



רשתות זרימה קדימה למשועממים בלבד

מאת מעיין אושרי

הקדמה

בגיליון 155, ספיר פדרובסקי פרסמה את המאמר [רשתות נוירונים למשועממים בלבד](#), בו היא סקרה בצורה תיאורטית ובחלקה מעשית את הנושא של רשתות נוירונים מסוג Feedforward Neural - FNN Network. מפורטים במאמר הסברים אודות האלגוריתמים והמתמטיקה של רשתות מסוג זה, אני ממליץ מאוד לקרוא אותו על מנת להבין בצורה מעמיקה את המאמר הבא (הוא גם טוב בלי קשר...).

במאמר זה אציג רשתות נוירונים מסוג אחר, המשמשות למטרות אחרות כגון ניתוח נתונים בצורה סדרתית, ואסקור שימושים של מודלים כאלו לזיהוי איומים. המאמר יכיל לא מעט מתמטיקה, אני חושב שזה הכרחי על מנת להבין את הדברים לעומקם בנושא. עם זאת, במהלך המאמר אני לא אתעמק במיוחד בפורמליות המתמטית אלא רק אציגה לצורכי הבנה ואתן אינטואיציות עבור אופרטורים וכללים שבהם משתמשים על מנת להגיע למסקנות.

חלק מהמשוואות והתרשימים המופיעים במאמר זה, לקוחים מהמצגת לקורס למידה חישובית (ובאישורו של המרצה).

מדוע צריכים רשתות מסוג שונה מ-FNN?

לרשתות FNN או גם CNN ישנו חיסרון על פני רשתות מסוגים רקורסיביים - שעוד מעט אסביר במפורט מה הן, החיסרון הוא בנוגע לניתוח בצורה סדרתית של נתונים, ברשתות FNN כאשר אימנו את דוגמת האימון המתויגת $\{x^{(i)}, y^{(i)}\}^1$, לפני דומגת אימון מתויגת אחרת $\{x^{(j)}, y^{(j)}\}$ וקיבלנו מטריצת משקלות (פרמטרים של המודל) Θ

מטריצת המשקלות אינה תשתנה כתלות בסדר של דוגמאות האימון. נסביר מדוע עבור שתי שיטות האימון לאלגוריתם ה-Descent Gradient לרשתות כאלה.

¹ זהו סימון לדוגמת האימון ה- i הנתונים הם $x^{(i)}$ והתיוג של הדוגמא הוא $y^{(i)}$ זהו סימון שששתמש בו לאורך המאמר

אך לפני כן, רק אזכיר בקצרה: באלגוריתם ה-Gradient Descent המשקלות (פרמטרים של המודל) ניתנות על ידי "ניסוי וטעייה" של המודל לנסות להביא למינימום פונקציית מחיר שתלוייה ברצוי (התיוג האמיתי של דוגמת האימון) ובמצוי (התיוג שאותו הרשת שלנו נותנת ביחס לדוגמת אימון) או כפי שניתן לרשום בצורה מתמטית $\sum_{i=1}^m E(h_{\theta}(x^{(i)}), y^{(i)})$ כלומר סכום כל השגיאות (מעבר על כל דוגמאות האימון ישנן m כאלו) שיש במודל בין פונקצייה שתלוייה במשקלות ודוגמאות האימון לבין התיוג האמיתי (E) כמובן יכולה להיות כל פונקציה שתייצג שגיאה בכל דרך שנרצה - ותתאים בצורה מיטבית למודל ישנה דרך להסיק פונקציות כאלו בצורה חכמה הנקראת שערך סבירות מירבית, את השיטה נשאיר למאמר אחר כבר, למתעניינים (MLE).

נשים לב שהדברים היחידים שמותר לנו לשנות הן המשקלות (θ) . לכן ניתן לרשום $cost(\theta) = \sum_{i=1}^m E(h_{\theta}(x^{(i)}), y^{(i)})$.

מה שאלגוריתם ה-gradient descent בעצם אומר לנו, זה לנסות את הדבר הבא: אם אנחנו רוצים להגיע למינימום המחיר אנחנו נעדין את הפרמטרים שלנו באופן שילכו לכיוון המינימום. ובצורה מתמטית זה נראה כך: $\theta = \theta - \alpha \nabla cost(\theta)$ (כאשר ∇ היא הנגזרת החלקית ביחס לוקטור θ , וצריך לשים לב לכך שרשמתי הכל בצורה וקטורית וזה בדיוק הגבול הדק בין שיטות האימון):

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

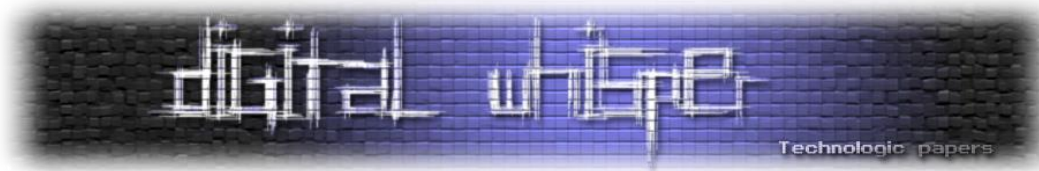
α - נקרא קצב הלמידה. זהו אלגוריתם המבצע באופן איטרטיבי צעד לכיוון הירידה התלול ביותר של J . "שתי השיטות לעדכון הן:

1. Batch Gradient Descent - השיטה עוברת על כל הדוגמאות עבור כל קבוצת האימון בכל צעד.
2. Incremental or Stochastic Gradient Decent - השיטה עוברת בכל צעד על דוגמת אימון אחת בלבד, ומעדכנת את וקטור הפרמטרים בהתאם לשגיאה.

עבור שיטת ה-full batch (שיטה 1) מה שמתבצע הוא: שעבור כל θ_j מעדכנים אותו רק לאחר מעבר על כל דוגמאות האימון (ולקחת הממוצע בדרך כלל), מכן זה ניכר שהשינוי של איזה דוגמת אימון תופיע לפני האחרת אינו משנה הרי חיבור (של השגיאות) הוא חילופי.

עבור שיטת ה-online (שיטה 2) כאשר יש שגיאה, מעדכנים אותו באופן מיידי אז כיצד נסביר את החוסר בשינוי? הרי פה נקבל בכל פעם נגזרת קצת אחרת: $\theta_{t+1} = \theta_t + \alpha \nabla cost(\theta_t)$ (וזה אחרי כל דוגמת אימון בודדת). אבל זה כבר תלוי בקצב הלמידה α , שעל מנת לבצע התכנסות צריך להיות קטן מספיק ולכן מבחינת סדרי הגודל (של θ לעומת α) אפשר לומר $\theta_{t+1} = \theta_t + \text{very small const} \approx \theta_t$, ולכן נוכל לומר שהשינוי במשקלות הוא $\Delta \theta_{epoch} = \sum_{i=1}^N \alpha \nabla cost_i(\theta_{start})$ - לכן גם כאן נשתמש בתכונה הכל כך חשובה של חיבור, הוא חילופי.

כמובן שבשיטה הזו יכולה להיות משמעות לסדר אימון הדוגמאות אך באופן כללי הוא לא משפיע בצורה משמעותית על הפרמטרים.



רק אציין שגם עבור שיטות כמו mini-batch שבה מעדכנים את הפרמטר על פי מספר דוגמאות קבוע (ולא כולן), זה מעין שילוב של השיטות הנ"ל שמשמש באופן תדיר בפרקטיקה, אין שינוי משמעותי עבור סדר דוגמאות האימון.

מדוע סדר יכול להוות מרכיב חשוב מאוד במודלים כאלו?

נסתכל לדוגמא על רשת פשוטה ואבסטרקטית (ללא פירוט מעמיק את זה נשמור לדוגמא שתלווה אותנו בהמשך) שחוזרה כמה זמן ייקח לסיים את המשימה על פי תיאור טקסטואלי של המצב:

נשתמש בעובדה שסדר המילים של קיים ולכל משנה את משמעות המשפט (דוגמא מתחום הלוגיקה מסדר ראשון):

• משפט I : קיים באג שכל אחד מהקבוצה A מנסה לפתור $(\exists y \forall x (...))$

• משפט II : כל אחד מהקבוצה A מנסה לפתור באג אחד $(\forall x \exists y (...))$

$$^2A = \{\text{תומר גולני, איתי ניר}\}$$

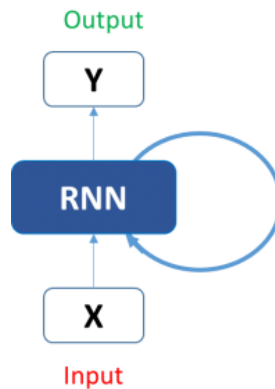
משמעות המשפטים היא שונה (בראשון כולם מנסים לפתור אותו באג ובשני כל אחד פותר באג אחר). וזה נובע מסדר הכמתים (הקשרים הלוגיים) אך רשת נירונים שתיקח נוכחות של מילים בלבד (0 לערך הקלט אם אין מילה ו-1 אם יש) לא תוכל להבחין בשונות של התוצאות ותחזה אותה תוצאה מה שייתן תוצאה שגויה (כמובן שבדוגמא הזו החוסר התאמה של המודל תלוי גם במבנה הרשת אך בפרק הבא נראה איך אפשר לפתור את הבעיה במקרה הכללי שלה).

אז מה האלטרנטיבה אם כך?

נעת נציג אלטרנטיבה ל-FNN שייתחשב בצורה מספקת אודות הסדר של הדוגמאות אימון (גם בו יש בעייה שנצטרך לפתור...) לשם כך נציג רשתות מסוג Recurrent Neural Network - RNN ישנם הרבה סוגים שונים ומשונים לרשתות כאלה נציג את הרעיון הבסיסי ולאחר מכן נדבר על הבעיות של רשתות כאלה.

² אלו סתם שמות מומצאים לצורך הדוגמא ילדו אותנו הרבה

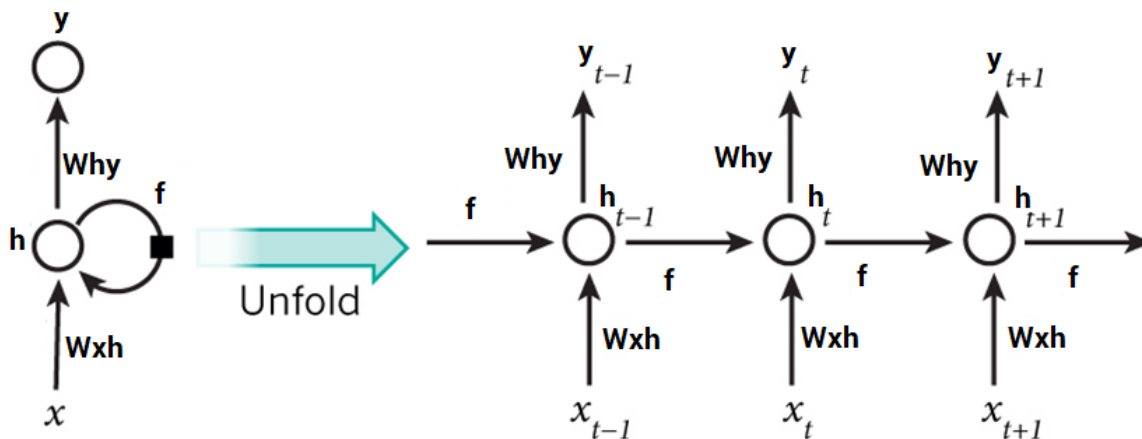
נירון RNN אחד נראה מהצורה הבאה:



[מקור]

הסבר על התרשים: כעת לקלט X נוסף גם מימד הזמן - הסדר שבו הדוגמאות אימון מופיעות לכן אפשר לסמן בכל פעם X_t את הקלט בזמן t במקרה מהסעיף הקודם הקלט יכול להיות כל מילה בנפרד (קיים ולאחר מכן באג וכן הלאה עבור המשפט הראשון), הנירון RNN בעצם בעל "שני פלטים" פלט אחד עבור הפלט האמיתי עבור הפלט ה- t Y_t וכמו כן הוא מעביר לנירון הבא לשלב $t+1$ זהו פלט שעובר כך בכל הרשת וככה בעצם מייצג את מימד הסדרתיות והחשיבות לזמן כי מתחשבים רק במה שבא לפני השלב ה- t .

