

**דוח פרויקט:** סמסטריאלי

**הנושא:**

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

מגישים:

חן דודאי ברק ממיסטבלוב

מנחה:

רון דורפמן

תום יורגנסון

**סמסטר:** חורף

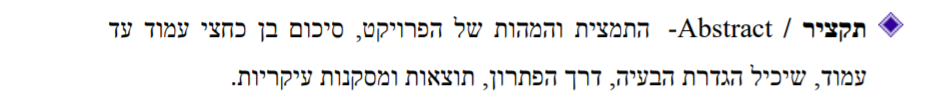
**שנה:** 2020

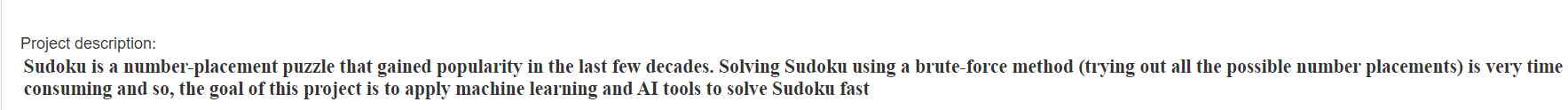
**תוכן עניינים**

**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.

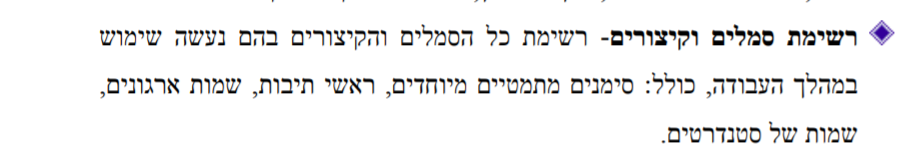




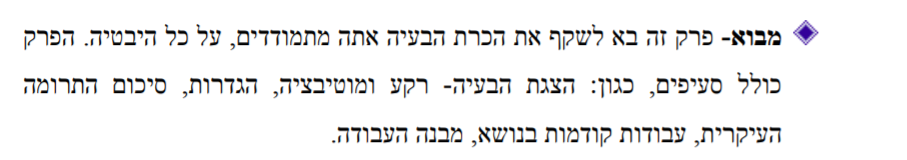
**Abstract**

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**

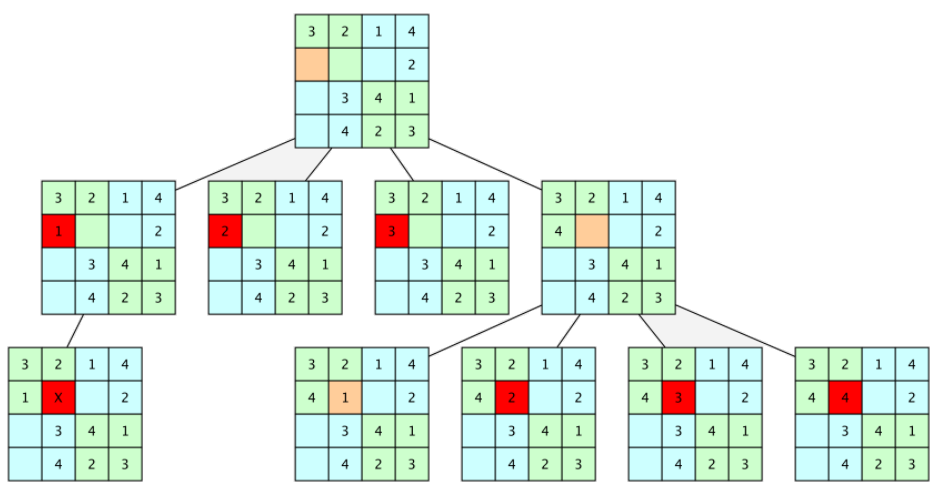
Backtracking (בעברית "עקיבה לאחור") הוא אלגוריתם אשר מבצע מעבר באופן רקורסיבי על כל האפשרויות של בעיה כלשהי במטרה למצוא חלק מהפתרונות או את כל הפתרונות של הבעיה. אלגוריתם זה למעשה מבצע חיפוש שחוסך מעבר על מספר רב של מועמדים לפתרון על ידי שימוש בתכונות ספציפיות של הבעיה. כלומר, האלגוריתם בונה באופן הדרגתי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

