מימוש קירוב פולינומיאלי ברשתות נוירונים

דוח פרויקט מתקדם זה הוגש כחלק מהדרישות לקבלת תואר

מוסמך למדעים .M.Sc במדעי המחשב

באוניברסיטה הפתוחה

החטיבה למדעי המחשב

על-ידי

קונסטנטין סוקרניק

העבודה הוכנה בהדרכתם של **פרופסור אהוד גודס ומר פיליפ דרבקו**

ספטמבר 2021

תוכן עניינים

רשימת איורים

1. תקציר

רשתות נוירונים משתמשות בפונקציות אקטיבציה (activation functions) ,פונקציית אקטיבציה של צומת מגדירה את הפלט של אותו צומת בהינתן קלט, בין הנפוצות שבהן אפשר למצוא את:

Relu: $f(x) = \max(0, x)$

Sigmoid: $f(X) = \frac{1}{1+e^{-x}}$

פונקציות מסוג זה ניתן לקרב על ידי פולינומים מדרגות שונות בשלל שיטות כמו פולינומי צ'בישב[2] , שיטת הקירובים הפשוטים ובעזרת נגזרות [3] שזאת שיטה מעניינת שלפיה מקרבים את הנגזרת של הפונקציה ואז מבצעים אינטגרציה לקירוב.

לכל אחת משיטות הקירובים יש יתרונות וחסרונות בדמות הדיוק וזמן הריצה, אנו נבחן אותם על מנת לבחור את השיטה המתאימה ביותר למטרה שלנו.

המטרה העיקרית בפרויקט הזה היא ייצוג פונקציית אקטיבציה כפולינום, לפולינומים יש סכימות לשיתוף סוד בהן הסוד הוא מקדמי הפולינום וניתן להפעיל את הסכימה הנבחרת לכל מספר של משתתפים.
בסכימה כזאת ניתן להבטיח סודיות ושאף צד לא יוכל לחשב את ערך הפולינום בנקודה כלשהי על ידי שיטות של חישוב מאובטח מרובה משתתפים כך שכל אחד מהמשתתפים מחזיק רק בחלק מהמידע וכולם צריכים לשתף פעולה על מנת להגיע לתוצאה.

הפרויקט ברובו יתבסס על הפרק החמישי במאמר [1], פרק זה מתאר כיצד ספק מידע (במקרה זה רשת נוירונים) יכול לשלוט על כמות השאילתות שהולכים להריץ על המודל וכמה שאילתות כל לקוח יוכל להריץ. נשתמש בקירוב פולינומיאלי ואז באמצעות חישוב מאובטח מרובה משתתפים נחלק את הפולינום למספר משתתפים, מקדמי הפולינום יהיו הסוד וכל המשתתפים יצטרכו לשתף פעולה על מנת לחשב את ערכו בנקודה.

חישוב מרובה משתתפים יבוצע באמצעות הפרוטוקול [4] בו כל אחד מהמשתתפים מחזיק מקדמי פולינום כך שסכומם הוא המקדמים האמיתיים אבל כל אחד מהם אינו יכול לבצע את החישוב בנקודה לבדו וזקוק למשתתפים האחרים.

בחלק האחרון של הפרויקט נראה איך אפשר לייצג רשת נוירונים שלמה בעזרת מספר פולינומים.

2. רקע

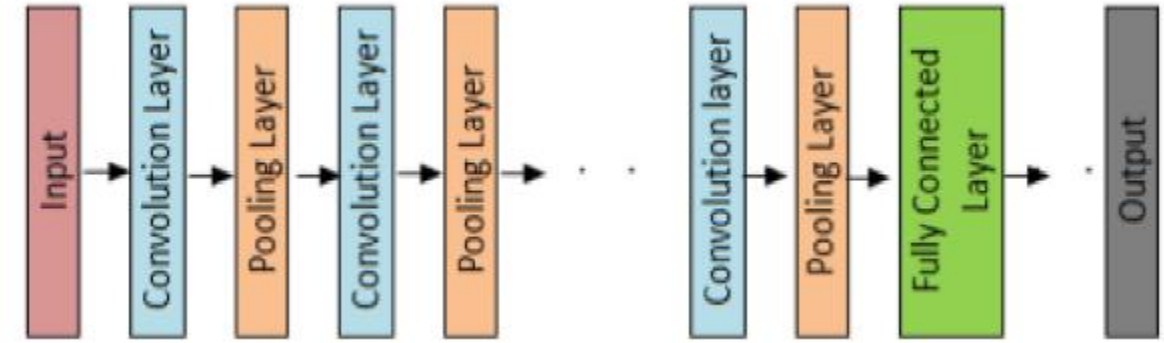
Deep learning (למידה עמוקה) הוא תת תחום של למידת מכונה העושה שימוש ברשתות נוירונים מלאכותיות לשם ביצוע משימות. בסיסו של תחום זה טמון בשאיפה לחקות את הדרך בה המוח האנושי פועל ולרתום את היעילות של מבנה הנוירונים לשם התמודדות עם אתגרים חישוביים מורכבים.

מדובר על מודל מתמטי חישובי המורכב ממספר "נוירונים" המסודרים בשכבות, כאשר כל נוירון יכול לתקשר עם מספר נוירונים אחרים במערכת .כל נוירון מסוגל לבצע פעולות חישוביות פשוטות, ובתורו להעביר את המידע שהסיק לשאר הנוירונים .באופן זה, עם ההתקדמות של המידע בשכבות, המערכת הופכת את המידע הגולמי למידע בעל ערך. כפועל יוצא מכך, המערכת לומדת לקבל החלטות מדויקות יותר.

2.1. רשתות נוירונים

קיימת ארכיטקטורה של למידה עמוקה הנקראת CNN (רשת נוירוניות קונבולוציונית). זוהי רשת רב שכבתית השואפת השראה מהתפיסה הויזואלית של יצורים חיים.
הארכיטקטורה נהפכה לפופולרית החל משנת 2012 אחרי הביצועים הטובים שהפגינה ב AlexNet[5], המקורות שלה מובילים אחורה עד לשנת 1980.

CNN מורכבת משכבת קלט, שכבות נסתרות ושכבת פלט. כל שכבות האמצע נקראות מוסתרות מכיוון שהקלטים והפלטים שלהן מוסוות על ידי פונקציית ההפעלה בשכבה הבאה. השכבות הנסתרות כוללות שכבות המבצעות קונבולוציה. בדרך כלל זה כולל שכבה שמבצעת מכפלה סקלרית (dot product) של קרנל הקונבולוציה עם השכבה של מטריצת הקלט. התוצאה היא בדרך כלל Frobenius inner product, ופונקציית האקטיבציה שלה היא בדרך כלל Relu.



איור 1: מבנה של CNN

מבנה של רשת CNN מתחיל בשכבת קלט, לאחר מכן יש שכבות של קונבולוציה ושכבות של Pooling לסירוגין עד שבסוף יש שכבה של Fully Connected ואז הפלט.
שכבה של קונבולוציה היא השיכבה העיקרית של הרשת, היא מכילה מספר קרנלים של קונבולוציה (פילטרים) שביחד עם הקלט מתקבלת feature map.
שכבה של pooling תפקידה לקחת את התוצאה של השכבה הקודמת (feature map) ולכווץ ל-feature map יותר קטן.
השכבה הלפני אחרונה (fully connected) היא פשוט שכבה שבה כל נוירון מחובר לכל נוירון בשכבה לפניה.

תפקידן של פונקציות אקטיבציה בכל רשת נוירונים הוא למפות את הקלט לפלט בצורה לא לינארית, כלומר להוסיף אי לינאריות לרשת. כפי שמפורט בצורה יותר מורחבת בנספח, חישוב של שכבה נסתרת ברשת כולל חישוב הכפלה במשקלים, הוספת ביאס ואז הפעלת פונקציית אקטיבציה לקבל הפלט של הניורון. פונקציית האקטיבציה בעצם מחליטה אם הניורון יהיה מופעל או לא.

