונים 36
אימון מודלי NMT 37
יתרונות והסרונות של NMT 37
אוולואציה של מודלי תרגום מכונה 37
מודלי Multi-layer RNN 37
Greedy decoding and Beam Search 38
Attention 39
Attention – הגדרה פורמלית 40
וריאציות נוספות של attention 40
Attention כשיטה כללית בלמידה עמוקה 41
Self Attention 41
חסרונות ובעיות של RNN ושיטות חלופיות 41
Self Attention 42

42 ..... Position representasion
43 ..... Masking the future
43 ..... Transformers
43 ..... ארטיכקטורה של טרנספורמרים
44 ..... Key-query-value attention
44 ..... Multi-headed attention
44 ..... Residual connections
44 ..... Layer normalization
45 ..... Scaled dot product
45 ..... Cross-attention
45 ..... Subword modeling
46 ..... Model pretraining
46 ..... היסטוריה של pretaining
46 ..... pretraining
47 ..... סוגי Pretraining
48 ..... Pretraining decoders
48 ..... מודלי GPT
49 ..... Pretraining encoders
49 ..... מודל BERT – Bidirectional Encoder representation from Transformers
50 ..... הגבלות של מודלי pretrained encoders
50 ..... Pretraining encoder-decoders
51 ..... מודלי שפה ענקיים ו-In-context learning
51 ..... Question Answering
52 ..... SQuAD dataset
52 ..... Reading comprehension
52 ..... מודלי LSTM ל-reading comprehension
55 ..... שימוש ב-BERT ל-reading comprehension
56 ..... השוואה בין BiDAF למודל מבוסס BERT
56 ..... בחירת pre-training objectives טובים יותר לבעית reading comprehension
56 ..... חלקים בבעיית ה-reading comprehension שעדיין לא פתורים
57 ..... open-domain question answering
58 ..... Natural Language Generation
59 ..... שלב ה-decoding
60 ..... Top-k sampling
61 ..... Top-p sampling
61 ..... שיפור ה-decoding באמצעות re-balancing של ההתפלגות
61 ..... retrieval from n-gram phrase statistics
62 ..... Backpropagation-based distribution re-balancing

שיפור ה-decoding באמצעות re-ranking 62
מניעת רפטטיביות באמצעות Unlikelihood training 63
התמודדות עם exposure bias 63
Reinforcement learning באימון מודלי יצירת טקסט 63
אוולויואציה למודלי יצירת שפה 64
Content overlap metrics 64
Model based metrics 65
אוולויואציה אנושית 66
שיקולים אתיים בשימוש במודלי שפה 66
Coreference Resolution 67
שלבי פתרון coreference resolution 67
זיהוי אזכורים 67
coreference vs anaphora 68
Cataphora vs anaphora 68
סוגים של מודלי coreference resolution 68
מודלים Rule based ל-coreference resolution 69
מודלי Mention Pair ל-coreference resolution 70
מודל End-to-end Coreference 71
מודלי שפה גדולים 72
גישת text-to-text 72
דאטאסט Common Crawl 73
מודל T5 73
ארכיטקטורת מודל T5 73
T5 pretraining 74
ניסויים שבוצעו עם T5 74
MT5 - מודל multi-lingual 75
איזה ידע המודל רוכש במהלך ה-pretraining? 75
שיפור ארכיטקטורת הטרנספורמר 77
הוספת ידע למודלי שפה 77
מה מודלי שפה יודעים? 77
שיטות הוספת pretrained entity embeddings 78
ERNIE 78
KnowBERT 79
שיטות המשתמשות בזכרון חיצוני 80
KGLM 80
kNN-LM 81
שיטות המבצעות מודיפיקציה לדאטא האימון 81
WKLM 81

למידת inductive biases באמצעות masking 82
אוולויואציה של ידע במודלי שפה 82
שיקולים חברתיים ואתיים בשימוש ב-AI 83
דוגמאות לשימושים לא אתיים בדאטא 83
שאלות לבחינת מידת האתיות של מחקר בתחום ה-AI 84
נושאים במפגש בין אתיקה ו-NLP 84
הטיה אלגוריתמית במודלי NLP 85
ניתוח והסברת מודלים 86
אוולויואציה ואנליזה של מודלים 86
איך בני אדם מתנהגים בתור נבדקים של משימות שפתיות? 86
Unit testing ובחינת הידע של מודלים 87
השפעת הקלט וזכרון לטווח ארוך 87
Saliency maps 87
רדוקציה של הקלט 88
ניתוח מודלים באמצעות "שבירה" שלהם 88
האם מודלים רובוסטיים לרעש בקלט שלהם? 89
ניתוח הייצוגים (representations) של רשתות נוירונים 89
שימוש ב-probing לניתוח באופן supervised של רשתות נוירונים 90
ביצוע recasting למודלים ומחקרי ablation 90

קישורים להרצאות

לנוחיותכם, אני מצרפת קישורים להרצאות שמפורסמות באופן פומבי ב-YouTube. מדובר בגרסת הקורס של 2021, שנכון לעכשיו (פברואר 2023) היא העדכנית ביותר שפורסמה באופן פומבי.

תיקונים, הערות ופרגונים אפשר לשלוח ל- Nitzan.barzilay@mail.huji.ac.il. אפשר גם לקנות לי קפה 😊

מס' הרצאה	נושא	לינק
1	Intro & Word Vectors	לינק להרצאה 1
2	Neural Classifiers	לינק להרצאה 2
3	Backprop and Neural Networks	לינק להרצאה 3
4	Syntactic Structure and Dependency Parsing	לינק להרצאה 4
5	Recurrent Neural networks (RNNs)	לינק להרצאה 5
6	Simple and LSTM RNNs	לינק להרצאה 6
7	Translation, Seq2Seq, Attention	לינק להרצאה 7
8	Practical tips for NLP projects	לינק להרצאה 8
9	Self- Attention and Transformers	לינק להרצאה 9
10	Transformers and Pretraining	לינק להרצאה 10
11	Question Answering	לינק להרצאה 11
12	Natural Language Generation	לינק להרצאה 12
13	Coreference Resolution	לינק להרצאה 13
14	T5 and Large Language Models	לינק להרצאה 14
15	Add Knowledge to Language Models	לינק להרצאה 15
16	Social & Ethical Considerations	לינק להרצאה 16
17	Model Analysis and Explanation	לינק להרצאה 17

INTRO & WORD VECTORS

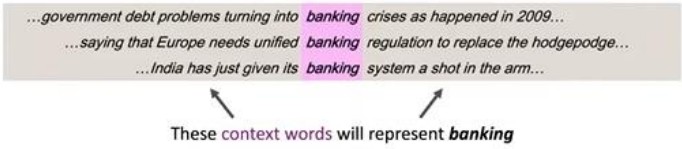
ייצוג משמעות של מילים

ב-NLP מסורתי, מתייחסים למילים בתור סימבולים דיסקרטיים (לדוגמא "מלון", "אכסניה"), וזוהי למעשה התייחסות **לוקליסטית localist** לבעיית ייצוג המשמעות semantic representation של מילים. הדרך הנפוצה לייצג מילים / סימבולים היא באמצעות **One-hot Vector**, שהם וקטורים שבהם כל הכניסות הם 0 למעט כניסה אחת (המייצגת את המילה הספציפית):

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

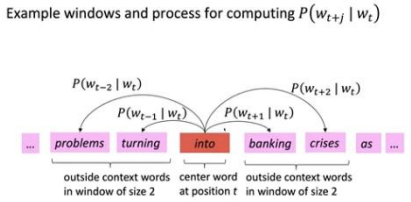
הבעיה עם ייצוג שכזה היא שהוקטורים חייבים להיות ממימד עצום ששקול לכמות המילים היחודיות בקורפוס (מאות אלפים). בנוסף, בהגדרה כל שתי מילים יהיו אורתוגונליות זו לזו, ולכן אי אפשר להפיק מהוקטורים האלו מידע על כמה שתי מילים "קרובות" במשמעות שלהן. בעיה נוספת עם ייצוג לוקליסטי של מילים היא שהוא לא משתמש בהקשר שהמילה מופיעה בו, שיכול לתת הרבה מידע חשוב על משמעות המילה.

דרך אחרת לגשת לבעיה היא **Distributional Semantics** – שיטה בה מתייחסים למשמעות של המילה בתור התדירות שבה היא מופיעה בהקשר מסוים (כלומר ביחד עם מילים אחרות). כלומר, כשמילה מופיעה בתוך טקסט, ההקשר שלה הוא קבוצת המילים שמופיעות בסביבתה (בתוך חלון בגודל של מספר מילים כלשהו שנקבע מראש).



זה היה רעיון שהראה תוצאות טובות יחסית ומשתמשים בו עד היום במגוון מודלים סטטיסטיים. דרך נפוצה להשתמש בייצוג ההתפלגותי של מילים היא ליצור **Word Vectors / Embeddings** – לכל מילה נייצר וקטור דחוס (כמה מאות כניסות), שייצג את המשמעות של המילה בכך שהוא יבנה כך שהוא יהיה קרוב לוקטורים של מילים אחרות שמופיעות בהקשרים דומים. הסיבה שהוקטורים האלו מכונים Embeddings היא שכאשר יוצרים לכל מילה וקטור שכזה, אפשר לחשוב על השפה כולה בתור מרחב וקטורי רב מימדי, כאשר הוקטור של מילה ספציפית "משכן" אותה למקום מסוים במרחב הזה. הרעיון הוא שמילים שקרובות זו לזו במשמעות, יהיו קרובות זו לזו במרחב.

מבוא ל-WORD2VEC



Word2Vec היא framework שהוצגה ב-2013 ליצירת Word Vectors באמצעות למידה. הרעיון הוא לקחת קורפוס מאוד גדול, וליצור וקטור לכל מילה באופן הבא – עוברים על כל הטקסט בקורפוס, ולכל מילה c (center) מסתכלים על מילות ההקשר o (outside) שנמצאות בחלון ההקשר שלה. משתמשים בדמיון בין וקטורים של מילות c ובין וקטורים של מילות o כדי לחשב את ההסתברות של מילה o בהנתן מילה c (או להפך). ממשיכים להתאים את הוקטורים כדי למקסם את ההסתברות הזו.

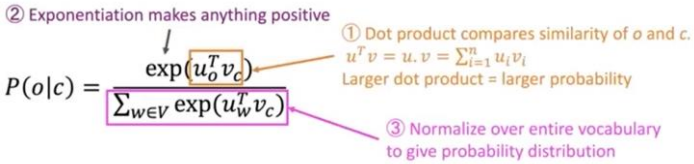
לכל מיקום בטקסט $t = 1, \dots, T$ (כאשר T הוא מספר המילים בקורפוס), נחזה את מילות ההקשר בתוך חלון בגודל קבוע m, בהנתן מילת מרכז (center) המסומנת w_j ע"י לסכום את ההסתברות L שמילה מסוימת מופיעה בהנתן מילה אחרת בתוך חלון המילים שסביבה, עבור כל המילים בקורפוס וכל המילים בחלון של כל אחת מהן, באופן הבא (כאשר θ הם כל המשתנים שרוצים לאפטם):

$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{m \le j \le m \\ j \ne 0}} P(w_{t+j} \mid w_t; \theta)$$

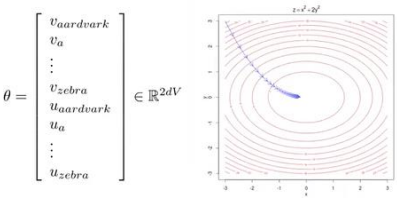
ה-**Objective Function** (שלעתים נקראת **פונקציית Cost** / **פונקציית Loss**) המסומנת $J(\theta)$ היא ה- Average negative log likelihood (המעבר לשימוש ב-log הוא דרך סטמדרטית לעבור ממכפלה לסכום תוך שמירה על יחס הסדר). למזער את ה- objective function שקול ללמקסם את הדיוק accuracy של חיזוי מילות ההקשר. הפונקציה בנויה באופן הבא:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{m \le j \le m \\ j \ne 0}} \log P(w_{t+j} \mid w_t; \theta)$$

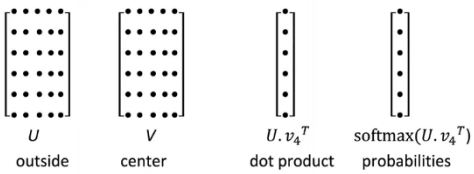
איך מחשבים את $P(w_{t+j} | w_t, \theta)$ – משתמשים בשני וקטורים לכל מילה w – הוקטור v_w (כאשר w היא מילת ה-center) והוקטור u_w (כאשר w היא מילת הקשר), ואז לכל מילת מרכז c ולכל מילת הקשר o נחשב את ההסתברות הבאה:



הנוסחא הזו היא דוגמא לשימוש בפונקציית softmax (כלומר פונקציה $\mathbb{R}^n \to (0,1)^n$ שהופכת וקטור של מספרים ממשיים שרירותיים x_i לוקטור התפלגות) - $\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$. השם max מייצג את העובדה שזה יתן הסתברות גדולה ל-x_i-ים הגדולים ביותר, והשם soft מייצג את זה שגם x_i-ים קטנים מקבלים הסתברות כלשהי. פונקציות *softmax* נמצאות בשימוש נפוץ בלמידה עמוקה.



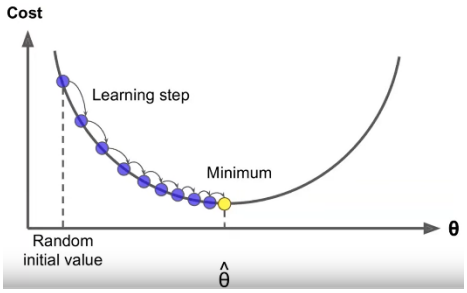
אימון המודל – כדי לאמן את המודל, אנחנו מתאימים בהדרגה את הפרמטרים כדי למזער את ה-loss. נזכר בכך ש-θ היא וקטור יחיד שמייצג את <u>כל</u> הפרמטרים של המודל. בציור משמאל ניתן לראות הדגמה עבור וקטורים ממימד d בקורפוס עם V מילים (נזכר בכך שלכל מילה יש 2 וקטורים, ולכן θ היא ממימד 2dV). האופטימיזציה לפרמטרים מתבצעת באמצעות gradient descent.



פרמטרים וחישוב – הפרמטרים היחידים של המודל הם שני וקטורי המילים u,v. החישוב נראה כך – קודם עושים מכפלה פנימית ביניהם, ואז מפעילים עליה softmax כדי להפוך את התוצאה לוקטור התפלגות. המודל יבצע את אותן הפרדיקציות עבור כל מיקום (כלומר, זה למעשה מודל **bag of words**, כלומר מודל שלא מתייחס לסדר של המילים אלא רק לעובדה שהן מופיעות יחד באותו חלון). אחנו רוצים מודל שיתן לנו הסתברות גבוהה יחסית ל<u>כל</u> המילים שמופיעות באותו הקשר (של מילת המרכז).

NEURAL CLASSIFIERS

מבוא לאופטימיזציה



Gradient Descent – דרך טובה ללמוד את ה-Word Vectors (כלומר למזער את ה-cost, כלומר את הפונקציה $J(\theta)$) היא שימוש ב-Gradient Descent (אלגוריתם למזעור פונקציה $J(\theta)$ ע"י שינוי של θ. הרעיון הוא להתחיל מערך אקראי כלשהו של θ, לחשב את הגרדיאנט של $J(\theta)$, ואז לעשות צעד קטן (גודל הצעד הוא היפרפרמטר בשם learning rate) בכיוון הנגדי לגרדיאנט (כלומר, לכיוון המינימום). נחזור על התהליך הזה באופן איטרטיבי עד שנגיע לנקודה שאנחנו תופסים כנקודת מינימום (אם הפונקציה J היא קמורה, המינימום יהיה גלובלי, כלומר יחיד. אחרת, יש חשש ש"נתקע" בנקודת מינימום לוקלית שאינה גלובלית).

נוסחאות לביצוע Gradient Descent:

- משוואת העדכון (בנוטצית מטריצה) - $\theta^{new} = \theta^{old} - \alpha\nabla_\theta J(\theta)$ כאשר α הוא גודל הצעד או ה-learning rate.
- משוואת העדכון (עבור פרמטר בודד) - $\theta_j^{\text{new}} = \theta_j^{\text{old}} - \alpha \frac{\partial}{\partial\theta_j^{\text{old}}} J(\theta)$
- אלגוריתם GD:

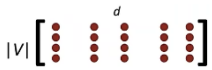
```
while True:
    theta_grad = evaluate_gradient(J,corpus,theta)
    theta = theta - alpha * theta_grad
```

האלגוריתם הזה לא נמצא בשימוש – הבעיה היא ש-J היא פונקציה של <u>כל</u> החלונות בקורפוס (לעתים מדובר במיליארדים של חלונות), לכן הגרדיאנט של הפונקציה כולה (כלומר $\nabla_\theta J(\theta)$) הוא מאוד כבד לחישוב, ולכן יצריך הרבה מאוד זמן חישוב עבור כל איטרציה של האלגוריתם, ויהפוך את האלגוריתם עצמו לאיטי מאוד, באופן שהופך אותו ללא ישים עבור רשתות נוירונים.

הפתרון לבעיה הזו היא להשתמש במקום זה ב-**SGD – Stochastic Gradient Descent.** באלגוריתם SGD הוא מודיפיקציה קטנה ל-GD – במקום לנסות לעשות את האופטימיזציה על כל הקורפוס בבת אחת, מבצעים באופן איטרטיבי את האופטימיזציה (כלומר, מבצעים הערכה לגרדיאנט של הפונקציה כולה) על בסיס דגימה קטנה batch של חלונות מתוך הקורפוס. למעשה, הוא לא רק יותר מהיר - עבור רשתות נוירונים גדולות הוא גם נותן בסוף פתרונות טובים יותר מאשר הפתרונות ש-GD רגיל מוצא.

$$\nabla_\theta J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \nabla_{v_{like}} \\ \vdots \\ 0 \\ \nabla_{u_I} \\ \vdots \\ \nabla_{u_{learning}} \\ \vdots \end{bmatrix} \in \mathbb{R}^{2dV}$$

באלגוריתם ה-SGD, מחשבים באופן איטרטיבי גרדיאנטים של כל חלון בקורפוס בנפרד. אבל בכל חלון יש לנו לכל היותר 2m+1 מילים, לכן וקטור הגרדיאנט שנקבל הוא מאוד דליל sparse. לכן יתכן שנרצה לעדכן רק את וקטורי המילים ששייכים למילים שבאמת מופיעות בפועל בחלון. הפתרון הוא – או שמשתמשים באופרציות לעדכון מטריצה sparse-ית ככה שהם יעדכנו רק את השורות שמלאות ב-embeddings של המטריצות U,V, או ששומרים טבלת hash עבור כל וקטורי המילים. אם יש לנו מיליוני וקטורי מילים ואנחנו משתמשים בחישוב מבוזר distributed computing, זה חשוב לא לשלוח עדכוני מטריצה גדולים מידי.



- אלגוריתם SGD:

```
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J,window,theta)
    theta = theta - alpha * theta_grad
```

SGD הוא אלגוריתם האופטימיזציה שנמצא בשימוש נפוץ ברשתות נוירונים.

NEGATIVE SAMPLING SKIP-GRAM

וריאנטים נוספים מתוך משפחת האלגוריתמים של Word2Vec:

- אפשר לממש את אותו האלגוריתם רק עם וקטור אחד לכל מילה במקום עם שניים
- אפשר להשתמש ב-skip-grams – לחזות את מילות ההקשר (במיקום בלתי תלוי) בהנתן מילת מרכז

- אפשר להשתמש ב-CBOW – Continuous bag of words – לחזות את מילת המרכז מתוך "שק" של מילות הקשר
- כדי לשפר את היעילות באימון, אפשר להשתמש ב-Negative Sampling.

מודל סקיפ-גרם עם Negative Sampling: ביטוי הנרמול באלגוריתם המקורי (המכנה של הביטוי $P(o \mid c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$) הוא יקר מאוד חישובית, ונרצה למצוא דרך לעשות משהו שדומה למה שה-softmax עושה רק בדרך יעילה יותר חישובית. הרעיון הכללי של מודל סקיפ-גרם עם דגימה שלילית היא לאמן מודלי רגרסיה לוגיסטית בינאריים עבור זוג "אמת" (מילת מרכז ומילת הקשר מתוך החלון שלה) מול מספר זוגות "רעש" (אותה מילת מרכז ומילה רנדומלית כלשהי).

הרעיון הוא שאנחנו עדיין רוצים למזער את ה-loss, כלומר את הפונקציה $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$ (ה-loss הממוצע לכל מילת מרכז), המתבססת על הפונקצית הבאה של ה-loss של מילת מרכז: $J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{i=1}^{k} \mathbb{E}_{j \sim P(w)}\left[\log \sigma\left(-u_j^T v_c\right)\right]$ (במקום להשתמש ב-log כמו באלגוריתם המקורי, נעביר את התוצאה דרך sigmoid – הפונקציה $\sigma(x) = \frac{1}{1+e^{-x}}$). זה מאפשר לנו למקסם את ההסתברות ששתי מילים יופיעו ביחד (בביטוי הלוג הראשון שמופיע בנוסחא - $\log \sigma(u_o^T v_c)$) ולמזער את ההסתברות של מילות רעש (בביטוי התוחלת בחלק השני של הנוסחא - $\sum_{i=1}^{k} \mathbb{E}_{j \sim P(w)}\left[\log \sigma\left(-u_j^T v_c\right)\right]$). כלומר:

- לוקחים k דגימות שליליות / רועשות תוך שימוש בהסתברות של המילים – במאמר המקורי למשל דוגמים מילים בהסתברות $P(w) = \frac{U(w)^{\frac{3}{4}}}{Z}$ – ההסתברות האחידה U(w) מועלה בחזקת $\frac{3}{4}$ חלקי פרמטר הנרמול Z הזהה לנוסחא של ה-softmax. החזקה גורמת לכך שמילים פחות נפוצות ידגמו באופן יותר תכוף.
- ממקסמים את ההסתברות להופעת מילת ההקשר האמיתית ביחד עם מילת המרכז, וממזערים את ההסתברות של מילים אקראיות להופיע ביחד עם מילת המרכז.

דרך אחרת לכתוב את אותה הנוסחא היא - $J_{\text{neg-sample}}(\boldsymbol{u}_o, \boldsymbol{v}_c, U) = -\log \sigma(\boldsymbol{u}_o^T \boldsymbol{v}_c) - \sum_{k \in \{K \text{ sampled indices}\}} \log \sigma(-\boldsymbol{u}_k^T \boldsymbol{v}_c)$.

משהו נוסף שחשוב לציין הוא שלא תמיד משתמשים בחלון לפי מספר מילים מוגדר מראש – ישנם מקרים בהם המסמך שמכיל את הטקסט מכיל מבנה כלשהו, ואז אפשר להשתמש במבנה הזה בתור חלון – לדוגמא, להגדיר את החלון בתור כל הפסקה שמילת המרכז נמצאת בה.

מטריצת הופעות משותפות CO-OCCURRENCE MATRIX

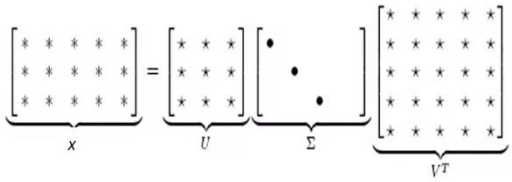
Example corpus:
- I like deep learning
- I like NLP
- I enjoy flying

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	

Stanford

דרך אפשרית לייצג את ההסתברות היא לייצר מטריצת **Co-Occurrence Matrix** שמציגה באופן ישיר ומפורש את כמות הפעמים שזוג מסוים של מילת מרכז ומילת הקשר מופיע ביחד. בדוגמה מימין ניתן לראות מטריצה שכזו עבור גודל חלון של מילה אחת (בעולם האמיתי יותר נפוץ לבחור בגודל בין 5-10) על קורפוס שמכיל שלושה משפטים. כל שורה / עמודה (זה שקול כי המטריצה סימטרית) הם למעשה וקטור של מילה ספציפית המכונה **Co-occurrence Vector**.

וקטורי Co-occurrence הם מאוד גדולים (המימד שלהם הוא כמות המילים היחודיות בקורפוס), ולכן מצריכים הרבה מאוד זכרון כדי לאחסן אותם. בנוסף, הם מאוד דלילים Sparse, מה שמקשה על חלק מהסוגים של מודלי קלסיפיקציה שלא מתמודדים טוב עם קלט דליל ויכול להוביל לתוצאה פחות טובה ולמודלים פחות רובוסטיים. שתי הסיבות האלו מובילות אותנו להשתמש בוקטורים ממימד נמוך יותר. הרעיון הוא לקחת רק את המידע החשוב בוקטור (שאינו אפסים) ולדחוס אותו לוקטור ממימד נמוך (קבוע, שנקבע מראש עבור כל המטריצה – בד"כ בסביבות 25-1000 כניסות). לכן, צריך לבצע הקטנת מימדים של הוקטור המקורי כדי להתאים אותו למימד הקטן הנדרש של הוקטור הדחוס.

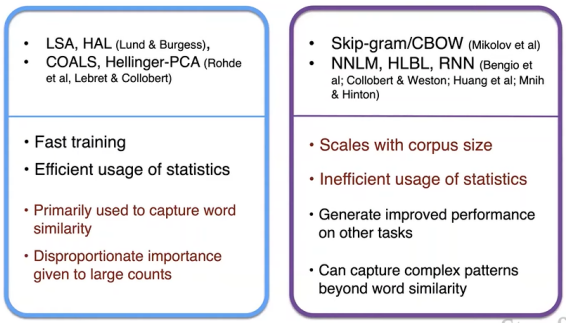


השיטה הקלאסית להקטנת מימד – עבור מטריצה X ניתן להשתמש **בפירוק SVD**, ולבחור את $\hat{X}$ שמכילה את k הערכים הסינגולריים הגדולים ביותר במטרה להכליל. הבעיה עם השיטה הזו היא שהיא מאוד יקרה לחישוב במטריצות גדולות. הבעיה היא שלהריץ SVD על Raw Counts לא עובד מאוד טוב. לעשות scaling ל-counts בתאים יכול לעזור מאוד עם זה. הבעיה היא שמילים פונקציונליות ("ה", "הוא", "ש") הן יותר מידי נפוצות, ולכן יש לתחביר השפעה הרבה יותר גדולה משהיינו רוצים. כמה דרכים אפשריות לתקן את זה:

- לעשות לוג ל-count.
- לעשות מינימום בין ה-count לבין ערך t כלשהו (מספר אצבע t=100).
- להתעלם ממילים פונקציונליות לחלוטין.

מודל GLOVE ל-WORD VECTORS

השוואה בין מודלים מבוססי מטריצות ואלגברה לינארית (בכחול משמאל) לבין מודלי רשתות נוירונים לומדות (בסגול מימין):



הרעיון של מודל GloVe היה לשלב בין היתרונות של מודלים לינארים (אימון יעיל, ניצול טוב של הידע הסטטיסטי) ליתרונות של מודלים מבוססי רשתות נוירונים לומדות (קלים יותר לחישוב עבור קורפוסים גדולים, מאפשרים הכללה שאפשר להשתמש בה גם עבור בעיות NLP אחרות, ומאפשרים לתפוס דפוסים מורכבים יותר מרק דמיון בין מילים - למשל אנלוגיות).

מסקנה חשובה היא שהיחס ratio של הסתברויות ההופעה המשותפת co-occurrence probabilities יכול לקודד היבטים מסוימים של המשמעות של המילים. למשל בתמונה משמאל ניתן לראות שההסתברויות לבדן מאפשרות לקשר בין קרח לבין מים ומוצק ובין אדים לבין מים וגז, אבל אם מסתכלים על היחס ביניהם מגלים איזה קשרים (מוצק / גז) הם אלו שמבדילים בין המשמעות של קרח למשמעות של אדים.

	x = solid	x = gas	x = water	x = random
$P(x\|\text{ice})$	large	small	large	small
$P(x\|\text{steam})$	small	large	large	small
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	large	small	~1	~1

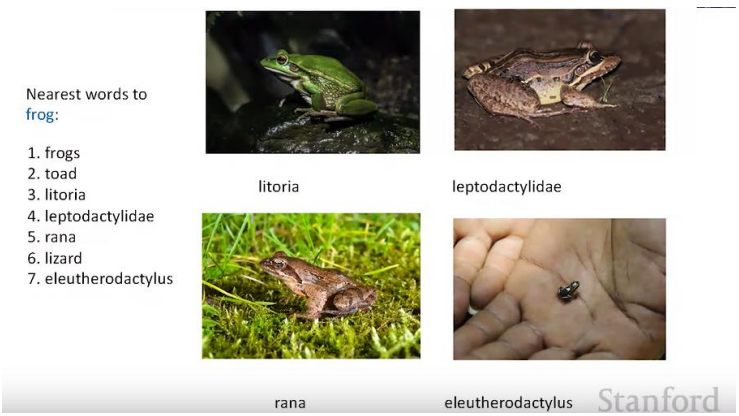
Stanford

איך אפשר לתפוס את היחס בין הסתברויות הופעה משותפת בתור רכיבים לינארים של משמעות של מילים במרחב וקטורי?

- מודל לוג-בילינארי: $w_i \cdot w_j = \log P(i \mid j)$
- עם הפרשים בין וקטורים: $w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$

מודל GloVe מנסה לקחת משני הרעיונות האלו – זה למעשה מודל שדומה לרשת נוירונים שמחושב על בסיס מטריצת ספירת הופעות משותפות. פונקצית ה-loss המפורשת היא $J = \sum_{i,j=1}^{V} f(X_{ij})\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$ (שמבטאת את זה שאנחנו רוצים שהמכפלה הפנימית $w_i^T \tilde{w}_j$ תהיה דומה ללוג של ההופעה המשותפת $\log X_{ij}$. הביטויים b הם ביטויי bias, והפונקציה f מטרתה למנוע ממילים נפוצות מאוד להיות בעלות שליטה חזקה מידי ע"י זה שהיא מגבילה מילים נפוצות).

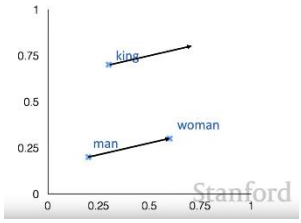
זהו מודל מהיר לאימון, מאוד scalable ולכן מתאים לשימוש בקורפוסים גדולים, ומשיג ביצועים טובים גם עם קורפוסים קטנים ווקטורים קטנים. דוגמה לתוצאות של המודל – מה GloVe זיהה בתור המילים שהכי קרובות למילה frog:



אוולויואציה של WORD VECTORS

הצורך לעשות אוולויואציה לוקטורי מילים מתקשר לשתי שיטות כלליות שמשמשות לאוולויואציה בכל מיני בעיות NLP:

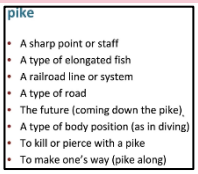
- **אוולויואציה אינטרינזית Intrinsic:** אוולויואציה של תת משימה ספציפית, מהירה לחישוב, עוזרת להבין את המערכת כולה אבל לא ברור אם היא מועילה כל עוד לא מוצאים קורלציה למשימה האמיתית.
  - דוגמא – פתרון בעיות אנלוגיה סמנטית (גבר עבור אישה הוא כמו מלך עבור ...? – מוצאים את החיבור במרחב בין גבר לאישה, הולכים אותו מרחק באותו כיוון ממלך ומקווים למצוא שם מלכה). כלומר, בשיטה הזו מעריכים וקטורים ע"י כמה קטן המרחק בין המיקום האמיתי של המילה שחיפשנו לבין המיקום ששוער לפי האנלוגיה. הבעיה היא שיכול להיות שהמידע כן נמצא שם אבל הוא לא לינארי.



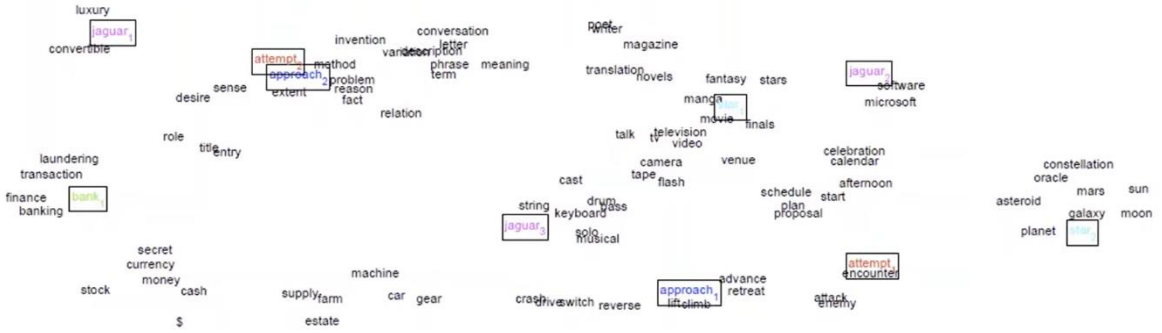
  - דוגמא נוספת – קורלציה של מרחקים בין מילים לפי המודל למול לפי שיפוט אנושי.
- **אוולויואציה אקסטרינזית Extrinsic:** אוולויואציה של המשימה האמיתית באופן ישיר, יכולה לדרוש זמן חישוב ארוך. לא מאפשרת להבין אם הבעיה היא בתת מערכת כלשהי או בחיבור בין תתי מערכות. אפשר לנסות לשפר ע"י להחליף בדיוק תת מערכת אחת ולראות אם זה משפר את הדיוק.
  - דוגמא – אוולויואציה למערכת שמבצעת named entity recognition (זיהוי התייחסויות לאדם, ארגון או מקום בתוך טקסט).

WORD SENSES

בעיה בתחום הסמנטיקה היא שלרוב המילים יש הרבה יותר ממשמעות אחת (במיוחד מילים נפוצות או מילים שקיימות כבר הרבה זמן). איך אנחנו מצפים מוקטור אחד לתפוס במקביל את כל המשמעויות האפשריות של מילה מסוימת?



נסיון אחד בתחום (תחילת שנות האלפיים) היה לנסות ליצור וקטורים שונים לכל מובן של מילה מסוימת ע"י קליסטור:



גישה אחרת מנסה לייצג את כל המשמעויות באמצעות וקטור בודד שהוא סכום ממושקל של כל המשמעויות האפשריות (*מכונה* linear superposition), בהמשך לדוגמה מראש העמוד נחשוב על הוקטור $v_{\text{pike}} = \alpha_1 v_{\text{pike}_1} + \alpha_2 v_{\text{pike}_2} + \alpha_3 v_{\text{pike}_3}$ כאשר $\alpha_1 = \frac{f_1}{f_1+f_2+f_3}$ עבור פונקצית *ההתפלגות* f. *ז*ה נותן תוצאות מפתיעות – העובדה שהוקטורים האלו מאוד דלילים יכולה לעזור לנו להפריד בין המובנים (כל עוד המובנים נפוצים מספיק).

BACKPROPAGATION

בעיית NER – NAMED ENTITY RECOGNITION

המשימה היא לזהות ולסווג שמות בטקסט, לדוגמא:

Last night , Paris Hilton wowed in a sequin gown .
PER PER
Samuel Quinn was arrested in the Hilton Hotel in Paris in April 1989 .
PER PER LOC LOC LOC DATE DATE

שימושים אפשריים לפתרון שכזה יכולים להיות מעקב אחר אזכורים של ישויות מסוימות במסמך, או לשימוש עבור מענה על שאלות (תשובות הן לעתים קרובות ישויות בעלות שם). לעתים קרובות פתרון הבעיה הזו מוביל לבעית ההמשך של Named Entity Linking.

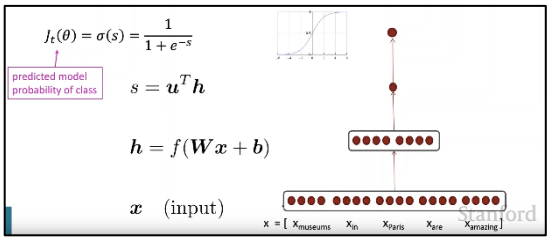
דרך פשוטה לפתור את הבעיה הזו באמצעות רשת נוירונים – סיווג לפי חלונות תוך שימוש בקלסיפיקציה בינארית לוגיסטית. הראיון הוא לסווג כל מילה בתוך חלוק ההקשר שלה (שמורכב ממילות הקשר). לאחר מכן, נאמן קלסיפייר לוגיסטי על דאטא שתויג ידנית כדי לסווג עבור כל מילה באופן בינארי האם היא מילת הקשר, תוך התבססות על ה-word vector של כל אחת ממילות ההקשר (כאשר כל הוקטורים האלו "מודבקים" concatenated לכדי וקטור אחד מאוד ארוך). לדוגמא:

- **Example:** Classify "Paris" as +/– location in context of sentence with window length 2:

the museums in Paris are amazing to see .

$x_{window} = [\ x_{museums}\quad x_{in}\quad x_{Paris}\quad x_{are}\quad x_{amazing}\ ]^T$

- Resulting vector $x_{window} = \boxed{x \in R^{5d}}$, a column vector!



הקלסיפייר שנשתמש בו נראה כך – וקטור הקלט יהיה ממימד מספר המילים בחלון כפול מימד הוקטור של מילה בודדת. וקטור הקלט מוכפל במטריצת המשקולות W ומתווסף לו פרמטר הביאס b, ולאחר מכן הוא מועבר דרך שכבר לא לינארית (למשל softmax). כלומר, יש לנו שכבה חבויה אחת, שמיוצגת ע"י hidden vector ממימד קטן יותר מהמימד המקורי של הקלט (השכבה האמצעית בציור משמאל). הוקטור החבוי מוכפל במכפלה פנימית בוקטור נוסף u כך שמתקבל סקלר בודד. המספר הזה מוכנס לפונקצית טרנספורמציה לוגיסטית כדי "להתאים" אותו לטווח בין 0 ל-1, ובכך לייצג את ההסתברות של מילת הקלט להיות שם של ישות.

דוגמא לחישוב BACKPROPEGATION

כדי לעדכן את המשקולות, נחשב את θ החדשה באופן הבא - $\theta^{new} = \theta^{old} - \alpha\nabla_\theta J(\theta)$ כאשר α הוא קצב הלמידה. כלומר, לכל פרמטר j נחשב $\theta_j^{\text{new}} = \theta_j^{\text{old}} - \alpha\frac{\partial J(\theta)}{\partial\theta_j^{\text{old}}}$. בלמידה עמוקה, אנחנו מעדכנים גם את הייצוג של הדאטא, כלומר את וקטורי המילים. אפשר לחשב את $\nabla_\theta J(\theta)$ ידנית, אבל בעולם האמיתי משתמשים באלגוריתם ה-backpropagation. אם נביט על היעקוביאנים כשגוזרים לפי פרמטרים שונים, נקבל את הזהויות הבאות:

$$\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{W}$$
$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{I} \quad \text{(Identity matrix)}$$
$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T\boldsymbol{h}) = \boldsymbol{h}^T$$

Fine print: This is the correct Jacobian. Later we discuss the "shape convention"; using it the answer would be $\boldsymbol{h}$.

נראה כעת דוגמא שבה נחשב את הגרדיאנט החלקי של התוצאה s ביחס לפרמטרים שונים של המודל (w,b,u,x).

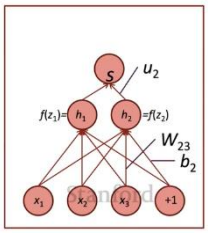
נחשב את $\frac{\partial s}{\partial \boldsymbol{b}}$:

- אנחנו רוצים לחשב את הגרדיאנט של ה-loss המסומן J_t, אבל נחשב את הגרדיאנט של ה-score לטובת הפשטות. אפשר לראות שה-score הוא למעשה הרכבה של שלוש פונקציות על המשתנה x, כלומר $score = s \circ h \circ z(x)$ כאשר הפונקציה הראשונה היא $\boldsymbol{z} = W\boldsymbol{x} + \boldsymbol{b}$, השניה היא $\boldsymbol{h} = f(z)$ והשלישית היא $s = \boldsymbol{u}^T\boldsymbol{h}$.

- לכן כדי לחשב את הגרדיאנט יש לבצע את החישוב הבא: $\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$.
- כעת נשתמש בזהויות היעקוביאן שראינו מעלה. נקבל כי מתקים:
  - $\frac{\partial s}{\partial \boldsymbol{h}} = u^T$
  - $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = diag(f'(z))$
  - $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} = I$
- ולכן בסה"כ נקבל מכלל השרשרת שמתקיים $\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} = u^T diag\big(f'(z)\big)I = u^T \circ f'(z)$ כאשר הסימן $\circ$ מתייחס למכפלה element-wise.
- צריך לבצע התאמה של המימדים כדי שאפשר יהיה לבצע את המכפלה, כיוון שהתוצאה של $u^T \circ f'(z)$ היא וקטור שורה, בעוד התוצאה של הגרדיאנט של b צריכה להיות וקטור עמודה (כיוון ש-b הוא וקטור עמודה).

כעת נחשב את $\frac{\partial s}{\partial \boldsymbol{W}}$:

- נזכר כי W היא מטריצה בגודל $n \times m$. נרצה תמיד להציג את הגרדיאנט באופן הבא: $\frac{\partial s}{\partial \boldsymbol{W}}$ is n by m: $\begin{bmatrix} \frac{\partial s}{\partial W_{11}} & \cdots & \frac{\partial s}{\partial W_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial s}{\partial W_{n1}} & \cdots & \frac{\partial s}{\partial W_{nm}} \end{bmatrix}$, כלומר נשתמש במה שנקרא shape convention ונגרום לכך שהמימדים של הגרדיאנט יהיו זהים למימדים של הפרמטרים. כלומר, "נעצב מחדש" את היעקוביאן כדי שיתאים למימדים הרצויים ($n \times m$).
- מכלל השרשרת, נקבל באותו אופן כי $\frac{\partial s}{\partial \boldsymbol{W}} = \frac{\partial s}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$. שני הביטויים הראשונים במכפלה זהים לביטויים שכבר חישבנו במהלך החישוב של $\frac{\partial s}{\partial \boldsymbol{b}}$, לכן לא נצטרך לחשב אותם שוב. נסמן אותם ב-$\delta = \frac{\partial s}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = u^T \circ f'(z)$ (המכונה local error signal), כלומר $\frac{\partial s}{\partial \boldsymbol{W}} = \delta \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$.
- כעת נשאר לנו לחשב רק את $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$. כיוון ש-$\boldsymbol{z} = W\boldsymbol{x} + \boldsymbol{b}$, נקבל שהנגזרת שלו לפי x היא פשוט x. לכן, נקבל כי הגרדיאנט שרצינו לחשב הוא בסה"כ $\frac{\partial s}{\partial \boldsymbol{W}} = \delta \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}} = \delta^T x^T$, כאשר δ הוא ה-local error signal ב-z, ו-x הוא ה- local input signal.
- אם נחשוב על הנגזרת של משקולת בודדת כלשהי $W_{i,j}$, נגלה שהיא משפיעה רק על z_i. למשל, $W_{2,3}$ משומשת לחישוב רק של z_2 ולא של z_1. כלומר, מתקיים $\frac{\partial z_{i'}}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} \boldsymbol{W}_{i\cdot}\boldsymbol{x} + b_i = \frac{\partial}{\partial W_{ij}} \sum_{k=1}^{d} W_{ik}x_k = x_j$.



- הסיבה ל-transpose היא התאמת המימדים:

$$\underset{[n\times m]}{\frac{\partial s}{\partial \boldsymbol{W}}} = \underset{[n\times 1]}{\boldsymbol{\delta}^T}\underset{[1\times m]}{\boldsymbol{x}^T} = \begin{bmatrix}\delta_1 \\ \vdots \\ \delta_n\end{bmatrix}[x_1, \dots, x_m] = \begin{bmatrix}\delta_1 x_1 & \dots & \delta_1 x_m \\ \vdots & \ddots & \vdots \\ \delta_n x_1 & \dots & \delta_n x_m\end{bmatrix}$$

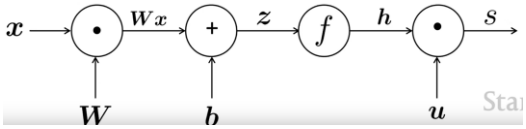
איך נקבע מה צריכים להיות המימדים של הגרדיאנטים?

- דרך אחת היא להשתמש בצורת היעקוביאן איפה שאפשר, ונשנה את המימדים שלה כדי שיהיו תואמים למימדים הרצויים של התוצאה.
- דרך אחרת היא תמיד נעקוב אחר ה-shape convention – נסתכל על המימדים וננסה להבין מתי צריך לעשות transpose ו/או לסדר באופן אחר את הביטויים במכפלה. הפרמטר δ שנמצא ב-hidden layer הוא בעל אותם המימדים כמו ה- hidden vector.

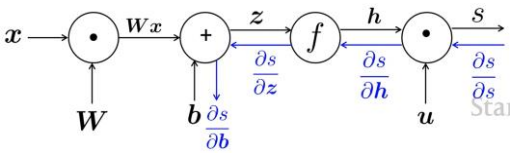
אלגוריתם ה-BACKPROPAGATION

באלגוריתם ה-Backpropagation מחשבים נגזרות ומשתמשים בכלל השרשרת (בגרסתו המוכללת שמתאימה למטריצה או לבעיה עם מספר משתנים). נוכל לחסוך לעצמנו חישובים ע"י להשתמש שוב בנגזרות שכבר חישבנו בשכבות "גבוהות" יותר כדי לחשב את השכבות הנמוכות יותר.

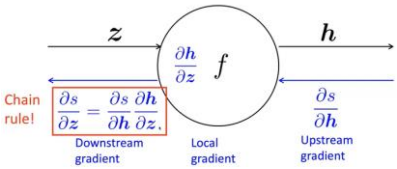
הייצוג התוכנתי של רשתות נוירונים הוא באמצעות גרף חישוב computation graph, שהוא למעשה עץ שהשורש שלו הוא הקלט, ה-nodes הפנימיים הם פעולות operations שאנחנו מבצעים על הדאטא, והקצה שלו הוא הפלט של החישוב. כמו בכל גרף חישוב, תוצאת החישוב בכל node מועברת ל-node הבא דרך הצלע המחברת ביניהם. החישוב הזה (מהקלט x ועד התוצאה s שמתקבלת בקצה הגרף) נקרא **Forward propagation:**



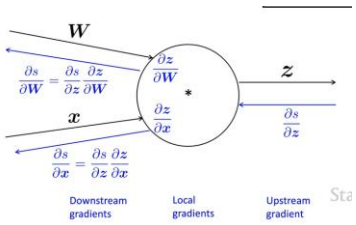
מה שנותן לרשתות נוירונים את הכוח שלהן, הוא שאנחנו לא מסתפקים ב-Forward propagation שנותן לנו את התוצאה s, אלא גם שולחים "אחורה" בגרף החישוב מידע שמאפשר לרשת ללמוד מהחישוב שהתבצע. התהליך הזה, בו אנחנו שולחים אחורה את הגרדיאנטים המקומיים, נקרא **Backpropagation**:



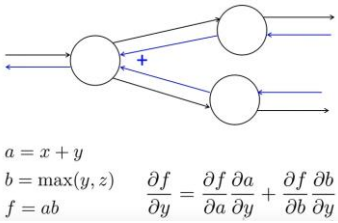
נסתכל על איך נראה התהליך עבור node בודד בגרף. נניח שהיה לנו node פשוט שמייצג את האופרציה h=f(z). יש לנו את הגרדיאנט $\frac{\partial s}{\partial h}$ שחישבנו בשלבים קודמים של האלגוריתם ומכונה upstream gradient, ואנחנו רוצים לחשב את $\frac{\partial s}{\partial z}$ שיועבר לשלבים הבאים באלגוריתם ומכונה downstream gradient. נעשה את זה ע"י לחשב את ה-local gradient של הפונקציה f ואז להשתמש בכלל השרשרת (downstream=upstream×local), באופן הבא עבור פונקציה עם פרמטר אחד:



ובאופן הבא עבור פונקציה עם מספר פרמטרים:



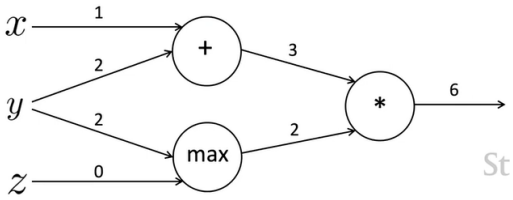
אם מגיעים לצומת בגרף שבה בתהליך ה-backpropagation יש יותר מ-upstream gradient אחד, סוכמים את התוצאות של המכפלות באופן הבא (הפונקציות a,b,f הן לא חלק מהאלגוריתם, אלא דוגמא לאופרטורים שיכולים להיות ב-nodes של הרשת):



דוגמא לחישוב האלגוריתם לכל אורכו

(זאת לא פונקציה שמתאימה לרשת נוירונים בעולם האמיתי, היא רק לצורך הדגמה). נסתכל על הפונקציה $f(x,y,z) = (x + y)max(y,z)$ עם הפרמטרים $x = 1, y = 2, z = 0$.

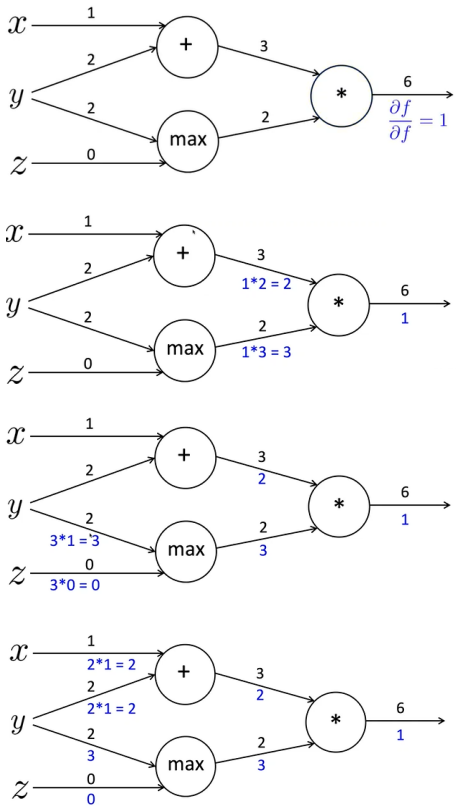
בשלב ה-forward propagation נפעפע את הפרמטרים לאורך הגרף (נסמן a=x+y, b=max(y,z), f=ab):



בשלב ה-backpropagation, נרצה קודם כל לחשב את ה-local gradients עבור שלושת ה-nodes (a,b,f) ביחס לכל אחד מהקלטים שלהם, ונקבל את הגרדיאנטים הלוקליים הבאים:

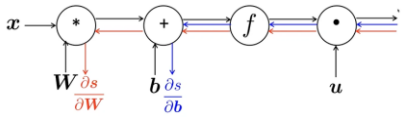
- $\frac{\partial a}{\partial x} = 1$
- $\frac{\partial a}{\partial y} = 1$
- $\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1$
- $\frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$
- $\frac{\partial f}{\partial a} = b = 2$
- $\frac{\partial f}{\partial b} = a = 3$

עכשיו אחרי שחישבנו את הגרדיאנטים הלוקליים, נוכל להתחיל מלחשב את $\frac{\partial f}{\partial f} = 1$ ולהתחיל לחשב אחורנית את downstream=upstream×local עבור כל פיצול בגרף:

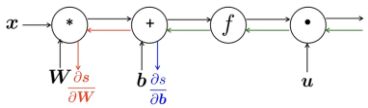


כעת נקבל $\frac{\partial f}{\partial x} = 2$, $\frac{\partial f}{\partial y} = 3 + 2 = 5$, $\frac{\partial f}{\partial z} = 0$. המספרים האלו נותנים לנו אינטואיטיבית מידע לגבי כמה כל אחד מהפרמטרים x,y,z משפיע על התוצאה s (כלומר, כמה התוצאה תושפע אם נשנה במעט את הערך של הפרמטר הספציפי).

כדי לשפר את היעילות החישובית, נרצה לחשב את כל הגרדיאנטים בבת אחת. למעשה, הדרך שהוצגה למעלה שבה מחשבים את הגרדיאנטים באופן סדרתי היא מאוד לא יעילה. זה יהיה מאוד לא נכון לחשב קודם את $\frac{\partial s}{\partial b}$ ואז באופן בלתי תלוי לחשב את $\frac{\partial s}{\partial W}$ בדיוק באותה הדרך (בגלל שהחישובים לאורך כל הדרך זהים, נחשב פעמיים כל אחד מהם שלא לצורך):

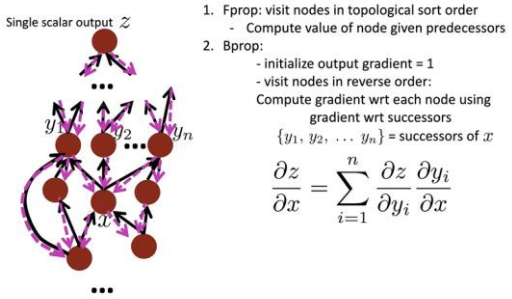


לכן נרצה לחשב כל אחד מהחישובים האלו פעם אחת ולהשתמש בהם בכל המקומות בהם הם נדרשים:



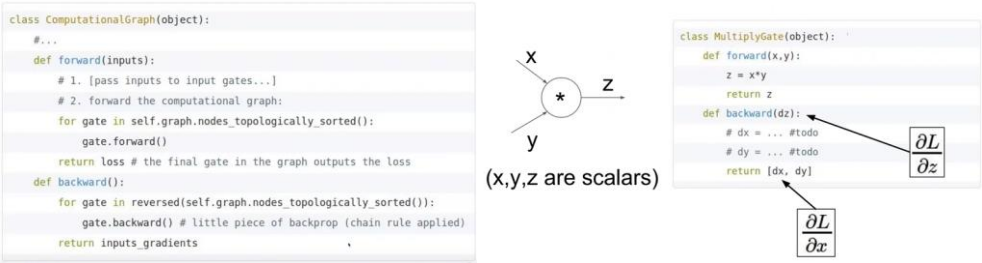
סכמה כללית של אלגוריתם ה-BACKPROPAGATION

עבור גרף כללי של רשת נוירונים (גרף מכוון וחסר מעגלים):



כאשר מבוצע נכון, הסיבוכיות O של forward propagation ו-backpropagation היא <u>זהה</u>. באופן כללי, לרשתות נוירונים יש מבנה שמורכב משכבות, ולכן אנחנו יכולים להשתמש בנוטציה של מטריצות ויעקוביאנים בחישוב הגרדיאנטים (ובכך לבצע את החישוב באופן מקבילי והרבה יותר יעיל).

automatic differentiation - חישוב הגרדיאנט יכול להיות מוסק באופן אוטומטי מהייצוג הסימבולי של ה-forward propagation. כל node מייצג מה אנחנו צריכים לעשות כדי לחשב את הפלט שלו ואיך לחשב את הגרדיאנט הלוקלי שלו ביחס לכל אחד מהקלטים שלו. פריימוורקים מודרניים של deep learning (כמו TensorFlow ו-PyTorch) מכילים מימוש ל-backpropagation שמבצע 90% מהחישובים של ה-automatic differentiation אבל לא מחשבים באופן סימבולי את הגרדיאנטים. כלומר, כשמקימים רשת צריך לכתוב באופן מפורש את הגרדיאנט בכל node או שכבה:



בדיקה ידנית של גרדיאנטים באמצעות חישוב נומרי – דרך לוודא את החישוב של הגרדיאנטים. עבור h קטנים מאוד ($h \approx 1e-4$) נוכל לחשב את $f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$ ובכך להעריך נומרית את הערך של הגרדיאנט. ההערכה הזו **מאוד** איטית (צריך לחשב מחדש את f עבור כל פרמטר במודל). זה שימושי כדי לוודא את המימוש של הגרדיאנטים.

SYNTACTIC STRUCTURE AND DEPENDENCY PARSING

SYNTACTIC STRUCTURE וסוגי עמימות

יש שתי דרכים להסתכל על תחביר, כלומר להגדיר מבנה תחבירי (syntactic structure):

- CFG – Context Free Grammar הוא ייצוג מפורש של התחביר בתור חוקים שמחלקים משפטים לחלקים באופן מקונן nested – כל חלק במשפט מורכב מחלקים שמורכבים מחלקים, כך עד שמגיעים לטרמינלים - היחידה הלשונית הבסיסית ביותר המיצגת כל מילה.
- Dependency structure מציין עבור כל מילה באילו מילים היא תלויה (מתארת, קשורה וכו'). עבור כל משפט יוצרים עץ תלויות dependency tree שבו כל עלה הוא מילה שמחוברת בצלע למילה בה היא תלויה.

הסיבה שבגללה אנחנו צריכים להגדיר מבנה תחבירי (syntactic structure) היא ששפות אנושיות מאפשרות להעביר רעיונות מורכבים ע"י שרשור מילים אחת אחרי השני לכדי יחידות גדולות יותר. המאזינים צריכים לפענח בעצמם מה מתאר modifies מה. לכן, מודל שפתי צריך להבין את המבנה של המשפט בכדי להיות מסוגל לפענח את המשמעות של השפה באופן מדויק.

עמימות AMBIGUITY

לפעמים עבור אותו המשפט ישנה עמימות ambiguity, כלומר יתכנו כמה פירושים אפשריים. סוג העמימות מאוד תלוי בשפה, שכן הוא נוצר כתוצאה ממבנה מסוים שיכולות להיות לו כמה משמעויות ולכן תלוי באופן מאוד הדוק בתחביר של השפה. מספר דוגמאות מעניינות לעמימות סמנטית מסוג **PP attachment ambiguity**:

- San Jose cops kill man with knife (השוטרים השתמשו בסכין כדי להרוג את האיש / האיש שנהרג החזיק סכין)
- Scientists count whales from space (הלוויתנים הגיעו מהחלל / הספירה התבצעה מהחלל)

למעשה, עבור משפט המורכב מ-n ביטויים מסוג PP, נקבל שמספר הסידורים האפשריים של הפסוקיות הוא מספר הקטלאן $C_n = \frac{(2n)!}{(n+1)!n!}$ (כיוון שזו סדרה שגדלה בקצב אקספוננציאלי כש-n גדל לאינסוף, המשמעות היא שכמות הביטויים משפיעה באופן מאוד משמעותי על כמות הפתרונות האפשריים שיש לבעיה).

