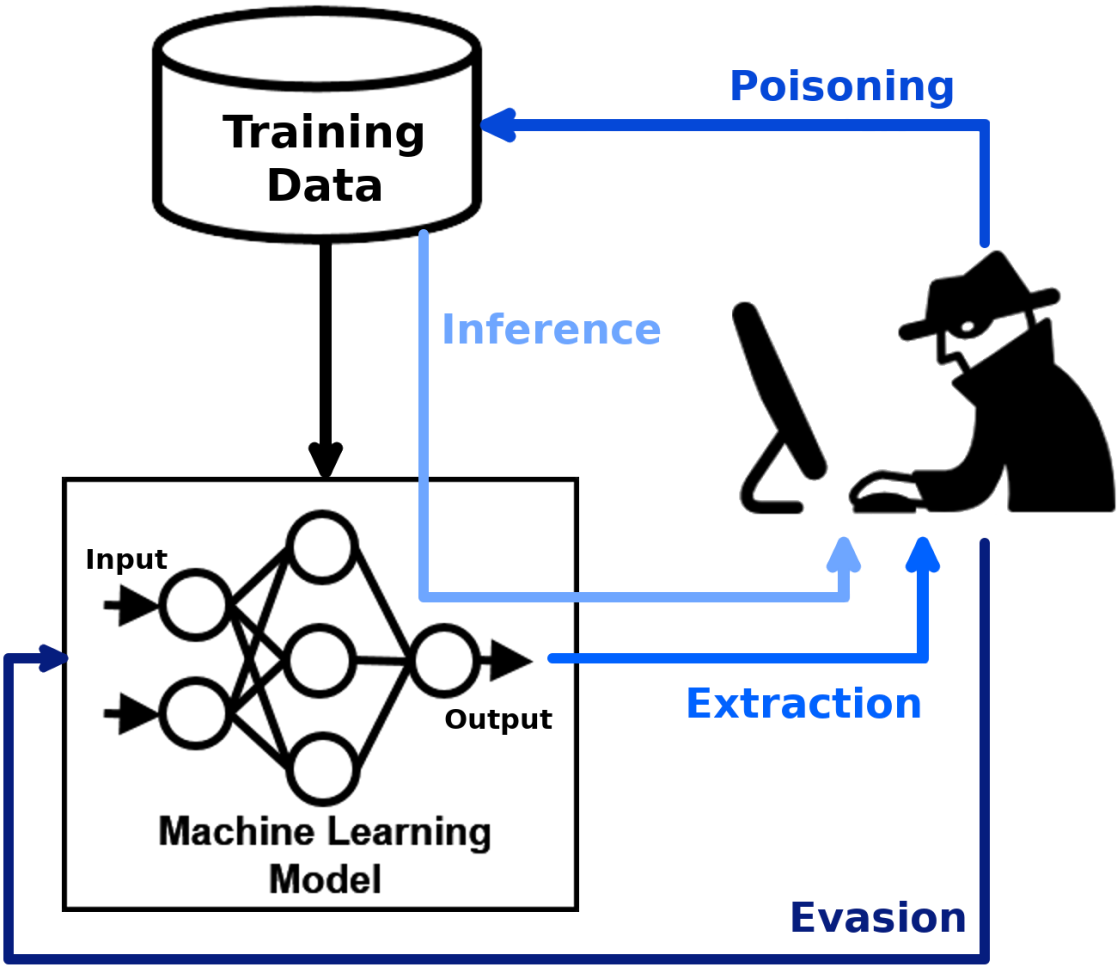
עבודה מסכמת בקורס מבוא לחומרה בטוחה-בניות מתקפות

SCA-DL : Side Channel Attacks Deep Learning



מוגש ע"י טום אשקורי 205702608

ג'סיקה ג'אנוס 327083184

תוכן עינייינים

מבוא

בעבודה זו אנו נבחן וננתח את היכולת לתקוף מערכת קריפטוגרפית שעושה שימוש בפרמטר סודי ,למשל מפתח סודי, לצורך חישובים שונים כמו הצפנה או חישוב ביניים כמו חישוב של s-box.

בניגוד לדרכי פעולה קלאסיות לחילוץ אותו פרמטר, אנו נממש ונעשה שימוש בכלים של למידת מכונה ובפרט נשתמש בכלים של למידה עמוקה שהם הרבה יותר "חזקים" כדי ללמוד את הפרמטר הסודי של המערכת תוך התמודדות עם אמצעי הגנה שהולכים ומשתפרים.

חשוב לציין כי במהלך הדוח אנו מסתמכים על העובדה כי קיימת לנו גישה אל המערכת ואנחנו יכולים להפיק מידע שימושי ממנה כרי זליגות ערוץ צד.

קיימים שיטות שונות לחלץ את "המידע" אבל לא נתייחס לכך לצורך הניתוח רק נוסיף שהדגימות נלקחו באמצעות probes מתוך רכיב 5 רכיבים עם רמת הגנה הולכת ומתחזקת. הדגימות נעשו במעבדה של איתמר שנמצאת בבר אילן בקמפוס הנדסה.

למידת מכונה

למידת מכונה מאפשרת למחשב ללמוד מתוך דוגמאות וכך לבנות אלגוריתם שמטרת הלמידה שלו יכולה להיות מידול, חיזוי או גילוי של עובדות או מידע.

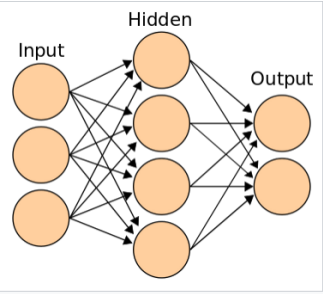
המטרה הבסיסית של למידת מכונה היא היכולת להכליל מתוך הניסיון, כלומר היכולת לבצע חיזוי (או סיווג וכו') באופן מדויק ככל האפשר על מידע שעדיין לא נצפה, על בסיס צבירת ניסיון ממידע קיים. הנחת היסוד פה היא שככל שיש יותר מידע אז המודל שנבנה יהיה יותר מדוייק.

בדרך כלל דוגמאות המשמשות ללימוד נוצרות מאיזושהי התפלגות לא ידועה והמכונה הלומדת בונה מודל מוכלל מאותו מרחב של ההתפלגות, מה שמאפשר ביצוע מדויק באופן מספק על דוגמאות חדשות.

בעצם ערכי ה target (המטרה) נגזרים מה data (המידע) באופן של המודל שמצאנו.

למידת מכונה היא הבסיס ללמידה עמוקה.

למידה עמוקה היא חלק ממשפחה מצומצמת יותר של שיטות למידת מכונה המבוססות על רשתות נוירונים (עצבית) מלאכותיות עם למידת ייצוג. הלמידה יכולה להיות בפיקוח, בפיקוח למחצה או ללא פיקוח.



תרשים המדגים את אופן הפעולה של רשת עצבית מלאכותית.

שם התואר "עמוק" בלמידה עמוקה מתייחס לשימוש במספר רבדים (שכבות) ברשת כדי לחלץ בהדרגה תכונות ברמה גבוהה יותר מהקלט הגולמי. לדוגמה, בעיבוד תמונה, שכבות נמוכות עשויות לזהות קצוות, בעוד שכבות גבוהות יותר עשויות לזהות את המושגים הרלוונטיים לאדם כמו ספרות או אותיות או פרצופים.

למידה עמוקה היא וריאציה מודרנית העוסקת במספר בלתי מוגבל של שכבות בגודל מוגבל, המאפשרת יישום מעשי ויישום מיטבי, תוך שמירה על אוניברסליות תיאורטית בתנאים מתונים. בלמידה עמוקה מותר לשכבות להיות הטרוגניות ולסטות באופן נרחב ממודלים קונקנציוניסטים בעלי מידע ביולוגי, למען יעילות, יכולת אימון והבנה.

תהליך למידה עמוקה יכול ללמוד אילו תכונות למקם בצורה אופטימלית באיזו שכבה בעצמו אבל זה לא מבטל את הצורך בכוונון ידני. לדוגמה, מספר משתנה של שכבות וגדלי שכבות יכולים לספק דרגות שונות של הפשטה.

רוב המודלים המודרניים של למידה עמוקה מבוססים על רשתות נוירונים מלאכותיות(ANN), במיוחד על רשתות נוירונים קונבולוציוניות (CNN). אם כי הם יכולים לכלול גם משתנים סמויים המאורגנים בשכבה במודלים מחוללים עמוקים כמו הצמתים ברשתות אמונות עמוקות (deep belief networks).

כפי שציינו למעלה, הלמידה מסווגת לעיתים לפי 2 קטגוריות - בפיקוח או ללא פיקוח.

<u>אלגוריתמים מפוקחים (supervised)</u>: כאשר יש לנו מאגר של נתונים (data) ותשובות (target value) שהושגו בצורה כזו או אחרת (מדידות, עבודה של מומחים, וכו') ואנו רוצים שהמחשב ינסה לחקות את התשובות.

האלגוריתמים האלה דורשים שליטה על תהליך התחקור, סיפוק הן את הקלט והן את הפלט הרצוי, ובנוסף לספק משוב על הדיוק של התחזיות אחרי שנעשתה הרצה של אלגוריתם מספר פעמים.

צריך לקבוע אילו משתנים, או תכונות צריך לנתח ובאיזה מודל צריך להשתמש כדי לבנות תחזיות מדוייקות, לאחר השלמת תהליך הלמידה של האלגוריתם ובחינת טיב האלגוריתם, ניתן ליישם את מה שנלמד על הנתונים החדשים.

- Imbalanced data

  ברוב המקרים אין שיוויון בגודל של קבוצות ערך המטרה ולכן זה יכול לגרום לbias של המודל כלפי הערך עם כמות הגדולה יותר של דגימות.

  אצלנו כאשר נשתמש במודל HW (נדבר בהמשך) נתקל בבעיה זאת אחרי הכל הפיזור של הביטים עוקב אחרי התפלגות מסויימת ורוב הערכים בעלי 4 ביט בלבד.

  ישנם דרכים רבות ומגוונות להתמודד עם הבעיה הזאת, אחת הנפוצות בה עשינו שימוש היא אלגוריתם SMOTE אשר משכפל בצורה חכמה את המידע של קבוצות המיעוט תוך שמירה על "קירבה" בתוך אותה קבוצה.

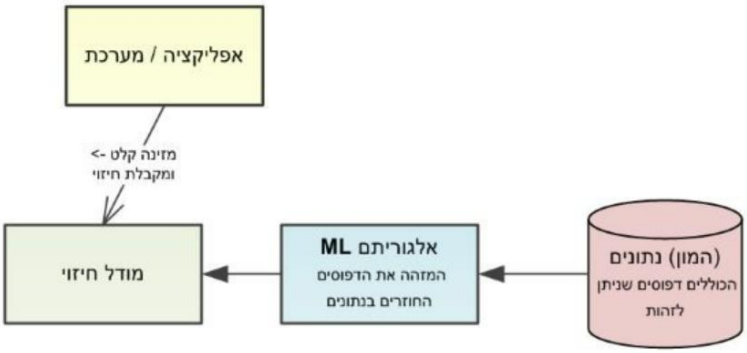
<u>אלגוריתמים לא מפוקחים (unsupervised)</u>: יש לנו סט של נתונים, אך ללא תשובות מהם אנו מנסים להפיק תובנות. אלגוריתמים אלה לא צריכים לעבור תהליך למידה מבוקר, הם מאתרים דפוסים נסתרים בנתוני הקלט.

גם כאן, אנו בוחרים אלגוריתם שיבנה לנו מודל, לו נזין נתונים ונקבל הצעה לחיזוי. ההערכה של הצלחת המודל תהיה תהליך קצת שונה- אולי תהליך ידני, אולי הרצה של סימולטור כזה או אחר שינסה לקבוע את הצלחת המודל.

ישנם אלגורתמים כאלו אשר משתמשים בגישה איטרטיבית הקרויה למידה עמוקה (deep learning) לסקירת נתונים וכך מגיעים למסקנות.

שיטות אלה נקראים גם neural networks שמשמשים למשימות עיבוד מורכבות יותר, כולל עיבוד תמונות, ניתוח דיבור-טקסט וכו'. הם פועלים על ידי סריקה של מיליוני דוגמאות של נתוני דגימה ומזהים באופן אוטומטי קשרים בין משתנים רבים.

לאחר תהליך למידה זה, האלגוריתם יכול להשתמש באוסף של קשרים שיצר לעצמו כדי לתת תשובה לגבי נתונים חדשים.



נסביר על **<u>רשתות נוירונים</u>** (ANN):

ANN מבוסס על אוסף של יחידות מחוברות או צמתים הנקראים נוירונים מלאכותיים.

רשת נוירונים (מלאכותית) מדגימה באופן רופף את הנוירונים במוח ביולוגי. כל חיבור, כמו הסינפסות במוח ביולוגי, יכול להעביר אות לנוירונים אחרים. נוירון מלאכותי מקבל אות ואז מעבד אותו ויכול לאותת לנוירונים המחוברים אליו, החיבורים נקראים קצוות. לנוירונים ולקצוות יש בדרך כלל משקל שמסתגל ככל שהלמידה מתקדמת, המשקל מגדיל או מקטין את עוצמת האות בחיבור.

לנוירונים עשוי להיות סף כזה שאות נשלח רק אם האות המצטבר חוצה את הסף הזה. בדרך כלל, נוירונים מצטברים לשכבות.

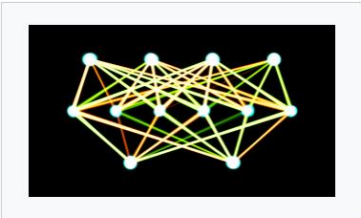
רשת נוירונים מורכבת ממספר רב של שכבות (Hidden layer) , כך שכל שכבה מבצעת חישוב פשוט יחסית. לכל שכבה מספר כניסות ויציאה אחת שערכה הוא פונקציה לא לינראית כלשהי של הכניסות. הרשת יוצרת גרף מכוון ומשוקלל.

שכבות שונות עשויות לבצע טרנספורמציות שונות על הכניסות שלהן. אותות עוברים מהשכבה הראשונה (שכבת הקלט), לשכבה האחרונה (שכבת הפלט), אולי לאחר חציית השכבות מספר פעמים. בין שתי שכבות, דפוסי חיבור מרובים אפשריים.

הם יכולים להיות 'מחוברים באופן מלא' (fully connected), כאשר כל נוירון בשכבה אחת מתחבר לכל נוירון בשכבה הבאה.

הם יכולים להיות איגום (pooling), כאשר קבוצת נוירונים בשכבה אחת מתחברת לנוירון בודד בשכבה הבאה, ובכך מפחיתה את מספר הנוירונים בשכבה זו. נוירונים עם קשרים כאלה בלבד יוצרים גרף א-ציקלי מכוון והם ידועים כ- feedforward networks. לחילופין, רשתות המאפשרות חיבורים בין נוירונים בשכבות זהות או קודמות ידועות כ- recurrent networks.

