

**דוח פרויקט:** סמסטריאלי

**הנושא:**

**solving Sudoku using machine learning and AI tools**

מגישים:

חן דודאי ברק ממיסטבלוב

מנחה:

רון דורפמן

תום יורגנסון

**סמסטר:** חורף

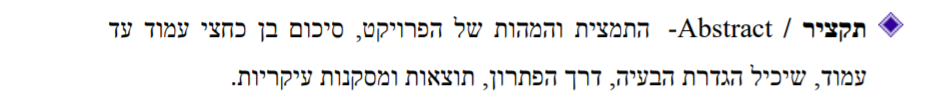
**שנה:** 2020

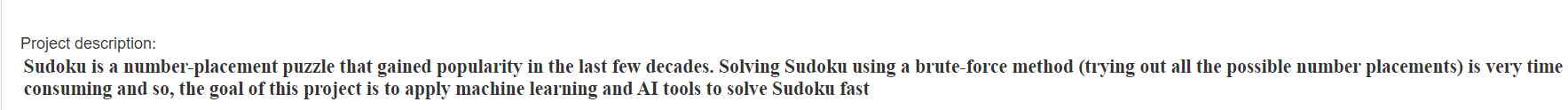
**תוכן עניינים**

**רשימת איורים**

**תקציר**

בשנים האחרונות משחק הסודוקו צבר תאוצה והפופולריות שלו בקרב הציבור הרחב גדלה משמעותית. בד בבד הפופולריות של פתרונות המבוססים על למידת מכונה צברו תאוצה גם כן. ניתן לפתור לוח סודוקו באמצעות שיטה ישירה (brute-force), על ידי ניסוי של כל הפתרונות האפשריים בזה אחר זה, תוך שימוש ב-backtracking. אולם, שיטה זו היא בעלת סיבוכיות חישוב גבוהה מאוד ולוקחת זמן רב לפתרון. בפרויקט זה עשינו שימוש בשיטות מבוססות למידה מכונה. מצד אחד שיטות מבוססות למידת מכינה פותרות מהר יותר בזמן מבחן, אך מצד שני הן דורשות סט אימון גדול. נוסו ארכיטקטורות deep learning שונות ושיטות אימון שונות, על מנת לפתור הבעיה באופן היעיל והטוב ביותר.

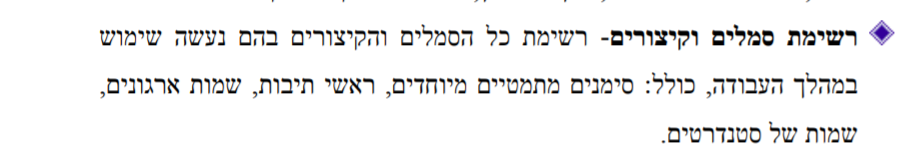




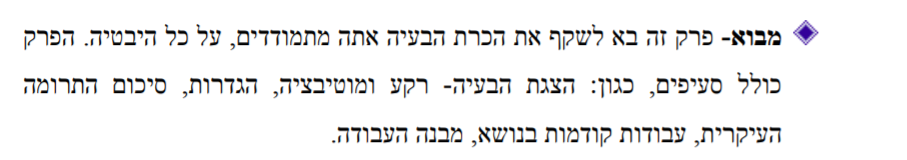
**Abstract**

The Sudoku game is a number placement puzzle that gained popularity in the newspapers and among the public in the last few years. Furthermore, machine learning algorithms has gain massive popularity also. One way to solve Sudoku automatically is with brute-force algorithm, and trying every possible solution using backtracking algorithm. The problem with this method is that it is very time consuming and has high complexity. In this project we solved this problem using deep learning algorithm, which on one hand is less time consuming in test time, but on the other hand demands big database of Sudoku puzzles. We applied different deep learning algorithms and different training methods to solve this problem the best and efficient way possible.

**רשימת סמלים וקיצורים**



1. **מבוא**



1. **סודוקו – רקע כללי**

סודוקו הינו תשבץ מספרים שבו צריך למקם ספרות על לוח משובץ שגודלו 9X9, המורכב מ-9 ריבועים בני 9 משבצות כל אחד. פירושו המילה "סודוקו" ביפנית היא "מספר יחיד", משום שמטרת המשחק היא למקם 9 ספרות (הספרות 1 עד 9) על גבי לוח המשחק כך שבכל טור, בכל שורה, ובכל ריבוע, יופיע כל סמל בדיוק פעם אחת. בלוח המשחק נתונים כמה ספרות, ויש להתייחס אליהן בעת מיקום הספרות החדשות במהלך המשחק.

תשבץ הסודוקו זכה לפופולריות רבה ביפן בשנת 1986 ובבריטניה, בקנדה, ובישראל בשנת 2005 בעקבות קידומו בעיתונות. ישנם חוקים המייחסים לפתרון תשבצי סודוקו סגולות של שיפור או שימור כישורים שכליים.