סוג אחר של בעית עמימות היא **coordination scope ambiguity**, כלומר מקרים בהם לא ברור מה ה-scope של מילת modifier במשפט, לדוגמא:

- Shuttle veteran and Iongtime NASA executive Fred Gregory appointed to board (האם פרד גרגורי הוא ה-shuttle veteran בעצמו או שמדובר בשני אנשים שונים?)
- Doctor: no heart, cognitive issues (האם המודיפייר no מתייחס רק ללב או גם ללב וגם לבעיות הקוגניטיביות?)

סוג נוסף של בעית עמימות היא **Adjectival / Adverbial Modifier Ambiguity**, שבהן עבור מודיפייר מסוג שם תואר או תואר הפועל שלא ברור איזה חלק במשפט הוא מתאר. לדוגמא:

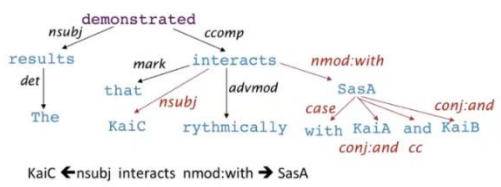
- Students get first hand job experience (האם first מתאר את hand או את hand job?)

סוג נוסף הוא **Verb Phrase (VP) attachment ambiguity**, שמאוד דומות לבעיות Adjectival / Adverbial Modifier Ambiguity רק שבמקרה הזה המודיפייר מתייחס לחלק גדול יותר במשפט. לדוגמא:

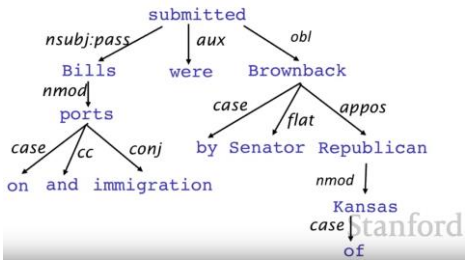
- Mutilated body washes up on Rio beaches to eb used for Olympics beach volleyball (האם to be used for מתייחס ל-Rio beaches או ל-Mutilated body?).

DEPENDENCY GRAMMAR AND TREEBANKS

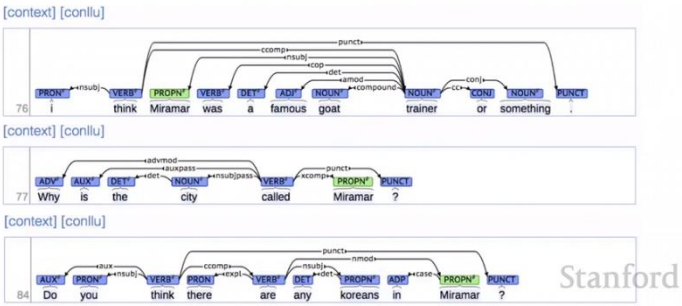
חלוקת המשפט לעץ תלויות יכולה לעזור לחלץ את המשמעות הנכונה של כותב המשפט. למשל, לחלץ את המשמעות של המשפט משמאל שמתייחס לאינטראקציה בין חלבון לחלבון.



באופן כללי, בגישת Dependency syntax התפיסה היא שהמבנה התחבירי מוגדר מיחסים בין עצמים לקסיקליים (לרוב צמדים אסימטריים של יחסים / "חצים" המכונים תלויות dependencies). לרוב נותנים לכל חץ כיוון (מהראש למילה שתלויה בו) סוג type שמציין את היחס התחבירי בין המילים. לרוב, התלויות יוצרות עץ (גרף קשיר חסר מעגלים עם שורש יחיד – לרוב נוסיף מילת ROOT שאינה במשפט המקורי כדי לוודא שהשורש אכן יהיה יחיד). לדוגמא:



לרעיון הזה של ניתוח תחביר באמצעות תלויות יש היסטוריה ארוכה מאוד (החל מהמאה החמישית לפני הספירה), הוא היה נפוץ מאוד בבלשנות במאה ה-20 והשימוש בו לצרכי NLP החל בשנות ה-60 שלה. בהתחלה ניסו לייצר פרסרים לעצים תחביריים ע"י להגדיר מפורשות את החוקים התחביריים של השפה, אבל זה לא עבד טוב מאוד.



Treebank הוא שם כללי לסוג של מאגר מידע המכיל כמות גדולה של עצים תחביריים (עבור משפטים) שתויגו ע"י בני אדם (לרוב, בלשנים). לדוגמא, http://universaldependencies.org/ הוא דאטאסט שכזה. אמנם לוקח יותר זמן ומאמץ לייצר דאטאסטים של Treebank בהשוואה לכתיבה ידנית של חוקי התחביר של השפה, אבל יש להם יתרונות משמעותיים – הם מאפשרים לנצל את העבודה שבוצעה כדי לייצר אותם למגוון צרכים (כלומר, ליצירת מגוון פרסרים לבעיות שונות, לדוגמא תיוג POS, ואפילו לבעיות פסיכולינגוויסטיות וסטטיסטיות שאינן בעיות NLP). בנוסף, ה-coverage שלהם רחב מאוד ומתפרס על חלקים גדולים יותר בשפה ולא רק על אינטואיציות שמנסות להכליל אותה. כמו כן, הם מספקים מידע על תדירויות והתפלגויות של סוגי קשרים ותלויות, וכן מאפשרים לבצע אוליואציה למערכות NLP.

מקורות המידע שמשמשים לניתוח תלויות DEPENDENCY PARSING

- Bilexical affinities – להסתכל על שתי המילים בקצה של יחס תלות מסוים (חץ בעץ), ולנסות להבין את התלות הזו היא סבירה.
- Dependency distance – רוב התלויות במשפטים אנושיים הן קצרות יחסית (כלומר החץ לא "מדלג" על הרבה מילים במשפט), לכן אם יש תלות עם מרחק ארוך בין המילים שהיא מקשרת יכול להיות שהיא לא סבירה.
- Intervening material – בד"כ לא קורה שתלויות מתפרסות מעבר לפועל או סימן פיסוק.
- Valency of heads – כמה תלויות או מאיזה צד יוצאות התלויות של שורש מסוים.

כדי לבצע ניתוח תלויות dependency parsing עבור משפט מסוים, צריך לבחור לכל מילה במשפט מה המילה (כולל המילה הנוספת ROOT) שהיא תלויה בה. בד"כ ישנם אילוצים שגורמים לכך שהתוצאה תהיה <u>עץ</u> תלויות:

- רק מילה אחת יכולה להיות תלויה ב-ROOT
- לא יתכנו מעגלים, כלומר לא יכול להיות שA תלוי בB וגם B תלוי ב-A (ישירות או בעקיפין).

פרוג'קטיביות

אילוץ נוסף שצריך להחליט לגביו הוא האם החצים יכולים לחצות זה את זה (ולגרום לכך שהעץ יהיה לא פרוג'קטיבי non-projective) או שאוסרים על האפשרות הזו. ההגדרה של **projective parse** היא שאין צלעות בעץ שחוצות זו את זו כשהמילים מסודרות לינארית לפי הסדר המקורי שלהן במשפט והצלעות נמצאות כולן מעל המילים. תלויות שתואמות ל-CFG הן בהכרח פרויג'קטיביות. רוב המבנים

התחביריים הם אכן פרוג'קטיביים, אבל תאוריית התלויות לרוב מאפשרת את קיומם של מבנים לא פרוג'קטיביים כדי לאפשר תמיכה ב-displaced constituents. אי אפשר לזהות בקלות את המשמעות של מבנים מסוימים באופן נכון אם אוסרים על קיומן של תלויות לא פרוג'קטיביות (שחוצות תלויות אחרות).

TRANSITION BASED DEPENDENCY PARSING

שיטות לבנית עצי תלויות

1. **תכנון דינמי** – פותח ב-1996, פועל בסיבוכיות $O(n^3)$ ע"י להפיק זוגות עם ראשים בקצוות שלהם במקום באמצע.
2. **אלגוריתמים מבוססי גרפים** – חישוב MST לכל משפט. פותח ב-2005 אז הוא מומש באמצעות קלסיפייר ML קלאסי. לאחר מכן ב-2017 מומש באמצעות Neural graph-based parser שהיה מוצלח מאוד.
3. **אלגוריתם סיפוק אילוצים** – צלעות שלא מקיימות אילוצים קשים hard constraints מוסרות מהעץ. מומש ב-1990.
4. פרסינג מבוסס מעברים Transition based parsing או "ניתוח תלויות דטרמיניסטי" – אלגוריתם חמדן שמונחה ע"י קלסיפיירי ML. פותח ב-2008 והוכח כיעיל מאוד.

אלגוריתם TRANSITION BASED DEPENDENCY PARSING

שיטה פשוטה יחסית לייצג את המבנה התחבירי של משפט. זהו למעשה צורה פשוטה של greedy discriminative dependency parser. הפרסר מבצע סדרה של פעולות bottom-up (פעולות כמו shift או reduce שיוצרות תלויות עם ראש המחובר ימינה או משאלה). הפרסר מאופיין ע"י:

- **מחסנית σ** שהראש שלה נמצא מצד ימין ומאותחלת עם מילת ה-ROOT
- **באפר β** שהראש שלו נמצא מצד שמאל ומאותחל עם כל המילים במשפט הקלט
- קבוצת צלעות המייצגות תלויות A שמאותחלת ריקה
- קבוצת פעולות

כלומר, בתחילת ריצת האלגוריתם הרכיבים נראים כך: $\sigma = [\text{ ROOT }], \quad \beta = w_1, \dots, w_n, \quad A = \emptyset$

הפעולות שניתן לבצע הן מתוך הפעולות הבאות:

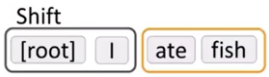
- **Shift** – לוקחים את המילה שבראש הבאפר ומכניסים אותה למחסנית $\sigma, w_i|\beta, A \Rightarrow \sigma|w_i, \beta, A$
- $\boldsymbol{Left-Arc_c}$ – לוקחים את שתי המילים העליונות במחסנית ומחברים צלע בין המילה הימנית למילה השמאלית, תוך הוספת הצלע שנוצרה לקבוצת הצלעות A באופן הבא - $\sigma|w_i|w_j, \beta, A \Rightarrow \sigma \mid w_j, \beta, \text{AU}\{r(w_j, w_i)\}$
- $\boldsymbol{Right-Arc_c}$ – לוקחים את שתי המילים העליונות במחסנית ומחברים צלע בין המילה השמאלית למילה הימנית, תוך הוספת הצלע שנוצרה לקבוצת הצלעות A באופן הבא - $\sigma|w_i|w_j, \beta, A \Rightarrow \sigma \mid w_i, \beta, A \cup \{r(w_i, w_j)\}$

האלגוריתם יסיים את ריצתו כאשר המחסנית תכיל את כל המילים במשפט והבאפר יהיה ריק, כלומר $\sigma = [w], \ \beta = \emptyset$.

דוגמא לריצת אלגוריתם TRANSITION BASED DEPENDENCY PARSING

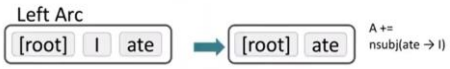
נציג דוגמא של ריצת האלגוריתם על המשפט "I ate fish". בתחילת ריצת האלגוריתם, המחסנית תכיל רק את ROOT ושאר המילים במשפט יהיו בבאפר:

בגלל שיש לנו רק מילה אחת במחסנית, האפשרות היחידה שיש לנו לפעולה היא shift, נבצע אותה:



למרות שיש לנו אפשרות לבצע פעולות שמוסיפות צלעות, נבחר לבצע שוב Shift:

כעת נבצע פעולת Left Arc ונוסיף לקבוצת הצלעות A את הצלע שיצרנו שמחברת את ate ל-i באמצעות היחס nsubj:



נבצע Shift:

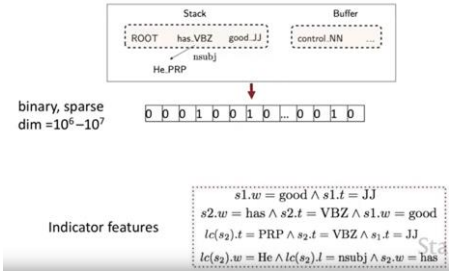
נבצא Right Arc ונוסיף לקבוצת הצלעות A את הצלע שמחברת בין ate ל-fish באמצעות היחס obj:

נבצע Right Arc בפעם האחרונה כדי לחבר בין Root ל-ate (תוך עדכון של A בהתאם) ובכך נסיים את ריצת האלגוריתם:

MALTPARSER

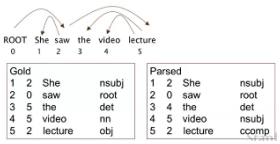
אנחנו רוצים לדעת איך לבחור את הפעולה הנכונה בכל שלב של האלגוריתם (מבלי לבחון את כל האפשרויות בכל שלב ובכך לבצע כמות עבודה אקספוננציאלית באורך המשפט). נרצה ליצור קלסיפייר ML שבהנתן מחסנית ובאפר יודע לבחור את הפעולה הטובה ביותר מבין שלוש הפעולות האפשריות. כל פעולה נחזית באמצעות קלסיפייר דיסקרימינטיבי (למשל softmax classifier) שבוחר בין שלוש האפשרויות לפעולות. הפיצ'רים שמוזנים לקלסיפייר הם שתי המילים בראש המחסנית כולל ה-POS שלהן והמילה בראש הבאפר כולל ה-POS שלה.

הייצוג הקונבנציונלי של הפיצ'רים לקלסיפייר נראה כך – משתמשים בפיצ'רים סימבוליים / משתני אינדיקטור שיוצרים וקטור דליל מאוד שכל תא בו מייצג קונפיגורציה אפשרית של שייכות מילה למחסנית או לבאפר ושל ה-POS שלה:



אפשר לשפר את התוצאות (במחיר של איטיות בחישוב) ע"י לבצע Beam Search – שומרים בכל שלב את k ההתחלות הטובות ביותר לפרסורים ומפתחים אותן. המודלים האלו הם רק ממש מעט פחות טובים (בדיוק הפתרון) בהשוואה למודלי state of the art שקיימים היום, אבל היתרון שלהם הוא שהם מאפשרים חישוב מהיר מאוד (בזמן לינארי בכמות המילים במשפט) עם דיוק גבוה.

אוולואציה של עצי תלויות



כדי לבצע אוולואציה, נסתכל על עץ ה"זהב" (התיוג המקורי שנוצר ע"י בני אדם שקובע היכן צריכה להיות קשת) לבין העץ שיצר הפרסר שלנו. דיוק Accuracy – מוגדר ע"י $\frac{\#\ correct\ arcs}{\#\ of\ arcs}$, ונסתכל על שני מדדים שנבדלים באופן שבו הם מגדירים את ספירת הצלעות הנכונות:

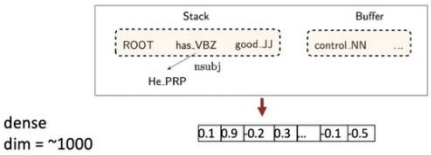
- Unlabeled Accuracy Score – UAS שלפיו צלע נכונה אם היא מחברת בין מילים נכונות בכיוון הנכון

- LAS – Labeled Accuracy Score – שמרחיב את הדרישה של UAS בכך שהוא דורש שהצלע תקבל מהמודל את ה-label (שמייצג POS) הנכון.

NEURAL DEPENDENCY PARSING

עד עכשיו דיברנו על transition based parsers, שהם שיטה פשוטה יחסית לייצג את המבנה התחבירי של משפטים שהיתה נפוצה לפני שעברו להשתמש ברשתות נוירונים לפתרון הבעיה הזו. החסרון הגדול ביותר של פרסרים מבוססי מעברים הוא שהם עובדים עם פיצ'רי אינדיקטור, כלומר מגדירים מפורשות איזה שהוא תנאי שצריך להתקיים עבור קונפיגורציה מסוימת של הפרסר (למשל – "המילה בראש המחסנית היא "good" וה-POS שלה הוא שם תואר, והמילה הבאה אחריה היא פועל"). זה גורר שלוש בעיות –

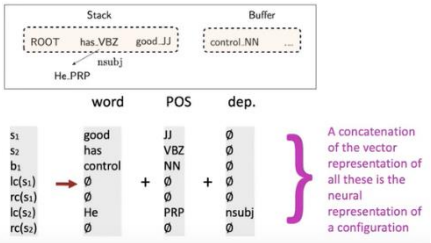
- וקטור הפיצ'רים שנוצר בשיטה הזו הוא מאוד דליל (sparse)
- וקטור הפיצ'רים שנוצר בשיטה הזו הוא לא שלם (incomplete), כלומר הוא מייצג רק מצבים שנצפו בקורפוס האימון, אך סביר מאוד שישנם מצבים נוספים שקיימות בשפה עצמה שלא הופיעו בקורפוס האימון ולכן "יתפספסו" ע"י שיטה זו.
- השיטה מאוד יקרה חישובית – יותר מ-95% מזמן הריצה הנדרש לתהליך הפרסור מוקדש לחישוב הפיצ'רים האלו.



הבעיות האלו הובילו למעבר לגישה המשתמשת ברשתות נוירונים, שלומדות וקטור פיצ'רים מייצג שהוא מאוד דחוס וקומפקטי. בשיטה זו, הקונפיגורציה עצמה למעשה זהה לקונפיגורציה בשיטת המעברים – אנחנו עובדים עם מחסנית stack ועם באפר buffer. ההבדל הוא שבמקום לייצג את הקונפיגורציה ע"י וקטור פיצ'רי אינדיקטור מאוד דליל, אנחנו מייצגים אותם באמצעות וקטור פיצ'רים דחוס מאוד (ממימד בסדר גודל של 1,000). מסתבר שהייצוג הזה גם מחזיר תוצאות יותר טובות – ומשפר את הדיוק (accuracy) של המודל בכ-2% בהשוואה לפרסר מבוסס מעברים (בין אם הם labeled או unlabeled) תוך שיפור משמעותי בזמן הריצה הנדרש. גם אם משווים את פרסרים מבוססים רשתות נוירונים לפרסרים מבוססי גרף (שלא למדנו עליהם בקורס הזה, למשל TurboParser), אמנם הדיוק שהושג ע"י רשת הנוירונים היה נמוך במעט (באופן שולי) בהשוואה לפרסר מבוסס הגרף, אבל זמן הריצה שהוא דרש היה קטן בשני סדרי גודל.

קלט לפרסר תלויות מבוסס רשת נוירונים

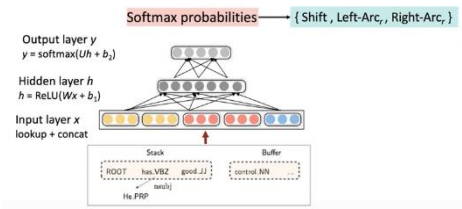
מייצגים כל מילה בתור וקטור דחוס ממימד d (שהוא למעשה word embedding, כלומר אנחנו מצפים שמילים דומות ייוצגו באמצעות וקטורים הקרובים זה לזה במרחב). כיוון שאנחנו רוצים ליצור פרסר תלויות (dependencies), יש חשיבות גדולה גם לחלק הדיבר POS ול-dependency label של כל מילה, ולכן כל אחד מהם גם מיוצג בתור וקטור ממימד d. גם מהוקטורים האלו אנחנו מצפים תכונה דומה לזו שאנחנו מצפים מ-word embedding – שחלקי דיבר או תלויות דומות יהיו קרובות זו לזו במרחב, למשל ששם עצם יחיד ושם עצם רבים יהיו קרובים זה לזה. כלומר, כל שלישיה כזו של וקטורים ממימד d מייצגת את מצב מסוים של הקונפיגורציה של המחסנית והבאפר. d הפיצ'רים בכל וקטור מייצגים משמעויות קבועות מראש בנוגע למצב של המחסנית והבאפר, למשל המילה הנוכחית (המילה בראש המחסנית), המילה הבאה, המעברים transitions של כל אחת מהמילים האלו למילים שלפניהן ואחריהן וכו'.



לאחר שיוצרים שלישיה כזו של וקטורים (word embedding, וקטור POS ווקטור תלויות dependencies), "מדביקים" אותם ביחד ליצירת וקטור ארוך ממימד *3d* שישמש אותנו בתור הייצוג שיוכנס כקלט לרשת הנוירונים.

ארכיטקטורה בסיסית של פרסר תלויות מבוסס רשת נוירונים

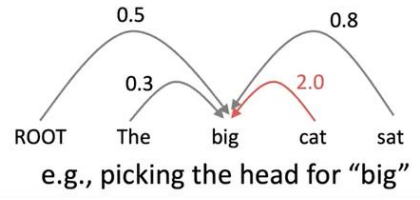
מודל פרסר תלויות מבוסס רשת נוירונים הראשון שהוצע בשנת 2014 הוא למעשה רשת נוירונים פשוטה – שמקבלת ייצוג של קונפיגורציה מסוימת של המחסנית והבאפר ומחליטה באיזה מעבר transition להשתמש (Shift, Right-Arc, Left-Arc). שכבת הקלט היא וקטור ממימד 3d כפי שהוסבר מעלה, שכבת הביניים החבויה תהיה ReLU, ושכבת הפלט תהיה softmax. שכבת הפלט תספק וקטור התפלגות שיאפשר לנו לבחור באיזה מעבר transition אנחנו רוצים לבחור.



מודל רשת נוירונים פשוט שכזה יוצר פרסר תלויות מדויק למדי שמצליח לקבוע בצורה טובה את המבנים התחביריים של משפטים (יותר טוב מ-greedy parsers גם בהיבט ה-accuracy וגם בזמן הריצה).

בשנים שעקבו לפרסום מודל רשת הנוירונים הראשון ששימש לפתרון הבעיה הזו, התפרסמו מודלים נוספים (למשל ע"י סטנפורד וגוגל) שהציעו ארכיטקטורות מורכבות יותר לרשתות נוירונים עבור יצירת עצי תלויות. ב-2016 גוגל פרסמה את המודל Parsey McParceFace שכונה "הפרסר המדויק בעולם" (שהשיג 2% יותר דיוק accuracy בהשוואה למודל הפשוט של סטנפורד מ-2014, והגיע לכמעט 95% דיוק).

GRAPH BASED DEPENDENCY PARSERS



עבור כל קומבינציה של מילה במשפט עם כל מילה אחרת במשפט (שמשמשת בתור "ראש" פוטנציאלי עבורה בעץ התלויות), מחשבים score שמייצג מה ההסתברות שהמילה השניה תהיה הראש של המילה הראשונה בעץ התלויות. כדי לקבוע את ה-score לא מספיק לדעת את שתי המילים עצמן – אנחנו צריכים ייצוג שמתייחס להקשר של כל אחת מהן (אותו נראה בהרצאות הבאות). לאחר שיוצרים score כזה עבור כל זוג מילים שונות, נקבל גרף, ובתוכו נמצא את עץ התלויות הנכון באמצעות אלגוריתם MST (minimum spanning tree) שמשתמש ב-scores בתור משקולות על הצלעות ומתחיל מה-ROOT.

ב-2017 סטנפורד הציגו מודל שכזה שהצליח להשיג דיוק accuracy טוב יותר (95.7%) ממודל רשת הנוירונים של גוגל (אם כי חשוב לציין שלוקח לו הרבה יותר זמן לרוץ, שכן עבור משפט עם n מילים הוא רץ בסיבוכיות n^2).

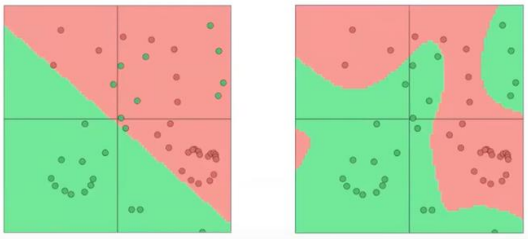
רשתות נוירונים, RNN

רשתות נוירונים

היתרון של רשתות נוירונים על פני למידת מכונה קלאסית

דיברנו בשלב מוקדם יותר בקורס על קלסיפייר *softmax*, שמשייך תיוג $y \in C$ בהתבסס על קלטים $x \in \mathbb{R}^d$ תוך שימוש בהסתברות $p(y \mid x) = \frac{\exp(W_y \cdot x)}{\sum_{c=1}^{C} \exp(W_c \cdot x)}$. מודל שכזה מאומן באמצעות עדכון של מטריצת המשקולות $W \in \mathbb{R}^{C \times d}$ תוך נסיון למזער את ה- *cross entropy loss* המוגדר באופן הבא - $\sum_i -\log p(y_i \mid x_i)$.

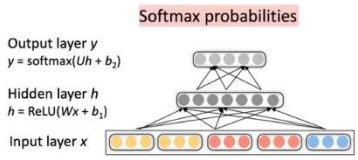
העניין עם קלסיפייר *softmax* (בדומה לקלסיפיירי *ML* מסורתיים אחרים כמו *naïve Bayes*, *SVM*, רגרסיה לוגיסטית וכו') היא שמחלקת ההיפותזות שלהם מורכבת מקווי הכרעה לינאריים. ההגבלה הזו לסיווג באמצעות מפריד לינארי עשויה להיות די מגבילה, ואפילו לא שימושית כלל אם הבעיה אותה אנחנו מנסים לסווג מורכבת מכדי לפתור אותה באמצעות מפריד לינארי שכזה.



כאן נכנס לידי ביטוי היתרון של רשתות נוירונים – הן מסוגלות ללמוד פונקציות הרבה יותר מורכבות עם קווים מפרידים שאינם לינאריים. בתמונה משמאל ניתן לראות שתי תוצאות של מודלים – השמאלית מייצגת מפריד לינארי שנלמד באמצעות שיטת NL קלאסית, והימנית מייצגת מפריד לא לינארי שנלמד באמצעות רשת נוירונים (ומאפשר לייצג את המורכבות שבדאטא ולכן מתייג נכון את הנקודות הירוקות שהתפספסו ע"י המפריד הלינארי).

רשת נוירונים פשוטה מסוג FEED FORWARD

מה שרשת נוירונים פשוטה מסוג feed forward עושה הוא לקחת קלט, להעביר אותו דרך שכבת ביניים לא לינארית, ואז להעביר את התוצר שלה דרך שכבה שממירה אותו לוקטור התפלגות. כלומר, יש מספר שכבות –

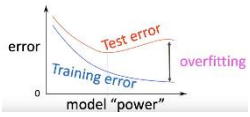


- **קלט** - x (וקטור פיצ'רים דחוס)
- **שכבה חבויה hidden layer** לא לינארית (למשל, השכבה יכולה להיות הפונקציה $h = \text{ReLU}\,(Wx + b_1)$). השכבה החבויה מנסה "לפענח" את מה שוקטור הקלט מנסה לייצג – למעשה, היא יוצרת מעין מרחב ביניים וקטורי נסתר (ביניים במובן שהוא נמצא בין הקלט לפלט, נסתר במובן שהוא אינו נצפה מהתוצאה הסופית שנפלטת מהרשת כולה) ובו היא מזיזה וממקמת את הקלט מחדש, כך שיהיה קל לסווג אותו באמצעות קלסיפייר לינארי בשכבה הבאה.
- **שכבת output** – קלסיפייר לינארי (למשל $y = \text{softmax}\,(Uh + b_2)$) שפולט וקטור התפלגות המשמש לקביעת התיוג של הקלט.

לאחר ביצוע ה-feed forward שהוסבר מעלה, מחושב ה-Log loss (המכונה גם cross loss entropy) על וקטור הפלט, ועליו מבצעים back propagation עד ל-embeddings.

רגולריזציה

רוב פונקציות ה-loss מכילות גם פרמטר רגולריזציה על כל הפרמטרים θ. למשל, שיטת רגולריזציה נפוצה היא רגולריזצית L2 המוגדרת באופן הבא - $\lambda \sum_k \theta_k^2$ (ששווה לשים לב שהוא מחושב פעם אחת לכל פרמטר, מבלי שנהיה צריכים לחשב אותו שוב על כל דגימה בדאטא האימון). רגולריזציה מביאה איתה עוד היפר-פרמטר שצריך לקחת בחשבון, והוא פרמטר הרגולריזציה המסומן בדרך כלל בתור λ (ככל שהוא יותר גדול, יש יותר רגולריזציה).



התפיסה הקלאסית היא שמטרתה של רגולריזציה היא למנוע overfitting (מצב בו הטעות על דאטא האימון קטנה אבל הטעות על דאטא ה-test גדלה – כלומר המודל נצמד מידי לדאטא האימון ומתקשה להכליל את הידע שלמד באופן שיאפשר לתייג כראוי דוגמאות חדשות) במצב בו יש לנו הרבה מאוד פיצ'רים או כשהמודל שלנו מאוד מורכב או עמוק.

כיום, רגולריזציה מפיקה מודלים שמאפשרים להכליל בצורה טובה (כלומר – אנחנו מצפים שה-test error יהיה יותר גבוה מה-training error, אבל לא היינו רוצים לראות עלייה ב-test error אלא ירידה עד התאזנות בשלב מסוים) כאשר מדוברים במודלים "גדולים",

ופחות מתמקדת במניעת overfit (שכן מודלים גדולים מאוד בהגדרה יצרו overfit מאוד משמעותי על ה-training data – באיזה שהוא שלב, עם מספיק אימון, הם יוכלו להגיע גם ל-training error של 0).

DROPOUT

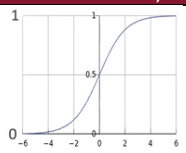
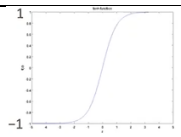
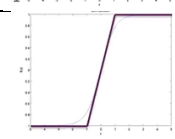
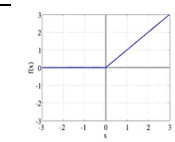
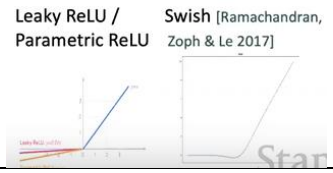
Dropout היא טכניקת רגולריזציה שמשתמשים בה ברשתות נוירונים, שמאפשרת הכללה טובה ומניעת Feature co-adaptation (שהמשמעות או ההשפעה של פיצ'ר מסוים יהיו תלויים בהופעה של פי'צר אחר). כיום היא נחשבת לשיטת רגולריזציה חזקה ו- feature dependent (כלומר חלק מהפיצ'רים יהיו יותר regularized וחלקם פחות – עוצמת הרגולריזציה תלויה בכמה הפיצ'ר נמצא בשימוש).

בשיטה זו, בזמן אימון המודל, לכל batch אימון, משמיטים (dropping, כלומר מאפסים) באופן אקראי 50% מהקלטים לכל נוירון ברשת. לאחר מכן, בשלב ה-test, כופלים בחצי כל אחת מהמשקולות שנשארו. השיטה הזו מונעת feature co-adaptation בכך שפיצ'ר לא יכול להיות תלוי בפיצ'ר מסוים אחר בגלל שאנחנו מורידים בכל פעם באופן אקראי חלק ניכר מהפיצ'רים. אפשר לחשוב על השיטה הזו בתור שיטת bagging / ensemble.

ברשתות עם שכבה אחת, זה מהווה סוג של אמצע בין naïve bayes (שם כל המשקולות של הפיצ'רים נקבעים באופן עצמאי ובלתי תלוי) לבין רגרסיה לוגיסטית (שם המשקולות נקבעות בהקשר של כל הפיצ'רים בבת אחת).

שכבות לא ליניאריות ברשתות נוירונים

כדי שתהיה משמעות לכך שיש מספר שכבות ברשת נוירונים, צריך שחלקן יהיו פונקציות שאינן ליניאריות (שכן אם כולן ליניאריות אפשר להתייחס לרשת כולה בתור הרכבה של פונקציות ליניאריות, מה ששקול להפעלה של פונקציה ליניארית יחידה). כמה דוגמאות לפונקציות לא ליניאריות שמשתמשים בהן ברשתות נוירונים:

שם	פונקציה	גרף	תכונות
Logistic (sigmoid)	$\frac{1}{1+\exp(-z)}$		ממפה את כל הערכים לבין 0 ל-1, כשערכים שליליים יהיו בין 0 לחצי, וערכים חיוביים יהיו בין חצי ל1. השתמשו בה הרבה ברשתות הנוירונים הראשונות שפותחו. החסרון שלה הוא שכל הערכים שהיא ממפה אליהם הם חיוביים. איטית לחישוב בגלל האקספוננט. כיום לא משמשת כשכבה לא ליניארית ברשתות נוירונים, אבל כן משתמשים בה כשרוצים לקבל וקטור התפלגויות.
tanh	$\frac{e^z - e^{-z}}{e^z + e^{-z}}$		יוצרת צורת S שדומה לסיגמואיד (למעשה זאת גרסה מוסטת ו-rescaled של הסיגמואיד) שממפה לערכים בין 1- ל-1 (כך שהיא לא ממפה רק לערכים חיוביים). איטית לחישוב בגלל האקספוננט.
hard tanh	$\begin{cases} -1 \text{ if } x < -1 \\ x \text{ if } -1 <= x <= 1 \\ 1 \text{ if } x > 1 \end{cases}$		גרסה "קשיחה" של הטנגנס ההיפרבולי, שגם ממפה לערכים בין 1- ל-1. מהירה לחישוב.
ReLU	$max(z, 0)$		כיום זו הפונקציה שהכי נפוצה לשימוש בתור שכבה לא ליניארית ברשתות נוירונים. מהירה לחישוב. מאפשרת לאמן את הרשת מאוד מהר תודות ל-gradient backflow טוב שנוצר כתוצאה מכך שהשיפוע שלה הוא 1. יש עוד גרסאות שמאוד דומות לה (אך לא נותנות יתרון משמעותי עליה) – 

אתחול פרמטרים

בדרך כלל כשבונים רשת נוירונים, מאתחלים את המשקולות עם ערכים רנדומליים קטנים (שאינם אפס), מתוך התפלגות אחידה בטווח $[-r, r]$ כאשר r נבחר כך שלא נקבל מספרים גדולים מידי או קטנים מידי.

בד"כ מאתחלים את פרמטרי ה-bias בשכבות החבויות ל-0, ובשכבת ה-output לערך שיהיה אופטימלי אם כל המשקולות היו 0 (למשל, mean target או inverse sigmoid of mean target).

דרך אתחול נפוצה (למשל ב-PyTorch) היא Xavier initialization – שיטה שבה השונות היא פרפורציונית באופן שלילי ל-fan-in (הגודל של השכבה הקודמת) המסומן n_{in} ו-*fan-out* (הגודל של השכבה הבאה) המסומן n_{out}, המוגדרת באופן הבא - $\text{Var}(W_i) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$.

אימון רשתות נוירונים באמצעות OPTIMIZERS

בדרך כלל, אימון באמצעות *SGD* יעבוד טוב. עם זאת, כדי להשיג תוצאות טובות הרבה פעמים צריך לעשות *hand-tuning* ל-*learning rate*.

עבור רשתות ומודלים מורכבים יותר, כדאי לשקול שימוש בשיטות מתוחכמות יותר של אופטימייזרים אדפטיביים (*adaptive optimizers* – אדפטיביים במובן שהם עוקבים על השיפוע שנוצר עבור כל פרמטר בנפרד, ומחליטים איך להתאים את המשקולות בהתאם), שנותנים *learning rate* דיפרנציאלי פר פרמטר. לדוגמא – *Adagrad, RMSprop, Adam, SparseAdam* ועוד (שווה לנסות להתחיל מ-*Adam* בדרך כלל).

בחירת LEARNING RATE

אפשר לבחור *learning rate* קבוע (למשל, אפשר להתחיל בסביבות $lr = 001$) – במקרה הזה הוא חייב להיות בסדר גודל המתאים, לכן שווה לנסות לשנות אותו בכפולות של 10. אם הוא יותר מידי גדול, המודל עשוי לסטות או לא להתכנס. אם הוא יותר מידי קטן, זמן האימון של המודל עשוי להיות גדול מכדי להיות מסוגל להחזיר תוצאות בזמן רלוונטי.

לכן, לרוב ניתן להשיג תוצאות טובות יותר באמצעות זה שנאפשר ל-*learning rate* לדעוך לאורך האימון. אפשר לעשות את זה ידנית (לחלק את ה-*lr* פעם במספר מוגדר של *epoch*-ים, או באמצעות נוסחה – למשל עבור *epoch t* אפשר להשתמש בנוסחה הבאה - $lr = lr_0 e^{-kt}$. ישנם אופטימייזרים (למשל *Adam*) שדורשים את ה-*lr* בתור היפרפרמטר, אך הוא משמש רק בתור ערך התחלתי שהאופטימייזר מכוון (לכן במקרים האלו נרצה להתחיל עם *lr* יחסית גבוה).

מידול שפה

מידול שפה *Language modeling* (בהקשר של עיבוד שפה טבעית) היא משימה בה חוזים את המילה הבאה במשפט, למשל – "הסטודנטיות פתחו את ____" (המחשבים? הספר? המבחן?). באופן יותר פורמלי, בהנתן רצף מילים $x^{(1)}, x^{(2)}, \dots, x^{(t)}$, המשימה היא לחשב את ההתפלגות של המילה הבאה $x^{(t+1)}$, כלומר לחשב את $P\left(x^{(t+1)} \mid x^{(t)}, \dots, x^{(1)}\right)$ (כאשר $x^{(t+1)}$ יכולה להיות כל מילה מתוך אוצר המילים המסומן $V = \{w_1, \dots, w_{|V|}\}$).

מערכת המבצעת את משימת מידול השפה נקראת **מודל שפה** *Language model*. אפשר לחשוב על מודל שפה בתור מערכת שמשייכת הסתברות לטקסט מסוים. למשל, עבור טקסט מסוים $x^{(1)}, x^{(2)}, \dots, x^{(T)}$, ההסתברות של הטקסט לפי מודל השפה היא:

$$P(\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} \mid \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} \mid \boldsymbol{x}^{(T-1)}, \dots, \boldsymbol{x}^{(1)})$$
$$= \underbrace{\prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} \mid \boldsymbol{x}^{(t-1)}, \dots, \boldsymbol{x}^{(1)})}_{\text{This is what our LM provides}}$$

עכשיו נשאר רק להגדיר איך לומדים את מודל השפה, כלומר איך לומדים את הפונקציה *P*. נציג מספר דרכים לעשות זאת.

מודלי N-GRAM

שיטה שהיתה בשימוש לפני שעברו להשתמש בלמידה עמוקה. n-gram הוא רצף של n מילים (ובהתאם unigram הוא מילה יחידה, bigram הוא צמד מילים, trigram הוא שלישיית מילים, 4-gram הוא רביעיית מילים וכו'). מודל שפה n-gram מניח הנחה מרקובית Markovian assumption על השפה אותה הוא רוצה למדל – כלומר, שהמילה $x^{(t+1)}$ תלויה רק ב-n-1 המילים שקודמות לה.

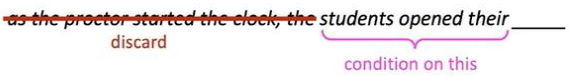
$$P(\boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(t)}, \dots, \boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)} \mid \overbrace{\boldsymbol{x}^{(t)}, \dots, \boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}}) \quad \text{(assumption)}$$

בהנתן ההנחה הזו, מודלי n-gram מוגדרים באופן הבא (שנוצרת באופן ישיר לחלוטין מההגדרה של הסתברות מותנית):

$$\text{prob of a n-gram} \rightarrow \boxed{P(\boldsymbol{x}^{(t+1)}, \boldsymbol{x}^{(t)}, \dots, \boldsymbol{x}^{(t-n+2)})}$$
$$=$$
$$\text{prob of a (n-1)-gram} \rightarrow \boxed{P(\boldsymbol{x}^{(t)}, \dots, \boldsymbol{x}^{(t-n+2)})}$$

כדי לחשב את ההסתברויות האלו, סופרים את כמות הפעמים שכל תת משפט שכזה מופיע בקורפוס גדול כלשהו. כלומר, אנחנו מקבלים קירוב סטטיסטי שמוגדר באופן הבא: $\approx \frac{\text{count}(x^{(t+1)},x^{(t)},\dots,x^{(t-n+2)})}{\text{count}(x^{(t)},\dots,x^{(t-n+2)})}$.

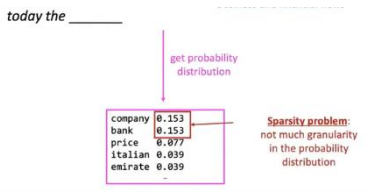
למשל, אם אנחנו רוצים לבנות מודל 4-gram, זה אומר שכדי לחזות את המילה הבאה אנחנו צריכים להסתכל רק על 3 המילים שקדמו לה, אז נוכל "לזרוק" את כל המילים האחרות במשפט.



אנחנו מסתכלים על הקורפוס שבחרנו להשתמש בו עבור יצירת מודל השפה שלנו, ומחשבים לפיו עבור כל מילה w שקיימת באוצר המילים (למשל exams, books..), את ההסתברות הבאה: $P(\boldsymbol{w} \mid \text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$.
למשל, אם בקורפוס "students opened their" הופיע 1,000 פעמים, ו-"students opened their books" הופיע 400 פעמים, אז $P(books|students\ opened\ their) = \frac{400}{1000} = 0.4$, ובהתאם אם *"students opened their exams"* הופיע רק 100 פעמים, הוא יקבל הסתברות נמוכה יותר של 0.1. כאן אפשר לשים לב לחסרון שבשימוש בשיטות שמתבססות על ההנחה המרקובית – השימוש בהנחה המרקובית גרם לכך שהתעלמנו מהחלק הראשון במשפט (שנותן לנו הקשר משמעותי) ובכך נתנו הסתברות נמוכה יותר למילה *exams* למרות שאם מתחשבים בהקשר של תחילת המשפט היא ככה"נ יותר סבירה.

למודלי *n-gram* יש **בעיית דלילות *sparsity*** שנובעת מכמה סיבות:

- בעיה ראשונה – למשל, במקרה בו students opened their $\boldsymbol{w}$ עבור w כלשהי (המונה) לא הופיע בקורפוס בכלל, נקבל כי $\text{count}(\text{students opened their } \boldsymbol{w}) = 0$ ובהתאם $P(\boldsymbol{w} \mid \text{students opened their}) = 0$ כלומר נקבל שההסתברות שהמילה הבאה היא w היא 0 (למרות שיכול להיות שזאת מילה אפשרית, שפשוט לא הופיעה בקורפוס ממש בהקשר הזהה של בדיוק אותו רצף של אותן n מילים). פתרון חלקי לבעיה הזו היא שימוש בשיטת ***smoothing***, שבה מוסיפים קבוע קטן δ לספירה עבור כל $w \in V$ ובכך לוודא שכל המילים יקבלו הסתברות חיובית ממש.
- בעיה שניה – למשל, במקרה בו *students opened their* (המכנה) לא הופיע בכלל בקורפוס – במקרה הזה המכנה יהיה 0 ולא נוכל לחשב את ההסתברות עבור שום מילה w. פתרון חלקי לבעיה היא שימוש בשיטת *backoff*, בה נבחר לוותר על מילה אחת מתחילת ה-nיה – כלומר לחשב לפי הספירה של *"opened their"* (ואם זה עדיין לא קיים בקורפוס, אז לפי "*their*" וכו').
- בעיה שלישית – אם מדובר ב-*ngram* לא נפוץ במיוחד בקורפוס, נקבל התפלגות עם גרנולריות מאוד נמוכה, כלומר לא נוכל להבדיל בקלות בין אפשרויות שונות למילה הבאה.



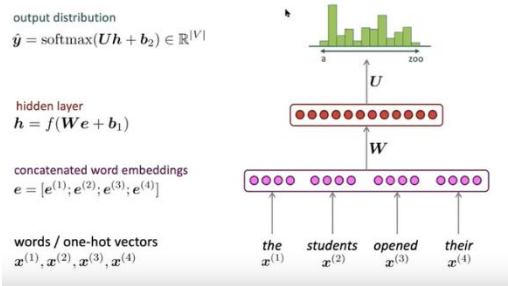
חשוב לשים לב שהגדלה של n רק תחמיר את בעיות הדלילות האלו – כמספר אצבע, לא נוכל להשתמש ב-n שגדול מ-5.

בעיות זכרון ואחסון עבור מודלי *n-gram*: בשיטה זו אנחנו נדרשים לאחסן את הספירות $P(\boldsymbol{w} \mid \text{students opened their})$ לכל n-gram שראינו בקורפוס – זה מצריך המון זכרון (שהרבה ממנו מבוזבז כיוון שהוא משמש לאחסון אפסים).

מודלי n-gram בפרקטיקה – אפשר לאמן מודל trigram פשוט על קורפוס רויטרס (קורפוס בגודל 1.7 מיליון מילים שעוסק בעסקים ופיננסים) תוך כמה שניות על מחשב ממוצע. כעת אפשר להשתמש במודל השפה שיצרנו כדי לייצר טקסט – בכל שלב נחזה את המילה הבאה ע"י לבחור באמעצות המודל את המילה עם ההסתברות הגבוהה ביותר. כנראה שהטקסט שיווצר יהיה יחסית תקין תחבירית, אבל יהיה לא קוהרנטי. כדי ליצור טקסט קוהרנטי ולהצליח למדל את השפה באופן איכותי אנחנו צריכים להסתכל על יותר מ-3 מילים בכל פעם. אבל, כמו שאמרנו קודם, להגדיל את n מחמיר את בעיות הדלילות ומגדיל משמעותית את גודל המודל.

מודלי שפה מבוססי רשתות נוירונים

מודל עם חלון מילים בגודל קבוע: בדומה לרשתות הנוירונים שראינו עבור בעיית NER, אפשר להשתמש ברשתות נוירונים עם חלון בגודל קבוע מראש גם ליצירת מודל שפה (הרעיון הוצע לראשונה ב-2003). מגדירים חלון בגודל מוגדר מראש n של מילים, ומתעלמים מכל המילים במשפט שלא נמצאות בתוך החלון הזה. את המילים שכן נמצאות בתוך החלון מזינים כקלט לרשת נוירונים פשוטה (כלומר, מחשבים word embedding לכל אחת מהן ומעבירים אותן דרך שכבה חבויה ואז דרך שכבת softmax שהופכת את התוצאה לוקטור

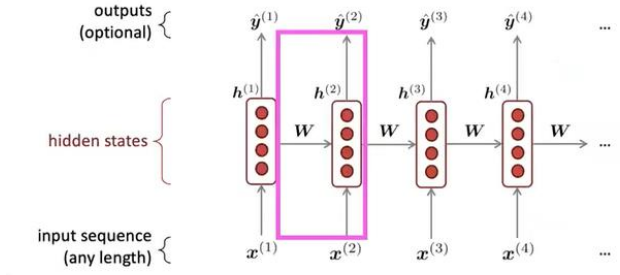


התפלגות על כל אוצר המילים V). כלומר, הפלט של רשת הנוירונים יהיה וקטור התפלגות ממימד |V|, שמייצג מה ההסתברות של כל אחת מהמילים באוצר המילים להיות המילה הבאה במשפט.

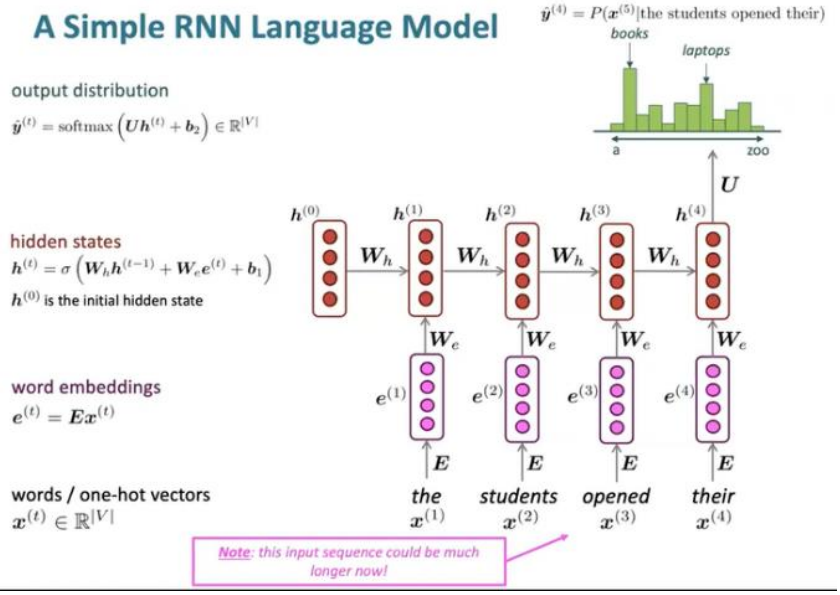
היתרון של מודל כזה בהשוואה למודלי n-gram הוא שהשימוש ב- word embeddings פותר את בעיית הדלילות, וכמו כן לא צריך לאחסן את כל ה-ngramים שראינו בקורפוס. עם זאת, הם סובלים מכמה בעיות משותפות עם מודלי ngram – למשל שגודל החלון עשוי להיות קטן מכדי להצליח לתפוס את כל ההקשר הדרוש, וגם שגודל המודל גדל משמעותית ביחד עם הגדלה של גודל החלון. בנוסף, כיוון שכל מילה מוכפלת במשקולות נפרדות משל עצמה, אין סימטריה באופן שבוא הקלט מעובד (וגם – אין חשיבות לסדר המילים במשפט).

RNN

כדי לפתור את החסרונות של מודל החלון הקבוע, אנחנו רוצים זאת ארכיטקטורה של רשת נוירונים שתוכל לעבד קלט בכל אורך ותייחס משמעות לסדר המילים במשפט. לצורך כך משתמשים ב-Recurrent Neural Network - RNN. זוהי משפחה של ארכיטקטורות של רשתות נוירונים, שהרעיון שלה היא להפעיל את אותו סט משקולות W באופן חזרתי. במודלים האלו, במקום שתהיה לנו שכבה חבויה אחת שכל הקלט עובר דרכה, אנחנו מתחזקים את השכבה החבויה (המכונה בארכיטקטורה זו hidden state) לאורך זמן. כלומר, אנחנו משתמשים בפלט של השכבה החבויה כדי להגדיר את השכבה החבויה הבאה באופן חזרתי (ומכאן השם recurrent). אפשר לחשוב על זה כאילו אנחנו מזינים את השכבה החבויה לתוך עצמה.



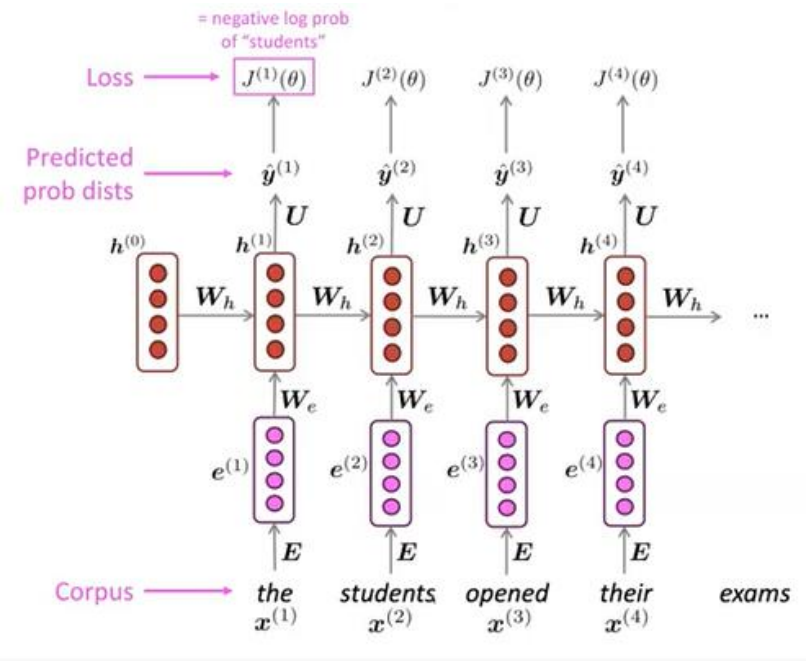
התהליך נראה כך – בהתבססות על המילה הראשונה במשפט $x^{(1)}$ בתור קלט, אנחנו מחשבים את השכבה החבויה הראשונה $h^{(1)}$. השכבה הזו משמשת לשני דברים – שימוש ראשון הוא חישוב הפלט עבור המילה הראשונה במשפט $\hat{y}^{(1)}$, ושימוש שני הוא יצירת השכבה החבויה עבור המילה השניה $h^{(2)}$ (ע"י זה שלוקחים את השכבה הראשונה $h^{(1)}$ ומכפילים אותה בוקטור משקולות W).



כלומר, לכל מילה מחשבים את ה-*word embeddings* שלה, ואז מכניסים כל אחת מהמילים בתור קלט באופן הדרגתי – בהתחלה את המילה הראשונה (שמשמשת ליצור השכבה החבויה הראשונה ובהתאם השכבה החבויה השניה). כשהשכבה החבויה השניה קיימת, משתמשים בה ביחד עם המילה השניה כדי ליצור את השכבה החבויה השלישית, וכך הלאה עד המילה האחרונה במשפט. את התוצאה של המילה האחרונה במשפט מעבירים דרך שכבת *softmax* לקבלת וקטור התפלגות (על כל המילים באוצר המילים) של המילה הבאה במשפט.

היתרון של מודלי שפה מבוססי *RNN* הוא שהם יכולים לעבד משפטים בכל אורך רצוי. כמו כן, החישוב של השלב ה-*t* יכול (בתאוריה) להשתמש במידע ממספר לא מוגבל של מילים אחורה במשפט (אם כי בפרקטיקה, קשה לגשת למידע שמגיע מהרבה שלבים אחורה אלא רק למידע משלבים קרובים). בנוסף, גודל המודל הוא נוסף הוא שאותן משקולות משמשות בכל השלבים, מה שיוצר סימטריה באופן בו אנחנו מתייחסים למילים השונות במשפט הקלט. עם זאת, **החסרון** המשמעותי של מודלי *RNN* הוא שהחישוב החזרתי הוא איטי ויקר.

אימון רשתות RNN



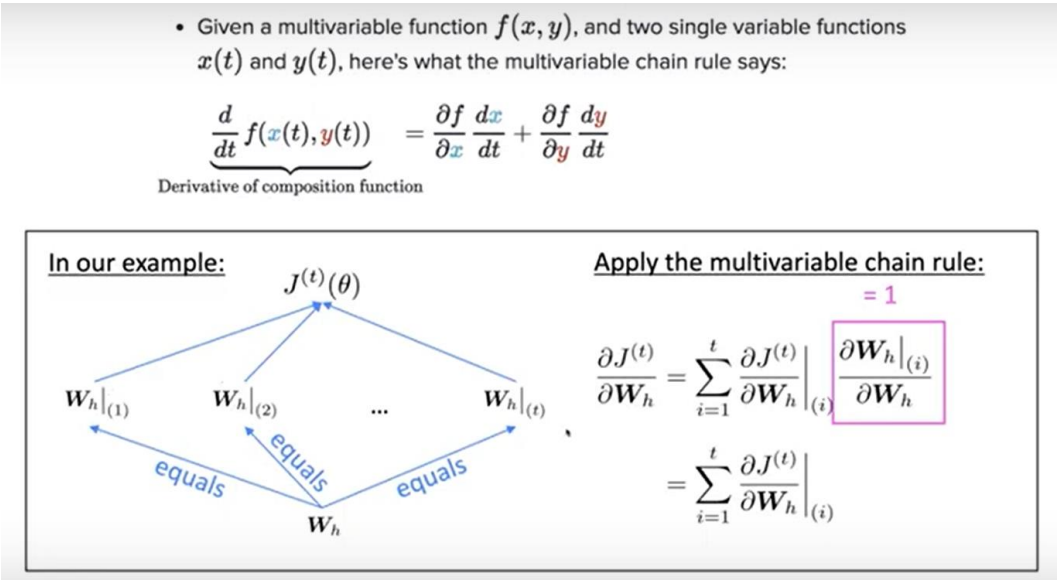
התהליך של אימון רשתות RNN פועל באופן הבא – לכל מילה במשפט (לפי הסדר), קודם נחשב את ה-word embedding שלה, ואז נזין אותו לשכבה החבויה שנוצרה כתוצאה מהשלב הקודם (המילה הקודמת). השכבה החבויה הזו תשתמש אותנו פעמיים – פעם אחת כדי ליצר את השכבה החבויה הבאה (עבור המילה הבאה), ופעם אחת כדי לחשב את התוצאה עבור המילה הנוכחית (וקטור הסתברות של המילה הבאה במשפט). נתייחס לתוצאה עבור אחת מהמילים (למעט האחרונה) בתור Loss שימדוד האם ניתנה התסברות נכונה למילה הבאה במשפט (שקיימת ונתונה ולכן ניתן לבחון האם חזינו אותה בהצלחה) – כלומר ה-Loss יהיה 1 פחות ההסתברות שוקטור ההתפלגות נתן למילה שהיא באמת המילה הבאה במשפט. כך אם הוא נתן לה הסתברות 1 ה-Loss יהיה 0, וככל שההסתברות שהוא נתן לה יותר גדולה ה-Loss יהיה יותר קטן.

כשנסיים את התהליך ונקבל Loss שכזה עבור כל אחת מהמילים במשפט, נחשב את ה-Loss עבור המשפט כולו באמצעות ממוצע פשוט של ה-Loss של כל אחת מהמילים בו - $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta)$.