ותרשים יותר מפורט של הרשת:



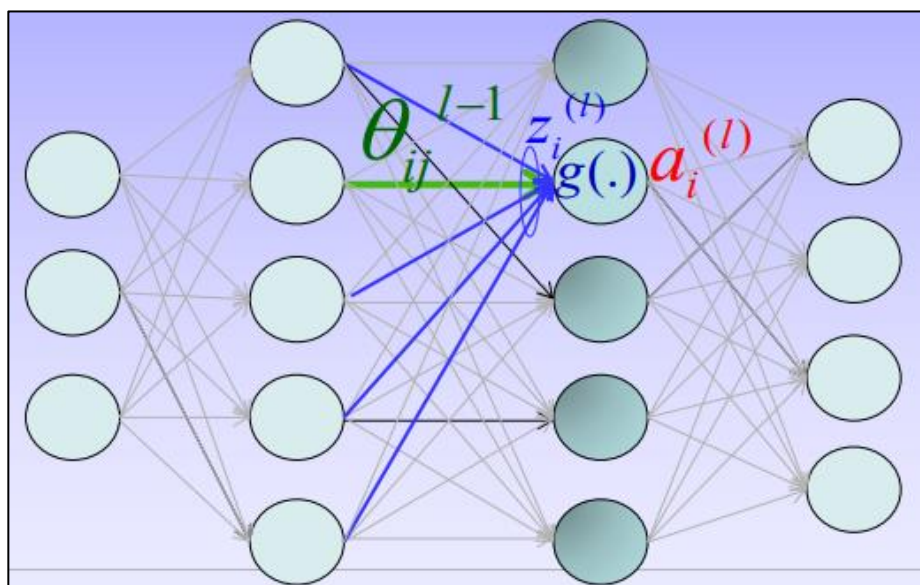
[מקור]

x_t היא הקלט בזמן t , Wxh הן המשקלות על הקלט, Why הן המשקלות אחרי כל השכבות החבויות שיש הן אחראיות על הפלט ה- t , y_t ועכשיו לחידושים: h_t הוא "המצב" שמחזיק מידע על הרשת בנקודת זמן ה- t , מכונה גם מצב נסתר (מכיוון שהמשתמש לא יודע שהוא קיים באמת). זה בעצם מסמל את כל מה שהתקבל בשלבים $1, 2, 3, \dots, t-1$ וככה בעצם נוצר מצב של חשיבות לזמן, כי כל פעם אנחנו מתחשבים רק במה שהיה לפנינו ולא בכל הקלטים f היא פונקציה על המצב הזה שדואגת להעביר אותו בצורה מיטבית לשלב הבא, הכל נראה בסדר אז היכן הבעיה?

עדכון המשקלות ובעיית הגרדיאנט הנעלם/מתפוצץ

נזכיר את הגישה שנקטנו ברשתות נוירונים רגילות על מנת לעדכן את המשקלות כך שימזערו את פונקציית המחיר (נעזר בטרמינולוגיה רבה מ-[רשתות נוירונים למשועממים בלבד](#) עמודים 22-23) אך כעת האלגוריתם יהיה טיפה שונה מאלגוריתם ה-Back Propagation:

המטרה של היא למזער את פונקציית המחיר ואת זה עושים באמצעות ה- gradient descent שאותו הזכרנו קודם, ולשם כך צריך למצוא את $\frac{\partial \text{cost}(\theta)}{\partial \theta_{ij}^{(l-1)}}$ או בצורה גרפית ותיאורית:



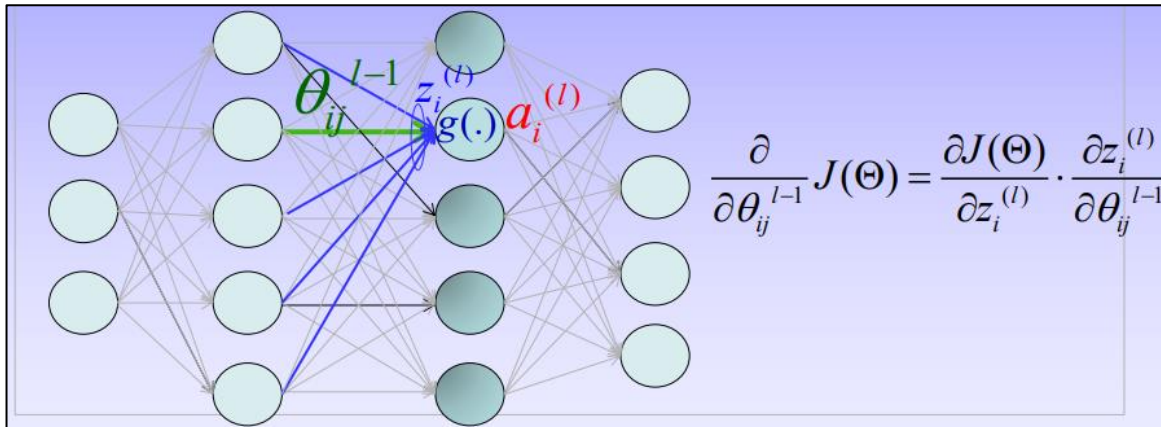
צריך למצוא את השינוי של פונקציית המחיר שנובעת מהמשקולת שמחברת בין הנוירון ה- j בשכבה ה- $l-1$ לבין הנוירון ה- i בשכבה ה- l וללכת במורדו על מנת למזער את פונקציית המחיר, וכן צריך לעשות זאת עבור כל המשקולות כדי לעדכן כל אחת מהן לצורתה האופטימלית (כן הכתיב המתמטי קצת מוזר אבל זו הקונבנציה), כפי שזכור מהמאמר הקודם זה לא דבר כל כך פשוט, כיוון שיש פה הרבה תלות בין המשקולות לבין עצמן וצריך לנתח את ההשפעות בזירות וכך להגיע לתוצאה המתאימה.

אעבור במהירות על איך זה מתבצע ברשתות FNN רק לשם תזכורת מכיוון שהנושא כוסה כבר במאמר הקודם שאני מאוד ממליץ לקרוא מי שעדיין לא קרא והגיע לפה. נשתמש בסימונים שמוצגים באיור הבא:

$$z_i^{(l)} = \sum_{r=0}^{S_{l-1}} \theta_{ir}^{(l-1)} a_r^{(l-1)}$$

כלומר $z_i^{(l)}$ הוא מה שנכנס ללנוירון ה- i בשכבה ה- l (מכפלה של המשקולות שמחברות בין השכבות לבין הפלטים של השכבה הקודמת - כפי שמוצג בציור).

וגם נגדיר את $\delta_i^{(l)} \equiv \frac{\partial J(\Theta)}{\partial z_i^{(l)}}$ על מנת לקצר את הסימנים (עוד מעט נבין את משמעות הסימון). נשתמש גם בכלל השרשרת לנגזרות הלא $((2x+3)^2)' = (2x+3)' \cdot ((...)^2)'$ או בצורה יותר פורמלית $f(g(x))' = f'(g(x)) \cdot g'(x)$ ועכשיו עבור נגזרות חלקיות - $\frac{\partial a}{\partial c} = \frac{\partial a}{\partial b} \cdot \frac{\partial b}{\partial c}$ (זה גם ממש כיף לראות בצורת הכתיב של לייבניץ כי זה ממש נראה שאפשר "לצמצם") נעשה בו שימוש לגבי התכונה הבאה:



ואם נחשוב על ההיגיון של זה, הפונקציית מחיר מושפעת מהמשקולות שמחברת בין j ל- i קודם כל מהשפעת כל הקלטים והמשקולות (צירוף שלהם - זה בעצם $z_i^{(l)}$) ואז מהשפעת המשקולות הספציפית על החיבור הזה, ולכן מבחינת שינוי של רישום לפי הסימנים שהגדרנו נקבל:

$$\frac{\partial}{\partial \theta_{ij}^{l-1}} J(\Theta) = \delta_i^{(l)} \cdot \frac{\partial z_i^{(l)}}{\partial \theta_{ij}^{l-1}}$$

ולכן רק נשאר לחשב את הביטוי הימני במכפלה ועם קצת מתמטיקה נקבל:

$$\frac{\partial z_i^{(l)}}{\partial \theta_{ij}^{l-1}} = \frac{\partial}{\partial \theta_{ij}^{l-1}} \sum_{r=0}^{S_{l-1}} \theta_{ir}^{l-1} a_r^{(l-1)} = a_j^{(l-1)}$$

ההיגיון מאחורי זה הוא שכמה שהכמות שינוי שנוצר על חיבור כל הפלטים וצירופם עם המשקולות על ידי המשקולות הספציפית תלוי בפלט הספציפי הרי כך הם קשורים (במכפלה):

$$\frac{\partial}{\partial \theta_{ij}^{l-1}} J(\Theta) = \delta_i^{(l)} \cdot a_j^{(l-1)}$$

קיבלנו את הכלל הבא שמספר לנו בדיוק באיזה כיוון צריך לעדכן כל משקולת אבל שלא ניתן לרישום היפה להטעות אנחנו עדיין לא באמת יודעים כיצד לחשב את ה- $\delta_i^{(l)}$ שמופיע כחלק מרכזי בעדכון הזה.

אז איך נחשב אותו? זה דורש מעט יותר מאמץ, נתחיל מלנסות לנתח רק את השכבה האחרונה - השכבה ה-L:

$$\delta_i^{(L)} \equiv \frac{\partial J(\Theta)}{\partial z_i^L}$$

זה כמובן תלוי בפונקציית המחיר לא אדגים כאן את החישוב תוצאות החישוב עבור פונקציות כגון [MSE](#) או [cross entropy loss function](#) נקבל עבור כולן תוצאת חישוב דומה שהיא $a_i^{(L)} - y_i$ וזה אכן הפירוש לסימן שמתמטיקאים ופיזיקאים (ועכשיו גם במובן של מדעי המחשב ולמידה חישובית) אוהבים כל כך לייחס לסטייה מהמקור - רצוי פחות מצוי או מצוי פחות רצוי, את החישוב לא הראתי כי הוא נראה שונה בצורה ניכרת בין פונקציות העלות השונות.

אוקי, אז גילנו שבשכבה האחרונה אכן השגיאה היא מצוי פחות רצוי אבל צריך גם לחשב את ה- δ -ות עבור שאר השכבות.

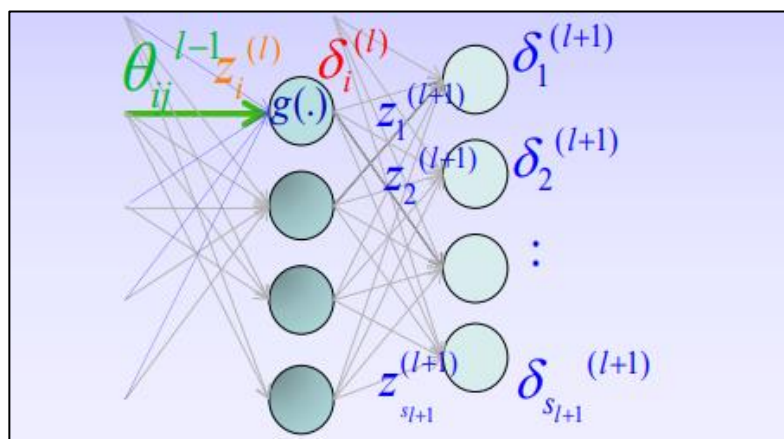
נסביר את כלל השרשרת עבור פונקציות בכמה משתנים לשם כך: נניח שישנם שני משתנים בלבד (אפשר להכליל ובכל זאת זו רק אינטואיציה ולא [הוכחה פורמלית](#)). אם אנחנו על הר וגובהה הר תלוי בקורדינאטות (x,y) שבהן אנו נמצאים כעת ואנחנו מתקדמים כעת על ההר הזה (כעבור זמן מה) אז השינוי בזמן שעברנו יהיה שווה לכמה להתקדם בכיוון x משפיע כפול כמה התקדמנו בו ועוד ההשפעה של התקדמות בציר y כפול כמה התקדמנו בו. או מתמטית:

$$\frac{dz}{dt} = \frac{\partial z}{\partial x} \cdot \frac{dx}{dt} + \frac{\partial z}{\partial y} \cdot \frac{dy}{dt}$$

ניעזר בכלל הזה על מנת להגיע למסקנה לגבי מציאת ה- δ -ות:

$$\delta_i^{(l)} = \frac{\partial J(\Theta)}{\partial z_i^{(l)}} = \sum_{k=1}^{s_{l+1}} \frac{\partial J(\Theta)}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}} \quad l = L-1, L-2, \dots$$

