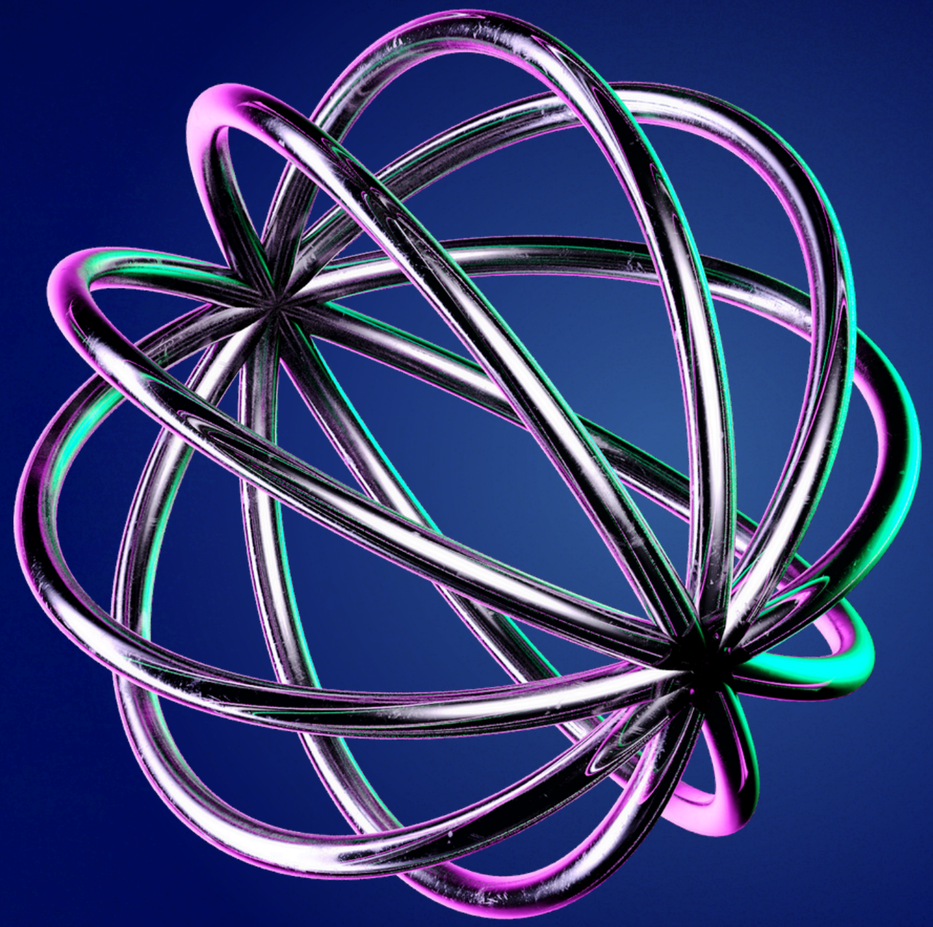
Hebb net
רשתות סיווג

מגישים:
דביר אופטובסקי
שירה דואק
איתיאל טרבלסי



רקע:

- **ממציא המודל:**

  דונלד הב (Donald Hebb), נוירופסיכולוג קנדי.

  פרסם את המודל בשנת 1949 בספרו "The Organization of Behavior".

- **ביולוגיה:**

  איך קשרים סינפטיים מתחזקים כששני נוירונים פועלים ביחד

- **המטרה של המודל:**

  להסביר כיצד המוח לומד ומשמר מידע באמצעות קשרים סינפטיים בין נוירונים.

- **חוק הב (Hebb's Rule):**
  - **"Neurons that fire together, wire together"**

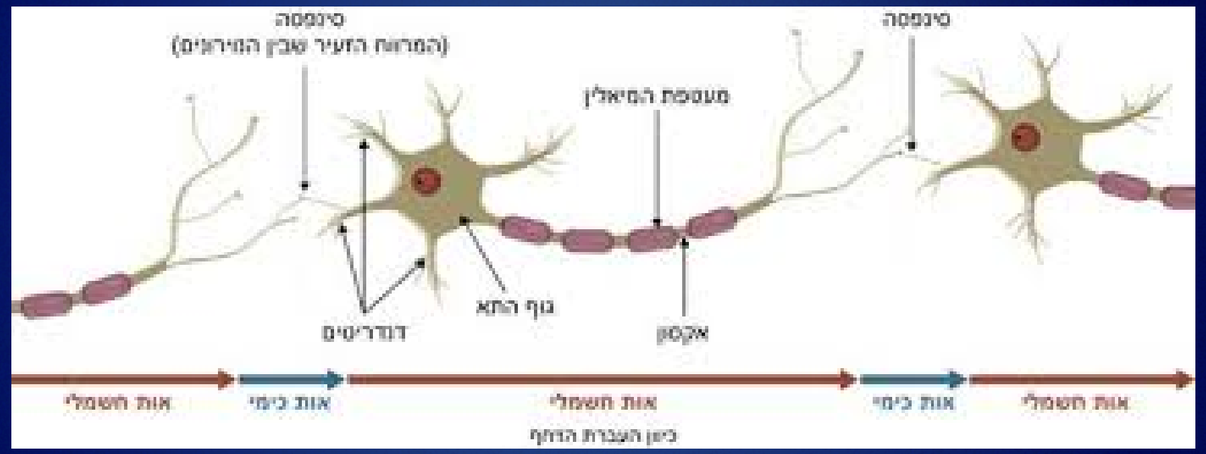
    "נוירונים שפועלים יחד, מתחזקים יחד".