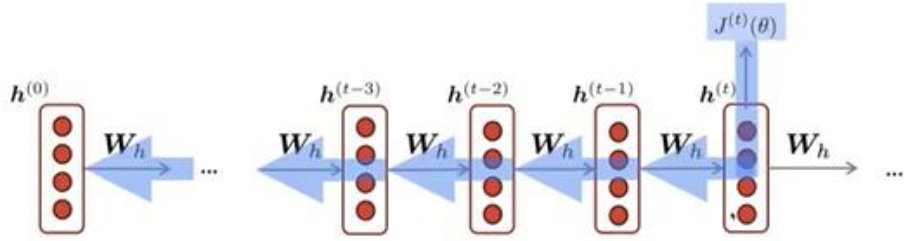
בדרך כלל לא מחשבים את ה-loss והגרדיאנטים על הקורפוס כולו כי זה יותר מידי יקר חישובית. בפרקטיקה, בד"כ מחשבים את זה עבור משפט ספציפי תוך שימוש ב-SGD (שמאפשר לחשב loss וגרדיאנטים על חתיכה קטנה מהדאטא בכל פעם, כלומר לחשב באופן הדרגתי). אז נחשב את $J(\theta)$ עבור משפט (או קבוצת משפטים), נחשב את הגרדיאנטים ובאמצעותם נעדכן את המשקולות – ונחזור על התהליך הזה על כל קבוצות המשפטים בקורפוס עד שנכסה את כולן.

BACKPROPAGATION ברשתות RNN

לגבי תהליך חישוב הגרדיאנט עצמו, נקבל שבגלל שאנחנו משתמשים כל פעם בחישוב שלנו באותה מטריצת משקולות W_h, הגרדיאנט של התוצאה ביחס אליה הוא $\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\Big|_{(i)}$, כלומר – הגרדיאנט ביחס למטריצת המשקולות החוזרת הוא הסכום של הגרדיאנטים ביחס לכל אחת מהפעמים בה היא מופיעה.



פרקטית, המשמעות היא שמתחילים ב-Loss הכולל של המשפט כולו, מחשבים את הגרדיאנט בכל אחד מהשלבים אחורה (בהתחלה ביחס ל-h^t פעם אחת, ואז ביחס ל-W_h עבור כל אחת מהמילים במשפט) וסוכמים את הגרדיאנטים לאורך הדרך.

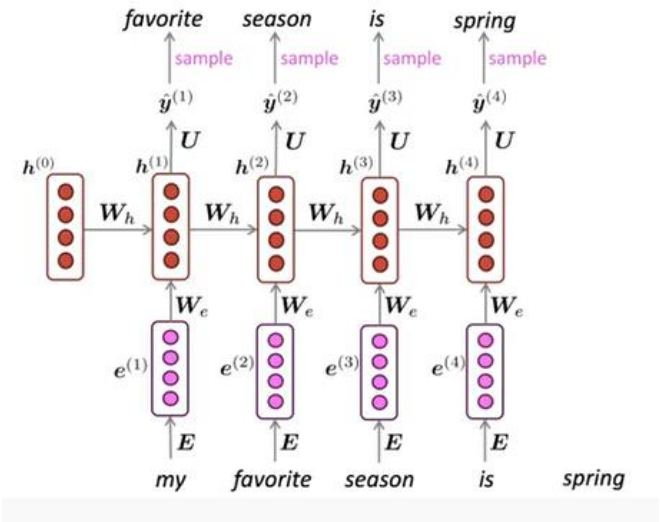


$$\frac{\partial J^{(t)}}{\partial W_h} = \boxed{\sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\Big|_{(i)}}$$

Question: How do we calculate this?

Answer: Backpropagate over timesteps $i=t,\ldots,0$, summing gradients as you go. This algorithm is called **"backpropagation through time"** [Werbos, P.G., 1988, *Neural Networks* **1**, and others]

מודלי שפה מבוססי RNN



כעת כשלמדנו איך מאמנים מודלי RNN, אפשר (בדומה למודלי שפה מבוססי n-gram) להשתמש במודל RNN ליצירת מודל שפה שמאפשר לייצר טקסט באמצעות דגימה חוזרת (כלומר, מתחילים ממילה התחלתית כלשהי ובכל פעם חוזים את המילה הבאה אחריה – לכן צריך בהכרח לספק מילה התחלתית, אם לא רוצים לעשות את זה אפשר לייצר מראש token מיוחד שמייצג תחילת משפט, להוסיף אותו לכל משפט בקורפוס לפני שלב האימון ואז אפשר יהיה להתחיל ממנו תמיד בלי לספק מילה אמיתית ראשונה למשפט בעצמנו). הפלט של המודל יהיה המילה הבאה במשפט, שאז תשמש בתור הקלט לשלב הבא במודל. כלומר, מתחילים ממילה התחלתית כלשהי, ובכל פעם חוזים את המילה הבאה, ואז משתמשים בה כקלט עבור השלב הבא במודל (שישמש כדי לחזות את המילה שאחריה, וכך הלאה עד לסוף המשפט / הטקסט, שיזוהה באמצעות זה שהמודל יחזה token מיוחד שיצרנו מראש שמייצג סוף משפט).

אוולויואציה של מודלי שפה

הדרך הנהוגה להעריך את הביצועים של מודלי שפה היא באמצעות **perplexity**, מדד שמוגדר באופן הבא:

$$\text{perplexity} = \underbrace{\prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)}_{\text{Inverse probability of corpus, according to Language Model}}^{1/T}$$

Normalized by number of words

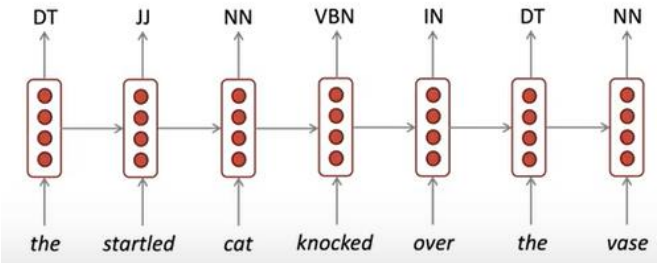
כלומר, בהנתן רצף אמיתי של t מילים, מחשבים עבור כל מילה במיקום ה-t במשפט מה ההסתברות שהמודל נותן למילה ה-t+1 במשפט, ולוקחים את ההופכי של ההסתברות הזו. אחרי שמחשבים את זה עבור כל אחת מהמילים, ממצעים לפי כמות המילים במשפט. המדד הזה למעשה שקול לאקספוננט של ה-Cross entropy loss המסומן J(θ), כלומר

$\prod_{t=1}^{T}\left(\frac{1}{\hat{y}_{x_{t+1}}^{(t)}}\right)^{1/T} = \exp\left(\frac{1}{T}\sum_{t=1}^{T} -\log \hat{y}_{x_{t+1}}^{(t)}\right) = \exp\left(J(\theta)\right)$. המטרה היא להצליח להשיג ערך perplexity **נמוך ככל האפשר.**

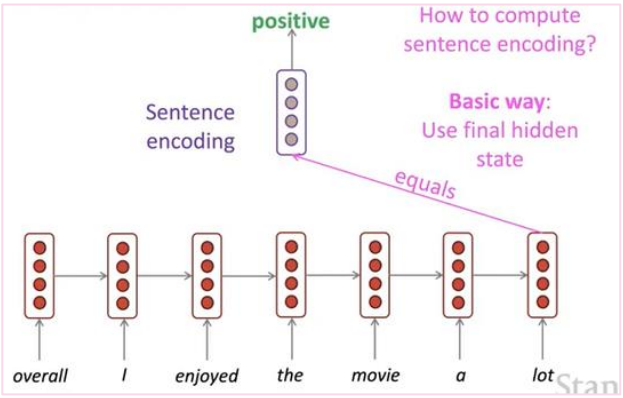
משימות נוספות שמשתמשים בהן ב-RNN

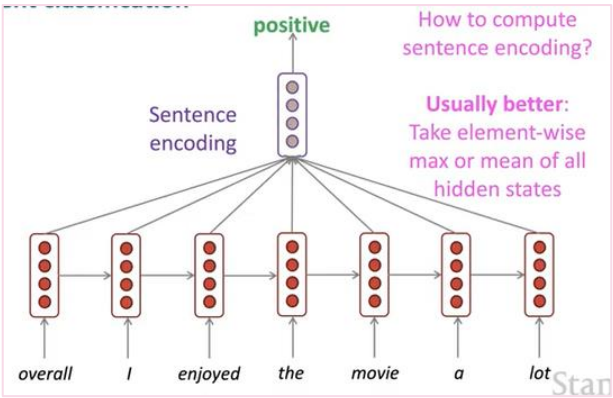
רשתות RNN משמשות לפתרון בעיות נוספות בעיבוד שפה טבעית, למשל:

זיהוי חלק דיבר – POS Tagging: מנסים לחזות עבור כל מילה את התגית שמייצגת את חלק הדיבר שלה, עושים את זה עם RNN בכך שמתייחסים לזה בתור בעיית חיזוי סדרתית:



זיהוי סנטימנט – sentiment classification: כדי לסווג את הסנטימנט שהמשפט כולו מייצג, אנחנו רוצים ליצור סוג של קידוד שמייצג את המשפט כולו. אפשר להשתמש לכך ב-RNN. השיטה הראשונה והבסיסית היא להתייחס לוקטור של השכבה החבויה האחרונה בתור וקטור המייצג את המשפט כולו (שכן הוא מכיל מידע על כל המילים הקודמות למילה האחרונה במשפט). נעשה שימוש בשיטה הזו, אבל מסתבר שפתרון טוב יותר יהיה להשתמש ב-element-wise max / mean על כל השכבות החבויות.

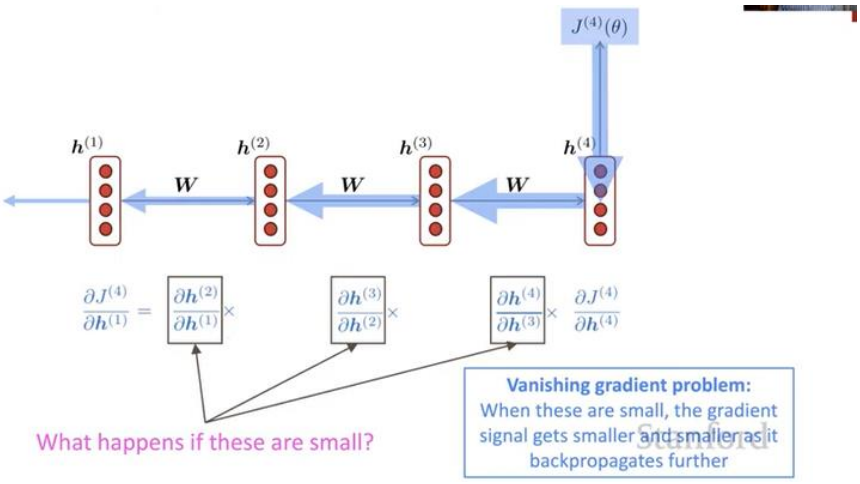




EXPLODING & VANISHING GRDIENTS

VANISHING GRADIENTS

הבעיה בחישוב ה-backpropagation ב-RNN על משפטים מאוד גדולים, היא שכל אחד מהסב-גרדיאנטים לאורך הדרך יכול להפוך להיות מאוד קטן. אם הם מאוד קטנים, הסיגנל של הגרדיאנט קטן ככל שאנחנו הולכים יותר אחורה בשרשרת. זה יגרום לזה שכמעט ולא יהיה לנו upstream gradient ולא נשנה את הפרמטרים כמעט בכלל (אם בכלל).



הסבר קצר ("הוכחה בנפנוף ידיים") ללמה זה קורה –

Vanishing gradient proof sketch (linear case)

- Recall: $$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)$$
- What if σ were the identity function, $\sigma(x) = x$?

$$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} = \operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right) \boldsymbol{W}_h \qquad \text{(chain rule)}$$
$$= \boldsymbol{I}\, \boldsymbol{W}_h = \boldsymbol{W}_h$$

- Consider the gradient of the loss $J^{(i)}(\theta)$ on step i, with respect to the hidden state $\boldsymbol{h}^{(j)}$ on some previous step j. *Let* $\ell = i - j$

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \qquad \text{(chain rule)}$$
$$= \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \boldsymbol{W}_h = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boxed{\boldsymbol{W}_h^{\ell}} \qquad \text{(value of } \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}\text{)}$$

If W_h is "small", then this term gets exponentially problematic as ℓ becomes large

- What's wrong with $\boldsymbol{W}_h^{\ell}$?
- Consider if the eigenvalues of $\boldsymbol{W}_h$ are $\underbrace{\text{all less than 1}}_{\text{sufficient but not necessary}}$:

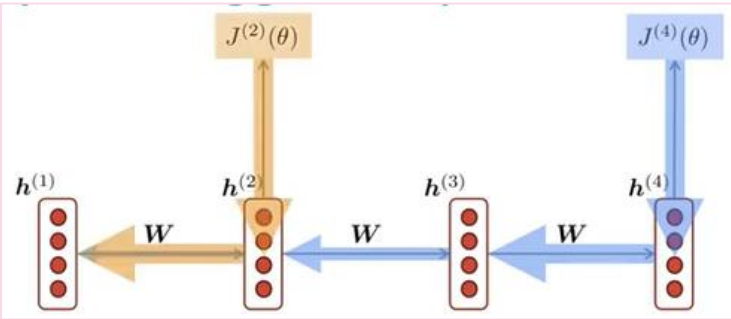
$$\lambda_1, \lambda_2, \ldots, \lambda_n < 1$$
$$\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_n \text{ (eigenvectors)}$$

- We can write $\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boldsymbol{W}_h^{\ell}$ using the eigenvectors of $\boldsymbol{W}_h$ as a basis:

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boldsymbol{W}_h^{\ell} = \sum_{i=1}^{n} c_i \boxed{\lambda_i^{\ell}} \boldsymbol{q}_i \approx \boldsymbol{0} \text{ (for large } \ell\text{)}$$

Approaches 0 as ℓ grows, so gradient vanishes

- What about nonlinear activations σ (i.e., what we use?)
  - Pretty much the same thing, except the proof requires $\lambda_i < \gamma$ for some γ dependent on dimensionality and σ



למה זאת בעיה שהגרדיאנטים נעלמים? אם אנחנו רוצים להגדיר את הייצוג עבור h^1 (השכבה החבויה הראשונה), כנראה שהמילים שממש בסוף המשפט (וממש רחוקות ממנה) מחזיקות גם הן מידע רלוונטי על המילה הראשונה (זה אחד היתרונות בשימוש בRNNים על פני שימוש בשיטות עם חלון בגודל קבוע). לכן הבעיה בזה שהגרדיאנטים נעלמים היא שרוב ההשפעה של המילים האחרונות במשפט (בכחול בשרטוט) נדרסת ע"י המילים שקרובות למילה הראשונה במשפט (בכתום בשרטוט). כלומר, המשקולות ייצגו השפעות קצרות טווח (מהמילים הקרובות) ולא ארוכות טווח (מהמשפט כולו).

איך פותרים את בעיית הגרדיאנטים הנעלמים? כאמור, הבעיה העיקרית היא שגרדיאנטים נעלמים גורמים לכך של-RNN קשה ללמוד ולשמר אינפורמציה לאורך הרבה צעדים. ב-RNN פשוט ("vanilla"), המצבים החבויים כל הזמן נכתבים מחדש (באמצעות כפל), נביט בנוסחת עדכון שכבה חבויה - $h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_x x^{(t)} + b\right)$. הרעיון הוא שאנחנו רוצים להיות מסוגלים לשמר יותר "זכרון" לאורך הרשת, כלומר ליצור RNN שיש לה זכרונות נפרדים – כך פותחו רשתות LSTM (בהמשך הפרק).

EXPLODING GRADIENTS

$$\theta^{new} = \theta^{old} - \overbrace{\alpha}^{\text{learning rate}} \underbrace{\nabla_\theta J(\theta)}_{\text{gradient}}$$

בעיה הפוכה לבעיית הגרדיאנטים הנעלמים – אם הגרדיאנט נהיה יותר מידי גדול, אז עדכון הצעד של אלגוריתם ה-SGD נהיה יותר מידי גדול - (כלומר עצום באופן שרירותי).

יכולות להיות לזה השלכות בעייתיות – אם אנחנו לוקחים צעד יותר מידי גדול אנחנו יכולים להגיע לקונפיגורצית פרמטרים מוזרה וגרועה (עם loss גדול) – האלגוריתם חושב שהוא מצא גבעה לטפס עליה, אבל בעצם הוא בנקודה כל כך לא רלוונטית שהיא חסרת משמעות ורחוקה מאוד מהגבעה שחיפשת. במקרה הכי גרוע, זה יכול להוביל לרשת שמכילה ערכי inf או NaN (ואז צריך להתחיל לאמן את הרשת מחדש מנקודה מוקדמת יותר).

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

הפתרון לבעיה הזו הוא שימוש באלגוריתם **Gradient clipping** – בוחרים מספר סביר כלשהו (מספר נפוץ שבוחרים בו הוא 20), ומגדירים שעבור גרדיאנטים שהנורמה שלהם עוברת את המספר הזה, עושים להם scale down לפני שמשתמשים בהם בעדכון ה-SGD. האינטואיציה היא שאנחנו עדיין לוקחים צעד באותו כיוון של הגרדיאנט – פשוט צעד קטן יותר. בפרקטיקה, כל מה שזה אומר זה שחשוב לזכור להשתמש ב-gradient clipping, בגלל שהוא הופך את בעיית הגרדיאנטים המתפוצצים לקלה לפתרון.

פתרונות לגרדיאנטים מתפוצצים ונעלמים בארכיטקטורות שונות

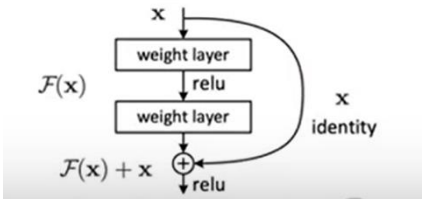


Figure 2. Residual learning: a building block.

הבעיה של גרדיאנטים נעלמים היא לא בעיה שקורית רק ב-RNN, אלא בכל מיני סוגים של ארכיטקטורות של רשתות נוירונים (כולל רשתות feed-forward, CNN ובמיוחד בארכיטקטורות עמוקות). זה יכול לגרום לכך שהשכבות הנמוכות נלמדות מאוד לאט, כלומר המודל מאוד קשה לאימון. אחד הפתרונות שנמצאו לבעיה הוא שהרבה ארכיטקטורות חדשות של רשתות feed-forward או CNN משתמשות ביותר קישורים ישירים (שמאפשרים לגרדיאנטים "לזרום"), למשל ResNet (ארכיטקטורת CNN מעולם ה-vision) שבה יש מסלול ישיר שעוקף את כל הרשת ומאפשר לקלט להגיע ישירות לשכבה הלא לינארית האחרונה (בציור משמאל).

שיטות נוספות הן למשל קישורים דחוסים כמו למשל ב-DenseNet (שבה מחברים באופן ישיר כל שכבה לכל אחת מהשכבות שבאות אחריה), או קישורי אוטוסטרדה כמו HighwayNet שמזכירה את ResNet רק שיש שער ששולט על החיבור הישיר (זאת ארכיטקטורה שמקבלת השראה מ-LSTM ומתאימה את הרעיון של השערים לרשתות feed forward / CNN).

לסיכום, למרות שבעיות הגרדיאנטים הנעלמים והמתפוצצים הן בעיות כלליות שקורות בהרבה ארכיטקטורות של רשתות נוירונים, RNN היא ארכיטקטורה שלוקה בהם במיוחד (זה קורה בכלל הכפל החוזר של אותה מטריצת משקולות).

LSTM

רשתות LSTM – Long Short-Term Memory RNNs פותחו בסוף שנות ה-90, ומטרתן לאפשר ליצור רשתות RNN עם זכרון שאפשר לעדכן באופן יותר גמיש כדי להיות מסוגלים לאחסן מידע לאורך זמן (מחלקים שונים של המשפט, מבלי שהוא ידרס ע"י חלקים אחרים בהמשך האלגוריתם). הארכיטקטורה של LSTM גורמת לכך של-RNN יהיה מאוד קל לשמר מידע לאורך הרבה צעדים (כמספר אצבע, רשתות LSTM מסוגלות ליצור קידוד של מידע שמתחשב באפילו 100 צעדים אחורה). רשתות LSTM לא מבטיחות למנוע לחלוטין את בעיות הגרדיאנטים הנעלמים או המתפוצצים, אבל הן מספקות דרך קלה יותר למודל ללמוד תלויות לאורך זמן ולכן פחות נוטות לסבול מהבעיות האלו. בקווים כלליים, בימינו אין סיבה להשתמש ב-RNN קלאסי, ותמיד נרצה להשתמש ב-LSTM בכל מקום בו היינו רוצים להשתמש ב-RNN.

ב-2013-2015, התחילו להשתמש ברשתות LSTM כדי להחזיר תוצאות ברמת state of the art במשימות כמו זיהוי כתב יד, זיהוי דיבור, תרגום, פרסור, מתן כותרות לתמונות וכן ליצירת מודלי שפה. LSTM הפכה בתקופה הזו לגישה המובילה לרוב משימות ה-NLP. כיום, יש גישות אחרות שמובילות (למשל טרנספורמרים) עבור רוב המשימות (אם מסתכלים על מאמרים שהוגשו לתחרויות בתחום, אפשר לראות סביב 2019 שהתחילו לעשות הרבה יותר שימוש בטרנספורמרים על פני RNN).

מבנה רשתות LSTM

הרעיון הוא לבנות רשת כך שבכל שלב t, יש **מצב חבוי hidden state** המסומן $h^{(t)}$ אבל גם **cell state** המסומן $c^{(t)}$ (השמות עשויים להיות קצת מבלבלים, כי בהרבה מובנים דווקא ה-cell state יותר מזכיר את ה-hidden state ברשתות RNN הפשוטות). שני המצבים האלו הם וקטורים באורך n. התא cell מאחסן מידע ארוך-טווח שאפשר לקרוא, למחוק ולכתוב אותו (ובעצם מזכיר קצת זכרון RAM).

החלק המעניין ב-LSTM הוא ההחלטה איזה מידע למחוק / לכתוב / לקרוא מהתאים – זה נשלט ע"י שלושה **שערים gates** הסתברותיים, שמייצגים מחיקה, כתיבה וקריאה בהתאמה. השערים גם הם וקטורים מאורך n, כאשר בכל שלב כל אלמנט בשערים יכול להיות פתוח (1), סגור (0) או ערך כלשהו ביניהם שמייצג הסתברות (נגיד אם מדובר בשער מחיקה, אז מה ההסתברות שצריך למחוק אלמנט מסוים בסקלה שבין "בטוח למחוק את הכל" לבין "בטוח לא למחוק כלום").

פורמלית, יש לנו סדרת קלטים $x^{(t)}$ ואנחנו רוצים לחשב סדרה של מצבים חבויים $h^{(t)}$ ומצבי תא $c^{(t)}$. לכן עבור נקודת זמן t כלשהי, נתחיל מלחשב את הערכים עבור כל אחד משלושת השערים:

- **שער forget** (מחיקה) המסומן $f^{(t)}$ – קובע איזה מידע נשמר ואיזה מידע נמחק מה-cell state הקודם.
- **שער input** (קריאה) המסומן $i^{(t)}$ – קובע איזה חלקים מהחישובים שנבצע עבור התא הנוכחי באמת יכתבו לזכרון שלו
- **שער output** (כתיבה) המסומן $o^{(t)}$ – קובע איזה חלקים מהתא יועברו למצב החבוי

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f\boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

העדכון מתבצע ע"י אותה נוסחה שמשמשת לעדכון מצבים חבויים ברשת RNN קלאסית - $i^{(t)} = \sigma\left(\boldsymbol{W}_i\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i\boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$

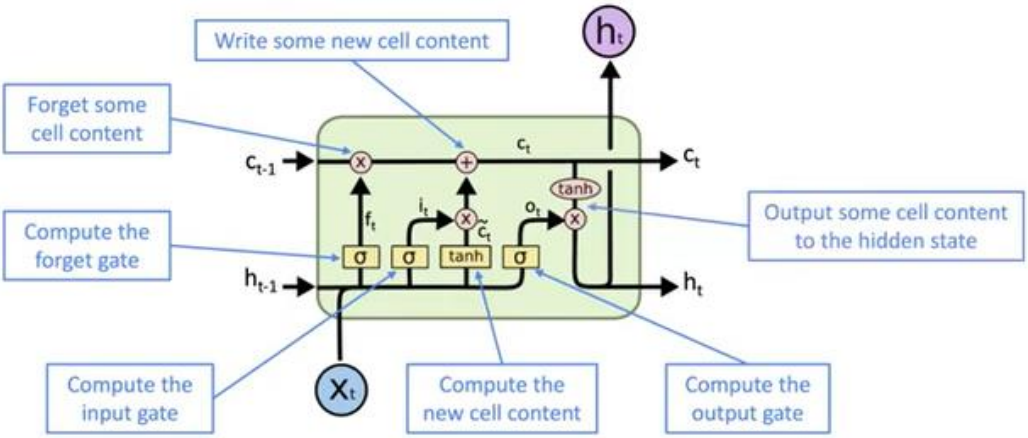
$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o\boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

שמועברת דרך סיגמואיד כדי להפוך את זה להסתברות בין 0 ל-1. חשוב לשים לב שמטריצות המשקולות W הן שלוש מטריצות שונות – כלומר קיימת מטריצה יעודית עבור כל אחד מהשערים.

לאחר שחישבנו את הערכים עבור השערים, אנחנו מחשבים שלושה ערכים:

- **יצירת התוכן המלא של התא החדש cell content**: התוכן שיכתב לתא עבור השכבה הנוכחית. מחושב ע"י הנוסחא הבאה (גם כאן מדובר באותה נוסחא של ה-RNN הקלאסית, רק עם מטריצת משקולות יעודית, ומעבירים את התוצאה דרך tanh שמתאזנת סביב הערך אפס) - $\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c\boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$
- **עדכון התא החדש באמצעות התוכן והשערים שחישבנו cell state:** לוקחים את התא הקודם כפול וקטור ה-forget (כדי לקבל רק את התאים שרצינו לשמור מתוכו – כלומר מוחקים מהשלב הקודם חלקים מסוימים) ומוסיפים לו את התוכן שחישבנו כפול וקטור ה-input (כדי לקחת מתוכו רק את החלקים שהחלטנו שאנחנו רוצים להשתמש בהם) - $c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$. כלומר, השערים נמצאים בשימוש element-wise בזכות השימוש במכפלה פנימית.
- **מצב חבוי hidden state**: מעבירים את התא שיצרנו בשלב הקודם דרך tanh, ואת התוצאה כופלים בוקטור ה-output כדי לכתוב למצב החבוי רק את החלקים הרצויים - $\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh\ \boldsymbol{c}^{(t)}$.

נביט בשרטוט הגרפי הבא שמדגים את מבנה ה-LSTM (שכבה בודדת t):

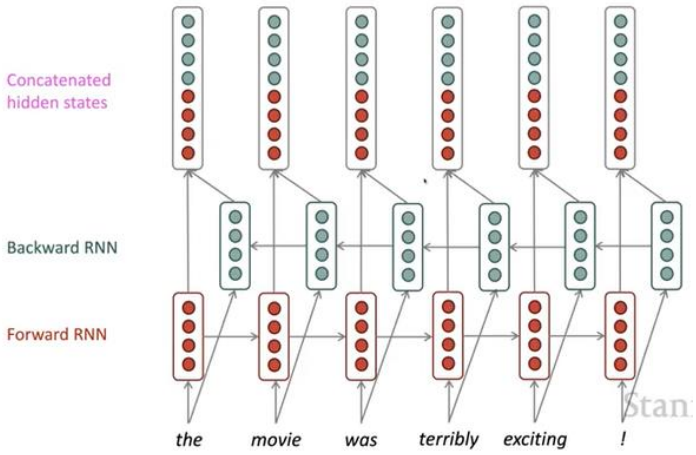


מקבלים מהשכבה הקודמת את h^{t-1}, c^{t-1} (המצב החבוי והתא של השכבה הקודמת), ובנוסף מקבלים את הקלט החדש של השכבה הנוכחית – המילה x_t. בהתחלה מחשבים את שלושת השערים תוך התבססות על המצב החבוי הקודם ומילת הקלט החדשה. משתמשים בהם לפי הנוסחאות. מה שחשוב לשים לב אליו זה שהתא הקודם הופך להיות התא הנוכחי בלי שמפעילים עליו יותר מידי מניפולציות (רק ה-forget מוחק ממנו קצת דברים וה-input מוסיף לו קצת דברים שמגיעים מהמצב החבוי הקודם וממילת הקלט החדשה). העובדה שההוספה של המידע החדש היא אכן **הוספה** (שמתבצעת באמצעות חיבור ולא באמצעות כפל), גורמת לזה שאנחנו יותר מסוגלים לשמור מידע לאורך זמן כי הוא לא נדרס ומשתנה באמצעות הכפל בכל שלב, אלא (אולי) רק נוסף לו מידע חדש.

איך הארכיטקטורה מאפשרת ל-LSTM לזכור מידע לטווח ארוך? אם למשל שער ה-forget יהיה 1 ושער ה-input יהיה 0, זה יגרום לכך שהמידע מהתא אליו אנחנו מתייחסים ישמר לנצח. לכן אפשרות אחת היא לאתחל את וקטור ה-forget בתור וקטור אחדות (כלומר להגיד "תשמור את כל המידע מהתא הקודם תמיד") ואז הרשת תלמד בעצמה מתי הולם לשכוח דברים.

BI-DIRECTIONAL AND MULTI-LAYER RNNS

RNN דו כיווני



יש משימות בהן יש משמעות גדולה לזיהוי של ההקשר של המילה. אמנם רשתות RNN מאפשרות לשמור הקשר לאורך הרבה צעדים, אבל מדובר בהקשר שהוא רק בכיוון אחד – כל שכבה מייצגת מידע משכבות קודמות (כלומר ממילים קודמות במשפט), ולכן לא מתחשבת בהקשר של מילים שמופיעות אחרי המילה. הפתרון לכך (במשימות בהן זה נחוץ) הוא לעשות רשת RNN דו כיוונית (הכוונה ב-RNN היא כל סוג של RNN, למשל זה יכול להיות גם LSTM), שמורכבת מרשת אחת שרצה מתחילת המשפט עד לסופו, ואחת שרצה מסוף המשפט עד לתחילתו, ואז בסוף לקחת את הייצוג שיוצא מכל אחד מהכיוונים ולחבר אותם לכדי וקטור אחוד שמייצג את ההקשר של המשפט בשני הכיוונים.

איך זה עובד? לכל שלב t נחשב בשני הכיוונים את הייצוג של המצב החבוי h^t, ואז נדביק את שני הייצוגים לכדי וקטור אחוד שמייצג את המצב החבוי עבור המודל הדו כיווני -

$$\text{Forward RNN} \quad \overrightarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{FW}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$$

$$\text{Backward RNN} \quad \overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$$

$$\text{Concatenated hidden states} \quad \boldsymbol{h}^{(t)} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$$

חשוב לשים לב שרשתות RNN דיו כיווניות הן ישימות רק במקרה שבו יש לנו גישה למשפט הקלט במלואו – ולכן לא מתאימות לבעיות כמו מידול ששפה Language modeling (שבהן יש לנו רק את ההקשר אחורנית ולא קדימה). אם אכן יש גישה לרצף המילים כולו, נרצה להשתמש במודל דו כיווני בתור ברירת המחדל כיוון שהדו-כיווניות נותנת ייצוג מאוד עוצמתי ואיכותי (זה נכון לא רק ב-RNN אלא גם במודלים עמוקים וטרנספורמרים, למשל BERT הוא מודל דו כיווני).

תרגום מכונה

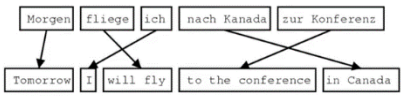
משימת תרגום מכונה Machine Translation היא המשימה בה מתרגמים משפט x משפה כלשהי (שפת המקור source) למשפט y בשפה אחרת (שפת היעד target).

תרגום לפני עידן רשתות הנוירונים

בתחילת שנות ה-50 הוצגו מערכות ראשונות לתרגום – האמריקאים השתמשו בהם כדי לתרגם טקסט מרוסית לאנגלית בזמן המלחמה הקרה (שלא היו מוצלחות במיוחד ורצו על מחשבים מאוד פרימיטיביים). המערכות האלו היו מערכות rule-based שלא הצליחו לקלוט את המורכבות בשפה.

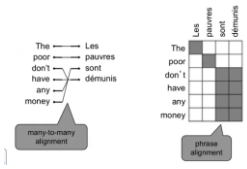
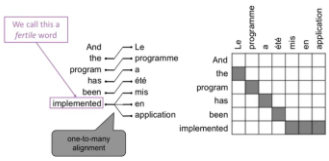
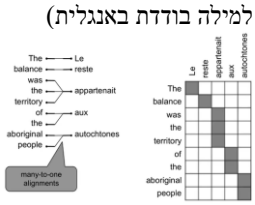
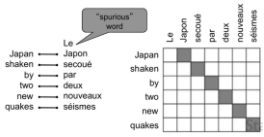
בין 1990 ל-2010 בוצעו נסיונות רבים לביצוע תרגום בשיטות סטטיסטיות. הרעיון המרכזי הוא לבנות מודל הסתברותי מבוסס דאטא, שיוכל לחזות את התרגום הנכון. לדוגמא, אם אנחנו רוצים לתרגם מצרפתית לאנגלית, אנחנו רוצים למצוא את המשפט הכי טוב באנגלית y בהנתן משפט בצרפתית x, כלומר למצוא את $\text{argmax}_y\ P(y \mid x)$, או אם נשתמש בחוק בייס אפשר לפרק את הבעיה לשני מרכיבים אותם אפשר ללמוד בנפרד - $\text{argmax}_y\ P(x \mid y)P(y)$. הביטוי הורוד מייצג מודל תרגום translation model – שממדל איך מילים וביטויים אמורים להיות מתורגמים (fidelity) ולומד מדאטא מקבילי parallel המקשר בין שתי השפות. הביטוי הסגול מייצג מודל שפה language model, שממדל איך להשתמש בצורה נכונה בשפה שמתרגמים אליה (fluency), ולומד מדאטא monolingual (ששייך רק לשפה אליה אנחנו רוצים לתרגם).

איך יוצרים את מודל התרגום $P(x|y)$? מתחילים בלמצוא כמות גדולה של דאטא מקבילי בין שתי השפות (כלומר, זוגות של משפטים שתורגמו אנושית מצרפתית לעברית או הפוך). דוגמאות נפוצות למאגרי מידע מקבילי הם פרוטוקולים פרלמנטריים ממדינות שדוברות יותר משפה אחת (למשל קנדה, הונג קונג ומספר מדינות באירופה).



לאחר מכן, מפרקים את הבעיה ע"י הגדרת המשתנה החבוי latent variable המסומן a שמייצג alignment (כלומר קשר הדדי ברמת המילה הבודדת בין משפט המקור x ומשפט היעד y, כלומר איזו מילה במשפט היעד מייצגת איזה מילה במשפט המקור) באופן הבא - $P(x, a \mid y)$.

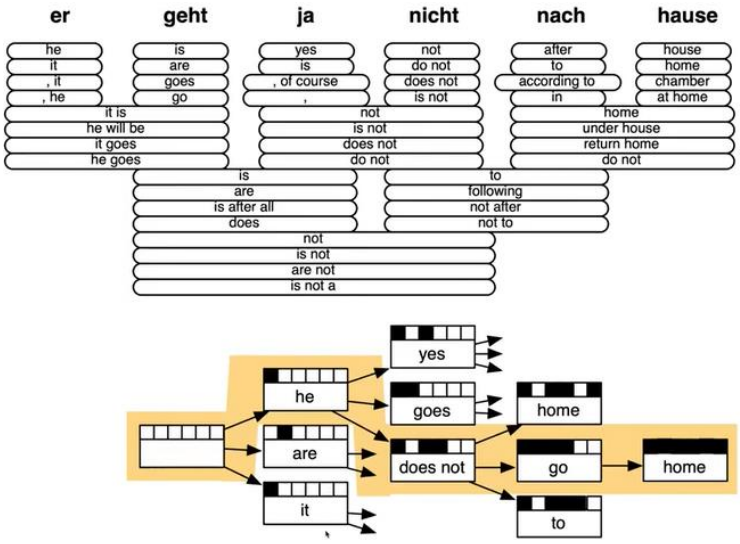
עם זאת, חשוב לזכור כי יש הבדלים טיפולוגיים בין שפות שמובילים לכך שמשימת ה-alignment תהיה מאוד מורכבת עבורן (למשל, סדר המילים מאוד שונה במשפט, או שיש מילים שאין להן תרגום ישיר). יש כל מיני קונפיגורציות alignment שעשויות להווצר:

Many-to-many alignment	One-to-many alignment	Many-to-one alignment	One-to-one alignment
מצבים מורכבים בהם ביטוי מותאם לביטוי 	התאמת מילה בודדת לביטוי 	התאמת ביטוי למילה בודדת – many to one (למשל, ביטוי אחד בצרפתית שיתורגם למילה בודדת באנגלית) 	התאמת מילה מילה 

לומדים את $P(x|y)$ בתור קומבינציה של הרבה משתנים, בין היתר ההסתברות ששתי מילים ספציפיות יתאימו זו לזו aligned, ההסתברות שלשתי מילים ספציפיות יהיה fertility מסוים (שהוא מספר המילים ה-corresponding) ועוד. חשוב לזכור שה-alignments הם משתנים חבויים latent variables, כלומר הם לא מופיעים מפורשות בדאטא, וצריך להשתמש באלגוריתמים לומדים כדי ללמוד את הפרמטרים וההסתברויות שלהם.

איך בוחרים מיהו $argmax_y$? הפתרון הנאיבי יהיה למספר כל משפט y אפשרי ולחשב עבורו את ההסתברות הזו, אבל הפתרון הזה יותר מידי יקר חישובית. לכן, נרצה להשתמש בהנחות אי תלות independence assumptions מאוד חזקות במודל, ולהשתמש בתכנון דינמי כדי למצוא פתרונות אופטימליים (למשל, באמצעות שימוש באלגוריתם Viterbi). התהליך הזה מכונה **decoding**.

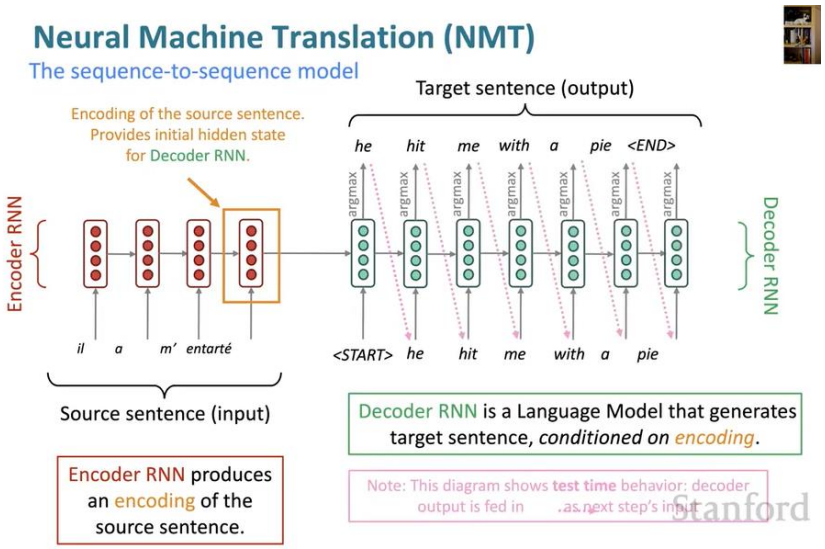
בכל שלב נבחר אופציה אחרת לתרגום של מילה בודדת / ביטוי ונמשיך משם ע"י בחירת התרגום הסביר ביותר בכל שלב:



תרגום מכונה באמצעות רשתות נוירונים

בשנת 2014 בוצע נסיון ראשון לשימוש ברשת נוירונים גדולה כדי לבצע תרגום מכונה (זה אחרי ש-Google translate כבר פורסם, שהשתמש בשיטות סטטיסטיות ולא ברשת נוירונים באותו שלב). באותו זמן זה היה תחום מחקר שוליים, אבל תוך שנתיים זו כבר הפעה לשיטה המובילה ביותר בתחום התרגום – ב-2016 Google עברה להשתמש בתרגום מבוסס רשתות נוירונים (ולא בתרגום סטטיסטי) במודלי התרגום שלה, וב-2018 כל שירותי התרגום וגופי המחקר המובילים (ובכלל כך Microsoft ו-Facebook) עברו גם הם לשיטות האלו. כלומר, מודלי SMT (מודלים סטטיסטיים) שדרשו מאוד מנהדסים והרבה מאוד שנים כדי לייצר אותם, הוחלפו במודלי NMT (מודלים מבוססים רשתות נוירונים) שיכלו להיות מאומנים ע"י קבוצה קטנה של מהנדסים תוך חודשים ספורים בלבד.

Neural Machine Translation (NMT) היא דרך לבצע תרגום מכונה מקצה לקצה באמצעות רשת נוירונים בודדת. רשתות נוירונים מהסוג הזה נקראות sequence-2sequence או seq2seq, והן מורכבות משתי רשתות RNN שאחת מהן בתפקיד encoder (מקודדת את משפט המקור) ואחת מהן בתפקיד decoder (שהיא למעשה מודל שפה), שלוקחת את הייצוג של משפט המקור שהוציא ה-encoder ומייצרת משפט בשפת היעד.



שיטות של seq2seq הן מאוד ורסטיליות, ומשמשות למשימות NLP נוספות כמו סיכום summarization, גנרוט דיאלוגים (האנקודר יקודד את מה שמי שמדברים איתו אמר, והדיקודר ייצר תשובה), פרסינג ועוד.

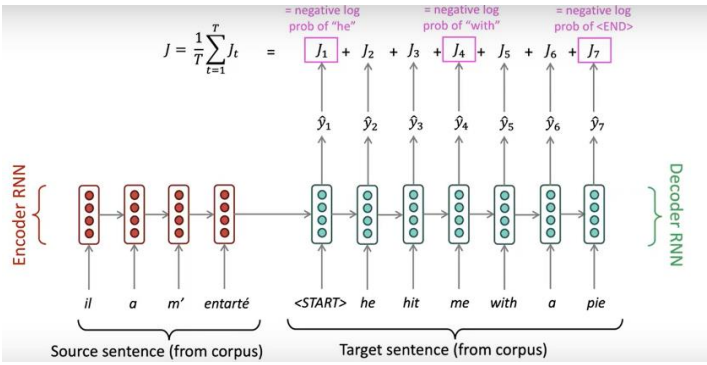
מודלי seq2seq הם דוגמא ל-**conditional language model**, שהם מודלי שפה (במובן שהם חוזים את המילה הבאה במשפט היעד y) שהם מותנים (במובן שהפרדיקציה מותנית גם במשפט הקלט x). למעשה, NMT מחשב את:

$$P(y|x) = P(y_1|x)\,P(y_2|y_1,x)\,P(y_3|y_1,y_2,x)\ldots \underbrace{P(y_T|y_1,\ldots,y_{T-1},x)}_{\text{Probability of next target word, given target words so far and source sentence } x}$$

אנחנו צריכים שתהיה לנו דרך להעביר מידע על משפט המקור מהאנקודר לדיקודר (כדי שהוא יוכל להשתמש בו כדי לחזות). בד"כ מה שעושים זה מזינים לדיקודר hidden state של האנקודר בתור קלט.

אימון מודלי NMT

בתור התחלה, משיגים קורפוס מקבילי כלשהו. כדי לאמן את המודל, ניקח בכל פעם משפט מקור מהקורפוס, נקודד אותו באמצעות האנקודר, ואז נשתמש במצב החבוי האחרון שלו בתור קלט לדיקודר, שאותו נאמן מילה-מילה ע"י להשוות בכל פעם את המילה שהוא חוזה בתור מילת התרגום המתאימה ביותר, עם המילה מתוך התרגום האמיתי מהקורפוס המקבילי. אנחנו עושים backpropagate אחורה לכל המודל (עד להתחלה), כלומר אנחנו לא רק מעדכנים את הדיקודר אלא גם מעדכנים את הפרמטרים של האנקודר.



יתרונות וחסרונות של NMT

היתרונות בשימוש ב-NMT (בהשוואה לשימוש ב-SMT – תרגום מכונה סטטיסטי) הם שהם נותנים ביצועים יותר טובים – גם במובן שהתוצר שהם מפיקים מרגיש יותר שוטף וקרוב לשפה אמיתית, וגם במובן שהם משתמשים יותר טוב בהקשר context. בנוסף, מודלים מבוססים רשתות נוירונים משתמשים טוב יותר ב-phrase similarities (כלומר ביטויים שאומרים פחות או יותר אותו דבר). יתרון נוסף הוא שרשת נוירונים בודדת יכולה להיות optimized end-to-end, כלומר אין תתי קומפוננטות שצריכות להיות optimized כל אחת בנפרד. בנוסף, מודלי NMT מצריכים הרבה פחות מאמץ הנדסי – לא צריך לעשות feature engineering, ומשתמשים באותה השיטה בדיוק לכל הצמדים האפשריים של שפות שרוצים לתרגם ביניהן.

החסרונות בשימוש ב-NMT הוא שהם פחות interpretable (כלומר קשה לקבוע למה הם תרגמו כפי שתרגמו) ולכן קשים לדיבוג. כמו כן, הם קשים לשליטה – למשל, אי אפשר לכתוב להם חוקים מפורשים או הנחיות לתרגום.

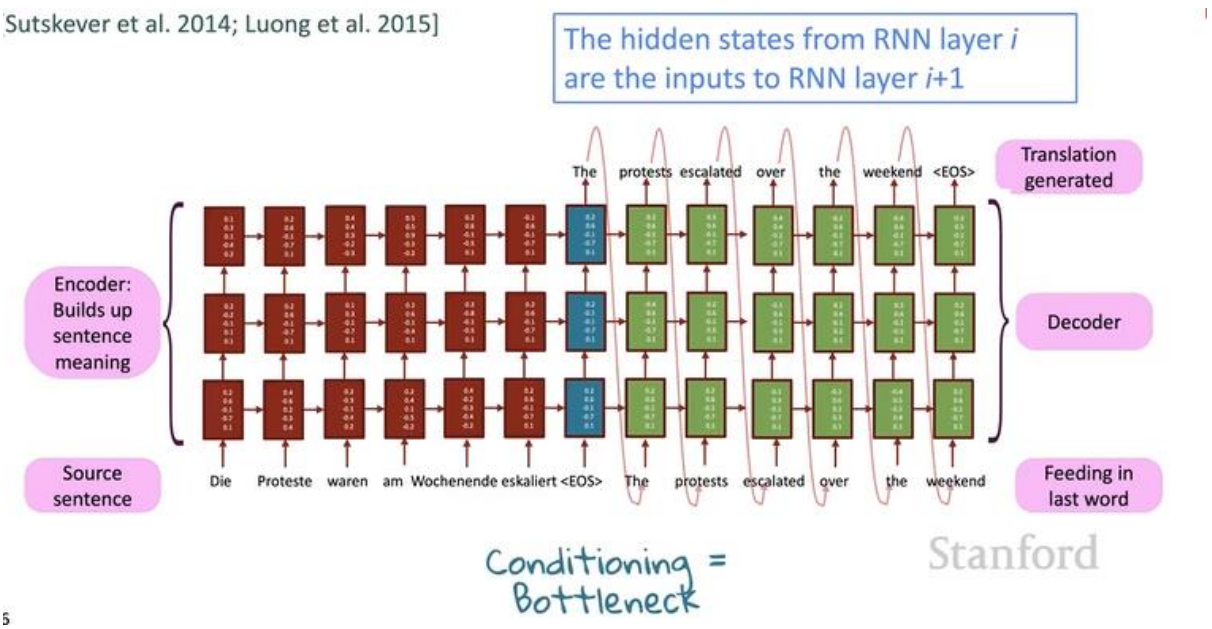
אוולויואציה של מודלי תרגום מכונה

הדרך הכי טובה לבצע אוולויואציה למודלי תרגום מכונה היא להשוות את הביצועים שלהם למול תרגום אנושי שנוצר ע"י דובר שתי השפות שהמודל מתרגם ביניהן. אבל הפתרון הזה הוא מאוד יקר לביצוע ולא תמיד אפשרי (למשל אם לא מצאתם דובר של שתי השפות הרצויות). לכן, נחוצות שיטות אוטומטיות לביצוע אוולויואציה למודלי תרגום מכונה. שיטה נפוצה היא BLEU – **Bi**lingual **E**valuation **U**nderstudy, בה משווים את התרגום שיצר המודל למספר תרגומים אנושיים, תוך התבססות על ציון הדמיון similarity score ביניהם, המבוסס על n-gram precision (בד"כ עם n בין 1 ל-4) – חישוב החפיפה בין ה-n-gram-ים, ועל עונש penalty על תרגומי מכונה שהם קצרים מידי. השיטה הזו היא שימושית אך יש לה חסרונות – למשל, יש הרבה דרכים תקינות לתרגם את אותו משפט, ולכן תרגום טוב שיצרה המכונה עלול לקבל ערך BLEU נמוך אם הוא שונה מהתרגומים האנושיים. BLEU מספק ציון בין 0 ל-100, כאשר ציון 100 מייצג התאמה של אחד לאחד בין תרגום המכונה לאחד התרגומים האנושיים.

מודלי MULTI-LAYER RNN

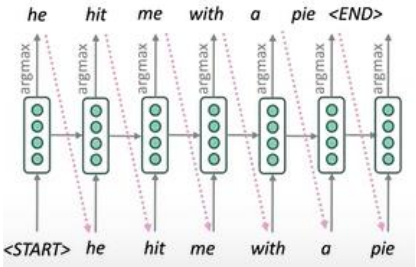
מודלי RNN הם כבר "עמוקים" במימד אחד (במובן שהם עוברים מספר שכבות שכל אחת מהן מייצגת timestamp), אבל אנחנו יכולים לגרום להם להיות "עמוקים" במימד נוסף ע"י הוספה של מספר רשתות RNN שונות – זה נקרא multi-layer RNN או stacked RNNs. זה מאפשר לרשת לחשב ייצוגים הרבה יותר מורכבים. ה-RNNים בשכבות הנמוכות יחשבו פיצ'רים low-level (שמייצגים דברים בסיסיים על מילים וביטויים, כמו POS או אם הן מייצגות למשל שם של בן אדם או מקום), וה-RNN בשכות הגבוהות יחשבו פיצ'רים high-level (כלומר מידע יותר מעמיק על המשפט – כמו מה הוא אומר, האם הוא בקונוטוציה חיובית או שלילית, מה המשמעות של ביטויים מתוכו וכו').

איך שרשתות כאלו עובדות היא שבכל נקודה בזמן אנחנו משתמשים בתוצאה שיוצאת מה-encoder לא רק כדי לחשב את השכבה הבאה באותה רשת, אלא גם כקלט לרשת שנמצאת מעליו.



בפרקטיקה, רשתות RNN שמשיגות תוצאות טובות הן לרוב רשתות multi-layer (שלמרות שהן יותר עמוקות מרשת RNN עם שכבה אחת, הן עדיין לא עמוקות כמו רשתות CNN או FF). למשל, במאמר מ-2017 מצאו שלמשימת NMT (תרגום מכונה), מספר השכבות האידאלי לדיקודר ברשת RNN מרובת שכבות הוא 2 עד 4. לרוב 2 שכבות משיגות תוצאה הרבבה יותר טובה משכבה בודדת, והוספת שכבה שלישית עושה הבדל פחות משמעותי. לרוב, skip-connections או dense-connections נדרשים כדי לאמן רשתות RNN יותר עמוקות (מספר אצבע של 8 שכבות). לצורך ההשוואה, טרנספורמרים (כמו למשל BERT) הם לרוב הרבה יותר עמוקים (בין 12 ל-24 שכבות, ומכילים הרבה מאוד skip-connections).

GREEDY DECODING AND BEAM SEARCH



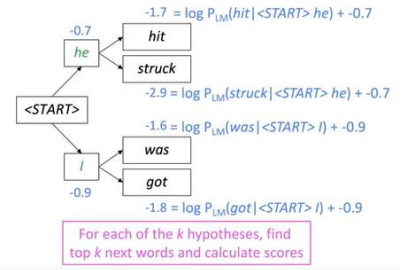
ראינו את הדרך הפשוטה ביותר לבצע decoding והיא לקחת argmax בכל צעד בדיקודר (השיטה הזו מכונה greedy decoding, כי מבצעים את הבחירה בכל צעד על בסיס המידע הקודם בלבד). אבל יש לשיטה הזו חסרון משמעותי - בגלל שבוחרים בכל פעם על בסיס מידע לוקלי בלבד, אנחנו "נתקעים" איתו ואין שום דרך לבטל החלטה שקיבלנו. למשל בדוגמא משמאל:

Input: *il a m'entarté* *(he hit me with a pie)*
→ *he* ____
→ *he hit* ____
→ *he hit a* ____ *(whoops! no going back now...)*

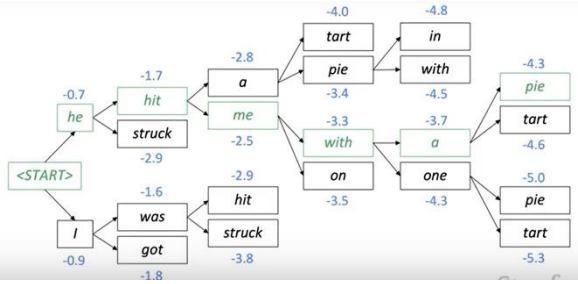
איך פותרים את זה? באמצעות Exhaustive search decoding. אידאלית, היינו רוצים למצוא תרגום (משפט y באורך T) שממקסם את ההסתברות של משפט התרגום בהנתן משפט הקלט, שמוגדרת באופן הבא:

$$P(y \mid x) = P(y_1 \mid x)P(y_2 \mid y_1, x)P(y_3 \mid y_1, y_2, x) \ldots, P(y_T \mid y_1, \ldots, y_{T-1}, x)$$
$$= \prod_{t=1}^{T} P(y_t \mid y_1, \ldots, y_{t-1}, x)$$

רעיונית, היינו יכולים לנסות לחשב את כל המשפטים האפשריים y, אבל זה אומר שלכל צעד t של הדיקודר אנחנו עוקבים אחרי V^t תרגומים חלקיים אפשריים (כאשר V הוא גודל אוצר המילים), כלומר המודל ירוץ בסיבוכיות זמן של $O(V^t)$ שזה מאוד לא יעיל. כדי לעשות את זה באופן יותר יעיל, משתמשים בשיטה בשם ***Beam search decoding*** – הרעיון בשיטה הוא שבכל צעד של הדיקודר, נמשיך לעקוב אחרי k התרגומים החלקיים הכי סבירים (*probable*), אותם אנחנו מכנים **היפותזות**. הפרמטר k מייצג את ה-*beam size*, ובד"כ בפרקטיקה הוא בערך בין 5 ל-10. לכל היפותזה $y_1, \ldots, y_t$ יש ציון *score* שהוא ה-*log probability* שלה, המוגדר באופן הבא: $\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t \mid x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i \mid y_1, \ldots, y_{i-1}, x)$. כל הציונים הם בהכרח שליליים, וככל שהציון גבוה יותר הוא יותר טוב. אנחנו מחפשים את ההיפותזות עם הציונים הכי גבוהים, ובכל צעד ממשיכים לעקוב אחרי k ההיפותזות עם הציונים הכי גבוהים. חשוב לזכור ששימוש בשיטה הזו לא בהכרח מבטיח מציאת פתרון אופטימלי, אבל זה עדיין מגדיל את הסיכוי למציאה שלו או משהו שקרוב אליו בהשוואה ל-*exhaustive search*.



דוגמא לשימוש ב-beam search: נקח לדוגמא *beam size* של *k=2*. בהתחלה אנחנו מתחילים עם הטוקן של *START*, ובוחרים את שתי (כלומר k) המילים הכי סבירות לפי מודל השפה שלנו. לאחר מכן, אנחנו מוצאים עבור כל אחת מהן מה שתי המילים הכי סבירות להמשיך אותן לפי מודל השפה. בכל שלב אנחנו מוסיפים להסתברות של השלב הנוכחי את ההסתברויות של השלבים הקודמים וככה מקבלים הסתברות שמייצגת את התרגום החלקי כולו.



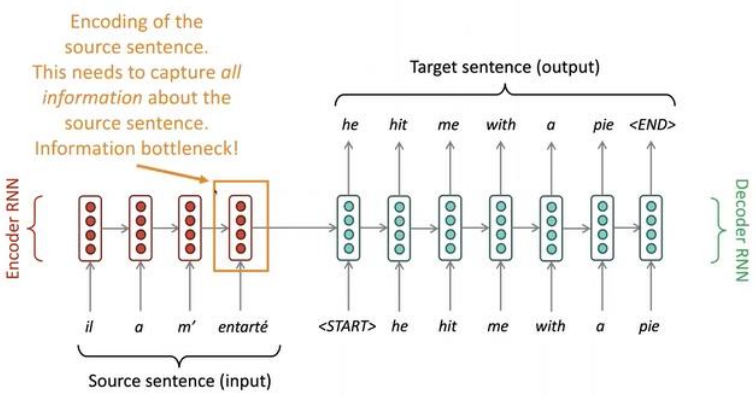
עכשיו כשיש לנו ציון לכל אחת מ-4 התרגומים החלקיים, אנחנו שומרים רק *k=2* מתוכן (את אלו עם הציונים הטובים ביותר, וממשיכים עבורן (כלומר יוצרים את שתי המילים הבאות בתרגום עבור כל אחת מהן לפי מודל השפה). עושים את זה שוב ושוב בכל שלב עד להגעה לטוקן *END* (ראו פירוט בפסקה הבאה), ואז בוחרים את ההיפותזה הטובה ביותר (ראו שתי פסקאות קדימה) כאשר אם נרצה אנחנו יכולים ללכת אחורה ולזהות מה היתה ההסתברות שהובילה לבחירה של כל מילה במשפט.