לפי כלל השרשרת שהסברנו את האינטואיציה שלו קודם לכן המתמטיקה תקפה, הסיבה לכך שנעשה צעד כזה הוא מאופן מבנה הרשת כפי שנראה פה:



בעצם ההשפעה של $z_i^{(l)}$ על פונקציית המחיר תלויה בהשפעות של כל אחד מה- $z_k^{(l+1)}$ על הפונקציית מחיר (כי דרכם ההשפעה הזאת עוברת לפונקציית המחיר). נשתמש בביטויים המגדירים את $z_i^{(l)}, a_i^{(l)}$:

$$z_i^{(l)} = \sum_{j=0}^{S_{l-1}} \theta_{ij}^{(l)} a_j^{(l-1)}$$

$$a_i^{(l)} = g(z_i^{(l)})$$

כאשר $g(z)$ היא פונקציית האקטיבציה של נירון. נחשב את $\frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}}$:

$$\frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}} = \frac{\partial}{\partial z_i^{(l)}} \sum_{j=0}^{S_{l-1}} \theta_{kj}^{(l+1)} a_j^{(l)} = \frac{\partial}{\partial z_i^{(l)}} \sum_{j=0}^{S_{l-1}} \theta_{kj}^{(l+1)} g(z_j^{(l)})$$

$$= g'(z_i^{(l)}) \cdot \theta_{ki}^{(l+1)}$$

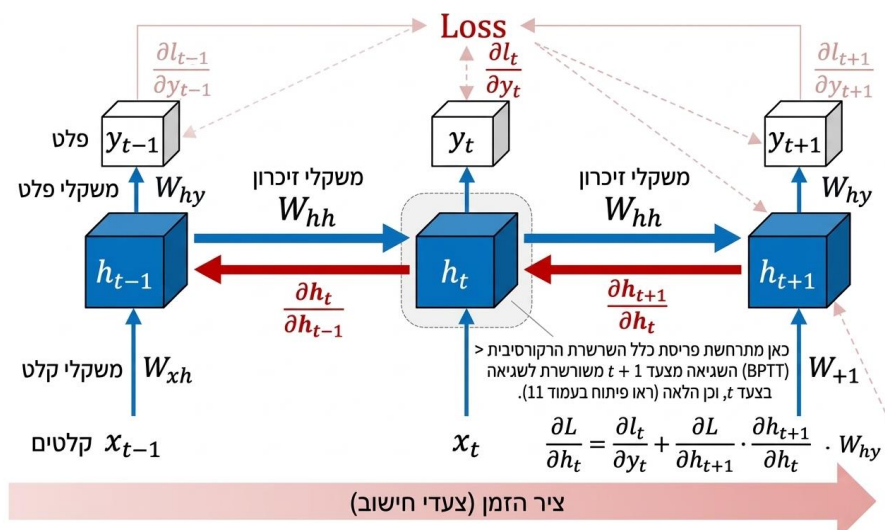
ולכן:

$$\delta_i^{(l)} = \sum_{k=1}^{S_{l+1}} \theta_{ki}^{(l)} \delta_k^{(l+1)} g'(z_i^{(l)})$$

$$= g'(z_i^{(l)}) \sum_{k=1}^{S_{l+1}} \theta_{ki}^{(l)} \delta_k^{(l+1)}$$

כלומר ה- δ של השכבה הנוכחית תלוי ב- δ -ות של השכבות הבאות בצורה רקורסיבית עד השכבה האחרונה. וזה בעצם התנאי בסיס של הרקורסיה הזו, מכן השם המאוד מתאים של הנוסחא צריך להזרים לאחר את הנתונים אודות ה- δ -ות מהשכבות הבאות על מנת לעדכן את הנוכחיות, בהחלט יפה.

נעבור עכשיו לאלגוריתם ה-BPTT - BackPropagation Through Time שמחשב את כל השגיאות ואיך לעדכן רשתות מסוג RNN: עד עכשיו עם אלגוריתם ה-BP הרגיל הזנו את השגיאות מהשכבה ה- ℓ לשכבה ה- $\ell - 1$ בצורה רקורסיבית, עכשיו בגלל מבנה הרשת זה כבר לא יעבוד, נבחן את התרשים הבא שמתאר את BPTT:



בעצם, בתרשים מעלה, מתוארת הלמידה התקנית של BPTT וגם מוצגת נקודת הכשל שלו, בתרשים רואים את הרשת הפרוסה כאשר W_{xh} היא מטריצת המשקלות שמופעלת (הכוונה למוכפלת ולאחר מכן הפעלת פונקציית אקטיבציה והוספת קבוע בדרך כלל) על כל קלט בזמן מסויים ו- W_{hh} היא מטריצת המשקלות שמופעלת על המצב הנוכחי של הרשת h_t וגם W_{hy} שמופעלת על הפלט מהשכבות הנסתרות, חשוב לזכור שזו אותם מטריצות משקלות עבור כל זמן מכיוון שצריך לשים לב שזה בעצם קורה בתהליך רקורסיבי ולכן הן לא משתנות (אם הן כן היו אומנם זה היה פותר הרבה בעיות אך צריך לשים לב שצריך לאמן עוד מספר פרמטרים עצום, וזה כבר לא פרקטי).

החצים הכחולים מתארים את התקדמות הרשת זו דומה מאוד להתקדמות ברשת רגילה מסוג Feedforward neural network וכמו השם אנו רואים שזה גם מה שקורה פה, מזינים את הקלט בכל שלב זמן (לדוגמא מילה אחת מתוך משפט) ובכל שלב יוצא פלט עבור הנקודה הנוכחית בזמן y_t שמתקבל על ידיו (וגם בשילוב של h_{t-1}) וגם מועבר מצב הרשת h_t בכל פעם קדימה.

לגבי חישוב פונקציית המחיר בתרשים סימנתי כי הוא תלוי בכל הפלטים שהיו עד לזמן שעצרנו (כמובן שאפשר לתת חשיבות פחותה לפלטים מזמנים ותיקים יותר וכדומה) זאת ארכיטקטורה שנקראת many-to-many שפולטת רצף עד כה, נותנים רצף ומקבלים רצף זה בעצם איך ש-LLM למיניהם עובדים בפשטות. ישנה עוד גישה שהפלט בזמן t אומנם תלוי בקלטים קודמים אך רק הוא משנה לנו ולכן פה פונקציית המחיר תהיה תלויה רק בפלט של הזמן t , גישה כזו נקראת many-to-one.

רק נציין ש-BP רגיל ייכשל עבור RNN מכיוון שכעת חסרה ההנחה הסמוייה שגרמה לו לעבוד כראוי על רשתות נוירונים רגילות, החוסר תלות של המשקלות בין השכבות השונות הרי ברשתות רגילות לכל שכבה הייתה מטריצת משקלות $\Theta^{(\ell)}$ שאחראית לחבר בין השכבה ה- ℓ לשכבה ה- $\ell-1$ ואין בין המטריצות השונות תלות כעת מכיוון שמדובר באותם משקלות עבור כל הקלטים נקבל מצב שזה יחשב רק עבור $t=1$ כראוי את המשקלות.

נתחיל לפתח את ה-BPTT עבור המשקלות W_{hh} (שם מסתתרת הבעיה המרכזית ולכן שם גם נפסיק את הדיון שלנו לגבי ה-BPTT).

כעת נשתמש בסימון של numpy עבור הנגזרות וכדומה כלומר הביטוי $\frac{\partial L}{\partial W_{hh}}$ לא מדבר על נגזרת חלקית לפי מטריצה של הפונקציית מחיר (מה ההשפעה של המשקלות של המצבים h_t על הפונקציית מחיר ואז יהיה ניתן ללכת לכיוון שממזער את הפונקציית מחיר), אלא הביטוי הוא כתיב מקוצר של:

$$\frac{\partial \text{cost}(\text{weights})}{\partial W_{hh}} = \begin{pmatrix} \frac{\partial L}{\partial W_{hh11}} & \dots & \frac{\partial L}{\partial W_{hh1H}} \\ \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial W_{hhH1}} & \dots & \frac{\partial L}{\partial W_{hhHH}} \end{pmatrix}$$

למי שחד הבחנה שם לב שהמימדים של המטריצה הזו של המשקלות של למצבים הן מימד $H \times H$ כאשר כל מצב h_t הוא ממימד $H \times 1$ והקלט (לכל נקודת זמן) הוא ממימד $N \times 1$.

כעת נרצה לחשב את $\frac{\partial L}{\partial W_{hh}}$, המטריצה הזו משפיעה על הפונקציית מחיר רק על ידי המצבים $h_i, i \in [1, T]$ (כל המצבים), אז נחשב את $\frac{\partial L}{\partial h_t}$ לפי כלל השרשרת שהסברנו קודם את האינטואיציה לנכונותו נקבל נתיבים, נתיב אחד הוא הנתיב שבו המידע משפיע על הפלט הנוכחי בזמן t של הרשת (הרי מה שיוצא מהשכבות הנסתרות עובר טרנספורמציה קלה - אקטיבציה ושינוי על ידי מטריצת הפלט, על מנת לקבל את הפלט ולכן תהיה השפעה מהנתיב הזה), בנוסף יש את הנתיב שממשיך על מנת "לעזור" למצב הנסתר הבא וגם בו אנחנו מתחשבים (וגם על הדרך משתמשים בכלל השרשרת - הרי ההשפעה של המצב הנסתר הנוכחי מהנתיב הזה הוא ההשפעה של המצב הבא שאליו אנחנו מעברים וזה כפול כמה המצב הנוכחי משפיע על המצב הבא).

נבחן את הקשר בין h_j ל- h_{j-1} מתקיים על פי מבנה הרשת $h_j = g(W_{hh} \times h_{j-1} + W_{hx} \times x_j + b)$.

זו תוצאה ישירה של מבנה הרשת, המצב הנסתר j מתקבל על ידי טרנספורמציה על המצב הנסתר הקודם ועוד התחשבות בקלט הנוכחי (j) (ובהתאם על כל אחד מכפלה במטריצה שגורמת לטרנספורמציה עליהם) מסמלת את פונקציית האקטיבציה שפועלת על התהליך הנ"ל (איבר איבר) ו- b הוא איבר ה-bias שנהוג להוסיף. נסמן $z_i = W_{hh} \times h_{i-1} + W_{hx} \times x_i + b$ ולפי כלל גזירת פונקציה מורכבת (נגזרת רגילה כפול נגזרת פנימית - הרגיל שאנחנו מכירים רק הסימונים טיפה השתנו) נקבל:

$\frac{\partial z_{t+1}}{\partial h_t} = \frac{\partial g(z_{t+1})}{\partial z_{t+1}} \cdot \frac{\partial z_{t+1}}{\partial h_t}$ נמשיך לפתח את הביטוי נסתכל על $\frac{\partial z_{t+1}}{\partial h_t}$ ניתן לראות כי בצמוד ל- h_t מופיע W_{hh} ולכן זה גם תוצאת הגזירה (כמו לגזור $(2x)' = 2$), $\frac{\partial z_{t+1}}{\partial h_t} = W_{hh}$. נמשיך לביטוי השני: $\frac{\partial g(z_{t+1})}{\partial z_{t+1}}$ כפי שציניתי פונקציית האקטיבציה פועלת על כל איבר במטריצה המתקבלת (מקבלת וקטור ממימד $H \times 1$ ומחזירה כזאת גם) סה"כ נקבל כי הגזירה היא $\frac{\partial g(z_{t+1})}{\partial z_{t+1}} = \text{diag}(g'(z_{t+1}))$ כלומר מטריצה אלכסונית שבכל איבר באלכסון שלה בשורה ה- i מכיל את $g'(z_{t+1,i})$.

את הסיבה הטכנית לתוצאה הזו לא אסביר (למתעניינים מוצג הסבר [כאן](#)) אך האינטואיציה של תוצאה זו היא האי תלות בין שינויים של נירונים שונים לדוגמא: התוצאה של הנירון הראשון תלויה רק בקלט הראשון, שינוי בקלט של הנירון השני, השלישי או המאה לא ישנה בשום צורה את התוצאה של הנירון הראשון ולכן מכיוון שהנגזרת המודדת כיצד פלט i משתנה בתגובה לקלט j כאשר $i \neq j$ היא בדיוק 0, כי אין שום תלות ביניהם. הנגזרות היחידות שיש להן ערך נמצאות במקומות שבהם $i = j$ (השפעת קלט על הפלט של עצמו), והמיקומים האלו בתוך מטריצת נגזרות יוצרים בדיוק את האלכסון הראשי, ושאר המטריצה נשארת מלאה באפסים.