בארכיטקטורה של CNN לאחר כל שכבת לימוד (כמו קונבולוציה או fully connected) יש שכבה לא לינארית של אקטיבציה. בזכות האי-לינאריות מודל ה CNN יכול ללמוד דברים יותר מורכבים ולמפות קלט לפלט בצורה לא לינארית. התכונה החשובה של פונקציית אקטיבציה שהיא תהיה דיפרנציאבילית, בפונקציה של משתנה אחד היא תהיה פשוט גזירה. תכונה זאת חשובה כדי שיהיה אפשר לעשות "פעפוע לאחור" ($backpropogation$) על מנת לאמן את המודל.

אלגוריתם הפעפוע לאחור הוא כנראה אבן הבניין הבסיסית ביותר ברשת נוירונים. הוא הוצג לראשונה בשנות ה-60 וכמעט 30 שנה לאחר מכן (1989) נהפך לפופולרי על ידי רומלהארט, הינטון ווליאמס במאמר בשם "Learning representations by back-propagating errors".
האלגוריתם משמש לאימון יעיל של רשת נוירונים באמצעות שיטה הנקראת כלל השרשרת. במילים פשוטות, לאחר כל מעבר קדימה דרך רשת, הפעפוע לאחור מבצע מעבר לאחור תוך התאמת הפרמטרים של המודל (משקלים והטיות).

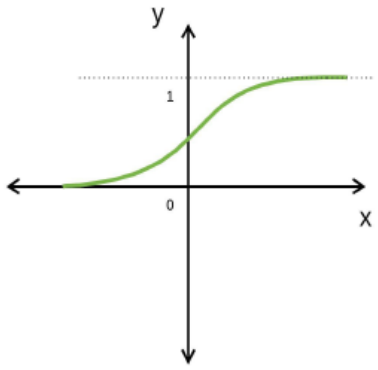
2.2. פונקציות אקטיבציה

ישנן כמה פונקציות אקטיבציה נפוצות שמשתמשים בהן, נראה סקירה קצרה של כל אחת ונתמקד בהמשך רק בשתיים: $Relu, Sigmoid$.

2.2.1. הפונקציה Sigmoid

פונקציה המוגדרת על ידי: $f(x) = \frac{1}{1+e^{-x}}$

הקלט של הפונקציה הוא כל הממשיים, הפלט בטווח של $[0,1]$, בעלת צורה של S.

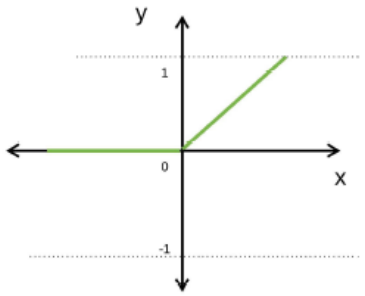


איור 2: הפונקציה sigmoid

2.2.2 הפונקציה Relu

פונקציה המוגדרת על ידי: $f(x) = \max(0, x)$

הפונקציה שנמצאת הכי הרבה בשימוש ברשתות מסוג CNN. היתרון שלה הוא שהיא מאוד פשוטה וממירה כל קלט למספר אי-שלילי, הקלט של הפונקציה הוא כל הממשיים והפלט בטווח $[0, \infty)$.



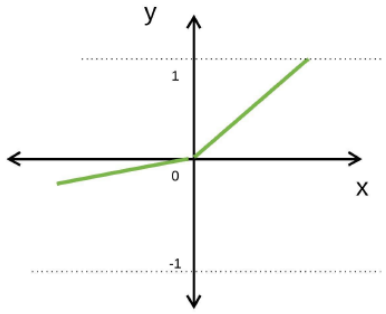
איור 3: הפונקציה Relu

החיסרון של הפונקציה הוא שעלול להיווצר מצב כזה שהמשקולות של נוירון מסוים יעודכנו בצורה כזאת שתמיד יקבלו מספרים שליליים והנוירון לא יהיה יותר פעיל אף פעם. בגלל מצבים כאלה המציאו וריאציות של הפונקציה.

2.2.3. הפונקציה Leaky Relu

פונקציה המוגדרת על ידי: $f(x) = \begin{cases} x & if\ x > 0 \\ mx & if\ x \leq 0 \end{cases}$

המטרה של הפונקציה היא לפתור את הבעיה של Relu הרגילה, m הוא קבוע קטן – לדוגמא 0.001.

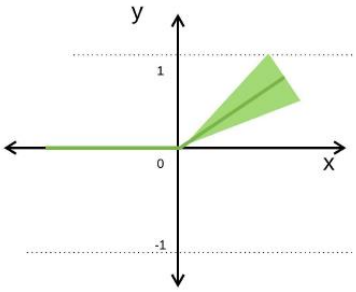


איור 4: הפונקציה Leaky Relu

2.2.4. הפונקציה Noisy Relu

פונקציה המוגדרת על ידי: $f(x) = \max(0, x + Y)$

כאשר $Y \sim \eta(0, \sigma(x))$ היא רעש גאוסיאני.



איור 5: הפונקציה Noisy Relu

2.3. קירוב פולינומיאלי

פולינום מסדר n הוא פונקציה מהצורה $p(x) = a_0 + a_1 x + \cdots + a_n x^n$, לפולינומים יש צורה מאוד פשוטה שבעזרתה קל מאוד לבצע פעולות כמו גזירה ואינטגרציה. תכונות אלה הופכות את הפולינומים למועמדים מושלמים לקירוב של פונקציות.

בהינתן סט נקודות במרחב מהצורה (x_i, y_i) עבור $i = 0,1,2, \dots, n$ המייצגות קלט ופלט של פונקציה כלשהי f המטרה בקירוב פולינומיאלי היא לייצר פולינום p שימזער את פונקציית ההפסד, בהרבה מהמקרים סכום ריבועי השגיאה: $\sum_{i=0}^{n}(p(x_i) - y_i)^2$.

2.3.1. קירוב צ'בישב

פולינום צ'בישב-[2] מסדר n נתון על ידי הפונקציה $T_n(x) = \cos(n \cdot arcos(x))$ בעזרת זהויות טריגונומטריות ניתן להגיע לתוצאה שאינה טריגונומטרית:

$$T_0(x) = 1$$

$$T_1(x) = x$$

$$T_2(x) = 2x^2 - 1$$

$$T_3(x) = 4x^3 - 3x$$

$$T_4(x) = 8x^4 - 8x^2 + 1$$

...

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

עבור פונקציה $f(x)$ בטווח $[-1,1]$, מחשבים וקטור של N מקדמים $c_o, \dots, c_{N-1}$ והנוסחה של הקירוב נתונה על ידי:

$$f(x) \approx \left[\sum_{k=0}^{N-1} c_k T_k(x)\right] - \frac{1}{2}c_0$$

בשביל לעבור מהטווח $[-1,1]$ לכל טווח $[a,b]$ מבצעים החלפת משתנה לקירוב $y \equiv \frac{x-\frac{1}{2}(b-a)}{\frac{1}{2}(b-a)}$

```python
import math

class Chebyshev:
    """
    Chebyshev(a, b, n, func)
    Given a function func, lower and upper limits of the interval [a,b],
    and maximum degree n, this class computes a Chebyshev approximation
    of the function.
    Method eval(x) yields the approximated function value.
    """
    def __init__(self, a, b, n, func):
        self.a = a
        self.b = b
        self.func = func

        bma = 0.5 * (b - a)
        bpa = 0.5 * (b + a)
        f = [func(math.cos(math.pi * (k + 0.5) / n) * bma + bpa) for k in range(n)]
        fac = 2.0 / n
        self.c = [fac * sum([f[k] * math.cos(math.pi * j * (k + 0.5) / n)
                  for k in range(n)]) for j in range(n)]

    def eval(self, x):
        a,b = self.a, self.b
        assert(a <= x <= b)
        y = (2.0 * x - a - b) * (1.0 / (b - a))
        y2 = 2.0 * y
        (d, dd) = (self.c[-1], 0)             # Special case first step for efficiency
        for cj in self.c[-2:0:-1]:            # Clenshaw's recurrence
            (d, dd) = (y2 * d - dd + cj, d)
        return y * d - dd + 0.5 * self.c[0]   # Last step is different
```

איור 6: קירוב צ'בישב בשפת פייתון

מימוש הקירוב מורכב משני שלבים, תחילה מחשבים את וקטור המקדמים c על ידי שערוך הפונקציה func ב-n נקודות. בשלב השני מקבלים נקודה x לקירוב, מבצעים החלפת משתנה ל-y ומחשבים את הקירוב בעזרת וקטור המקדמים c.