כדי למצוא את הפלט של הנוירון, ראשית עלינו לקחת את הסכום המשוקלל של כל הכניסות, משוקלל לפי משקלי החיבורים מהכניסות לנוירון. אנו מוסיפים ערך הטיה לסכום זה. סכום משוקלל זה נקרא לפעמים activation. הסכום המשוקלל הזה מועבר לאחר מכן דרך פונקציית activation (בדרך כלל לא ליניארית) כדי להפיק את הפלט.



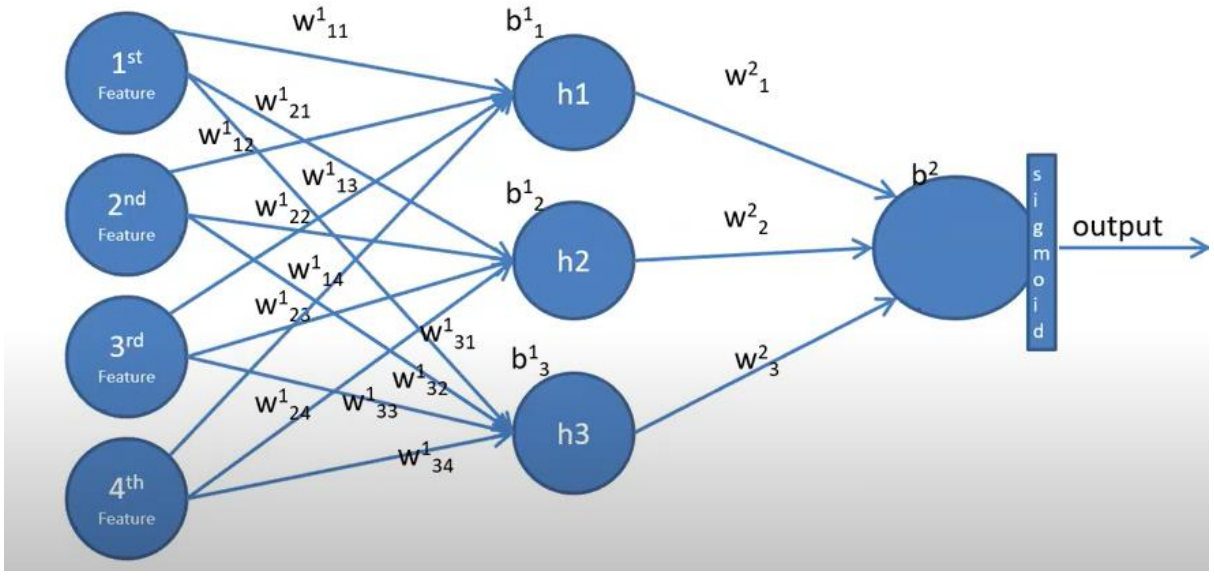
A single-layer feedforward artificial neural network with 4 inputs, 6 hidden and 2 outputs. Given position state and direction outputs wheel based control values.

למידה היא התאמה של הרשת להתמודדות טובה יותר עם משימה על ידי התחשבות בתצפיות לדוגמה. הלמידה כוללת התאמת המשקלים של הרשת כדי לשפר את הדיוק של התוצאה, זה נעשה על ידי מזעור השגיאות שנצפו. הלמידה הושלמה כאשר בחינת תצפיות נוספות אינה מפחיתה באופן מועיל את שיעור השגיאות. גם לאחר למידה, שיעור השגיאות בדרך כלל אינו מגיע ל-0. אם לאחר הלמידה, שיעור השגיאות גבוה מדי, יש למדל מחדש את הרשת בדרך כלל.

באופן מעשי זה נעשה על ידי הגדרת cost function המוערכת מעת לעת במהלך הלמידה. כל עוד התפוקה שלו ממשיכה לרדת, הלמידה ממשיכה. cost function מוגדרת לעתים קרובות כסטטיסטיקה שניתן רק להעריך את ערכו. הפלטים הם מספרים, כך שכאשר השגיאה נמוכה, ההבדל בין הפלט לתשובה הנכונה קטן. למידה מנסה לצמצם את סך ההבדלים בין התצפיות. ניתן לראות את רוב מודלי הלמידה כיישום פשוט של תיאוריית האופטימיזציה ואומדן סטטיסטי.

קצב הלמידה מגדיר את גודל הצעדים המתקנים שהמודל נוקט כדי להתאים לטעויות בכל תצפית. קצב למידה גבוה מקצר את זמן האימון, אך עם דיוק אולטימטיבי נמוך יותר, בעוד שקצב למידה נמוך יותר לוקח זמן רב יותר, אך עם פוטנציאל לדיוק רב יותר. אופטימיזציות כגון Quickprop מכוונות בעיקר להאיץ את מזעור השגיאות, בעוד שיפורים אחרים מנסים בעיקר להגביר את האמינות.

היפרפרמטר הוא פרמטר קבוע שערכו נקבע לפני תחילת תהליך הלמידה. ערכי הפרמטרים נגזרים באמצעות למידה. דוגמאות להיפרפרמטרים כוללות קצב למידה, מספר השכבות הנסתרות וגודל batch. הערכים של היפרפרמטרים מסוימים יכולים להיות תלויים בערכים של היפרפרמטרים אחרים. לדוגמה, הגודל של כמה שכבות יכול להיות תלוי במספר הכולל של שכבות.



בתמונה אפשר לראות רשת ניורונים עם שכבה אחת עם שלוש ניורונים וארבע כניסות ויציאה אחת.

נדבר עכשיו על סוגי רשתות :

Recurrent neural network:

אלו רשתות שבהן קשרים בין צמתים יוצרים גרף מכוון או בלתי מכוון לאורך רצף זמני. זה מאפשר לה להפגין התנהגות דינמית זמנית. נגזר מ- feedforward neural networks גם RNN יכולים להשתמש במצב הפנימי שלהם (זיכרון) כדי לעבד רצפים באורך משתנה של כניסות, זה הופך אותם למתאימים למשימות כגון זיהוי כתב יד מחובר או זיהוי דיבור. רשתות אלה יכולות להריץ תוכניות שרירותיות לעיבוד רצפים שרירותיים של קלט.

המונח RNN מתייחס למחלקה של רשתות עם infinite impulse response, בעוד ש- CNN (convolutional neural network) מתייחס ל finite impulse response. שני מחלקות הרשתות מפגינות התנהגות דינמית זמנית.

finite impulse recurrent network היא גרף א-מחזורי מכוון שניתן לפרוש ולהחליף ב- feedforward neural network, בעוד ש- infinite impulse recurrent network היא גרף מחזורי מכוון שלא ניתן לפרישה.

לשני סוגי הרשתות לעיל יכולות להיות מצבים מאוחסנים נוספים, והאחסון יכול להיות בשליטה ישירה של רשת הניורונים. האחסון יכול גם להיות מוחלף על ידי רשת אחרת או גרף אחר אם זה כולל עיכובים בזמן או יש לולאות משוב.

מצבים מבוקרים כאלה מכונים כ-Gated state או Gated memory, והם חלק מ- long short-term memory networks (LSTM) ו- gated recurrent units (GRU). זה נקרא גם Feedback Neural Network (FNN).

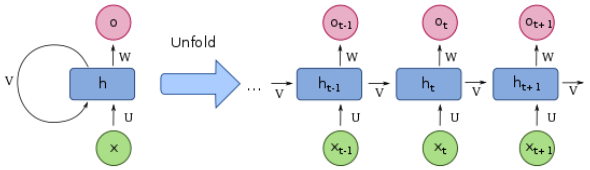
RNN סובלות מזיכרון לטווח קצר. אם רצף ארוך מספיק, הם יתקשו לשאת מידע משלבי זמן מוקדמים יותר למאוחרים יותר. אז אם אתה מנסה לעבד פסקה של טקסט כדי לבצע תחזיות, RNN's עשויים להשמיט מידע חשוב מההתחלה.

במהלך back propagation, RNN סובלות מבעיית הגרדיאנט(שיפוע) הנעלם. שיפועים הם ערכים המשמשים לעדכון משקלי הרשתות. בעיית השיפוע הנעלם היא כאשר השיפוע מתכווץ כאשר הוא מתפשט בחזרה לאורך זמן. אם ערך השיפוע הופך לקטן במיוחד, הוא לא תורם יותר מדי למידה.

RNN משתמש בזכרון רציף (sequential memory), זכרון רציף הוא מנגנון שמקל על זיהוי דפוסים סדרתיים.

3 הבניות המוכרות של RNN הם :

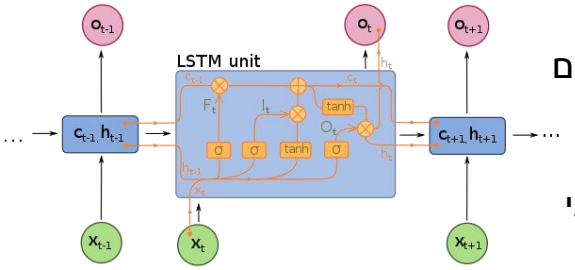
-Fully recurrent neural networks (FRNN)



מחברת את הפלטים של כל הנוירונים לכניסות של כל הנוירונים. זוהי טופולוגיית הרשת הכללית ביותר מכיוון שניתן לייצג את כל הטופולוגיות האחרות על ידי קביעת משקלי חיבור לאפס כדי לדמות את היעדר הקשרים בין אותם נוירונים.

האיור עשוי להטעות מכיוון שטופולוגיות מעשיות של רשתות מאורגנות לעתים קרובות ב"שכבות" והציור נותן את המראה הזה. עם זאת, מה שנראה כשכבות הם למעשה שלבים שונים בזמן של אותה רשת.

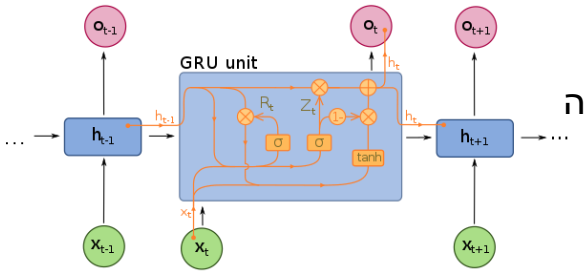
-Long short-term memory (LSTM)



זו מערכת המונעת את בעיית ה- vanishing gradient. LSTM מתוגבר בדרך כלל על ידי שערים חוזרים הנקראים "forget gates" ומונע שגיאות המופצות לאחור מלהיעלם או להתפוצץ. במקום זאת, שגיאות יכולות לזרום לאחור דרך מספר בלתי מוגבל של שכבות וירטואליות הנפרשות במרחב.

כלומר, LSTM יכול ללמוד משימות הדורשות זיכרונות של אירועים שקרו אלפי או אפילו מיליוני צעדי זמן בדידים קודם לכן. LSTM עובד גם בהינתן עיכובים ארוכים בין אירועים משמעותיים ויכול להתמודד עם אותות המשלבים רכיבים בתדר נמוך וגבוה.

-Gated recurrent units (GRU)



הוא משומש בצורה המלאה ובמספר גרסאות מפושטות. הביצועים שלהם על מודלים של מוזיקה פוליפונית ומודל אותות דיבור נמצאו דומים ל- LSTM. יש להם פחות פרמטרים מאשר LSTM, מכיוון שחסר להם שער פלט.

ל-GRU יש פחות פעולות. לכן, הם קצת יותר מהירים לאימון מאשר LSTM. אין מנצח ברור מי מהם טוב יותר. חוקרים ומהנדסים מנסים בדרך כלל את שניהם כדי לקבוע איזה מהם עובד טוב יותר עבור המקרה הספציפי שלהם.

Convolutional neural network:

הן ידועות גם בשם Shift Invariant או Space Invariant Artificial Neural Networks (SIANN), המבוססות על ארכיטקטורת המשקל המשותף של גרעיני הקונבולציה או המסננים המחליקים לאורך תכונות הקלט ומספקות תגובות שוות תרגום הידועות כ- feature maps.

CNNs הם גרסאות מוסדרות של multilayer perceptrons.

multilayer perceptrons מתכוונות בדרך כלל לרשתות מחוברות לחלוטין, כלומר, כל נוירון בשכבה אחת מחובר לכל הנוירונים בשכבה הבאה. ה"קישוריות המלאה" של רשתות אלו גורמת להן לנטייה ל- overfitting. דרכים אופייניות לרגולציה, או מניעת overfitting כוללות: ענישת פרמטרים במהלך האימון (כגון ירידה במשקל) או קישוריות חיתוך (חיבורים שדילגו, נשירה וכו').

רשתות CNN נוקטות בגישה שונה כלפי רגוליזציה: הם מנצלים את הדפוס ההיררכי בנתונים ומרכיבים דפוסים בעלי מורכבות הולכת וגוברת באמצעות דפוסים קטנים ופשוטים יותר המוטבעים במסננים שלהם. לכן, בקנה מידה של קישוריות ומורכבות, רשתות CNN נמצאים בקצה התחתון.

CNNs משתמשים במעט יחסית עיבוד מקדים בהשוואה לאלגוריתמים אחרים. משמעות הדבר היא שהרשת לומדת לייעל את המסננים (או ה-kernels) באמצעות למידה אוטומטית, בעוד שבאלגוריתמים מסורתיים מסננים אלו מהונדסים ידנית. עצמאות זו ממידע קודם והתערבות אנושית בחילוץ תכונות היא יתרון מרכזי.

משמש בעיקר לעיבוד תמונות ולכן לא נשתמש בזה בפרוייקט שלנו.

Deep neural network:

רשת נוירונים עמוקה (DNN) היא רשת נוירונים מלאכותית (ANN) עם שכבות מרובות בין שכבות הקלט והפלט. ישנם סוגים שונים של רשתות נוירונים אך הם תמיד מורכבים מאותם מרכיבים: נוירונים, סינפסות, משקלים, הטיות ותפקודים. רכיבים אלו פועלים בדומה למוח האנושי וניתן לאמן אותם כמו כל אלגוריתם ML אחר.

DNNs יכולים למדל קשרים לא ליניאריים מורכבים. ארכיטקטורות DNN מייצרות מודלים קומפוזיציוניים שבהם האובייקט מתבטא כקומפוזיציה מרובדת של פרימיטיבים, השכבות הנוספות מאפשרות הרכבה של תכונות משכבות נמוכות יותר, ועשויות ליצור מודלים של נתונים מורכבים עם פחות יחידות מאשר רשת רדודה בעלת ביצועים דומים.

ארכיטקטורות עמוקות כוללות גרסאות רבות של כמה גישות בסיסיות. כל ארכיטקטורה מצאה הצלחה בתחומים ספציפיים. לא תמיד ניתן להשוות את הביצועים של ארכיטקטורות מרובות, אלא אם כן הם הוערכו על אותם מערכי נתונים.