אז סה"כ קיבלנו:

$$\frac{\partial h_{t+1}}{\partial h_t} = \text{diag}(g'(z_{t+1})) \times W_{hh}$$

אוקי אז למה זה כל כך משפיע? אם נפתח קצת בצורה רקורסיבית את הנוסחא שקיבלנו עבור $\frac{\partial L}{\partial h_t}$ נקבל:

$$\begin{aligned} \frac{\partial L}{\partial h_t} &= \frac{\partial l_t}{\partial h_t} + \frac{\partial L}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial h_t} = \\ &= \frac{\partial l_t}{\partial h_t} + \left(\frac{\partial l_{t+1}}{\partial h_{t+1}} + \frac{\partial L}{\partial h_{t+2}} \cdot \frac{\partial h_{t+2}}{\partial h_{t+1}} \right) \cdot \frac{\partial h_{t+1}}{\partial h_t} = \\ &= \frac{\partial l_t}{\partial h_t} + \frac{\partial l_{t+1}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_t} + \frac{\partial L}{\partial h_{t+2}} \frac{\partial h_{t+2}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_t} = \end{aligned}$$

אם נמשיך את התהליך עד לשלב האחרון (מספר הפעמים שמבצעים את הרקורסיה) מקבלים שבנתיב שאחראי על הקשרים ארוכי הטווח את המכפלה הבאה:

$$\begin{aligned} \frac{\partial l_T}{\partial h_T} \cdot \frac{\partial h_T}{\partial h_{T-1}} \cdot \frac{\partial h_{T-1}}{\partial h_{T-2}} \dots \frac{\partial h_{t+1}}{\partial h_t} &= \frac{\partial l_T}{\partial h_T} \cdot \prod_{k=t}^{T-1} \frac{\partial h_{k+1}}{\partial h_k} = \\ &= \frac{\partial l_T}{\partial h_T} \cdot \prod_{k=t}^{T-1} (\text{diag}(g'(z_{k+1})) \cdot W_{hh}) \end{aligned}$$

אם עדיין הבעייה לא קרצה לכם אז אני אגלה שהיא טמונה בשילוב של כך שהגזרת של הפונקציית האקטיביציה חסומה ב-1 ולכן רוב האלכסון יכול ערכים קטנים ממנו (לרוב ReLU שהנגזרת שלו היא: $f(x) = \frac{\text{sign}(x)+1}{2}$ (סתם כתיב לקטן מאפס זה 0 וגדול ממנו 1) לא ישמש את הרשת).

אבל הבעייה העיקרית היא העלאת המטריצה בחזקה סוג של זה בתוספת שינוי שלא משפיע בכל פעם:

$$\left\| \prod (\text{diag} \cdot W_{hh}) \right\| \leq \prod \|\text{diag}\| \cdot \|W_{hh}\|$$

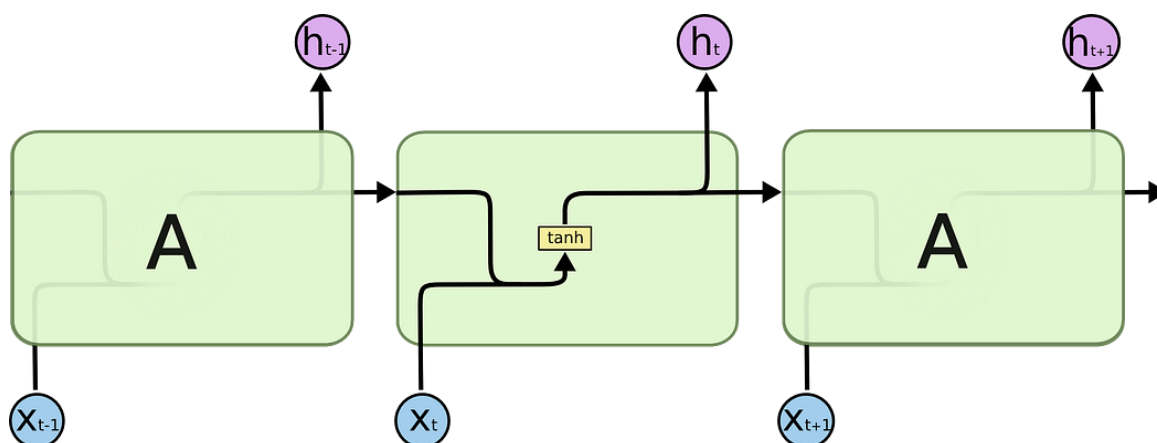
הערכים שם או יגדלו בצורה אקספוננציאלית או ידעכו כך, נשים לב שזו מטריצה ולכן לא ברור כל כך שהכפלה מספר פעמים באותו רכיב תעשה אפקט של חזקה - הרי מדובר בכפל שונה, אך ניתן להשתכנע בכך מהנושא של ערכים עצמיים ובפרט [מטריצה לכסינה](#).

לסיכום חלק זה:

1. חישוב הגרדיאנט של שגיאה עתידית לפי מצב קודם מחייב פריסה של כלל השרשרת
 2. הפריסה יוצרת מכפלה מתמשכת של שני רכיבים המשולבים יחד לסירוגין: מטריצת המשקולות המשותפת W_{hh} והמטריצה האלכסונית המייצגת את נגזרות האקטיבציה המקומית $\text{diag}(g'(z))$
 3. הנגזרות של פונקציות האקטיבציה הקלאסיות חסומות ב-1 (למשל במקסימום 0.25 עבור Sigmoid)
 - לכן, המטריצה האלכסונית מחדירה שברים עשרוניים שגורמים לדעיכה בכל מעבר
 4. הפעלת אותה מטריצת משקולות לאורך כל צעדי הזמן יוצרת אפקט ישיר של העלאה בחזקה (מה שידוע כפונקציה מעריכית).
 5. בדומה להתנהגות של ערכים עצמיים במטריצה לכסינה, אם הבסיס שמיוצג על ידי המטריצה קטן מ-1 המכפלות החוזרות ידעו לאפס (היעלמות) ואם הוא גדול מ-1 הן יזנקו עד לקריסה (התפוצצות).
 - כן הייתה פה הרבה מתמטיקה והיא יחסית מורכבת, היה אפשר לשמור על הדיון ברמת האינטואיציה בלבד, אך אני חושב שלהבין לעומק את התהליכים צריכים לראות מה באמת קורה.
- אוקי אז הבנו שהפתרון התיאורטי הוא לשבור את רצף ההכפלות הללו שגורמת לדעיכה/התפוצצות במהירות רבה מידי (אקספוננציליות), את הרצף הזה אפשר לשבור באמצעות נתיב עוקף שאולי ייתבצע על ידי חיבור (אבל בו בזמן לסבך את הארכיטקטורה של כל נירון - זהו בדיוק LSTM עליו נדבר כעת)

הפתרון שעובד: LSTM הארכיטקטורה והתיאוריה

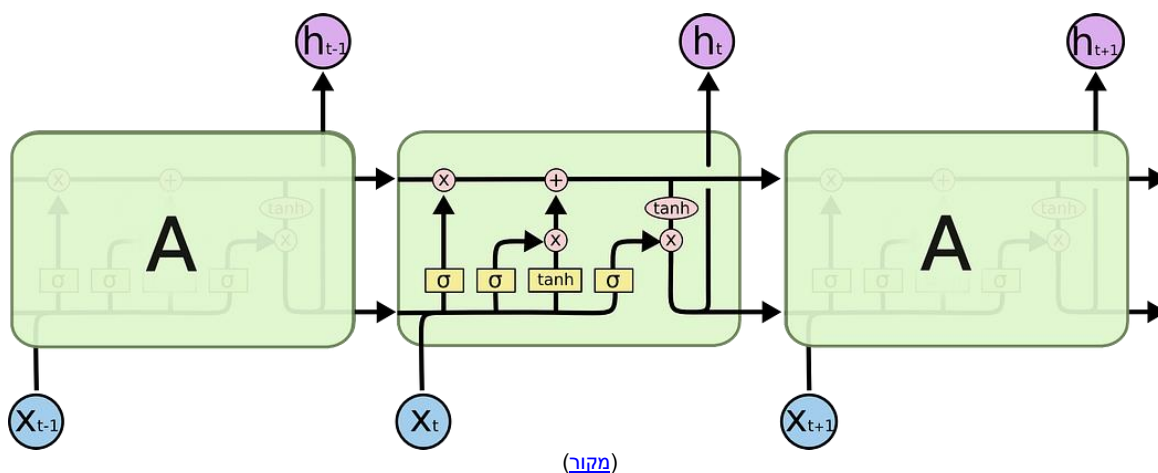
עד כה הרשת פעלה בצורה הבאה:



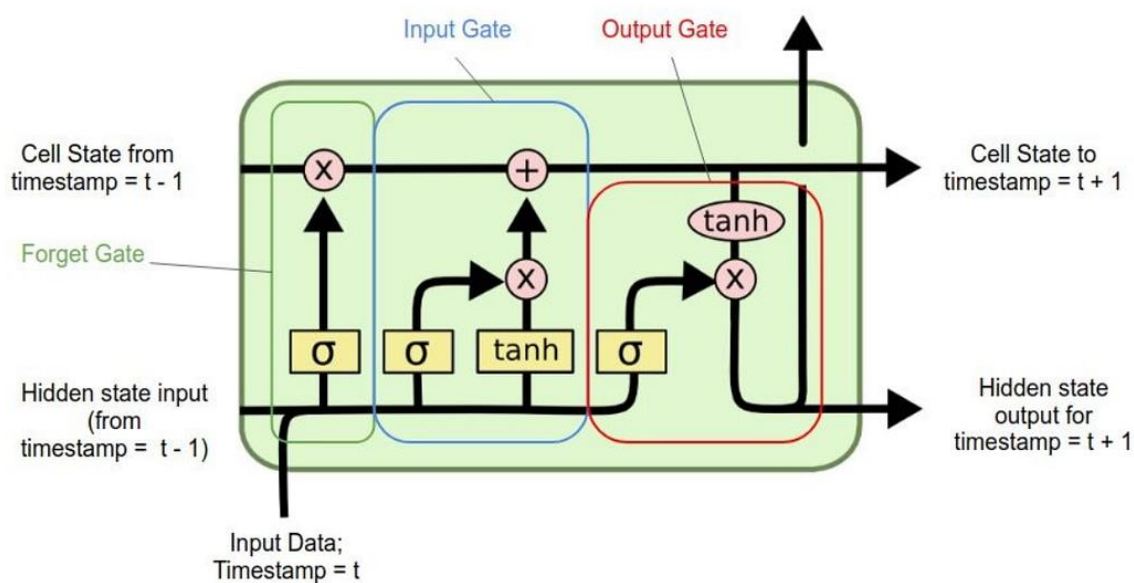
[מקור]

כלומר קיבלנו קלט הוצאנו כפלט מצב נסתר של הקלט הנ"ל (עם טרספורמציה עליו) וגם שלחנו אותו כ"קלט" למצב הנסתר הבא (גם פה עם טרספורמציה עליו). בצורה הזאת קיבלנו את הבעייה העיקרית בארכיטקטורה הזו - נטייה לחוסר התחשבות בנתונים מהמצב הנסתר והתחשבות רק בקלט הנוכחי.