2.3.2. שיטת הקירובים הפשוטים

מודל פולינומי-[6] ב-n משתנים $(i = 1 \dots n)$ מסדר m נתון על ידי המשוואה:
$$y_i = b_0 + b_1 x_i + b_2 x_i + \cdots + b_m x^m + \varepsilon$$
ניתן לכתוב את המודל בצורה מטריציונית על ידי: $\vec{y} = X\vec{\beta} + \vec{\varepsilon}$.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ 1 & x_3 & x_3^2 & \dots & x_3^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

איור 7: מודל רגרסיה פולינומית כמערכת משוואות לינארית

ניתן לפתור את מערכת המשוואות הלינארית המופיעה באיור 7 על ידי: $\vec{\beta} = (X^T X)^{-1} X^T \vec{y}$

בהנחה ש $m < n$ א המטריצה $X^T X$ היא הפיכה בתנאי שכל ערכי x_i הם יחודיים.

בשפת פייתון קיים מימוש למודל הקירובים הפשוטים בספריית $numpy$ בפונקציה:

$$numpy.polyfit(x, y, deg, rcond = None, full = False, w = None, cov = False)$$

הפרמטרים שלה הם שני וקטורים של נקודות והערך של הפונקציה בנקודות אלו: x, y והסדר של הפולינום-deg.

2.3.3. קירוב על ידי נגזרת

השיטה במאמר-[3] מציעה במקום לקרב את הפונקציה, לקרב את הנגזרת שלה. ניקח לדוגמא את הפונקציה $Relu$, הנגזרת שלה היא פונקציית מדרגה של-0 בטווח $(-\infty, 0)$ ואז 1 בטווח $(0, \infty)$, הפוקציה אינה גזירה בנקודה אפס. ניתן לקרב בצורה יותר טובה פונקציה שהיא רציפה וגזירה אינסוף פעמים, במקום לקרב את הפונקציה $Relu$ ננסה לקרב את הנגזרת שלה – פונקציית מדרגה.
פונקציית $Sigmoid$ עונה להגדרות שאנחנו צריכים, היא חסומה, רציפה וגזירה אינסוף פעמים, בנוסף הצורה שלה מזכירה את הנגזרת של $Relu$ למעט בסביבה הקרובה של אפס.
המאמר מציע לקרב את הפונקציה $Sigmoid$ בעזרת פולינום, לחשב את האינטגרל של הפולינום ולהשתמש בו בתור פונקציית האקטיבציה.

נשתמש בפולינום טיילור כדי לקבל קירוב פולינומיאלי לפונקציית ה $Sigmoid$, לאחר מכן נשתמש בפונקצייה: $numpy.polyint(p, m = 1, k = None)$ עם הערך $m = 1$ על מנת לחשב את האינטגרל של הפולינום.

2.4. חישוב מאובטח מרובה משתתפים

חישוב מרובה משתתפים יתבסס על האלגוריתם שהוצג ב-[4]. הסכמה מבוססת על חלוקת סוד בין N משתתפים כך שגם אם $N-1$ משתתפים משתתפים פעולה הם עדיין לא יוכלו לשחזר את הסוד ללא המשתתף החסר.

Algorithm 1 Add.split procedure: given an element $s \in \mathbb{F}_p$, where $\mathbb{F}_p$ is a finite field containing p-primer elements, the procedure returns N additive secret shares whose sum is s.

Require: p - prime number, $s \in \mathbb{F}_p$ - a secret to share and $N \in \mathbb{N}$.

$\gamma_1, \ldots, \gamma_{N-1} \leftarrow$ - randomly chosen from $\mathbb{F}_P$

$\gamma_N \leftarrow s - \sum_{i=1}^{N-1} \gamma_i$

return $(\gamma_1, \ldots, \gamma_N)$ - a sequence of secret shares of s.

איור 8: אלגוריתם לחלוקת סוד בין N משתתפים

באיור 8 מוצג אלגוריתם לחלוקת סוד בין N משתתפים, האלגוריתם מבוסס על על בחירת $N-1$ מספרים רנדומליים ואז עבור המספר האחרון בוחרים את הסוד מינוס הסכום של המספרים שנבחרו עד כה.
בעזרת האלגוריתם בהינתן פולינום, נחלק את המקדמים שלו בין המשתתפים. כאשר נרצה לחשב את ערך הפולינום בנקודה מסוימת, כל משתתף יחשב את הערך כפי שידוע לו והסכום של כל החישובים יהיה הערך של הפולינום בנקודה. אחד היתרונות של האלגוריתם שהוא יעיל לשימוש מכיוון שנזדקק רק ל-2 סבבים של תקשורת בין המשתתפים, אחד לחלוקת הסוד והשני לחישוב של ערך הפולינום בנקודה, כל הפעולות הן חיבור וכפל. יתרון נוסף של האלגוריתם שהוא לא מוגבל לכמות משתתפים מסוימת ומתאים לכל מספר משתתפים שנרצה, ככל שיהיו יותר משתתפים ככה הבטיחות של האלגוריתם רק תגדל.
החיסרון שלו הוא שבכל חישוב של ערך הפולינום בנקודה, הערך שלו עלול להיות חשוף למשתתפים וככל שיהיו יותר נקודות כאלה יוכלו המשתתפים לשחזר את הפולינום.

Algorithm 2 Add.msplit procedure: given an element $s \in \mathbb{F}_p$, where $\mathbb{F}_p$ is a finite field containing p-primer elements, the procedure returns N additive secret shares whose sum is s.

Require: p - prime number, $s \in \mathbb{F}_p$ - a secret to share and $N \in \mathbb{N}$.

$\frac{\gamma_1}{\beta_1}, \ldots, \frac{\gamma_{N-1}}{\beta_{N-1}} \leftarrow$ - where γ and β are randomly chosen from $\mathbb{F}_P$

$\gamma_N \leftarrow s - \sum_{i=1}^{N-1} \gamma_i$

β_N - randomly chosen from $\mathbb{F}_P$

return $(\frac{\gamma_1}{\beta_1}, \ldots, \frac{\gamma_N}{\beta_N})$ - a sequence of secret shares of s.

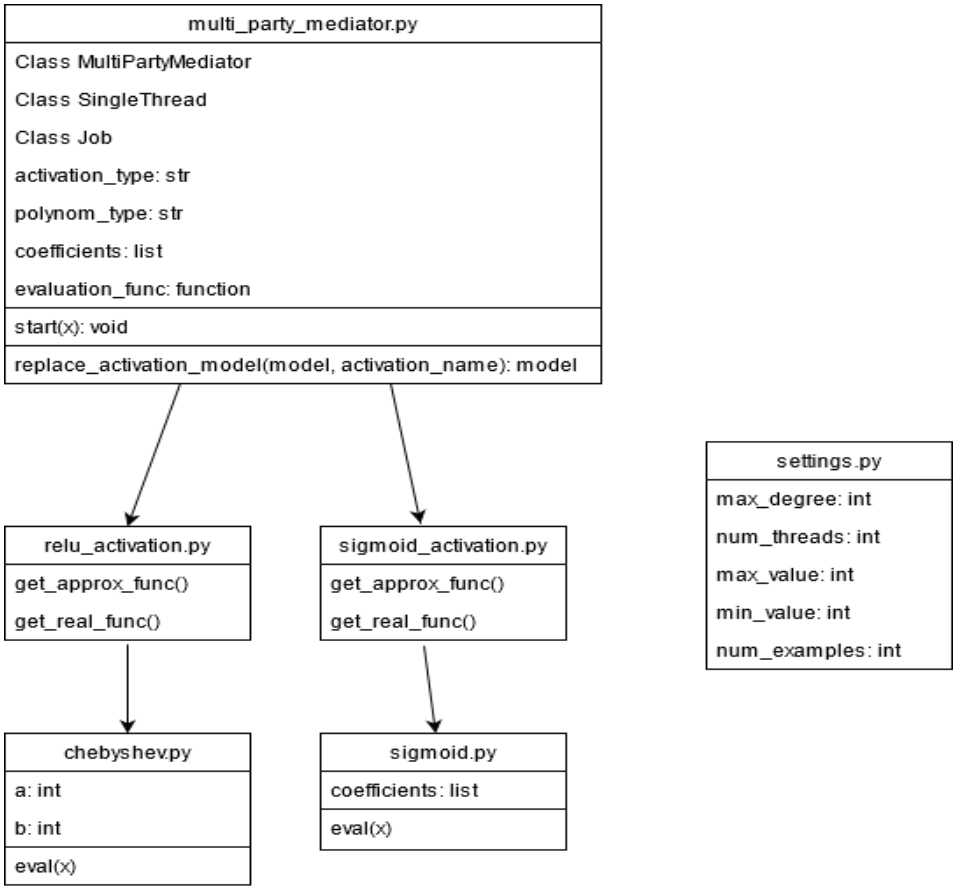
איור 9: אלגוריתם לחלוקת סוד בין N משתתפים שממסך את הקלט