פתרון סודוקו תקף הוא גם ריבוע לטיני. ריבוע לטיני הינו תחום אשר נחקר רבות, זהו ריבוע של n שורות ו-n עמודות, שבכל שורה ובכל עמודה שלו כתובים אותם n סמלים שונים, בלי חזרות. אולם, קיימות הרבה פחות אפשרויות לסודוקו מאשר ריבועים לטיניים, משום שפתרון סודוקו דורש אילוץ נוסף, אזורי הבלוקים.

מספר האפשרויות ללוחות סודוקו של 9X9 הוא 6,670,903,752,021,072,936,960. (6.67\*10^21 בקירוב). המספר המקסימלי של נתונים שניתן לספק מבלי שהפתרון יקבע ביחידות הוא גודל הלוח עצמו פחות 4 (במקרה של לוח 9X9 זה 77). המספר המינימלי של נתונים שיש לספק על מנת שהפתרון יקבע ביחידות הוא 17 נתונים.

בעיית הסודוקו היא בעיה NP – שלמה, ולא קיים פתרון אלגוריתמי יעיל לפתרון סודוקו. לכן אלגוריתם מקובל לפתרון סודוקו הוא על ידי backtracking, כלומר מעבר רקורסיבי על כל האפשרויות. נפרט על אלגוריתם ה-backtracking בפרק 3.

איור - סודקו

1. **Backtracking**

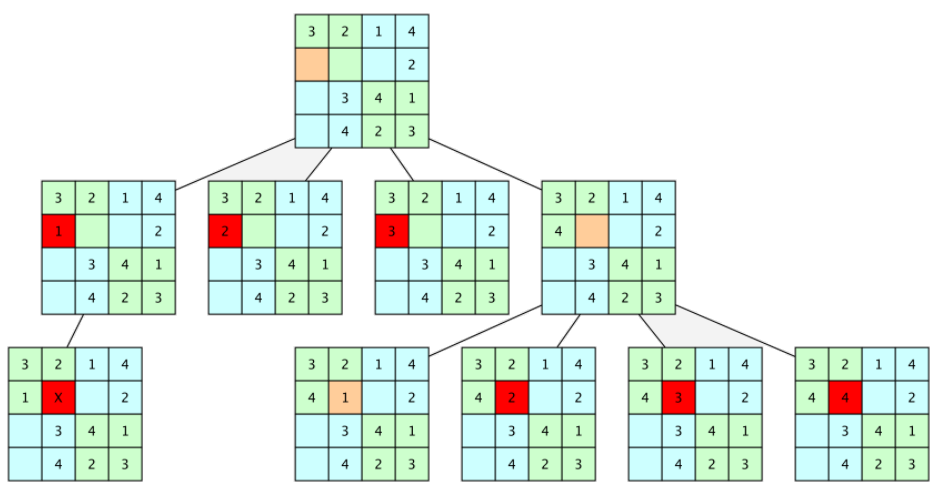
Backtracking (בעברית "עקיבה לאחור") הוא אלגוריתם אשר מבצע מעבר באופן רקורסיבי על כל האפשרויות של בעיה כלשהי במטרה למצוא חלק מהפתרונות או את כל הפתרונות של הבעיה. אלגוריתם זה למעשה מבצע חיפוש שחוסך מעבר על מספר רב של מועמדים לפתרון על ידי שימוש בתכונות ספציפיות של הבעיה. כלומר, האלגוריתם בונה באופן הדרגתי מועמדים לפתרון, וזונח מועמדים לפתרון אשר בהכרח לא יכולים להיות פתרון תקין לבעיה.

אלגוריתם backtracking מיועד עבור מציאת פתרון לבעיות שניתן לפתור בהן פתרונות חלקיים. דוגמא נפוצה לכך היא בעיה שבה יש מספר משתנים, ולכל משתנה צריך להתאים ערך מסוים כך שיתקיימו מספר אילוצים, כמו למשל בעיות סודוקו, תשבצים, תשחצים וכו'.

אלגוריתם backtracking לוקח בדרך כלל זמן ריצה רב, משום שסיבוכיות הריצה שלו גבוהה מאוד. על כן, בדרך כלל נעשה שימוש באלגוריתם זה כאשר אין פתרון יעיל אחר לבעיה.

ניתן לתאר את בעיית ה- backtrackingכעץ החלטות שבו השורש הוא המצב ההתחלתי (הבעיה ההתחלתית) והעלים של העץ הן הפתרונות (כולל פתרונות לא נכונים). כל צומת בעץ הוא פתרון חלקי, וקיים קשת מצומת א' לצומת ב' רק אם ניתן להגיע מפתרון חלקי א' לפתרון חלקי ב' בצעד אחד. הפתרון פועל בצורה הבאה: האלגוריתם מתחיל מהשורש ובכל פעם מתבצע האלגוריתם על כל אחד מהילדים של השורש בעץ. אם מגיעים לצומת בעץ שמייצגת פתרון שלא אפשרי אז חוזרים אחורה ולא ממשיכים. באופן זה ניתן לפסול פתרונות רבים מבלי לבדוק אותם.