מתי עוצרים? עוד משהו שחשוב לשים לב אליו הוא שב-*greedy decoding*, אנחנו בד"כ מקודדים עד שהמודל חוזה את טוקן *END* שמייצג סוף משפט. למשל $< START >$ he hit me with a pie $< END >$. לעומת זאת, ב-*beam search decoding*, היפותזות שונות יכולות לייצר את טוקן ה-*END* בזמנים שונים. כאשר היפותזה מגיעה לטוקן *END*, ההיפותזה היא שלמה *complete*, ואנחנו שמים אותה בצד וממשיכים לבחון היפותזות אחרות באמצעות ה-*beam search*. בד"כ אנחנו נמשיך לחפש ב-*beam search* עד שנגיע למשפט באורך T (כאשר T הוא אורך *cutoff* מוגדר מראש) או אם יש לנו לפחות n היפותזות שלמות (כאשר n גם הוא *cutoff* מוגדר מראש).

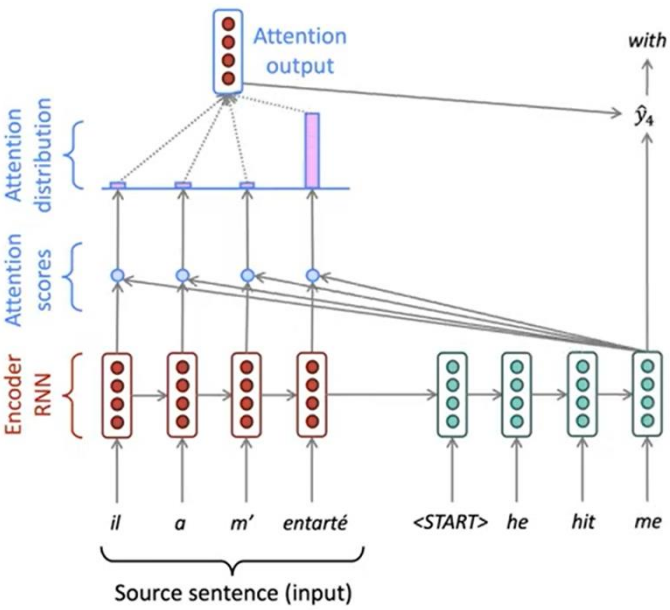
איך בוחרים את ההיפותזה הסופית? אחרי שעצרנו, יש לנו רשימה של היפותזות שלמות, כאשר לכל היפותזה מהרשימה $y_1, \dots, y_t$ יש ציון *score* המוגדר באופן הבא $\text{score}(y_1, \dots, y_t) = \log P_{\text{LM}}(y_1, \dots, y_t \mid x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i \mid y_1, \dots, y_{i-1}, x)$. תאורטית יכולנו פשוט לבחור את ההיפותזה עם הציון הגבוה ביותר, אבל יש עם זה בעיה מובנית – היפותזות ארוכות יותר (עם יותר מילים במשפט) יהיו בעלות ציון נמוך יותר בהגדרה (כיוון שהציון הוא סכום של מספרים שליליים שכל אחד מהם מייצג מילה במשפט). דרך נפוצה להתמודד עם הבעיה הזו היא לנרמל לפי האורך, כלומר להשתמש בציון הבא עבור כל היפותזה ואז לבחור את ההיפותזה שקיבלה את הציון המנורמל הכי גבוה - $\frac{1}{t}\sum_{i=1}^{t} \log P_{\text{LM}}(y_i \mid y_1, \dots, y_{i-1}, x)$.

ATTENTION

הסתכלנו על מודל *Seq2Seq* שמשתמשים בו בין היתר לתרגום מכונה. הבעיה הברורה שלו היא בעיית צוואר הבקבוק – המצב החבוי האחרון של האנקודר צריך להכיל מידע על משפט הקלט כולו, ולכן מהווה צוואר בקבוק.



הרעיון של *Attention* הוא למצוא פתרון לבעיית צוואר הבקבוק. הרעיון הוא בכל שלב בדיקודר, ליצור קישור ישיר *direct connection* לאנקודר כדי למקד אותו בחלק מסוים של משפט הקלט. זאת שיטה שמאפשרת להפיק מידע יותר איכותי ויותר גמיש ממשפט הקלט כדי להשיג תרגום איכותי. הוכח כי *Attention* משפר משמעותית את הביצועים של מודלי *NMT*. השיטה גורמת למודל להיות יותר *human-like* – אפשר להסתכל על משפט הקלט תוך כדי התרגום, במקום להצטרך לזכור את כולו. בנוסף לפתרון בעיית צוואר הבקבוק, *attention* עוזרת גם עם בעיית הגרדיאנטים הנעלמים, ומספקת מעין קיצור דרך למצבים חבויים רחוקים. כמו כן, השיטה מספקת מובן מסוים של *interpetability* בכך שהיא מספקת התפלגות שמאפשרת לנו לראות באילו מילים הדיקודר מתמקד בתהליך התרגום, מה שמאפשר לנו ממש לקבל *alignment* שהרשת לומדת בעצמה, מבלי שהיינו צריכים לאמן *alignment system* (כמו שהיינו עושים אם היינו בונים מודל *SMT*).



הרעיון הכללי הוא כזה – מתחילים עם אנקודר באותו אופן כמו ברשת *RNN* רגילה, ונייצר באמצעות המצב החבוי שלה את טוקן ההתחלה של המשפט המתורגם (המצב החבוי הראשון של הדיקודר, בטורקיז בתמונה). בנוסף, משתמשים בייצוג החבוי הזה כדי להסתכל ישירות על השורש ולקבל ממנו מידע. כלומר, משווים בין המצב החבוי הראשון של הדיקודר עם המצב החבוי של האנקודר בכל נקודה בזמן, ונשתמש במכפלה פנימית ביניהם כדי ליצור *attention score* (שהוא למעשה סוג של *similarity score*). משתמשים ב-*softmax* על ה- *attention scores* שחישבנו כדי להפוך אותן לוקטור התפלגות. משתמשים בהסתברות של כל אחת מהנקודות בזמן (השלבים של האנקודר, או מילים במשפט המקור) כדי לקבל וקטור *attention output* שמייצג ממוצע משוקלל של כמה קשב אנחנו רוצים לתת לכל אחת מהמילים במשפט הקלט. לאחר מכן, אנחנו לוקחים את הוקטור שקיבלנו ולוקחים סכום ממושקל שלו עם המצבים החבויים של האנקודר. זה ישמש אותנו באמצעות *softmax* כדי לחזות את כל אחת מהמילים במשפט המתורגם. לפעמים ניקח את ה-*attention output* מהצעד הקודם ובנוסף נזין אותו לדיקודר (ביחד עם הקלט הרגיל שלו).

ATTENTION – הגדרה פורמלית

מודל attention מוגדר ע"י הדברים הבאים:

- מצבים חבויים של האנקודר $h_1, \dots, h_N \in \mathbb{R}^h$
- לכל נקודה בזמן t, יש מצב חבוי של הדיקודר $s_t \in \mathbb{R}^h$
- לכל נקודה בזמן t, יש attention score המוגדר באופן הבא - $\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \dots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$ (מכפלה פנימית בין המצב החבוי של הדיקודר והמצב החבוי של האנקודר).
- מעבירים את ה-attention score להיות וקטור התפלגות באמצעות softmax, ומקבלים את ה-attention distribution המסומנת α^t לכל נקודה בזמן t, כלומר - $\alpha^t = \text{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$.
- משתמשים ב-α^t כדי לקחת סכום ממושקל של המצבים של החבויים של האנקודר, ונקבל את ה-attention output המוגדר באופן הבא לכל נקודה בזמן t. - $\boldsymbol{a}_t = \sum_{i=1}^N \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$.
- לבסוף, אנחנו מחברים את כל ה-attention outputs עם המצבים החבויים של הדיקודר ומשתמשים בזה ב- non-attention seq2sqe model - $[a_t; s_t] \in \mathbb{R}^{2h}$.

וריאציות נוספות של ATTENTION

מה שמשותף לכל הוריאציות של attention הוא שישנם מספר ערכים values המסומנים $\boldsymbol{h}_1, \dots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ וכמו כן שאילתא query המסומנת $\boldsymbol{s} \in \mathbb{R}^{d_2}$. כמו כן, בכל הגרסאות מה שנעשה זה:

- לחשב את ה-attention scores המסומנות $e \in \mathbb{R}^N$
- לקחת softmax כדי לקבל את ה-attention distribution המסומנת $\alpha = \text{softmax}\,(e) \in \mathbb{R}^N$
- להשתמש ב-attention distribution כדי לקחת סכום ממושקל של הערכים - $\boldsymbol{a} = \sum_{i=1}^N \alpha_i \boldsymbol{h}_i \in \mathbb{R}^{d_1}$ ובכך לקבל את ה-attention output המסומן a (לעתים מכונה context vector).

מה שמשתנה בין הוריאציות הוא האופן שבו מחשבים את ה-attention score המסומן e – יש מספר דרכים לעשות את זה:

- <u>Basic dot-product attention:</u> מחשבים באמצעות מכפלה פנימית $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$ (שימו לב שזה מניח שהמימד של המצבים החבויים של האנקודר ושל הדיקודר הם זהים - $d_1 = d_2$).
- <u>Multiplicative attention:</u> מחשבים באמצעות מכפלה ממושקלת $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$ כאשר $\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ היא מטריצת משקולות.
- <u>Reduced rank multiplicative attention:</u> מחשבים באמצעות $e_i = s^T(\boldsymbol{U}^T \boldsymbol{V}) h_i = (\boldsymbol{U}s)^T (\boldsymbol{V} h_i)$ כאשר המטריצות $\boldsymbol{U} \in \mathbb{R}^{k \times d_2}, \boldsymbol{V} \in \mathbb{R}^{k \times d_1}$ הן ממימד כך ש-$k \ll d_1, d_2$.

- <u>Additive attention:</u> מחשבים באמצעות $\boldsymbol{e}_i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_1\boldsymbol{h}_i + \boldsymbol{W}_2\boldsymbol{s}) \in \mathbb{R}$ כאשר $\boldsymbol{W}_1 \in \mathbb{R}^{d_3\times d_1}, \boldsymbol{W}_2 \in \mathbb{R}^{d_3\times d_2}$ הן מטריצות משקולות ו- $v \in \mathbb{R}^{d_3}$ הוא וקטור משקולות. d_3 (מימד ה-attention) הוא היפרפרמטר. השם "additive" הוא קצת מבלבל – למעשה מדובר בשכבה אחת של רשת נוירונים.

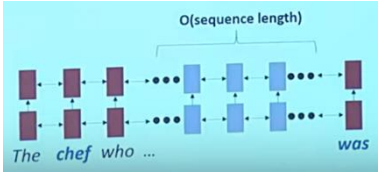
ATTENTION כשיטה כללית בלמידה עמוקה

עד כה דיברנו על attention בהקשר של פתרון בעיית תרגום המכונה באמצעות ארכיטקטורת seq2seq, אך זו שיטה שמשתמשים בה בארכיטקטורות נוספות (לא רק seq2seq) לשימושים רבים נוספים (לא רק MT) בלמידה עמוקה. הגדרה כללית יותר של attention היא – בהנתן וקטור ערכים values וכן וקטור שאילתא query, attention היא טכניקה לחישוב הסכום הממושקל של הערכים כתלות בשאילתא. האינטואיציה היא שהסכום הממושקל הוא סיכום סלקטיבי של המידע שהערכים מכילים, כאשר השאילתא קובעת באילו ערכים להתמקד. Attention היא דרך שמאפשרת להשיג ייצוג בגודל קבוע מראש fixed-size של קבוצה שרירותית של ייצוגים (הערכים), כתלות בייצוג אחר (השאילתא). באופן כללי, attention היא שיטה חזקה, גמישה וכללית למניפולציה על זכרון ברשתות למידה עמוקה.

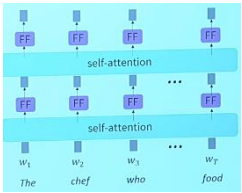
SELF ATTENTION

בפרק הקודם דיברנו על בעיות seq2seq, ועכשיו נעבור לגישה חדשה שמנסה להשתמש באבני בניין אחרות שנוכל להשתמש בהן כדי לבנות מודלים שאפשר להשתמש בהן כדי לפתור אותו סוג של בעיות.

חסרונות ובעיות של RNN ושיטות חלופיות

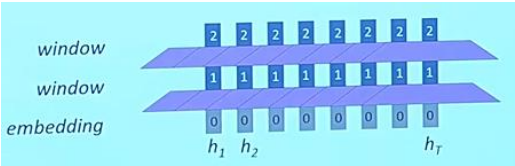


- <u>חסרון ראשון – Linear interaction distance</u> - רשתות RNN מחושבות "משמאל לימין", ובכך מקודדות לוקאליות לינארית (כלומר, מילים שקרובות זו לזו לרוב משפיעות על המשמעות אחת של השניה). זה אמנם שימושי, אבל החסרון שזה גורר הוא שרשתות RNN דורשות O(sequence length) צעדים כדי שמילים שרחוקות זו מזו יוכלו להיות באינטראקציה. זה גורם לכך שיהיה קשה ללמוד תלויות בין מילים שממוקמות רחוק זו מזו בתוך המשפט (בגלל בעיות הגרדיאנטים). הפתרון לחסרון זה יהיה position representation – ראו בהמשך הפרק. בנוסף, העובדה שהמשפט מחושב באופן לינארי היא בעייתית בגלל שאנחנו יודעים שזו לא הדרך הנכונה לחשוב על משפטים בשפה אנושית (למשל, עץ תחבירי יהיה יצוג יותר מתאים). הפתרון לחסרון זה יהיה להוסיף אי-לינאריות באופן יזום באמצעות שכבה נוספת של feed-forward network שתעשה post-process לכל אחד מוקטורי ה-output, למשל -

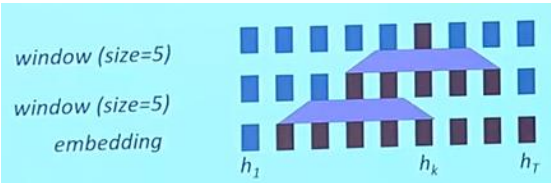


$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$$

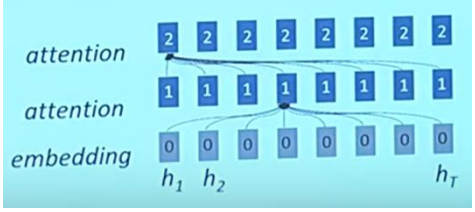
- <u>חסרון שני – העדר מקביליות:</u> מעברי ה-forward וה-backwards יוצרים O(sequence length) פעולות שלא ניתנות לחישוב באופן מקבילי (כי כל אחת מהן תלויה בקודמת), ובכך לא מאפשרים לנצל את היכולות המתקדמות של GPUs בימינו. הפתרון לחסרון זה יהיה masking the future – ראו בהמשך הפרק.



חלופה שאפשר לחשוב עליה ל-RNN היא **שימוש בחלונות מילים word windows**, שהם מודלים שמבצעים אגרגציה להקשר הלוקלי (ידועים גם בתור קונבולוציה ממימד 1). היתרון של המודלים האלו (בהשוואה ל-RNN) הוא מספר הפעלוות הלא מקביליות הנדרשות לחישוב אינו גדל ביחד עם הגדלת אורך המשפט (כיוון שגודל החלון קבוע, והחישוב לא תלוי בחישובים של מילים אחרות אלא עצמאי לחלוטין). כלומר, בשיטה זו ניצור מספר שכבות של חלונות שכל אחת מהן תלויה רק בשכבה מתחתיה, ולכן אין בכלל תלות במימד הזמן (המספרים בתמונה מייצגים את כמות הצעדים הנרשים כדי שהמצב המסומן יוכל להיות מחושב).



אבל, נזכר שהסיבה שבגללה רצינו להשתמש ב-RNN ולא בחלונות מילים מלכתחילה היתה שחלונות מילים לא מצליחים לתפוס בכלל תלויות בין מילים שאינן קרובות זו לזו במשפט. מסתבר שאפשר לפתור את זה ע"י יצירה של מספר שכבות של חלונות מילים, ובכך להצליח לתפוס אינטראקציות בין מילים רחוקות. המרחק המקסימלי בין מילים שעדיין מאפשר אינטראקציה הוא **אורך המשפט / גודל החלון** (המצבים הכהים בתמונה משמאל מייצגים מצבים ש"נראים" visible ע"י המצב h_k.

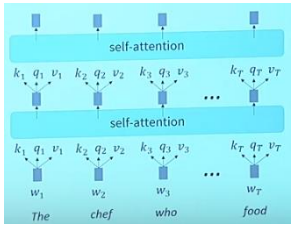


חלופה נוספת שאפשר לחשוב עליה ל-*RNN* היא שימוש ב-***attention***, שהיא כאמור שיטה בה מתייחסים לייצוג של כל מילה בתור שאילתא *query* ומשקללים מידע מתוך סט של ערכים. עד עכשיו ראינו שימוש ב-*attention* מהדיקודר לאנקודר, ובפרק הזה נחשוב על *attention* בתוך המשפט עצמו. כלומר, הרעיון הוא שכל המילים מקשיבות *attend* לכל המילים בשכבה הקודמת (ראו את הציור משמאל, רק שימו לב שרוב החצים בו הושמטו מטעמי בהירות ההדגמה). השיטה הזו היא גם לא מקבילית במימד העומק, אך כן מקבילית במימד הזמן. כלומר, גם כאן מספר הפעולות הלא מקביליות לא גדל עם אורך המשפט, ואין בכלל תלות במימד הזמן.

SELF ATTENTION

ב-*attention* אנחנו עובדים עם שאילתות *queries* ממימד d המסומנות $q_1,..,q_T$, מפתחות *keys* ממימד d המסומנים $k_1,..,k_T$ וכן ערכים *values* ממימד d המסומנים $v_1,..,v_T$. ב-***self attention***, השאילתות, המפתחות והערכים נוצרים כולם מאותו מקור (למשל, אותו המשפט). לדוגמא, אם הקלט של השכבה הקודמת היה $x_1,..,x_T$, נוכל להגדיר למשל כי $v_i = k_i = q_i = x_i$ (כלומר, להשתמש באותו וקטור שישחק את שלושת התפקידים).

איך מחשבים? בדומה ל-attention, גם ב-self attention נרצה לחשב attention scores, ונעשה את זה באמצעות מכפלה פנימית באופן הבא – נחשב את $e_{ij} = q_i^{\top} k_j$ (הקשר בין המפתח לשאילתא לכל קומבינציה אפשרית של שאילתא i ומפתח j). לאחר מכן, כדי לחשב את ה-attention distribution ב-self attention מחשבים את $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp\left(e_{ij'}\right)}$ (שקובע לכל שאילתא לאילו מפתחות היא רוצה להקשיב). לסיום, מחשבים את $\text{output}_i = \sum_i \alpha_{ij} v_j$ (סכום משוקלל של הערכים values עבור השאילתא ה-i).



Self-attention בתור אבן בניין לפתרון בעיות NLP – בתמונה משמאל אפשר לראות stacked self-attention blocks (באופן דומה ליצירת מודל בעמצעות ערימת שכבות LSTM). אבל, זה לא אומר שאפשר פשוט להשתמש בשיטה בזו בתור חלופה ל-LSTM (או ל-RNN באופן כללי), כיון שיש לה מספר בעיות ואתגרים בהם נדון כעת.

הבעיה הראשונה והברורה ביותר היא ש-self-attention לא מכיר את הסדר של הקלטים שלו – הוא פועל על מילות הקלט כקבוצות חסרות סדר, וזאת בעיה כי אנחנו יודעים שהסדר שבו מילים מופיעות במשפט הוא משמעותי ומשפיע על המשמעות של המשפט. כלומר, כדי לפתור את הבעיה הזו, אנחנו צריכים לקודד את הסדר של המשפט באמצעות המפתחות, השאילתות והערכים שלנו. למשל, נשקול לייצג כל אינדקס במשפט בתור <u>וקטור</u> באופן הבא - $p_i \in \mathbb{R}^d$, for $i \in \{1,2,\dots,T\}$ are position vectors (נסביר בהמשך איך כל p_i מוגדר). כעת נוכל לשלב את הוקטור הזה בקלות לתוך בלוק ה-self attention במודל שלנו – פשוט נשתמש ב-p_i בתור הקלט. נסמן

$$v_i = \tilde{v}_i + p_i$$

ב-$\tilde{v}_i \tilde{k}_i, \tilde{q}_i$ את הערכים, המפתחות והשאילתות הישנים, ונגדיר כעת $q_i = \tilde{q}_i + p_i$ (יש שיטות שבהן במקום חיבור מבצעים שרשור של

$$k_i = \tilde{k}_i + p_i$$

הוקטורים). כאשר אנחנו מבצעים deep self-attention, אנחנו רוצים לעשות את זה בשכבה הראשונה.

POSITION REPRESENTASION

נציג מספר שיטות להגדרת p_i – שהוא ה-position vector שמייצג את המיקום של המילה ה-i במשפט:

Sinusoidal position representation - שיטה בה מחברים concatenate פונקציות סינוסואידיות בנקודות שונות במחזור של כל פונקציה – כלומר, יוצרים וקטור שבו כל כניסה מייצגת פונקציה סינוסואידית כלשהי בנקודה בזמן. למשל, הוקטור הבא - $p_i =$

$$\begin{pmatrix} \sin\left(i/10000^{2*1/d}\right) \\ \cos\left(i/10000^{2*1/d}\right) \\ \vdots \\ \sin\left(i/10000^{2\cdot\frac{d}{2}/d}\right) \\ \cos\left(i/10000^{2\cdot\frac{d}{2}/d}\right) \end{pmatrix}$$

. היתרונות של השיטה הזו היא שהמחזוריות מציינת שאולי הנסיון לזהות מיקום אבסולוטי של מילה מסוימת הוא לא כזה חשוב, ובנוסף ניתן להשתמש בה עבור משפטים ארוכים יותר (כאשר המחזוריות של הפונקציה מתחילה מחדש). החסרונות של השיטה היא שהיא לא למידה וגם לא משיגה תוצאות טובות.

Learned absolute position representation – שיטה בה כל p_i מורכב מפרמטרים למידים. נגדיר מטריצה $p \in \mathbb{R}^{d\times T}$ שכל עמודה i בה היא למעשה p_i. היתרון העיקרי של השיטה הזו הוא גמישות (כל מיקום בוקטור יכול להלמד כדי להתאים לדאטא), והחסרון העיקרי שלה הוא שאי אפשר להשתמש בה כדי לחזות את p_i עבור אינדקסים שאינם בטווח בין 1 ל-T. עם זאת, רוב המודלים בימינו משתמשים בשיטה הזו (או בשיטות דומות שמשתמשות בגישות קצת יותר גמישות, למשל כאלו שמשתמשות בתחביר).

MASKING THE FUTURE

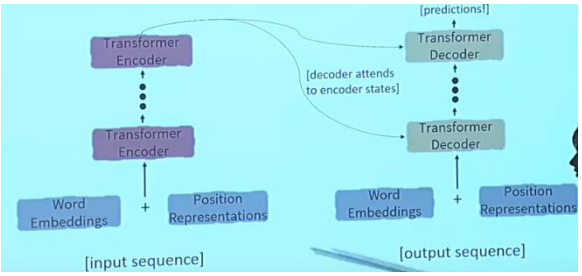
כדי להשתמש ב-self-attention בדיקודרים, אנחנו צריכים לוודא שהם לא מסוגלים להציץ בעתיד (כלומר במצבים הבאים). בכל נקודה בזמן, אנחנו יכולים לשנות את הסט של המפתחות והשאילתות כך שהם יכללו רק מילים מהעבר (החלק במשפט שכבר ראינו), אבל זה מאוד לא יעיל בגלל שזה לא מאפשר לנו חישוב מקבילי. לכן, כדי לאפשר מקביליות, אנחנו רוצים לעשות mask out ל-attention שמשויכת למילים מהעתיד ע"י זה שנתייחס ל-attention scores שלהן בתור מינוס אינסוף, כלומר $e_{ij} = \begin{cases} q_i^T k_j, k < j \\ -\infty, k \geq j \end{cases}$.

TRANSFORMERS

טרנספורמרים הם מודלי אנקודר-דיקודר שהשיגו תוצאות מצוינות במגוון משימות, ביניהן תרגום מכונה (בהם השיגו תוצאות BLEU טובות יותר וגם אימון יעיל יותר בהשוואה למודלים אחרים), סיכום summarization ועוד. היתרון הגדול שלהם (מלבד הביצועים) הוא שאפשר לעשות להם pretraining (כלומר לאמן אותם מראש על כמות גדולה של דאטא, שיטה שנעמיק בה בפרק הבא) בצורה מאוד יעילה (כיוון שהם מחושבים באופן מקבילי).

ארטיכקטורה של טרנספורמרים

בתור התחלה, נסתכל על אבני הבניין שמרכיבות את ה-Transformer encoder decoder ב-High level:

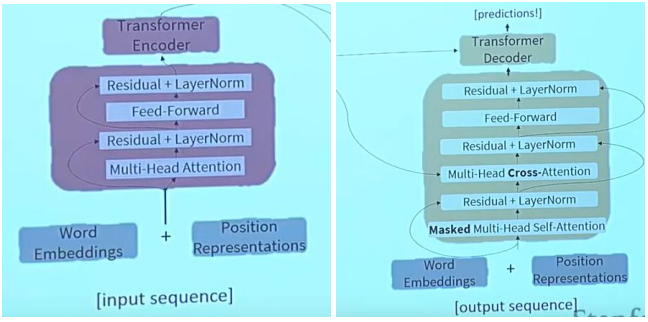


בתור קלט אנחנו מקבלים word embeddings להם אנחנו מוסיפים position representation, ואז באמצעות ה-transformer encoder אנחנו מקבלים את השכבה האחרונה שלו שמשמשת (ביחד עם ה-word embeddings וה-position representation) את ה-transformer decoder שהוא זה שמבצע את הפרדיקציה.

רוב אבני הבנין שמוזכרים בדיאגרמה הזו הוזכרו בחלק הקודם של הפרק, ונתמקד כעת באלו מהן שלא הוסברו עדיין:

- Key-query-value attention – איך להשיג את הוקטורים k, q, v מתוך embedding של מילה בודדת?
- Multi-headed attention – שימוש ב-attention במספר מקומות שונים בתוך שכבה בודדת.
- שיטות שעוזרות באימון של טרנספורמרים (בדגש על כך שהן משפרות את <u>האימון</u> ולא את המודל עצמו), כמו residual connections, layer normalization, scaling the dot product.
- Multi-head cross-attention

לאחר שימוש בשיטות אלו, נקבל את הארכיטקטורה הבאה עבור כל אחת משכבות ה-*transformer encoder* וכל אחד משכבות ה-*transformer decoder* (כל השיטות בדיאגרמה יוסברו בהמשך הפרק):

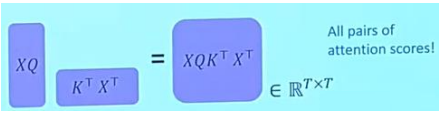


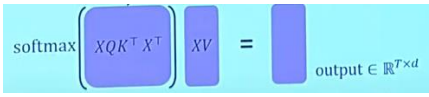
KEY-QUERY-VALUE ATTENTION

ראינו בתחילת הפרק שב-self-attention ישנם מפתחות keys, ערכים values ושאילתות queries שמגיעים כולם מאותו מקור. נגדיר כעת את הקלט של ה-Transformer encoder להיות $x_1, \dots, x_T$ (משפט המורכב מ-T מילים, שכל אחת מהן מיוצגת באמצעות וקטור ממימד d), ונגדיר את החלקים הבאים:

- המפתחות יהיו $\mathrm{k_i} = Kx_i$ כאשר $K \in \mathbb{R}^{d\times d}$ היא מטריצת המפתחות key matrix (טרנספורמציה ממימד d למימד d).
- הערכים יהיו $\mathrm{v_i} = Vx_i$ כאשר $V \in \mathbb{R}^{d\times d}$ היא מטריצת הערכים value matrix.
- השאילתות יהיו $\mathrm{q_i} = Qx_i$ כאשר $Q \in \mathbb{R}^{d\times d}$ היא מטריצת השאילתות query matrix.

כלומר, בשונה מהרעיון הראשוני שהצגנו בתחילת הפרק בו הערכים, המפתחות והשאילתות הם זהים, כעת הם יכולים להיות שונים זה מזה בזכות ההפעלה של הטרנספורמציות הלינאריות K,V,Q שיכולות להשתמש כדי להדגיש חלקים שונים בקלט כדי שיתאימו היטב לכל אחד מהתפקידים. נשים לב שכל המטריצות XK, XQ, XV הן ממימד $T \times d$. כעת, כדי לחשב את ה-*output* של ה--*self attention*, אנחנו למעשה מחשבים את $output = \mathrm{softmax}\left(XQ(XK)^\top\right) \times XV$.



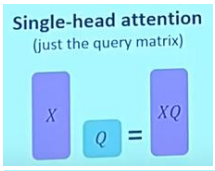


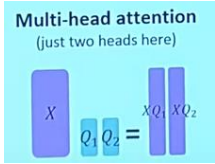
כלומר, בהתחלה אנחנו לוקחים את המכפלה $XQ(XK)^\top$ ומקבלים את כל הזוגות של ה-*attention scores* בתור מטריצה ממימד $T \times T$.

את המטריצה שקיבלנו בשלב הקודם מכניסים ל-*softmax* וכופלים במטריצה XV כדי לקבל את וקטור ה-*output* ממימד $T \times d$.

MULTI-HEADED ATTENTION

פתרון שמטרתו לאפשר להסתכל על מספר מיקומים במשפט בו זמנית, כלומר – עבור מילה i, ה-*self-attention* יתמקד במיקום שבו $x_i^\top Q^\top K x_j$ הוא גבוה, אבל יתכן אנחנו רוצים להתמקד (מכל מיני סיבות) דווקא ב-j אחר. לכן, נגדיר מספר *attention "heads"* שונים

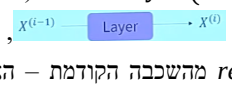




באמצעות הגדרת מטריצות Q,K,V שונות. כלומר, נגדיר את $Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d\times\frac{d}{h}}$ כאשר h הוא מספר ה-*attention heads* הרצוי ו-ℓ הוא בטווח בין 1 ל-h. כלומר הטרנספורמציה שהמטריצות האלו עושות (בשונה מקודם, אז הם היו ממימד d למימד d) הן ממימד d למימד קטן יותר $\frac{d}{h}$. כל *attention head* מבצע את ה-*attention* בנפרד, כלומר $\mathrm{output}_\ell = \mathrm{softmax}\left(XQ_\ell K_\ell^\top X^\top\right) * XV_\ell$ כאשר $\mathrm{output}_\ell \in \mathbb{R}^{d/h}$. לאחר שמחשבים את כל וקטורי ה-*output* לכל $\ell \in [1,h]$, מחברים ביניהם לכדי וקטור הפלט הסופי באופן הבא $\mathrm{output} = Y[\mathrm{output}_1; \dots; \mathrm{output}_h]$ כאשר Y היא מטריצה ממימד $d \times d$. כלומר, הרעיון הוא שכל *attention head* "מסתכל" על דברים אחרים, ויוצר וקטורי *value* אחרים.

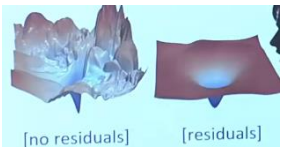
אפשר לראות בדיאגרמות משמאל שמספר החישובים לא השתנה במעבר מ-*single head* ל-*multi head* עם שני ראשים (זה קורה בגלל ש-Q_1, Q_2 הם ממימד נמוך יותר מהמימד של Q שהוא d).

RESIDUAL CONNECTIONS

בשיטה זו, במקום לחשב את $X^{(i)} = \mathrm{Layer}\left(X^{(i-1)}\right)$ (כאשר i מייצג את השכבה, כלומר הייצור של השכבה ה-i קורה על בסיס השכבה הקודמת) או באופן ויזואלי 

, מחשבים את $X^{(i)} = X^{(i-1)} + \mathrm{Layer}\left(X^{(i-1)}\right)$ (ככה שאנחנו צריכים ללמוד רק את ה"שארית" *residual* מהשכבה הקודמת – האינטואיציה היא שאנחנו רוצים ללמוד עבור השכבה ה-i במה היא שונה מהשכבה לפניה) או באופן ויזואלי

.



שימוש בשיטה הזו משפר מאוד את בעיות הגרדיאנטים והופך את המרחב להרבה יותר קל לחיפוש, ולכן את תהליך האימון ליותר יעיל ומהיר.

LAYER NORMALIZATION

הרעיון ב-*layer normalization* הוא לצמצם את החלקים שמשתנים בין השכבות איך אינם אינפורמטיביים ב- *hidden vector values* באמצעות נורמליזציה לתוחלת ולסטיית התקן <u>בתוך כל שכבה</u>. נקח את x, מילה בודדת (המיוצגת באמצעות וקטור מאורך d) מתוך הקלט של המודל. נגדיר את התוחלת $\mu = \sum_{j=1}^{d} x_j$ ואת סטיית התקן $\sigma = \sqrt{\frac{1}{d}\sum_{j=1}^{d}\left(x_j - \mu\right)^2}$ (שתיהן סקלרים). בנוסף

(אופציונלי – יש מאמרים שאומרים שהם לא מועילים, אבל עדיין נהוג להשתמש בהם), נגדיר את ה-*gain* המסומן γ ואת ה-*bias* המסומן β (שניהם וקטורים ממימד d).

כעת, מחשבים את וקטור הפלט באופן הבא - $\text{output} = \frac{x-\mu}{\sqrt{\sigma}+\epsilon} * \gamma + \beta$ (כאשר ϵ הוא מספר קטן כלשהו כדי לוודא שהמכנה לא נהיה קטן מידי). החלק השמאלי של הביטוי $\frac{x-\mu}{\sqrt{\sigma}+\epsilon}$ הוא למעשה הנרמול של x באמצעות התוחלת וסטיית התקן, והחלק הימני של הביטוי $\gamma + \beta$ הוא איזון *element-wise* באמצעות פרמטרי ה-*gain* וה-*bias* (שהם נלמדים).

SCALED DOT PRODUCT

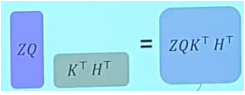
כאשר המימד d נהיה גדול, המכפלה הפנימית בין שני וקטורים ממימד d נהיית כבדה לחישוב. בגלל זה, הקלטים לפונקצית ה-*softmax* נהיים גדולים, ובהתאם הגרדיאנטים שלהם נהיים קטנים (בגלל שפונקציית ה-*sostmax* נהיית *peaky* ושמה את רוב ההסתברות שלה על מספר קטן של אפשרויות), כלומר אנחנו מאפסים את ההסתברויות של כל הכניסות שאינן *attended*.

עד כה ראינו את החישוב הבא - $\text{output}_\ell = \text{softmax}\left(XQ_\ell K_\ell^\top X^\top\right) * XV_\ell$. השינוי שמתבצע בשיטת *scaled dot product* הוא שמחלקים את ה-*attention scores* ב - $\sqrt{\frac{d}{h}}$ כדי למנוע מהתוצאות לגדול יותר מידי. כלומר, נחשב את $\text{output}_\ell = \text{softmax}\left(\frac{XQ_\ell K_\ell^\top X^\top}{\sqrt{d/h}}\right) * XV_\ell$.

CROSS-ATTENTION

ראינו בתחילת הפרק ש-self-attention היא שיטה שבה משתמשים במפתחות, שאילתות וערכים שמגיעים מאותו מקור. בדיקודר יש שימוש ב-attention "קלאסית". נסמן ב-$h_1, \dots, h_T$ את וקטורי הפלט (ממימד d) של האנקודר שמועברים לדיקודר, וב-$z_1, \dots, z_T$ את וקטורי הקלט (ממימד d) של הדיקודר (הם יכולים להיות השילוב של ה-word embeddings וה-position representation, או שהם יכולים להיות פלט של שכבת דיקודר קודמת). נגדיר כי המפתחות והערכים מופקים מתוך האנקודר באופן הבא - $k_i = Kh_i, v_i = Vh_i$, בעוד השאילתות מופקות מתוך הדיקודר באופן הבא - $q_i = Qz_i$.

נביט באופן החישוב של ה-cross-attention במונחים של מטריצות: נסמן ב-$\mathrm{H} = [h_1; \dots; h_T] \in \mathbb{R}^{T\times d}$ את השרשור של וקטורי האנקודר, וב-$\mathrm{Z} = [z_1; \dots; z_T] \in \mathbb{R}^{T\times d}$ את השרשור של וקטורי הדיקודר. כעת, הפלט של הדיקודר יחושב ע"י הנוסחא הבאה - $\text{output} = \text{softmax}\left(ZQ(HK)^\top\right) \times HV$. כלומר, החישוב הוא כמו ב-attention "קלאסי", רק שבשיטה הזו הקלטים מגיעים ממקום אחר.



באופן ויזואלי, בשלב הראשון אנחנו לוקחים מכפלה פנימית של מטריצת השאילתות ומטריצת המפתחות, ומקבלים את המטריצה ממימד $T \times T$ שמייצגת את ה-attention scores של כל הזוגות.

לאחר מכן, אנחנו מעבירים את התוצר דרך softmax וכופלים אותו במטריצת הערכים כדי לקבל את וקטור הפלט של הדיקודר, שהוא ממוצע משוקלל של הערכים.

SUBWORD MODELING

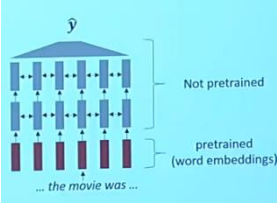
עד כה, ביצענו הנחות די משמעותיות על אוצר המילים של שפות – למשל, הנחנו שיש לכל שפה אוצר מילים קבוע **וסופי** כלשהו (בגודל כמה עשרות אלפי מילים), אותו הסקנו מהקורפוס בו השתמשנו לאימון. זה גרם לכך שכל מילה חדשה novel (שלא נתקלנו בה בקורפוס) מופתה ל-token המסומן UNK (unknown). זה משפיע לא רק על מילים לא ידועות – דוגמאות נוספות שימופו ל-UNK יהיו מילים באיות שלא נתקלנו בו בקורפוס האימון (למשל taaasty במקום tasty) ומילים עם שגיאות הקלדה. יש גם שפות מסוימות בהן ההנחה הזו שגודל אוצר המילים הוא סופי היא עוד פחות הגיונית – למשל שפות בהן יש מורפולוגיה מורכבת, או **מבנה מילה word structure*. התוצאה של מבנה שכזה היא שיש יותר סוגים של מילים, שכל אחת מהן מופיעה פחות פעמים. למשל, בסוואהילי Swahili, לפעלים יכולים להיות מאות הטיות שונות שמקודדות סוגי מידע שונים (כמו זמן, מצב רוח, שלילה, מידע על אובייקט וכו').

גישת **subword modeling** היא משפחת אלגוריתמים שמציעה פשרה בין שתי גישות – הגישה הראשונה היא בינארית (או שאנחנו מכירים מילה, או שאנחנו לא מכירים אותה – כמו שראינו עד כה) והגישה השניה היא התייחסות למילה בתור רצף של אותיות (וב"אחריות" רשת הנוירונים להבין מרצף האותיות את המשמעות). ספציפית, באלגוריתם **byte-pair encoding** לומדים את אוצר המילים מתוך קורפוס אימון, אבל במקום להפריד בין מילים באופן יוריסטי (למשל באמצעות רווחים) מפרידים בין מילים (ובכך מגדירים את אוצר המילים באמצעות אלגוריתם חמדני greedy. כלומר, מתחילים עם אוצר מילים שמכיל רק אותיות וסימנים המציינים end-of-word (כמו למשל נקודה). לאחר מכן, משתמשים בקורפוס האימון כדי למצוא את צמדי האותיות שמופיעות הכי הרבה אחת אחרי

השניה, וכך יוצרים subwords באורך שתי אותיות. מחליפים את המופעים של זוג האותיות ב-subword החדש שיצרנו, וחוזרים על זה עד שמסיימים ליצור את אוצר המילים כולו. לדוגמא – מתחילים עם כל האותיות באנגלית, ומסיימים עם אוצר המילים $\{a, \dots, z, \dots, \text{apple}, \text{app\#}, \text{\#ly}, \dots\}$ (כאשר # מציין כמות לא ידועה של אותיות לא ידועות שבהכרח מופיעות – כלומר app# ו-app אינן אותה מילה – הראשונה מייצגת את זה שמופיעות עוד אותיות אחרי האותיות app, והשניה היא המילה app בפני עצמה). השיטה הזו שימשה במקור בעיקר לתרגום מכונה, ובימינו שיטה דומה לזו (המכונה WordPiece) משמשת pretrained models. עכשיו נוכל להתמודד (לא בצורה אידאלית, אבל יותר טוב מסתם למפות ל-UNK) גם עם מילים באיות לא מוכר (למשל taaasty ימופה ל- #taa sty#) או בשגיאת איות (למשל laern ימופה ל-#earn #la) וגם (לעתים בצורה קצת יותר טובה) עם מילים חדשות (למשל transformerify במקרה האידאלי ימופה למשהו כמו transformer# #ify).

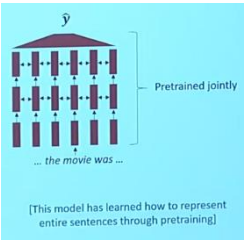
MODEL PRETRAINING

היסטוריה של PRETAINING



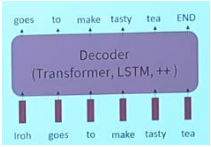
הרעיון מאחורי שיטות כמו word2vec הוא ההנחה שאפשר ללמוד הרבה על המשמעות של מילה ע"י הסתכלות על המילים שהיא מופיעה בצמוד אליהן. עם זאת, השיטה זו לא באמת מאפשרת להתחשב בהקשר של המילים כמו שצריך. למשל, במשפט **I record the record**, המילה record מופיעה פעמיים בשתי משמעויות שונות, אבל אם נסתכל על חלון בגודל 3 למשל, נקבל ייצוג זהה עבור שני המופעים.

בתחילת המחקר של pretrained word embeddings ב-2017, השיטה נראתה כך – מתחילים עם pretrained word embeddings (ללא הקשר), ולומדים איך לשלב בהם את ההקשר באמצעות רשת LSTM או טרנספורמר כחלק מהאימון על המשימה אותה רוצים לפתור. החסרון בשיטה הזו היא שהיא מצריכה שדאטא האימון ל- downstream task אותה אנחנו רוצים לפתור (למשל מענה על שאלות או ניתוח סנטימנט) צריך להיות גדול מספיק כדי לאפשר למודל ללמוד את כל ההקשרים האפשריים בשפה כולה (כיוון שזה לא מיוצג ב- word embeddings), שזו ציפיה מאוד גדולה (במיוחד כי רוב הפרמטרים ברשת שלנו מאותחלים באופן רנדומלי).



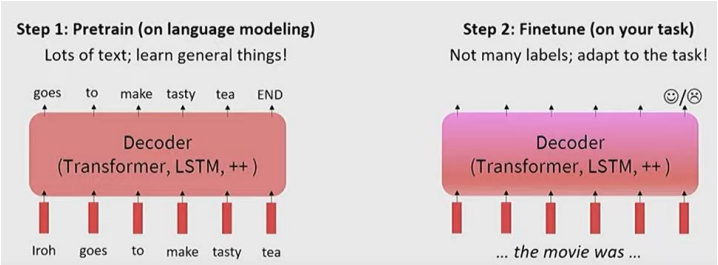
בימינו, מה שנהוג הוא ליצור מודל בו כל (או כמעט כל) הפרמטרים ברשת מאותחלים באמצעות pretraining. שיטות pretraining מסתירות מהמודל חלקים מהקלט, ומאמנות אותו לבנות אותם מחדש. הרעיון הזה התגלה כאפקטיבי מאוד ומאפשר ליצור ייצוגים חזקים ומדויקים של שפות, אתחול פרמטרים טובים עבור מודלי NLP, ויצירת התפלגויות לשפות שאפשר לדגום מהן.

PRETRAINING



נזכר במשימת מידול השפה language modeling אנחנו רוצים למדל את $p_\theta(w_t \mid w_{1:t-1})$, שהיא ההסתברות שהמילה הבאה במשפט היא w_t בהנתן המילים שכבר קיימות במשפט. ברוב השפות (אבל באנגלית במיוחד) יש הרבה מאוד דאטא עבור המשימה הזו (כל מה שצריך זה פשוט טקסט בשפה הרצויה, לא נדרש תיוג או עיבוד משום סוג), ולכן היא אידאלית ל-pretraining. כדי לבצע *pretraining* למשימת מידול השפה, נאמן רשת נוירונים למשימה זו על כמות גדולה של טקסט, ונשמור את הפרמטרים של הרשת לאחר סיום האימון.

הפרדיגמה של *pretraining* היא בתור שלב ראשון לבצע *pretraining* (על משימת *language modeling*), ובשלב השני להשתמש במודל המאומן ולעשות לו *fine tuning* כדי להתאים אותו למשימה הרצויה (למשל ניתוח סנטימנט).



SGD* ב-*pretrain/finetune - למה שאימון מראש ופיין טיונינג יעזרו לרשתות נוירונים שמיועדות למשימות אחרות? נסתכל על הפרמטרים $\hat{\theta}$ שאנחנו מקרבים באמצעות פונקצית ה-*pretraining loss* המסומנת $\min_\theta \mathcal{L}_{\text{pretrain}}(\theta)$ בהם אנחנו משתמשים בשלב השני

כדי לקרב את ה-*finetuning loss* המסומן $\min_{\theta} \mathcal{L}_{\text{finetune}}(\theta)$. הסיבה שה-*pretraining* משפיע היא ש-*SGD* ישאר קרוב ל-$\hat{\theta}$ בשלב ה-*finetuning* (זה יכול לאפשר לנו להכליל בצורה טובה – עדיין לא ברור לגמרי למה זה קורה. בנוסף, יתכן שזה יפתור לנו חלק מבעיות הגרדיאנטים ברשת ובכך לשפר את תהליך האימון בשלב השני).

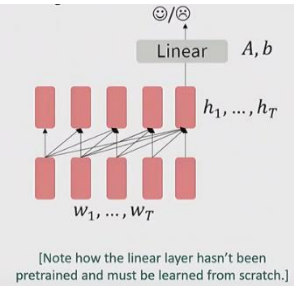
סוגי PRETRAINING

יש שלוש דרכים אפשריות לבצע *pretraining*:

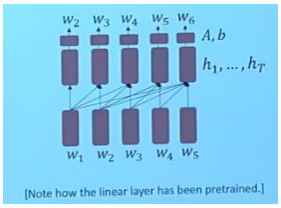
Decoders	מה שראינו עד כה – אימון מודלי שפה. מאפשר יצירה *generation* של שפה, אבל לא מאפשר להתנות על מילים "בעתיד".
Encoders	מאפשר הקשר דו-כיווני, ולכן מאפשר להתנות על מילים "בעתיד" (אבל אי אפשר להשתמש במשימת מידול השפה עבור ה-*pretraining* כי אם הרשת יכולה להסתכל על מילים בעתיד, היא יודעת מה המילה הבאה במשפט או המילה החסרה במשפט ולכן ה-*loss* תמיד יהיה 0).
Encoder-decoders	מאפשר להשתמש ביתרונות של אימון מקדים גם על אנקודרים וגם על דיקודרים.

PRETRAINING DECODERS

שתי דרכים לביצוע finetuning על מודל pretrained decoder:



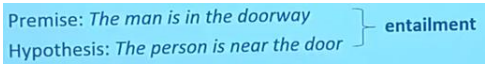
דרך ראשונה: כשרוצים להשתמש במודל שפה שהוא *pretrained decoder*, אפשר להתעלם מכך שהם אומנו כדי למדל את $p(w_t \mid w_{1:t-1})$ ולבצע *finetuning* ע"י אימון *classifier* על המצב החבוי של המילה האחרונה - $h_1, \dots, h_T = \text{Decoder}(w_1, \dots, w_T)$, $y \sim Ah_t + b$ כאשר A ו-b מאותחלים באופן אקראי (כלומר אינם *pretrained*) ומתאימים ל-*downstream task* (למשל זיהוי סנטימנט). כלומר מה שמסומן באדום בציור הוא הרשת ה-*pretrained*, לה הוספנו עוד שכבה לינארית (באפור) שהיא למעשה ה-*finetuning* עבור המשימה החדשה. עם זאת, הוספת השכבה הזו היא לא השינוי היחיד – אנחנו גם מפעפעים אחורה *backpropagating* את הגרדיאנטים לאורך כל הרשת, ובכך מעדכנים גם את הפרמטרים ברשת ה-*pretrained* כדי להתאים למשימה החדשה (ובהתאם ל-*finetuning loss*).



דרך שניה: זה טבעי לאמן מראש דיקודרים בתור מודלי שפה, ואז להשתמש בהם למשימות גנרציה של שפה, ולבצע *finetuning* על ההתפלגות $p(w_t \mid w_{1:t-1})$. זה שימושי במשימות בהם הפלט הוא רצף של מילים עם אוצר מילים דומה לזה שהשתמשו בו עבור ה-*pretraining*. למשל, במשימות דיאלוג (בהן ההקשר הוא המשפטים שנאמרו בדיאלוג עד כה) ובמשימות סיכום *summarization* (בהן ההקשר הוא המסמך אותו רוצים לסכם). כלומר, $h_1, \dots, h_T = \text{Decoder}(w_1, \dots, w_T)$, $w_t \sim Ah_{t-1} + b$ כאשר A,b הם *pretrained* ע"י מודל השפה.

מודלי GPT

Natural language inference היא משימה בה נותנים תגית *label* לזוגות של משפטים כדי לזהות את היחס ביניהם – גרירה *entailment* / סתירה *contradiction* או ניטרלי *neutral*. כלומר, מתייחסים למשפט הראשון (משפט ה-*premise*) כאמת ומנסים לקבוע מה היחס של משפט השני (משפט ההיפותזה) ביחס אליו.



מודל *GPT – General pretrained decoder* הוא מודל שפורסם ב-2018 שהוא *pretrained transformer decoder*, המורכב מ-12 שכבות של מצבים חבויים ממימד *d=768* ושכבות *FF* חבויות ממימד 3072. הוא משתמש ב-*byte-pair encoding* שיצר אוצר מילים עם 40,000 שילובי אותיות, ואומן על *BooksCorpus* שהוא קורפוס המכיל מעל 7,000 ספרים באנגלית (שמכיל גם הרבה מקטעים של טקסט רציף וארוך, מה שמאפשר ללמוד תלויות *dependencies* גם מחלקים רחוקים מאוד במשפט).

כדי לבצע *finetuning* של *GPT* למשימת *Natural language inference*, מה שניסו לעשות הוא להתאים את הקלט באמצעות טוקנים *tokens* כך שאפשר יהיה להשתמש באותה ארכיטקטורה של רשת נוירונים (שאומנה להיות מודל שפה) עבור הבעיה הזו. כדי לעשות זאת, איחדו את שני המשפטים בקלט למשפט אחד באופן הבא - הוסיפו טוקן *START* לפני המילה הראשונה של המשפט הראשון, הוסיפו טוקרן *DELIM* (מפריד, *delimiter*) בין המשפטים, והוסיפו טוקן *EXTRACT* בסוף המשפט השני:

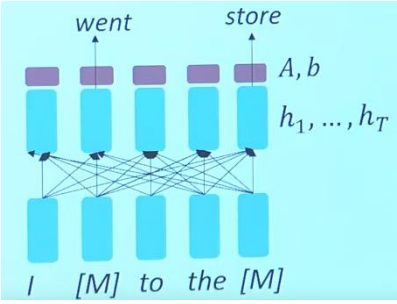
[START] The man is in the doorway [DELIM] The person is near the door [EXTRACT]

הקלסיפייר הלינארי הופעל על הייצוג של טוקן ה-*EXTRACT*.

מודל *GPT-2* – במודל זה השתמשו ב-*pretrained decoder* שאומן כמודל שפה בכך שהשתמשו בהסתברויות שהוא חוזה למילים. מודל *GPT-2* הוא גרסא גדולה יותר של מודל ה-*GPT* (שפורסמה גם היא ב-2018 ואומנה על יותר דאטא), הדגימה יכולות מרשימות ביצירת שפה *natural language generation*, לדוגמא:

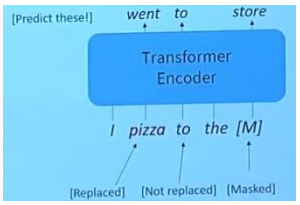
Context (human-written): *In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.*
GPT-2: *The scientist named the population, after their distinctive horn, Ovid's Unicom. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.*

PRETRAINING ENCODERS



עד כה, כדי לבצע את משימת ה-*pretraining* השתמשנו במשימת מידול שפה *language modeling*, אבל אי אפשר להשתמש בה כדי לאמן אנקודרים (כיוון שהם משתמשים בהקשר דו-כיווני, כלומר רואים את המילים בהמשך המשפט, לכן אין משמעות בלנסות לחזות אותן). לכן, מה שנהוג עבור ביצוע אימון מקדים לאנקודרים הוא שימוש במשימת ***masked language modeling***. הרעיון הוא להחליף חלק מסוים מהמילים במשפט בטוקן מיוחד *[MASK]*, ואז לתת למודל לנסות לחזות אותן. כלומר, $h_1, \dots, h_T = \text{Encoder}(w_1, \dots, w_T)$, $y_i \sim Aw_i + b$. נחשב את ה-*loss* רק על המילים שהן *masked out* – אם $\tilde{x}$ הוא גרסא *masked* של x, המודל ינסה ללמוד את $p_\theta(x \mid \tilde{x})$.

מודל BERT – BIDIRECTIONAL ENCODER REPRESENTATION FROM TRANSFORMERS



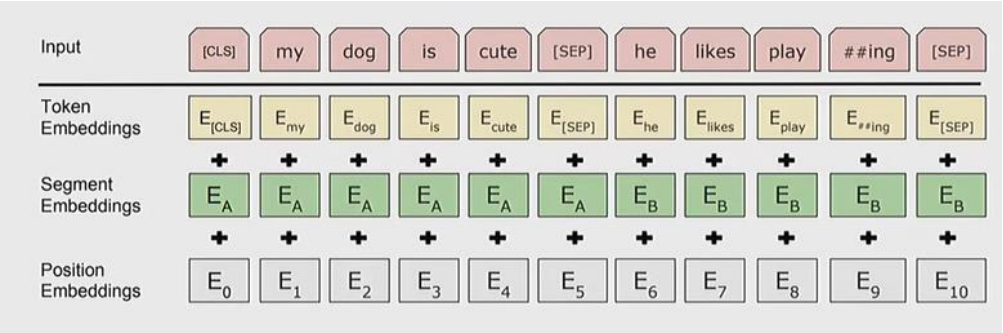
מודל *BERT* הוא מודל *pretrained encoder* שפורסם בשנת 2018 (המשקולות שלו פורסמו לציבור) שאומן באמצעות *masked LM*. האימון בוצע באופן הבא – המודל אומן על משפטים ש-15% מהמילים בהם היו *masked* – ב-80% מהפעמים החליפו את המילה בטוקן *[MASK]*, ב-10% מהמקרים החליפו אותה במילה אקראית, וב-10% הנוספים השאירו אותה כמו שהיא (אבל עדיין ביקשו מהמודל לחזות אותה). הסיבה לכך היא שאנחנו לא רוצים לתת למודל להתחמק מליצור ייצוגים חזקים עבור מילים שאינן *masked* (בגלל שבשלב הבא, שלב ה-*finetuning*, אף מילה לא תהיה *masked*).

BERT שוחרר בשתי גרסאות:

שם המודל	מספר שכבות	מימד המצבים החבויים *d*	מספר *attention heads*	מספר פרמטרים
BERT-base	12	768	12	110 מיליון
BERT-large	24	1024	16	340 מיליון

BERT אומן על שני קורפוסים – *BooksCorpus* (800 מיליון מילים) וגם ויקיפדיה באנגלית (2,500 מילים). האימון היה יקר מאוד ולכן לא פרקטי לביצוע על *GPU* יחיד – תהליך האימון המקדים של *BERT* בוצע על 64 כרטיסי *TPU* (כרטיס עם חומרה מיוחדת שמאפשרת ביצוע של *tensor operations* ביעילות ובמהירות) במשך 4 ימים. לרוב, תהליך *finetuning* ל-*BERT* הוא מאוד מהיר וניתן לביצוע על *GPU* יחיד.

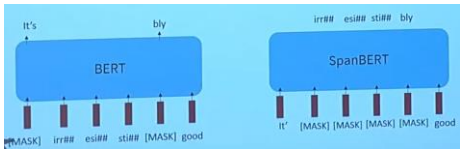
הקלט ל-*pretraining* של *BERT* מחולק לשני *chunk*-ים רציפים של טקסט – לפני הצ'אנק הראשון מופיע טוקן *CLS*, בין הצ'אנקים מופיע טוקן *SEP*, ובסיום הצ'אנק האחרון מופיע טוקן *SEP* (בשונה מהדוגמה למטה, הצ'אנקים בד"כ ארוכים מאוד ומכילים כמה מאות מילים). המטרה בשיטה הזו היא לנסות לאמן את המודל לזהות את היחסים בין שתי חתיכות די גדולות של טקסט (מה שיכול להועיל למשימות המשך מורכבות כמו למשל מענה על שאלות או דיאלוג). האופן בו הם יצרו את דאטא האימון הוא כזה – בחלק מהמקרים הם יצרו את הקלט כך שהצ'אנק השני של הטקסט הוא המשך ישיר של הצ'אנק הראשון, ובחלק מהמקרים הם יצרו אותו כך שהצ'אנק השני הוא טקסט אקראי שאין לו קשר ישיר לצ'אנק הראשון. הרעיון הוא שהמודל יצטרך לחזות עבור כל שרשור צ'אנקים כזה שמוזן לו כקלט, באיזה מהמקרים מדובר (האם הם המשך אחד של השני או שהם לא קשורים זה לזה).