האלגוריתם שמוצג באיור 9 פותר את החיסרון של האלגוריתם הקודם על ידי מיסוך הקלט, עבור פולינום מסדר d במקום להעביר לכל משתתף את הנקודה x בה רוצים להעריך את הפולינום, נעביר למשתתף i את הוקטור $(\beta_1^i x, \beta_2^i x^2, \ldots, \beta_d^i x^d)$ והוא יחשב את הערך של הפולינום $p_i(x) = a_0^i + \left(\frac{a_1^i}{\beta_1^i}\right)(\beta_1^i x) + \cdots + (\frac{a_d^i}{\beta_d^i})(\beta_d^i x^d)$.
בשיטה זאת האלגוריתם יעבוד בדיוק כמו הקודם רק שהמשתתפים לא יהיו חשופים לקלט x ולכן לא יוכלו לאסוף מידע על הפולינום.

נממש את האלגוריתם על ידי שימוש בספריה $threading.Thread$, אם נרצה n משתתפים אז ניצור n תהליכים(threads) שידמו משתתפים. ניצור מתווך שיהיה אחראי להעביר לכל משתתף את המקדמים שלו ובנוסף להעביר להם נקודות בהן רוצים להעריך את הפולינום. כמו כן המתווך יהיה אחראי לחכות שכל המשתתפים יסיימו ולסכום את התוצאות שלהם על מנת לקבל את הערך בנקודה. כל משתתף יחזיק את מקדמי הפולינום שהוא קיבל בהתחלה ובנוסף יהיה לו תור ($queue.Queue$) שיקבל נקודות בהן רוצים להעריך את הפולינום. בסיום החישוב כל תהליך יכתוב את התוצאה לתוך תור של משימות שהסתיימו.

2.5. ארכיטקטורה ונתוני מבחן

נסתכל על המחלקות החשובות בפרויקט בעזרת דיאגרמה:



איור 10: דיאגרמת המחלקות העיקריות

המחלקה העיקרית היא $MultiPartyMediator$ שחושפת פונקציה ($replace\_activation\_model$) על מנת להחליף פונקציות אקטיבציה במודל נתון בקירוב פולינומי של אותה הפונקציה ומחליפה אותה להיות הפונקציה start.

כאשר תקרא הפונקציה start(x) על מנת לקבל קירוב עבור הערך x, הפעולה תתבצע לפי מספר המשתתפים שהוגדר בקובץ settings.py ($num\_threads$). הפרמטרים של קירוב הפולינום נקבעים עוד בשלב ההחלפה, טווח הערכים הוא $[min\_value, max\_value]$, סדר הפולינום המקרב הוא max_degree, מספר הנקודות שלפיהן

חושב הפולינום הוא $num\_examples$.
הפונקציה שאותה מקרבים יכולה להיות או $Relu$ או $Sigmoid$ ואז משתמשים ב $relu\_actuvation$ או $sigmoid\_activation$ בהתאמה.

בשביל לדמות מספר משתתפים גבוה מאחת המחלקה $MultiPartyMediator$ משתמשת בשתי מחלקות פנימיות:

- Job: מייצגת חישוב יחיד של משתתף לחישוב הערך של הפולינום שלו
- $SingleThread$: מייצגת חוט (Thread) שמריץ Jobs.

בנוסף המחלקה אחראית לחלק את מקדמי הפולינום בין המשתתפים כפי שמופיע בהמשך.

נתוני האימון והמבחן לקוחים מהמאגר של $Mnist$ שכולל 60 אלף תמונות בגודל 28×28 בגווני אפור של ספרות בכתב יד. השתמשתי במסווג CNN שנוצר מראש ללא החלפה של פונקציות אקטיבציה בתור הבסיס לבדיקות.

3. תיאור הפרויקט

הפרויקט מורכב מחמישה חלקים עיקריים:

1. נבחן מספר שיטות לקירוב פולינומיאלי על מנת לבחור את השיטה המתאימה לנו ביותר, נממש את הקירובים בשפת פייתון ונשווה את השיטות מבחינת מהירות ביצוע ודיוק.
נעשה זאת בעזרת הרצת כמה אלפי ערכים מיוצרים רנדומלית בטווח $[-n, n]$ על הפולינום מול הפונקציה המקורית.

2. נממש את החלפת פונקציות ההפעלה במודל בקירוב הפולינומיאלי שנבחר בשלב הראשון.

3. מימוש חישוב מאובטח מרובה משתתפים, כך שעבור n-משתתפים נצטרך שיתוף פעולה של כולם על מנת לקבל את ערך הפולינום בנקודה, נבחן את השפעת זמן הריצה מול מספר המשתתפים. החישוב ימומש על ידי מתווך שייצר מקדמים רנדומליים לכל אחד מהמשתתפים למעט המשתמש האחרון שמקדמיו יהיו המקדמים המקוריים פחות כל המקדמים האחרים, על מנת להקשות על המשתתפים לשחזר את הפולינום המקורי ניתן לחלק את המקדמים של משתתף i בערך רנדומלי B_i ואז במקום להעביר לו לחישוב הפולינום את הערך x נעביר לו את $b_i x$.

4. נבחן את ההשפעה של השלבים הקודמים על ביצועי המודל מול ביצועים של מודל שלא עבר החלפה של פונקציות ההחלפה, נרוץ על כמה אלפי דוגמאות ממאגר הנתונים של MNIST המכילים תמונות של ספרות שנכתבו בכתב יד ונבחן את ביצועי הרשת, נסתכל על הדיוק וזמני הריצה כפונקציה של מספר המשתתפים ודרגת הפולינום.

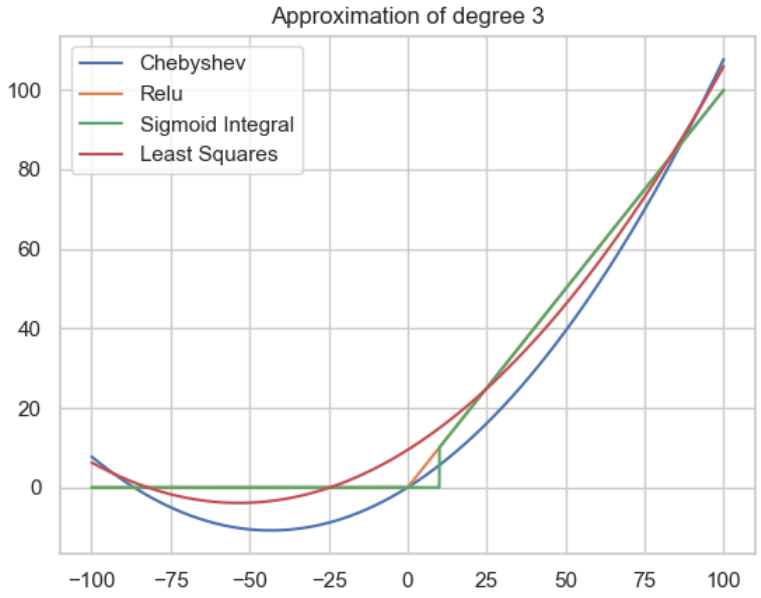
5. לבסוף נראה איך אפשר לנסות לייצג שכבות של רשת נוירונים כפולינומים, בשביל להשוות את התוצאות נראה איך מחשבים ידנית את השכבות ברשת (נספח).