כעת נעבוד עם ארכיטקטורה שמתמודדת עם הבעיה הזו:



בואו ניכנס לציור היפה שיש בכל נירון כזה ברשת ובבין מה המטרה של כל רכיב (שער):



כפי שניתן לראות ישנם שלושה שערים:

1. שער השכחה

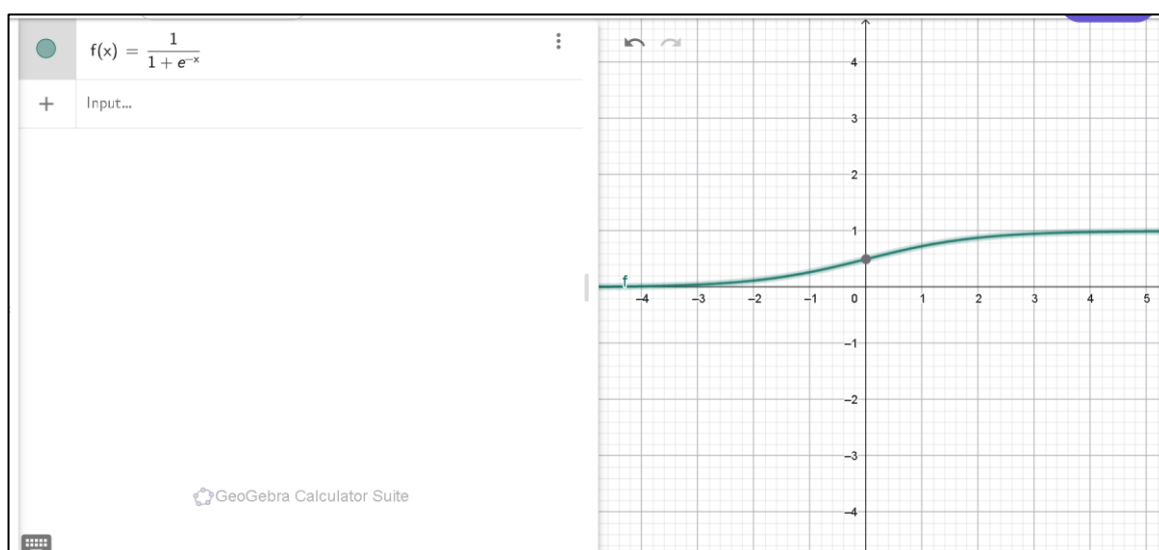
2. שער הקלט

3. שער הפלט

שער השכחה:

לפני שנתחיל לדבר על מה ששער השכחה מציג לנו צריך להבחין כי בעצם נוסף למצב הנסתר עוד מצב אודות התא, ויש לו את השם המאוד מקורי מצב התא הוא מהווה בארכיטקטורה הזו מסלול יותר ישיר עבור מצבים מהתאים הקודמים וכך מתבצעת מניעה עבור בעיית הגרדיאנט המתפוצץ/נעלם שראינו קודם.

השער הבא לוקח את השרשור של הקלט הנכנס בשלב הנוכחי (השלב ה- t) ומשרשר אותו (ממש מצמיד לסופו) את המצב הנסתר מהשלב הקודם מה שמסומן כ- $[x_t, h_{t-1}]$ את הוקטור המשרשר הוא כעת מכפיל במה שנותר לנו לאמן עבור השער הזה מטריצת המשקלות W_f (כסימון למילה *forget*) ולתוצאה הזו כנהוג נוסיף איבר הטייה b_f וקטור קבועים שיתווסף לתוצאה שקיבלנו ועל כל הדבר הזה נפעיל את פונקציית הסיגמואיד המוכרת שנזכור שנותנת תוצאה בין 0 ל-1, בצורה הבאה:



ולכן היא מעולה ללקיחת אחוזים של שימוש שאותם אנו רוצים מאיברים מסויימים וכמו כן גם להסתברות. בכתוב מתמטי השער נראה כך: $f_t = \sigma(W_f \times [h_{t-1}, x_t] + b_f)$, כלומר נקבל וקטור שמורכב מ- H שורות שכולו ממספרים בין 0 ל-1 - שנותנים משמעות כלשהי, אז למה זה נקרא שער? כי כעת אנחנו לוקחים את המצב תא הקודם C_{t-1} ומבצעים כפל איבר איבר בין התוצאה שקיבלנו לבין המצב תא הזה ובסימון הנהוג כעת C_t יהיה מורכב מ- $C_t = f_t \odot C_{t-1}$ (בהמשך נראה ממה עוד הוא מורכב) וכך עבור איברים שנרצה "לשכוח" - לא להעביר סביר שהתוצאה של השער עליהם תהיה ≈ 0 (כך גם עבור מידע שנרצה לשמור ≈ 1).

שער הקלט:

השער הבא אחראי על הוספת מידע חדש לתוך מצב התא הנוכחי C_t על מנת לעשות זאת הוא פועל בשני שלבים 'יצירת המידע הפוטנציאלי' שעשוי להיכנס ו'יצירת' 'מסיכה שמחליטה איזה חלק מהמידע שנוצר באמת ייכנס לזיכרון, בדומה מאוד לשער השכחה גם פה המסיכה היא מהצורה הבאה:

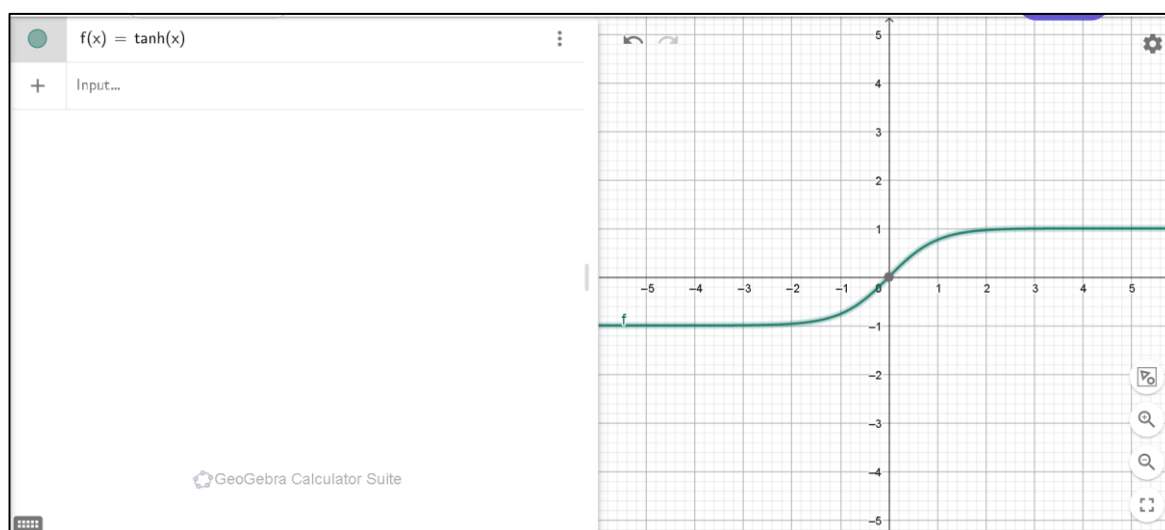
$$i_t = \sigma(W_i \times [h_{t-1}, x_t] + b_i)$$

מדובר פה במטריצה ואיבר הטייה שונים מהשער הקודם אך המנגנון של הסינון מידע זהה פה.

כעת צריך להחליט איזה מידע מהקלט הנוכחי נכנס ולכן גם שמו של השער לשם כך גם נשתמש בקלט הנוכחי ובמצב הנסתר שקיבלנו קודם לכן:

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

נשים לב לפונקציית אקטיבציה השונה במעט שמשומשת כאן הטאנגס ההיפרבולי, הפונקציה הזו דומה לסיגמואיד רק שהיא יכולה לקבל ערכים שליליים בנוסף, זה נובע בעיקר מהסיבה שאילו היינו משתמשים בסיגמואיד על מנת לכווץ את הערכים היינו מקבלים רק ערכים חיוביים ולכן מצב התא תמיד היה גדל. כפי שניתן לראות הערכים חסומים $\tanh(x) \in [-1, 1]$:



כעת נרשום את משוואת מצב התא סופי (גם לאחר שער השכחה וגם לאחר שער הקלט):

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

כלומר מתרחש חיבור של מידע נוסף מהקלט הנוכחי וגם עליו שולטים כמה ממנו יהיה רלוונטי לכל אלמנט.

שער הפלט:

כעת השער הזה אחראי על הפלט של הזמן הנוכחי t הפלט הזה הוא גם המצב הנסתר h_t גם לשער הזה כמובן שקיימת מסכה הרי הוא לא ישאיר את שני השערים הנוספים עם שוני מהותי משלו והיא נראית מהצורה הבאה (דומה למסכות הקודמות):

$$o_t = \sigma(W_o \times [h_{t-1}, x_t] + b_o)$$

וכעת המסכה הזו עוזרת להחליט איזה מידע רלוונטי כעת (לטווח הקצר), ונקבל את הפלט (והמצב הנסתר) לפי הכלל הבא:

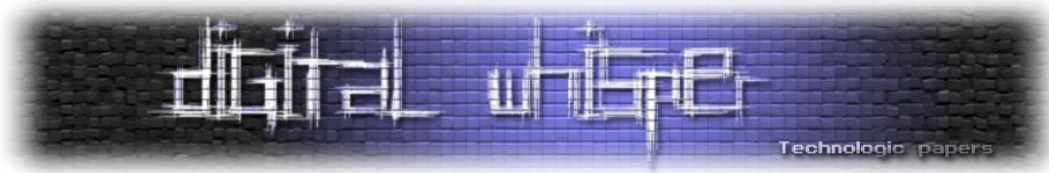
$$h_t = o_t \odot \tanh(C_t)$$

בואו נדבר קצת על משמעותו של כל רכיב פה.

כתזכורת LSTM פירשו Long Short - Term Memory, זה נשמע כמו סתירה פנימית (הרי המילים הראשונות הן בעצם ניגודים, לטווח הארוך לעומת טווח הקצר). בעצם, המצב תא הוא הרכיב בעל הזיכרון לטווח ארוך LTM-Long Term Memory המידע זורם לאורכו עם לא הרבה התערבות חיצונית והוא יציב בעקבות הכיוון, המסיכות ופעולת החיבור לעומת הכפל שאותה ראינו ב-RNN וכך המצב תא משמר מידע רלוונטי גם לאורך צעדים רבים. וגם פה יש את המצב הנסתר כמו ב-RNN רגיל וגם שם הוא היה הרכיב לזיכרון קצר (STM-Short Term Memory). לאחר מספר שלבים גדול ראינו שהמידע עבורו דעך, ההבדל הוא שכעת יש גם זיכרון לטווח ארוך שפחות מסתמך על הקלט הנוכחי וגם את הרכיב הזה ששם משקל גדול לקלט הנוכחי וחוזר תוצאות לטווח יותר קצר.

כעת אימון הרשת הוא דומה מאוד לאלגוריתם ה-BPTT שראינו במקרה של RNN וזה גם הגיוני הרי LSTM הוא סוג מסויים של רשת נוירונים רקורסיבית, נשים לב שגם פה המשקלות W_o, W_f, W_c, W_i והוקטורי הטיית b_f, b_o, b_c, b_i הם גלובליים (כלומר הם לא משתנים כתלות בשלבים בזמן - הרי זה תהליך רקורסיבי והפריסה שאנחנו עושים היא רק לצורך נוחות).

אני חושב שלכל התהליך שהצגנו של בעייה ופתרון הוא ממש אלגנטי, וגם לראות את המתמטיקה שאומנם נראית מפחידה במבט ראשון אך תמיד בכל פעולה ניתן למצוא את המוטיבציה שלשמה היא בוצעה וגם לראות את האינטואיציה מאחוריה.



פרויקט ופרקטיקה של המודל

אני חושב שהשילוב של הפרקטיקה מצד אחד והתיאוריה מצד שני הוא הכרחי ופותח אופקים חדשים [הפרויקט](#) שפיתחתי שואב את הבסיס התיאורטי שלו מהמאמר: ["Hunting for Insider Threats Using LSTM-based Anomaly Detection"](#). המציג התמודדות עם אחת הבעיות המורכבות ביותר בעולם אבטחת המידע: איומים פנימיים - תוקפים הפועלים מתוך הארגון, מחזיקים בהרשאות לגיטימיות, ומכירים את המערכת מבפנים.

זו נשמעת בעייה קשה מאוד להתמודדות ואכן היא כזאת, הרי ברגע שנותנים הרשאות מספקות (ולפעמים יותר מידי) לעובדים זו הופכת להיות תחרות בין המערכת ניתור על העובדים לבין העובד, הרבה פעמים את התחרות הזו, זה משימה יחסית פשוטה (דורשת הבנה ותחכום אבל לא יותר מידי) לנצח.