DNNs הם בדרך כלל feedforward networks שבהן הנתונים זורמים משכבת הקלט לשכבת הפלט מבלי לחזור אחורה. בתחילה, ה-DNN יוצר מפה של נוירונים וירטואליים ומקצה ערכים מספריים אקראיים או משקולות, לקשרים ביניהם.
המשקולות והכניסות מוכפלות ומחזירות פלט בין 0 ל-1. אם הרשת לא זיהתה במדויק תבנית מסוימת, האלגוריתם יתאים את המשקולות. כך האלגוריתם יכול להפוך פרמטרים מסוימים למשפיעים יותר, עד שהוא יקבע את המניפולציה המתמטית הנכונה לעיבוד מלא של הנתונים.

פונקציות אקטיבציה מוכרות:

Name	Plot	Equation	Derivative (with respect to x)	Range
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$
Softsign [7][8]		$f(x) = \frac{x}{1+\lvert x\rvert}$	$f'(x) = \frac{1}{(1+\lvert x\rvert)^2}$	$(-1, 1)$
Inverse square root unit (ISRU)[9]		$f(x) = \frac{x}{\sqrt{1+\alpha x^2}}$	$f'(x) = \left(\frac{1}{\sqrt{1+\alpha x^2}}\right)^3$	$\left(-\frac{1}{\sqrt{\alpha}}, \frac{1}{\sqrt{\alpha}}\right)$
Rectified linear unit (ReLU)[10]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$
Leaky rectified linear unit (Leaky ReLU)[11]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Parameteric rectified linear unit (PReLU)[12]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Randomized leaky rectified linear unit (RReLU)[13]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ [1]	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$

המשומשות ביותר ביניהן הן : ReLU , Leaky ReLU , ELU .

עד כה דיברנו על למידת מכונה באופן כללי ועכשיו קצת ננבור בסוג התקיפה ומה אנחנו בפועל עושים:

חומרה בטוחה

SCA

SIDE CHANNEL ATTACKS

זו שיטה להתקפות על רכיבי חומרה באמצעות כלים שמנצלים תכונות לא רצויות של אופי הרכיב כמו זרמי זליגה, פליטות אלקטרומגנטיות ועוד.

התקיפות יכולות להיות על שבבים כאלה או אחרים למשל כאלה שמחשבים הצפנת או השבב הקטן שנמצא על כרטיסי האשראי של כולנו(מפחיד), כמו תקיפת AES.

חלק מתנאי הקדם לחלק מהתקיפות מניחות ידע שיש לתוקף על אופי החישוב או בפועל איך הוא נראה ואפילו הרכיב יכול להיות בפועל בידי התוקף וככה הוא יכול ללמוד על אופיין הזליגות וכך לחלץ את הפרמטר הסודי למשל המפתח בו הרכיב משתמש כדי להצפין.

באופן כללי כל המתקפות מנצלות את העובדה שמידע נזלג בערוץ "פיזי" כתוצאה מהאופרציה הקריפטוגרפית.

נדבר בקצרה גם על כלים שנועדו להתמודד עם התקפות כאלה.

Counter Measures

ישנם שני גישות:

1. להפחית או להעלים את זליגת האינפורמציה

למשל על ידי הקטנת ערך הזליגה על ידי חיבור קבל או נגד לקווי האספקה

2. להפחית או להסיר את הקשר, יש לציין הסטטיסטי, בין הזליגות לאינפורמציה

למשל על ידי מיסוך או הזזה בזמן של ערכי הזליגה או למשל באמצעות הוספת נגד משתנה שמקבל ערכים רנדומים ובכך בכל חישוב ערכי הזליגה יהיו שונים גם אם החישוב הוא אותו דבר

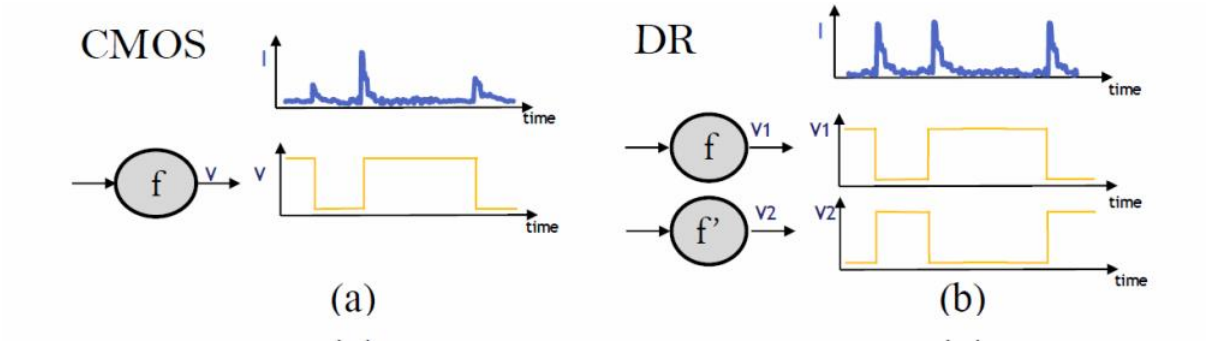
Amplitude Balancing

בחלק מהרכיבים ממומשת הגנה שנקראת DUAL RAIL

כפי שניתן לראות בתמונה עם הCMOS שעבור חישוב f מסויימת נקבל אופיין זליגה מסויים אנחנו רוצים למסך אותו ולכן נחשב את הפונקציה המשלימה.

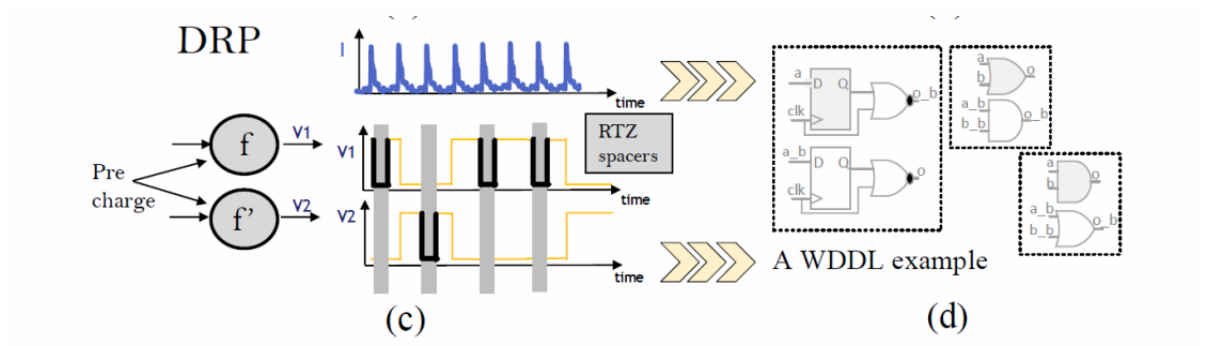
איך זה עוזר?

בזמן חישוב הערך הסודי יתבצע במקביל חישוב של f` אשר יגרום לזליגה לשנות את אופייה ובכך להוסיף רעש אלגו' כדי למסך את ה"מידע" שזולג מחישוב עם מפתח מסויים.



שיטה נוספת היא Dual Rail Precharge –

שרוצה למקסם את הרעש האלגו' ע"י גרימה לכך שבכל מחזור חישוב תמיד יהיה שינוי ערך במוצא ובדיוק אחד ככה בפועל הספק הזליגה של המערכת הופכת יוניפורמית ללא תלות במפתח



כמובן כמו בכל פרוייקט הנדסי יש להתחשב בעלות השטח של המימוש וגם בצריכת אנרגיה מוגברת ולכן לא לכל הפרוייקטים רמות ההגנה הללו מתאימות בנוסף בעולם האמיתי אין סימטריה מוחלטת הן מבחינת השערים ואפיון הזליגה שלהם וגם אם נרד לרמת הlayout ייתכן פיזור לא אחיד של הרכיבים ולכן עדיין נוכל לדלות מידע מהחישוב. כמובן נדרש להרבה יותר מדידות .

TAvsCPA

CPA

Correlation Power Analysis

בעצם זוהי התקפה שמאפשרת לנו לחלץ את הערך הסודי באמצעות ניתוח סטטיסטי של זרמי הזליגה של הרכיב. למתקפה זו כמה שלבים אבל בבסיסה עומדת ההנחה שזרמי הזליגה של הרכיב מתנהגים בצורה מסויימת ,כלומר לפי מודל power consumption.
נשווה בין כמה מודלים:

direct.1:

בבסיסו עומדת ההנחה שיש יחס של נגיד 1 ל1 בין הערך שחושב לבין זרמי הזליגה כלומר עבור ערך ciper מאורך 8 ביט כאשר הפלט של החישוב הקריפטוגרפי הוא 255 זאת אומרת שעבור הערך הזה תהיה הזליגה הגובה ביותר.

hamming weight.2:

המודל מניח שההספק מקושר למספר ה1 במוצא ולכן במודל רכיב של asics כאשר החישוב מתבצע במקביל דרך המון רכיבים סביר להניח שצריכת האנרגיה תלויה בכמות האחדות שבערך המוצא.

hamming distance.3:

בעצם המודל מניח שמדובר ברכיב שמשתמש בflip flop ולכן מהבנה שלהם גם אם המעבר הוא מ0 ל1 וגם אם 1 ל0 תמיד אחד הנורים יהפוך ל1 ויצרוך אנרגיה מהספק.

כל מודל מניח שבנקודת זמן מסויימת של החישוב ערך הביניים הוא x ולכן הצריכה היא $f(x)$.

בהמשך התוקף שומר טרייסים על חישוב שהמערכת מבצעת על הודעות שונות, בוחרים חלק קטן מתוך המפתח שאותו תוקפים ומחשבים קורלציה בין כל אפשרות של המפתח לבין ערכי הזליגה שמידלנו לפי הנחת המודל שלנו ובעצם בוחרים את האפשרות שערך pearson correlation שלה הוא הגבוה ביותר

לסיכום:

1. בוחרים מודל צריכה
2. מקליטים זרמי זליגה
3. מנתחים סטטיסטית את הזרמים אל מול ערכי ההיפותזה של המפתח
4. בוחרים את זה עם הערך הקורלטיבי הגבוה ביותר

עד כה דיברנו על CPA עכשיו נעבור לדבר על TA

<u>TA</u>

מתקפות אפיון הן סוג של SCA שמניח כי לתוקף כוח רב, שהוא מחזיק בהעתק של המערכת, ולכן יכול לעשות כרצונו ולהכניס איזה ערכים שיבחר ויוכל ליצור המון דוגמאות של זרמי זליגה ובעצם לאפיין כך את המערכת.

גם היא מחולקת לכמה שלבים:

1. אסיפה של זליגות ערוץ צד
2. אפיון הרכיב
3. "תקיפה" אסיפה של זליגות מרכיב קורבן
4. שימוש באיפיון כדי לחלץ את הפרמטר הסודי

שלב האיפיון דומה מאוד לאופן שבו פועלים בCPA אבל ההבדל העיקרי הוא יכולת השליטה ברכיב עצמו ויכולת חזקה מאוד של איסוף מידע שבעתיד יובילו לכך שנצטרך לאסוף מעט מאוד זליגות מרכיב קורבן כדי לחלץ את המפתח בקלות

חשוב להזכיר כלי חשוב שבו עושים שימוש בכדי לחלץ אינפורמציה מתוך הזליגות.

SNR

Signal to Noise ratio

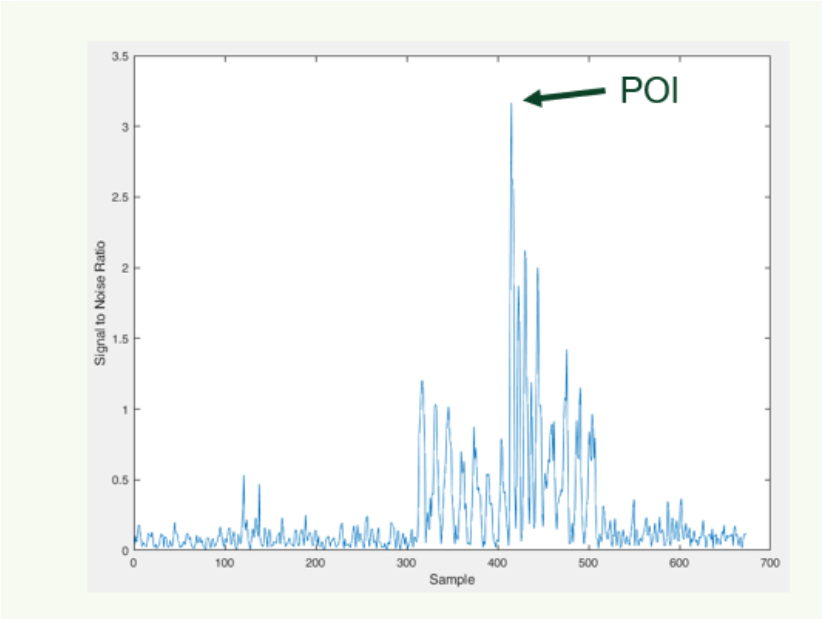
$$SNR = \frac{Var(E(Tr))}{E(Var(Tr))} = \frac{info}{noise}$$

חישוב או מאפיין שמחשב את היחס בין ה"מידע" הנזלג מתוך הדגימות אל מול ה"רעש"

עבור ערכים נמוכים או שהאינפורמציה נמוכה או הרעש גבוה, כך או כך היכולת לחלץ את הפרמטר הסודי נמוכה יותר וכך גם בכיוון השני.

מעבר לכך לsnr התפלגות בזמן ולכן כמו בתמונה למטה יש איזורים או נק' מעניינות בזמן שם מתחבא מידע רב, כמובן כתלות במודל הנקודות הללו יכולות לזוז או באופן כללי הsnr יהיה נמוך יותר, יכול להיות שנצטרך יותר דגימות וכו'

בתמונה למטה רואים חישוב snr מתוך מטלה 5 שבא חישבנו snr בהתבסס על מודל HD



וכאשר חילצנו את המפתח היה ניתן לראות בברור כי סביב הנקודה הזאת הקורלציה למפתח הנכון עולה משמעותית ונוכל להפריד בין המפתח הנכון להיפוטזות הלא נכונות.

עד כה דיברנו קצת על למידת מכונה בנפרד ועל נושאים בתוך חומרה בטוחה

נעבור לדבר על השילוב בינם:

SCA-DL

Side channel attacks – Deep Learning

הרעיון המרכזי מאחורי הגישה הוא שימוש בכלים של למידת מכונה ולמידה עמוקה על מנת לנבא את המפתח ובמילים אחרות לבצע TA ובמקום ניתוחים סטטיסטיים מלמדים מודל ונותנים לו להסיק את הדפוסים וההקשר הקורלטיבים בין הזליגות לערכי המפתח וחישוב המתבצע

כמובן גם כאן יש חשיבות לבחירת מודל הצריכה ואסיפה של מספיק דוגמאות ובניגוד מוחלט לשיטות תקיפה, נאמר קלאסיות, כאן יש לנו המון "חופש" בבחירת המודל, בחירת הפרמטרים שלו, האופי שבו אנחנו מעבדים את המידע לפי אימון המודל ועוד הרבה.