ניתן כעת דוגמא לפתרון בעיית הסודוקו באמצעות אלגוריתם backtracking. אלגוריתם backtracking זוהי הדרך הפשוטה ביותר לפתור את בעיית הסודוקו. עוברים בשיטה זו על כל האפשרויות באופן הבא: בוחרים מקום ריק כלשהו על הלוח שלא מופיע שם ספרה, וממקמים במקום זה את ספרה חוקית כלשהי, ואז עוברים למיקום הריק הבא על הלוח באופן דומה עד לפתרון הבעיה. אם האלגוריתם לא מצא באחד השלבים ספרה חוקית לשים באחד המקומות הריקים, אז הפתרון לא חוקי, ויש לחזור אחורה ולנסות אפשרויות אחרות. כפי שצוין קודם לכן, שיטה זו בעלת סיבוכיות זמן גבוהה מאוד.

****

1. **למידה עמוקה** 
   1. **רקע כללי**

למידת עמוקה היא חלק מגוון שיטות בלמידת מכונה המבוססות על רשתות נוירונים מלאכותיות, אשר משתמשות בשכבות מרובות על מנת לחלץ מאפיינים ממידע גולמי. למשל, בעיבוד תמונה, השכבות הראשונות ברשת יזהו מאפיינים כמו פינות, ואילו השכבות המתקדמות יזהו מאפיינים בעלי משמעות אנושית כמו זיהוי ספרות, פנים וכדומה.

רשת נוירונים עשויה לשמש הן לרגרסיה (קירוב פונקציה בעלת טווח יציאה רציף) והן לסיווג. תהליך הלימוד מסתמך על פונקציית השגיאה. נסמן ב-y את התגית הנכונה של הדוגמא וב-o את מוצא הרשת (תגית החיזוי).

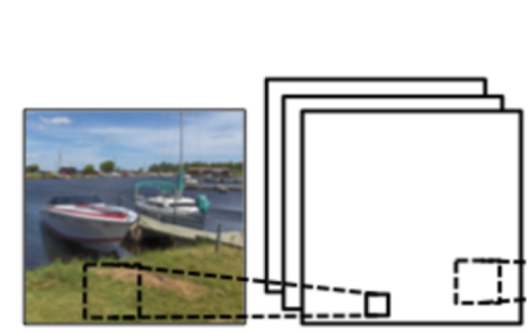
קיימות פונקציות שגיאה שונות כגון:

1. פונקציית השגיאה הריבועית- המציגה את סכום ההפרשים בריבוע בין תגית החיזוי לתגית הנכונה של הדוגמא, .
2. פונקציית השגיאה היא פונקציית "האנטרופיה הצולבת", .

האופטימיזציה של הרשת נעשית באמצעות אלגוריתם back-propagation, אשר משמש בעיקר לאימון של רשתות נוירונים בשילוב עם שיטות אופטימיזציה מבוססות גרדיאנט. כאשר דוגמא חדשה מוצגת לרשת, היא מפעפעת דרכה עד לשכבה האחרונה (forward pass). התוצאה המתקבלת מושווית לתוצאה הרצויה דרך פונקציית הפסד כלשהי לקבלת השגיאה. בשלב זה מתבצע ה-propagation-back: השגיאה מפעפעת אחורה באמצעות הגראדינט, מהשכבה האחרונה אל הראשונה (כאר האלגרויתפ מתבסס על עקרון "כלל השרשרת"). בצורה זו ניתן לחשב את הגרדיאנט של השגיאה ביחס לכל אחד מהמשקולות הנלמדים על הרשת.