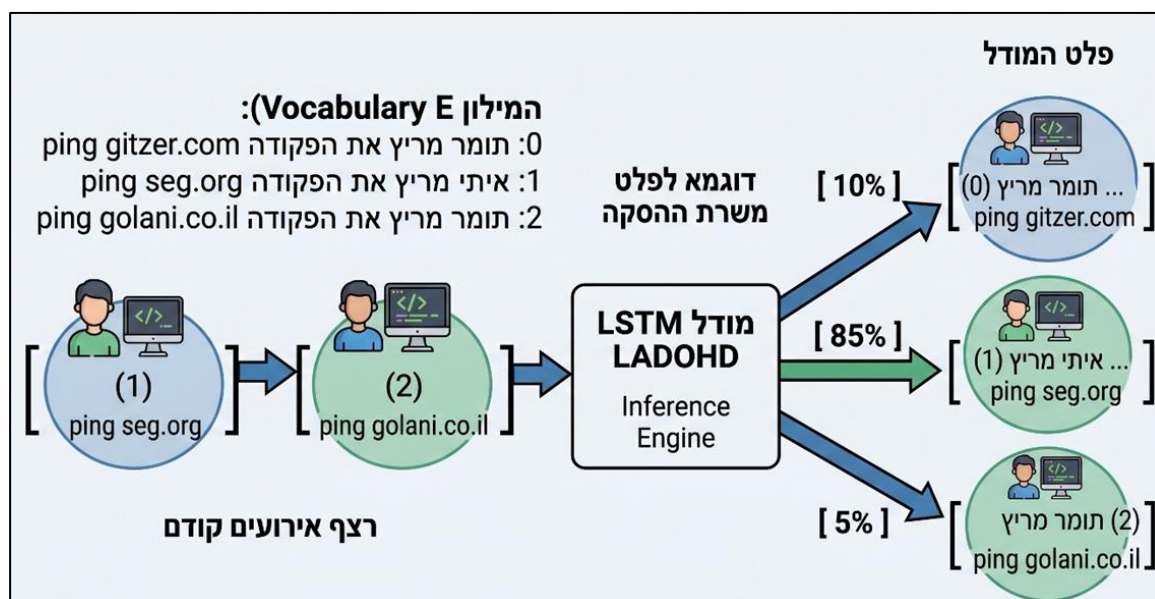
בדומה ממש להבדלים בין רשת RNN לבין LSTM אלו גם ההבדלים בין מערכות ניתור מסורתיות (שהיו בשימוש בסביבות העשור האחרון) הן ניתנות להטעיה על ידי ערבוב מספיק של פעולות שנחשבות לגיטימיות (שפירות) לבין פעולות זדוניות (אנומליות) כלומר גם פה המערכות סבלו מחוסר זיכרון (בעייה זו לעיתים מכונה OAR-Order Aware Recognition).

קצת על הפרויקט:

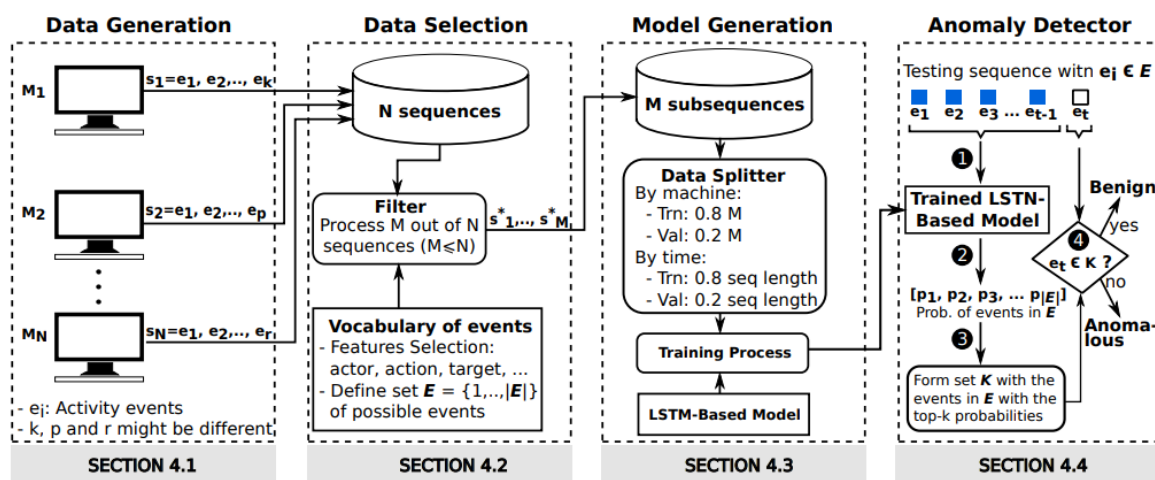
ראשית היה צריך לחפש דוגמאות אימון (למודל) שייצגו כראוי את המורכבות של תוקפים אמיתיים שמייצרים הרבה "רעש" - פעולות שנחשבות רגילות ונורמליות על מנת להתחמק ממערכות ניתור, את ההקלטה של הנתונים מבצעים באמצעות מנגנון EDR - Endpoint Detection & Response, מערכות EDR פועלות כקו ההגנה הראשון ישירות על תחנות הקצה בארגון (לדוגמא המחשבים שאיתם עובדים בחברה), מנטרות את הפעילות ברשת ובמחשב, ושומרות את המידע במסד נתונים מרכזי לצורך ניתוח והתרעה.

בכל תחנת קצה מותקן סוכן מקומי שבניגוד לאנטי-וירוס קלאסי שמחפש חתימות קוד של קבצים זדוניים (בעיקר), סוכן ה-EDR אוסף את הנתונים בזמן אמת כדי לזהות דפוסי התנהגות חריגים. הסוכן המקומי עוקב אחר כל פעולה במערכת ומתעד אותה, כל אירוע כזה כולל מידע מקיף המורכב לרוב מזהות הגורם המבצע, תיאור הפעולה והאובייקט שעליו הפעולה בוצעה. ולאחר מכן הרשומות נשלחות בזמן אמת לשרת שמנתח את ההתנהגויות על ידי למידה מכונה כדי לאתר כל התנהגות שנראית חשודה.

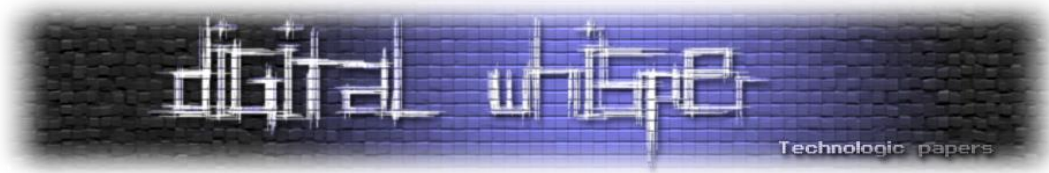
מטרת המודל היא בהינתן האירועים (תכף אגדיר מה זה במדויק) לתת לכל אירוע אחר שאנחנו מכירים (מתוך מילון האירועים שמהערכת מכירה) הסתברות להופיע אחרי לדוגמא (אבסטרקטית):



המערכת בכללותה עובדת כפי שמתואר בתרשים הבא:



בשלב הראשון צריכים להגדיר את מילון האירועים (vocabulary of events), על מנת להקל על עבודה עם מחרוזות במייצג של אירוע היה צורך בהמרת אירוע למזהה ייחודי ומספרי. המאגר שבו השתמשתי בפרויקט הוא: [קישור למאגר](#), הוא מבוסס על אותה שכבת ניטור מסוג EDR המתאורת למעלה (רישום שיטתי של אירועים במערכת הפעלה). הוא כולל עמודות כגון, שם המכונה, מקור התוכנית, חותמת זמן ארץ ועוד...



על מנת לעבד את הנתונים עם המודל היה צורך בהמרת תכונות רלוונטיות "לאירוע". אז בתחילה מיינתי את כל הנתונים מהנתוני אימון לפי חותם זמן (שהאירוע קרה בו) על מנת שלאחר מכן אוכל ליצור אירועים רציפים בזמן ביעילות וגם ולאחר מכן התייחסות רק ל-actor מסויים כלומר איפה הופעל האירוע - זה מפאת יעילות וסינון מידע הרי הרבה פעמים לא מעניין אותנו שהעובד פתח את האפליקציה של מזג האוויר (שהריצה שאילתה **בעצמה** לשרת שמחזיר לה את הנתונים - כך שאין סכנה ממשית לרוב):

```
def load_and_filter_logs(file_path, target_actor="powershell.exe") -> pd.DataFrame:
    """
    Read raw EDR logs (CSV format) and filters them by specific actor.

    Args:
        file_path (str): path to the log file.
        target_actors (str, optional): The process name to monitor. Defaults to
        "powershell.exe".
    Returns:
        pd.DataFrame: A filtered DataFrame ordered by time.
    """
    #read the log file (in csv format)
    df = pd.read_csv(file_path)
    #sort the logs by the time they occurred
    df = df.sort_values(by='Timestamp')
    #take just the rows that the actor is the one we filtering (and make a copy to not
    change the original)
    filtered_df = df[df['Actor'] == target_actor].copy()
    return filtered_df
```

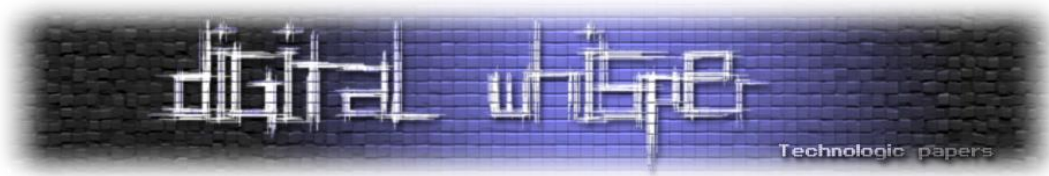
לאחר מכן יוצרים את המילון אירועים על פי עמודות נבחרות:

```
def build_event_vocabulary(df, feature_columns):
    """
    Implements the transformation function F_T(.) to map raw features into a
    categorical vocabulary.

    Args:
        df (pd.DataFrame): The filtered logs.
        feature_columns (list): List of columns to define a unique event (e.g ,
        ['EventType', 'Action', 'Target'] (column names)).

    Returns:
        tuple: (List of sequential event IDs, Dictionary mapping feature tuples to IDs)
    """
    #for each combination of those columns make unique event
    unique_events = df[feature_columns].drop_duplicates()
    #map each event to unique ID
    vocab = {tuple(row) : idx for idx, row in enumerate(unique_events.values)}
    #make all the rows in the logs to sequence of ID's (continuous list) (the df is
    sorted by Timestamps)
    event_sequence = [vocab[tuple(x)] for x in df[feature_columns].values]
    return event_sequence, vocab
```

נשים לב שהנתונים ממויינים לפי החותמת זמן כבר (מקריאה לפונקציה הקודמת ובכללי הנתונים באו ממויינים בצורה הזו כבר) ולכן אפשר ליצור בצורה הזאת רצף אירועים שלוקח את הערך במילון לאירועים (שהרגע יצרנו) ועובר עליהם בצורה כרונולוגית (הרי זה ממויין). בנוסף ניתן להבחין בעוצמה של pandas שניתן פשוט לבחור את העמודות מהן אנחנו רוצים ליצור אירועים ייחודיים ולתת לכל אחד מהם זהות ייחודית (פשוט אינדקס במקרה פה).



כעת צריך להכין את נתוני האימון למודל עצמו הרי הוא לא ממש יכול לקבל 158,000 אירועים (שתכף נראה כיצד בחרתי להגדיר) שאולי חלק מהם יסוננו ולקבל את האירוע שבא אחריו וכך להתאמן.