החופש הרב נותן לנו המון כוח מצד אחד ואילו מצד שני קשה מאוד למצוא את הפרמטרים המתאים ביותר וגישה נפוצה בתעשייה היא ליצור גריד של פרמטרים ולחשב את הביצועים של המודל על כל צירוף שלהם ולבחור את אלו הטובים ביותר.

נשמע נחמד אבל בפועל אנחנו צריכים לטעון לזיכרון כמות רבה של מידע אימון וגם מורכבות המודל גוזרת כוח חישוב ועוד זיכרון כך שבמחשבינו הפרטיים RAM GB8 והמעבדים קצת מקשים את העבודה.

לדוגמא נתקלנו בריצות שכל מחזור חישוב לוקח בין שעה וחצי ל3 כמובן אפשר להוריד פרמטרים או למשל כדי להקטין את כמות המשקולות במקום לעבוד עם מודל direct אפשר לעבוד עם מודל HW.

ביצוע TA-DL

נחזור לרגע על השלבים שמרכיבים את התקיפה

1. אסיפה של זליגות ערוץ צד
2. אפיון הרכיב
3. "תקיפה" אסיפה של זליגות מרכיב קורבן
4. שימוש באיפון כדי לחלץ את הפרמטר הסודי

השוני בין הגישות הוא בעיקר בביצוע האפיון, כלומר במקום להשתמש בסטטיסטיקה וקירובים גאוסיים נותנים למודל ללמוד את אופי הזליגות דבר אשר חזק יותר כי אנחנו לא מניחים שום התפלגות ושום כלום על המפתח מראש.

*דיברנו בעבר על POI וCNN אחת התכונות הבולטות של רשתות cnn זה שהם מבצעים מאין חלון על האינפוטים וכך יכולים לגלות בעצמם את תהליך ה feature selection שבלעדיו מודלים פשוטים של למידת מכונה פשוט לא יעבדו(כי המידע שנכנס לאימון אצלהם יכול להיות עם snr נמוך)

<u>נחזור לאפיון:</u>

לוקחים את הטרייסים שמדדנו אפשר ומומלץ לבצע איזשהו עיבוד מקדים כדי להוריד את הרעש או לבצע הורדת מימד(pca) כדי להקל על תהליך הלמידה של המודל אבל לשמור באופן כללי על "כשירות" המידע

מכאן והלאה עזבנו את העולם של חומרה בטוחה (בערך) ואנחנו נכנסים לעולם של פרוייקטים בלמידת מכונה וזה אומר שההתעסקות עם המידע היא שונה.

מכאן והלך הזליגות הם הtrain set וערך המטרה או הlable הוא המפתח\משקל המינג ועוד.

<u>כלומר</u>

X = traces

Y=keys\SBOX(key xor plain)\HW(SBOX(key xor plain))

עכשיו נותר להרכיב את המודל ולהחליט באיזה רשת אנחנו נשתמש ואז לאמן ו"לתקוף" שאצלנו זה אומר פשוט לבצע פרדיקציה על חלק מהמידע אותו המודל לא ראה מעולם.

<u>RNN</u>

עבור הפרוייקט שלנו רצינו להשוות בין הרשתות השונות, כלומר לממש ולאמן רשת rnn פשוטה ולהשתמש בה כדי לחזות את ערך המטרה ולאחר מכן לממש באמצעות שכבות LSTM ושכבות GRU ולהשוות בין הפרמטרים השונים בינהם, למשל מספר השכבות, כמות הזליגות הנדרשות כדי למצוא את ה"דפוסים", הדיוק המרבי(בשלב התקיפה מה אחוז החיזויים הנכון), קצב הלמידה, הloss ועוד.

חשוב לציין שרצינו לעשות זאת אבל נתקלנו בבעיית למידה שהקראת Underfitting.

מאפיין של בעיה כזאת הוא שערכי הloss עבור הרשומות בtrain ובvalidation די שקולים ולכן נראה שהמודל לא מצליח למצוא את הדפוסים של המידע.

*<u>נזכיר בקצרה מה זאת פונקציית הloss בכלל</u>: דיברנו בהתחלה על פונקציות cost שערכיות את טיב הלמידה או את טיב השגיאה כלומר כמה החיזוי של המודל עבור הרשומות הנוכחיות סוטה מהערכי הלייבלים

ישנן שלל פונקציות כאלו כמו MSE,RMSE אבל לשם פשטות החישובים נבחר **<u>categorical_crossentropy</u>**

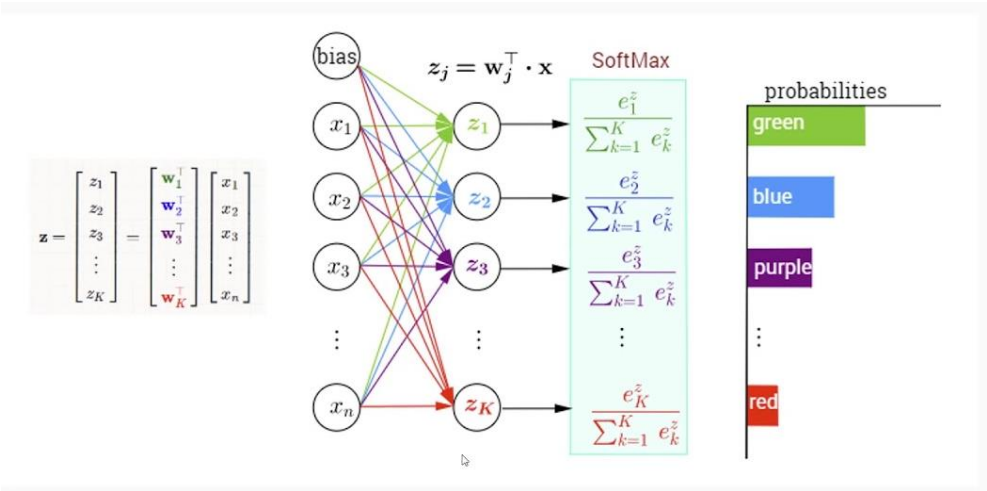
ראשית categorical זה אומר קידוד של הלייבלים כמעין קידוד של one_hot כלומר אם ישנם 9 קלאסים מ0-8 אז אחרי המרה הלייבלים הופכים להיות ווקטורים מאורך 9 כאשר כל פוזיציה מסמלת קלאס אחר.

לאחר מכן מחשבים את האנטרופיה על הערכים הללו וזה מייצגת את פונקציית הLoss

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\ \text{size}}} y_i \cdot \log \hat{y}_i$$

בנוסף מומלץ, בכל מודל שעוסק בקלסיפיקציה, להוסיף בסוף שכבה שהיא dense fully connected מאחר וידוע כי השכבות הללו עושות את עבודות הסיווג בצורה טובה

עבור אותה שכבה נשתמש בפונקצית אקטיבציה 'softmax'



פונקציה "לוגיסטית" מאוד חשובה שעושים בה שימוש מאוד נרחב בסיווג עם המון קלאסים והיא בעצם מנרמלת את הפלטים של השכבה האחרונה

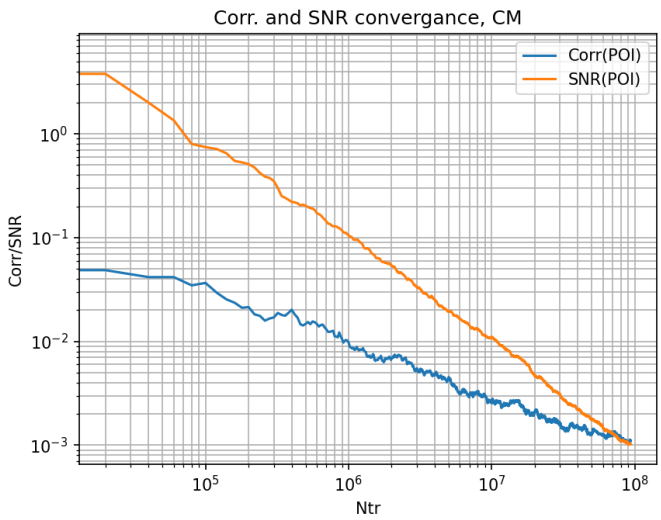
$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

להתפלגות של הסתברות מסויימת עבור הקלאסים השונים

Results

ניסינו וריאציות שונות הן במבנה הרשת והן בכמות המידע וצורתו אשר נשלחת ללמוד מתוכו את הקורלציה למפתחות, כאשר בכולם קיבלנו תוצאות דיי דומות ולכן נראה כאן מקבץ מייצג.

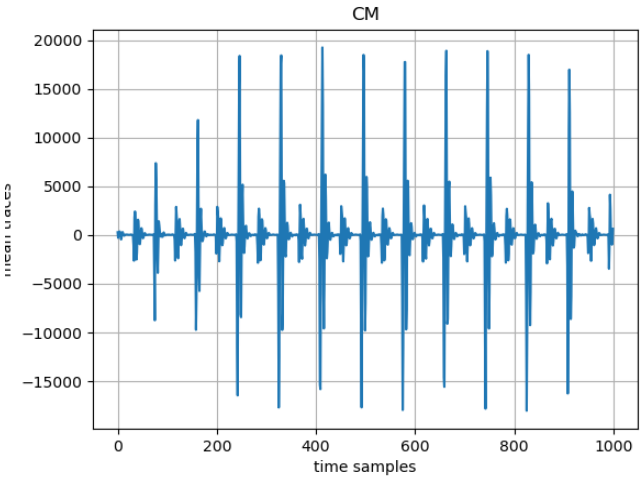
תחילה נראה את התפלגות הSNR והקורלציה כתלות במספר הטרייסים עבור הרכיב בלי ההגנות:

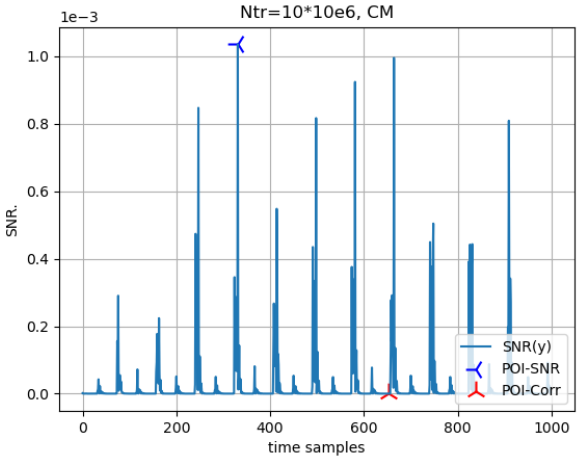


ניתן לראות כי עבור מעט מעוד טרייסים(10^5>) הSNR גבוה מ0 לכן אנחנו מצפים שנוכל ללמוד לאפיין בצורה טובה את המודל מה גם שזאת אינדיקציה לכך שאין הרבה הגנות ונוכל לתקוף עם מעט מאוד טרייסים רכיב חדש שלא ניראה.

בחלק מן ההרצות ניסינו לבצע feature selection בהתבסס פרמטר chi^2 לקביעת קורלציה מיטבית ובכך ליצור את המידע החדש אותו נזין

נסתכל לרגע על *SNR* של טרייס אחד(מימין *SNR* משמאל):





ניתן לראות כי ישנן המון נקודות עניין *POI* ולכן שוב אנו מצפים לחלץ את המידע.

לפי נקודות העיניין האלה, נבחרת הטובה ביותר ונלקחת סביבה כמות נקודות שאנחנו החלטנו כמובן פעלנו גם בגישה ההפוכה של לקחת את כל הטרייס ולבחור מתוכו 10 או יותר נקודות שירכיבו את סדרות הזמן שיוכנסו לתוך האלגו'.

<u>דוגמאות לריצות אימון ופרדיצה:</u>

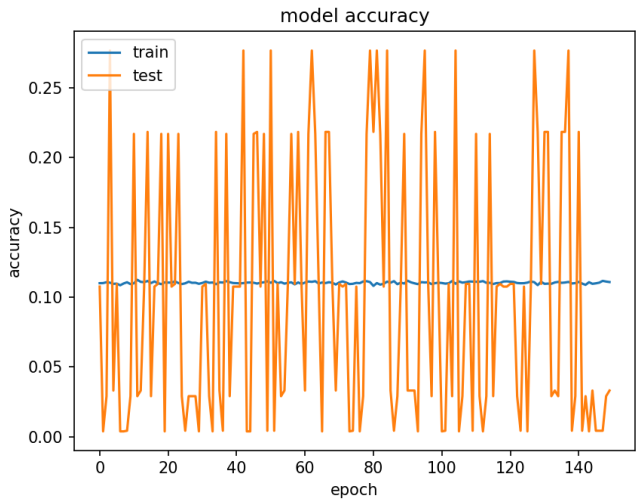
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 6, 64)             145920

 lstm_1 (LSTM)               (None, 64)                33024

 dense (Dense)               (None, 9)                 585

 activation (Activation)     (None, 9)                 0

=================================================================
Total params: 179,529
Trainable params: 179,529
Non-trainable params: 0
_________________________________________________________________
Train...
Epoch 1/150
3380/3380 [==============================] - 158s 42ms/step - loss: 2.1974 - accuracy: 0.1101 - val_loss: 2.1942 - val_accuracy: 0.1076
Epoch 2/150
3380/3380 [==============================] - 134s 40ms/step - loss: 2.1974 - accuracy: 0.1101 - val_loss: 2.1929 - val_accuracy: 0.0039
Epoch 3/150
3380/3380 [==============================] - 79s 23ms/step - loss: 2.1973 - accuracy: 0.1108 - val_loss: 2.1991 - val_accuracy: 0.0291
```

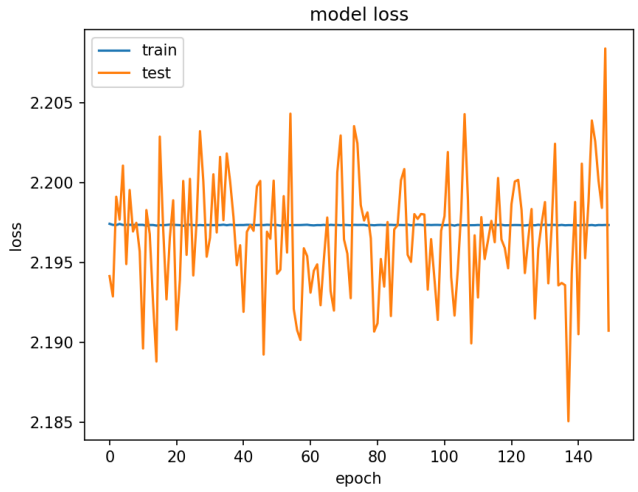
```
Epoch 143/150
3380/3380 [==============================] - 57s 17ms/step - loss: 2.1974 - accuracy: 0.1089 - val_loss: 2.1953 - val_accuracy: 0.0291
Epoch 144/150
3380/3380 [==============================] - 56s 17ms/step - loss: 2.1973 - accuracy: 0.1108 - val_loss: 2.1988 - val_accuracy: 0.0039
Epoch 145/150
3380/3380 [==============================] - 57s 17ms/step - loss: 2.1974 - accuracy: 0.1096 - val_loss: 2.2039 - val_accuracy: 0.0332
Epoch 146/150
3380/3380 [==============================] - 57s 17ms/step - loss: 2.1973 - accuracy: 0.1100 - val_loss: 2.2026 - val_accuracy: 0.0044
Epoch 147/150
3380/3380 [==============================] - 56s 17ms/step - loss: 2.1974 - accuracy: 0.1104 - val_loss: 2.2001 - val_accuracy: 0.0044
Epoch 148/150
3380/3380 [==============================] - 111s 33ms/step - loss: 2.1973 - accuracy: 0.1117 - val_loss: 2.1984 - val_accuracy: 0.0044
Epoch 149/150
3380/3380 [==============================] - 137s 41ms/step - loss: 2.1974 - accuracy: 0.1112 - val_loss: 2.2084 - val_accuracy: 0.0291
Epoch 150/150
3380/3380 [==============================] - 138s 41ms/step - loss: 2.1973 - accuracy: 0.1109 - val_loss: 2.1907 - val_accuracy: 0.0332
6/6 [==============================] - 4s 263ms/step - loss: 2.1907 - accuracy: 0.0332
Test score: 2.1907386779785156
Test accuracy: 0.03316326439380646
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

כאן אימנו על K500 טרייסים שחולקו לקבוצות כמובן ערכי ה y הם צורת HW של המוצא הsbox.