4. תוצאות

בפרק זה נראה את התוצאות של הקירובים לפונקציות האקטיבציה בנפרד וכחלק מרשת נוירונים, כמו כן נראה את ההשפעה של חישוב מרובה משתתפים.

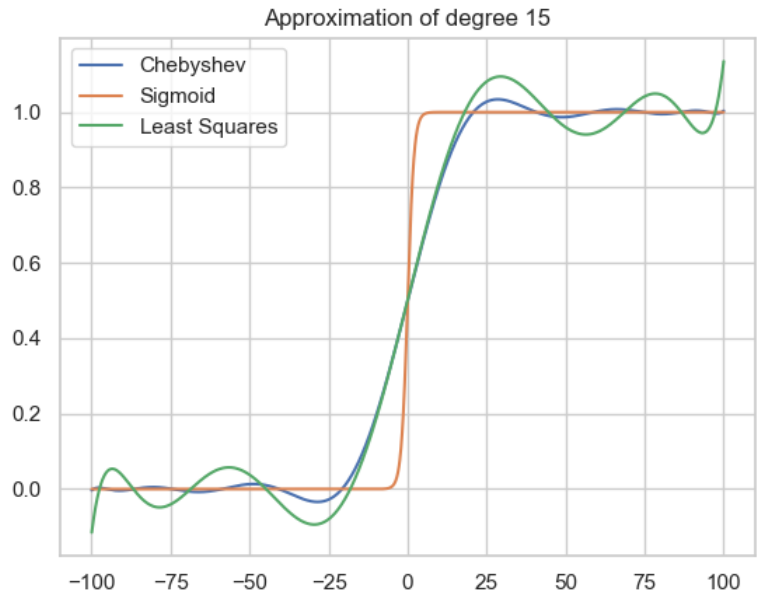
4.1. קירוב פולינומיאלי

את סט הנקודות של הפונקציות יצרתי בצורה אקראית עם 10000 נקודות בטווח x של $[-100,100]$ בעזרת הפונקציה $random.uniform(a,b)$ שמחזירה מספר ממשי בטווח $[a,b]$.



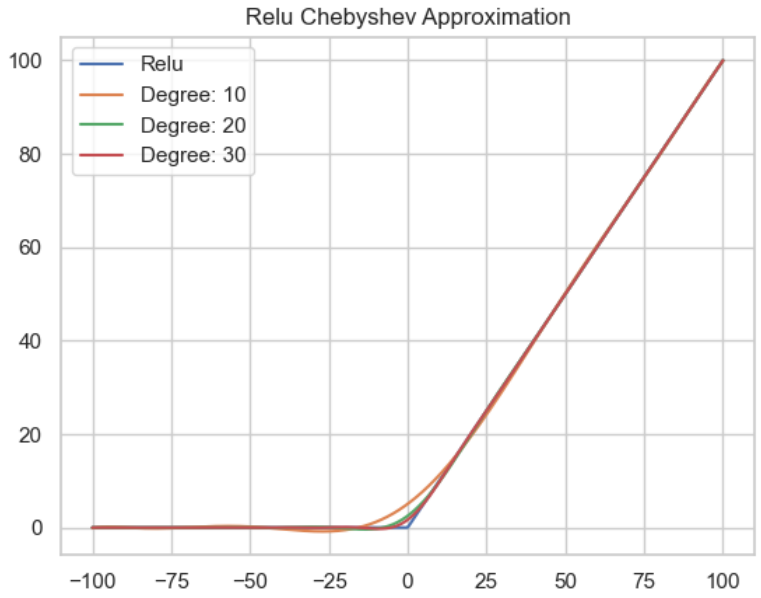
איור 11: קירוב פולינומיאלי מסדר 3 בשלוש שיטות שונות של Relu

באיור 11 ניתן לראות את התוצאות של קירוב פולינומיאלי של הפונקציה $Relu$ מסדר 3 בשיטות שהוזכרו קודם לכן בסעיף זה – שיטת הקירובים הפשוטים, צ'בישב וקירוב על ידי נגזרת. לפי איכות התוצאות הקירוב שהכי הזכיר את את הפוקציה המקורית $Relu$ היה קירוב לפי נגזרת (בירוק), הדבר החריג בקירוב זה הוא שגיאה סביב החלק החיובי מימין לאפס כנראה בגלל שכפי שהזכרתי הפונקציה אינה גזירה בנקודה אפס.



איור 12: קירוב פולינומיאלי מסדר 15 בשתי שיטות שונות של Sigmoid

באיור 12 ניתן לראות את התוצאות של קירוב פולינומיאלי של הפונקציה Sigmoid מסדר 15 לשיטות שהוזכרו קודם לכן בסעיף זה – שיטת הקירובים הפשוטים וצ'בישב.
ניתן לראות שהקירוב לפי צ'בישב נותן תוצאות עקביות בשני המקרים, בגלל שהוא נותן תוצאות טובות וקל מאוד לשימוש על כל סוג של קלט כולל tensors כפי שנראה בהמשך, בחרתי להתמקד בשיטה זאת.

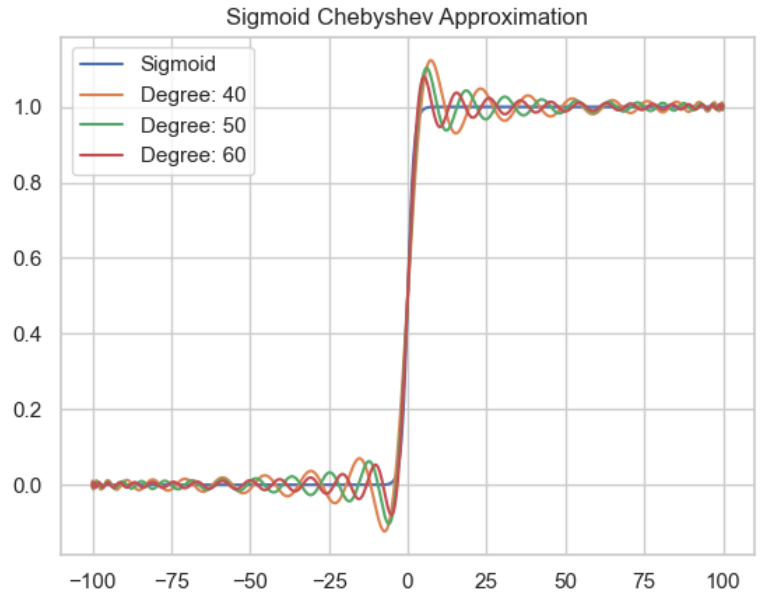


איור 13: קירוב פולינומיאלי של Relu לפי סדר הפולינום (צ'בישב)

באיור 13 ניתן לראות תוצאות של קירוב פולינומיאלי של הפונקציה $Relu$ בקירוב על ידי פולינום צ'בישב מדרגות שונות. ככל שהדרגה יותר גבוהה ככה גם הקירוב יותר קרוב לפונקציה המקורית.
על מנת לקבל מושג לגבי סדר הפולינום שנרצה להשתמש בו בהמשך, נסתכל בטבלה שתציג את ריבוע גודל השגיאה הממוצע לפי סדר הפולינום:

סדר הפולינום	ריבוע השגיאה
10	1.1070
20	0.1381
30	0.0416
40	0.0179
50	0.0093
60	0.0055
70	0.0033

מתוצאות הטבלה רואים שריבוע השגיאה הממוצע יורד בחדות עד סדר 40 ולאחר מכן הירידה של השגיאה מתונה יותר, מכיוון שדרגה גבוהה של פולינום מוסיפה לזמן הריצה נרצה להשתמש בדרגה כמה שיותר נמוכה אבל עדיין לקבל דיוק כמה שיותר טוב.