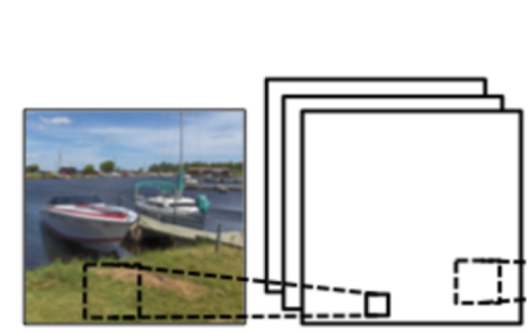
1. פונקציית השגיאה הריבועית- המציגה את סכום ההפרשים בריבוע בין תגית החיזוי לתגית הנכונה של הדוגמא, .
2. פונקציית השגיאה היא פונקציית "האנטרופיה הצולבת", .

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.

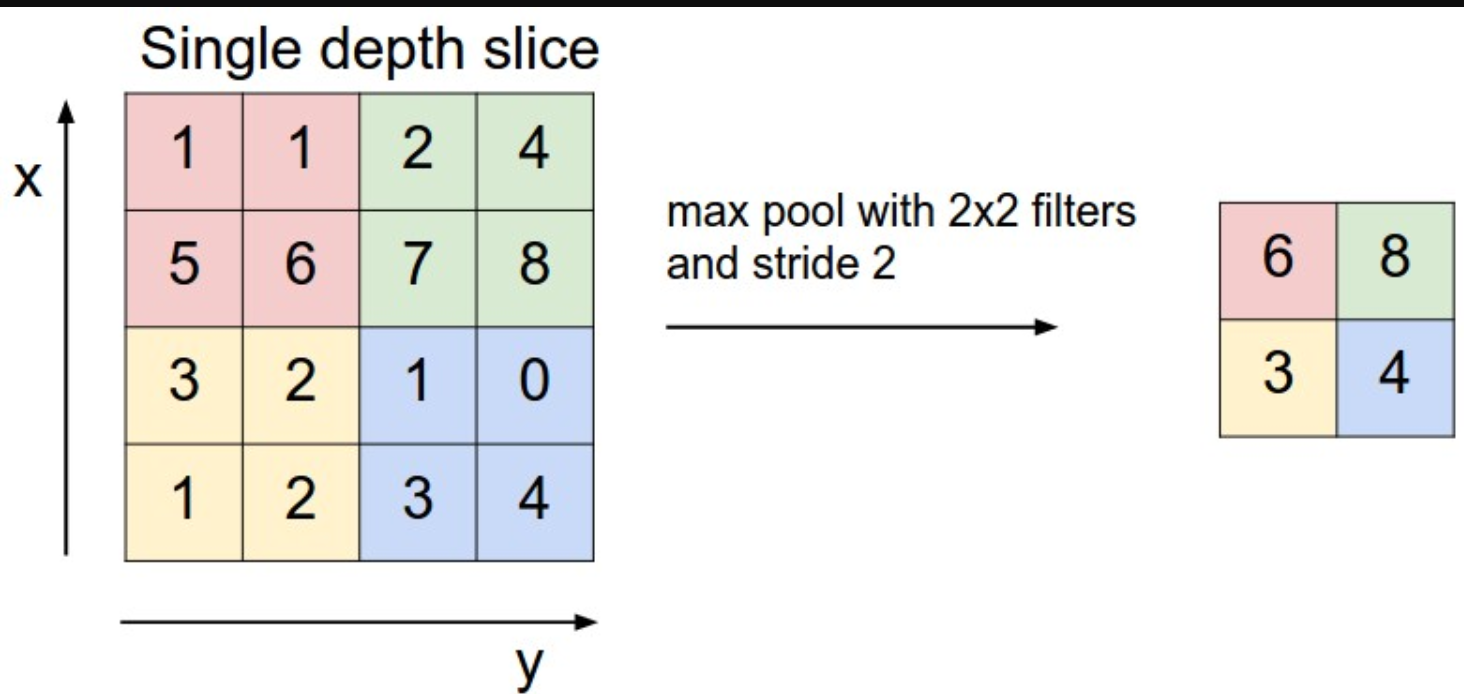
רשתות קונבולוציה ( CNN – Convolutional Neural Networks) הן רשתות נוירונים עמוקות שמשתמשות בשכבות קונבולציה. השכבות הטיפוסיות בהן הן:

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.