ניתן לראות במיידי כי המודל לא התמודד בצורה טובה עם מטלה (שנראית פשוטה)

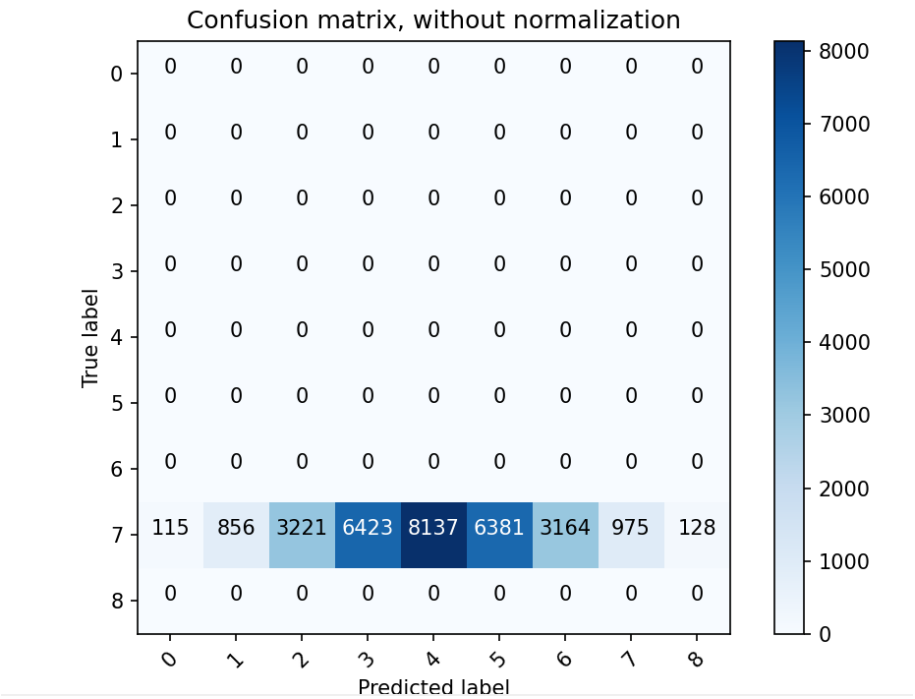
ערך הדיוק נמוך וערך הלוס גבוה מ1.





בנוסף אם נצייר את הCM(מטריצה מאוד שימושית עבור למידת מכונה)

ניתן לראות אילו ערכים המודל ניחש אל מול אלו שהיו בפועל



*עבור חלק מן ההרצות שכחנו להחליף בין הכותרות.(הTRUE אמור ללכת לפי HW אבל כאן זה הפוך)

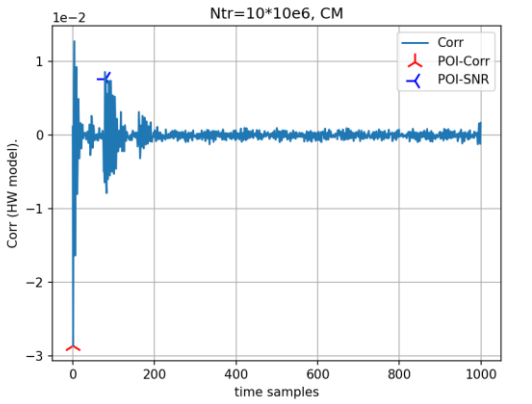
בלי קשר ניתן לראות שהמפתח מתפלג לפי משקל המינג וגם שהדיוק נמוך כי אחרת אלו היו אותם המספרים אבל ממוקמים על האלכסון כלומר המודל שלנו כאן ניחש תמיד משקל 7.

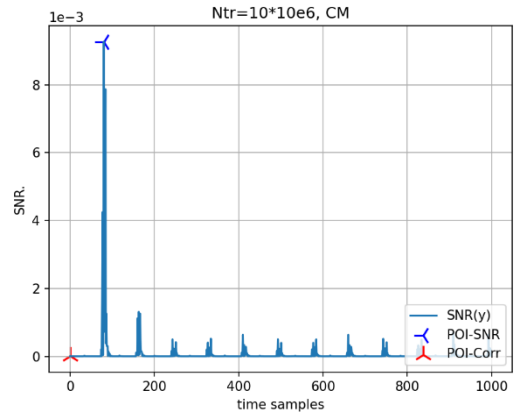
(מודל טיפש)

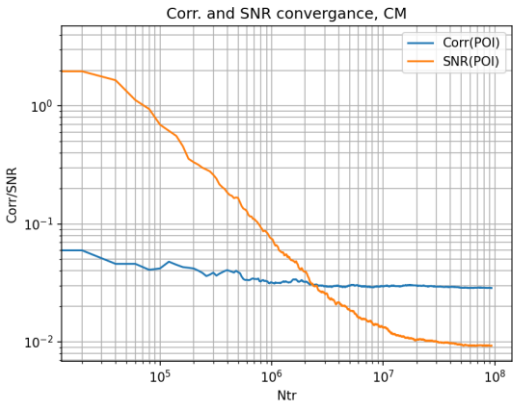
```
Acc,13/20 test: 0.0
Rank (Guessing Entropy),14/20 test: 255.0
Root Mean Squared Error (RMSE), 14/20 test: 3.336205659013502
Acc,14/20 test: 0.0
load_data 9 of 20:  75%|          | 15/20 [00:03<00:01,  3.64it/s]Rank (Guessing Entropy),15/20 test: 255.0
Root Mean Squared Error (RMSE), 15/20 test: 3.3412125145844374
Acc,15/20 test: 0.0
Rank (Guessing Entropy),16/20 test: 255.0
Root Mean Squared Error (RMSE), 16/20 test: 3.337594176007762
Acc,16/20 test: 0.0
load_data 9 of 20:  85%|          | 17/20 [00:04<00:00,  3.58it/s]Rank (Guessing Entropy),17/20 test: 255.0
Root Mean Squared Error (RMSE), 17/20 test: 3.3390915995642247
Acc,17/20 test: 0.0
Rank (Guessing Entropy),18/20 test: 255.0
Root Mean Squared Error (RMSE), 18/20 test: 3.3439225741362764
Acc,18/20 test: 0.0
load_data 9 of 20:  95%|          | 19/20 [00:04<00:00,  3.50it/s]Rank (Guessing Entropy),19/20 test: 255.0
Root Mean Squared Error (RMSE), 19/20 test: 3.3191294790373815
Acc,19/20 test: 0.0
Rank (Guessing Entropy),20/20 test: 255.0
Root Mean Squared Error (RMSE), 20/20 test: 3.324923775101652
Acc,20/20 test: 0.0
load_data 9 of 20: 100%|          | 20/20 [00:05<00:00,  3.87it/s]
```

ניתן לראות כי בנוסף אנחנו מחשבים גם RMSE וגם את הPGE כדי לבצע הערכה נוספת של טיב החיזוי.(עבור הריצה הספיציפית הזאת אין ולכן נדבר על זה בהמשך)

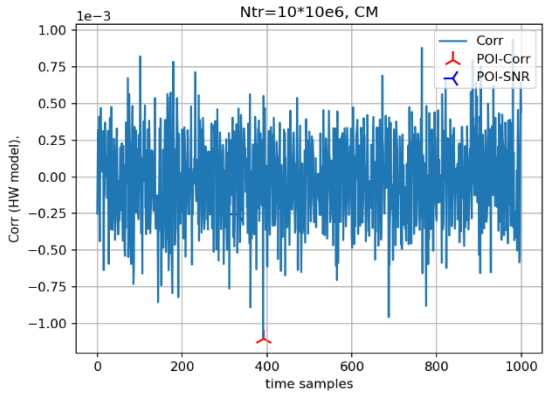
עם HW

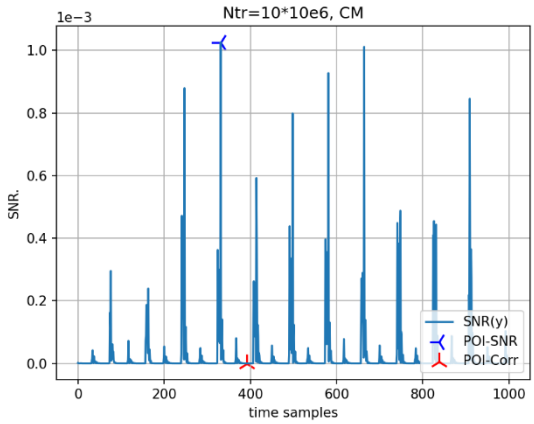


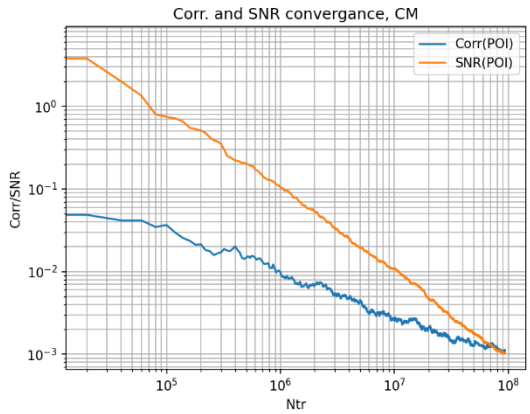


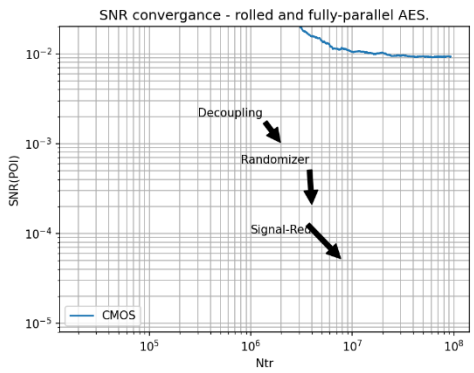


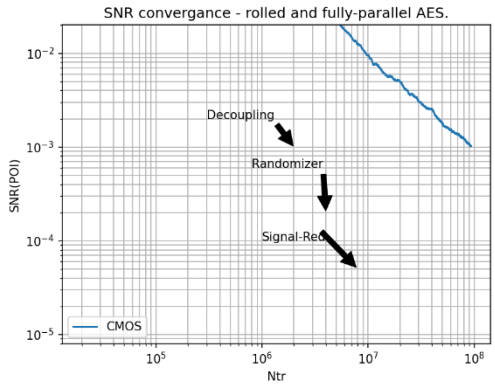
עם HD

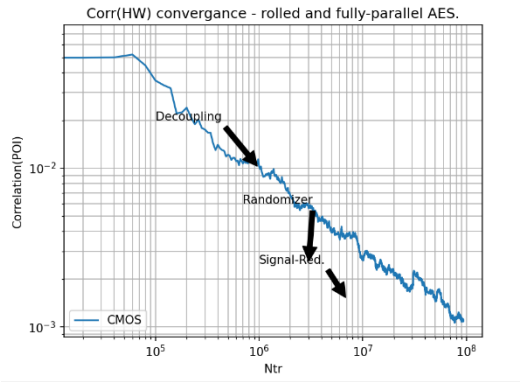


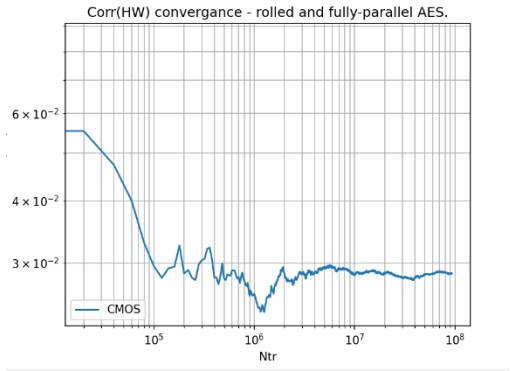


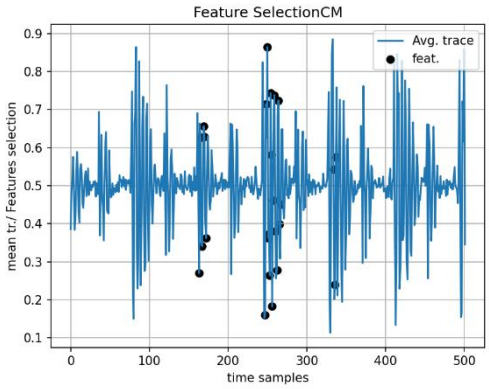












אם נבצע השוואה פשוטה בין המודלים עבור הרכיב המדובר, לדעתנו ניתן לראות כי מודל HW טוב יותר משום:

1.זהו חישוב של sbox שמתבצע ב10 מחזורי חישוב ורואים זאת בגרף הsnr שהמידע מתפלג לאורך הטרייס ומתאים ל10 מחזורים בעוד שעבור HD רוב המידע מתנקז בנקודה אחת

2.הקורלציה גבוהה יותר עבור HW

דוגמא 2: עם HW

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 55, 9)             1359855

 lstm (LSTM)                 (None, 55, 128)           70656

 lstm_1 (LSTM)               (None, 64)                49408

 dense (Dense)               (None, 9)                 585

=================================================================
Total params: 1,480,504
Trainable params: 1,480,504
Non-trainable params: 0
_________________________________________________________________
Train...
Epoch 1/150
3720/3720 [==============================] - 489s 130ms/step - loss: 7.4519 - accuracy: 0.1111 - val_loss: 9.8768 - val_accuracy: 0.1091
```