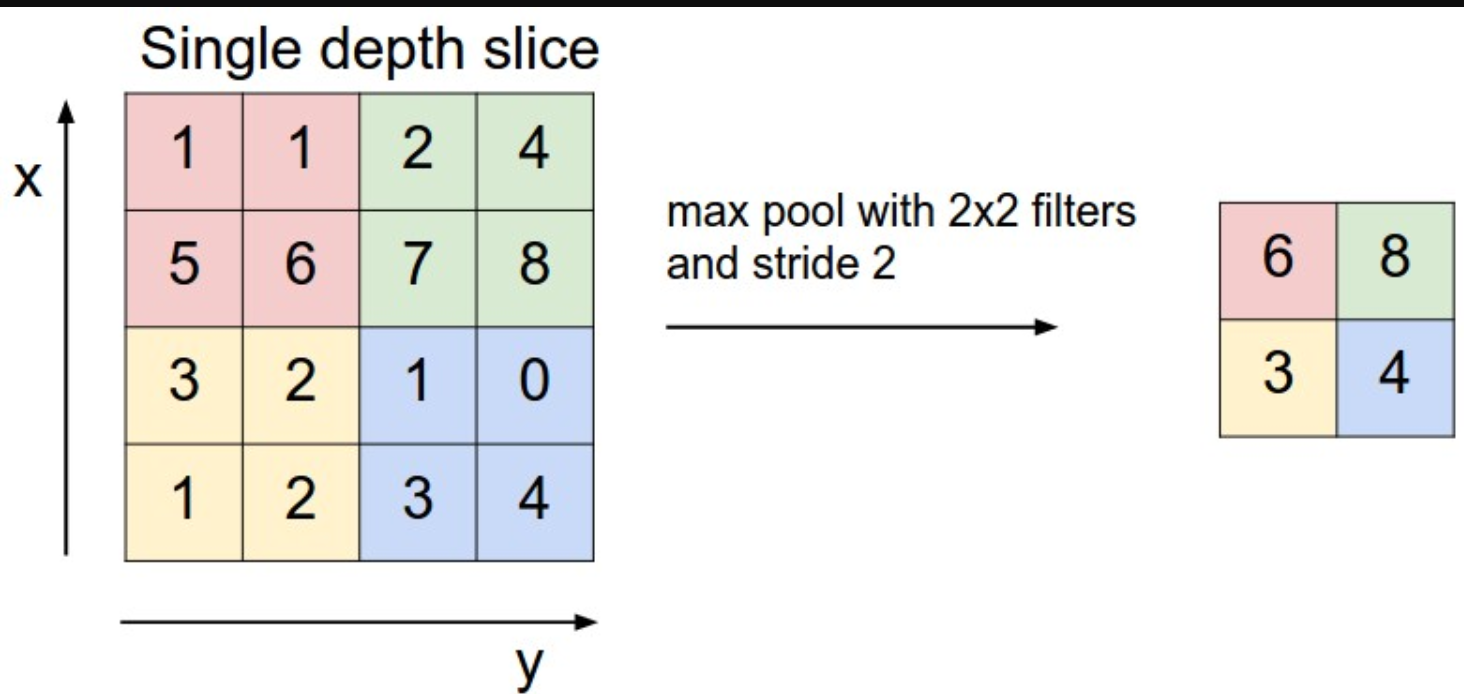
אלגוריתם SGD (Stochastic Gradient Descent) מנסה למזער את פונקציית השגיאה, והוא מעדכן כל משקל עבור כל דוגמא ע"י החסרת הגרדיאנט מוכפל בגודל צעד כלשהו. קיימים אלגוריתמים נוספים נמסים למזער את פונקציית השגיאה, כמו ADAM, אשר קובע גודל צעד שונה לכל משקל בהתאם לגרדיאנטים הראשונים המתקבלים עבורו. בשונה מ-SGD, זהו אלגוריתם מסתגל.

רשתות קונבולוציה ( CNN – Convolutional Neural Networks) הן רשתות נוירונים עמוקות שמשתמשות בשכבות קונבולציה. השכבות הטיפוסיות בהן הן:

שכבת קונבולוציה – הפרמטרים של השכבה מורכבים מאוסף של מסננים נלמדים, אשר יש להם שדה קלט (receptive field) קטן, אך מתרחבים לעומק הקלט. עם ההתקדמות של המסנן, הוא עובר קונבולוציה עם החלק המתאים בתמונה, והרשת לומדת מסננים אשר מבחינים במאפיין ספציפי במרחב מסוים בתמונה. רשתות קונבולוציה פופלריות מאוד בתחום ראייה ממוחשבת ובהן נשתמש בפרויקט שלנו.

איור 24. דוגמא לשכבת קונבולציה בתמונה

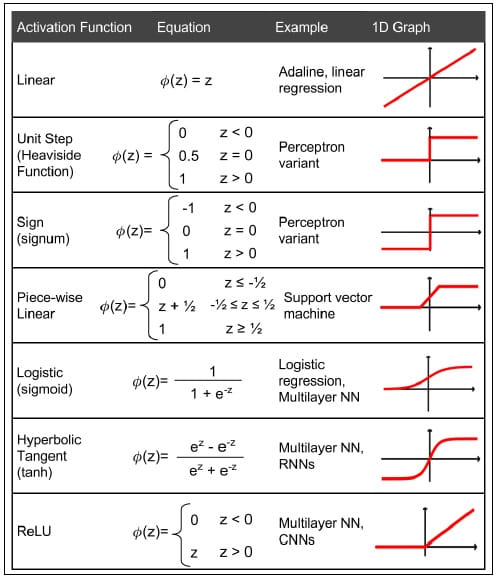
שכבת pooling – שכבה שתפקידה להוריד את הרזולוציה המרחבית של התמונה. השיטה הנפוצה ביותר היא באמצעות max pooling, המחלקת את התמונה לאוסף של חלקים לא חופפים, וכפלט מוציאה את המקסימום של כל חלק. שכבה זו מסייעת בהורדת כמות הפרמטרים במערכת וכך מקלה על העומס החישובי של אימון הרשת, וכן עוזרת למנוע התאמת היתר (overfitting).