הדרך שבה הוא מחלק את נתוני האימון היא לרצפים בגודל קבוע ולאחר מכן התיוג של הרצף הזה (התוצאה האמיתית שלו) היא האירוע הבא בסדרה ואכן זה באמת מה שקורה בפועל וכך הרשת לומדת את הקשרים בין האירועים השונים ונותנת הסתברות לכל אירוע להיות הבא:

```
def creating_training_sequence(event_sequence, seq_length=64):
    """
    Generates 3D tensors for LSTM training using a sliding window approach.

    Args:
        event_sequence (list): The full chronological sequence of event IDs.
        seq_length (int): The BPTT unrolling window size (default: 64).

    Returns:
        tuple: (X_train numpy array, y_train numpy array)
    """
    X, y = [], []
    #make sliding windows (each windows of events is with a length of seq_length )
    #the output is the next event each time (we want to give probability to the next
    event in real time)
    #if the event is with normal probability(top K probable events) (according to the
    connection the model learns-the training data is this windows)
    #this is benign action else this is anomalous action
    for i in range(len(event_sequence) - seq_length):
        X.append(event_sequence[i:i+seq_length])
        y.append(event_sequence[i+seq_length])
    return np.array(X), np.array(y)
```

ניכנס לאיך שהמודל כעת בנוי ונסביר קצת מהאינטואיציה מאחורי זה:

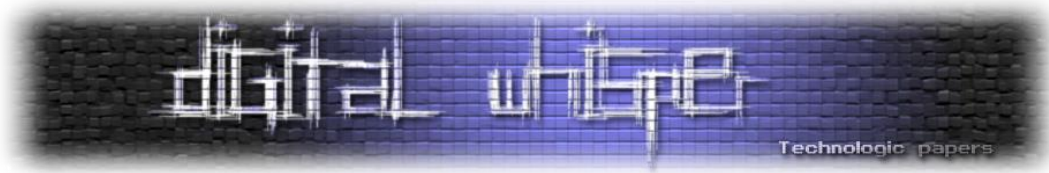
```
def build_ladohd_model(vocab_size=175, embedding_dim=16, hidden_size=64,
seq_length=64):
    """
    Builds the LADOHD LSTM anomaly detection model.

    Args:
        vocab_size (int): Total number of unique system events(fixed number).
        embedding_dim (int): Dimension of the dense embedding vectors.
        hidden_size (int): Number of features in the LSTM hidden state.
        seq_length (int): Length of the input event sequences (BPTT window).

    Returns:
        tf.keras.Model: The compiled sequential model.
    """
    model = Sequential([
        # 1. Embedding Layer: Converts categorical event IDs into dense vectors
        Embedding(input_dim=vocab_size, output_dim=embedding_dim,
input_length=seq_length),

        # 2. LSTM Layers: 3 stacked layers to learn complex temporal patterns
        # return_sequences=True ensures the entire sequence is passed to the next layer
        LSTM(hidden_size, return_sequences=True),
        LSTM(hidden_size, return_sequences=True),
        LSTM(hidden_size, return_sequences=False),

        # 3. Fully Connected Layer: Extracts non-linear features from the LSTM outputs
        Dense(100, activation='relu'),
```



```
# 4. Output Layer: Unnormalized log probabilities (logits) for each event in
the vocabulary
Dense(vocab_size)
])
return model
```

אוקי זה נראה קצר ופשוט וזה העוצמה של tensorflow - [בפריקט](#) עצמו לא החסרתי בכלל בפירוטים אודות כל שכבה ומאפייניה אך אסביר גם כאן:

השכבה הראשונה היא מסוג Embedding זו השכבה שאחראית תחזוקת הקשרים בין אירועים הראשונית שמתבצעת הרי כעת יש לנו מילון אירועים בגודל `vocab_size` אז איך נייצג באמת כל אירוע וניתן כקלט למודל?

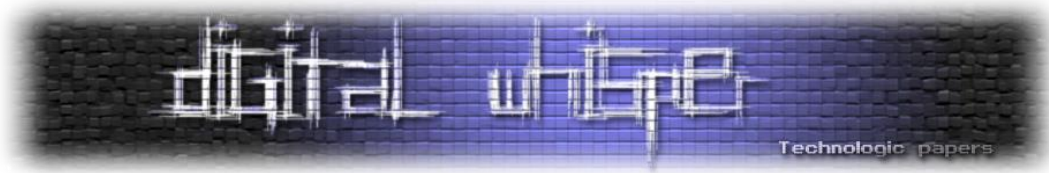
אופציה אחת היא לייצג כל אירוע בוקטור מלא באפסים למעט המקום שמייצג את ה-`id` של האירוע. שם נשים 1 - מה שנקרא *one-hot-encoding*. לשיטה הזאת כמה חסרונות הבולטים שבהם הם בזבוז המקום היחסי (כל וקטור הוא בגודל `vocab_size` שרובו מבוזבז על 0), ופעמים רבות ניתן לשים עוד מידע. לדוגמא המרחק ביניהם או המכפלה ביניהם (ועוד הרבה מרכיבים) יכולים לייצג קשר מסויים ביניהם, זה בדיוק מה שהשכבה הזו עושה: היא מתרגמת כל מאורע במילון (`vocab_size`) לוקטור באורך מסויים (`embedding_dim`) - שבדרך כלל קטן מהרבה מגודל המילון, וכעת הוא יודע לייצג בצורה חכמה הקשרים בין המרחק ועוד רכיבים זה בעצם כמו טבלת גיבוב בוחרים אירוע ומקבלים וקטור של מספרים ממשיים שמחזיק בתוכו הרבה מידע סמנטי אודות הקשרים בין האירועים וכך מוביל לחיסכון בזמן עיבוד וזיכרון.

כעת, משורשרים לשכבה הזו שלושה רשתות LSTM הכה מדוברות במאמר הזה. הפרמטר `return_sequence = True` מציין כי יש להעביר לשכבה הבאה את כל הנתונים עד כה (היא פולטת את כל השלבים בזמן לפלט הסופי), וכך נוצרת האינטואיציה הבאה לרשתות (שמוכיחה את עצמה במבחן המעשי).

LSTM ראשון אחראי לבניית הזיכרון הבסיסי - הרשת עובדת כפי שהסברנו עם השערים, פולטת את כל השלבים בזמן. LSTM שני אחראי לחילוץ תבניות יותר עמוקות - הרשת כבר מנסה לחפש קשרים בין תבניות התנהגות של אירועים גם פה פולטת את כל השלבים בזמן. LSTM שלישי כעת השכבה הזו פולטת רק וקטור אחד שמסכם את כל ההתנהגות של החלון.

לאחר מכן, הוקטור שיוצא מן השכבת LSTM האחרונה נפלט לשכבת NN רגילה כאשר כל ניוון מהשכבה הקודמת מחובר לכל אחד מהשכבה הבאה (מכן השם Dense) הפונקציית אקטיבציה היא *ReLU* כך שהיא מוסיפה מורכבות לרשת וכך מביאה יותר אפשרות למידול יותר מורכב אך לא בצורה מאוד ניכרת (רק הופכת שליליים ל-0) וכעת אליה משורשרת שכבת הפלט שלכל אחד מהאירועים נותנת מספר שיוכל אחר כך להיות מומר על ידי *softmax* להסתברות של כל אירוע להופיע אחרי.

(האינטואיציה לא תמיד ברורה ועדיין לא נמצא המבנה או האלגוריתם המשולם לבניית מודל טוב ביותר ובעצם האינטואיציה הזאת היא מה שיש על מנת להגיע לתוצאות מירביות).



כעת אימון המודל מתבצע עם אלגוריתמים ופונקציות שעברו הרבה אופטימיזציות מצד *tensorflow* אני לא אסביר בצורה מפורטת כל רכיב אך אציין בקצרה מה תפקידם ואיך הם עושים זאת (בפרויקט עצמו פירטתי ושמתי קישורים רלוונטיים להעמקה):

```
def train_ladohd_model(model, X_train, y_train, epochs=50, batch_size=64,
validation_split=0.2):
    """
    Trains the LSTM model using the prepared sequences of benign events.

    Args:
        model: The compiled Keras sequential model.
        X_train (np.array): Input sequences (Sliding windows).
        y_train (np.array): Target next-events.
        epochs (int): Maximum number of training iterations.
        batch_size (int): Number of sequences to process before updating weights.
        validation_split (float): Fraction of data to use for validation.

    Returns:
        History object containing training metrics.
    """
    #define this process of earlyStopping to prevent overfitting
    early_stopping = EarlyStopping(
        monitor='val_loss',
        patience=5,
        restore_best_weights=True
    )
    #see https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit
    #X_train and y_train are the data to train on
    #epochs is the number of iteration over the whole dataset, the training can be stop
    before the number of epochs is the maximum (early_stopping)
    #batch_size is the number of samples that the model reads in one time (the mini-
    batch I explained)
    #validation_split takes the friction given from the dataset and uses it to
    validates the model (and not for training)
    #callbacks are list of function that performed during the trainings
    history = model.fit(
        X_train,
        y_train,
        epochs=epochs,
        batch_size = batch_size,
        validation_split= validation_split,
        callbacks= [early_stopping],
        verbose = 1
    )
    return history
```

בעצם מנגנון ה-*EarlyStopping* בודק בכל פעם את השיפור בפונקציית המחריר וברגע שאין שיפור יותר מאשר סף מסויים נעצור את ההרצה לפני סיום כל ה-*epochs* (איטרציות) זה גם מונע את תופעת ההתאמת יתר (overfitting) לנתוני האימון הספציפיים.

על יתר הפרמטרים פירטתי בהערות למעלה כפי שניתן לראות.

מנגנון נוסף מעניין שניתקלתי בו הוא ה-dynamic threshold:

```
def detect_anomaly_dynamic(model, previous_sequence, actual_next_event):
    """
    Classifies an event as benign or anomalous using a dynamic threshold[cite: 1].
```

```

Args:
    model: The trained LADOHD model.
    previous_sequence (np.array): The history window (e.g., length 64) of event
IDs.
    actual_next_event (int): The ID of the event that actually occurred.

Returns:
    bool: True if the event is anomalous, False if it is benign.
"""
#make the tensor with one more dimension inorder to enable tf to process the mini-
batch in diffrenct dimension.
seq_tensor = np.expand_dims(previous_sequence, axis=0)

#get raw unnomalized scores (logits) of all the possible vocab_size events , for
every timestamp
logits = model.predict(seq_tensor, verbose=0)

#get the prediction just for final timestamp and apply the softmax function
(convert each probability to number between 0 to 1 and the sum is 1)
probabilities = tf.nn.softmax(logits[0, -1, :]).numpy()

#round to 3 places decimals
rounded_probs = np.round(probabilities, decimals=3)

#we need to calculate the probability  $P(e_t | e_{(t-1)}, e_{(t-2)} \dots)$  , for this we
select K events that are most probable
# (K is a parameter set staticly or dynamicly) if the next event is not from the K
benign events this events is anomalous.
#The static treshhold is worse than the dynamic one in real life cases.
#A method to make this treshhold is to take the most repeated (mode) probability and
make it a treshhold.
mode_result = stats.mode(rounded_probs, keepdims=False)
dynamic_threshold = mode_result.mode
#take the probability of the next event to occur
actual_event_prob = probabilities[actual_next_event]
#if lower than the treshhold this is an anomanlous
is_anomalous = actual_event_prob <= dynamic_threshold

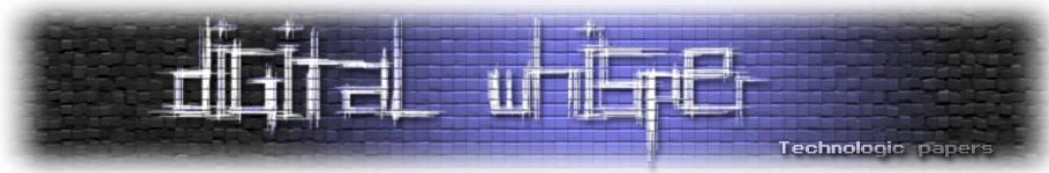
return is_anomalous

```

בעצם ישנן כמה גישות בקביעה של האם אירוע כלשהו הוא אנומליה השיטה הנאיבית היא קביעת ערך סף מסויים "אם ההסתברות של אירוע מסויים הוא 0.1% ומטה קבע אותו כאנומליה - אם עד עכשיו התאמנו על נתונים רגילים - אירועים שמסווגים כרגילים ופתאום משהו מאוד לא סביר קרה אז הוא אנומליה".

השיטה הזו יכולה לעבוד עד מקרים כמו עדכון תוכנה וכדומה שפתאום ישנם הרבה אירועים לא מוכרים עם הסתברות נמוכה והמערכת מוצפת בהתראות שווא - זה לא מצב טוב. מה שקורה בפועל בקוד הוא הפעלת פונקציית ה-softmax שלה יש את התכונה המאוד מיוחדת להמיר גדלים של מספרים ולשמור על היחסים ביניהם אך להפוך כל מספר להסתברות (סכומם הוא 1 וכל מספר קטן מ-1 ואי שלילי)

זה מכיוון שהשכבת פלט עוד לא המירה להסתברות מכיוון שלכל אחד מהציונים שיצא (Logits) ישנם עוד משמעויות שיכולות לעזור. לאחר מכן עושים מהלך מעניין ומוציאים את השכיח מבין כלל ההסתברויות והוא יהווה הערך סף הדינמי (מכן המשמעות שהוא דינאמי, הוא משתנה מפעם לפעם).