```
Epoch 1/150
3720/3720 [==============================] - 489s 130ms/step - loss: 7.4519 - accuracy: 0.1111 - val_loss: 9.8768 - val_accuracy: 0.1091
Epoch 2/150
3720/3720 [==============================] - 472s 127ms/step - loss: 7.9829 - accuracy: 0.1111 - val_loss: 6.2413 - val_accuracy: 0.1091
Epoch 3/150
3720/3720 [==============================] - 479s 129ms/step - loss: 7.2949 - accuracy: 0.1111 - val_loss: 6.2413 - val_accuracy: 0.1091
Epoch 4/150
3720/3720 [==============================] - 477s 128ms/step - loss: 7.2881 - accuracy: 0.1111 - val_loss: 6.2413 - val_accuracy: 0.1091
Epoch 5/150
3720/3720 [==============================] - 486s 131ms/step - loss: 7.2754 - accuracy: 0.1111 - val_loss: 6.2413 - val_accuracy: 0.1091
Epoch 6/150
3720/3720 [==============================] - 486s 131ms/step - loss: 7.2982 - accuracy: 0.1111 - val_loss: 6.2413 - val_accuracy: 0.1091
Epoch 7/150
3720/3720 [==============================] - 495s 133ms/step - loss: 7.2955 - accuracy: 0.1111 - val_loss: 6.2413 - val_accuracy: 0.1091
Epoch 8/150
3720/3720 [==============================] - 506s 136ms/step - loss: 7.3022 - accuracy: 0.1111 - val_loss: 6.2413 - val_accuracy: 0.1091
Epoch 9/150
3720/3720 [==============================] - 503s 135ms/step - loss: 7.2938 - accuracy: 0.1111 - val_loss: 6.2413 - val_accuracy: 0.1091
Epoch 10/150
3720/3720 [==============================] - 505s 136ms/step - loss: 7.2966 - accuracy: 0.1111 - val_loss: 6.2413 - val_accuracy: 0.1091
```

```
3720/3720 [==============================] - 673s 181ms/step - loss: 8.9545 - accuracy: 0.1114 - val_loss: 9.8768 - val_accuracy: 0.2169
Epoch 143/150
3720/3720 [==============================] - 674s 181ms/step - loss: 8.9545 - accuracy: 0.1111 - val_loss: 9.8768 - val_accuracy: 0.2169
Epoch 144/150
3720/3720 [==============================] - 677s 182ms/step - loss: 8.9546 - accuracy: 0.1112 - val_loss: 9.8768 - val_accuracy: 0.2169
Epoch 145/150
3720/3720 [==============================] - 682s 183ms/step - loss: 8.9545 - accuracy: 0.1112 - val_loss: 9.8768 - val_accuracy: 0.2169
Epoch 146/150
3720/3720 [==============================] - 681s 183ms/step - loss: 8.9545 - accuracy: 0.1109 - val_loss: 9.8768 - val_accuracy: 0.2169
Epoch 147/150
3720/3720 [==============================] - 680s 183ms/step - loss: 8.9546 - accuracy: 0.1113 - val_loss: 9.8768 - val_accuracy: 0.2169
Epoch 148/150
3720/3720 [==============================] - 680s 183ms/step - loss: 8.9545 - accuracy: 0.1110 - val_loss: 9.8768 - val_accuracy: 0.2169
Epoch 149/150
3720/3720 [==============================] - 687s 185ms/step - loss: 8.9545 - accuracy: 0.1111 - val_loss: 9.8768 - val_accuracy: 0.2169
Epoch 150/150
3720/3720 [==============================] - 694s 187ms/step - loss: 8.9544 - accuracy: 0.1112 - val_loss: 9.8768 - val_accuracy: 0.2169
7/7 [==============================] - 5s 696ms/step - loss: 9.8768 - accuracy: 0.2169
Test score: 9.876802444458008
Test accuracy: 0.2168518453836441
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

כאן אימנו על K500 טרייסים.

והשינוי המרכזי הוא מהוספת שכבה נוספת עליונה ,embbeding , שלמעשה עושה טרנספורמציה למידע ומעבירה אותו לקידוד של ווקטור ומחשבת מימד שהוא לפי כמות הערכים השונה בסט הלמידה.

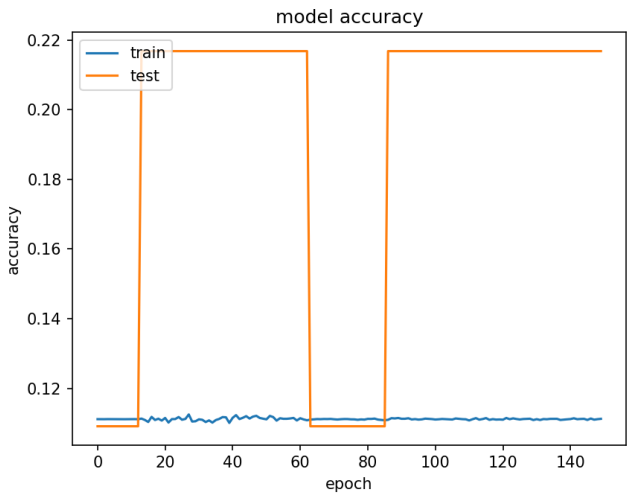
שכבה זאת עוזרת למודל להתמודד עם מטלות של Multiclassification ואחרי שלב הלמידה יכולה בקלות להתמודד עם מידע שלא ראתה באמצעות חישוב מרחקים(כמובן הכל מוחבא(hidden layers)).

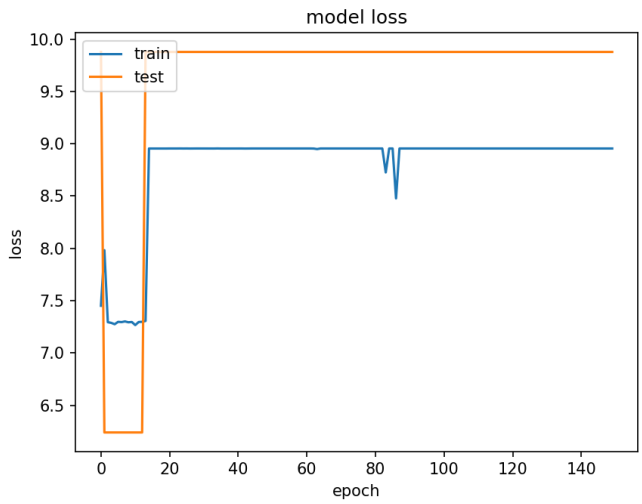
עד כה נשמע מצויין אך אף פעם אין ארוחות חינם, הפעם אנו נשלם בזמן הלמידה.

פשוט לראות זאת ע"י כמות הפרמטרים שגדלה משמעותית $1.5\ M$ אל מול $200\ K$,

זמן הריצה של כל epoch עלה פי 10 כלומר מדקה ל10 דקות

<u>גם כאן המודל לא צלח את המשימה.</u>





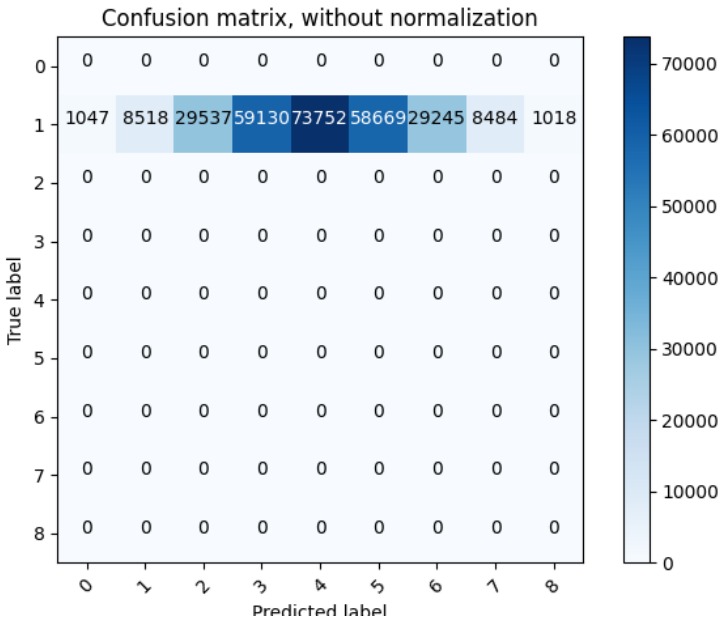
<u>דוגמא 3:</u>

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 6, 64)             30720

 lstm_1 (LSTM)               (None, 6, 64)             33024

 lstm_2 (LSTM)               (None, 32)                12416

 dense (Dense)               (None, 9)                 297

 activation (Activation)     (None, 9)                 0

=================================================================
Total params: 76,457
Trainable params: 76,457
Non-trainable params: 0
_________________________________________________________________
Train...
Epoch 1/100
12572/12572 [==============================] - 245s 18ms/step - loss: 2.1978 - accuracy: 0.0514 - val_loss: 2.1923 - val_accuracy: 0.1095
Epoch 2/100
12572/12572 [==============================] - 205s 16ms/step - loss: 2.1979 - accuracy: 0.0815 - val_loss: 2.2082 - val_accuracy: 0.0040
```

```
12572/12572 [==============================] - 174s 14ms/step - loss: 2.1975 - accuracy: 0.0513 - val_loss: 2.1870 - val_accuracy: 0.1095
Epoch 95/100
12572/12572 [==============================] - 174s 14ms/step - loss: 2.1975 - accuracy: 0.0645 - val_loss: 2.1874 - val_accuracy: 0.1095
Epoch 96/100
12572/12572 [==============================] - 174s 14ms/step - loss: 2.1974 - accuracy: 0.0757 - val_loss: 2.2015 - val_accuracy: 0.0039
Epoch 97/100
12572/12572 [==============================] - 174s 14ms/step - loss: 2.1975 - accuracy: 0.0444 - val_loss: 2.1937 - val_accuracy: 0.1095
Epoch 98/100
12572/12572 [==============================] - 174s 14ms/step - loss: 2.1974 - accuracy: 0.0684 - val_loss: 2.2048 - val_accuracy: 0.0039
Epoch 99/100
12572/12572 [==============================] - 174s 14ms/step - loss: 2.1975 - accuracy: 0.0366 - val_loss: 2.1837 - val_accuracy: 0.2194
Epoch 100/100
12572/12572 [==============================] - 174s 14ms/step - loss: 2.1975 - accuracy: 0.0888 - val_loss: 2.1949 - val_accuracy: 0.2194
54/54 [==============================] - 4s 63ms/step - loss: 2.1949 - accuracy: 0.2194
Test score: 2.1948633193969727
Test accuracy: 0.21940608322620392
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

כאן ניסינו להכניס יותר מדידות ולכן ביצענו אימון על $1.3Mil$ טרייסים

כמובן שגם המודל הנ"ל נכשל

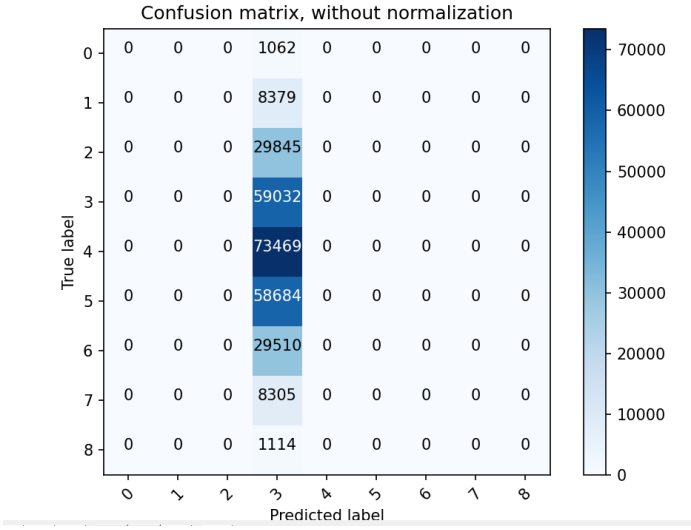


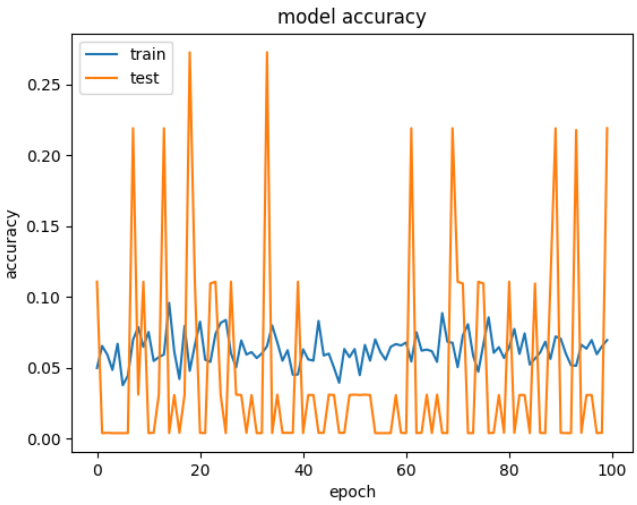
דוגמא 4 :

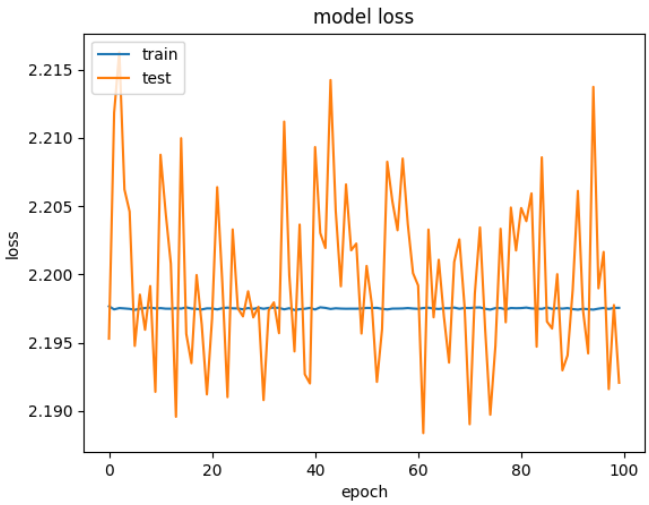
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 6, 64)             30720