איור 25. דוגמא לשכבת pooling

שכבת ReLU – שכבת האקטיבציה אשר הקלט שלה הוא . היתרון שלה על פונקציות אקטיבציה אחרות, כמו טנגנס היפרבולי או פונקצית הסיגמואיד

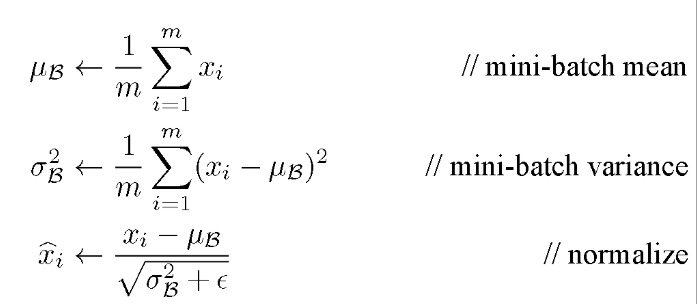
(), הוא שאין לה חלק בו הפונקציה נכנסת לרוויה והגרדיאנטים מתאפסים, דבר המאיץ את התכנסות האלגוריתם. פונקציית האקטיבציה ReLU נפוצה מאוד בארכיטקטורות של למידה עמוקה, וגם אנחנו בפרויקט זה השתמשנו בפונקציה זו על מנת ללמד את הרשת.



איור26 . דוגמאות לפונקציות אקטיבציה

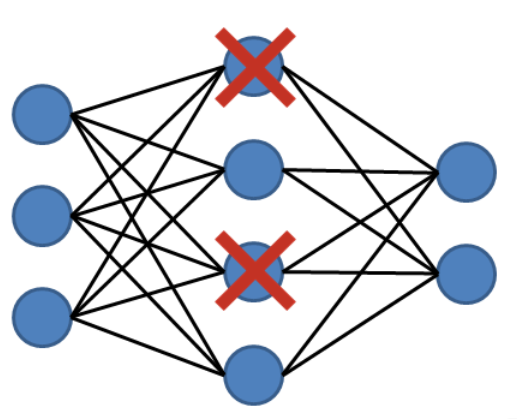
שכבת batch normalization – שכבה המנרמלת את הפלט של השכבה שקדמה לה ע"י החסרת ממוצע ה-batch של הנתונים וחלוקה בסטיית התקן שלו. שכבה זו מאיצה את תהליך הלימוד, מכיוון שהגרדיאנטים של פונקציית השגיאה כפונקציה של הפרמטרים השונים ברשת אינם נכנסים לרוויה כתוצאה מפונקציית האקטיבציה.

כך עבור קלט וגודל batch של m דגימות - פעולת הנרמול היא:

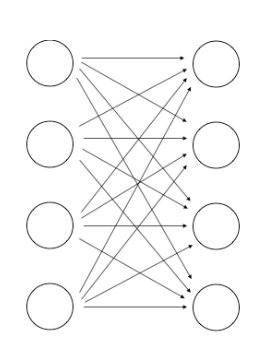


לאחר הנרמול הקלט הוא .

שכבת dropout – שכבה המבצעת רגולריזציה להורדת התאמת היתר ברשת ע"י כיבוי כל נוירון בהסתברות מסוימת הניתנת לשכבה כפרמטר.



איור 27. דוגמאות לשכבת dropout

שכבת fully connected – נמצאת בדרך כלל בסוף הרשת. הנוירונים בשכבה זו מקושרים לכל הפלטים בשכבה הקודמת, והם מפעילים התמרה אפינית באמצעות כפל מטריצי על הקלט שלהם.

איור 28. דוגמא לשכבת fully connected המקושרת לשכבה שקדמה לה.

שכבת ה-loss – שכבה זו היא למעשה פונקציית השגיאה. פונקציית השגיאה גדולה יותר ככל שקיימים הבדלים בין תיוגי האמת של הקלט לעומת התיוגים המתקבלים מהרשת. דוגמאות לפונקציות שגיאה:

1. **סקר ספרות**

5.1. Curriculum learning

במאמר [1] מוסבר על אימון רשתות באמצעות למידה עמוקה על ידי שיטת Curriculum learning. במאמר מוסבר כי בדרך כלל כאשר מאמנים רשתות על ידי למידה עמוקה, האימון נעשה באמצעות שימוש ב-mini-batches, אשר מוגרלים באופן אקראי מתוך סט האימון. אולם בשיטת Curriculum learning, ה-mini-batches לא מוגרלים באופן אקראי אלא באופן ממוין לפי רמת קושי. כלומר כדי לאמן באמצעות Curriculum learning, צריך למיין את סט האימון לפי רמות קושי שונות, וצריך להגריל את ה-mini-batches לפי רמות הקושי בסדר עולה.