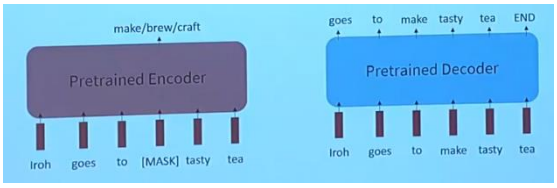
כיום, ביצוע *finetuning* ל-*BERT* הוא פרקטיקה מאוד נפוצה במגוון ורסטילי של בעיות *NLP*. משתמשים במגוון גדול דאטאסטים ומשימות כדי לבצע אוולוואציה ל-*BERT* עצמו, למשל הדאטאסטים הבאים ש-*BERT* הראה עבורם תוצאות מרשימות לאחר *finetuning* (ובפרט תוצאות הכי טובות שהושגו עד כה) - *QQP* (*Quora Question Pairs*) לזיהוי וניסוח שאלות, *QNLI* בנושא *natural language inference* על דאטא של שאלות ותשובות, *SST-2* לניתוח סנטימנט ועוד.

הרחבות של *BERT* – ישנן הרבה הרחבות של מודל *BERT* שמבצעים שיפורים לתהליך ה-*pretraining*, למשל:

- ***roBERTa*** – באופן כללי פשוט מאמן את *BERT* לאורך יותר זמן (כי *BERT* הוא *underfit* – זה משהו שהוצג לראשונה במאמר עצמו של *roBERTa*) ומוריד את הפרדיקציה של המשפט השני.
- ***SpanBERT*** – עושה *masking* לרצפים רצופים של מילים (מה שהופך את זה ליותר מאתגר עבור המודל). למשל, הדוגמא הימנית בתמונה תהיה השינוי ש-*SpanBERT* יעשה בהשוואה לקלט של *BERT* המקורי (שמוצג בדוגמא השמאלית בתמונה), מה שהופך את החיזוי להרבה יותר מאתגר.

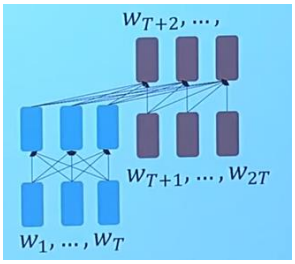


הגבלות של מודלי PRETRAINED ENCODERS



כפי שראינו, מודלי *pretrained encoders* כמו *BERT* משיגים תוצאות טובות על *finetuning* עבור מגוון גדול של בעיות. אבל אם המשימה אותה אנחנו רוצים לפתור מערבת יצירה *generation* של רצף מילים, נעדיף שלא להשתמש בהם אלא ב-*pretrained decoder*. הסיבה לכך היא ש-*BERT* וגם *pretrained encoders* אחרים לא יוצרים רצפים בצורה שהיא אוטורגרסיבית (מילה אחת בכל פעם).

PRETRAINING ENCODER-DECODERS



כמו שראינו עד כה, ל-*pretrained decoders* יש יתרונות מובהקים (יכולת לייצר רצפי טקסט) וגם ל-*pretrained encoders* יש יתרונות (ביצועים מצוינים על משימות שאינן גנרציה של רצפי טקסט), לכן נרצה ליצור מודל *pretrained encoder-decoder* שישלב את היתרונות של שני סוגי המודלים. כדי לעשות זאת, נשתמש במשימה כמו מידול שפה שבה התחילית של כל קלט מסופק לאנקודר ואינו נחזה על ידו. החלק של האנקודר מרוויח מההקשר הדו כיווני, והחלק של הדיקודר משמש כדי לאמן את המודל כולו באמצעות *LM*.כלומר -

$$h_1, \dots, h_T = \text{Encoder}\ (w_1, \dots, w_T)$$

$$h_{T+1}, \dots, h_{2T} = \text{Decoder}\ (w_1, \dots, w_T, h_1, \dots, h_T)$$

$$y_i \sim Aw_i + b, i > T$$

דוגמא למודל שכזה הוא **מודל *T5*** שפורסם ב-2018, שבו מחליפים רצפי מילים באורכים שונים (לא באורך ספציפי כלשהו שהמודל יודע מה הוא) מתוך הקלט ב-*placeholders* יחודיים, ואז מאפשרים לדיקודר לנסות לפענח את הרצפים שהוסרו. השיטה הזו נקראת ***span corruption*** שהוא סוג של ***denoising***.

Original text Thank you for inviting me to your party last week.	טקסט מקורי
Inputs Thank you <X> me to your party <Y> week.	קלטים לאנקודר
Targets <X> for inviting <Y> last <Z>	פלט רצוי של הדיקודר

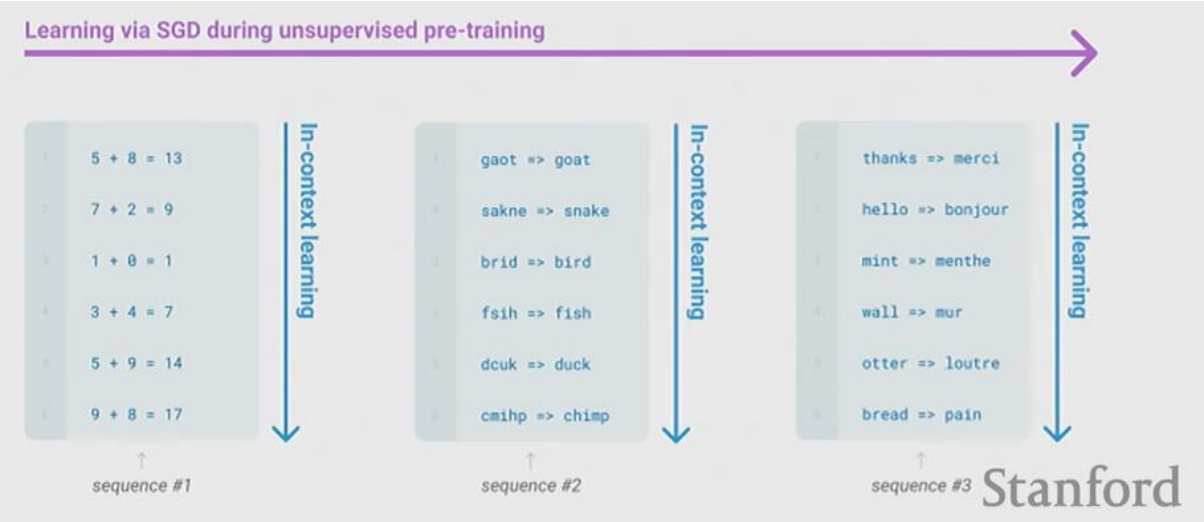
מאמר מ-2018 הראה שמודלי *encoder-decoder* משיגים תוצאות טובות יותר על מגוון משימות בהשוואה לדיקודרים, וש-*span corruption* עובד יותר טוב מ-*LM*. מה שמעניין מאוד במודל *T5* הוא שאפשר לעשות לו *finetuning* כדי לפתור מגוון רחב מאוד של משימות שקשורות למענה על שאלות, באופן שדורש שליפה של מידע אמיתי על העולם שמאוחסן בפרמטרים שלו. למשל, השלמה של פרטי טריויה ("אוניברסיטת סטנפורד ממוקמת ב___, קליפורניה"), בחירה של מילה שמבטאת סנטימנט הולם ("הדבר היחיד שנהניתי ממנו היה הפופקורן והשתיה, הסרט עצמו היה ___"), בחירה של מילה הגיונית תחבירית ("___ את המזלג על השולחן"), בחירה של מילה עם משמעות מתאימה ("הלכתי לצלול כדי לראות דגים, צבים וגם ___") ועוד מגוון רחב של משימות.

מודלי שפה ענקיים ו-IN-CONTEXT LEARNING

עד כה, ראינו שתי דרכים להשתמש במודלים *pretrained* – הראשונה היא לדגום מתוך ההסתברות שהם מגדירים (למשל באמצעות *prompt* ש"יתניע" תהליך גנרציה של טקסט מהמודל) והשניה היא לעשות להם *finetuning* כך שיתאימו למשימה חדשה שמעניינת אותנו. מודלי שפה ענקיים מצליחים לבצע למידה כלשהי **מבלי לבצע צעדי גרדיאנט נוספים** פשוט מתוך הדוגמאות שמסופקות להם בהקשר הולם. **מודל *GPT-3*** הוא דוגמא קלאסית לזה. בעוד *T5* מכיל 11 מיליארד פרמטרים, מודל *GPT-3* מכיל 175 מיליארד פרמטרים.

נראה כי דוגמאות ה-*in-context* רומזות למודל מהי המשימה שצריכה להיות מבוצעת, וההסתברות המותנית מאפשרת לחקות את ביצוע המשימה ברמה כזו או אחרת. למשל, אם הקלט יהיה: $hello \rightarrow bonjour$, נוכל לתת למודל מילה כלשהי באנגלית ולצפות שהוא יחזיר את התרגום ההולם שלה בצרפתית (נראה שהוא למד משהו על המשימה מההקשר של הקלט שניתן לו). זה עובד יפה על מגוון עצום של משימות, למשל פתרון תרגילי חיבור, תיקון שגיאות כתיב, תרגום ועוד.

$$thanks \rightarrow merci$$
$$min \rightarrow menthe$$

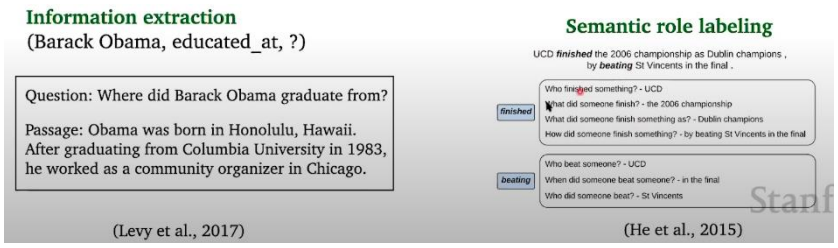


יש הרבה מחקר שמתקיים כדי להבין למה התופעה הזו (שהמודל מצליח להבין מההקשר של הקלט מה המשימה), אבל עוד לא לגמרי ברור למה היא מתרחשת והמודלים האלו עדיין לא מובנים היטב.

QUESTION ANSWERING

המטרה במשימת question answering היא לבנות מערכת שמאפשרת יצירה אוטומטית של תשובות בשפה טבעית לשאלות שנכתבו ע"י בני אדם. מערכות שכאלו צריכות להתבסס על בסיס ידע כלשהו – הוא יכול להיות פסקה פשוטה של טקסט, ארכיון של האינטרנט כולו, מאגרי מידע, טבלאות, תמונות ועוד. סוג השאלות שיכולות להשאל יכולות להיות – עובדתיות / לא עובדתיות, שאלות פתוחות או סגורות, פשוטות לעומת compositional ועוד. סוג התשובות שעשויות להדרש יכולות להיות קטע טקסט קצר, פסקה, רשימה, כן/לא ועוד. בהרצאה זו נתמקד בבעיית מענה על שאלות בתצורת reading comprehension - המערכת מקבלת קטע טקסט ומתבקשת לענות על שאלות עובדתיות שהתשובה אליהן מופיעה בטקסט. הבעיה הזו מעניינת כיוון שהיא מתאימה למגוון שימושים, ובנוסף משמשת לביצוע אוולוואציה לכמה מערכות תוכנתיות מבינות שפה אנושית. בנוסף, הרבה בעיות NLP אחרות יכולות להיות מצומצמות לבעיות reading comprehension (למשל information extraction או semantic role labeling).

*Wendy Lehnert 1977: "Since questions can be devised to query **any aspect** of text comprehension, the ability to answer questions is the **strongest possible demonstration** of understanding".*



SQUAD DATASET

דאטאסט supervised גדול מאוד עם 100k שלשות מתויגות של (פסקה, שקלה, תשובה). הפסקאות לקוחות מתוך ויקיפדיה באנגלית, והאורך של כל אחת מהן הוא בד"כ 100-150 מילים. השאלות הן crowd-sourced. התשובות הן סגמנטים קצרים של טקסט (או מספר סגמנטים לא רציפים) מתוך הפסקה המקורית. זאת הגבלה משמעותית – לא על כל שאלה ניתן לענות באופן הזה.

In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity. The main forms of precipitation include drizzle, rain, sleet, snow, graupel and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud. Short, intense periods of rain in scattered locations are called "showers".

What causes precipitation to fall?
gravity

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals to form precipitation?
within a cloud

הדאטאסט הזה נחשב לפופולרי ביותר בתחום ה-reading comprehension, הוא נחשב בתור "almost solved" וה-state-of-the-art כבר עקפו ביצועים אנושיים על הדאטאסט הזה. יש שתי מטריקות שמשמשות כדי להעריך כמה טובות התוצאות של מודל כלשהו על הדאטאסט הזה – הראשונה היא התאמה מדויקת (0 או 1) והשניה היא F1 score (התאמה חלקית). הדאטאסט בנוי ככה שיש לכל שאלה 3 תשובות אמיתיות gold answers (בגלל שיכולות להיות כמה תשובות הגיוניות). כדי לבחון הצלחה של מודל, משווים את התשובה שהמודל החזיר לכל אחת מה-gold answers ולוקחים את הציון הגבוה מבין שלוש האפשרויות, ואז ממצעים על כל הדגימות בדאטאסט כדי לקבל את ציון ה-F1 score הכללי של המודל. ביצועים אנושיים על הדאטאסט הזה מוערכים ב-EM=82.3 (התאמה מושלמת) ו-F1=91.2.

Q: What did Tesla do in December 1878?

A: {left Graz, left Graz, left Graz and severed all relations with his family}

Prediction: {left Graz and severed}

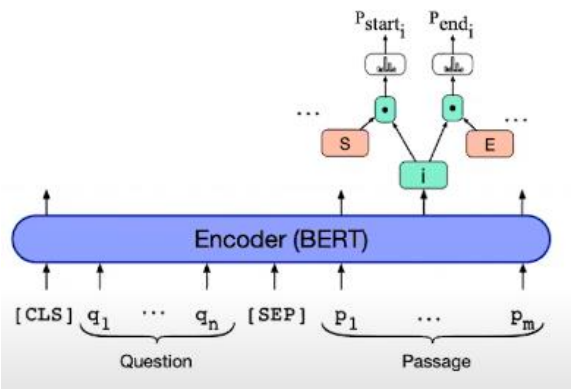
Exact match: max{0, 0, 0} = 0

F1: max{0.67, 0.67, 0.61} = 0.67

READING COMPREHENSION

כדי לבנות מודל שיפתור את בעיית ה-reading comprehension על הדאטאסט של SQuAD נפרמל את הבעיה באופן הבא:

- **הקלט** לבעיה הוא $C = (c_1, c_2, \dots, c_N), Q = (q_1, q_2, \dots, q_M), c_i, q_i \in V$ עם N~100 (אורך הפסקה) M~15 (אורך השאלה).
- **הפלט** של הבעיה הוא $1 \le start \le end \le N$ - מקטע מילים רציף מתוך הפסקה המקורית.

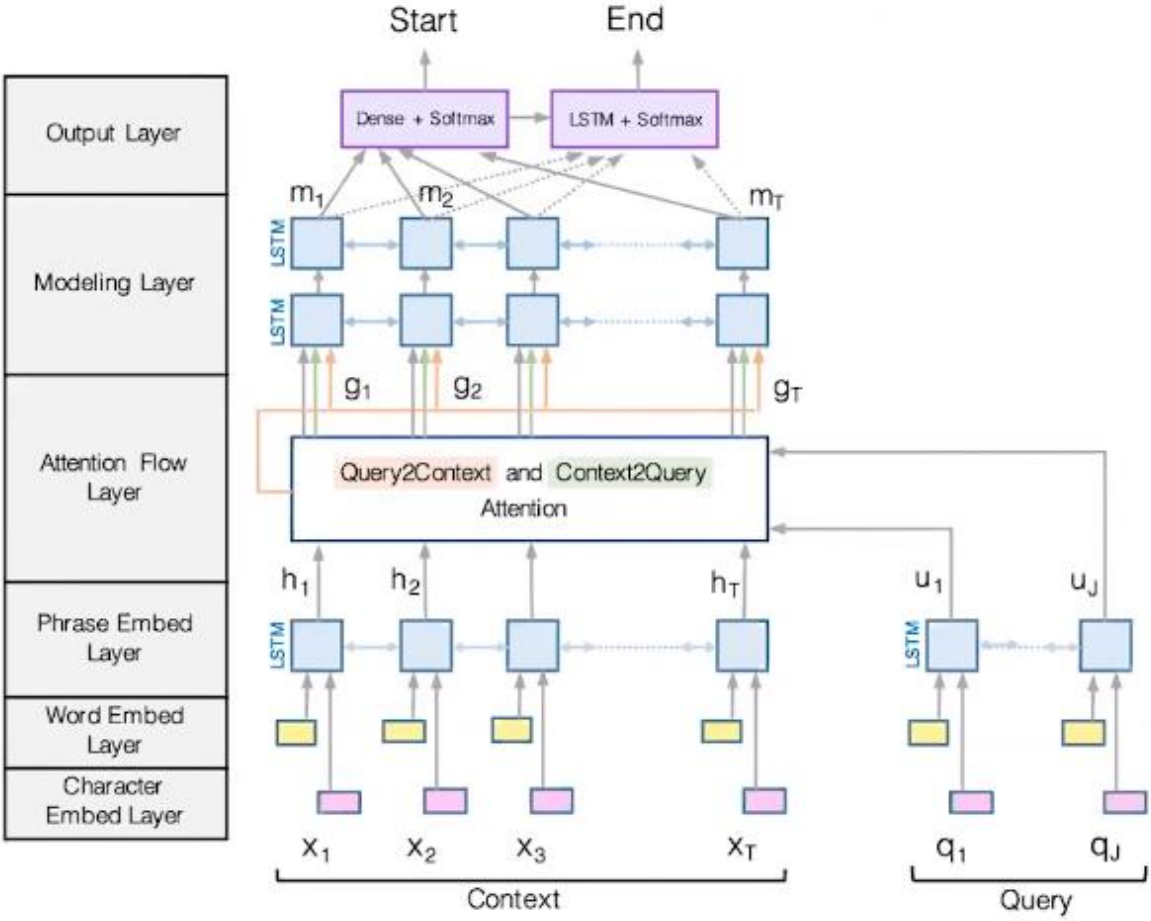


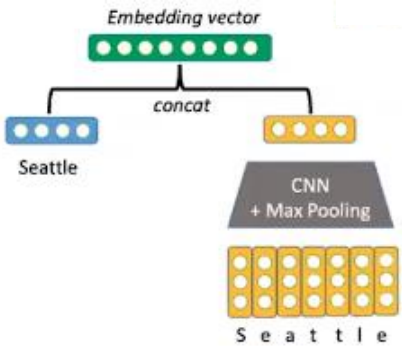
בשנים 2016-18 פורסמו מספר מודלים מבוססי LSTM שפתרו את הבעיה באמצעות attention, והחל משנת 2019 החלו להתפרסם מודלים דמויי-BERT (בתמונה משמאל).

מודלי LSTM ל-READING COMPREHENSION

ראשית, נשים לב שיש דמיון רב בין מודלים כאלו לבין מודל seq2seq שראינו בחלק מוקדם יותר של הקורס – רק שהפעם, במקום לקשר בין משפטי מקור source למילת מטרה target, אנחנו מקשרים בין פסקת הטקסט (הקלט לבעיה) לשאלה (עליה רוצים לענות), כאשר האורכים שלהם מאוד לא מאוזנים (הפסקה משמעותית יותר ארוכה מהשאלה). כמו כן, במקרה של reading comprehension אנחנו צריכים למדל אילו מילים מתוך הפסקה הן הכי רלוונטיות לשאלה (ובפרט, לאילו מילים מתוך השאלה). לכן attention היא מילת המפתח כאן – בדומה לאופן שבו במודל seq2seq ניתן לזהות מהן המילים במשפט המקור source שהכי רלוונטיות למילת המטרה target הנוכחית. אנחנו לא צריכים autoregressive decoder שיודע לייצר את משפט המטרה target מילה אחרי מילה – אנחנו צריכים לאמן שני קלסיפיירים שיחזו את מילת ההתחלה ומילת הסיום שמייצגים את הגבולות של התשובה בתוך הפסקה המקורית.

נתמקד במודל **BiDAF – Bidirectional Attention Flow Model** שפורסם ב-2017 (לפני BERT).





שלוש השכבות הראשונות (התחתונות) של המודל הן שכבות ה-**encoder**. בשכבות אלו משתמשים בשרשור concatenation של word-embeddings (ספציפית, GloVe) ו-character-embeddings (שנוצרו מרשת CNN) עבור כל מילה בפסקה (ה-context) ועבור כל מילה בשאלה (ה-query):

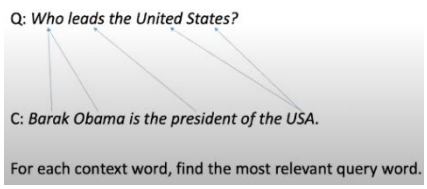
- כאשר המילה היא מה-context - $e(c_i) = f([\text{GloVe}\ (c_i); \text{charEmb}\ (c_i)])$
- כאשר המילה היא מה-query - $e(q_i) = f([\text{GloVe}\ (q_i); \text{charEmb}\ (q_i)])$

כאשר f היא רשת נוספת (שלא הורחב עליה בהרצאה, אך המאמר מכיל את הפרטים המלאים).

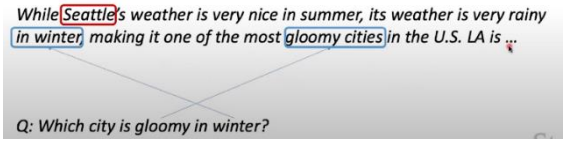
לאחר מכן, משתמשים בשני רשתות LSTM דו-כיווניות (בנפרד) כדי להפיק contextual embedding הן עבור ה-context והן עבור ה-query (שהם ממימד H מוגדר מראש כלשהו), ואז נשרשר את שני הכיוונים לכדי וקטור אחד עבור הפסקה ווקטור אחד עבור השאלה (שהם ממימד 2H כל אחד):

$$\vec{\mathbf{c}}_i = \text{LSTM}\left(\vec{\mathbf{c}}_{i-1}, e(c_i)\right) \in \mathbb{R}^H \qquad \vec{\mathbf{q}}_i = \text{LSTM}\left(\vec{\mathbf{q}}_{i-1}, e(q_i)\right) \in \mathbb{R}^H$$
$$\overleftarrow{\mathbf{c}}_i = \text{LSTM}\left(\overleftarrow{\mathbf{c}}_{i+1}, e(c_i)\right) \in \mathbb{R}^H \qquad \overleftarrow{\mathbf{q}}_i = \text{LSTM}\left(\overleftarrow{\mathbf{q}}_{i+1}, e(q_i)\right) \in \mathbb{R}^H$$
$$\mathbf{c}_i = [\vec{\mathbf{c}}_i; \overleftarrow{\mathbf{c}}_i] \in \mathbb{R}^{2H} \qquad \mathbf{q}_i = [\vec{\mathbf{q}}_i; \overleftarrow{\mathbf{q}}_i] \in \mathbb{R}^{2H}$$

השכבה הרביעית היא שכבת ה-**attention**. במאמר הם מציעים שתי שיטות לביצוע attention:



- <u>שיטה ראשונה – context-to-query attention</u>: לכל מילת context, נבחר מהן המילים הכי רלוונטיות עבורה מתוך מילות ה-query. זו למעשה משימת alignment.



- <u>שיטה שניה – query-to-context attention</u>: בשיטה זו בוחרים את מילות ה-context שהכי רלוונטיות עבור מילת query ספציפית כלשהי.

במודל BiDAF משתמשים בשתי שיטות ה-attention הנ"ל באופן הבא – ראשית, מחשבים similarity score לכל זוג של מילת context ומילת query (משתמשים בזוג ה-embeddings שקיבלנו משכבת ה-encoder $(\mathbf{c}_i, \mathbf{q}_j)$) באופן הבא –

$$S_{i,j} = \mathbf{w}_{\text{sim}}^{\top}[\mathbf{c}_i; \mathbf{q}_j; \mathbf{c}_i \odot \mathbf{q}_j] \in \mathbb{R}\mathbf{w}_{\text{sim}} \in \mathbb{R}^{6H}$$

כלומר, משרשרים את הייצוג של מילת ה-context, את הייצוג של מילת ה-query, ואת המכפלה element-wise ביניהם, וכופלים את הוקטור המשורשר בוקטור משקולות (שנלמד). לאחר מכן, מחשבים את ה-context-to-query attention (מי המילים ב-query שהכי רלוונטיות למילת ה-context c_i):

$$\alpha_{i,j} = \text{softmax}_j(S_{i,j}) \in \mathbb{R} \qquad \mathbf{a}_i = \sum_{j=1}^{M} \alpha_{i,j}\mathbf{q}_j \in \mathbb{R}^{2H}$$

כלומר, לוקחים את ה-similarity score בין מילת ה-context (הקבועה אותה אנחנו רוצים לבחון) לבין כל אחת ממילות ה-query, מעבירים אותן דרך softmax, ואז משרשרים עם וקטור ה-embedding של אותה מילת query $\mathbf{q}_j$ – וסוכמים את התוצאה של החישוב הזה עבור כל אחת ממילות ה-query.

לאחר מכן, נחשב את ה-query-to-context attention (אלו מילות context הכי רלוונטיות עבור מילת query כלשהי $\mathbf{q}_j$) באופן הבא:

$$\beta_i = \text{softmax}_i\left(\max_{j=1}^{M}(S_{i,j})\right) \in \mathbb{R}^N \qquad \mathbf{b}_i = \sum_{i=1}^{N} \beta_i \mathbf{c}_i \in \mathbb{R}^{2H}$$

כלומר, מחשבים את ה-similarity score עבור השילוב בין $\mathbf{q}_j$ עם כל אחת ממילות ה-*context*, לוקחים את התוצאה הגבוהה ביותר, מעבירים אותה דרך *softmax*, משרשרים אותה כל פעם עם וקטור ייצוג של מילת *context* אחרת, וסוכמים את כולם.

הפלט הסופי של שכבת ה-*attention* הוא השרשור הבא: $\mathbf{g}_i = [\mathbf{c}_i; \mathbf{a}_i; \mathbf{c}_i \odot \mathbf{a}_i; \mathbf{c}_i \odot \mathbf{b}_i] \in \mathbb{R}^{8H}$

שתי השכבות העליונות הן שכבות ***modeling & output.***

- שכבת ה-*modeling* מקבלת את הייצוג $\mathbf{g}_i$ משכבת ה-*attention* ומכניסה אותו לשתי שכבות נוספות של *LSTM* דו-כיווני. כלומר, בעוד שכבת ה-*attention* ממפה את הקשרים בין ה-*query* וה-*context*, שכבת ה-*modeling* ממפה אינטראקציות בין מילות ה-*context* עצמן - $\mathbf{m}_i = \text{BiLSTM}(\mathbf{g}_i) \in \mathbb{R}^{2H}$ (זה מחושב עבור כל אחת ממילות ה-*context* בפסקה).
- לאחר מכן, הפלט של שכבת ה-*modeling* $\mathbf{m}_i$ מוזן לשכבת ה-*output*, שמורכבת משני קלסיפיירים שחוזים את המיקום של מילת ההתחלה ומילת הסיום שתוחמות את התשובה בתוך הפסקה המקורית. כדי לעשות את זה, בהתחלה משרשרים בין הפלט של שכבת ה-*attention* לפלט של שכבת ה-*modeling* (עבור מילת ההתחלה זה מתבצע באופן ישיר, עבור מילת הסיום זה מתבצע לאחר העברה של $\mathbf{m}_i$ ברשת *LSTM* דו כיוונית נוספת $(\mathbf{m}_i' = \text{BiLSTM}(\mathbf{m}_i) \in \mathbb{R}^{2H}$), כופלים בוקטור משקולות (נלמד) ולבסוף מעבירים דרך *softmax* -

$$p_{\text{start}} = \text{softmax}\left(\mathbf{w}_{\text{start}}^{\top}[\mathbf{g}_i; \mathbf{m}_i]\right) \qquad p_{\text{end}} = \text{softmax}\left(\mathbf{w}_{\text{end}}^{\top}[\mathbf{g}_i; \mathbf{m}_i']\right)$$

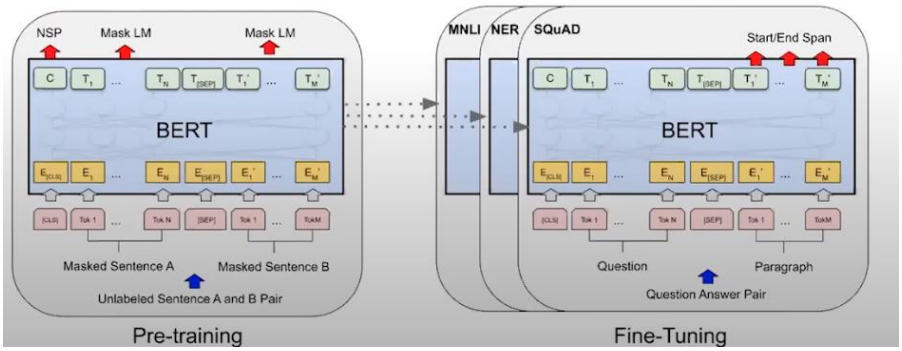
התוצאה הסופית שמתקבלת היא $\mathbf{w}_{\text{start}}, \mathbf{w}_{\text{end}} \in \mathbb{R}^{10H}$.

ה-*training loss* הסופי מוגדר ע"י $\mathcal{L} = -\log p_{\text{start}}(s^*) - \log p_{\text{end}}(e^*)$ כאשר s^*, e^* הם מיקומי ההתחלה והסיום של ה- *gold answer* עבור הדגימה.

המודל הזה השיג ציון *F1* של 77.3 על *SQuAD*. הסרה של *context-to-query attention* הורידה את הביצועים בכ-10%, הסרה של *query-to-context attention* הורידה את הביצועים בכ-4%, והסרה של *character embeddings* הורידה את הביצועים בכ-2%. נכון לזמן פרסום המאמר (2017) זה היה המודל עם הביצועים הטובים ביותר על הדאטאסט הזה.

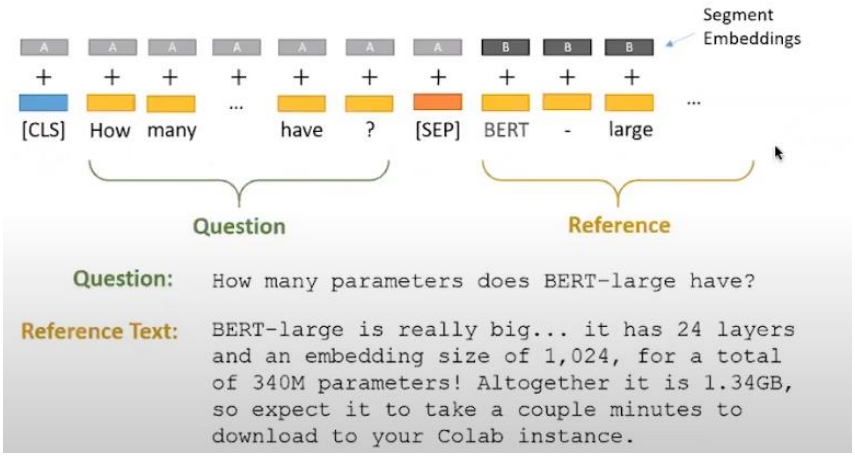
שימוש ב-BERT ל-READING COMPREHENSION

כפי שהזכרנו בחלק הקודם של הקורס, *BERT* הוא טרנספורמר דו-כיווני מסוג *encoder* שאומן על כמויות גדולות של דאטא (ויקיפדיה + *BooksCorpus*) באמצעות *MLM* (*Masked language model*) ו-*NSP* (*Next sentence prediction*). מדובר במודל בעל 12 שכבות ומעל 110 מיליון פרמטרים בגרסה הבסיסית שלו, ו-24 שכבות עם 330 מיליון פרמטרים בגרסה הגדולה שלו.



כיצד אפשר להשתמש ב-*BERT* למשימת ה-*reading comprehension*? ראשית, נפרמל את הבעיה באופן הבא:

- השאלה *question / query* תסומן בתור *segment A*
- הפסקה *passage / context* תסומן בתור *segment B*
- התשובה *answer* תיוצג ע"י בחירת שני אינדקסים (התחלה וסיום) מתוך סגמנט *B*



האופן שבו הדאטא מוזן למודל הוא להתחיל בטוקן *CLS*, אחריו להכניס את השאלה, אחריה טוקן *SEP*, ולבסוף הפסקה. המודל ייצר *sentence embeddings* הן עבור השאלה (סגמנט *A*) והן עבור הפסקה (סגמנט *B*). ה-*loss* לאימון השכבה האחרונה יוגדר באופן הבא - $\mathcal{L} = -\log p_{\text{start}}(s^*) - \log p_{\text{end}}(e^*)$ עבור

$$p_{\text{start}}(i) = \text{softmax}_i\left(\mathbf{w}_{\text{start}}^{\top}\mathbf{h}_i\right)$$
$$p_{\text{end}}(i) = \text{softmax}_i\left(\mathbf{w}_{\text{end}}^{\top}\mathbf{h}_i\right)$$

כאשר $\mathbf{h}_i$ הוא ה-*hidden vector* שמודל *BERT* החזיר עבור הטוקן c_i.

במודל הזה, כל הפרמטרים של *BERT*, ביחד עם שני הפרמטרים החדשים שיצרנו $\mathbf{h}_{\text{start}}, \mathbf{h}_{\text{end}}$ (שני וקטורים מאורך 768 כל אחד, בסה"כ 1,536 פרמטרים נוספים על 110 מיליון הפרמטרים שיש ל-*BERT* עצמו) משמשים לאופטימיזציה של פונקציית ה-*loss*.

המודל מחזיר תוצאות מדהימות (ברמת בן אנוש או אפילו טוב יותר), כאשר החלפה של *BERT* במודלים *pretrained* גדולים ואיכותיים יותר שיפרה את התוצאות אפילו יותר (הרבה מעבר לרמה של בן אנוש).

	F1	EM
Human performance	91.2*	82.3*
BiDAF	77.3	67.7
BERT-base	88.5	80.8
BERT-large	90.9	84.1
XLNet	94.5	89.0
RoBERTa	94.6	88.9
ALBERT	94.8	89.3

השוואה בין BIDAF למודל מבוסס BERT

- ל-*BERT* יש משמעותית יותר פרמטרים (110 או 330 מיליון לעומת *BiDAF* עם 2.5 מיליון)
- *BiDAF* בנוי על מספר שכבות של *LSTM* דו כיווני, בעוד *BERT* בנוי על טרנספורמרים, מה שגורם לכך שיותר קל לאמן אותו באופן מקבילי (כיוון שאין בארכיטקטורה שלו אלמנט של *recurrence*).
- *BERT* הוא *pretrained* לעומת *BiDAF* שרק משתמש ב-*GloVE* ושאר הפרמטרים שלו נלמדים באופן *supervised* ספציפית עבור הבעיה הזו.

אך המודלים האלו גם דומים במובן כלשהו – בעוד *BiDAF* (ומודלים אחרים בארכיטקטורה דומה) מנסה למפות את האינטראקציה בין הפסקה והשאלה, *BERT* משתמש ב-*self-attention* על השרשור של השאלה והפסקה, מה ששקול לנוסחא הבאה –

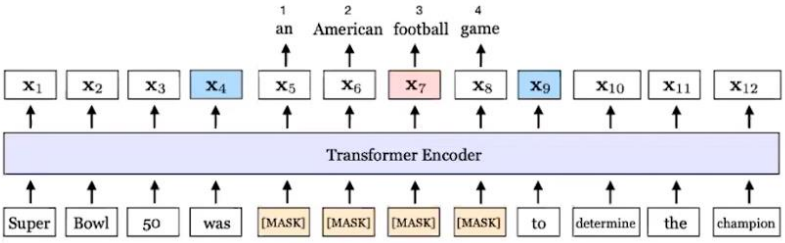
$$\text{attention}(\mathrm{P},\mathrm{P}) + \text{attention}(\mathrm{P},\mathrm{Q}) + \text{attention}(\mathrm{Q},\mathrm{P}) + \text{attention}(\mathrm{Q},\mathrm{Q})$$

כלומר, *BERT* למעשה "תופס" *captures* את אותו הדבר ש-*BiDAF* מנסה למדל באופן מפורש.

בחירת PRE-TRAINING OBJECTIVES טובים יותר לבעית READING COMPREHENSION

ניקח לדוגמא את הפונקציה הבאה (שהוצגה במאמר על *SpanBERT* ב-2020) –

$$\mathcal{L}(\text{football}) = \mathcal{L}_{\text{MLM}}(\text{football}) + \mathcal{L}_{\text{SBO}}(\text{football})$$
$$= -\log P(\text{football} \mid \mathbf{x}_7) - \log P(\text{football} \mid \mathbf{x}_4, \mathbf{x}_9, \mathbf{p}_3)$$



הפונקציה הזו משתמשת בשני רעיונות עיקריים:

- *Masking* של רצפי מילים רציפים *continuous spans* במקום להחליף 15% מהמילים במילים רנדומליות
- שימוש בשתי *end points* של רצף המילים כדי לחזות את כל המילים ה-*masked* ביניהן, מה ששקול ללדחוס את האינפורמציה של כל רצף המילים לשתי המילים שתוחמות אותו - $\mathbf{y}_i = f(\mathbf{x}_{s-1}, \mathbf{x}_{e+1}, \mathbf{p}_{i-s+1})$

שימוש בשיטה כזו ל-*pretraining* שיפר משמעותית (בכמה אחוזים שלמים) את הביצועים של מודלים שהשתמשו במודל ה-*pretrained* כדי לבצע *finetuning* לבעיית ה-*reading comprehension*.

חלקים בבעיית ה-READING COMPREHENSION שעדיין לא פתורים

זה שמודלים עוקפים תוצאות אנושיות על בעיית ה-*reading comprehension*, לא אומר שהיא נחשבת לפתורה – מודלים קיימים עדיין טועים מאוד ב-*adversarial examples* שלקוחות מ-*out-of-domain distributions*. למשל, בדוגמא הבאה, המודל בחר כתשובה (בכחול) משפט שמאוד דומה לשאלה, אך התשובה היא שגויה (כיוון שהמודל סימן משפט שמתייחס לטורניר הלא נכון):

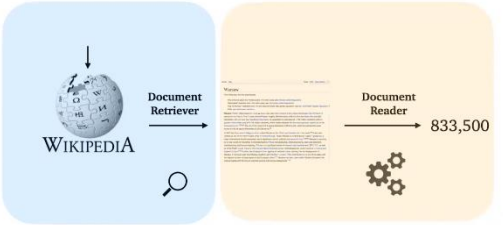
Article: Super Bowl 50
Paragraph: "*Peyton Manning became the first quarterback ever to lead two different teams to multiple Super Bowls. He is also the oldest quarterback ever to play in a Super Bowl at age 39. The past record was held by John Elway, who led the Broncos to victory in Super Bowl XXXIII at age 38 and is currently Denver's Executive Vice President of Football Operations and General Manager. Quarterback Jeff Dean had jersey number 37 in Champ Bowl XXXIV.*"
Question: "*What is the name of the quarterback who was 38 in Super Bowl XXXIII?*"
Original Prediction: John Elway
Prediction under adversary: Jeff Dean

כמו כן, מודלים שאומנו על דאטאסט ספציפי מתקשים להכליל לדאטאסטים אחרים –

		Evaluated on				
		SQuAD	TriviaQA	NQ	QuAC	NewsQA
Fine-tuned on	SQuAD	**75.6**	46.7	48.7	20.2	41.1
	TriviaQA	49.8	**58.7**	42.1	20.4	10.5
	NQ	53.5	46.3	**73.5**	21.6	24.7
	QuAC	39.4	33.1	33.8	**33.3**	13.8
	NewsQA	52.1	38.4	41.7	20.4	**60.1**

OPEN-DOMAIN QUESTION ANSWERING

דיברנו עד כה רק על מענה על שאלות באמצעות *reading comprehension* – אך משימה הרבה יותר מעניינת, מאתגרת לפתרון ושימושית היא משימת *open-domain question answering*, בה המודל נדרש להשתמש במאגר מידע כללי וגדול על מנת לענות על שאלות. כלומר, אנחנו לא מניחים שיש פסקה ספציפית שמסופקת למודל והיא המקור לתשובה. במקום זה, אנחנו מניחים רק שיש לו גישה למאגר גדול של מסמכים (למשל, ויקיפדיה) שאנחנו לא יודעים איפה התשובה ממוקמת בתוכו, והמטרה של המודל היא להיות מסוגל לענות על כל שאלה פתוחה תוך התבססות על המידע במאגר. חשוב לעשות את האבחנה בין מודלים כאלו לבין מודלים שהם <u>*closed-domain*</u> (שמתמודדים עם שאלות המשויכות לדומיין ספציפי – למשל רפואה, תמיכה טכנית ועוד).



דרך אחת לפתור את הבעיה היא באמצעות ***retriever-reader framework*** – ה-*retriever* מקבל את השאלה ומנסה למצוא מספר מצומצם של מסמכים מתוך הדאטאסט שרלוונטיים לשאלה. לאחר מכן, ה-*reader* עובר על המסמכים שזוהו ע"י ה-*retriever* ומחפש בהם את התשובה לשאלה.

פורמלית, נגדיר את הבעיה באופן הבא:

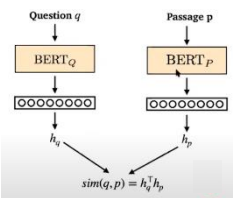
Input: a large collection of documents $\mathscr{D} = D_1, D_2, \ldots, D_N$ and Q
Output: an answer string A

Retriever: $f(\mathscr{D}, Q) \longrightarrow P_1, \ldots, P_K$ K is pre-defined (e.g., 100)
Reader: $g(Q, \{P_1, \ldots, P_K\}) \longrightarrow A$ A reading comprehension problem!

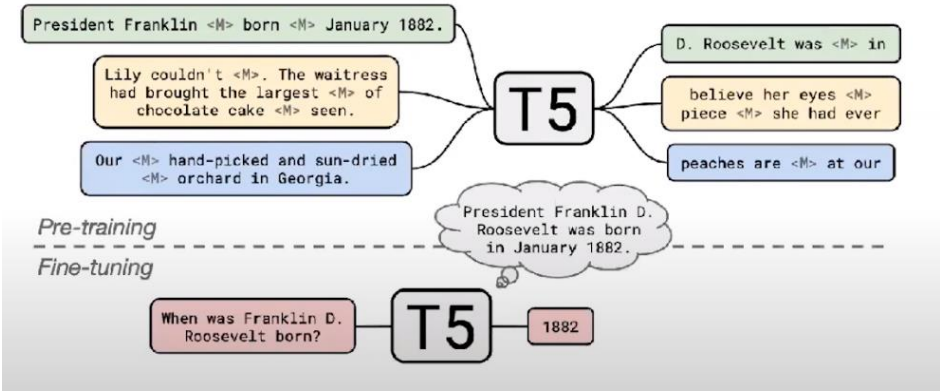
לדוגמא, במודל *DrQA* שפורסם ב-2017, ה-*retriever* הוא *TF-IDF* וה-*Reader* הוא מודל *reading comprehension*.

איך מאמנים את ה-*retriever*?

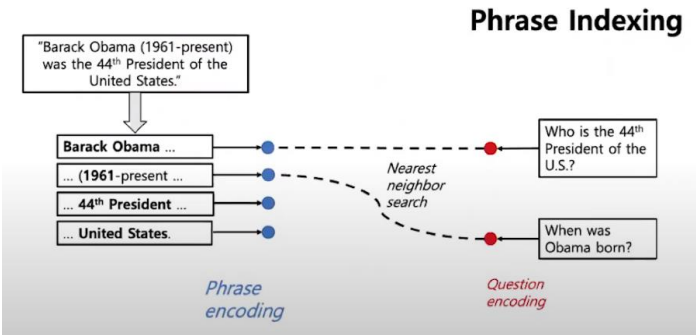
- אפשר לאמן את ה-*retriever* ואת ה-*reader* באופן משותף, למשל – כל פסקה של טקסט יכולה להיות מקודדת לוקטור באמצעות *BERT*, וה-*retriever score* יכולה להיות מוגדרת באמצעות המכפלה הפנימית בין הייצוג של השאלה לבין הייצוג של הפסקה (הרעיון הזה הוצג לראשונה ב-2019).
- אפשר להשתמש ב-*DPR – Dense Passage Retrieval*, כלומר לאמן את ה-*retriever* באמצעות זוגות של שאלות ותשובות ברשת סיאמית (הרעיון הזה הוצג לראשונה ב-2020) – ראו בתמונה משמאל.



בנוסף, מאמר מ-2020 הציג מודל ל-*open-domain QA* ללא שימוש בשלב *retriever* מפורש – הם השתמשו במודל *pretrained* מאוד גדול, ועשו לו *finetuning* לבעיה הזו. המאמר הראה שמסתבר שהפרמטרים של מודל השפה המאומן מכילים ייצוג די נאמן של כמות מאוד גדולה של מידע, ואפשר להשתמש בו כדי לענות על שאלות.



כמו כן, מאמר אחר מ-2020 הראה שאפשר לקודד כמות עצומה של ביטויים (למשל 60 מיליארד ביטויים בדאטאסט של ויקיפדיה) תוך שימוש בוקטורים דחוסים *dense* ואז רק להשתמש בחיפוש השכן הקרוב ביותר, מה שמייתר את השימוש במודל *BERT* להסקה בזמן אמת, ולמעשה יוצר מודל *QA* שאין לו חלק של *reader*.

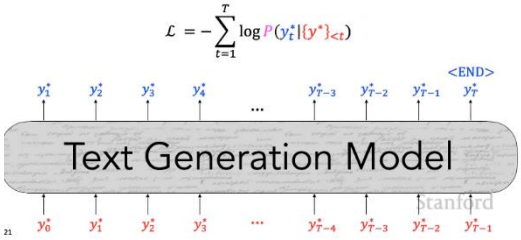


NATURAL LANGUAGE GENERATION

NLG – natural language generation הוא תת תחום של *NLP*, שהמשימה בו היא לבנות מערכות שמייצרות באופנט אוטומטי טקסט כתוב בשפה טבעית שהוא קוהרנטי ושימושי. זה יכול להיות שימושי למגוון שימושים – למשל תקצור ותמצות, דיאלוג (כמו *AI assistants*). תחום חדש יחסית ונוסף הוא *data to text generation* – משימה בה המודל מקבל דאטא (למשל טבלה, גרף ידע ועוד) וצריך להפיק סיכום של פרטי המידע המעניינים מתוכו. תחום נוסף הוא *visual description* – משימה בה מודל מקבל תמונה ומתבקש לייצר תיאור מילולי למתרחש בה. רעיון נוסף לשימוש הוא שימוש יצירתי – כתיבת פוסטים, כתיבת קוד, ניסוח מאמרים, כתיבת סיפורים ושירה וכו'.

במודלי יצירת שפה שהם אוטורגרסיביים *autoregressive*, בכל צעד t, המודל מקבל רצף של טוקנים של טקסט (מילים) $\{y\}$ באורך קטן מ-t, ומוציא כפלט טוקן חדש $\hat{y}_t$.

כלומר, בכל שלב t, המודל מחשב וקטור של *scores* עבור כל טוקן באוצר המילים - $S = f(\{y_{<t}\}, \theta) \in \mathbb{R}^V$ כאשר f הוא המודל שלנו. לאחר מכן, מחשבים את ההתפלגות P מעל כל המילים באוצר המילים $w \in V$ באמצעות הנוסחא הבאה: $P(y_t = w \mid \{y_{<t}\}) = \frac{\exp(s_w)}{\sum_{w' \in V} \exp\left(s_{w'}\right)}$ (ששקולה ל-*softmax*).



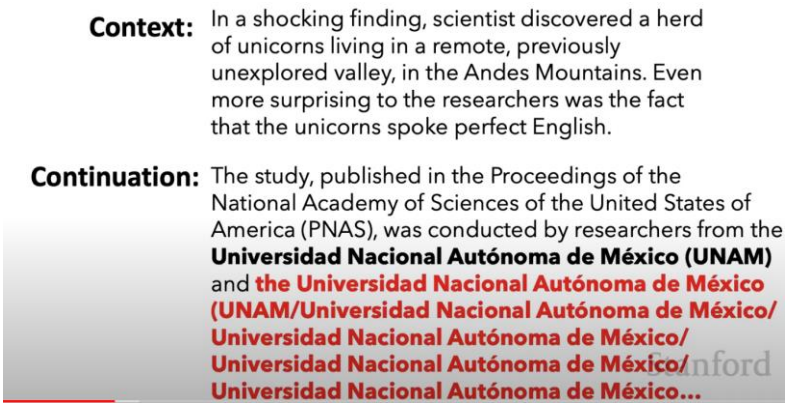
לאחר אימון המודל, בשלב ה-*inference*, אלגוריתם ה-*decoding* (שלעתים מכונה *teacher forcing*) מגדיר פונקציה g שבוחרת טוקן מתוך ההתפלגות שנוצרה, כלומר: $\hat{y}_t = g\big(P(y_t \mid \{y_{<t}\})\big)$. אנחנו מאמנים את המודל למזער את ה-*negative log-likelihood* בפרדיקציה של הטוקן הבא במשפט, כלומר: $\mathcal{L}_t = -\log P(y_t^* \mid \{y_{<t}^*\})$ כאשר סוכמים את $\mathcal{L}_t$ לכל המילים ברצף. חשוב לשים לב שזאת פשוט משימת קלסיפיקציה כאשר כל מילה w מאוצר המילים היא *class*. ה-*label* של כל צעד הוא המילה y_t^* מרצף האימון. הטוקן הזה לעתים מכונה "*gold token*" או "*ground truth token*".

שלב ה-DECODING

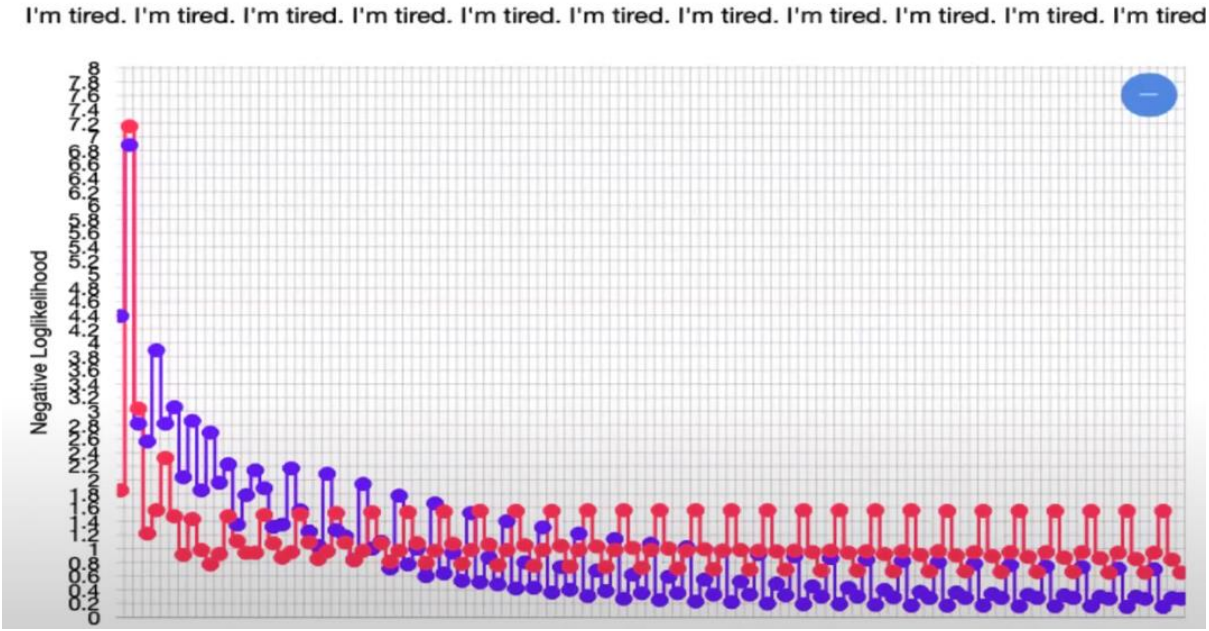
כפי שהוזכר קודם, אלגוריתם ה-*decoding* מגדיר פונקציה g שבוחרת טוקן כלשהו מבין ההתפלגות שנוצרה על אוצר המילים כולו. בחלק קודם של הקורס, ראינו שימוש באלגוריתמים חמדנים *greedy*, למשל –

- אלגוריתם *Argmax*, שבוחר את הטוקן עם ההסתברות הגבוה ביותר מתוך ההתפלגות באופן הבא - $\hat{y}_t = \underset{w \in V}{\operatorname{argmax}} P(y_t = w \mid y_{<t})$.
- אלגוריתם *beam search* שהוצג בפרק על *machine translation* – זהו אלגוריתם חמדן, אך בהשוואה ל-*argmax* הוא שוקל מספר גדול יותר של "מועמדים".

אמנם אלגוריתמים חמדניים השיגו ביצועים טובים בבעיות תרגום או תמצות, הם נוטים להיות בעייתיים עבור סוגים אחרים של בעיות ייצור טקסט כיוון שהם נוטים להכנס למצב שבו הם חוזרים על עצמם שוב ושוב:

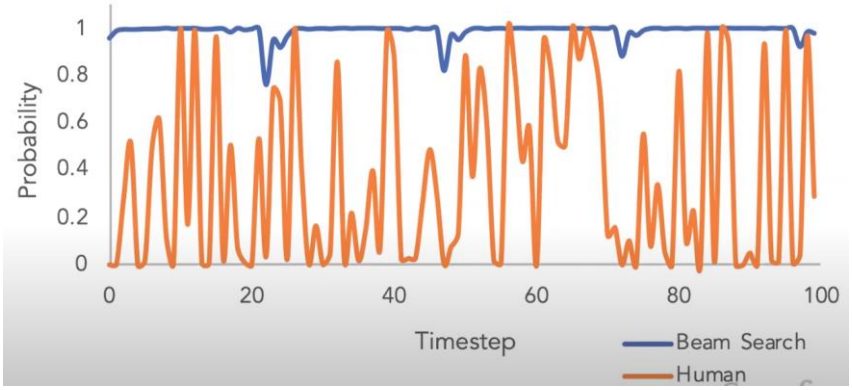


זה קורה גם במודלים כמו *LSTM* וגם במודלים מבוססי טרנספורמרים – למשל ניתן לראות בגרף הבא שעבור מודלים משני הסוגים האלו (אם כי זה קורה מעט פחות ב-*LSTM*), ה-*negative log likelihood* יורד עבור חיזוי הטוקנים במשפט מאוד מאוד חזרתי (כלומר המודלים חושבים שזה מאוד הגיוני להמשיך לחזות שוב ושוב את אותן המילים):



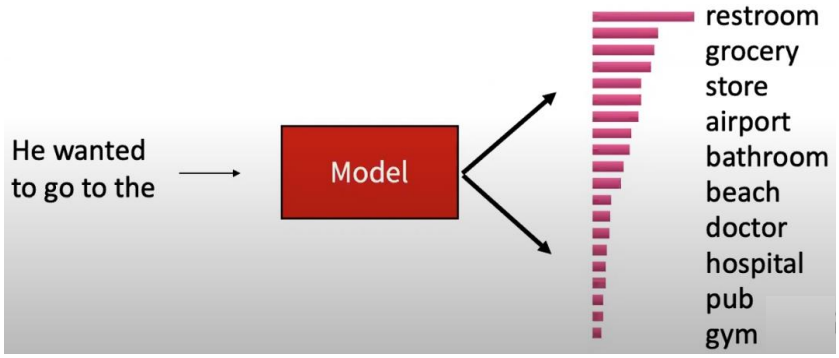
כדי להפחית חזרתיות, הפתרון הפשוט הוא פתרון יוריסטי שמונע באופן מפורש מהמודל לחזור על *n-grams*. פתרון מורכב יותר יהיה למזער את ה-*embedding distance* בין משפטים עוקבים (הוצע ב-2018), אם כי הפתרון הזה לא עוזר עם בעיה של חזרתיות בתוך אותו המשפט. פתרון אחר הוא לוודא שה-*loss* מתכנס, מה שמונע ממכניזם ה-*attention* להפנות את כל הקשב לאותן מילים. פתרון עדכני יותר (פורסם ב-2020) הוא "*unlikelihood objectives*" ש"מענישים" כל "דור" של טוקנים שכבר נראו בעבר.