איור 14: קירוב פולינומיאלי של Sigmoid לפי סדר הפולינום (צ'בישב)

באיור 14 ניתן לראות תוצאות של קירוב פולינומיאלי של הפונקציה Sigmoid בקירוב על ידי פולינום צ'בישב מדרגות שונות. ככל שהדרגה יותר גבוהה ככה גם הקירוב יותר קרוב לפונקציה המקורית.
על מנת לקבל מושג לגבי סדר הפולינום שנרצה להשתמש בו בהמשך, נסתכל בטבלה שתציג את ריבוע גודל השגיאה הממוצע לפי סדר הפולינום:

סדר הפולינום	ריבוע השגיאה
10	0.0120
20	0.0047
30	0.0023

40	0.0012
50	0.0007
60	0.0004
70	0.0002
80	0.0001
90	0.0001

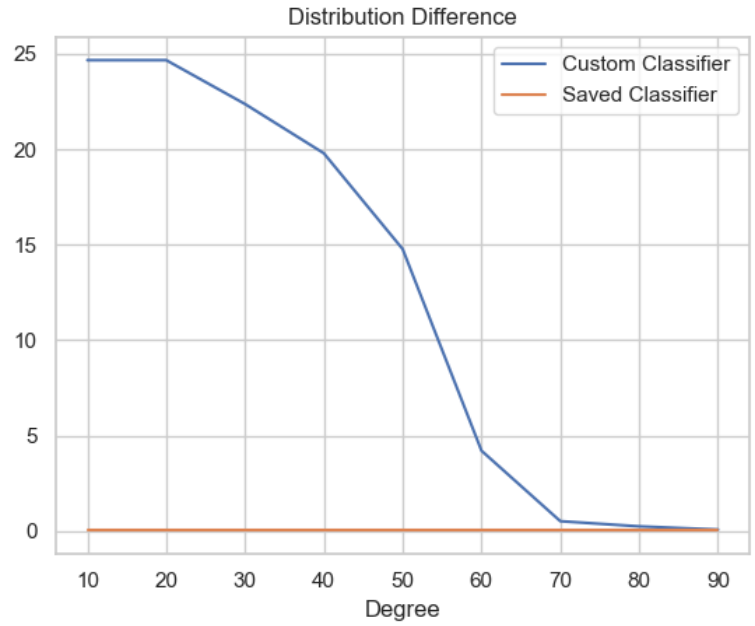
הפונקציה Sigmoid הינה חסומה בטווח $(0,1)$ ולכן בגלל זה ריבוע השגיאה שרואים הוא מסדר הרבה יותר קטן מאשר קודם. השגיאה הופכת להיות מאוד נמוכה סביב דרגה 50 והופכת לאפסית מדרגה 80.

4.2. קירוב פולינומיאלי ברשת נוירונים

על מנת לבצע החלפה של פונקציות אקטיבציה בצורה גנרית לכל רשת נוירונים נתונה בניתי פונקציה שנקראת $replace\_activation\_model(classifier, activation\_name)$, היא מקבלת את המסווג ואת השם של פונקציית האקטיבציה שרוצים להחליף ($Relu$ או $Sigmoid$) ומבצעת את ההחלפה בכל השכבות של המסווג בהן מופיעה פונקציית אקטיבציה מהסוג הנתון, הפונקציה מחזירה מסווג חדש שהוחלפו בו כל פונקציות האקטיבציה לקירוב פולינומיאלי עם פולינום צ'בישב.

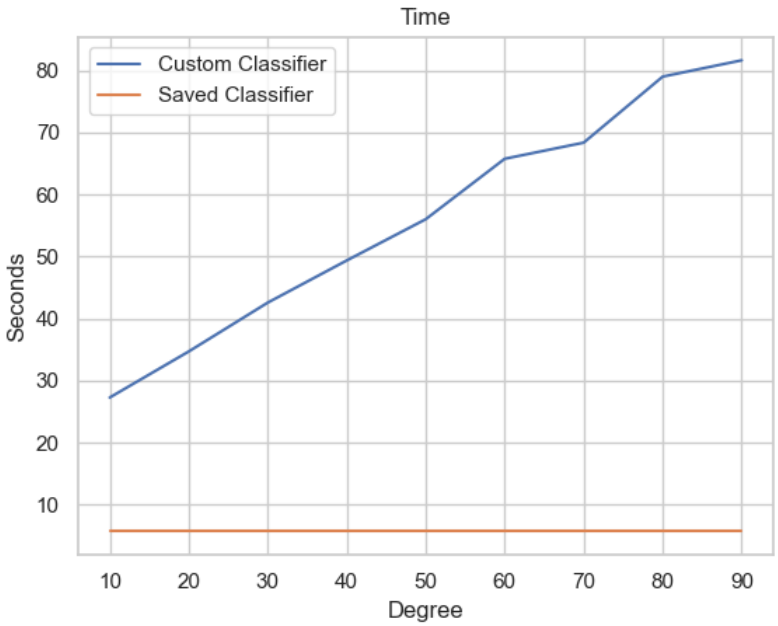
הדרגה של הפולינום נקבעת בקובץ $settings.py$ לפי המשתנה max_degree.
על מנת לבדוק את ההשפעה של החלפת פונקציות האקטיבציה בפולינום, נבצע את הפעולות הבאות:

1. נטען מסווג שאומן מראש ונשמר בקובץ, המסווג מורכב מכ-20 שכבות שונות בהן 5 פונקציות אקטיבציה מסוג $Relu$.
2. נשלח את המסווג לפונקציה $replace\_activation\_model$ על מנת להחליף את פונקציות ה Relu בקירוב פולינומיאלי.
3. בעזרת data$tf.keras.datasets.mnist.load_$ נטען 60 אלף דוגמאות אימון ו-10 אלף דוגמאות בדיקה של תמונות בגודל $28x28$ המייצגות ספרות.
4. נחשב את האנטרופיה היחסית$(Kullback - Leibler\ divergence)$ של דוגמאות האימון והבדיקה, האנטרופיה היחסית היא מדד לאופן שבו התפלגות הסתברות אחת שונה מהשנייה, ככל שהערך קרוב יותר לאפס ככה ההתפלגויות יותר דומות אחת לשניה.
5. מריצים את המסווג שהוחלפו בו פונקציות האקטיבציה על כמות גדולה של דוגמאות אימון(1000) ומשווים את האנטרופיה היחסית של דוגמאות האימון והתוצאה שלו.
6. מבצעים את אותו הדבר כמו 5 רק עם המסווג המקורי.
7. משווים את התוצאות של סעיפים $5,6$ ביחס לסדר הפולינום.



איור 15 אנטרופיה יחסית לפי דרגת הפולינום

באיור 15 רואים את האנטרופיה היחסית לפי דרגת הפולינום עבור המסווג שעבר החלפה לפונקציות האקטיבציה מול אחד שלא. ככל שדרגת הפולינום עולה ככה ביצועי המסווג שלנו דומים יותר למסווג המקורי.