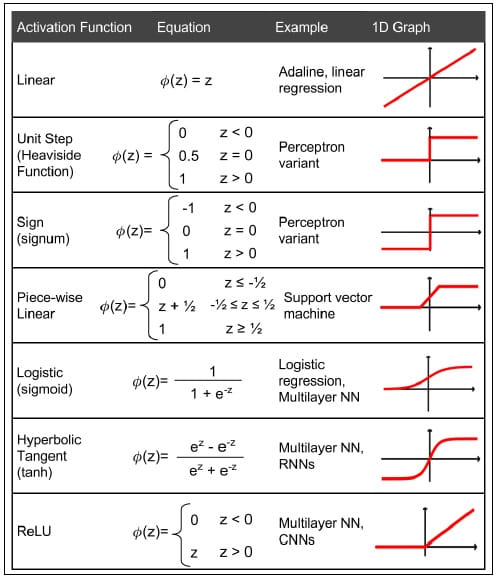
איור 24. דוגמא לשכבת קונבולציה בתמונה

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

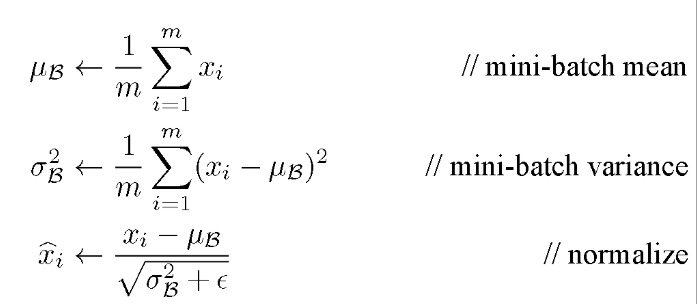
שכבת ReLU – שכבת האקטיבציה אשר הקלט שלה הוא . היתרון שלה על פונקציות אקטיבציה אחרות, כמו טנגנס היפרבולי או פונקצית הסיגמואיד (), הוא שאין לה חלק בו הפונקציה נכנסת לרוויה והגרדיאנטים מתאפסים, דבר המאיץ את התכנסות האלגוריתם. פונקציית האקטיבציה ReLU נפוצה מאוד בארכיטקטורות של למידה עמוקה, וגם אנחנו בפרויקט זה השתמשנו בפונקציה זו על מנת ללמד את הרשת.



איור26 . דוגמאות לפונקציות אקטיבציה

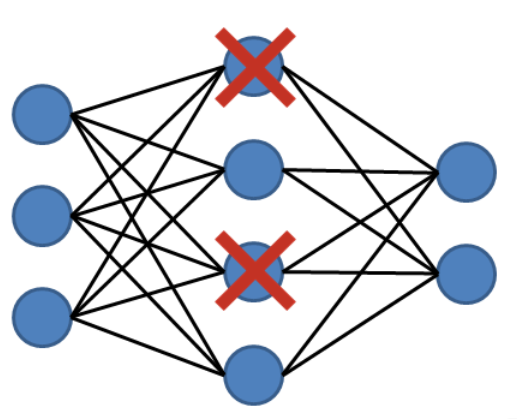
שכבת batch normalization – שכבה המנרמלת את הפלט של השכבה שקדמה לה ע"י החסרת ממוצע ה-batch של הנתונים וחלוקה בסטיית התקן שלו. שכבה זו מאיצה את תהליך הלימוד, מכיוון שהגרדיאנטים של פונקציית השגיאה כפונקציה של הפרמטרים השונים ברשת אינם נכנסים לרוויה כתוצאה מפונקציית האקטיבציה.

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

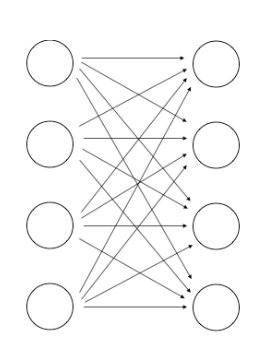


לאחר הנרמול הקלט הוא .

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. Curriculum learning
2. **סקר ספרות**
3. **הפתרון בפרויקט שלנו – תיאור כללי**



**?. סיכום ומסקנות**

**?. נספחים**

**?. ביבליוגרפיה**