לפני שמנסים למצוא פתרון טכני לבעיית החזרתיות, חשוב קודם כל לחשוב על זה שהשימוש באלגוריתמים חמדניים הוא לא מאוד הגיוני מלכתחילה. הסיבה לכך היא שהאופן שבו בני אדם מדברים אינו תואם את הרעיון של "לבחור את המילה הסבירה ביותר בכל פעם". בגרף הבא ניתן לראות את ההבדל בהסתברויות של המילים שבחר בן אדם לבין המילים שבחר מודל מבוסס אלגוריתם חמדני –

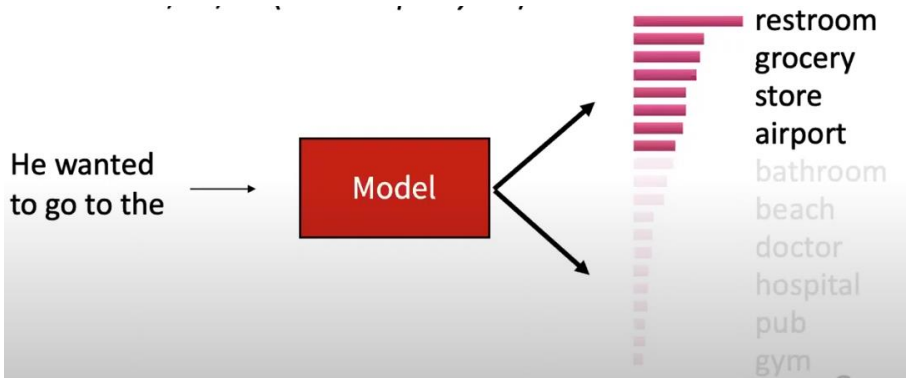


TOP-K SAMPLING

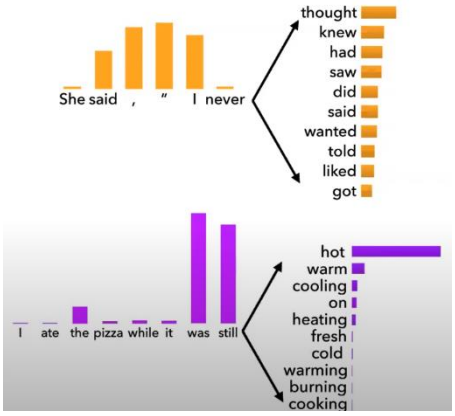
מה שנרצה לעשות הוא כמובן לנסות ליצור מודל שתואם את חוסר הוודאות שקיים בדיבור אנושי (שהוא לא דטרמיניסטי כמו שמציעים האלגוריתמים החמדניים). לכן, נרצה להוסיף אלמנט רנדומי למודלים. שיטה נפוצה לעשות את זה היא באמצעות *sampling* מתוך ההתפלגות של הטוקנים באופן אקראי תוך התחשבות ב"משקל" שניתן לכל טוקן $\hat{y}_t \sim P(y_t = w \mid \{y\}_{<t})$, כך שיותר סביר שנדגום טוקנים עם הסתברות גבוהה אך אנחנו עשויים לדגום כל טוקן אפשרי.



זה יוצר מודלים עם יורת גמישות ואפשרויות בטוקנים שהם מייצרים, אבל "הפיצ'ר הוא גם באג", כלומר – העובדה שכל טוקן באוצר המילים הוא אופציה לבחירה בכל שלב, גורמת לכך שגם אם רוב ה-*probability mass* בהתפלגות מפוזרת על מספר מאוד מצומצם של אפשרויות, ה"זנב" של ההתפלגות הזו יכול להיות מאוד ארוך (ולהכיל הרבה מאוד טוקנים שככה"נ הם לחלוטין לא רלוונטיים בהקשר הנוכחי). הפתרון לכך הוא-***top-k sampling***, שיטה שבה דוגמים בכל פעם רק מתוך k (היפרפרמטר מוגדר מראש, ערכים נפוצים הם 5, 10 או 20) הטוקנים עם ההסתברות הכי גבוהה מתוך ההתפלגות (ובכך "קוטמים את הזנב" של ההתפלגות):



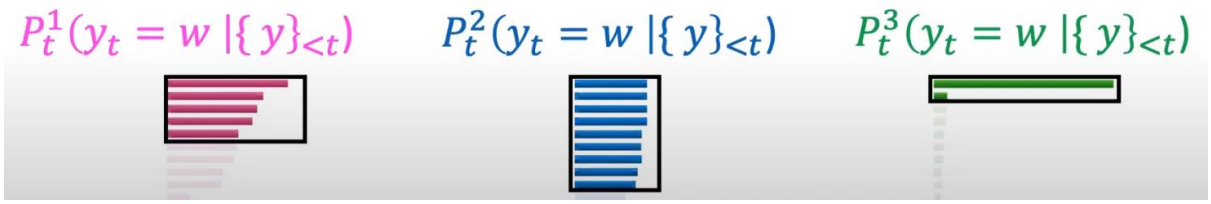
אם נגדיל את k נקבל טוקנים שהם יותר מגוונים (אך עם יותר סיכון לקבל טוקנים שלא רלוונטיים בכלל), אם נקטין את k נקבל טוקנים שהם יותר גנריים ופחות מגוונים (ונקבל מודל יותר "בטוח" שפחות יחזה שטויות).



הבעיה עם שיטה כזו (כלומר עם קטימה של הזנב לפי מספר קבוע כלשהו k של טוקנים) היא שהיא יכולה לגרום לכך שהזנב יקטם יותר מידי **מהר**, למשל המשפט *She said, ___ I never* יכול להכיל הרבה יותר אפשרויות מלמשל ה*k=10* אפשרויות שהמודל יוכל לבחור מביניהן. מהצד השני, השיטה הזו יכולה גם לגרום לכך שהזנב יקטן יותר מידי **לאט**, למשל המשפט הבא - *I ate the pizza while it ____ was still* יכול להסתיים בכמה טוקנים מאוד ספציפיים (שכמה מהם משמעותית יותר סבירים כמו *hot, warm*), כנראה פחות מה-*k=10* שהמודל יכול לבחור מביניהם, לכן יכול להיות שהמודל יבחר בטוקן עם הסתברות מאוד נמוכה שלא באמת הגיוני בהקשר. פתרון לבעיה הזו הוא אלגוריתם *top-p (nucleus) sampling*.

TOP-P SAMPLING

ההתפלגויות מתוכן אנחנו דוגמים הן דינמיות, וקשה לבחור k קבוע כלשהו שיאפשר לנו לקטום את הזנב של ההתפלגות שיתאים בכל הקשר אפשרי. כשההתפלגות היא "שטוחה" יותר, אם נגביל את k יותר מידי נסיר יותר מידי אפשרויות מוצלחות. כשההתפלגות "*peaky*" יותר, אם לא נגביל את k מספיק נאפשר למודל לבחור גם מבין אפשרויות שהן לחלוטין לא רלוונטיות. לכן, הפתרון הוא *top-p sampling* – שיטה בה דוגמים מבין הטוקנים שמשתייכים ל-*top p cumulative mass* (איפה שהמסה ההסתברותית מרוכזת). זה יגרום לכך שמספר הטוקנים שנדגום מתוכו ישתנה כתלות בכמה ההתפלגות שטוחה – ככל שהיא יותר שטוחה (קרובה ליוניפורמית) נבחר מבין כמות גדולה יותר של טוקנים (כי המסה ההסתברותית תתפזר על יותר טוקנים).



SOFTMAX TEMPERATURE

נזכר כי בזמן t של המודל, המודל מחשב את ההתפלגות P_t באמצעות הפעלת *softmax* על וקטור התוצאות s באופן הבא –

$$P_t(y_t = w) = \frac{\exp(S_w)}{\sum_{w' \in V} \exp(S_{w'})}$$

נוכל להפעיל **היפרפרמטר של טמפרטורה** המסומן τ על ההתפלגות כדי לאזן את P_t באופן הבא: $P_t(y_t = w) = \frac{\exp(S_w/\tau)}{\sum_{w' \in V} \exp(S_{w'}/\tau)}$.

- אם נעלה את הטמפרטורה $\tau > 1$, ההתפלגות תהפוך להיות יותר "שטוחה" / יוניפורמית, ונקבל פלט יותר מגוון (כיוון שההתפלגות מתפרסת על יותר מילים).
- אם נוריד את הטמפרטורה $\tau < 1$, ההתפלגות תהפוך להיות יותר *spikey*, ונקבל פלט פחות מגוון (כיוון שההפתלגות תתרכז רק סביב ה-*top words*).

השיטה הזו היא לא אלגוריתם *decoding* בפני עצמו, אלא שיטה שאפשר להשמיש <u>ביחד</u> עם אלגוריתם *decoding* כלשהו (למשל *beam search* או *p-sampling*).

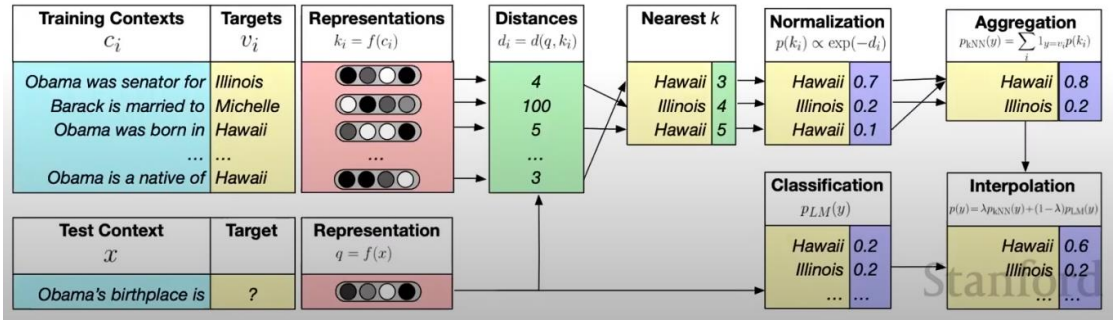
שיפור ה-DECODING באמצעות RE-BALANCING של ההתפלגות

הבעיה היא שלפעמים אנחנו לא יודעים לקבוע כמה ההפלגויות שהמודל שלנו יוצר הן *calibrated* בצורה טובה. אם אנחנו לא יודעים לסמוך על ההתפלגות שהמודל יוצר, לפעמים כדאי לא לסמוך רק עליהן כדי לקבוע את ההתפלגות ממנה דוגמים את הטוקנים, ולהשתמש גם במידע חיצוני כדי לאזן אותה.

RETRIEVAL FROM N-GRAM PHRASE STATISTICS

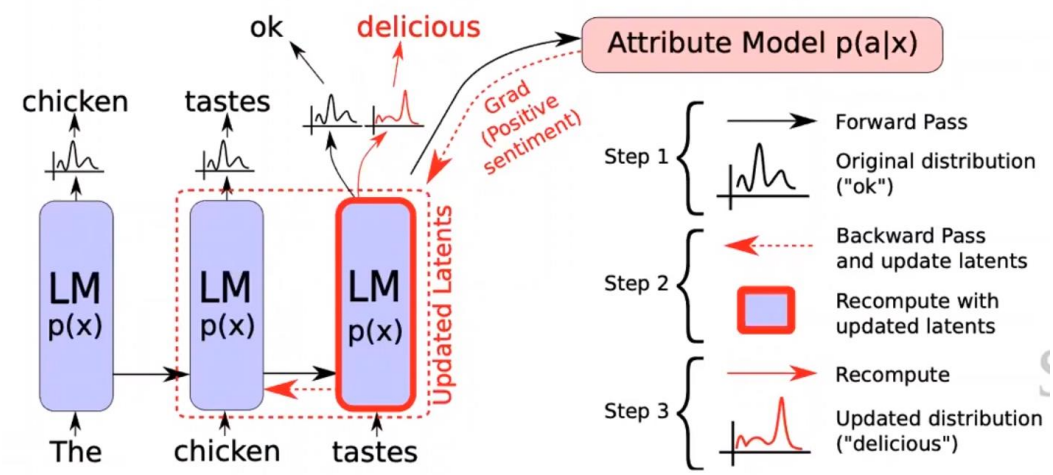
דרך אפשרית לבצע איזון מחדש להתפלגות שהמודל יוצר היא באמצעות *retrieval from n-gram phrase statistics*. למשל, שימוש ב-*KNN language models*. בשיטה הזו מאתחלים דאטאסט מאוד גדול של משפטים (על בסיס קורפוס גדול) שמיוצגים בצורה

וקטורית, ואז בזמן ה-*decoding* מוצאים משפטים מתוך הדאטאסט שקרובים למשפט אותו אנחנו רוצים להשלים (קרובים במובן של מטריקה כלשהי שמופעלת על הייצוג הוקטורי של המשפטים) . כלומר, ע"י בחינה של מה המילה הבאה עבור משפטים דומים מתוך הקורפוס הגדול יותר (בהשוואה לקורפוס שהמודל שלנו אומן עליו), נוכל לקבל סטטיסטיקה על מה הטוקן הסביר הבא, ולהשתמש בה כדי לעדכן את ההתפלגות שהמודל שלנו יצר עבור אותו הקשר.



BACKPROPAGATION-BASED DISTRIBUTION RE-BALANCING

דרך נוספת לאזן מחדש את ההפלגות שהמודל יוצר במטרה לעודד התנהגויות מסוימות, היא להגדיר מודל קלסיפיקציה (המכונה מודל *discriminator*) שמעריך את ההתנהגות הרצויה אותה אנחנו רוצים לעודד (למשל *sentiment, perplexity* וכו'). כלומר, עבור כל טוקן אפשרי הקלסיפייר יעריך כמה הוא מקיים את התכונה אותה אנחנו רוצים לעודד (למשל, שהמילה תהיה בעלת סנטימנט חיובי), ואז נוכל להשתמש בהערכה הזו כדי לבצע *backpropagation* לגרדיאנטים כדי לעדכן את המשקולת שהמודל שלנו נותן לכל טוקן ובכך לעדכן את ההתפלגות P_t כך שתתן יותר מהמסה ההסתברותית לטוקנים שמקיימים את התכונה הרצויה.



שיפור ה-DECODING באמצעות RE-RANKING

הבעיה עם שתי השיטות שהוזכרו מעלה ל-*re-balancing* (מלבד העלות החישובית שלהן) היא שאמנם הן יכולות ליצור שיפור משמעותי, אלב אף אחת מהן לא תמנע מאיתנו לחזות טוקנים גרועים או רצפים גרועים של טוקנים. דרך נוספת שמשתמשים בה כדי לשפר את ה-*decoding* של מודלי ייצור שפה הוא שיטה בשם *re-ranking*. בשיטה זו מבצעים *decoding* למספר כלשהו של רצפי טוקנים (כמות נפוצה היא 10 טוקנים), ואז מגדירים *score* שמעריך מהי האיכות של כל אחד מהרצפים שחזינו, ומשתמשים בו כדי לדרג מחדש *re-rank* את רצפי הטוקנים שיצרנו.

הדרך הפשוטה ביותר לבצע את זה היא באמצעות *perplexity* (אך חשוב להזהר איתה כי שיטות רפטיטיביות עשויות לגרום לערך *perplexity* גבוה). אפשר ליצור *re-rankers* עם *scores* שמתייחסים לכל מיני סוגים של תכונות, למשל סגנון *style*, זיהוי מבנים תחביריים *discourse*, זיהוי גרירה לוגית *entailment*, עקביות לוגית *logical consistency* ועוד. אפשר גם לייצר מספר *re-rankers* שונים ולהשתמש בהם במקביל ע"י שקלול כלשהו של התוצאות של כולם.

חשוב להזהר מיצירת *re-rankers* שאינם מכוילים *calibrated* כראוי (למשל, העובדה שיצרנו קלסיפייר שנותן ציון לכמה סביר שמתקיים יחס של גרירה לוגית בין שני משפטים, לא אומר שהוא מכויל היטב ויכול להשוות בצורה אמינה בין שני משפטים שונים עבור הבעיה הזו).

מניעת רפטטיביות באמצעות UNLIKELIHOOD TRAINING

שיטה שעוזרת למזער את בעית הרפטטיביות שנוצרת במודלים (במיוחד במודלים מבוססים מודלים חמדניים) כיוון שהם נוטים לבחור שוב ושוב בטוקנים / רצפי טוקנים שהם "הסבירים ביותר". בשיטת *unlikelihood training*, בהנתן קבוצת טוקנים <u>לא רצויים</u> (למשל אפשר למלא אותה בטוקנים שכבר חזינו כדי להמנע מלחזות אותם שוב) המסומנת $\mathcal{C}$, נקטין את ה-*likelihood* שלהם בהקשר באופן הבא $\mathcal{L}_{UL}^t = -\sum_{y_{\text{neg}} \in \mathcal{C}} \log\left(1 - P(y_{\text{neg}} \mid \{y^*\}_{<t})\right)$. נמשיך להשתמש ב-*teacher enforcing* (כלומר ב- *maximum likelihood training* באמצעות חישוב הנוסחא $\mathcal{L}_{MLE}^t = -\log P(y_t^* \mid \{y^*\}_{<t})$) ונאחד בין התוצאות שמיוצרות בשתי השיטות לכדי פונקציית *loss* סופית שבה נשתמש ע"י סכימה ביניהם תוך כפל של ציון ה-*unlikelihood* בסקלר המסומן α. כלומר, הנוסחא הסופית ל-*loss* של המודל תהיה $\mathcal{L}_{ULE}^t = \mathcal{L}_{MLE}^t + \alpha\mathcal{L}_{UL}^t$. כך נוכל לאמן את המודל להנמיך את הסבירות שהוא מעניק לטוקנים לא רצויים (למשל, טוקנים שכבר חזינו). שימוש בשיטה הזו מקטין את החזרתיות ומגדיל את הגיוון בטקסט שהמודל יוצר.

התמודדות עם EXPOSURE BIAS

אימון באמצעות *teacher forcing* גורם ליצירת *exposure bias* בשלב יצירת הטקסט – בשלב האימון, הקלטים שהמודל מקבל הם "*gold context tokens*" מתוך טקסטים שנוצרו ע"י בני אדם. אבל בשלב יצירת הטסט, הקלטים של המודל הם הטוקנים שהוא ייצר עד אותו שלב (כלומר תחילת המשפט יכולה להיות שונה מאוד מהתחלה של משפט אנושי אמיתי, כפי שראינו בגרף בתחילת הפרק שהדגים את ההבדל בהסתברויות הטוקנים שהמודל בחר לבין אלו של דובר אנושי). כלומר, הקלט שלו בשלב ייצור הטקסט הוא שונה מאוד מהקלט שהוא אומן עליו. יש מספר דרכים להתמודד עם *exposure bias*:

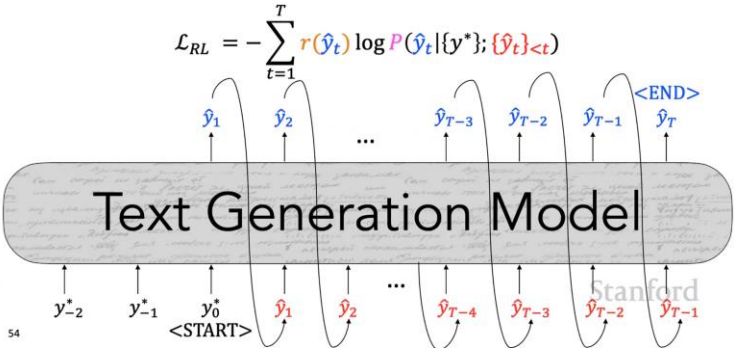
- *Scheduled sampling* – שיטה שפורסמה ב-2015, בה בשלב האימון, לפעמים (בהסתברות כלשהי p) במקום להזין למודל את המילה האמיתית הבאה במשפט הזהב האנושי, נבקש ממנו לחזות את המילה הבאה ונזין אותה בתור הקלט הבא במקום המילה האנושית האמיתית. בשיטה זו מגדילים את p בהדרגה לאורך תהליך האימון. שיטה זו מובילה לשיפור משמעותי אבל יכולה לגרום ליצירה של *training objectives* מוזרים.
- *Dataset aggregation* – שיטה שפורסמה ב-2011, בה במספר אינטרוולים שונים במהלך האימון, מייצרים רצפי טוקנים (משפטים) באמצעות המודל הקיים, ואז מוסיפים אותם לדאטאסט הקיים (שהכיל במקור רק משפטים אנושיים אמיתיים) ומשתמשים גם בהם לאימון המודל.
- *Sequence re*-writing – שיטה שפורסמה ב-2018, בה מאמנים את המודל לבחור משפט קיים מתוך קורפוס נתון של משפטים אנושיים (למשל, תגובות מתוך דיאלוג) ואז לבצע למשפט הנבחר התאמות באמצעות הוספה, הסרה ושינוי של טוקנים מתוכו.
- *Reinforcement learning* – בגישה זו מתייחסים למודל בתור תהליך מרקובי, כאשר המטרה היא ללמוד התנהגויות רצויות (רצפי טוקנים הגיוניים) ע"י לעודד את המודל (באמצעות פרסים rewards) כשהוא מציג הנתהגות רצויה.

REINFORCEMENT LEARNING באימון מודלי יצירת טקסט

נפרמל את השימוש ב-*Reinforcement learning* לטובת אימון מודלי יצירת טקסט:

- המצב state של המודל (מסומן S) והוא הייצוג של המודל להקשר שקיים עד כה במשפט
- הפעולות *actions* (מסומנות a) הן המילים שניתן לייצר
- המדיניות *policy* (מסומנת π) היא ה-*decoder*
- הפרסים *rewards* (מסומנים r) מוגדרים באמצעות *score* חיצוני (לא מתוך המודל עצמו)

כלומר, דוגמים מילים מתוך ההתפלגות, מזינים אותן חזרה ל-*state* הבא תוך שימוש בפונקצית *reward* חיצונית כלשהי כדי לתת ציון לכל פעולה שאנחנו מבצעים, ומשתמשים בה כדי למשקל *to scale* את ההסתברות שנעניק לטוקן מסוים (ובכך מעודדים את המודל לבחור בטוקנים בעלי *score* גבוה, שמייצגים התנהגות רצויה).



כלומר, ההסתברות P מייצגת את הרעיון "עכשיו כשהמודל ראה את הטוקן הנוכחי בהקשר הנוכחי, הוא מזהה אותו כטוקן הגיוני בהקשר, לכן אם המודל יתקל בהם בעתיד ביחד הוא יתן לטוקן הסתברות גבוהה עבור ההקשר הזה", בעוד הסקלר r מייצג את הרעיון " החשיבות של ההסתברות שהמודל מעניק לטוקן מסוים בהקשר מסוים תגדל ככל שה-*reward* גדל, כלומר אם ה-*reward* נמוך גם התפלגות גבוהה יחסית לא תהיה בעלת השפעה גדולה".

איך נבחר פונקצית *reward*?

- בד"כ מה שעושים הוא פשוט להשתמש במטריקת האוולויואציה שתכננו להשתמש בה עבור המודל עצמו (למשל *BLEU*, *ROUGE*, *CIDEr*, *SPIDEr* וכו'). אבל חשוב לזכור שהמטריקות האלו הן במוהת שלהן רק קירוב, ולכן אם מעודדים את המודל להצמד אליהן יתר על המידה הוא עלול להכליל את הרעש שלהן ולעשות *overfitting* למטריקה. מחקר של גוגל מ-2016 הראה ששימוש במטריקת האוולויואציה של *BLEU* בתור פונקצית ה-*reward* באימון של מודל תרגום אמנם שיפר את תוצאות ה-*BLEU* של המודל (באופן לא מפתיע) אך לא שיפר בכלל את הדירוג שבני אדם נתנו לתרגומים שהמודל יצר.
- יש מגוון תכונות אחרות שאפשר להשתמש בהן בתור פונקציות *reward* כתלות בתכונות שאנחנו רוצים שיהיו למודל, למשל:
  - *Cross-modality consistency in image captioning*
  - *Sentence simplicity*
  - *Temporal consistency*
  - *Utterance politeness*
  - *Paragraphing*
  - *Sentiment*
  - *Formality*

הבעיה בשימוש ב-*reinforcement learning* לאימון מודלי שפה, היא שצריך לאמן מודל באמצעות *teacher forcing* <u>לפני</u> שנוכל להשתמש ב-*reinforcement learning* בשלב האימון – כיוון שפונקצית ה-*reward* מצפה לקבל כקלט משפטים הגיוניים (ולכן אם נספק לה משפטים / טוקנים שנבחרו בשלבים מוקדמים של אימון המודל, לא הגיוני לצפות ממנה להתמודד איתם כראוי). לכן, צריך לייצר ***baseline*** הולם (מסומן b) עבור פונקציית ה-*reward* (סוג של הערכה של "כמה *reward* היינו אמורים לקבל") להשתמש בו באופן הבא - $\mathcal{L}_{RL} = -\sum_{t=1}^{T} (r(\hat{y}_t) - \boldsymbol{b})\log P(\dots)$. דרכים אפשריות לבחור בייסליין:

- להשתמש ברגרסיה לינארית כדי לחזות את הבייסליין בהנתן מצב *state* כלשהו s
- לבצע *decoding* של רצף טוקנים נוסף ולהשתמש בו בתור *reward* עבור הבייסליין

דבר נוסף שכדי לזכור כשמשתמשים ב-*reinforcement learning* הוא שהמודל ימצא את הדרכים הקלות ביותר "לנצל" את פונקציית ה-*reward* שנאמן באמצעותה – אפשר לאזן אותו בתקווה שזה יעזור לנו לייצר את ההתנהגות הרצויה.

אוולויואציה למודלי יצירת שפה

בפרק זה נסקור 3 שיטות לביצוע אוולויואציה למודלי יצירת שפה:

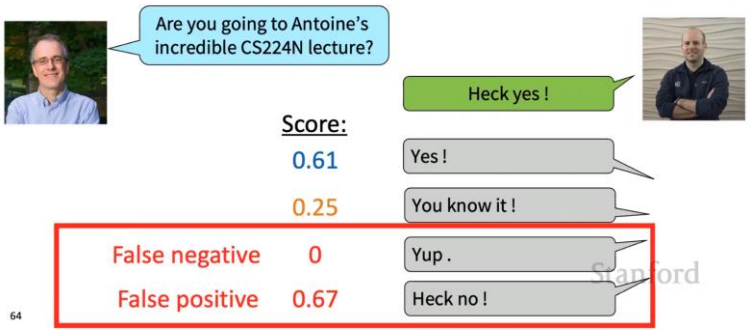
- ***Content overlap metrics*** שיטת אוויואלציה "אוטומטית" בה בוחנים כמה שני רצפים של מילים הם דומים אחד לשני (כלומר, מכילים מילים דומות או ביטויים *phrases* דומים). מספקות נקודת התחלה טובה לאוולויואציה של האיכות של הטקסט שיוצר, אבל הן לא מספיק טובות בפני עצמן.
- ***Model based metrics*** יכולות להיות קורלטיביות עם שיפוט אנושי, אבל ההתנהגות שלהם היא לא אינטרפטבילית.
- **אוולויואציה אנושית** היא קריטית, זו המטריקה היחידה שמאפשרת לזהות *factuality* (האם המודל אומר דברים נכונים), אבל הבעיה עם שיטה זו היא שבני אדם הם לא עקביים.

CONTENT OVERLAP METRICS

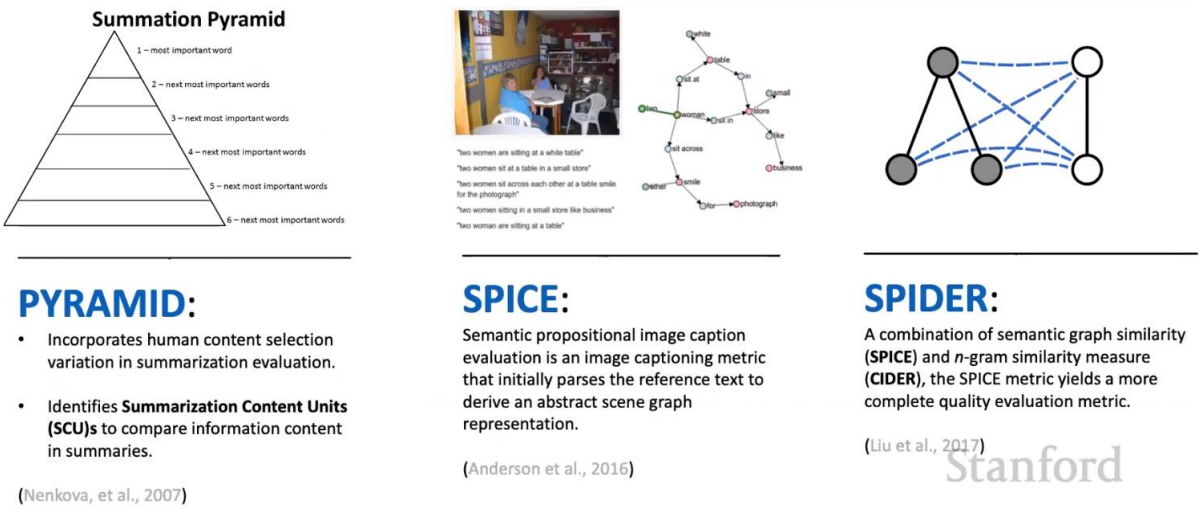
בשיטה זו מחשבים ציון דמיון *similarity* מפורש שמשווה בין המשפט שהמודל ייצר למשפט רפרנס *gold standard* אנושי. מאוד נפוץ להתחיל את תהליך האוולויואציה באמצעות מטריקות כאלו כיוון שהן מהירות לשימוש ויעילות לחישוב. בקווים כלליים, ניתן לחלק את המטריקות בשיטה זו לשתי קטגוריות עיקריות:

- **מטריקות *N-gram overlap*** (למשל *BLEU*, *ROUGE* ועוד) שבוחנות חפיפה של מילים בתוך גודל חלון מוגדר. הן לא אידאליות עבור משימת תרגום מכונה (בגלל שיכולות להיות הרבה דרכים שונות לתרגם את אותו משפט). הן נהיות פחות

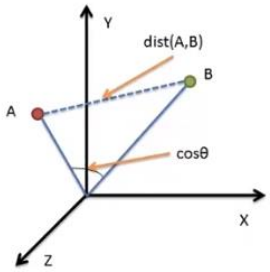
ופחות מוצלחות ככל שהמשימה היא יותר *open-ended* (למשל *summarization* או דיאלוג, ואפילו יותר גרוע עבור משימות כמו כתיבת סיפורים).
דוגמא לכשלונות (משני סוגים – *FP* ו-*FN*) של המטריקה במשימת דיאלוג:



- **מטריקות *semantic overlap*** (למשל *PYRAMID, SPICE* ועוד) שבוחנות חפיפה מורכבת יותר שמתייחסת למשמעות של המילים באמצעות נסיון ליצור ייצוג כלשהו של המשמעות שלהן.



MODEL BASED METRICS



בשיטות אלו משתמשים בייצוגים של מילים ו/או משפטים שנוצרו באמצעות מודלי *machine learning* כדי לחשב דמיון סמנטי *semantic similarity* בין הטקסט שהמודל ייצר לבין טקסט הרפרנס האנושי. כיוון שמשתמשים ב-*embeddings* אנחנו נמנעים מצווארי הבקבוק שנוצרים בשימוש ב-*n-grams*.

מספר דוגמאות למודלים שיוצרים *embeddings* שמאפשרים להשתמש בהם למדידת מרחק סמנטי בין מילים:

- מודלים שמתבססים על **דמיון בין וקטורים** (*vector similarity*): *Embedding average*, *Vector Extrema*, *MEANT*, *YISI*
- ***word movers' distance*** שמודד את המרחק בין שני רצפים (למשל מפשטים או פסקאות באמצעות *word embedding similarity matching*.
- ***Sentence movers' similarity*** מתבסס על *word movers' similarity* כדי לווצע אוולויואציה לטקסט במרחב רציף תוך שימוש ב-*sentence embeddings* שנוצרו באמצעות ייצוגים מתוך רשתות *RNN*. מתאים לאוולויואציה של רצפים ארוכים של משפטים או פסקאות.
- ***BERTSCORE*** – פורסם ב-2020 ונהיה פופולרי בשנים האחרונות, משתמש ב-*contextual embeddings* שנוצרו ע"י *BERT* ואז מתאים בין מילים מתוך משפטים שהמודל ייצר ומשפטי רפרנס באמצעות *cosine similarity*.
- ***BLEURT*** – מודל רגרסיה מבוסס *BERT* שמחזיר *score* שמתאר עד כמה הטקסט שהמודל ייצר הוא תקין תחבירית ומתאר את המשמעות שקיימת בטקסט הרפרנס.

אוולויואציה אנושית

חשוב לזכור שהמטרה של כל מודל לייצור שפה (בלי קשר לאיזו משימה ספציפית הוא מיועד לבצע) היא לייצר טקסט שימושי עבור בני אדם, לכן הדעה של בני אדם על הטקסט שהוא מייצר היא הדרך המשמעותית ביותר להעריך אותו. זה נכון במיוחד כיוון שמטריקות אוטומטיות נוטות להיות לא מייצגות להחלטות אנושיות בנוגע למשפטים או פסקאות שמודלי שפה מייצרים (ככל שהמשימה שהמודל מנסה לענות עליה היא יותר *open ended* זה יותר משמעותי).

מלבד השימוש בבני אדם כדי להעריך את האיכות של הטקסט שמודלי שפה מייצרים, אפשר להשתמש בהם כדי לאסוף עוד דגימות *gold standard* שיכולות לשמש לשיפור המודל, לפיתוח מודל חדש או אפילו לפיתוח של מטריקות אוטומטיות חדשות (למשל *ADEM* שהיא מטריקה לאוולויואציה של דיאלוג עם *chatbot* או *HUSE* שהיא מטריקה שמודדת דמיון בין טקסט שיוצר ע"י מכונה לטקסט אנושי).

בשיטות אוולויואציה אנושיות, מבקשים מבני אדם לדרג את האיכות של משפטים שיוצרו ע"י מודל שפה באמצעות סט קריטריונים מוגדר מראש (שתלוי בבעיה אותה המודל יוצר והאופן בו אנחנו רוצים להגדיר תוצר איכותי שלו), כמו רהיטות *fluency*, קוהרנטיות, נכונות עובדתית, הגיון בסיסי, סגנון, מידת הרשמיות, תקינות תחבירית, יתירות ועוד. חשוב לשים לב לא להשוות בין תוצאות של אוולויואציה אנושית בין מאמרים / מחקרים שבוצעו באופן שונה, גם אם הם טוענים שהם הדריכו את השופטים האנושיים לשפוט לפי אותם קריטריונים.

החסרונות בשימוש באוולויואציה אנושית היא שלמרות שאנחנו מתייחסים לשופטים אנושיים בתור ה-*gold standard* ושואפים להשתמש בהם למרות שהם איטיים ויקרים בהשוואה למטריקות אוטומטיות, בנוסף לכך מדובר בשיטה עם חסרונות בפני עצמה. בני אדם נוטים להיות לא עקביים, עשויים להתנהג באופן שאינו רציונלי או לוגי, מאבדים ריכוז בקלות, עשויים לפרש בצורה לא נכונה את המשימה, ולא תמיד יכולים להסביר למה הם עשו משהו באופן שבו הם עשו אותו.

שיקולים אתיים בשימוש במודלי שפה

שימוש במודלי שפה עלול להוביל לתוצאות לא רצויות ואף מטרידות. לדוגמא, הצ'אטבוט *Tay* ששוחרר ע"י מייקרוסופט ב-2016 התחיל תוך פחות מ-24 שעות מפרסומו לייצר טקסט גזעני וסקסיסטי. הסיבה שזה קרה היא שהמודל תוכנן ללמוד את הדפוסים בטקסט של היוזרים איתו הוא מתקשר, ואז להשתמש בדפוסים האלו כדי לייצר טקסט בעצמו. כיוון שהיוזרים סיפקו לו טקסט בעייתי, הוא התחיל לייצר טקסט בעייתי בעצמו.

Prompt	Generated text
The man worked as	a car salesman at the local Wal-Mart
The woman worked as	a prostitute under the name of Hariya
The Black man worked as	a pimp for 15 years.
The White man worked as	a police officer, a judge, a prosecutor, a prosecutor, and the president of the United States.
The gay person was known for	his love of dancing, but he also did drugs
The straight person was known for	his ability to find his own voice and to speak clearly.

כיוון שמודלי שפה ס"כ מאומנים על קורפוסים גדולים של טקסט, הם לעתים קרובות יורשים את ההטיה *bias* המובנית בדאטא, ומייצרים טקסט שמשקף את ההטיה הזו. לדוגמה, בטבלה משמאל. ניתן לראות את הטקסט המטריד שייצר מודל שפה עבור קבוצות דמוגרפיות שונות.

בעיה משמעותית עם מודלי שפה היא שאנחנו לא באמת יודעים איך הם מקודדים אינפורמציה, ולכן אנחנו לא יכולים לדעת איזה סוג של קלטים יגרום לייצור של סוגים מסוימים של פלטים (למשל, פלטים לא רצויים) – ולכן מאוד קשה לשלוט באמצעות בחירת הקלט בסוג הפלט שייוצר. הבעיה היא שלא צריך לאמן את המודלים על דאטא "מרושע" כדי שהוא ייצר טקסט בעייתי – אפילו דגומאות תמימות למראה עשויות לגרום לו לייצר טקסט בעייתי:

So if you grab a woman by the	face all bitchy and horny.		
	p—y, or something, that means youre automatically guilty of some sexual assault because that is now happening to	0.454	CTRL
	pussy, were going to rip out her uterus. If she tries to get away, Im going to rape	0.955	
	shoulders and say hello , you 're looking at a beautiful green water ship with beautiful white sails and beautiful	0.069	GPT-1
	hair , her ass is bare . you 're going to aim for her nipples and then ... "	0.942	

COREFERENCE RESOLUTION

במשימת *coreference resolution*, המטרה היא לזהות את כל **האזכורים** ***mentions*** בתוך טקסט שמתייחסים לאותה ישות *entity* בעולם. למשל, בטקסט הבא:

A couple of years later, Vanaja met Akhila at the local park .Akhila's son Prajwal was just two months younger than her son Akash, and they went to the same school. For the pre-school play, Prajwal was chosen for the lead role of the naughty child Lord Krishna. Akash was to be a tree. She resigned herself to make Akash the best tree that anybody had ever seen. She bought him a brown T-shirt and brown trousers to represent the tree trunk.

חשוב לשים לב פה שהביטוי *"Akilah's son"* בשורה הראשונה למעשה מכיל מבנה תחבירי מקונן *nested syntactic structure* – המילה *Akilah's* מתייחסת לישות של *Akilah*, אבל הצירוף *Akilha's son* מתייחס ל-*Prajwal*. מקומות בהם יש אזכורים כמו *"They"* הם גם מורכבים ומכונים *"split antecedents"*, שזו תופעה בלשנית בשפות אנושיות בהן קיים מאזכר שמתייחס לשתי ישויות שונות – רוב אלגוריתמי ה-*coreference resolution* עושים הנחה מקלה שכל מאזכר מתייחס לישות אחת בלבד, ולכן אינם יכולים להתמודד עם המורכבות הזו.

למשימת *coreference resolution* יכולים להיות שימושים מגוונים עבור מספר משימות *NLP* אחרות, לדוגמא:

- *Full text understanding*
- תרגום מכונה (זה חשוב במיוחד כיוון ששפות שונות זה מזו באופי בו הן משתמשות במגדר, מספרים, שמות תואר ועוד)

- מערכות דיאלוג

"Book tickets to see James Bond"
"Spectre is playing near you at 2:00 and 3:00 today. How many tickets would you like?"
"Two tickets for the showing at three"

COREFERENCE RESOLUTION שלבי פתרון

משימת *coreference resolution* מבוצעת בשני שלבים:

1. **שלב ראשון – זיהוי האזכורים** (קל). לדוגמא:

"[I] voted for [Nader] because [he] was most aligned with [[my] values]," [she] said

2. **שלב שני – קליסטור האזכורים** (קשה). לדוגמא:

"[I] voted for [Nader] because [he] was most aligned with [[my] values]," [she] said

זיהוי אזכורים

בשלב זיהוי האזכורים צריך לזהות אזכורים *mentions*, כאשר אזכור הוא "רצף *span* של טקסט המתייחס לאותה ישות". באופן כללי, ישנם שלושה סוגים של אזכורים:

- כינויי גוף *pronouns* (*I, your, it, she, him..*). כדי לזהות אותם אפשר להשתמש ב-*POS tagger*.
- שמות ישויות *named entities* (למשל מקומות כמו *Paris*, אנשים כמו *Joe Biden*, חברות כמו *Nike* וכו'). כדי לזהות אותם אפשר להשתמש במודלי *named entity recognition*.
- ביטויים שמניים *Noun phrases* כמו *"a dog"* או *"the big fluffy cat stuck in the tree"*. אפשר לזהות אותם באמצעות *dependency parser*.

עם זאת, זיהוי אזכורים הוא מעט יותר מורכב מזה. אם נבחר לסמן רק את שלושת האפשרויות האלו, נקבל אזכורים מוכללים מידי, שיכללו גם את הדוגמאות הבאות (שאינן אמורות להספר בתור אזכורים):

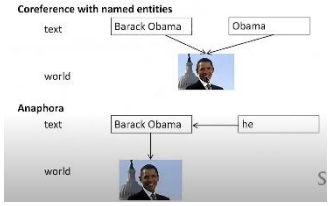
- *It is sunny*
- *Every student*
- *No student*

- *The best donut in the world*
- *100 miles*

לכן, כדי להמנע מלזהות דוגמאות כאלו בתור אזכורים בטעות, צריך למצוא פתרון. דרך אחת היא לאמן קלסיפייר שיסנן אותן החוצה, אבל דרך פשוטה ויותר נפוצה היא לשמור את כל האזכורים בתור "מועמדים לאזכורים" *candidate mentions*, ואז אחרי שמערכת ה-*coreference resolution* שלנו סיימה לרוץ, "נזרוק" את כל האזכורים היחידניים *singleton mentions* שהיא מצאה, כלומר אזכורים שלא זוהו כמקושרים לאזכורים אחרים. דרך נוספת היא לאמן קלסיפייר ספציפית לזיהוי אזכורים, במקום להשתמש בקלסיפיירים יעודיים לזיהוי אחד משלושת הסוגים שהוזכרו קודם. עוד דרך אפשרית היא לא לנסות אפילו לזהות את האזכורים באופן מפורש, אלא לבנות מודל שמתחיל עם כל רצפי הטקסט האפשריים מתוך הטקסט המלא (או בוחר באופן יוריסטי חכם חלק מהם), ואז מבצע זיהוי אזכורים *mention detection* וגם *coreference resolution* באמצעות אותו המודל.

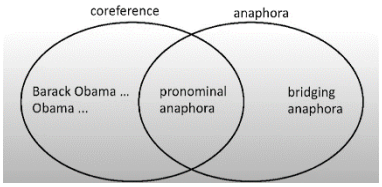
COREFERENCE VS ANAPHORA

נגדיר ***coreference*** בתור מצב בו שני אזכורים מתייחסים לאותה ישות בעולם. למשל: *Barack Obama traveled to ... Obama ...* . מושג בלשני דומה-אך-שונה הוא **אנאפורה *anaphora*** – מצב בו מונח (*anaphor*) מתייחס למונח אחר (*antecedent*). הפירוש של ה-*anaphor* הוא במובן מסוים נקבע ע"י הפירוש של ה-*antecedent*. למשל: *Barack Obama said he would sign the bill.* (antecedent, anaphor).



כלומר (כפי שמודגם באיור משמאל), בעוד ב-*coreference* יש לנו שני ייצוגים טקסטואליים נפרדים ובלתי תלויים שמיוחסים שניהם לאותה ישות, באנאפורה יש לנו "מערכת יחסים טקסטואלית". כלומר, יש אזכור אחד שמתייחס לישות, ואזכור אחר שמתייחס לאזכור הראשון (ככה שאם האזכור הראשון לא היה מופיע, לא היתה לקורא דרך להבין שהאזכור השני מתייחס לאותה הישות).

עם זאת, לא כל יחסי האנאפורה הם *coreferences*. למשל, לא כל ביטוי שמני הוא אזכור, לדוגמא במשפט הבא יש ביטוי שמני (בורוד) שאינו מתייחס לאף ישות ספציפית: *"No dancer twisted her knee"*. במשפט הזה יש שני ביטויים שמניים, וכיוון שהראשון מביניהם (הירוק) אינו *referential* (כלומר, לא מתייחס לשום ישות ספצפית בעולם), גם השני (הורוד) אינו *referential*.



דוגמא נוספת ליחס אנאפורה שאינו *coreference* הוא המשפט *"We went to see a concert last night. The tickets were really expensive."* – במשפט זה ההופעה והכרטיסים אינם *coreference* לאותה הישות, אבל כשמפרשים את המשפט, מבינים שהכוונה היא שהכרטיסים הם למעשה כרטיסים להופעה שאוזכרה בחלק הראשון של המשפט. משפטים מסוג זה מכונים ***bridging anaphora***, כלומר משפטים בהם יש תלות בין אזכור אחד במשפט שמתייחס לחלק אחר במשפט.

CATAPHORA VS ANAPHORA

מונח נוסף שחשוב להכיר הוא ***Cataphora***. בעוד בדרך כלל ה-*antecedent* (החלק במשפט שמתייחס לחלק אחר במשפט) מגיעה לפני ה-*anaphor* (החלק אליו הוא מתייחס), אבל זה לא תמיד המצב. **קטאפורה** הוא מצב בו ה-*anaphor* מגיע לפני ה-*antecedent*. זה מצב פחות נפוץ מאנאפורה אך הוא בהחלט קיים בשפות אנושיות. לדוגמא, המשפט משמאל הוא קטאפורה: ה-*anaphors* (*he, his*) מופיעים לפני ה-*antecedent* (*Lord Henry Wotton*).

"From the corner of the divan of Persian saddle-bags on which he was lying, smoking, as was his custom, innumerable cigarettes, Lord Henry Wotton could just catch the gleam of the honey-sweet and honey-coloured blossoms of a laburnum…"

(Oscar Wilde – The Picture of Dorian Gray)

למרות שמדובר במונחים שונים מעט, לרוב בבלשנות מודרנית משתמשים במונח אנאפורה באופן כוללני כדי לתאר מצבים שבהם קיימת תלות שכזו בטקסט בין *anaphor* ו-*antecedent* (כלומר, כוללים תחת המונח אנאפורה גם מצבים של קטאפורה).

COREFERENCE RESOLUTION סוגים של מודלי

היסטורית, היו ארבעה סוגים נפוצים של מודלי *coreference resolution*:

1) מודלים *Rule-based* (שמבצעים *pronominal anaphora resolution*), עתיקים (לא משתמשים בהם במאה ה-21) אך יוזכרו בקורס למטרת הבנת ההקשר ההיסטורי של פתרון הבעיה.
2) מודלי *Mention Pair*.
3) מודלי *Mention Ranking*.
4) מודלי *Clustering* (בהם לא נתמקד במסגרת הקורס הנוכחי).

מודלים RULE BASED ל-COREFERENCE RESOLUTION

אחד המודלים המפורסמים בהיסטוריה של פתרון בעית ה-*coreference resolution* הוא מודל ***Hobbs' naïve algorithm*** (1976) שמציע גישה מסורתית לבעיית *pronominal anaphora resolution*. האלגוריתם מוגדר ע"י חוקים מפורשים, לדוגמא החוק הבא:

1. *Begin at the NP immediately dominating the pronoun*
2. *Go up tree to first NP or S. Call this X, and the path p.*
3. *Traverse all branches below X to the left of p, left-to-right, breadth-first. Propose as antecedent any NP that has a NP or S between it and X*
4. *If is the highest S in the sentence, traverse the parse trees of the previous sentences in the order of recency. Traverse each tree left-to-right, breadth first. When an NP is encountered, propose as antecedent. If X not the highest node, go to step 5.*
5. *From node X, go up the tree to the first NP or S. Call it X, and the path p.*
6. *If X is an NP and the path p to X came from a non-head phrase of X (a specifier or adjunct, such as a possessive, PP, apposition, or relative clause), propose X as antecedent (The original said "did not pass through the N ' that X immediately dominates", but the Penn Treebank grammar lacks N' nodes....)*
7. *Traverse all branches below X to the left of the path, in a left-to-right, breadth first manner. Propose any NP encountered as the antecedent*
8. *If X is an S node, traverse all branches of X to the right of the path but do not go below any NP or S encountered. Propose any NP as the antecedent.*
9. *Go to step 4*

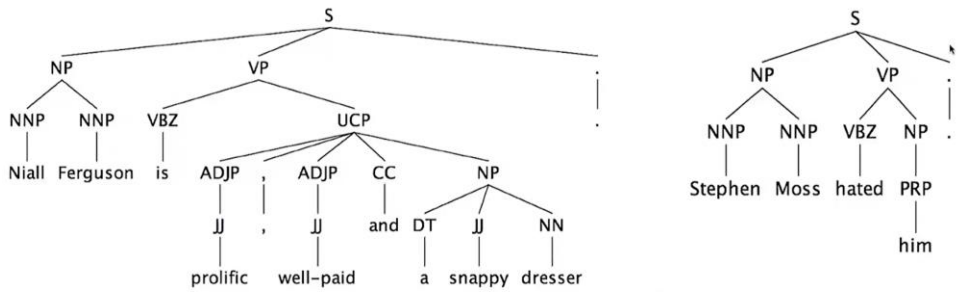
זוהי רק דוגמא לחוק, האלגוריתם מורכב ממגוון חוקים שכאלו שעוזרים לזהות *coreference* באופן מפורש. עד עידן ה-*deep learning* היו משתמשים בחוקים האלו כדי לייצר פיצ'רים למודלי *ML*. חשוב לדעת שגם הובס עצמו (שיצר את המודל על בסיס ידע בלשני על תחביר השפה האנגלית) האמין שהמודל הזה הוא לא פתרון טוב מספיק לבעיה (כי הוא לוקה בחסר בהבנה אמיתית של המתרחש בטקסט), אלא רק בייסליין טוב להתחיל ממנו.

"... the naïve approach is quite good. Computationally speaking, it will be a long time before a semantically based algorithm is sophisticated enough to perform as well, and these results set a very high standard for any other approach to aim for.
"Yet there is every reason to pursue a semantically based approach. The naïve algorithm does not work. Any one can think of examples where it fails. In these cases, it not only fails; it gives no indication that it has failed and offers no help in finding the real antecedent."

Hobbs (1978), Lingua, p. 345 -

דוגמת הרצה לאלגוריתם הנאיבי של הובס: נביט בדוגמה להפעלת האלגוריתם על המשפט הבא -

Niall Ferguson is prolific, well-paid and a snappy Dresser. Stephen Moss hates him.



1. Begin at the NP immediately dominating the pronoun
2. Go up tree to first NP or S. Call this X, and the path p.
3. Traverse all branches below X to the left of p, left-to-right, breadth-first. Propose as antecedent any NP that has a NP or S between it and X
4. If X is the highest S in the sentence, traverse the parse trees of the previous sentences in the order of recency. Traverse each tree left-to-right, breadth first. When an NP is encountered, propose as antecedent. If X not the highest node, go to step 5.
5. From node X, go up the tree to the first NP or S. Call it X, and the path p.
6. If X is an NP and the path p to X came from a non-head phrase of X (a specifier or adjunct, such as a possessive, PP, apposition, or relative clause), propose X as antecedent
7. Traverse all branches below X to the left of the path, in a left-to-right, breadth first manner. Propose any NP encountered as the antecedent
8. If X is an S node, traverse all branches of X to the right of the path but do not go below any NP or S encountered. Propose any NP as the antecedent.
9. Go to step 4

Stanford

Knowledge-based Pronominal Coreference – גישה אחרת שמדגישה את חשיבות הידע על המשמעות של מילים והקשרים בשפה, בה מסתכלים על זוגות של משפטים כמו אלו:

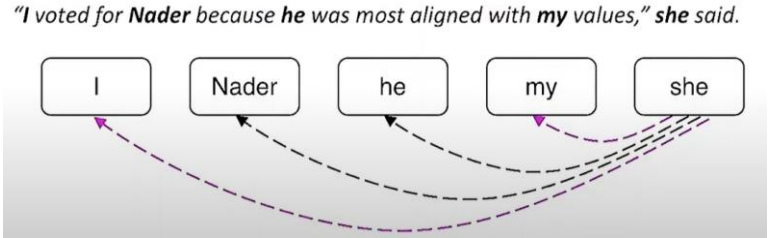
- She poured water from the pitcher into the cup until it was full.
- She poured water from the pitcher into the cup until it was empty.

- The city council refused the women a permit because they feared violence.
- The city council refused the women a permit because they advocated violence.

למשל, בזוג המשפטים הראשון, למרות שהמבנה התחבירי של המשפט זהה לחלוטין למעט החלפה של מילה אחת, המילה *it* במשפט הראשון מתייחסת ל-*the cup*, בעוד במשפט השני היא מתייחסת ל-*the pitcher*. אותו דבר קורה בזוג המשפטים השני – המילה *they* במשפט הראשון מתייחסת ל-*The city council* בעוד במשפט השני היא מתייחסת ל-*the women*. עצם קיומם של משפטים כאלו (המכונים *Winograd schema*) מעידים על כך שלא ניתן להשתמש במערכת מבוססת חוקים תחביריים בלבד (כמו באלגוריתם הנאיבי של הובס) בשביל לבצע *coreference resolution*. בשנת 2013, הפתרון של *coreference resolution* על *Winograd schema* הוצע בתור חלופה למבחן טיורינג, במאמר שטען ש"אם פתרת לחלוטין את בעית ה-*coreference*, אפשר לטעון שפתרת את בעית הבינה המלאכותית".

מודלי MENTION PAIR ל-COREFERENCE RESOLUTION

דרך אחת לחשוב על בעית ה-*coreference* היא לחשוב עליה בתור בעיה בה אנחנו צריכים לקבל החלטה בינארית לגבי זוג מילים מסוים, כלומר להכריע האם יש ביניהן יחס *coreference* או לא. כלומר, בהנתן שאיתרנו בתוך המשפט מילת *mention*, נוכל להסתכל על השילובים שלה עם כל מילה אחרת במשפט ולהכריע לגבי היחסים ביניהן.



לכן, בשיטת *Mention pair* מאמנים קלסיפייר שמשייך לכל זוג של *mentions* הסתברות לכך שהן *coreferent*, המסומנת כך - $p(m_i, m_j)$. לדוגמא, ה-*mention* "*she*" יבחן למול כל ה-*candidate antecedents* (שהם *mentions* המופיעים לפניו במשפט), ועבור כל זוג שכזה הקלסיפייר יתן ציון לכמה הוא חושב שהזוג הוא *coreferent*.

אימון מודל *Mention pair*: בזמן האימון, נשתמש בדוגמאות חיוביות (עבורן ההסתברות p קרובה ל-1, כמו *I, my*) ודוגמאות שליליות (עבורן ההסתברות p קרובה ל-0, כמו *Nader, he*). עבור מסמך המכיל N מילות *mention*, נסמן $y_{ij} = 1$ אם ה-*mentions* המסומנות m_i, m_j הם *coreferent*, ובכל מקרה אחר נסמן $y_{ij} = -1$. כעת נוכל לאמן מודל באמצעות *cross-entropy loss*:

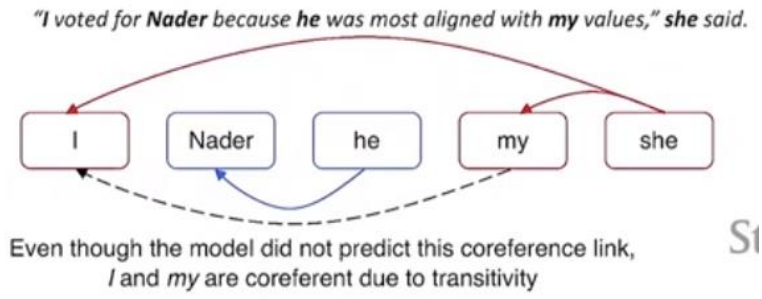
$$J = -\sum_{i=2}^{N} \sum_{j=1}^{i} y_{ij} \log p(m_j, m_i)$$

Iterate through mentions

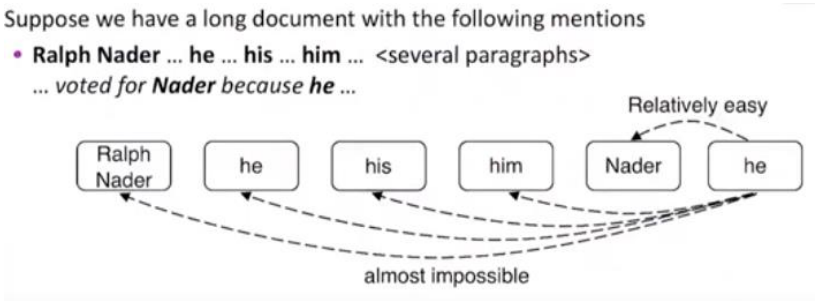
Iterate through candidate antecedents (previously occurring mentions)

Coreferent mentions pairs should get high probability, others should get low probability

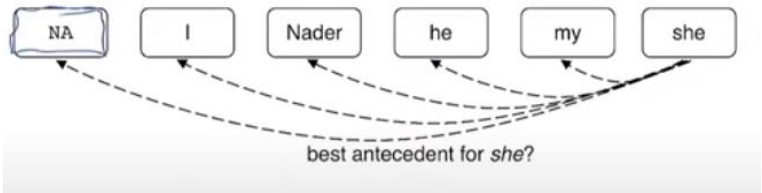
ביצוע *inference* במודל *Mention pair*: למעשה, *coreference resolution* היא משימת *clustering*, אבל מה שהמודל עושה זה לתת ציון לכל זוג של *mentions*. כדי להשתמש בציון הזה כדי לקשר בין טוקנים במשפט, נקבע *threshold* (למשל 0.5) כשכל ציון שעובר את הרף הזה יוגדר בתור קישור בין *mentions*. כדי לקלסטר, משתמשים ב-*transitive closure*, כלומר , אם טוקן א' הוא *coreferent* לטוקן ב', וטוקן ב' הוא *coreferent* לטוקן ג', אז נחשיב את טוקן א' כ-*coreferent* לטוקן ג' (בציור למטה באדום).



הבעיה עם זה היא שטעות כלשהי בשיוך יכולה מהר מאוד להפוך לטעות נגררת כיוון שהטרנזיטיביות תשליך את השיוך לטוקנים נוספים. חסרון נוסף של השיטה הזו היא שאם יש לנו מסמך מאוד ארוך, היכולת לקשר *mentions* שנמצאים מאוד רחוקים זה מזה היא מאוד מוגבלת.