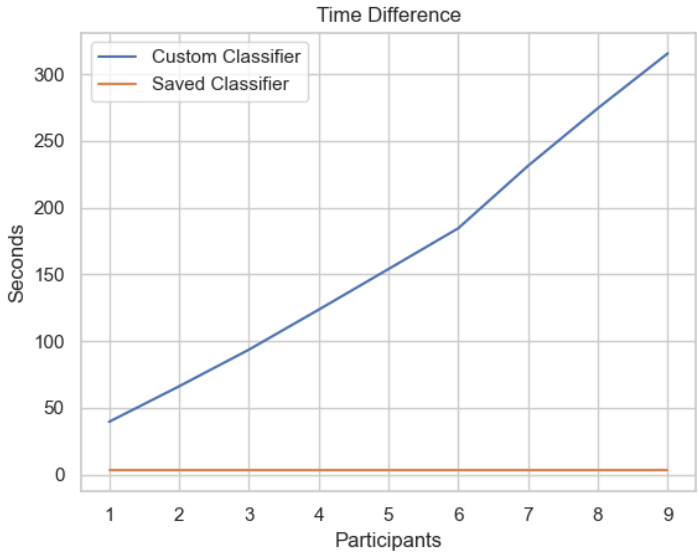


איור 16: זמן ריצה בשניות לפי דרגת הפולינום

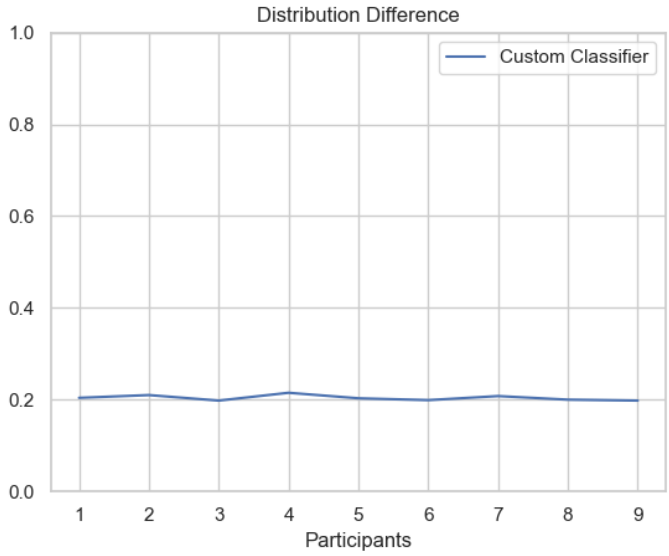
באיור 16 רואים את זמן הריצה של המסווג לפי דרגת הפולינום, ככל שדרגת הפולינום עולה גן זמן הריצה עולה לינארית. לכן נרצה להשתמש בדרגת פולינום כמה שיותר נמוכה תוך כדי שמירה על תוצאות כמה שיותר קרובות למסווג המקורי. לפי הגרפים דרגה של $70 - 80$ צפויה להביא לנו את הדרוש.

4.3. חישוב מאובטח מרובה משתתפים-תוצאות

הסדר של הפולינום יהיה קבוע ובהסתמך על התוצאות הקודמות בחרתי אותו להיות 80.



איור 17: זמן הריצה לפי כמות המשתתפים



איור 18: הבדל ההתפלגות לפי כמות המשתתפים

באיור 17 אנו רואים את ההשפעה של כמות המשתתפים על זמן הריצה הכולל של סיווג תמונות באמצעות המודל לאחר שעבר החלפת פונקציות אקטיבציה בפולינום, מקדמי הפולינום חולקו בין המשתתפים לפי האלגוריתם שהוצג מקודם. באופן לא מתפיע קיבלנו פונקציה כמעט לינארית במספר המשתתפים, ככל שיש יותר משתתפים אז לוקח יותר זמן להעריך את הפולינום בנקודה מכיוון שצריך לבצע את הפעולה עבור כל משתתף ואחר כך

לסכום את התוצאות. מכיוון שהאלגוריתם נבדק בריצה מקבילית על מחשב יחיד אז העומס עליו גדל. בחיים האמיתיים אפשר להריץ את האלגוריתם על מכונות נפרדות, לדוגמא מכונה נפרדת לכל משתתף ובמקרה כזה נצפה לראות תוצאות מאוד דומות לריצה עם משתתף יחיד.

באיור 18 אנו רואים את ההשפעה של כמות המשתתפים על הבדל ההתפלגות, כמובן אנו רוצים שההתפלגות של התוצאות שלנו תהיה כמה שיותר דומה לנתוני האימון. התוצאות דומות מבחינת סדר גודל לכל מספר של כמות משתתפים, ההבדלים הקטנים יכולים להיות מוסברים בגלל שנתוני המבחן נבחרים בצורה אקראית בכל בדיקה ולא כללו הרבה תמונות בכל פעם (כ-1000) בשביל לשמור על זמן ריצה כולל נמוך.

5. סיכום ומסקנות

במהלך הפרויקט סקרתי כמה שיטות שונות לקירוב פולינומיאל. ראינו בכל השיטות שככל שמאפשרים דרגת פולינום יותר גבוהה אז הדיוק מתקרב יותר לפונקציה המקורית עד שכמעט אין הבדל, החיסרון מגולם בזמן ריצה יותר גבוה.
בזכות רמת הדיוק של הקירוב הפולינומיאלי ניתן להשתמש בזה על רשתות נוירונים שכבר עברו אימון וחישבו להן את המשקלים על ידי החלפת פונקציות האקטיבציה בפולינומים.
היבט שלא נבדק במסגרת הפרויקט הוא החלפת פונקציות האקטיבציה עוד בשלב אימון הרשת, דבר זה יכול להקטין את השגיאה שתתקבל על נתוני המבחן. בכל מקרה כפי שראינו אם דרגת הפולינום מספיק גבוהה והדיוק שלו מספיק טוב אז כמעט ואין הבדל בביצועים על נתוני המבחן.

לאחר החלפת פונקציות האקטיבציה בפולינומים, ניתן להשתמש ביתרונות שמציעים הפולינומים על פני פונקציות רגילות, סכימות לשיתוף סוד. באמצעות שימוש בסכימה לשיתוף סוד ניתן לחלק את הבעלות על הפולינומים ובעצם בעלות על השימוש ברשת המאומנת בין כל מספר משתתפים. חלוקה כזאת תאפשר לבנות מודל עסקי לשימוש ברשתות נוירונים מאומנת. אומנם בניסויים שנערכו קיבלנו עלייה בזמני הריצה ככל שמספר המשתתפים גדל, בעולם האמיתי כל משתתף יכול לרוץ על מכונה נפרדת ואז רק יהיה צריך להוסיף את תוספת הזמן של תקורת התקשורת שיכולה להיות זניחה, אם לדוגמא המחשבים נמצאים באותה הרשת.

בשלב האחרון של הפרויקט (נספח) מוצגת התחלה של גישה שתאפשר ייצוג של כל הרשת באמצעות מספר סופי של פולינומים, אם אפשר לנסות לנחש את פונקציות האקטיבציה שחולקו בין המשתתפים, גישה של חלוקת כל הרשת בין המשתתפים כבר לא תאפשר שיחזור של הרשת בלי לאמן אותה מחדש.

את הקוד של הפרויקט ניתן למצוא בכתובת: https://github.com/kostia6/project

6. ביבליוגרפיה

[1] P. Derbeko, S. Dolev and E. Gudes *"Deep Neural Networks as Similitude Models for Sharing Big Data"* ,Proceedings of BigData 2019: 5728-5736

[2] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery Numerical *"Recipes in C The Art of Scientific Computing Second Edition"* , Cambridge University Press, 2002.

[3] E. Hesamifard, H. Takabi and M. Ghasemi "Cryptodl: Deep neural networks over encrypted data" , arXiv.org > cs > arXiv:1711.05189, 2017.

[4] D. Bitan and S. Dolev, "Optimal-round preprocessing-mpc via polynomial representation and distributed random matrix (extended abstract)," IACR Cryptology ePrint Archive, vol. 2019, p. 1024, 2019.

[Online]. Available: https://eprint.iacr.org/2019/1024

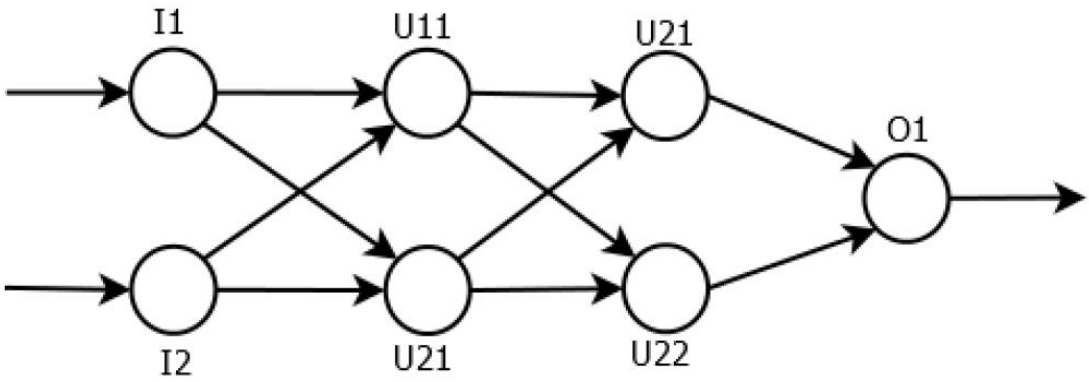
[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classication with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger,editors, Advances in Neural Information Processing Systems 25, pages 1097-1105 Curran Associates, Inc., 2012.