ניתן לראות ב-[section 6.1](#) השוואה בין שתי השיטות וקצת אינטואיציה על מדוע העדפנו להשתמש במדד של השכיח מאשר ערך קבוע. בסוף כפי שניתן לראות הבחירה על כיצד לסווג אירוע ייחודי לקחה בחשבון את השדות *Source*, *MachineName*:

```
# fill missing critical fields to prevent mapping errors during sequence generation
(all the not available with a field)
df_windows['Source'] = df_windows['Source'].fillna('Unknown_Source')
df_windows['MachineName'] = df_windows['MachineName'].fillna('Unknown_Machine')

# define the composite features that constitute a unique system event
# here we utilize the emitting source and the host machine
event_features = ['Source', 'MachineName']

# generate a continuous chronological sequence of event IDs and the corresponding
dictionary
print("Constructing categorical event vocabulary...")
event_sequence, event_vocab = build_event_vocabulary(df_windows, event_features)
vocab_size = len(event_vocab)
print(f"Vocabulary successfully constructed. Total unique system events identified:
{vocab_size}")
```

- *MachineName* הוא המזהה של תחנת קצה (כלומר ממש על איזה מחשב התבצע האירוע).
 - *Source* מציין את השירות, התכונה או רכיב מערכת ההפעלה שיצר את האירוע וביצע את הפעולה.
- הסיבות דווקא לבחירתם:

במצב בו היינו מוסיפים עוד שדות היינו מקבלים הרבה יותר אירועים (אני בטוח שאם היה את ה-*pid* של התהליך לדוגמא כמזהה נוסף היינו מקבלים מספר רב בהרבה של אירועים הרי *LAPTOP-1MKMTVPM* שהריץ *Chrome.exe* היה מוסיף לנו כבר עוד 50 אירועים לפחות למיליון - אומנם למערכות מציאותיות זה כמובן צעד הכרחי אך לפרויקט הזה התקורה הזאת רק על מנת להדגים קונספט לא שווה את זה).

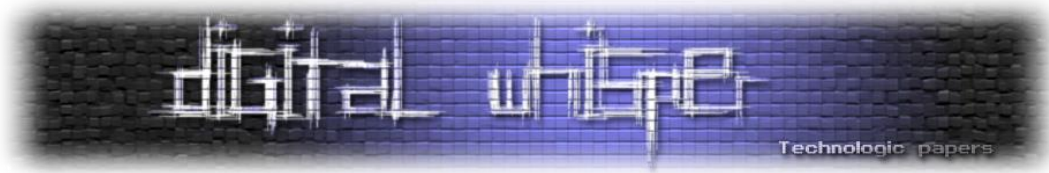
השילוב של השדות האלו יוצר מפתח עם מורכבות מספקת על מנת לזהות אנומליות (אם מנהל שיווק יתחיל להריץ דברים חריגים הקשורים ברשת זה ידליק את הנורת אזהרה). בנוסף במאגר נתונים שבו השתמשתי בפרויקט זה יוצר אוצר מילים קומפקטי אך מספיק לאימון מה שנותן דגש על חישוביות שתכף נראה מדוע היא חשובה.

כעת, בשלב הזה בפרויקט משהו הרגיש חסר רציני ממש לראות את המודל בפעולה אז פניתי לאיסוף נתונים מה *kernal* על מנת לבחון את המודל בפעולה:

נעזרתי בסיפריות וה-API הרלוונטיים שיש ל-*windows.h* להציע למשימה הזו ונתקלתי במערכת ה-ETW (*Event Tracing (for) Windows*) עם *tdh.h* (*trace data helper*). מה שיפה בסיפריות הללו שהן מאפשרות ממש להאזין לאירועים שקרו ברמת ה-*Ring 0* של ה-*kernal* אך הן פועלות ברמה 3 מה שמאפשר פיתוח נוח ואבסטרקטי.

ה-ETW יושב כצרכן בטבעת הגנה השלישית ורק מחכה לתשובות מ-*Ring-0*.

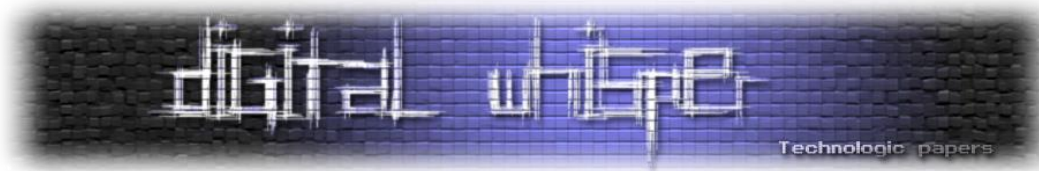
כפי שניתן לראות, הפונקציה שמקבלת משאב של אירוע בצורה יחסית פשוטה שמה במיליון גלובלי שנוצר לצורך ביצועים את הפורמט שאותו המודל מצפה לקבל:



```
//EVENT_RECORD structure https://learn.microsoft.com/en-us/windows/win32/api/evntcons/ns-evntcons-  
event_record  
//get an event - the parameter (PEVENT_RECORD) translate it into the id via the vocabulary and  
transfer it to the server  
VOID WINAPI ProcessSystemEvent(PEVENT_RECORD pEventRecord) {  
    if (pEventRecord->UserDataLength > 0 && pEventRecord->UserData != NULL) {  
        std::wstring machineName = EtwParser::GetEventProperty(pEventRecord,  
L"MachineName");  
        std::wstring source = EtwParser::GetEventProperty(pEventRecord, L"Source");  
        std::wstring compositeKey = source + L"|" + machineName;  
  
        if (eventVocab.find(compositeKey) != eventVocab.end()) {  
            int eventId = eventVocab[compositeKey];  
            if (globalTcpClient != nullptr) {  
                globalTcpClient->TransmitEvent(eventId);  
            }  
        }  
    }  
}
```

המחלקה שמשיגה את שמות ה-Property הרצויים כמו MachineName ו-Source גם היא תוכננה
באמצעות פונקציות יחסית אבסטרקטיות שמונעות ובו בזמן מקלות את הסיבוך הנוסף שדרוש על מנת
לגשת ל-kernal:

```
#include "EtwParser.hpp"  
#include <tdh.h> //https://learn.microsoft.com/en-us/windows/win32/api/tdh/ (essential for etw)  
//preprocessor directive (comment(lib, "lib_name")) that tells the compiler to leave a comment  
record in the generated object file  
//then automatically the linker reads this comment and from that links the specified library file to  
your executable.  
#pragma comment(lib, "tdh.lib")  
//The method get an event (PEVENT_RECORD) and a property in the event and return the property  
value in the event  
std::wstring EtwParser::GetEventProperty(PEVENT_RECORD pEvent, LPCWSTR propertyName) {  
    PROPERTY_DATA_DESCRIPTOR descriptor; //  
    //init the object above on the safe-way  
    ZeroMemory(&descriptor, sizeof(PROPERTY_DATA_DESCRIPTOR));  
    //set the fields of the descriptor as needed  
    descriptor.PropertyName = reinterpret_cast<ULONGLONG>(propertyName);  
    descriptor.ArrayIndex = ULONG_MAX;  
  
    DWORD bufferSize = 0;  
    // https://learn.microsoft.com/en-us/windows/win32/api/tdh/nf-tdh-  
tdhgetpropertysize get the size of the property variable  
    DWORD status = TdhGetPropertySize(pEvent, 0, NULL, 1, &descriptor, &bufferSize);  
    if (status != ERROR_SUCCESS || bufferSize == 0) {  
        return L"Unknown";  
    }  
    // https://learn.microsoft.com/en-us/windows/win32/api/tdh/nf-tdh-tdhgetproperty  
get property value(string)  
    std::vector<BYTE> buffer(bufferSize);  
    status = TdhGetProperty(pEvent, 0, NULL, 1, &descriptor, bufferSize,  
buffer.data());  
  
    if (status == ERROR_SUCCESS) {  
        return std::wstring(reinterpret_cast<wchar_t*>(buffer.data()));  
    }  
  
    return L"Unknown";  
}
```



הפונקציה מקבלת מבנה שמייצג אירוע ושם תכונה אותה אנחנו רוצים לקבל. כמובן אסור לשכוח שמדובר ב-cpp והממשק של Windows אז נאלצים ליצור כמה משתנים שלא נשתמש בהם אלא לקריאה של פונקציה ולאפס אותם וגם עוד המרות הכרחיות (למי שמתעניין הקישורים בהערות שרשמתי מציינים הכל - אני אישית ממליץ זה מנגנון מאוד יפה):

```
class EtwSession {
private:
    TRACEHANDLE hTrace;

    //https://learn.microsoft.com/en-us/windows/win32/api/evntrace/ns-evntrace-
    event_trace_logfilea
    //to see all the flag and parameter init in the c'tor of this class
    EVENT_TRACE_LOGFILE traceSession;

public:
    EtwSession() : hTrace(INVALID_PROCESSTRACE_HANDLE) {
        ZeroMemory(&traceSession, sizeof(EVENT_TRACE_LOGFILE));
        traceSession.LoggerName = const_cast<LPTSTR>(TEXT("System"));
        traceSession.ProcessTraceMode = PROCESS_TRACE_MODE_REAL_TIME |
        PROCESS_TRACE_MODE_EVENT_RECORD;
        traceSession.EventRecordCallback = ProcessSystemEvent;
        hTrace = OpenTrace(&traceSession);
        if (hTrace == INVALID_PROCESSTRACE_HANDLE) {
            std::cerr << "Initialization Error: Requires elevated administrator
            privileges." << std::endl;
        }
    }

    ~EtwSession() {
        if (hTrace != INVALID_PROCESSTRACE_HANDLE) {
            CloseTrace(hTrace);
        }
    }

    void ExecuteMonitor() {
        if (hTrace != INVALID_PROCESSTRACE_HANDLE) {
            std::cout << "Monitoring infrastructure active. Intercepting kernel
            events..." << std::endl;
            //https://learn.microsoft.com/en-us/windows/win32/api/evntrace/nf-evntrace-
            processtrace see what the function does
            ProcessTrace(&hTrace, 1, 0, 0);
        }
    }
};
```

גם מנגנון ההאזנה לנתונים עצמו רווי בטריקים והמרות, תחילה יוצרים *handler* (*htrace*) ולאחר מכן את המבנה שאותו ממלאים בפרטים אודות ההאזנה שנרצה לקבל לקבל (*traceSession*). את כל התהליכים בין המודל לבין האיסוף נתונים מהמערכות הפעלה גישרתי באמצעות הדבר המתאים ביותר פה הלא הוא *socket* שמוזמנים לראות איך הוא התבצע [בפרויקט](#) בנוסף יש שם גם דוקר שמקל על כל החיבור של הספריות השונות וכמו כן קובץ הרצה יחיד.

הפרויקט היה תהליך שלימד עוד הרבה וממליץ להסתכל עליו למעמיקים.



סיכום

במאמר הזה יצאנו "למסע כפול" העמקנו בתיאוריה המתמטית של למידת המכונה, ובסוף אפילו הגענו קצת למערכות הפעלה. פתחנו בהבנת המגבלות של רשתות Feedforward מסורתיות בניתוח נתונים סדרתיים, והמשכנו אל רשתות RNN והמכשול המתמטי המובנה שלהן - בעיית הגרדיאנט הנעלם הנובעת מהכפלת מטריצות מתמשכת. דרך פירוק אלגוריתם ה-BPTT ומבנה השערים, ראינו כיצד ארכיטקטורת LSTM פותרת באלגנטיות את כשל הזיכרון לטווח ארוך על ידי החלפת כפל שוחק בחיבור ליניארי של מסיכות מידע.

בנוסף מוצג פרויקט שבו הוצג ניסיון להתמודד עם אחד האיומים המורכבים באבטחת מידע: התוקף הפנימי. על ידי פיתוח סוכן Ring-3 ב-C++ הניגש ישירות לתשתית ה-ETW עקפנו את מנגנוני הניטור השטחיים, חילצנו תבניות התנהגותיות חסכוניות ברמת ה-Kernel והזנו אותן למודל LSTM מותאם. באמצעות קביעת רף אנומליה דינמי המבוסס על שכיחות סטטיסטית, יצרנו מערכת מודעת-הקשר (Order-Aware) שלומדת את השגרה הארגונית וחושפת איומים שהיה קשה ניתן להבחין בהם בצורה אחרת בזמן אמת.

על המחבר

אני מעיין אושרי בן 18 כרגע לומד לתואר במדעי המחשב וכעת בשנה האחרונה (בוגר תוכנית אודיסיאה), מתעניין בתחומי הלמידה חישובית, AI וסייבר ושואף לעסוק בהם בזמן הקרוב. ומאוד נהנה לקרוא את DigitalWhisper.

- [Github](#)
- מייל: maayanoshri1904@gmail.com

מקורות מידע

- [RNN](#)
- [RNN video](#)
- [BPTT](#)
- [LSTM](#)
- [LSTM](#)
- [LSTM](#)