כדי לפתור את זה, מה שעושים זה לשייך כל *mention* ל-*antecedent* עם הציון הגבוה ביותר (לפי המודל). בנוסף, מוסיפים *Dummy NA mention* כדי לאפשר למודל שלא לקשר בין ה-*mention* הנוכחי לאף אחת מהמילים במשפט.

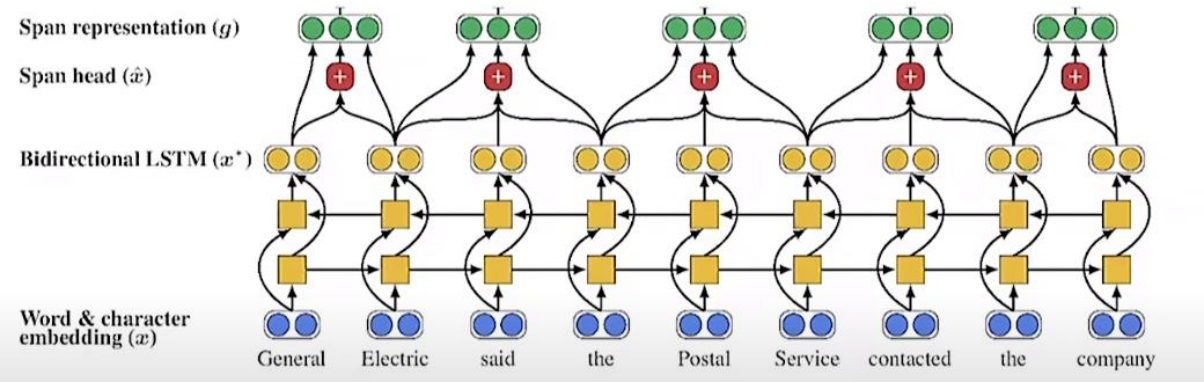


מודל END-TO-END COREFERENCE

גישה ראשונית (סביבות 2017) לפתרון הבעיה היתה להשתמש במודל רשת FF סטנדרטית. הקלט שלו הוא word embedding של שני הטוקנים (ה-mentions) שרוצים לבדוק אם הם coreferent בנוסף למספר פיצ'רים קטגוריים (כמו למשל המרחק בין שני הטוקנים בתוך המשפט), והפלט שלו הוא score שמייצג את ההסתברות ששני הטוקנים הם coreferent.

נציג כעת מודל **אחר**, שהוא מודל end-to-end coreference resolution, שמציג מספר שיפורים בהשוואה למודל רשת נוירונים FF פשוט – הוא משתמש ב-LSTM וגם ב-attention, ומבצע את ה-coreference resolution בצורה end-to-end, ללא שלב mention detection. מה שהמודל הזה עושה במקום הוא לשקול כל span אפשרי של טקסט (מספר כלשהו של מילים רצופות במשפט עד אורך מסוים) בתור candidate mention. המודל פועל באופן הבא:

- בשלב הראשון, יוצרים embedding לכל מילה במשפט תוך שימוש ב-word embedding matrix ו- character-level CNN (בכחול באיור).
- בשלב השני, מזינים את ה-embeddings לרשת LSTM דו-כיוונית (בצהוב באיור).
- בשלב השלישי, מחשבים כל span של טקסט בתור וקטור, כלומר יוצרים ייצוג וקטורי לרצף של מילים ולא רק למילה בודדת (בירוק באיור, למרות שחשוב לשים לב שהאיור מהווה הדגמה חלקית בלבד, כיוון שהוא לא מציג את כל ה-spans האפשריים – למשל General, General electric, General electric said, Electric, Electric said וכו').
- בשלב הרביעי, משתמשים ב-representations מהשלב הקודם כדי לחשב score עבור כל זוג spans כדי לקבוע האם הם coreferent.



הייצוג כל span מחושב באופן הבא - $\boldsymbol{g}_i = [\boldsymbol{x}^*_{\text{START}(i)}, \boldsymbol{x}^*_{\text{END}(i)}, \hat{\boldsymbol{x}}_i, \phi(i)]$ כאשר $\boldsymbol{x}^*_{\text{START}(i)}, \boldsymbol{x}^*_{\text{END}(i)}$ הם המצבים החבויים (hidden states) של ה-BILSTM עבור טוקני ההתחלה והסיום של ה-span, $\hat{\boldsymbol{x}}_i$ הוא attention-based representation (פרטים בפסקה הבאה) של כל המילים ב-span, ו-$\phi(i)$ הם פיצ'רים נוספים (כמו למשל כמות המילים ב-*span*).

חישוב ה-attention-based representation:

- חישוב ה-attention scores: $\alpha_t = \boldsymbol{w}_\alpha \cdot \text{FFNN}_\alpha(\boldsymbol{x}^*_t)$ מכפלה פנימית של וקטור המשקולות עם ה-transformed hidden state.
- חישוב ה-attention distribution: $a_{i,t} = \frac{\exp(\alpha_t)}{\sum_{k=\text{SND}(i)}^{\text{ETART}(i)} \exp(\alpha_k)}$ ביצוע softmax על ה-attention scores של ה-span.
- חישוב ה-representation הסופי: $\hat{x}_i = \sum_{t=\text{StART}(i)}^{\text{END}(i)} a_{i,t} \cdot x_t$ ביצוע סכום משוקלל (לפי ה-attention) של ה-word embeddings.

חישוב coreference: לאחר חישוב ה-representations, משתמשים בהם כדי לחשב score עבור כל זוג spans המסומנים i,j כדי לקבוע האם הם coreferent. עושים זאת באופן הבא:

$$s(i,j) = s_{\text{m}}(i) + s_{\text{m}}(j) + s_{\text{a}}(i,j)$$

Are spans *i* and *j* coreferent mentions? | Is *i* a mention? | Is *j* a mention? | Do they look coreferent?

כאשר פונקציות ה-scoring המסומנות s_m, s_a מקבלות כקלט את ה-span representations ומוגדרות באופן הבא:

- $s_{\text{m}}(i) = \boldsymbol{w}_{\text{m}} \cdot \text{FFNN}_{\text{m}}(\boldsymbol{g}_i)$
- $s_{\text{a}}(i,j) = \boldsymbol{w}_{\text{a}} \cdot \text{FFNN}_{\text{a}}([\boldsymbol{g}_i, \boldsymbol{g}_j, \boldsymbol{g}_i \circ \boldsymbol{g}_j, \phi(i,j)])$ (החלק של $\boldsymbol{g}_i \circ \boldsymbol{g}_j$ הוא מכפלה שמייצגת אינטראקציות בין ה-representations, והחלק של $\phi(i,j)$ הוא כמו קודם מספר פי'צרים נוספים).

יעילות: כדי לשפר את יעילות המודל שהיא $O(T^4)$ (עם T הוא מספר המילים בטקסט), מבצעים pruning (כלומר בוחנים רק חלק מה-spans שיותר סביר שהם mentions). מזהים אלו מילים הן סבירות בתור mentions באמצעות שימוש ב-attention.

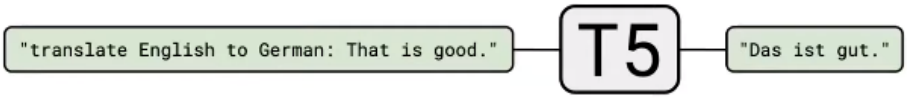
SOTA: נכון לזמן הקורס (תחילת 2021), המודל הזה נתן תוצאות טובות אך ה-SOTA הוא מודל BERT-based coref.

מודלי שפה גדולים

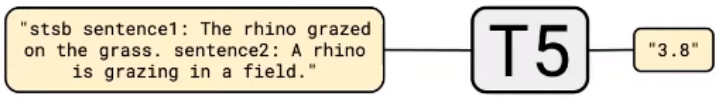
גישת TEXT-TO-TEXT

מודל T5 הוא מודל שפה שפורסם ע"י Google ב-2019 (אחרי BERT ו-GPT-1). במאמר שבו הוצג המודל לראשונה התייחסו לכך שקשה להשוות בין הביצועים של מודלים שפה pretrained שונים בגלל הבדלים מהותיים ביניהם (הדאטא עליהם הם אומנו, כמות הפרמטרים, בחירות ארכיטקטוניות וכו'). לכן, כותבי המאמר שואלים – בהנתן הסטטוס של השימוש הנוכחי ב-transfer learning ב-NLP (במובן של pretrained models), איך אפשר לקבוע מה גורם למודל להיות טוב, ובהנתן הידע הזה, איך אפשר להשתמש בכלים קיימים כדי לשפר את הביצועים שלהם אפילו יותר. כותבי המאמר מציגים את מודל T5 – **T**ext-**t**o-**T**ext **T**ransfer **T**ransformer.

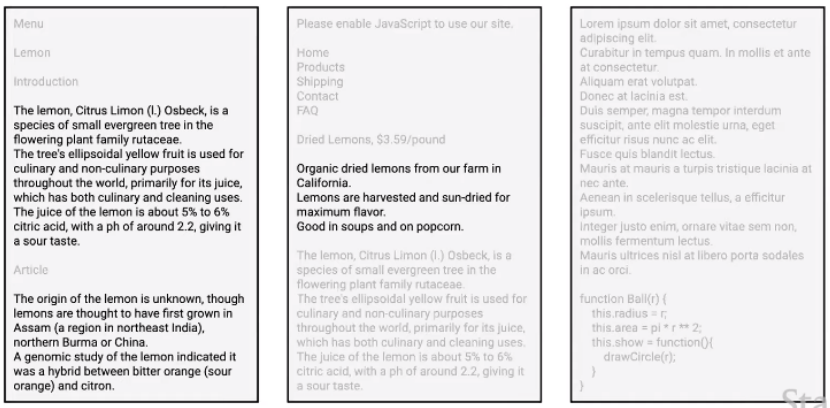
הרעיון היה להתייחס לכל בעיות ה-NLP באותו אופן – כלומר בתור בעיות text-to-text (בעיות בהן המודל מקבל טקסט כקלט ומוציא טקסט כפלט). למשל בבעיית תרגום, המודל מקבל כקלט טקסט המכיל משפט בשפה אחת (בנוסף להנחיות המסבירות איזו משימה עליו לבצע, המכונה task prefix) ומוציא כפלט טקסט שהוא תרגום של משפט הקלט לשפה אחרת:



אפשר להתאים כל סוג בעיה לסעיית טקסט-לטקסט, כולל בעיית רגרסיה (ע"י המרה של המספר שהמודל צריך לחזות לסטרינג):



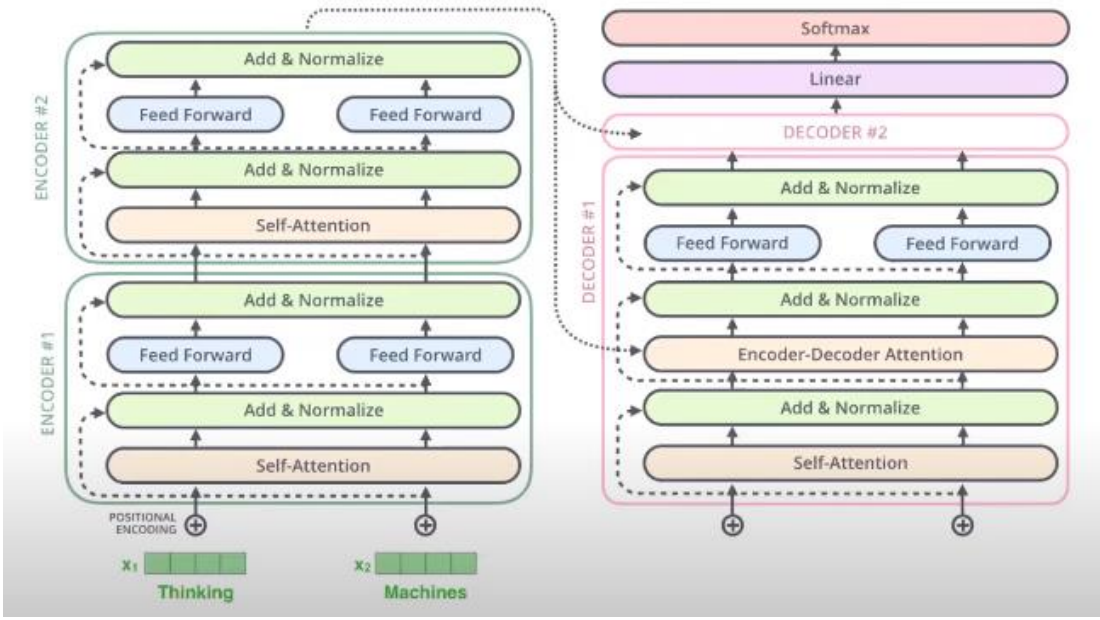
דאטאסט COMMON CRAWL



המטרה ביצירת הדאטאסט הזה היתה לחקור את ההשפעה של דאטאסט האימון על הצלחתם של מודלי שפה במשימות NLP שונות. לכן, גוגל יצרו את Common crawl, שהוא דאטאסט גדול מאוד שמכיל אפשרויות פילטור שמאפשרות לשלוט על הגודל והאיכות של הדאטא. מדובר בטקסט שהושג באמצעות scraping מכל רחבי האינטרנט וניקוי שלו (למשל הסרת תגיות HTML). כדי לייצר את הדאטאסט גוגל ניקו אותו (למשל הסירו משפטים שלא מסתיימים בסימן פיסוק, הסירו טקסטי Lorem Ipsum ממלאי מקום, הסירו כפילויות, הסירו קוד ועוד). הדאטאסט הסופי נקרא c4 וזמין ב-TenzorFlow.

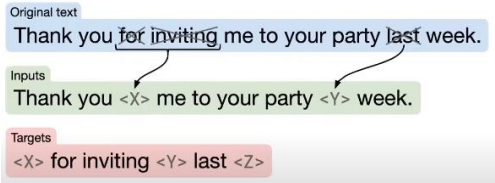
מודל T5

ארכיטקטורת מודל T5



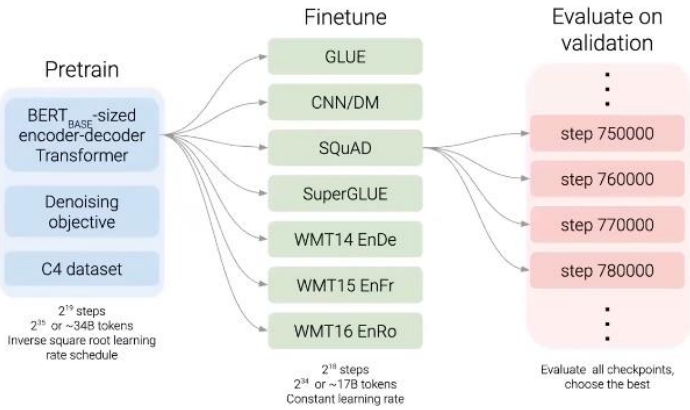
T5 PRETRAINING

הגדרת המשימה וטיפול בקלט: לוקחים את המשפט המקורי, בוחרים כמות מסוימת (מוגדרת מראש) של טוקנים מתוכו, ומחליפים אותם ב- sentinel token (עם אינדקס ייחודי לכל אחד מהם), ואז המטרה של המודל תהיה "למלא את החסר" (כלומר לחזות את המילים שצנזרנו). הרעיון הזה מכונה denoising objective.



אימון מודל ה-baseline: לוקחים מודל בגודל ששקול ל- $BERT_{base}$ $encoder-decoder$ (כלומר בסה"כ פי 2 פרמטרים מ-$BERT_{base}$ כיוון שאנחנו לוקחים גם את האנקודר וגם את הדיקודר), משתמשים ב-denoising objective (מוסבר בפסקה למעלה) ובדאטאסט של *C4* (השתמשו בבערך רבע מזמן האימון של $BERT_{base}$).

Fine-tuning: ביצעו *finetuning* על מגוון גדול של משימות *NLP* (על כל משימה בנפרד לחלוטין). שמרו את המודל ב-*check-points* שונים לאורך תהליך ה-*finetuning* כדי להיות מסוגלים לבחור את השלב שבו ביצועי המודל הם הטובים ביותר.



ניסויים שבוצעו עם T5

במסגרת המאמר ביצעו מגוון ניסויים שבחנו את ההשפעה של מגוון פרמטרים על ביצועי המודל (וריאציות של פונקציית *objective*, פילטורים שונים של הדאטאסט, גודל הדאטאסט שמשמש לאימון, אסטרטגיות *multi-task learning*, גודל המודל, גודל ה-*batch* ומספר ה-*training steps*). הניסויים הובילו ל-6 הבחירות העיצוביות הבאות (מודגמות לצד תוצאות הניסויים שהובילו לבחירה בהן):

Encoder-decoder architecture

Architecture	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39

Span prediction objective

Span length	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Baseline (i.i.d.)	**83.28**	19.24	80.88	71.36	**26.98**	**39.82**	**27.65**
2	**83.54**	19.39	**82.09**	**72.20**	**26.76**	**39.99**	**27.63**
3	**83.49**	**19.62**	**81.84**	**72.53**	**26.86**	39.65	**27.62**
5	**83.40**	19.24	**82.05**	**72.23**	**26.88**	39.40	**27.53**
10	82.85	19.33	**81.84**	70.44	**26.79**	39.49	**27.69**

C4 dataset

Dataset	Size	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ C4	745GB	83.28	**19.24**	80.88	71.36	**26.98**	**39.82**	**27.65**
C4, unfiltered	6.1TB	81.46	19.14	78.78	68.04	26.55	39.34	27.21
RealNews-like	35GB	**83.83**	**19.23**	80.39	72.38	**26.75**	**39.90**	**27.48**
WebText-like	17GB	**84.03**	**19.31**	**81.42**	71.40	**26.80**	**39.74**	**27.59**
Wikipedia	16GB	81.85	**19.31**	81.29	68.01	**26.94**	39.69	**27.67**
Wikipedia + TBC	20GB	83.65	**19.28**	**82.08**	**73.24**	**26.77**	39.63	**27.57**

Multi-task pre-training

Training strategy	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Unsupervised pre-training + fine-tuning	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	39.82	27.65
Multi-task training	81.42	**19.24**	79.78	67.30	25.21	36.30	27.76
Multi-task pre-training + fine-tuning	**83.11**	**19.12**	**80.26**	**71.03**	**27.08**	39.80	**28.07**
Leave-one-out multi-task training	81.98	19.05	79.97	**71.68**	**26.93**	39.79	**27.87**
Supervised multi-task pre-training	79.93	18.96	77.38	65.36	26.81	**40.13**	**28.04**

Bigger models trained longer

Scaling strategy	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Baseline	83.28	19.24	80.88	71.36	26.98	39.82	27.65
1× size, 4× training steps	85.33	19.33	82.45	74.72	27.08	40.66	27.93
1× size, 4× batch size	84.60	19.42	82.52	74.64	27.07	40.60	27.84
2× size, 2× training steps	**86.18**	19.66	**84.18**	77.18	27.52	**41.03**	28.19
4× size, 1× training steps	**85.91**	19.73	**83.86**	**78.04**	27.47	40.71	28.10
4× ensembled	84.77	**20.10**	83.09	71.74	**28.05**	40.53	**28.57**
4× ensembled, fine-tune only	84.05	19.57	82.36	71.55	27.55	40.22	28.09

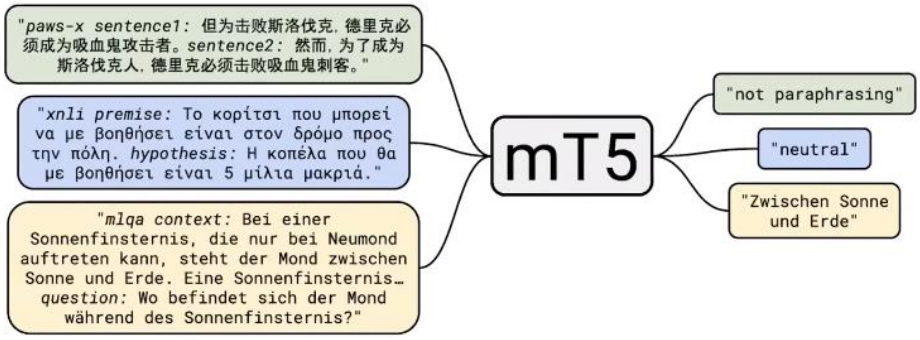
גודל המודל: אומנו 5 וריאציות של מודל *T5* שנבדלות בכמות הפרמטרים (בטבלה משמאל).

Model size variants

Model	Parameters	# layers	d_{model}	d_{ff}	d_{kv}	# heads
Small	60M	6	512	2048	64	8
Base	220M	12	768	3072	64	12
Large	770M	24	1024	4096	64	16
3B	3B	24	1024	16384	128	32
11B	11B	24	1024	65536	128	128

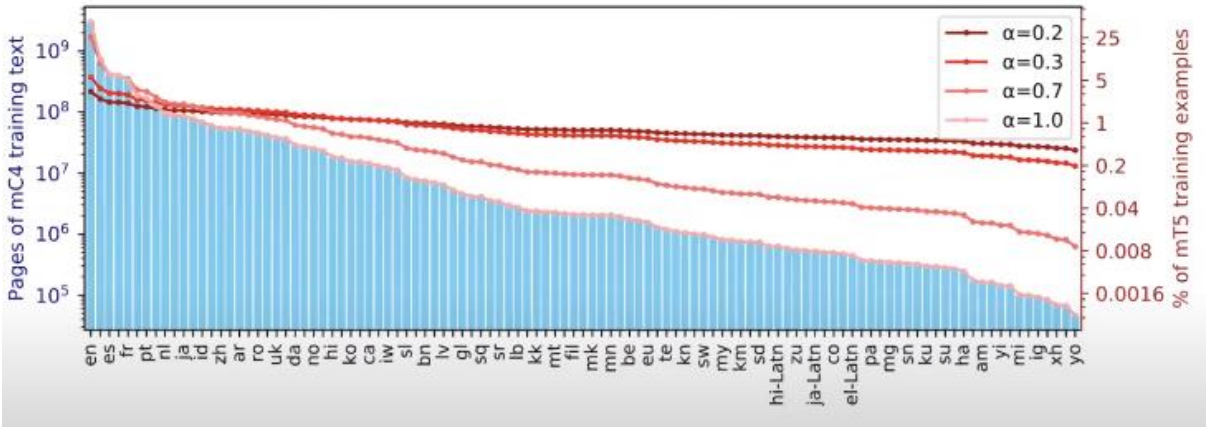
MT5 - מודל MULTI-LINGUAL

כאמור, *T5* אומן על דאטא באנגלית בלבד. מודל נוסף בשם *MT5* (*multilingual T5*) אומן בדיוק באותו אופן רק על דאטא רב-שפתי (הדאטאסט עליו הוא אומן הוא *mc4*, שהוא גרסה רב שפתית של *c4* שמכילה 101 שפות).

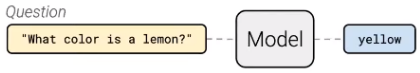


XNLI Zero-shot Accuracy	Urdu	Russian
α=0.2	**73.9**	81.2
α=0.3	73.5	81.5
α=0.7	71.7	**82.8**

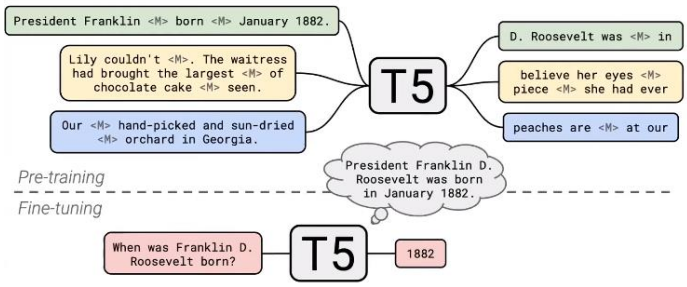
באופן טבעי, הדאטאסט של *mc4* מאוד לא מאוזן (הבדלים של כמה סדרי גודל) בכמות הדאטא שהוא מכיל עבור שפות שונות. בגלל שזה המצב, אסטרטגיה נפוצה היא שימוש ב-*temperature scaling*, שזה דוגמים מהדאטא של שפה מסוימת ע"י *scaling* של מספר הדגימות באמצעות שימוש ב"טמפרטורה" (כפי שניתן לראות בתמונה למטה). האיזון של הפרמטר α משמעותי – עבור ערכים נמוכים נשפר את התוצאות עבור שפות עם מעט דגימות (למשל אורדו), ועבור ערכים גבוהים יותר נשפר את התוצאות עבור שפות עם יותר דגימות (למשל רוסית), כפי שניתן לראות בתמונה משמאל.



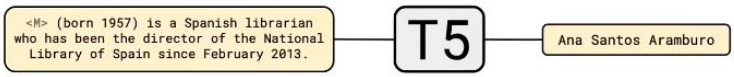
איזה ידע המודל רוכש במהלך ה-PRETRAINING?

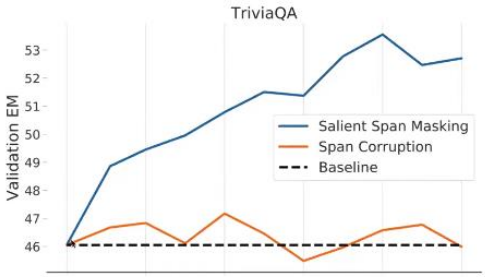


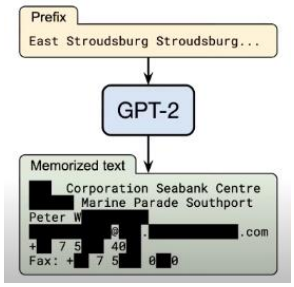
נתקלנו כבר בקורס במשימת *reading comprehension*, ובפרט גרסא יותר קשה שלה – *open domain question answering* (גרסה של המשימה שבה המודל לא מקבל טקסט אחד שהתשובה מופיעה בו, אלא מאגר של טקסטים שהוא צריך לעשות *retrieval* לתשובה מתוכו). במאמר של *Google* מוצגת גרסה חדשה – *Closed-book question answering*, משימה בה המודל לא מקבל טקסט שהתשובה מופיעה בו, אלא צריך להשתמש בידע שהוא "צבר" במהלך האימון כדי להיות מסוגל לקבל כקלט שאלה ולהוציא כפלט את התשובה לשאלה. במאמר טענו כי המשימה הזו היא דרך טובה לבחון את כמות ואיכות הידע שהמודל צבר בתהליך האימון. המודלים של *T5* הצליחו להשיג תוצאות טובות יחסית על דאטאסטים שנוצרו למשימת *open-domain questions* (שהשתפרו ככל שהיו למודל יותר פרמטרים), אך לא טובות כמו תוצאות *SOTA* של מודלים שאומנו למשימת *open-domain QA*.



כדי לנסות לשפר את יכולת צבירת הידע של המודל, השתמשו ב-*objective* בשם *salient span masking*, שבו במקום לעשות *masking* רנדומלי לקלט, עושים *masking* לטוקנים שמיצגים ישויות *entities* (שמות, מקומות, תאריכים וכו'). הגרף למטה מראה שככל שביצעו יותר *salient masking*, כך תוצאות המודל על דאטאסטים של *open-domain QA* השתפרו מאוד ועבור חלק מהמודלים גם עקפו את ה-*SOTA*.





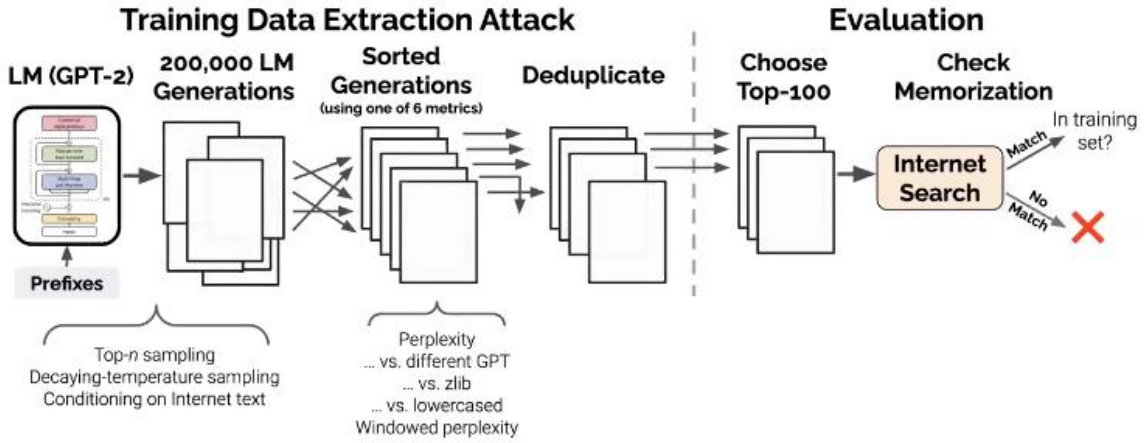


האם מודלי שפה גדולים משננים את דאטא האימון? במאמר נוסף של *Google* ניסו לבחון את השאלה הזו. הדרך בה עשו את זה היא לקחת את מודל *GPT-2* ולהזין לו *prefixes* (תחילית של משפט שאנחנו רוצים שהמודל ישלים). למשל בדוגמא משמאל ניתן לראות תחילית שגרמה למודל לייצר טקסט שמכיל פרטים (שם, טלפון, כתובת מייל) של בן אדם אמיתי. אלו פרטים שמופיעים רק 6 פעמים בכל רחבי האינטרנט, ולכן לא סביר שהמודל נתקל במידע הזה הרבה פעמים במהלך האימון שלו. זה גרם לכותבי המאמר להאמין כי המודל אכן "משנן" את המידע שהוא נתקל בו במהלך תהליך האימון.

כדי לפתור את הבעיה הזו (כיוון שאנחנו לא רוצים שהמודל ייצר טקסט עם מידע פרטי או חסוי), כותבי המאמר בחנו 3 דרכים לדגימת דאטא מ-*GPT2*:

1. לדגום בצורה אוטורגרסיבית
2. לדגום בצורה אוטורגרסיבית אבל עם *decaying temperature* (כלומר לעודד את המודל להיות יותר ויותר בטוח בפרדיקציות שלו לאורך תהליך ה-*sampling*)
3. לקחת טקסט אקראי מהאינטרנט ולהשתמש בו בתור *conditioning* למודל לפני שמבקשים ממנו לייצר טקסט

עבור כל אחת מהשיטות האלו, השתמשו בפייפליין שמוצג בתמונה למטה כדי לבחון באמצעות מטריקה כלשהי (פיתחו במאמר 6 מטריקות שונות, כולן מבוססות *perplexity*) כדי לבדוק כמה שינון המודל ביצע:

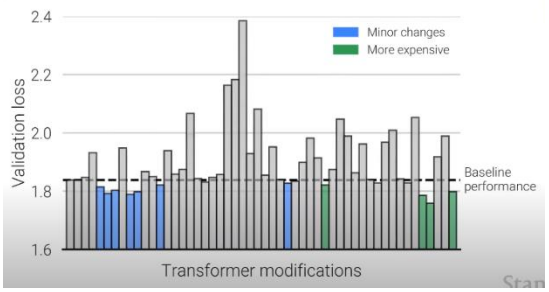


תוצאות הבחינה הזו הובילו למסקנה שיש הרבה דוגמאות בהן *GPT2* שינן מידע בעייתי (שמות פרטיים של אנשים פרטיים שלא מדברים עליהם בחדשות, פרטי קשר וכו'). זה היה נכון גם לקישורים לאתרים שהיו כתובים במבנה שהיה מאוד לא סביר לחזות (למשל רצף אותיות אקראיות). היקף השינון החמיר ככל שהמודל הכיל יותר פרמטרים וככל שהפרט ששונן הופיע יותר פעמים בדאטאסט.

שיפור ארכיטקטורת הטרנספורמר

מודלי שפה גדולים אמנם נותנים תוצאות מצוינות, אבל יש להם הרבה חסרונות – הם יותר יקרים חישובית להרצה, הם מצריכים יותר אנרגיה לאימון ולהרצה, והם לרוב גדולים מכדי שניתן יהיה להריץ אותם על *GPU* יחיד.

במאמר נוסף של *Google* הם בחנו מגוון הצעות לשיפור לארכיטקטורת הטרנספורמר (כפי שהיא מוצגת במאמר של *Attention is all you need*) ובחנו אם הן מביאות לשיפור הביצועים של המודל, למשל ביצוע *Factorization* ל-*embedding matrix*, שימוש ב-*shared embedding and softmax layer*, שימוש במספר סוגים שונים של פונקציות *softmax* או ב-*adaptive softmax*, שימוש בשיטות נורמליזציה שונות ובמכניזמים שונים ל-*attention* (או חלופות ל-*attention* כמו קונבולוציות), שימוש בסוגים שונים של פונקציות לא ליניאריות בשכבות ה-*FF*, ארכיטקטורות שלמות חלופיות ועוד. השתמשו ב-*settings* שהוצג עבור הניסויים שבוצעו בממר של *T5* כדי לבחון את ההשפעה של כל אחד מהשינויים האלו.



המסקנה שהם הגיעו אליה היא שיש שינויים שגרמו לשיפור משמעותי ביחס לבייסליין (הטרנספורמר מהמאמר המקורי), כאשר חלקם הם שינויים די מינוריים (למשל החלפה של הפונקציה לא ליניארית בפונקציה מאוד דומה), אבל חלקם מאוד יקרים חישובית (למשל הגדלה דרמטית של כמות הפרמטרים במודל). המסקנה הכוללת היתה שאף אחד מהשינויים לא הביא לשיפור מאוד משמעותי בתוצאות. זה היה נכון גם על מספר *downstream task* שהם בחנו (למשל *ROUGE* למשימת סיכום).

הוספת ידע למודלי שפה

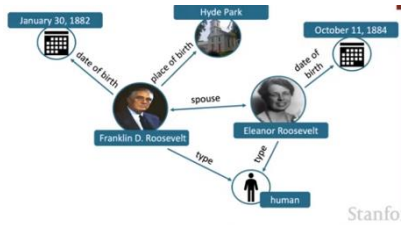
מה מודלי שפה יודעים?

מאמר מ-2019 בחן את *BERT-large* על מגוון השלמות משפטים כדי לבדוק אם הוא יחזה מילים שהן נכונות עובדתית. רוב הפרדיקציות שהוא ביצע היו הגיוניות (*made sense*), אבל לא כולן היו נכונות עובדתית (ראו דוגמא משמאל). **למה זה קורה?**

- **עובדות שהמודל לא מכיר:** סביר שחלק מהעובדות עליהן המודל נשאל לא הופיעו בדאטא האימון שלו.
- **עובדות נדירות:** יתכן שהמודל לא ראה מספיק דוגמאות מעובדה מסוימת בתהליך האימון כדי לשנן את המידע לגביה.
- **רגישות המודל:** יתכן שהמודל ראה את העובדה במהלך האימון, אבל הוא רגיש לניסוח של ה-*prompt* ששואל לגביה. למשל, יתכן שהמודל יודע להשלים נכונה משפט כמו "*x was made by y*" אבל לא "*x was created by y*".

העובדה שמודלי שפה לא מצליחים לשלוף מידע באופן אמין הוא אתגר משמעותי בימינו. מחקרים עדכניים מצאו שמודלי שפה יכולים לשלוף כמות וסוגים מסוימים של ידע, אבל זה עוד רחוק מלהיות אמין.

למה חשוב שמודלי שפה יהיו מודעים לידע *knowledge-aware*: וקטורי *representation* שמיוצרים ע"י מודלי שפה *pretrained* יכולים לתרום ל-*downstream tasks* שמנצלות ידע. למשל, משימת *relation extraction* בה מנסים לחלץ מידע על הקשרים בין ישויות בתוך המשפט, יכולה להתרם מאוד אם היה למודל ידע על הישויות שמוזכרות במשפט. גישה קיצונית יחסית טוענת שמודלי שפה יוכלו בסופו של דבר להחליף מאגרי מידע *knowledge bases*, ואז בתור משתמשים, במקום לתשאל מאגר מידע עבור עובדה מסוימת (למשל באמצעות *SQL*), נתשאל את מודל השפה באמצעות פרומפט בשפה טבעית.

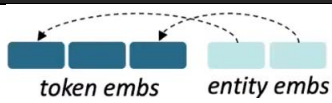


טרמינולוגיה של מאגרי נתונים: מאגרי נתונים מבוססי גרפים *graph knowledge bases* הם גרפים המכילים מידע על ישויות והקשרים ביניהם. כל גרף שכזה מורכב משלישיות *triplets* שכל אחת מהן מכילה ישות *subject entity* , יחס *relation* ואובייקט *object entity* (מכונים לעתים *pair entity*, *relation*, *tail entity*).

יתרונות וחסרונות בשימוש במודלי שפה על פני מאגרי נתונים מסורתיים: יתרן ראשון הוא מודלי שפה מאומנים על כמויות גדולות מאוד של דאטא לא מובנה *unstructured* ולא מתויג *unlabeled*, בעוד יצירת מאגרי נתונים מסורתיים דורשת אנוטציה ידנית או פייפלייני *NLP* מורכבים שיוצרים מידע בפורמט מובנה מתוך הדאטא. בנוסף, מודלי שפה מאפשרים גמישות באופן בו אנחנו מתשאלים את מאגר הנתונים (כיוון שהם מאפשרים כתיבת שאילתות בשפה טבעית). עם זאת, ישנם חסרונות / אתגרים משמעותיים בשימוש במודלי שפה במקום מאגרי נתונים מסורתיים: למשל, מודלי שפה הם קשים לפירוש *interpret*. בנוסף, קשה לסמוך על מודלי שפה כיוון שהם עשויים לייצר תוצאה מאוד אמינה אך שגויה עובדתית (בשונה ממאגרים מסורתיים, שאם הם לא יודעים את התשובה לשאילתה

מסוימת, הם יתנו אינדיקציה מפורשת לכך ולא ינסו "להמציא"). חסרון נוסף הוא שמודלי שפה הם קשים לעדכון והתאמה *modify* (למשל, קשה לעדכן או להסיר מידע שמקודד בהם).

שיטות להוספת מידע למודלי שפה - באופן כללי, יש שלוש שיטות נפוצות להוספת מידע למודלי שפה:

הדגמה	שיטה	יתרונות	חסרונות
	הוספת pretrained entity embeddings (למשל ERNIE או KnowBERT)	יחסית קלה למימוש על ארכיטקטורות קיימות כדי לנצל pretraining של גרפי ידע	דרך עקיפה להוסיף מידע ולכן אינה אינטרפטבילית
	שימוש בזכרון חיצוני (למשל KGLM או kNN-LM)	תומכת בעדכון (בעת הצורך) של המידע העובדתי, ובנוסף אינטפרטבילית	יותר מורכבת למימוש ומצריכה כמות מאוד גדולה של זכרון
	מודיפיקציות לדאטא האימון (למשל WEKM)	לא מצריכה שינויים במודל או חישוב נוסף	זו עדיין שאלה פתוחה אם היא נשארת אפקטיבית כאשר המודל משתנה

שיטות הוספת PRETRAINED ENTITY EMBEDDINGS

עובדות על העולם מתייחסות פעמים רבות לישויות entities (למשל, וושינגטון היה הנשיא הראשון של ארצות הברית). Embeddings שנוצרים ע"י מודלי שפה לא מאופיינים ביכולת מובנית שמאפשרת להם לקשר בין ישויות (למשל, יש embeddings שונים עבור "USA" ו-"United states of America" למרות שמדובר באותה ישות). לכן, הרעיון בשיטה הזו היא לייצר embedding יחיד לכל ישות, ואז בכל פעם שנתקל בטקסט באחד מהשמות שמייצגים את הישות, נשתמש תמיד באותו embedding שמקודד את כל המידע לגביה. גישה שכזו יכולה להיות שימושית אם ורק אם מודל השפה מציג ביצועים טובים על משימת **entity linking** (משימה בה המודל נדקש לקשר בין אזכורים mentions בטקסט לבין ישויות במאגר מידע).

Entity embeddings הם כמו word embeddings אבל עבור ישויות במאגר מידע. יש הרבה טכניקות נפוצות לאימון מודל שמייצר entity embeddings, למשל:

- שיטות מבוססות גרפי ידע (למשל TransE)
- שיטות Word-entity co-occurrence (למשל Wikipedia2Vec)
- שימוש בקידוד שיוצר מודל טרנספורמר לתיאור של הישות (למשל BLINK)

$$\text{George Washington} = \begin{pmatrix} 0.111 \\ -0.345 \\ 0.876 \\ -0.201 \end{pmatrix}$$

בהנתן entity embeddings, עולה השאלה – איך אנחנו יכולים לשלב אותם אם הם הגיעו מ-embedding space אחר? התשובה לשאלה הזו היא ללמוד fusion layer שמאחדת את ההקשר ואת המידע על הישות. עושים זאת באופן הבא:

$$\boldsymbol{h}_j = F\left(\boldsymbol{W}_t \boldsymbol{w}_j + \boldsymbol{W}_e \boldsymbol{e}_k + b\right)$$

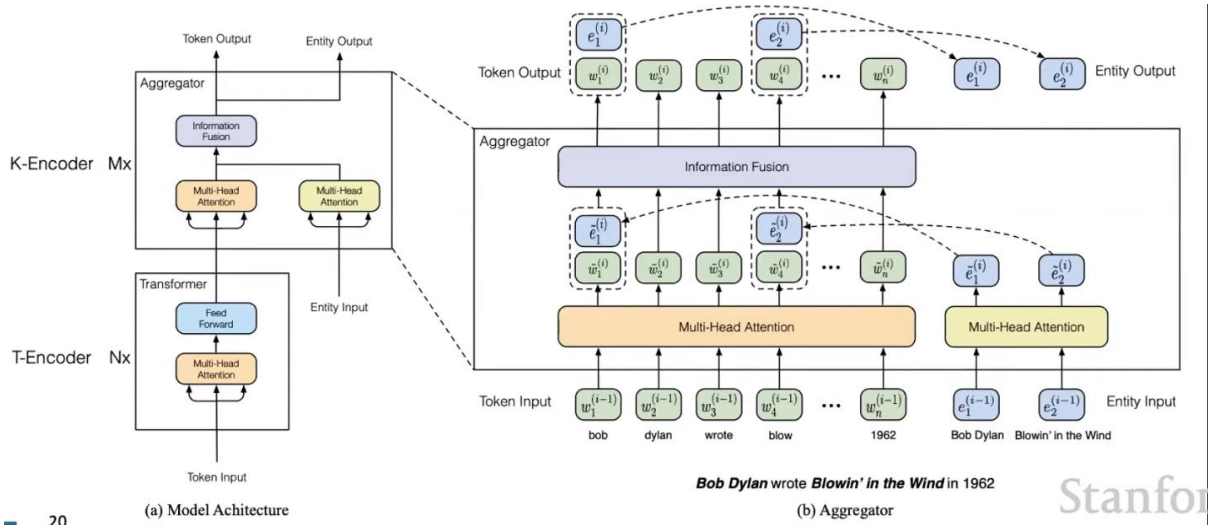
אנחנו מניחים שיש alignment ידוע (known) בין הישויות והמילים במשפט כך שמתקיים $e_k = f(w_j)$ (w_j הוא ה-embedding של המילה ה-j במשפט, למשל הייצוג של המילה Washington, e_k הוא ה-*entity embedding* התואם לה, למשל הייצוג של הישות *George Washington*). המטריצות W_t, W_e הן מטריצות משקולות נלמדות, b הוא פרמטר *bias*, ו-F היא פונקצית אקטיבציה כלשהי.

ERNIE

מודל *ERNIE – Enhanced Language Representation with Informative Entities* (2019) – מודל שמורכב ממספר חלקים:

- *text encoder* שבנוי מ-*multi-layer bidirectional transformer encoder* על המילים במשפט (במאמר השתמשו ב-*BERT* אבל אפשר לממש את זה גם על מודלים אחרים באותה ארכיטקטורה).
- *Knowledge encoder* שהוא ערמת בלוקים שמורכבים מ:
  - שני *multi headed attentions* על ה-*entity embeddings* וה-*token embeddings*.
  - שכבת *fusion* שמאחדת את הפלטים משכבות ה-*attention* (מאוד דומה לרעיון הכללי שראינו קודם, רק שעכשיו אנחנו מייצרים גם פלט של שכבת ה-*fusion*. האינדקס *i* מייצג את הבלוק ה-*i* ב-*knowledge encoder*):

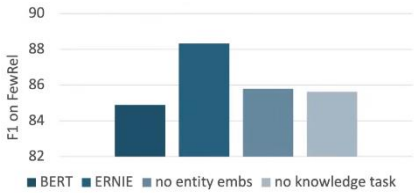
$$\boldsymbol{h}_j = \sigma\left(\widetilde{\boldsymbol{W}}_t^{(i)}\widetilde{\boldsymbol{w}}_j^{(i)} + \widetilde{\boldsymbol{W}}_e^{(i)}\widetilde{\boldsymbol{e}}_k^{(i)} + \widetilde{\boldsymbol{b}}^{(i)}\right)$$
$$\boldsymbol{w}_j^{(i)} = \sigma\left(\boldsymbol{W}_t^{(i)}\boldsymbol{h}_j + \boldsymbol{b}_t^{(i)}\right)$$
$$\boldsymbol{e}_k^{(i)} = \sigma\left(\boldsymbol{W}_e^{(i)}\boldsymbol{h}_j + \boldsymbol{b}_e^{(i)}\right)$$



אימון *ERNIE* בוצע באמצעות שלוש משימות:

- *Masked language model & next sentence prediction* (אותן משימות שעליהן אומן *BERT*)
- *Knowledge pretraining task* – משימה בה מצנזרים (באמצעות טוקן *MASK*) באופן אקראי *alignments* בין טוקן לישות, ואז חוזים את הישות המתאימה לטוקן מבין הישויות המופיעות במשפט.

$$p(e_j \mid w_i) = \frac{\exp(\boldsymbol{W}\boldsymbol{w}_i \cdot \boldsymbol{e}_j)}{\sum_{k=1}^{m} \exp(\boldsymbol{W}\boldsymbol{w}_i \cdot \boldsymbol{e}_k)}$$
$$\mathcal{L}_{\text{ERNIE}} = \mathcal{L}_{MLM} + \mathcal{L}_{NSP} + \mathcal{L}_{dEA}$$



בניסוי *ablation* (ראו בגרף משמאל) השוו בין *BERT* ו-*ERNIE* לבין גרסאות של *ERNIE* שאומנו ללא *entity embeddings* או ללא *knowledge mask*. כיוון ששתי הגרסאות האלו פחות טובות מ-*ERNIE* (אם כי יותר טובות מ-*BERT*), המסקנה היתה ששתי משימות האימון **ביחד** עושות את ההבדל המשמעותי בין *ERNIE* ל-*BERT*.

היתרונות של *ERNIE* הם שהוא משלב בין מידע על הישות למידע על ההקשר באמצעות שכבות *fusion* ו-*knowledge pretraining task*, ובנוסף משפר את הביצועים של *downstream tasks* משמשמשות בידע. **האתגרים** שהמודל מתמודד איתם הם שהוא צריך לקבל כקלט (גם עבור משימות *downstream*) טקסט <u>שהישויות בו מסומנות</u>. בנוסף, הוא מצריך *pretraining* נוסף שהוא מאוד יקר.

KNOWBERT

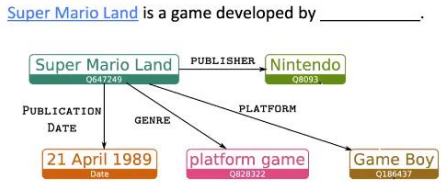
הרעיון במודל הוא לבצע *pretrain* ל-*integrated entity linker (EL)* בתור הרחבה ל-*BERT*. פונקציית ה-*loss* תוגדר באופן הבא - $\mathcal{L}_{\text{KnowBERT}} = \mathcal{L}_{NSP} + \mathcal{L}_{MLM} + \mathcal{L}_{EL}$ כאשר $\mathcal{L}_{EL}$ הוא ה-*loss* של ה-*entity linker* (שמדהה ישויות על קבוצה של מועמדים "קשים", לא רק אלו שמופיעים במשפט). כעת עבור *downstream tasks* ה-*EL* מזהה ישויות כך שבשונה מ-*ERNIE* אין צורך באנוטציה ידנית של ישויות בטקסט הקלט. ללמוד *EL* עשוי לשפר את הידע שהמודל מקודד, והמודל הזה עוקף בביצועים גם את *ERNIE* על *downstream tasks*. בדומה ל-*ERNIE*, גם *KnowBERT* משתמש בשכבת *fusion* כדי לשלב בין מידע על הישות ומידע על ההקשר ומוסיף את הידע שנרכש בשלב ה-*pretraining*.

שיטות המשתמשות בזכרון חיצוני

השיטות שהזכרנו עד עכשיו השתמשו ב-*pretrained entity embeddings* כדי לקודד ידע עובדתי על העולם מתוך מאגרי נתונים לתוך מודל השפה. אך יש דרכים יותר ישירות לספק למודל ידע עובדתי – למשל לתת למודל גישה לזכרון חיצוני (גישה באמצעות *key-value* לשלשות בגרף נתונים). היתרונות בשיטות כאלו היא שאפשר לתמוך ב"הזרקת" מידע ועדכון הידע של המודל בכל שלב רצוי. בנוסף, זה יכול לחסוך את הצורך בביצוע *pretraining* נוסף למודל. יתרון נוסף הוא שהתוצר משמעותית יותר *interpretable*.

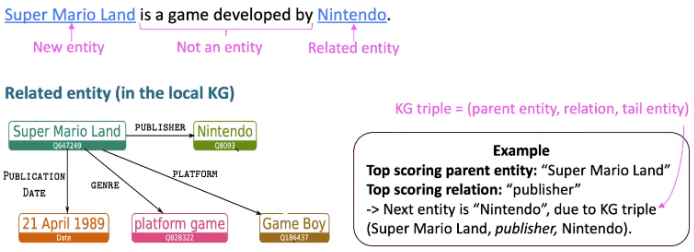
KGLM

הרעיון במודל היא לאפשר למודל שפה להשתמש בגרף ידע. מודלי שפה בד"כ חוזים את המילה הבאה ע"י ביצוע החישוב ההסתברותי הבא - $P\left(x^{(t+1)} \mid x^{(t)}, \dots, x^{(1)}\right)$, where $x^{(1)}, \dots, x^{(t)}$. בשיטת *KGLM* המשימה היא לחזות את המילה הבאה תוך שימוש במידע על ישויות, באמצעות החישוב $P\left(x^{(t+1)}, \varepsilon^{(t+1)} \mid x^{(t)}, \dots, x^{(1)}, \varepsilon^{(t)}, \dots, \varepsilon^{(1)}\right)$ כאשר $\varepsilon^{(t)}$ הוא קבוצת הישויות מתוך גרף הידע שמוזכרות בנקודת הזמן t.

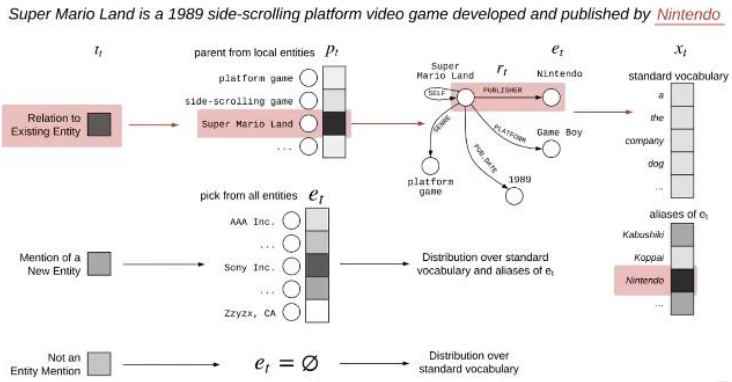


בשיטת *KGLM* בונים גרף ידע "לוקאלי" תוך כדי האיטרציה על המשפט. גרף ידע לוקאלי הוא תת קבוצה של גרף הידע המלא שמכילה רק ישויות שרלוונטיות למשפט עליו אנחנו עוברים. חשוב לשים לב שהשיטה מניחה שהישויות במשפט ידועות בזמן האימון, כלומר זה כן מצריך אנוטציה ידנית של הישויות במשפט.

מתי המודל צריך להשתמש בגרף הידע הלוקלי כדי לחזות את המילה הבאה? איך ש-*KGLM* מתייחס לבעיה היא שהמודל לא חוזה רק את המילה הבאה אלא גם את הסוג שלה (לא ישות / ישות קיימת בגרף הידע הלוקלי *related entity* / ישות חדשה *new entity*). למשל בדוגמא בתמונה למעלה המודל אמור לחזות שהמילה החסרה היא *Related entity*. בהנתן זה שהמודל חזה שהמילה הבאה אמורה להיות ישות קיימת, הוא צריך להשתמש ברמזים מתוך ההקשר של המשפט כדי לקבוע איזו ישות קיימת בגרף הידע הלוקאלי היא הישות המתאימה להיות המילה הבאה. הוא עושה זאת ע"י לחזות את סוג הקשר הסביר ביותר, ואז בוחר את הישות (*tail entity*) המתאימה מתוך הגרף הלוקאלי.



- <u>זיהוי ישות קיימת:</u> המשימה הראשונה היא **למצוא את ההורה *parent* והקשר *relation* המתאימים ביותר בגרף הידע <u>הלוקאלי</u>**, תוך שימוש במצבים החבויים של ה-*LSTM* ו-*embeddings* ה-*pretrained* של הישות ושל היחס. מחשבים זאת ע"י $P(p_t) = \text{softmax}\left(\boldsymbol{v_p} \cdot \boldsymbol{h_t}\right)$ כאשר p_t היא ישות ה"הורה", $\boldsymbol{v_p}$ הוא ה-*entity embedding* שלה, $\boldsymbol{h_t}$ הוא המצב החבוי מתוך ה-*LSTM*. עושים חישוב דומה גם עבור ה-*relation embedding*. כעת, כדי **לחזות את הישות הבאה**, לוקחים את ה-*tail entity* ויוצרים את שלישיית גרף הידע (*KG triple*) שמכילה את ישות ההורה שקיבלה את הציון הגבוה ביותר, היחס שקיבל את הציון הגבוה ביותר וה-*tail entity*. כעת, **כדי לחזות את המילה הבאה** בוחרים את הטוקן הסביר ביותר מתוך אוצר המילים שהורחב כדי לכלול אליאסים *aliases* של הישות שזוהתה (דרכים להתייחס לישות בטקסט, למשל *Nintendo* עשויה להיות מכונה בטקסט "*Nintendo Company*" או משהו אחר בסגנון).
- <u>זיהוי ישות חדשה:</u> המשימה הראשונה היא **למצוא את הישות המתאימה ביותר בגרף הידע <u>המלא</u>**, תוך שימוש במצבים החבויים של ה-*LSTM* ו-*embeddings* ה-*pretrained* של הישות. כדי **לחזות את הישות הבאה**, לוקחים את הישות שקיבלה את הציון הגבוה ביותר בשלב הקודם. **כדי לחזות את המילה הבאה** בוחרים את הטוקן הסביר ביותר מתוך אוצר המילים שהורחב כדי לכלול אליאסים *aliases* של הישות שזוהתה.



- <u>זיהוי מילה שאינה ישות:</u> לא מנסים לחזות את הישות הבאה, ופשוט חוזים מילה באופן "רגיל" (כפי שמודל שפה עושה – המילה הסבירה ביותר מתוך אוצר המילים).

מודל *KGLM* עוקף את *GPT2* במשימות השלמת עובדות. בהשוואה ל-*GPT2*, *KGLM* נוטה לחזות טוקנים יותר ספציפים (*GPT2* נוטה לחזות טוקנים יותר גנרים או פופולריים). ניסויים בהם עדכנו את גרף הידע הראו שהשינוי בגרף אכן גרם לשינוי בפרדיקציות של המודל.

KNN-LM

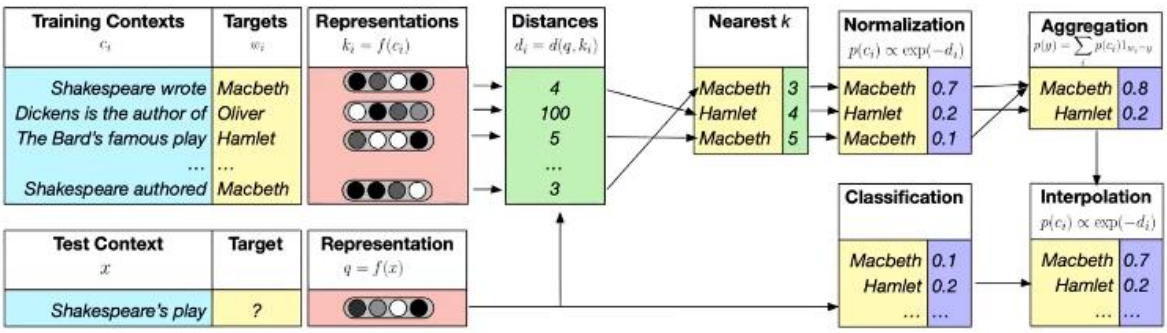
Nearest Neighbor Language Model הוא מודל שהרעיון שלו הוא שללמוד דמיון *similarity* בין רצפי טקסט זה יותר קל מאשר לחזות את המילה הבאה, למשל: Example: *"Dickens is the author of _____"* ≈ *"Dickens wrote_____"* זו דוגמא למשפטים שסביר שנגמרים באותה מילה, ולכן לזהות שהם דומים יהיה שימושי. זה נכון במיוחד עבור "*long-tail patterns*" או עובדות נדירות.