 lstm_1 (LSTM)               (None, 6, 64)             33024

 lstm_2 (LSTM)               (None, 32)                12416

 dense (Dense)               (None, 9)                 297

 activation (Activation)     (None, 9)                 0

=================================================================
Total params: 76,457
Trainable params: 76,457
Non-trainable params: 0
_________________________________________________________________
Train...
Epoch 1/100
12572/12572 [==============================] - 178s 13ms/step - loss: 2.1977 - accuracy: 0.0499 - val_loss: 2.1953 - val_accuracy: 0.1108
Epoch 2/100
```

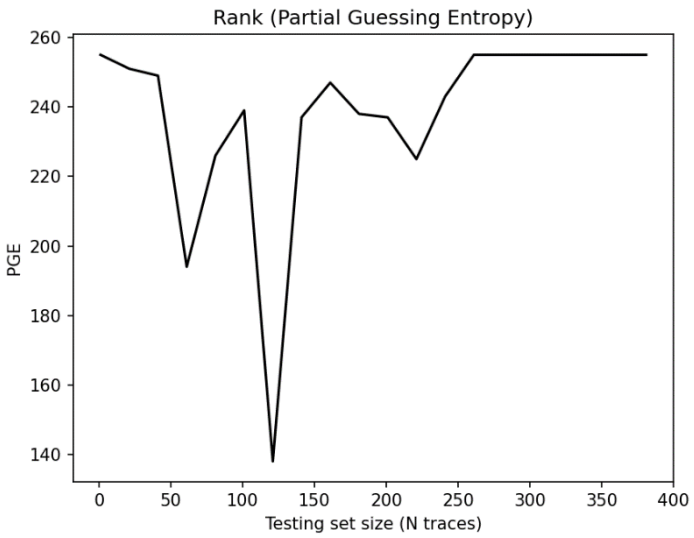
```
Epoch 94/100
12572/12572 [==============================] - 169s 13ms/step - loss: 2.1974 - accuracy: 0.0515 - val_loss: 2.1942 - val_accuracy: 0.2178
Epoch 95/100
12572/12572 [==============================] - 168s 13ms/step - loss: 2.1974 - accuracy: 0.0663 - val_loss: 2.2137 - val_accuracy: 0.0041
Epoch 96/100
12572/12572 [==============================] - 168s 13ms/step - loss: 2.1975 - accuracy: 0.0635 - val_loss: 2.1990 - val_accuracy: 0.0308
Epoch 97/100
12572/12572 [==============================] - 167s 13ms/step - loss: 2.1975 - accuracy: 0.0696 - val_loss: 2.2016 - val_accuracy: 0.0308
Epoch 98/100
12572/12572 [==============================] - 167s 13ms/step - loss: 2.1975 - accuracy: 0.0596 - val_loss: 2.1916 - val_accuracy: 0.0039
Epoch 99/100
12572/12572 [==============================] - 168s 13ms/step - loss: 2.1975 - accuracy: 0.0652 - val_loss: 2.1977 - val_accuracy: 0.0041
Epoch 100/100
12572/12572 [==============================] - 167s 13ms/step - loss: 2.1975 - accuracy: 0.0696 - val_loss: 2.1921 - val_accuracy: 0.2191
54/54 [==============================] - 4s 64ms/step - loss: 2.1921 - accuracy: 0.2191
Test score: 2.1920552253723145
Test accuracy: 0.219123974442482
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
Generating test predictions...
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1062
1	0.00	0.00	0.00	8379
2	0.00	0.00	0.00	29845
3	0.22	1.00	0.36	59032
4	0.00	0.00	0.00	73469
5	0.00	0.00	0.00	58684
6	0.00	0.00	0.00	29510
7	0.00	0.00	0.00	8305
8	0.00	0.00	0.00	1114
accuracy			0.22	269400
macro avg	0.02	0.11	0.04	269400
weighted avg	0.05	0.22	0.08	269400









מה זה Guessing Entropy?

מדד לכמה פעמים תוקף צריך לנחש את המפתח כדי לפגוע בהינתן המודל.

$$\sum_{i>0} i \cdot \Pr\left[p_i^{(k)} \leftarrow \mathcal{S}\right]$$

*באדיבות
https://www.usenix.org/legacy/publications/library/proceedings/sec04/tech/full_papers/davis/davis_html/node8.html

ניתן לראות שזה בדיוק תוחלת, ולכן אם נראה 0 נדע כי התוקף הצליח.

אולם ישנו סיכוי שהאלגוריתם ניחש במקרה את המפתח נכון ולא כי ידע לקשר בין הזליגות למפתח הנכון.

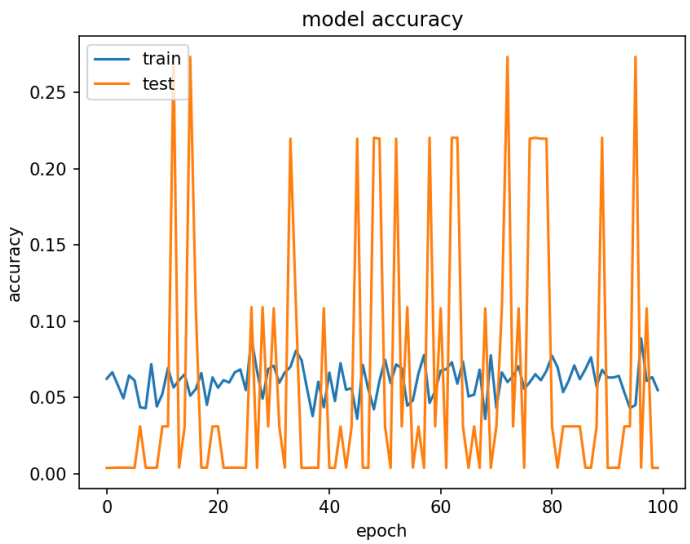
דוגמא 5 :

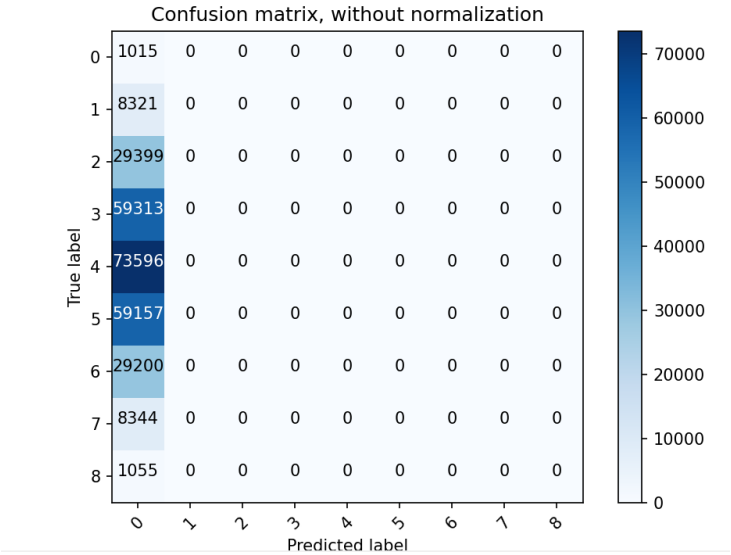
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 6, 64)             23040

 gru_1 (GRU)                 (None, 6, 32)             9312

 simple_rnn (SimpleRNN)      (None, 32)                2080

 dense (Dense)               (None, 9)                 297

 activation (Activation)     (None, 9)                 0

=================================================================
Total params: 34,729
Trainable params: 34,729
Non-trainable params: 0
_________________________________________________________________
Train...
Epoch 1/100
12572/12572 [==============================] - 245s 19ms/step - loss: 2.1986 - accuracy: 0.0622 - val_loss: 2.1906 - val_accuracy: 0.0038
Epoch 2/100
```

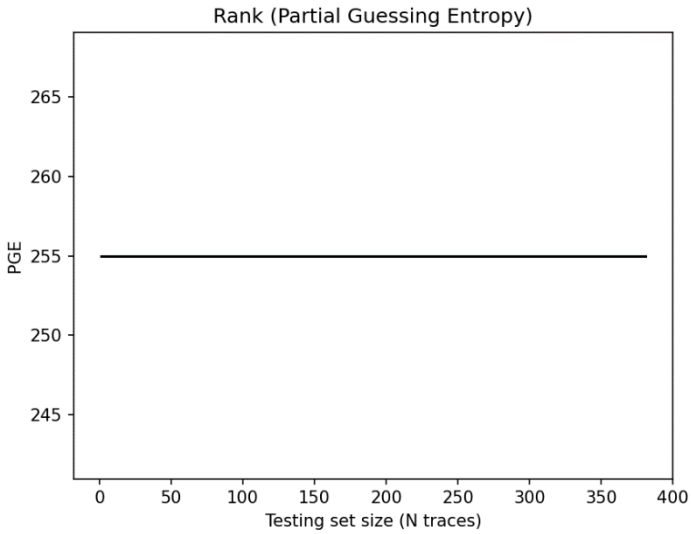
```
Epoch 93/100
12572/12572 [==============================] - 108s 9ms/step - loss: 2.1977 - accuracy: 0.0641 - val_loss: 2.1938 - val_accuracy: 0.0038
Epoch 94/100
12572/12572 [==============================] - 108s 9ms/step - loss: 2.1975 - accuracy: 0.0532 - val_loss: 2.1963 - val_accuracy: 0.0310
Epoch 95/100
12572/12572 [==============================] - 108s 9ms/step - loss: 2.1976 - accuracy: 0.0429 - val_loss: 2.2047 - val_accuracy: 0.0310
Epoch 96/100
12572/12572 [==============================] - 108s 9ms/step - loss: 2.1978 - accuracy: 0.0451 - val_loss: 2.1907 - val_accuracy: 0.2732
Epoch 97/100
12572/12572 [==============================] - 108s 9ms/step - loss: 2.1976 - accuracy: 0.0886 - val_loss: 2.1907 - val_accuracy: 0.0039
Epoch 98/100
12572/12572 [==============================] - 109s 9ms/step - loss: 2.1976 - accuracy: 0.0608 - val_loss: 2.1895 - val_accuracy: 0.1084
Epoch 99/100
12572/12572 [==============================] - 189s 15ms/step - loss: 2.1977 - accuracy: 0.0633 - val_loss: 2.2052 - val_accuracy: 0.0038
Epoch 100/100
12572/12572 [==============================] - 247s 20ms/step - loss: 2.1976 - accuracy: 0.0547 - val_loss: 2.1946 - val_accuracy: 0.0038
54/54 [==============================] - 5s 75ms/step - loss: 2.1946 - accuracy: 0.0038
Test score: 2.1945927143096924
Test accuracy: 0.0037676317151635885
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
Generating test predictions...
```

	precision	recall	f1-score	support
0	0.00	1.00	0.01	1015
1	0.00	0.00	0.00	8321
2	0.00	0.00	0.00	29399
3	0.00	0.00	0.00	59313
4	0.00	0.00	0.00	73596
5	0.00	0.00	0.00	59157
6	0.00	0.00	0.00	29200
7	0.00	0.00	0.00	8344
8	0.00	0.00	0.00	1055
accuracy			0.00	269400
macro avg	0.00	0.11	0.00	269400
weighted avg	0.00	0.00	0.00	269400

כאן במקום להשתמש בשכבות LSTM ניסינו באמצעות שכבות GRU לא ניתן להבין בשיפור משמעותי לא בתוצאות ולא בזמני ריצה ואפילו ניראה לפי הPGE שהוא לא מצליח בכלל.





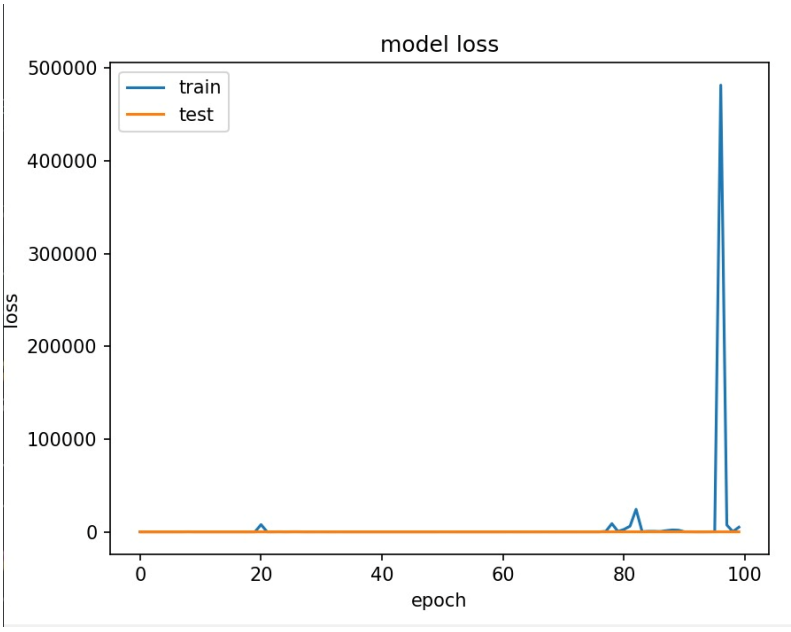


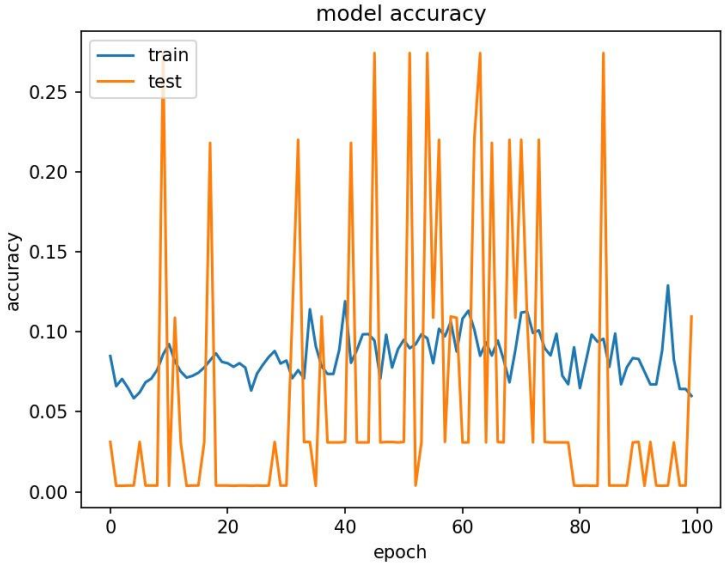
דוגמא 6:

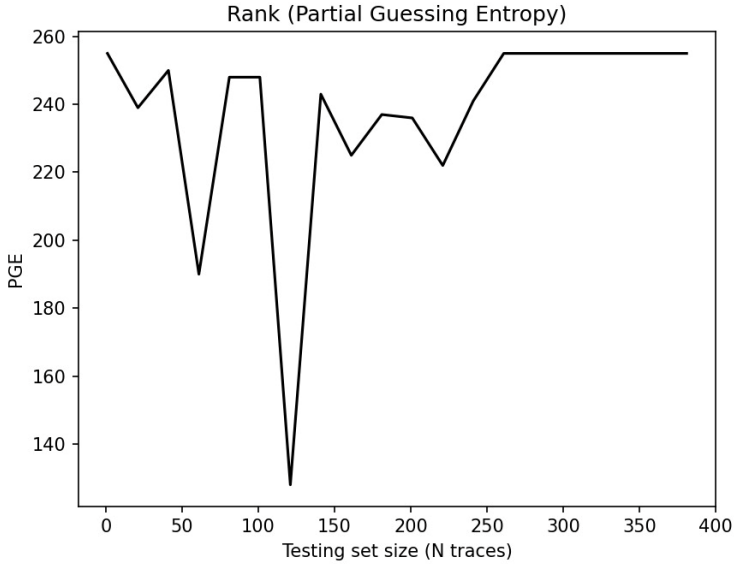
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 6, 128)            94208

 lstm_1 (LSTM)               (None, 64)                49408

 dense (Dense)               (None, 9)                 585

 activation (Activation)     (None, 9)                 0

=================================================================
Total params: 144,201
Trainable params: 144,201
Non-trainable params: 0
_________________________________________________________________
Train...
Epoch 1/100
12572/12572 [==============================] - 168s 13ms/step - loss: 2.2410 - accuracy: 0.0848 - val_loss: 2.2643 - val_accuracy: 0.0311
Epoch 2/100
12572/12572 [==============================] - 161s 13ms/step - loss: 2.2889 - accuracy: 0.0660 - val_loss: 2.2215 - val_accuracy: 0.0037
Epoch 3/100
```