[6] Shalabh, IIT Kanpur "Regression Analysis-polynomial Regression Models", chapter 12

[7] P. Derbeko, S. Dolev and E. Gudes "Communication-less Secure Multi-Party Computation of Deep Neural Network Represented by Polynomial", Proceedings of BigData, pp. 8-9

נספח

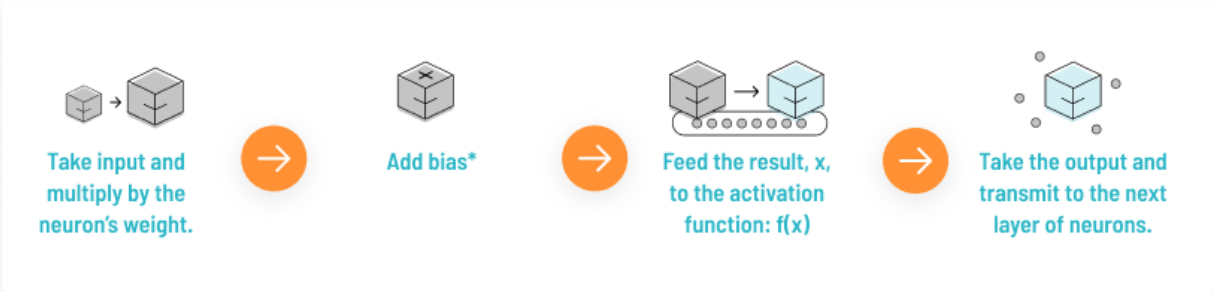
במאמר-[7] מציגים שיטה איך לייצג רשת נוירונים שלמה בעזרת פולינומים. כאשר ברשת נוירונים יש שכבה שמשתמשת בפונקציית אקטיבציה שניתן לקרב אותה בעזרת פולינום כפי שנעשה בסעיפים הקודמים, ניתן לקרב את כל השכבה בעזרת פולינומים. הרעיון הוא שבמקום לקרב בעזרת פולינום כל יחידה ברשת אפשר לקרב בעזרת פולינום שייצג את זרימת המידע ברשת.



איור 19: דוגמא לרשת נוירונים פשוטה

באיור 19 רואים דוגמא לרשת פשוטה המורכבת משכבת קלט, שתי שכבות נסתרות(dense layers) ושכבת פלט. עבור יחידה $U11$ הקלט שלה מגיע מ: $I1, I2$ ונניח שפונקציית האקטיבציה שלה היא $Relu$ שמקורבת בעזרת פולינום מדרגה d, אז הפולינום שלה יהיה $P_{11} = pol_{11}(\sum_i w_i I_i)$ כאשר w_i הם המשקולות של היחידה. באותה הצורה נחשב עבור $U21$ את $p_{21} = pol_{21}(\sum_i w_i p_{1i})$ או במקרה הכללי $p_{lj} = pol_{lj}(\sum_i w_i p_{(l-1)i})$.

בדוגמא הזאת קיבלנו שאפשר לייצג את הרשת בעזרת שני פולינומים P_{21}, P_{22} שיהיו מדרגה d^2 וישאר רק לחשב את שכבת הפלט שבה יש פונקציית אקטיבציה כמו $Softmax$.



איור 20: מבנה טיפוסי של שכבה נסתרת (Dense)

באיור 20 רואים מבנה של שכבה נסתרת, השכבה מורכבת מ-3 חלקים עיקריים:

1. משקל – מכפילים את המשקלים בערכי השכבה הקודמת
2. ביאס – שכבה חיבורית, דומה לקבוע בפונקציה לינארית
3. פונקציית אקטיבציה – במקרה שלנו היא תהיה $Relu$

בשביל לקבל פולינום מייצג לכל שיכבה ביצעתי ניסוי פשוט, לקחתי מסווג מאוד בסיסי של מודל המורכב מ:

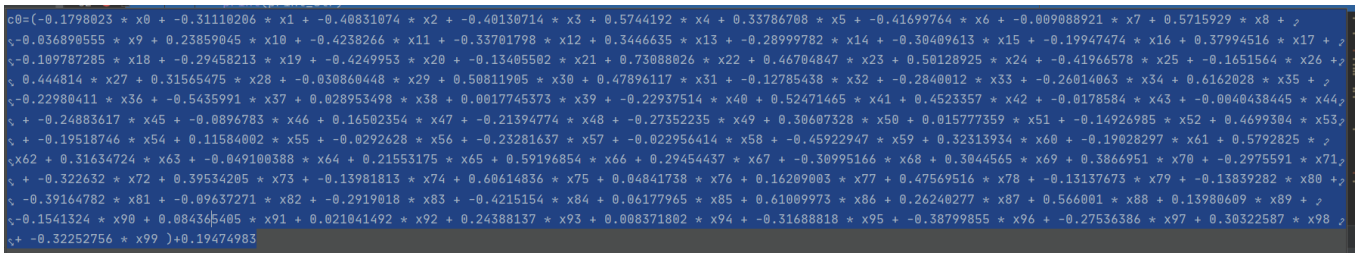
- שכבת קלט בגודל 28×28 ופלט בגודל 784
- שכבת נורמליזציה (Batch Normalization), קלט ופלט בגודל 784
- שכבה נסתרת (Dense), קלט בגודל 784 ופלט בגודל 300
- שכבה נסתרת (Dense), קלט בגודל 300 ופלט בגודל 100
- שכבה נסתרת (Dense), קלט בגודל 100 ופלט בגודל 10

בכל השכבות הנסתרות למעט האחרונה פונקציית האקטיבציה היא $Relu$ כאשר באחרונה היא $Softmax$.

נחשב את ייצוג הפולינום בכל שכבה נסתרת ושכבת נורמליזציה כפולינום, בשלב הראשון הפולינום מחושב לכל קודקוד בכל שכבה. נעבוד בכמה שלבים:

1. נעבור על שכבות המודל, עבור כל שכבה נשמור את השם לה, המשקלים ואפסילון (בשביל שכבת נורמליזציה).
2. נדפיס פעם אחת את הפולינום שמייצג את פונקציית האקטיבציה $Relu$ (חושב בעזרת קירוב בדומה לקודם), פה השתמשתי בספריה $sympy$ שעוזרת לפשט ביטויים אלגבריים על מנת להדפיס את הפולינום בצורה המינימליסטית שלו (ללא כפילויות של מקדמים לאותה הדרגה)
3. עבור כל שכבה שקיבלנו בשלב הראשון נבצע את החישובים המתאימים לשכבה, לדוגמא עבור שכבה נסתרת בהתאם לאיור 20 נבצע:
   a. נכפיל את השכבה הקודמת במשקולות של השכבה הנוכחית, כאשר הערכים של השכבה הקודמת מיוצגים כ-$x_1, \dots x_n$ אם נניח שהפלט של השכבה הקודמת הוא בגודל n.
   b. לערכים שקיבלנו בשלב הקודם נוסיף את הביאס (פעולה חיבורית)
   c. נדפיס את התוצאה

הערה: לא הפעלתי פונקציית אקטיבציה כי היא כבר הודפסה לפני ההתחלה והקלט שלה הוא הפלט של התהליך הזה.



איור 21: דוגמא להדפסת הניורון הראשון בשכבה האחרונה

באיור 21 רואים איך נראת הדפסה כזאת, בשכבה האחרונה הקלט הוא בגודל 100 ולכן רואים משתנים $x_0, \dots, x_{99}$ שמייצגים את ערכי השכבה הקודמת. ההדפסה היא עבור c_0 שמייצג את הניורון הראשון מתוך 10 בשכבת הפלט.

חלק זה של העבודה אינו שלם כי צריך להזין ערכים כמו שקיבלנו באיור 20 לתוך פונקציית האקטיבציה ובנוסף לאחד את הפולינומים על מנת לקבל ייצוג כמו שראינו באיור 19. חלק ההמשך נעשה במסגרת אחרת על ידי פיליפ.