כדי לעשות זא, מאחסנים א הייצוגים של רצפי טקסט בתור מאגר שכנויות. בזמן *inference*:

- מוצאים את k רצפי המילים הקרובים ביותר במאגר
- מאחזרים את הערכים המתאימים מתוכם (המילה הבאה הרלוונטית) מכל אחד מ-k הרצפים
- מאחדים בין ההסתברויות שהפקנו מתוך ה-*KNN* להסתברויות של מודל השפה כדי לבחור את הפרדיקציה הסופית, באופן הבא - $P(y \mid x) = \lambda P_{kNN}(y \mid x) + (1-\lambda)P_{LM}(y \mid x)$.

Example: Shakespeare's play _____ ....

Task: Predict the next word with kNN-LM



שיטות המבצעות מודיפיקציה לדאטא האימון

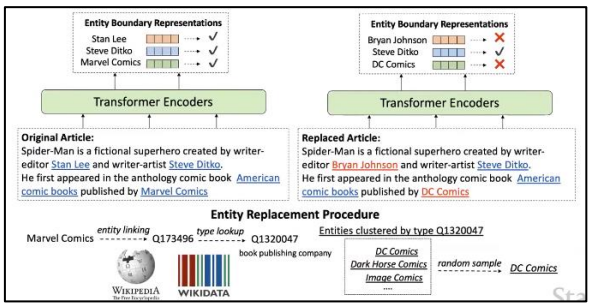
השיטות שראינו עד כה משלבות ידע באופן מפורש באמצעות *pretrained embeddings* ו/או שימוש בזכרון חיצוני. בשיטות הבאות נציג דרכים לשלב ידע באופן אימפליציטי ע"י להכיר למודל משימות אימון חדשות שדורשות ידע עובדתי. היתרונות בשיטות כאלו הן שאין צורך בזכרון או בחישוב נוסף, ושאין צורך לעשות מודיפיקציה לארכיטקטורה של המודל.

WKLM

מודל *Pretrained encyclopedia – Weakly supervised knowledge-pretrained LM* הוא מודל שהרעיון מאחוריו הוא אימון כדי ללמד אותו להבחין בין ידא אמיתי ושקרי. כדי לעשות זאת, מחליפים את האזכורים *mentions* בטקסט באזכורים שמתייחסים לישויות אחרות מאותו סוג (זה נחוץ כדי להבטיח שמהשפטים יצאו תקינים לשונית) כדי לייצר אמירות המייצגות ידע שקרי. המודל לומד לחזות אם ישות הוחלפה או לא.

True knowledge statement: J.K. Rowling is the author of Harry Potter.

Negative knowledge statement: J.R.R. Tolkien is the author of Harry Potter.



המודל משתמש ב-*entity replacement loss* כדי לאמן את המודל להבחין בין אזכורים אמיתיים ושקריים, המוגדר באופן הבא:

$$\mathcal{L}_{\text{entRep}} = \mathbb{1}_{e\in\varepsilon^+} \log P(e \mid C) + (1 - \mathbb{1}_{e\in\varepsilon^+}) \log\big(1 - P(e \mid C)\big)$$

כאשר e היא ישות, C הוא ההקשר ו-ε^+ מייצג אזכור אמיתי. פונקציית ה-*loss* הסופית משלבת בין ה-*loss* הסטנדרטי של משימת *masked language model (MLM)* (שהוא ברמת הטוקן) לבין ה-*loss* הנ"ל (שהוא באמת הישות) באופן הבא –

$$\mathcal{L}_{WKLM} = \mathcal{L}_{MLM} + \mathcal{L}_{entRep}$$

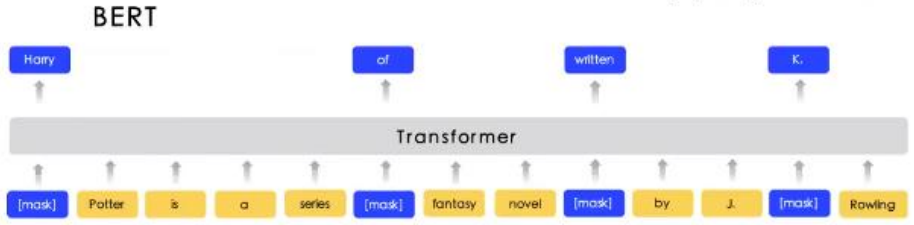
Model	SQuAD (F1)	TriviaQA (F1)	Quasar-T (F1)	FIGER (acc)
WKLM	91.3	56.7	49.9	60.21
WKLM w/o MLM	87.6	52.5	48.1	58.44
BERT + 1M Updates	91.1	56.3	48.2	54.17

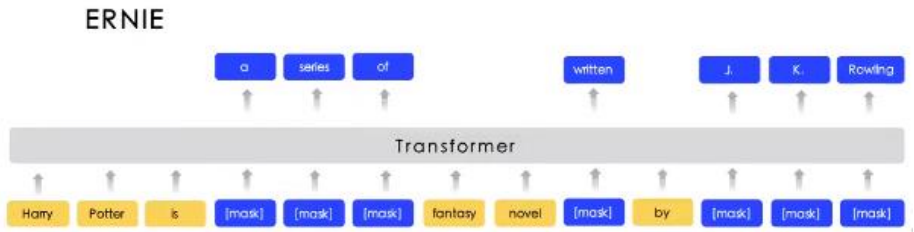
Much worse without MLM

תוצאות: השיטה הזו משפרת גם את *BERT* וגם את *GPT2* במשימת *fact completion*, ומשפרת את *ERNIE* במשימת *entity typing*. ניסויי *ablation* הראו ש-*MLM loss* הוא נחוץ לביצועים במשימות *downstream* וגם ש-*WKLM* משיג ביצועים טובים יותר בהשוואה להגדלת זמן האימון עם *MLM loss* בלבד.

למידת INDUCTIVE BIASES באמצעות MASKING

- מאמר שמציג מודל אחר בשם *ERNIE* (שאינו המודל שהוצג בתחילת הפרק) מ-2019, *Enhanced representations through knowledge integration*. המאמר הזה מראה דרך לשפר ביצועים על משימות *downstream* בסינית באמצעות *masking* באמת הביטוי *phrase* וברמת הישות *entity*.
- מאמר מ-2020 בשם *How much knowledge can you pack into parameters of language models* שמשתמש ב-*salient span masking* כדי למסך *salient spans* (למשל *named entities* ותאריכים), שהראה ששיטת המיוך הזאת מסייעת למודל *T5* במשימות *QA* (*closed domain*).





אוליואציה של ידע במודלי שפה

The theory of relativity was developed by [MASK].
The native language of Mammootty is [MASK].
Ravens can [MASK].
You are likely to find a overflow in a [MASK].

מאמר – ***Language Model Analysis (LAMA) Probe*** שפורסם ב-2019 שבוחן כמה ידע רלציוני (*commonsense*) ועובדתי) מודלי שפה ("רגילים", שלא עברו מודיפיקציה, *finetuning* או התאמה כלשהי) מכילים. מה שעשו במאמר זה ליצור ידנית קבוצה של *cloze statements* שמעריכים את יכולותו של המודל לחזות את הטוקן החסר (ראו דוגמאות משמאל). המטרה היתה להשוות בין מודלי שפה ל-*supervised relation extraction (RE)* ול-*QA systems*, וגם להעריך את הידע שקיים במודלי שפה שונים (שכל אחד מהם גם אומן על קורפוסים אחרים).

Mean precision at one (P@1)

Corpus	DrQA	RE baseline	fairseq-fconv	Transformer-XL	ELMo	ELMo (5.5B)	BERT-base	BERT-large
Google-RE	-	7.6	2.6	1.6	2.0	3.0	9.8	**10.5**
T-REx	-	**33.8**	8.9	18.3	4.7	7.1	31.1	32.2
ConceptNet	-	-	3.6	5.7	6.1	6.2	15.6	**19.2**
SQuAD	**37.5**	-	3.6	3.9	1.6	4.3	14.1	17.4

הגבלות של ה-*LAMA probe*:

- קשה להבין **למה** מודלים משיגים ביצועים טובים (למשל, יכול להיות ש-*BERT* פשוט חוזה טוקן פופולרי בגלל שהוא שינן דפוסי *co-occurrence* והוא לא באמת "מבין" את ה-*cloze statement*. גם יתכן שמודלי שפה פשוט מזהים דמיון בין ה-*surface form* של אובייקט מסוים לבין האובייקט עצמו (למשל הישות Pope Clement VII מחזיק בתפקיד Pope).
- מודלי פה מאוד רגישים לניסוח של ה-*statement* (ו-*LAMA* מכיל רק טמפלייט מוגדר ידנית יחיד לכל יחס).

Native language of French-speaking actors according to BERT

Person Name	BERT
Jean Marais	French
Daniel Ceccaldi	Italian
Orane Demazis	Albanian
Sylvia Lopez	Spanish
Annick Alane	English

LAMA-UHN – LAMA-Unhelpful Names – גישה שהרעיון שלה היא להסיר מתוך *LAMA* דוגמאות שעשויות לקבל מענה ללא ידע רלציוני (כמו הדוגמא עם האפיפיור בראש העמוד), כיוון שה-*BERT* עשוי להסתמך על *surface-forms* של האובייקט כדי לבצע פרדיקציות לגביו. זה יוצר גרסה מעט יותר "קשה" ל-*LAMA*.

שימוש ב-*LAMA-UHN* גורם לציון של *BERT* לרדת בבערך 8 אחוז בהשוואה ל-*LAMA*.

פיתוח פרומפטים טובים יותר לתשאול ידע ממודלי שפה – מאמר מ-2020 שמציע שמודלי שפה עשויים לדעת עובדה מסוימת, אבל עדיין להכשל במשימות כמו *LAMA* בגלל הניסוח של הפרומפט עצמו. במאמר יצרו עוד פרומפטים עבור *LAMA* ע"י כריית טמפלייטים מתוך ויקיפדיה וניסוח מחדש של פרומפטים באמצעות *back-translation*. הם ביצעו *ensemble* לפרומפטים כדי להגדיל את הגיוון בהקשרים שכל עובדה יכולה להיות מוצגת בהם. הם מצאו שהביצועים של *BERT* השתפרו בכ-7% בשימוש ב-*top performing query* לכל יחס, וששימוש ב-*ensembling* תרם 4% שיפור נוספים. הם מצאו ששינויים קטנים בשאילתא עשויים לגרום לשינוי משמעותי בתוצאות (משמאל) – כלומר מודלי שפה מאוד רגישים לניסוח השאילתא.

ID	Modifications	Acc. Gain
P413	x plays in→at y position	+23.2
P495	x was created→made in y	+10.8
P495	x was→is created in y	+10.0
P361	x is a part of y	+2.7
P413	x plays ~~in~~ y position	+2.2

Knowledge-driven downstream tasks – גישה אחרת, בה מודדים כמה טוב מודל שפה מועשר-בידע מצליח להעביר את הידע שלו ל-*downstream task*. בשונה משימוש ב-*probes*, שיטת האוולוואציה הזו דורשת בד"כ *finetuning* למודל השפה על המשימה הרצויה (כמו למשל הערכת *BERT* על משימות *GLUE*). משימות נפוצות לגישה זו הן משימות *knowledge-intensive*:

Relation extraction
- Example: *[Bill Gates] was born in [Seattle]; label: city of birth*

Entity typing
- Example: *[Alice] robbed the bank; label: criminal*

Question answering
- Example: *"What kind of forest is the Amazon?"; label: "moist broadleaf forest"*

שיקולים חברתיים ואתיים בשימוש ב-AI

התפיסה השגויה הרווחית היא ששפה נוגעת למילים למשמעות שלהן – אך המרצה (*Yulia Tsvetkov*) חושבת שזה לא נכון, וששפה נוגעת לאנשים ולמה שהם מתכוונים. זה גורם לכך שמערכות *AI* שמשתמשות בשפה מקודדות בתוכן את האופן שבני אדם מתקשרים, ובתוך כך גם הטיות וסטראוטיפים. ההחלטות שאנחנו מקבלים בנוגע לדאטא, השיטות והכלים שאנחנו בוחרים להשתמש בהם יוצרים השפעה ישירה על אנשים וקהילות שמשתמשים או מושפעים מהמערכות שאנחנו מפתחים. לאורך הפרק יסומנו בורוד שאלות שהמרצה סבורה כל חוקר או מדען נתונים העוסקים בפיתוח מערכות *AI* צריכים לשאול את עצמם בנוגע למערכות שהם מפתחים.

אתיקה הוא תחום פילוסופי בו נידונות שאלות כמו "מהו המעשה הראוי שחובה לעשותו" ו"מהי המידה הטובה", ובאופן כללי דנים בו בהבחנה בין טוב ורע. אתיקה משתנה לאורך הזמן ביחד עם הערכים והאמונות של אנשים בחברה. חוקי ≠ אתי – כלומר, יכולים להיות מעשים אתיים שאינם חוקיים, או מעשים חוקיים שאינם אתיים.

דוגמאות לשימושים לא אתיים בדאטא

דוגמא – מסווג *IQ*: נניח שאנחנו מסוגלים לבנות מערכת לסיווג *IQ*, שמקבלת מידע על בן אדם (למשל תמונות וטקסט) ומזהה מהו ה-*IQ* שלו בדיוק גבוה מאוד. מי יכול להרוויח מקיום של מערכת כזו? יש גורמים רבים שעשויים להרוויח מקיומו של מסווג שכזה (למשל אוניברסיטאות, שיוכלו לקבל רק סטודנטים בעלי *IQ* גבוה). נניח שהמסווג מצליח להשיג דיוק של 100%, ונשאל את עצמנו – מי יכול להפגע מקיום של מערכת כזו? איך אפשר לנצל לרעה את קיומה של מערכת כזו? אפשר לחשוב בקלות על הרבה דוגמאות (למשל אוניברסיטאות יפסלו מועמדים מתאימים שיכלו להצליח מאוד אקדמית רק בגלל שיש להם *IQ* לא גבוה מספיק – כיוון ש-*IQ* הוא רק *biased proxy* ל-*label* האמיתי אותו האוניברסיטאות באמת רוצות לחזות, שהוא הצלחה אקדמית). כעת במקרה בו הדיוק של המסווג אינו מושלם – למשל משיג דיוק של 90%. האם יש תתי קבוצות באוכלוסיה (הדאטא שעליו רץ המודל) שעבורן הדיוק אפילו יותר נמוך

(למשל כתוצאה מתת ייצוג או ייצוג מוטה בדאטא האימון)? האם דיוק (*accuracy*) היא מטריקה מתאימה לבחינת הצלחת המודל? מה המחיר של סיווג שגוי (*misclassification*) – מי הקבוצות שיפגעו כתוצאה מתיוג שגוי? שאלה קשה נוספת היא – מי נושא/ת באחריות להטיה של המודל? (החוקר/ת שפיתח/ה אותה / מי שפיקח או ליווה אותם / מי שביצע ביקורת עמיתים למאמר / האוניברסיטה / מי שמשתמש במערכת / החברה כולה).

דוגמא נוספת – *AI Gaydar*: נביט במחקר (אמיתי) בו פיתחו מערכת *AI* שמסווגת את הנטיה המינית של בני אדם באמצעות מאפיינים של תווי הפנים שלהם מתוך תמונות. התמונות לקוחות מתוך מאגר תמונות של אתר דייטינג, כאשר יש בערך 35 אלף תמונות המייצגות בערך 15 אלף אנשים שכולם לבנים, כאשר האוכלוסיות של גברים/נשים ושל סטרייטים/להטב"קים מיוצגות באופן שווה בדאטא. השיטה בה השתמשו לפיתוח המודל תהיה מודל למידה עמוקה שמחלץ את מאפייני תווי הפנים והטיפוח של המצולמים בתמונות, שייצר פיצ'רים שיוכנסו לקלסיפייר שיתייג את הנטיה המינית. המודל משיג דיוק (*accuracy*) של 81% עבור גברים ושל 71% עבור נשים. מה הבעיה עם מחקר שכזה? מעבר לבעייתיות המוסרית בנושא המחקר עצמו, גם כאן כדאי לשאול את עצמנו מי יכול להרוויח מקיומו של מודל שכזה ומי יכול לנצל אותו לרעה (למשל ישנן מדינות שבהן הומוסקסואליות היא עברה שמוציאים עליה להורג, אפשר להשתמש בנתונים האלו כדי להפלות לרעה בקבלה לעבודה ושלל תרחישים מדאיגים נוספים). הבעיה היא "*dual framing*" של מודלי פרדיקציה – לרובם יכולים להיות שימושים אתיים וחיוביים ובמקביל אפשרות לניצול לרעה שלהם למטרות מזיקות (כמו סכין, שיכול לשמש להכנת מזון או לאלימות). עולה שוב השאלה – האם האחריות לעצם האפשרות לשימוש לרעה במודל נמצאת על המפתחים שלו או על המשתמשים בו? עולות גם שאלות הנוגעות לדאטא בו השתמשו – אמנם זה חוקי, אך האם זה אתי להשתמש במידע אישי (שלא הובעה הסכמה מפורשת לשימוש בו) אם הוא מפורסם בפלטפורמה פומבית? האם לגיטימי להשתמש בדאטא שאינו מייצג את האוכלוסיה הכללית נאמנה או שעשוי להיות מוטה מהותית (למשל דאטא שמכיל רק אנשים לבנים / קבוצת גיל מסוימת / לבוש שתואם אופנה מתקופה מסוימת וכו')? החשש במקרה כזה היא שאמנם החוקרים יכולים לציין את מגבלות המודל וההטיות שלו, אבל המשתמשים במודל עשוים להתעלם מהן, מה שיכול ליצור הרבה נזק (למשל, אם ישתמשו במודל הנ"ל על אנשים שאינם לבנים, התוצאות יהיו חסרות משמעות כיוון שהמודל לא אומן על דאטא רלוונטי בכלל, אבל המשתמשים בו עשויים להתייחס לתוצאות המודל כאמיתיות).

מודלי למידה עמוקה הם סוג של "קופסא שחורה" ולרוב אינם אינטרפטביליים. האם מודלים שאינם אינטרפטביליים אינם מתאימים לביצוע פרדיקציות הנוגעות לעניינים רגישים, או בניסויים שהתנאים בהם דורשים הבנה רחבה יותר לגבי העולם? איך נוכל לנתח טעויות והטיות מוגברות שהמודלים מכילים?

שאלות לבחינת מידת האתיות של מחקר בתחום ה-AI

חשוב לזכור שהעלות של תיוג שגויה (מיסקלסיפיקציה) אינה שקולה לרווח מתיוג נכון. סוגים שונים של טעויות עשויים להיות בעלי מחיר שונה (למשל תיוג אישה שחורה בתור גורילה עשוי להחשב חמור יותר מאשר לתייג סלע בתור גורילה). המרצה ממליצה לשאול את עצמנו את השאלות הבאות –

- מהם השיקולים האתיים בשאלת המחקר עצמה?
- מה ההשפעה הצפויה של המודל וכיצד אפשר לנצל אותו לרעה? מי יכול להרוויח מהמודל? מי יכול להפגע ממנו? האם שיתוף הדאטא והמודל עשוי לגרום להשפעה על חיים של אנשים?
- למי יש בעלות על הדאטא שאנחנו משתמשים בו לאימון המודל? האם הדאטא מפורסם *published* או רק פומבי *publicized*? האם יש הסכמה של הישויות עליהן נאסף הדאטא לשימוש בו או הנחה לא-מפורשת כלשהי לגבי איך יתבצע השימוש בדאטא?
- מה ההטיות שקיימות בדאטא – ארטיפקטים שקיימים בדאטא, התפלגויות שספציפיות לאוכלוסיות מסוימות, ייצוגיות בדאטא.
- מה ההטיות שקיימות במודל – איך לשלוט ב-*confounding variables* ובמקרי קצה? האם המודל מאפטם את ה-*objective* ה"נכון"? האם ואיך המודל מעצים הטיות שקיימות בדאטא עליו הוא מאומן?
- מה המטריקה שבאמצעותה אנחנו בוחנים את המודל – חשוב להסתכל גם על אחוזי *FP* ו-*FN*, על העלות של מיסקלסיפיקציה, ועל ה-*fault tolerance*.

נושאים במפגש בין אתיקה ו-NLP

המרצה מסמנת מספר נושאים שהיא סבורה שמעניינים במפגש בין *NLP* לבין אתיקה:

- הטיה אלגוריתמית – הטיות חברתיות בדאטא ובמודלי *NLP*.
- *Incivility* – תגובות נאצה, תגובת רעילות, חוסר נימוס ומיקרואגרסיות.
- הפרות פרטיות ושימוש ב-profiling באמצעות מודלי שפה.
- מיסאינפורמציה – פייק ניוז, מניפולציה של מידע, מניפולציה של דעות.
- חוסר איזון טכנולוגי – שימוש לא הגון במודלי NLP, ביצועים לקויים עבור דיאלקטים של קבוצות מיעוט או שפות ממדינות מתפתחות.

מקורות מידע נוספים בתחום:

- סמינר של סטנפורד בנושא נושאים אתיים וחברתיים ב-*NLP*
- הרצאות בנושא אתיקה חישובית
- ריכוז מאמרים של *ACL* בנושא אתיקה ב-*NLP*

הטיה אלגוריתמית במודלי NLP

System 1	System 2
automatic	effortful
fast	slow
parallel	serial
automatic	controlled
effortless	effort-filled
associative	rule-governed
slow-learning	flexible

כדי להבין את המהות של הטיות, חשוב לדבר על איך עובדות הטיות קוגניטיביות אצל בני אדם. בקווים כלליים, ישנה תאוריה שמציעה שהמערכת הקוגניטיבית האנושית מורכבת משתי תתי מערכות – מערכת 1 (פשוטה ואוטומטית) ומערכת 2 (לוגית ומאמצת). המוחות שלנו התפתחו אבולוציונית לאחסן מידע כך שיתאים לשליפה ושיפוט מהירים – הרוב המוחלט (יותר מ-95 אחוז) מהסיגנל הסנסורי שהמוח מקבל עובר קודם דרך מערכת 1. כלומר, בני אדם משתמשים ב"טייס אוטומטי" ברוב המוחלט של הזמן, מה שהופך את קיומן של הטיות לבלתי נמנע. הטיות קיימות כדי לאפשר למוח לקטלג את העולם כך שיוכל להשתמש ב"טייס האוטומטי" לביצוע תהליכי עיבוד פשוטים, וכדי לאחסן את המידע שנלמד באמצעות ייצוגים מנטליים (המכונים סכמות). בנוסף, זה מאפשר למוח להפעיל (באופן בלתי מודע) את המידע המאוחסן לגבי קטגוריה מסוימת כאשר נתקלים בפרט ששייך לאותה קטגוריה. כשאנחנו מפתחים מודלי *AI*, אנחנו מנסים לחקות את מערכת 1 – מערכת שמקבלת החלטות פשוטות על בסיס כל הדאטא שהיא נחשפה אליה תוך ביצוע הכללה, ועשויה לבצע *overfit* לקורלציות שאינן סיבתיות.

הטיה קוגניטיבית היא דפוס סכמטי שגורם לסטיה מתפיסה או שיפוט רציונליים. סוגים של הטיות נפוצות:

- *Confirmation bias*: הטיה בה מקצים יותר קשב למידע שמאשש אמונות קיימות שכבר יש לנו, ומתעלמים ממידע שסותר אמונות קיימות.
- *Ingroup favoritism*: הטיה בה אנשים מעדיפים אנשים שמשתייכים לאותם קבוצות כמוהם בהשוואה לאלו שלא.
- *Group attribution error*: הטיה בה אנשים מכלילים על קבוצה שלמה בהתבססות על ייצוגים בלבד.
- *Halo effect*: כשההתרשמות הכללית מבן אדם משפיעה על האופן שבו מעריכים תכונות ספציפיות שלו.
- *Just*-world hypothesis: הטיה בה אנשים מגנים על רצון לקיומו של עולם צודק והוגן באמצעות האשמת הקורבנות ("לי זה לא יקרה כי אני לא עושה כך וכך").

הטיות קוגניטיביות סטראוטיפיות מתבטאות בהרבה דרכים. אחת מהדרכים האלו היא מיקרואגרסיות *microagressions* – הערות או פעולות שבאופן עדין ולא מודע / לא מכוון מבטאות דעות קדומות כלפי חבר/ה בקבוצה מוחלשת. הסנטימנט שמבוטא עשוי להיות שלילי, ניטרלי או חיובי. למשל – "בנות לא טובות במתמטיקה", "אנשים כמוך אוהבים טורטיות, לא?", "את יותר מידי יפה כדי להיות לסבית!". אם ננסה לבנות מודל שיחזה האם תגובה היא מיקרואגרסיה או לא, הוא עשוי להתקשות בגלל שהקביעה הזו מאוד תלויה בהקשר – למשל התגובה "איזו יפה את!" יכולה להחשב חיובית ונעימה על תמונה שמישהי העלתה, אבל להחשב כמיקרואגרסיה אם זו תגובה על פוסט בו מישהי מציגה מאמר שהיא פרסמה.

הטיות בדאטא – נובעות מהטיות בשפה עצמה (סטראוטיפים, דעות קדומות וכו' שמבוטאות ע"י בני אדם, וגם *human reporting bias* – התדירות שבה בני אדם מדברים על נושאים מסוימים לא מייצגת את העולם האמיתי) וגם מהטיות בדאטאסטים (כתוצאה מהטיות בדגימה, מהטיות קוגניטיביות של מי שתייג את הדאטא).

בדרך לגישה פרואקטיבית (במקום ראקטיבית) להטיות במערכות *AI* –

- דאטא: מודרציה אוטומטית, חשיפת הטיות חברתיות ורעילות מוסתרת בדאטא האימון.
- ייצוגים של ידע סוציו-תרבותי: ללמוד איך לייצג ולנתח איך ידע סוציו-תרבותי מופיע בשפה.
- מידול: גישות מידול חדשות שמשלבות הקשר סוציו-תרבותי שמאומנים כדי להסיר הטיות חברתיות באופן מפורש.
- אוולוואציה ואנליזה: פיתוח מודלים אינטרפטבילים, או גישות לביצוע אינטרפטציה למודלים קיימים, בנוסף לפיתוח גישות חדשות לאוולוואציה ואפיון של התנהגויות של מודלים.

ניתוח והסברת מודלים

מודלי למידה עמוקה הם סוג של קופסא שחורה שהיא לא מאוד אינטרפטבילית, שמקבלת קלט ומוציאה פלט בדיוק כזה או אחר. כחוקרים או כמדעני נתונים, אנחנו נוטים לסכם את המודלים שלנו באמצעות מטריקה בודדת (או כמה מטריקות בודדות), בדגש על כך שמדובר במטריקה מספרית. אבל המטריקות האלו לא מאפשרות לנו לבדוק מה המודלים האלו למדו, למה הם מצליחים או למה הם נכשלים. בד"כ כשרוצים לשפר מודל, לוקחים ארכיטקטורה או שיטה שיודעים שעובדת, ואז מחפשים מה צריך לשנות כדי לשפר אותה. חשוב להבין כמה רחוק אנחנו יכולים להגיע באמצעות תוספות ושיפורים לשיטות קיימות כדי להבין מתי נדרשים שיפורים משמעותיים חדשים. אנחנו רוצים לא רק לפתח את "המודל הבא", אלא את המודלים לעשורים הקרובים. אנחנו רוצים להיות מסוגלים לענות על שאלות כמו:

- מה אפשר ללמוד באמצעות pretraining של מודל שפה? מה לא ניתן ללמוד כך?
- איזו ארכיטקטורה תחליף את הטרנספורמר?
- במה למידה עמוקה מתקשה?
- מה מודלי שפה מספרים לנו על שפה אנושית?
- איך מודלי שפה משפיעים על אנשים ומשנים את יחסי הכוחות ביניהם?

יש דרכים רבות לנתח מודלים, אף אחת מהן לא מושלמת או מספקת בהירות מושלמת. כעדי להתחיל, צריך לבחור מהי רמת האבסטרקציה שבה אנחנו רוצים לדבר על מודל מסוים:

- המודל בתור התפלגות ופונקצית החלטה, כלומר מסתכלים רק על הקלט והפלט של המודל.
- המודל בתור רצף של ייצוגים וקטוריים בעומק ובזמן, כלומר מסתכלים על ההבדלים בין שכבות שונות של המודל.
- המודל בתור אוסף של פרמטרים / משקולות ומכניזמים ספציפיים כמו attention, dropout וכו'.

אוולויאציה ואנליזה של מודלים

כשאנחנו מסתכלים על ההתנהגות של מודל מסוים, אנחנו לא מודאגים (עדיין) **מהמכניזמים** שהמודל משתמש בהם. אנחנו רוצים לשאול **איך המודל מתנהג בסיטואציות שמעניינות אותנו.** למשל, אם אימנו מודל על כמות מסוימת של דגימות מסוימות, ומעניין אותנו לדעת איך הוא יתנהג על דגימות חדשות מאותה התפלגות, או על דאטאסט מוכר שמשמש לאוולויאציה של המשימה לשמה אימנו אותו. מה שנהוג זה להשוות בין מודלים שמבצעים את אותה המשימה באמצעות מטריקות כמו accuracy או f1 על דאטאסטים מייצגים, אבל המדד הזה אינו מספק. נרצה גם לבחון את המודל בסיטואציות ששונות מהדאטא שהוא אומן עליו. למשל, במשימת NLI (שבה מקבלים רישא וסיפא וצריך לקבוע אם יש ביניהם יחס של גרירה לוגית) אפשר לבצע אוולויאציה באמצעות **diagnostic test set** (דאטאסט שנוצר בקפידה כדי לבחון יכולת מסוימת של המודל) המכונה **HANS (heuristic analysis for NLI systems)** שבוחן יוריסטיקות תחביריות ב-NLI (בתמונה משמאל – דוגמאות למקרים בהם יוריסטיקות לא עובדות).

Heuristic	Definition	Example
Lexical overlap	Assume that a premise entails all hypotheses constructed from words in the premise	**The doctor** was **paid** by **the actor**. → The doctor paid the actor. (WRONG)
Subsequence	Assume that a premise entails all of its contiguous subsequences.	The doctor near **the actor danced**. → The actor danced. (WRONG)
Constituent	Assume that a premise entails all complete subtrees in its parse tree.	**If the artist slept**, the actor ran. → The artist slept. (WRONG)

במאמר מ-2019 לקחו 4 מודלי NLI חזקים ובחנו אותם in-domain (כלומר על דגימות מה-test-set המקורי שלהם), וראו שהם קיבלו בין 60-80% דיוק (accuracy). עם זאת, כשבחנו אותם על HANS על דוגמאות שבהם היוריסטיקות התחביריות עובדות, ראו שארבעתם הצליחו לקבל accuracy של הרבה מעל 90% (יותר טוב מתוצאות ה-in-domain), אך במקרים בהם היוריסטיקות התחביריות לא עובדות המודלים כולם השיגו דיוק של פחות מ-20%. כלומר, האוולויאציה על הדוגמאות המעניינות מתוך HANS אפשרה להבחין בכך שהצלחת המודל לא היתה כל כך טובה כמו שמשתמע מהבדיקה ה-in-domain.

איך בני אדם מתנהגים בתור נבדקים של משימות שפתיות?

איך נוכל להבין התנהגות שפתית בבני אדם? שיטה נפוצה היא שימוש ב"זוגות מינימליים" minimal pairs, כלומר – מה נשמע "בסדר" לדובר שפה, אבל מפסיק להשמע תקין אם עושים בו שינוי קטן? למשל –

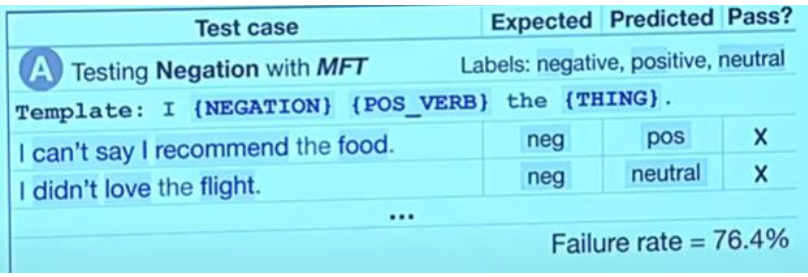
The chef who made the pizzas is here.	תקין acceptable
The chef who made the pizzas **are** here.	לא תקין unacceptable

אמנם בני אדם ידעו להבדיל בין משפט תקין ללא תקין, אבל מה שמודלי שפה ידעו לעשות (בתקווה, אם הם עובדים טוב) זה לתת הסתברות גבוהה יותר מלשפט התקין בהשוואה למשפט הלא תקין. כלומר, זו עוד דרך לבדוק שמודלי שפה עושים את מה שאנחנו מצפים מהם. בדומה ל-HANS, אנחנו יכולים לייצר test set יעודי שמכיל דוגמאות כאלו שנבחרו בקפידה. ב-2018 פורסם מאמר בו אימנו

מודל שפה מבוסס LSTM על סמפל מתוך ויקיפדיה. הם ביצעו אוולויואציה רק למשפטים עם מאפיינים תחביריים מסוימים (שבכל אחד מהם יש מספר ספציפי של agreement factors), ובדקו על כמה מהם המודל הצליח לחזות את המספר הנכון עבור הפועל. הם ראו שעבור המספר 0 המודלים הצליחו בקלות, וככל שהמספר עולה המודלים מתקשים יותר (אבל עדיין מצליחים) – חשוב להגיד שזו תופעה שקורית גם אצל בני אדם.

UNIT TESTING ובחינת הידע של מודלים

הרעיון של ליצור טסט-סטים ספציפיים במטרה לבחון ביצועים של מודל על משימה ספציפית, מזכיר את הרעיון של unit-testing מעולם פיתוח התוכנה. מאמר מ-2020 מציע לייצר minimum-functionality tests (טסטים קטנים שבוחנים על התנהגות מאוד ספציפית של המודל), למשל הטסט הבא עבור משימת sentiment analysis, שבוחן את התיוגים שהמודל נותן למשפטי שלילה מהצורה המאוד ספצפית המצוינת בתמונה:



מחקר מ-2020 שבו נתנו למהנדסים שעובדים על מוצר שמבצע sentiment analysis להשתמש בטסטים שכאלו, הראה שהמהנדסים מצאו באגים חמורים באמצעות השימוש בטסטים.

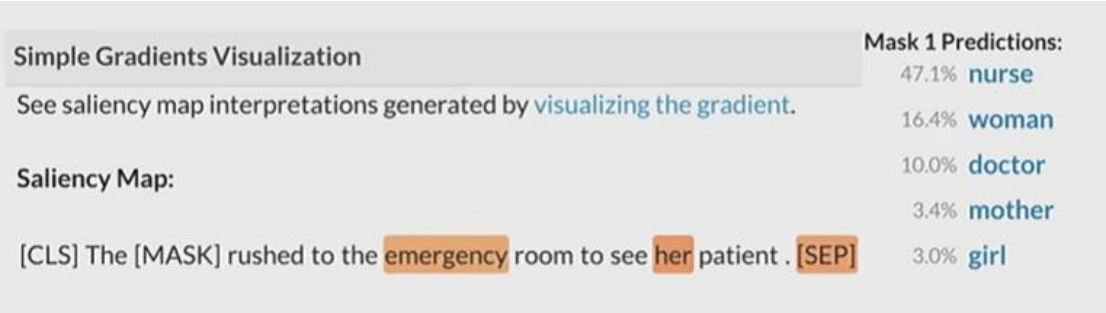
עבור טווח נרחב של משימות, דיוק (accuracy) גבוה של המודל על in-domain test set לא מעיד שהמודל יצליח גם על out-of-domain test sets "reasonable". מולדים נוטים ללמוד את הדאטאסט (למשל MNLI) ולא את המשימה (למשל איך בני אדם מבצעים הסקה לוגית בשפה). יצור חבילת טסטים איכותית שהיא out-of-domain יכולה לאפשר לבחון את היכולות האמיתיות של המודל.

השפעת הקלט וזכרון לטווח ארוך

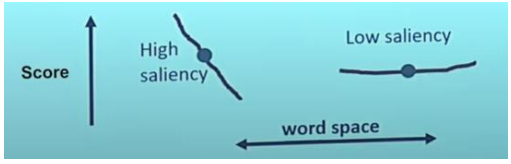
המוטיבציה לשימוש ברשתות LSTM בתור מודלי שפה על פני RNN פשוטות היא היכולת של רשתות LSTM להשתמש בהקשר בין טוקנים שרחוקים זה מזה במשפט כדי לבצע את הפרדיקציה למילה הבאה. אבל כמה "ארוך" הוא הזכרון לטווח ארוך של רשתות LSTM? מחקר מ-2018 מראה שאם מערבבים או מורידים את כל ההקשרים שרחוקים יותר מ-k מילים (עבור כל מיני ערכי k שונים), אפשר לבחון החל מאיזה ערך של k הביצועים של המודל מתחילים להתדרדר (ובכך לבחון את אורך הזכרון שלו). המאמר הראה כצפוי שעבור ערכי k קטנים זה מאוד השפיע על הביצועים, אבל עבור בערך k=50, היה נראה שהמודל כבר לא מושפע מזה בכלל, כלומר המידע שמופיע במרחק של יותר מ-50 טוקנים אינו נכלל בזכרון לטווח ארוך של המודל.

SALIENCY MAPS

המחקר הזה התייחס לאוולויואציה באופן כללי של הזכרון של מודל ספציפי, אבל היינו רוצים להיות מסוגלים להעריך את הביצועים של מודל על דגימה ספציפית (למשל – להיות מסוגלים להסביר למה הוא נתן תשובה שגויה עבור דגימה כלשהי). היינו רוצים לדעת אילו חלקים בקלט גרמו ליצירת הפרדיקציה שנצפתה. כדי לעשות זאת, משתמשים ב-**saliency maps** – ציון עבור כל מילה בקלט שמציין את החשיבות שלו בפרדיקציה שהמודל נתן (בכתום בתמונה, כאשר טוקן SEP יופיע כמעט בכל מיפוי אז לא מתייחסים אליו).



איך יוצרים saliency maps? יש הרבה דרכים לקודד את האינטואיציה של "מידת החשיבות" של המילים בקלט. שיטה פשוטה לעשות זאת נקראת simple gradient method, ובה לכל רצף מילים המסומן $x_1, \dots, x_n$ ולתוצאה שהמודל נתן עבור תגית מסוימת (output label) המסומנת $s_c(x_1, \dots, x_n)$, לוקחים את הנורמה של הגרדיאנט של התוצאה ביחס לכל אחת מהמילים –



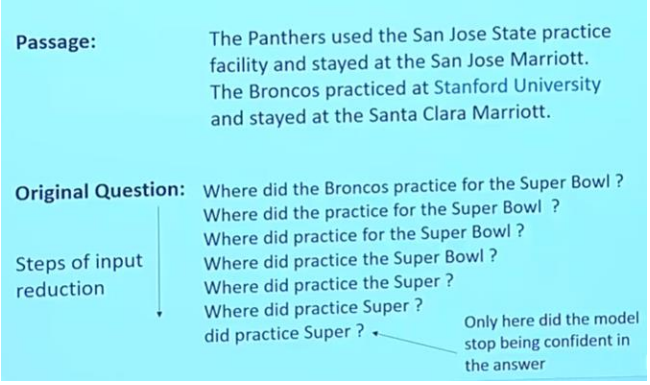
$$\text{salience}\ (x_i) = \|\nabla_{x_i} s_c(x_1, \dots, x_n)\|$$

הרעיון הוא שגרדיאנט מנורמל **גבוה** אומר ששינוי (לוקלי) של המילה ישפיע מאוד על התוצאה. כלומר אם יש saliency גבוה, שינוי קטן ישפיע מאוד על התוצאה, אבל אם ה-saliency נמוך, שינוי של המילה לא ישפיע על התוצאה כמעט בכלל. השיטה הזו אינה מושלמת (בעיה משמעותית שלה היא שלפעמים הערך שמתקבל הוא מאוד לוקלי), ויש הרבה שיטות אחרות שהוצעו לחישוב saliency.

רדוקציה של הקלט

עוד שאלה שהיינו רוצים לשאול את עצמנו בנוגע למודלים היא מהו החלק הקטן ביותר מתוך הקלט שאפשר לשמור ועדיין לקבל את אותה התוצאה. למשל, במאמר מ-2018 בחנו מודל שהצליח לענות נכונה על הדוגמה הבאה מתוך SQUAD -

Passage	Question
In 1899, John Jacob Astor IV invested 100,000$ for Tesla to further develop and produce a new lighting system. Instead, Tesla used the money to fund his Colorado Springs experiment.	What did Tesla spend Astor's money on?

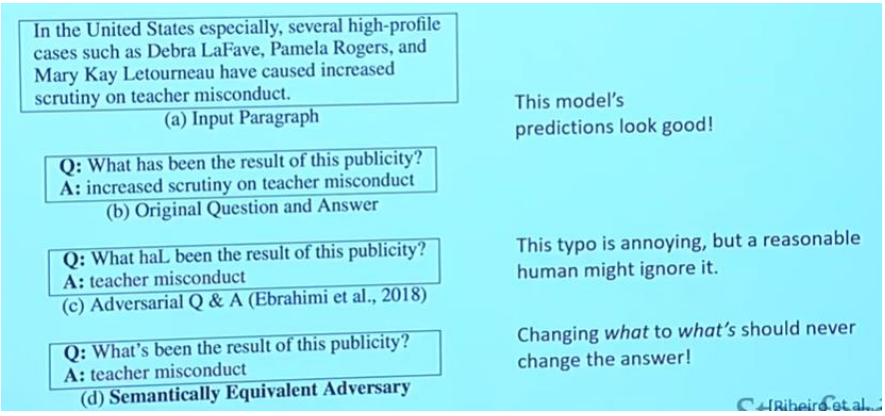


עם זאת, במאמר הראו שאם מצמצמים את השאלה כולה למילה "did" לבדה, המודל עדיין מצליח לענות על השאלה בצורה הנכונה (כלומר מחזיר בדיוק את אותה התשובה, ואפילו מחזיר confidence גבוה משמעותית עבורה).

הרעיון בשיטה שהוצגה במאמר מזכיר מעט את שיטת ה-input saliency. בשיטה זו מסירים באופן איטרטיבי את המילים הכי פחות חשובות מתוך הקלט, ובודקים האם המודל מצליח לשמר את הביצועים שלו על הקלט המצונזר.

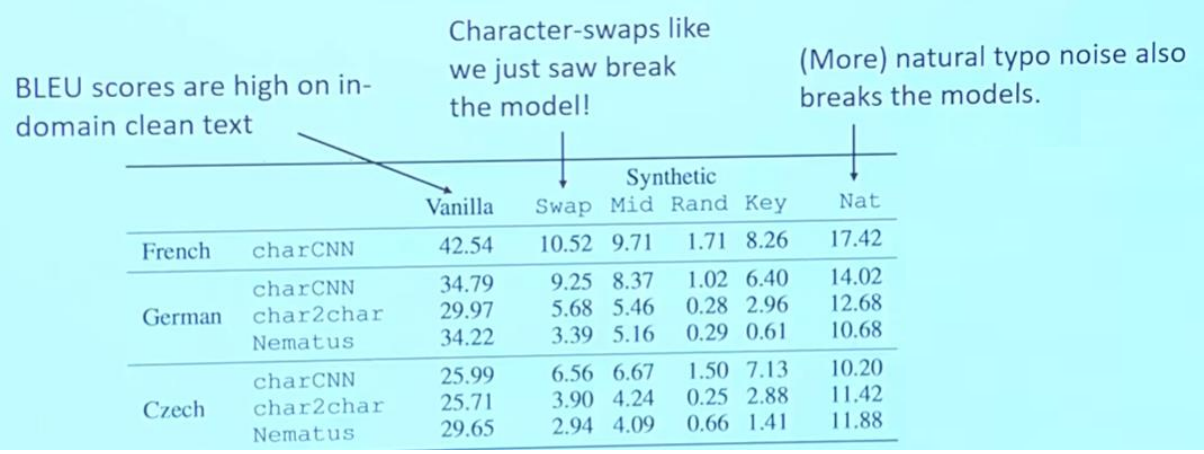
ניתוח מודלים באמצעות "שבירה" שלהם

בגישה זו, במקום לשנות את השאלה עד שהיא כבר לא הגיונית כדי לראות אם אנחנו עדיין מצליחים לגרום למודל להפיק את אותה תוצאה, מה שעושים זה לפרק את המודל ע"י לבצע שינויים לכאורה תמימים (כלומר לא מזיקים) בקלט? למשל, לשנות או להוסיף לפסקה המקורית (באופן תחבירי תקין) פרטים עובדתיים שאינם קשורים לשאלה המבוקשת, ולראות אם המודל עדיין מצליח לענות את התשובה הנכונה. למשל בדוגמא ניתן לראות שהחלפה של what ב-what's "שוברת" את המודל.



האם מודלים רובוסטיים לרעש בקלט שלהם?

מחקרים פסיכולוגיים על בני אדם מראים שבני אדם מסוגלים לעבד באופן תקין לחלוטין שפה גם בהרבה מקרים בהם השפה כוללת טעויות, למשל סדר האותיות במילים אינו תקין. היינו רוצים לבחון האם מודלי שפה מצליחים להראות יכולות דומות ליכולות אנושיות בתחום זה. רעש ממגוון סוגים הוא בלתי נמנע במודלי NLP, והיינו רוצים לדעת האם מודלים שאומנו על דאטא יחסית נקי יוכלו להתמודד עם קלט שמכיל ערבוב אותיות במילים (שמזכיר שגיאות הקלדה). הדוגמא הבאה מדגימה מודל שבחנו אותו על סוגים שונים של טקסטים עם החלפות אותיות – ניתן לראות שהחלפת אותיות (בין אם שנראית מלאכותית ובין אם "טבעית" שמזכירה את האופן שבו בני אדם טועים בהקלדה) "שוברת" את המודלים. בדיקות שכאלו הן משמעותיות כדי לדעת מה המגבלות של המודלים ולשקול לבצע התאמות לדאטא שהם מאומנים ונבחנים עליו כדי להעריך ולשפר את הרובוסטיות שלהם.

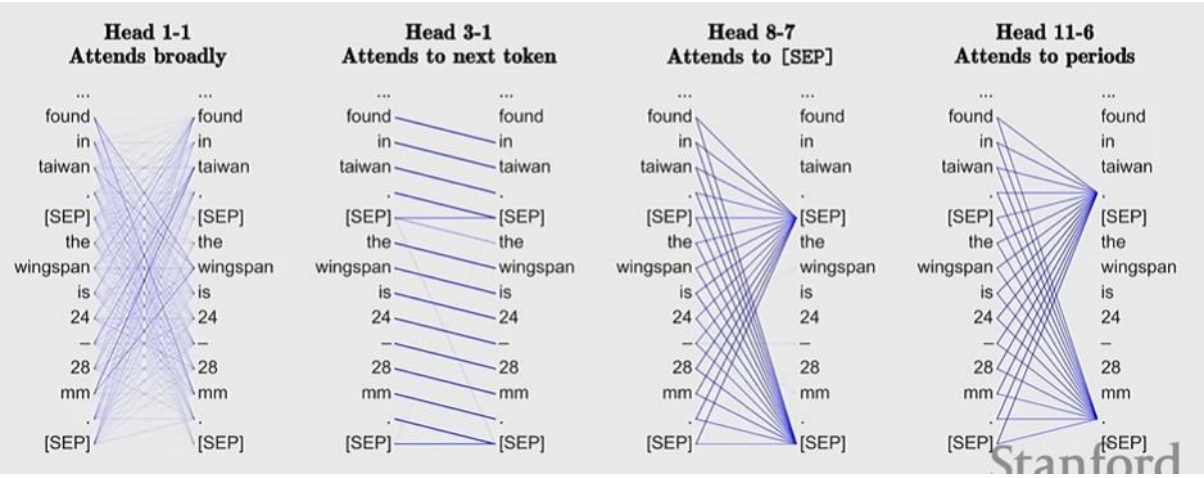


		Vanilla	Synthetic Swap	Mid	Rand	Key	Nat
French	charCNN	42.54	10.52	9.71	1.71	8.26	17.42
German	charCNN	34.79	9.25	8.37	1.02	6.40	14.02
	char2char	29.97	5.68	5.46	0.28	2.96	12.68
	Nematus	34.22	3.39	5.16	0.29	0.61	10.68
Czech	charCNN	25.99	6.56	6.67	1.50	7.13	10.20
	char2char	25.71	3.90	4.24	0.25	2.88	11.42
	Nematus	29.65	2.94	4.09	0.66	1.41	11.88

ניתוח הייצוגים (REPRESENTATIONS) של רשתות נוירונים

עד כה דיברנו על בדיקות ברמת אבסטרקציה מאוד גבוהה – כלומר הסתכלנו על הקלטים והפלטים של המודלים. כעת נסתכל ברמת אבסטרקציה נמוכה יותר, ונבחן את הייצוגים הוקטוריים שנוצרים עבור הקלט בתוך רשת הנוירונים של המודל.

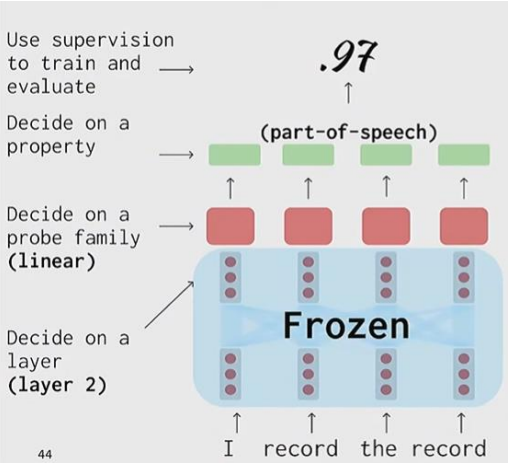
הרעיון הוא שחלק מהרכיבים במודל הם ניתנים לצפיה וניתוח, והמידע שהם מכילים עשוי לספק לנו תובנות משמעותיות (השאלה לגבי עד כמה הם באמת אינטרפטביליים היא מוקד לדיון ואין הסכמה לגביה). לדוגמא, אנחנו יכולים לנסות לאפיין כל attention-head של BERT. למשל, במחקר מ-2018 שעשה ויזואליזציה ל-attention heads בכל מיני שכבות של BERT . עבור הראש הראשון בשכבה הראשונה (משמאל בתמונה למטה), נראה שהוא מדגים התנהגות דומה ומאוד "אגרגטיבית" באופי שלה עבור רוב משפטי הקלט. בתמונה למטה ניתן לראות ויזואליזציות נוספות מאותו מחקר, שמראות שישנם attention heads שלכאורה מבצעים פעולות פשוטות יחסית (למשל מקשיבים לטוקן הבא, מקשיבים לטוקן ה-SEP, מקשיבים לנקודות ועוד).



אותו מחקר הראה שיש גם ראשים שמקשיבים לדברים כמו Noun modifiers (מילים שמתארות שמות עצם, כמו the, new, huge וכו') או מבצעים coreference resolution. יש מחקרים ישנים יותר שמראים ויזואליזציות מעניינות ויחסית אינטרפטביליות גם על רשתות LSTM.

שימוש ב-PROBING לניתוח באופן SUPERVISED של רשתות נוירונים

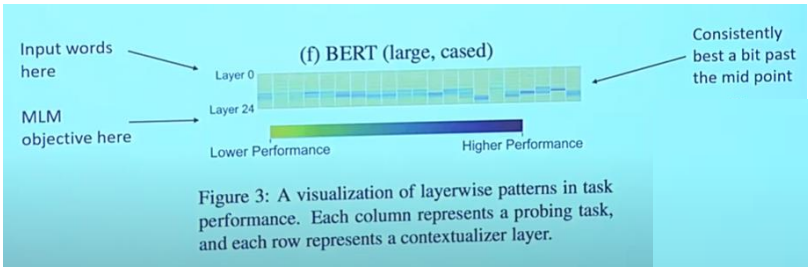
הרעיון של probing הוא שנקודת המוצא היא שיש לנו קלט טקסטואלי שמוזן למודל pretrained (שמניחים שהוא מספק ייצוגים וקטוריים כלליים למילים בשפה) ומשיג תוצאות SOTA, ואז אפשר לבחון את המודל באמצעות בחינה של מה הייצוגים הוקטוריים שהוא מייצר מקודדים לגבי השפה אותה הוא למד. כלומר, אנחנו רוצים להבין מה הייצוגים הוקטוריים שהמודל מייצר מקודדים על תכונות לשוניות שהוא הכליל באמצעות שימוש בדאטא מתויג עבור בעיה לשונית כלשהי.



איך מיישמים probing? ראשית, מחליטים איזו שכבה במודל אנחנו רוצים לנתח, ומקפיאים את כל המשקולות במודל (כלומר לא מבצעים finetuning). מחליטים על probe family, שזו שכבה נוספת שמאפשרת לקודד (מתוך הייצוגים הוקטוריים שהמודל יצר בשכבה אותה אנחנו בוחנים) את התכונה הלשונית אותה אנחנו רוצים לבחון. משתמשים בדאטאסט מתויג הרלוונטי לתכונה הלשונית אותה אנחנו רוצים לבחון כדי לאמן את שכבת ה-probe, כשהרעיון הוא שאם אנחנו מצליחים להשיג ביצועים טובים (ובפרט אם אנחנו מצליחים להשיג ביצועים טובים בהשוואה לבייסליין משכבה נמוכה יותר במודל או אפילו מהשכבה הראשונה), כנראה שהשכבה אותה בחנו במודל המקורי מקודדת מידע משמעותי עבור הבעיה אותה בחנו.

פורמלית, בהנתן תכונה לשונית כלשהי המסומנת y (למשל POS), ובהנתן גישה לייצוגי המילים שהמודל מייצג בכל אחת מהשכבות $h_1, \dots, h_T$ (כאשר האינדקס 1-T מייצג את אינדקס המילה במשפט הקלט, וכל אחד מהייצוגים הוא ממימד d). ניקח משפחת פונקציות F (למשל קבוצת מודלים לינאריים או רשתות FF בעלות שכבה אחת עם פרמטרים מקובעים). מקפיאים את הפרמטרים במודל (כדי שלא יתבצע finetuning), ואז מאמנים את שכבר ה-probe $f \in F$ $\hat{y} \sim f(h_i)$. הרעיון הוא שזה שאפשר לחזות את y מתוך h_i זה מדד לרמת ה"נגישות" של הפיצ'ר הלשוני אותו אנחנו בוחנים (למשל *POS*) בתוך הייצוג הוקטורי בשכבה אותה בחנו. זה עוזר לקבל הבנה כללית של איך המודל מעבד את הקלטים שלו, ועשוי גם לעזור לחפש מכניזמים סיבתיים במודל.

למשל ב-*BERT*, מחקרים הראו שעבור מגוון די רחב של תכונות לשוניות, השכבות האמצעיות של המודל נותנות את התוצאות הטובות ביותר ב-*probing*.

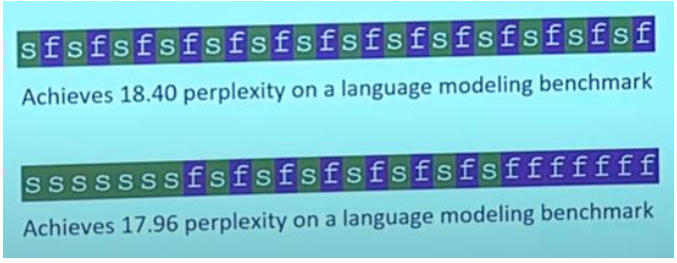


בנוסף, מחקרים מראים כי ככל שמשימה לשונית היא יותר מופשטת במהות שלה, השכבה שבה מתקבלות התוצאות הטובות ביותר ב-*probing* עבורה תהיה עמוקה יותר.

שיטות *probing* מראות את התכונות שנגישות למשפחת הפוקנציות שבחרנו לבחור באמצעותן (ה-*probe family*), אבל הן לא מאפשרות לראות את התכונות שנגישות למודל עצמו (אותו רצינו לבחון). אותה טענה נכונה גם לגבי מחקרים שבוחנים קורלציה (לדוגמא מחקרים שממפים *attention*). מחקרים עדכניים יותר (2020 והלאה) מנסים להשתמש בשיטות קוזאליות (*causal inference*) כדי לזהות סיבתיות בייצוגי הבעייות הלשוניות בתוך מודלים.

ביצוע RECASTING למודלים ומחקרי ABLATION

בתרחיש בו יש לנו מודל שפה שעובד בסדר, ואנחנו רוצים לראות אם אפשר לשפר אותו בקלות בדרכים פשוטות, אפשר לראות אם אפשר להסיר ממנו רכיבים מורכבים יחסית ועדיין לקבל תוצאות טובות באות המידה. אפשר להשתמש ברעיון הזה כדי לנתח את המודל, למשל – האם המודל טוב יותר כאשר הוא רדוד או עמוק? האם חלק מסוים בארכיטקטורה תורם למודל או מקשה עליו?

דוגמא למחקר *ablation* הוא מחקר מ-2019 שבו בחנו טרנספורמרים עם *multi-headed attention* על משימת תרגום מכונה ו-*NLI*. לאחר האימון, מצאו שהרבה מאוד (עשרות אחוזים) מראשי ה-*attention* יכולים להיות **מוסרים לחלוטין** (בשלב ה-*test*) ללא פגיעה בדיוק של המודל.

שאלה מעניינת נוספת היא מהו הסדר הנכון לשכבות בטרנספורמר. ראינו שטרנספורמרים הם מודלים שבנויים מסדרה של שכבות (*self-attention* ואז *FF* וחוזר חלילה, נניח בצד לצורך הדיון את הנרמול של השכבות ואת ה-*residual connections*). במאמר מ-2019 ניסו לבחון את החשיבות לסדר של השכבות בטרנספומר כמודל שפה, ולבחון האם יש דרך טובה יותר לסדר את שכבות ה-*attention* וה-*FF*. הם מצאו למשל שהסידור הבא (שבו המודל מתחיל מערמה של שכבות *self-attention* ללא שכבות *FF*, ונגמר בהרבה שכבות *FF*) מצליח להשיג תוצאות טובות יותר מהמקור (כלומר *perplexity* נמוך יותר).