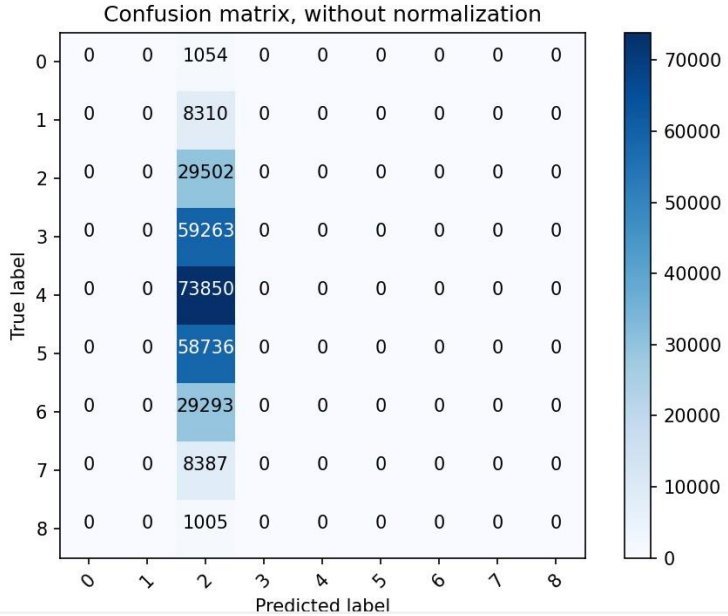
```
Epoch 93/100
12572/12572 [==============================] - 244s 19ms/step - loss: 2.2918 - accuracy: 0.0671 - val_loss: 2.1982 - val_accuracy: 0.0311
Epoch 94/100
12572/12572 [==============================] - 215s 17ms/step - loss: 3.0142 - accuracy: 0.0671 - val_loss: 2.2121 - val_accuracy: 0.0039
Epoch 95/100
12572/12572 [==============================] - 217s 17ms/step - loss: 37.1668 - accuracy: 0.0882 - val_loss: 2.1938 - val_accuracy: 0.0037
Epoch 96/100
12572/12572 [==============================] - 220s 17ms/step - loss: 144.2335 - accuracy: 0.1290 - val_loss: 2.2031 - val_accuracy: 0.0039
Epoch 97/100
12572/12572 [==============================] - 218s 17ms/step - loss: 481741.2812 - accuracy: 0.0826 - val_loss: 2.1957 - val_accuracy: 0.0308
Epoch 98/100
12572/12572 [==============================] - 223s 18ms/step - loss: 7356.8970 - accuracy: 0.0642 - val_loss: 2.2323 - val_accuracy: 0.0039
Epoch 99/100
12572/12572 [==============================] - 258s 21ms/step - loss: 194.1300 - accuracy: 0.0642 - val_loss: 2.2012 - val_accuracy: 0.0039
Epoch 100/100
12572/12572 [==============================] - 269s 21ms/step - loss: 5007.9536 - accuracy: 0.0599 - val_loss: 2.1689 - val_accuracy: 0.1095
54/54 [==============================] - 7s 105ms/step - loss: 2.1689 - accuracy: 0.1095
Test score: 2.168930768966675
Test accuracy: 0.10951001942157745
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
Generating test predictions...
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1054
           1       0.00      0.00      0.00      8310
           2       0.11      1.00      0.20     29502
           3       0.00      0.00      0.00     59263
           4       0.00      0.00      0.00     73850
           5       0.00      0.00      0.00     58736
           6       0.00      0.00      0.00     29293
           7       0.00      0.00      0.00      8387
           8       0.00      0.00      0.00      1005

    accuracy                           0.11    269400
   macro avg       0.01      0.11      0.02    269400
weighted avg       0.01      0.11      0.02    269400

0.10951002227171493
```









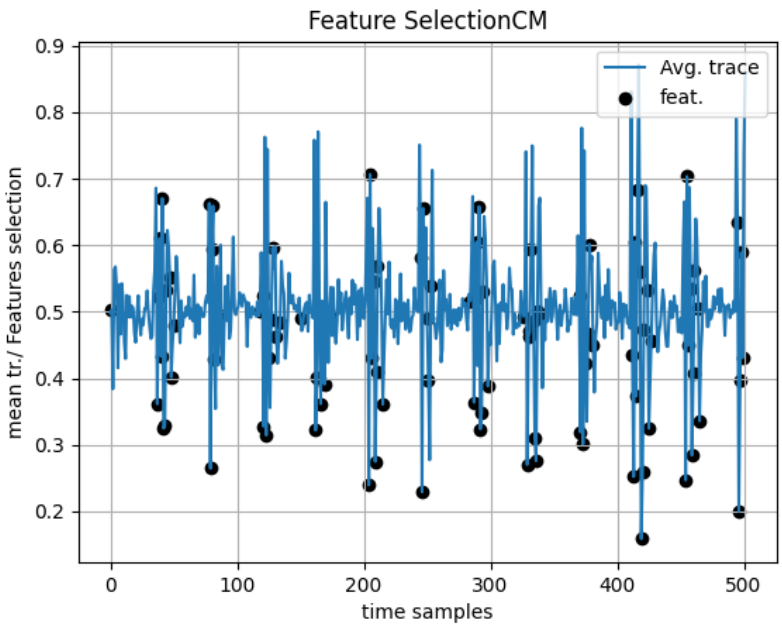
כמו בריצות קודמות הפעם הגדלנו את מורכבות השכבה הראשונה

הרצנו מעל ממיליון דגימות וגם הפעם הוא לא מצליח ללמוד.

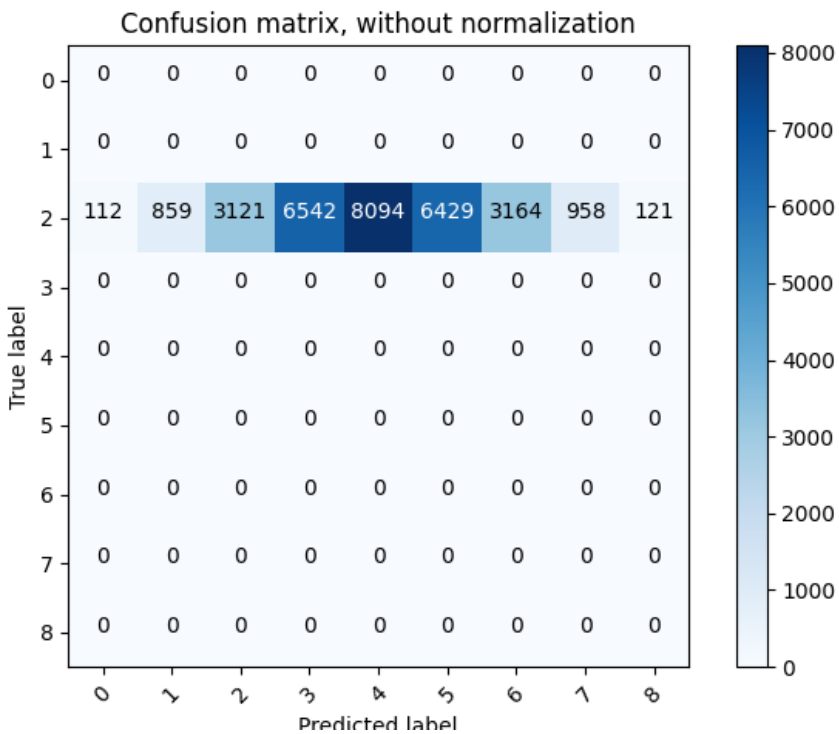
<u>דוגמא 7:</u>

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 11, 128)           91776

 gru_1 (GRU)                 (None, 11, 64)            37056

 gru_2 (GRU)                 (None, 11, 32)            9312

 gru_3 (GRU)                 (None, 32)                6240

 dense (Dense)               (None, 9)                 297

 activation (Activation)     (None, 9)                 0

=================================================================
Total params: 144,681
Trainable params: 144,681
Non-trainable params: 0
_________________________________________________________________
Train...
Epoch 1/70
339/339 [==============================] - 44s 99ms/step - loss: 2.2045 - accuracy: 0.1110 - val_loss: 2.2066 - val_accuracy: 0.1076
Epoch 2/70
339/339 [==============================] - 33s 96ms/step - loss: 2.2016 - accuracy: 0.1105 - val_loss: 2.1874 - val_accuracy: 0.2753
Epoch 3/70
339/339 [==============================] - 33s 97ms/step - loss: 2.2002 - accuracy: 0.1112 - val_loss: 2.2123 - val_accuracy: 0.0041
```

הפעם הרצנו עם פחות טרייסים כ$20K$ אבל הפעם הכנסנו את כל המדידה ומתוכה בחרנו את הפיצרים עם ההתאמה הטובה ביותר לפי chi^2

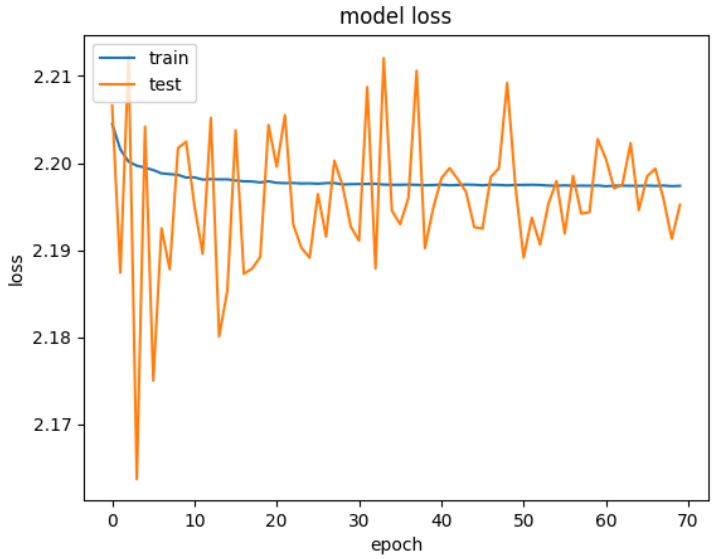


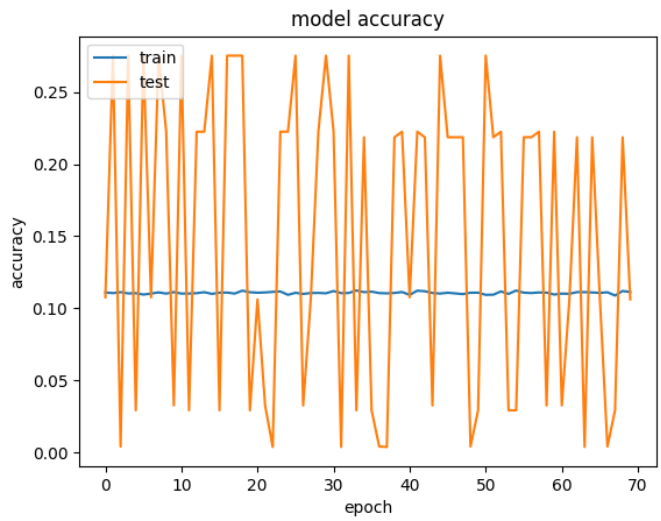
כך יצא שסביב כל POI נלקחו כמה ערכים



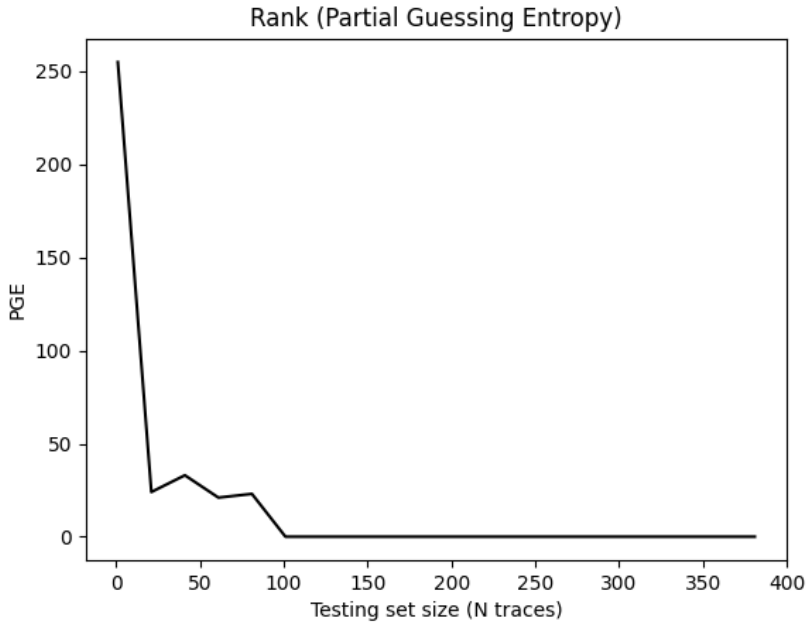
בנוסף הוספנו עוד שכבות והקטנו את כמות המידע שנזרק בין השכבות

ועדיין ניראה כי למודל הRNN קשה ללמוד את המפתח





למרות הביצועים הגרועים ניראה כי לפי הPGE המודל מצליח לנחש בהצלחה ובקביעות לאחר 100 ניחושים\דגימות.



סיכום ומסקנות

לסיכום נהנינו מהפרוייקטון, נדרשנו ללמוד המון, לחקור ולהרחיב בנושא, אך לצערנו לא קיבלנו את התוצאות שקיווינו לקבל, נראה שיש אופק מחקרי נרחב וישנן המון כיוונים ושיטות ונראה כי בעזרת למידת מכונה ולמידה עמוקה כמעט הכל אפשרי.

אפשר לראות שהדיוק שלנו לא עלה ברוב המקרים על 10% ובהתבסס על מחקרים שקראנו אפשרי להגיע למידת דיוק גבוהה יותר, אמנם הרכיבים הרבה פחות רועשים ובכל זאת התמודדנו עם רכיב עם SNR לא הכי נמוך.

כפי שהיה ניתן לראות בTA ,אחד החסמים על התקיפה (כתלות בSNR) הוא אסיפת מספיק דגימות בכדי לחלץ את האינפורמציה . אך ככל שהSNR גבוה יותר נצטרך יותר דגימות כאשר אצלנו זה התבטא בזמן למידה ממושך יותר. במחקרים אחרים הלמידה לקחה כ12 ימים ואצלנו רק כ3 ימים , כמובן אפשר להאיץ זאת עם ספריית CUDA אך לספרייה זו פונקציונליות נמוכה יותר (למשל הפרמטר שנקרא recurrent dropout לא קיים) מאשר מספריות פשוטות יותר ב Keras\Tensor Flow.