במאמר מוסבר כי כפי שמורים מלמדים תלמידים משימות מסובכות על ידי שימוש ב"תכנית לימודים" (Curriculum באנגלית), כך גם שימוש ב"תכנית לימודים" עבור אימון רשתות עשוי לתרום לאימון הרשת ולהביא לתוצאות טובות יותר. הכוונה ב"תכנית לימודים" היא למידה על פי רמת קושי עולה, דבר אשר מאפשר להשתמש בידע שנלמד ברמת קושי קלה יותר כדי ללמוד בצורה טובה יותר בעיה קשה יותר.

במאמר ביצעו את מיון סט האימון לפי רמות הקושי השונות על ידי שימוש בשתי שיטות שונות:

1. שיטה אחת היא שימוש ב-transfer learning מרשת אחרת:

בשיטה זו מיון סט האימון לפי רמות קושי נעשה לפי הדיוק של רשת מאומנת אחרת על סט זה. כלומר אם הרשת האחרת הצליחה היטב דוגמאות מסוימות, אז דוגמאות אלו נחשבות "קלות", ואם הרשת נכשלה מאוד עבור דוגמאות מסוימות במהלך האימון אז דוגמאות אלו נחשבות "קשות". איכות ההצלחה של הרשת נקבעת לפי פונקציית scoring אשר נותנת דירוג למידת ההצלחה של הרשת עבור כל דוגמא. שיטה זו עשויה לשפר את מהירות ההתכנסות, ואת הדיוק הסופי עבור רשתות CNN.

1. שיטה נוספת למיון סט האימון היא שימוש ב-bootstrapping:

בשיטה זו נעשה אין צורך בשימוש ברשת אחרת, אלא נעשה שימוש ב- bootstrappingעל מנת למיין את סט האימון לדרגות קושי שונות. בשיטה זו, דרגות הקושי נקבעות לפי התוצאה שמקבלת הרשת שהתאמנה ללא Curriculum על דוגמאות אלו. במידה והרשת שהתאמנה ללא Curriculum הצליחה היטב דוגמאות מסוימות, אז דוגמאות אלו נחשבות "קלות", ולהפך – אם הרשת נכשלה מאוד עבור דוגמאות מסוימות במהלך האימון אז דוגמאות אלו נחשבות "קשות". איכות ההצלחה של הרשת נקבעת לפי פונקציית scoring אשר נותנת דירוג למידת ההצלחה של הרשת עבור כל דוגמא.

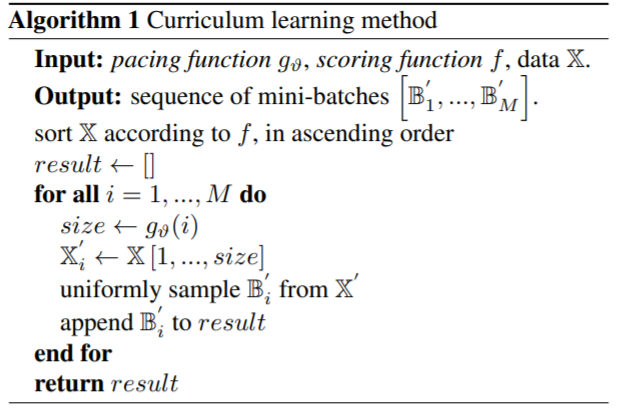
בנוסף, במאמר זה בדקו את "קצב הלימוד" שבו צריך ללמד את הרשת, כלומר באיזה קצב לעלות את רמת הקושי של הדוגמאות. ביצעו זאת על ידי שימוש בפונקציות pacing (קצב) שונות, אשר קובעות מהו הקצב שבו ה-data מוצג לרשת.

יתר על כן, במאמר זה מראים כי שימוש ב- Curriculum learningמשנה את האופטימיזציה של הרשת, וגורם להתכנסות להיות מהירה יותר, אך יחד עם זאת שומרת על המינימום הגלובאלי של הבעיה המקורית.

במאמר בדקו את הדברים השונים על סט האימון CIFAR-10, CIFAR-100 ו-ImageNet.

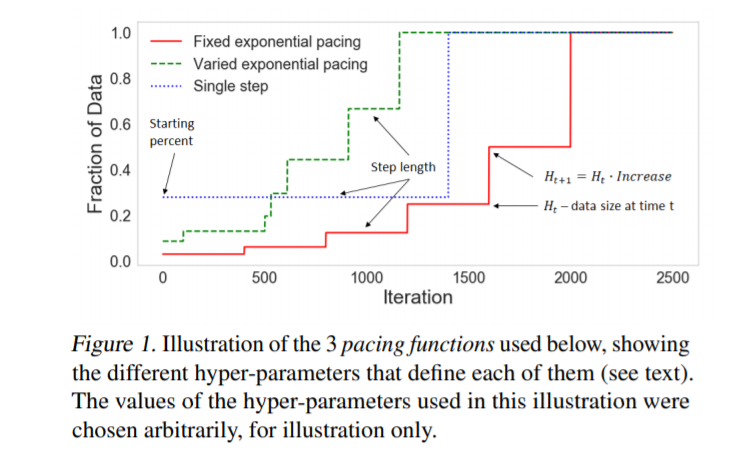
אלגוריתם ה-learning Curriculumשהוצג במאמר הוא כדלקמן:

הקלט הוא הדאטה, פונקציית ה-pacing ופונקציית ה-scoring. ממיינים את הדאטה X לפי סודר עולה של רמת קושי. רמת הקושי נקבעת לפי פונקציית ה-scoring. לאחר מכן, לוקחים מתוך הדאטה X רק את הדאטה עד רמת קושי מסוימת שנקבעת לפי פונקציית ה-pacing. לאחר מכן, מחלקים את הדאטה הנבחר ל-mini-batches. הפלט של האלגוריתם הוא ה-mini-batches בהתאם לפונקציות ה-pacing וה-scoring.



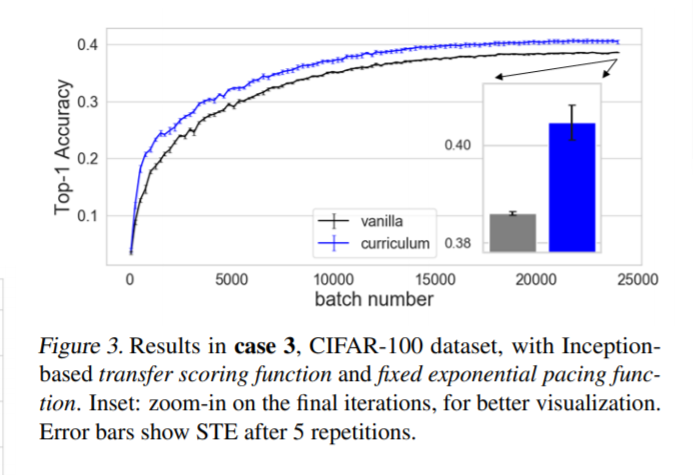
איור ???: אלגוריתם ה-learning Curriculumשהוצג במאמר [1].

נתבונן באיור [???] על 3 פונקציות pacing שהוצגו לדוגמא במאמר. הערכים שנבחרו עבור הפונקציות בדוגמא הם שרירותיים. ההבדל בין fixed exponential pacing לבין varied exponential pacing זה שעבור fixed exponential pacing מספר האיטרציות בכל צעד הוא קבוע. Single step זוהי פונקציית pacing אשר בעלת צעד אחד בלבד.



איור ???: דוגמא ל-3 פונקציות pacing שהוצגו במאמר.

נתבונן באיור [???] על התוצאות מתוך המאמר עבור Top-1 Accuracy על סט הולדיציה במהלך האימון עבור dataset של CIFAR-100 תוך שימוש ברשת קונבולוציה פשוטה שהוכנה ידנית. ניתן לראות שהדיוק עבור curriculum גבוה יותר.



איור ???: תוצאות עבור Top-1 Accuracy על סט הולדיציה במהלך האימון עבור dataset של CIFAR-100.

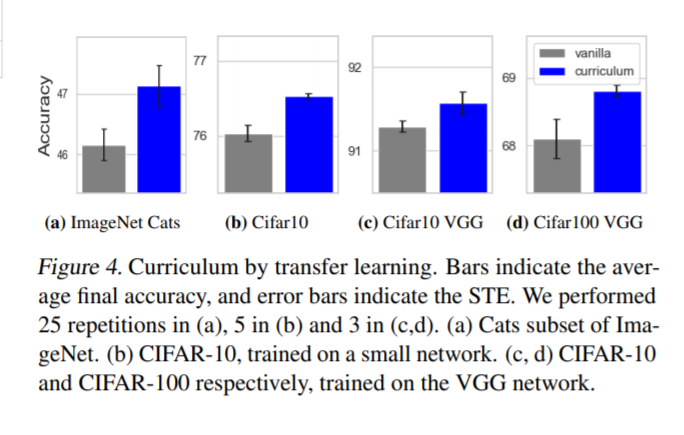
כמו כן, נתבונן באיור [???] על תוצאות עבור Top-1 Accuracy על סט המבחן עבור datasets שונים ורשתות שונות. ה-datasets והרשתות השונות שהשתמשו בהם הם כדלקמן (משמאל לימין):

Dataset - של ImageNet Cats באמצעות רשת קונבולוציה פשוטה שהוכנה ידנית.

Dataset - שלCIFAR-10 באמצעות רשת קונבולוציה פשוטה שהוכנה ידנית.

Dataset - שלCIFAR-10 באמצעות רשת VGG.

Dataset - שלCIFAR-100 באמצעות רשת VGG.



איור ???: תוצאות עבור Top-1 Accuracy על סט המבחן עבור datasets שונים ורשתות שונות.

יש לציין כי המושג Vanilla שמצוין באיור [???] זוהי שיטת האימון הרגילה שמשתמשים בה ללא Curriculum. ניתן לראות שהדיוק עבור curriculum גבוה יותר.

5.2. דוגמא לפתרון מאתר Kaggle [2]

בדוגמא זאת [2] מופיע dataset שהשתמשנו בו עבור הפרויקט שלנו. dataset זה מכיל מיליון לוחות ברמות קושי שונות, ??? ברק עושה פילוג של המחוקים???

הקוד שנכתב כאם הוא ב-keras ואילו אנחנו העדפנו לכתוב בפרויקט שלנו ב-pytorch.

באלגוריתם באתר Kaggle תחילה מתבצע "כיול": מתבצע אימון לרשת עם 0 מספרים מחוקים, כלומר הרשת מקבלת לוח מלא, ומאמנים את הרשת להוציא לוח מלא זהה.

לאחר מכן, מוחקים עבור כל לוח בסט האימון ספרה אחת בלבד, ומאמנים את הרשת למצוא את המספר הזה.

לאחר מכן, מוחקים שני מספרים ומאמנים את הרשת למלא את הלוח החסר. באופן זה, ממשיכים למחוק ספרות נוספות ובכל פעם מאמנים את הרשתות למלא את הלוח עבור מספר מחוקים הולך וגדל. באופן זה מאמנים את הרשת באופן הדרגתי בכל פעם למלא לוחות בדרגות קושי הולכות וגדלות (Curriculum learning). ממשיכים לאמן באופן זה עד 55 מספרים מחוקים. כאשר בכל שלב ניתן למחוק יותר מספרה אחת.

מספר האיברים המחוקים בכל שלב מתבצע באופן הבא לפי הסדר:

* Epoch אחד של איבר אחד מחוק.
* 2 epochs של 2 איברים מחוקים.
* 3 epochs של 3 איברים מחוקים.
* 4 epochs של 4 איברים מחוקים.
* 6 epochs של 6 איברים מחוקים.
* 8 epochs של 8 איברים מחוקים.
* 10 epochs של 10 איברים מחוקים.
* 12 epochs של 10 איברים מחוקים.
* 15 epochs של 10 איברים מחוקים.
* 20 epochs של 10 איברים מחוקים.
* 25 epochs של 10 איברים מחוקים.
* 30 epochs של 10 איברים מחוקים.
* 35 epochs של 10 איברים מחוקים.
* 40 epochs של 10 איברים מחוקים.
* 45 epochs של 10 איברים מחוקים.
* 50 epochs של 10 איברים מחוקים.
* 55 epochs של 10 איברים מחוקים.

בסט המבחן ממלאים את הלוח בכל פעם ספרה אחת בלבד. כלומר ממלאים את הלוח איבר-איבר ובכל פעם ממלאים את האיבר עם ההסתברות הגבוהה ביותר, כלומר האיבר שהרשת בטוחה בערך שלו במידה הרבה ביותר.

התוצאה אליה הגיעו היא 99.7% אבל זוהי תוצאה מטעה מכיוון שמספר האיברים המחוקים בסט המבחן הוא:

???לרשום התפלגות של מספר האיברים המחוקים בסט המבחן??

כלומר יחסית יש מעט איברים מחוקים בסט המבחן, ולכן הרשת מגיעה לתוצאות טובות יחסית. ככל שיש פחות איברים מחוקים, רמת הקושי של הלוח היא קלה יותר ולכן הרשת תטעה פחות ותגיע לתוצאות טובות יותר.

הרשת שבה השתמשו בנויה באופן הבא:

1. שכבת FC עם 64 נוירונים
2. שכבת ReLU
3. שכבת dropout עם הסתברות של 0.4
4. שכבת FC עם 64 נוירונים
5. שכבת ReLU
6. שכבת dropout עם הסתברות של 0.4
7. שכבת SoftMax

ה-loss שבו השתמשו הוא Categorial\_crossentropy. נשים לב שה-loss שבו נעשה שימוש בקוד זה הוא על כל המספרים בלוח ולא רק על המספרים המחוקים. כלומר, כאשר מחשבים את הloss- אז נלקחים בחשבון כל האיברים בלוח, גם איברים שהופיעו בלוח בהתחלה ולא היו מחוקים.

ה-optimizer שבו השתמשו הוא "Adam optimizer".

גודל ה-batch הוא: 128 לוחות.

להוסיף לסוף:

<https://www.kaggle.com/dithyrambe/neural-nets-as-sudoku-solvers>

1. **הפתרון בפרויקט שלנו – תיאור כללי**



**?. סיכום ומסקנות**

**?. נספחים**

**?. ביבליוגרפיה**