מודל Hebb Net מנסה להוות ייצוג מתמטי של הלמידה שמתרחשת במוח באמצעות חיזוק קשרים בין נוירונים. הוא לוקח את ההשראה מהתהליכים הביולוגיים ומתרגם אותם למונחים מתמטיים פשוטים.

הביולוגיה מאחורי כלל Hebb

כלל HEBB נולד מתוך הבנה של המנגנונים הביולוגיים במוח, שמסבירים כיצד נוירונים (תאי עצב) יוצרים קשרים חזקים יותר עם הזמן בתגובה ללמידה או גירוי חוזר. הרעיון הביולוגי מאחורי כלל זה מתרכז ב:

נוירונים: תאי העצב במוח אחראים להעברת מידע. כל נוירון מורכב מ:
דנדריטים: קולטים מסרים מתאי עצב אחרים.
גוף התא: מעבד את המידע.
אקסון: משדר את המידע הלאה לנוירונים אחרים.
סינפסה: האזור שבו האקסון של נוירון אחד "פוגש" דנדריט של נוירון אחר ומעביר מסרים באמצעות אותות כימיים וחשמליים.



פלסטיות: היכולת של הסינפסה להתחזק או להיחלש כתוצאה מפעילות.
כלל HEBB: "נוירונים שיורים יחד, חוטים יחד."
כלומר:
כאשר נוירון פרה-סינפטי (השולח) ונוירון פוסט-סינפטי (המקבל) מופעלים בו-זמנית, הקשר ביניהם מתחזק.
מנגנון זה מאפשר למוח לשפר את הקשרים הרלוונטיים ללמידה וזיכרון.

HEBBIAN LEARNING

עקרונות המודל:

- חיזוק קשרים סינפטיים, למידה מבוססת פעילות
- כל דוגמת נתונים נראת פעם אחת פר נוירון - (נראה את זה)
- פשוט ולא מכיל מורכבות

- אלגוריתם:

$$w_i\,(new) = w_i\,(old) + x_i\,y$$

$$b\,(new) = b\,(old) + y$$

נראה דוגמא ונבין....

דוגמא פשוטה:

C

1	1	1
1	0	0
1	1	1

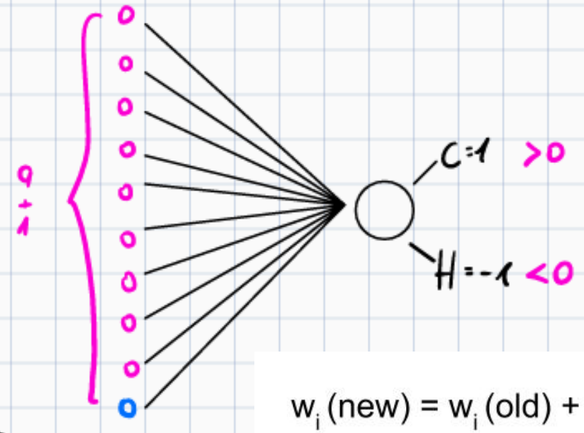
וקטור מוצג:

1 1 1 1 0 0 1 1 1 | b: 1

H

1	0	1
1	1	1
1	0	1

1 0 1 1 1 1 1 0 1 | b: 1



$$w_i\,(new) = w_i\,(old) + x_i\,y$$

$$b\,(new) = b\,(old) + y$$

$$W = \{\overbrace{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}^{\{0\}}\ 0\}$$

$$W = \{$$

$$x = W = \{0\} + \{1\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 1\} \cdot \overset{\text{העיפו}}{1}$$

$$W = \{\overbrace{1\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 1}^{x}\} + \{1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 1\} \cdot \overset{\text{העיפו}}{(-1)}$$

$$\boxed{W = \{0\ 1\ 0\ 0\ -1\ -1\ 0\ 1\ 0\ 0\}}$$

סמנו את האיזון

$$C \cdot W = 1\cdot 0 + 1\cdot 1 + 1\cdot 0 + 1\cdot 0 + 0\cdot(-1) + 0\cdot(-1) + 1\cdot 0 + 1\cdot 1 + 1\cdot 0 = 2 > 0 = C$$

$$H \cdot W = 1\cdot 0 + 0\cdot 1 + 1\cdot 0 + 1\cdot 0 + 1\cdot(-1) + 1\cdot(-1) + 1\cdot 0 + 0\cdot 1 + 1\cdot 0 = -2 < 0 = H$$

DATASET:

-1	-1	-1	1	1	-1	-1	-1
-1	-1	1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	1	-1
1	-1	-1	-1	-1	-1	-1	1
1	1	1	1	1	1	1	1
1	-1	-1	-1	-1	-1	-1	1
1	-1	-1	-1	-1	-1	-1	1
1	-1	-1	-1	-1	-1	-1	1

בהרצת הקוד נבדוק את
ההבדל כאשר יש ייצוג של 0
ו 1 וכאשר יש ייצוג של 1 ו -1

שהמחשבה היא שאנחנו
רוצים שמי שדלוק יתרום ומי
שלא דלוק יגרע, כשיש 0
הוא לא תורם ולא גורע

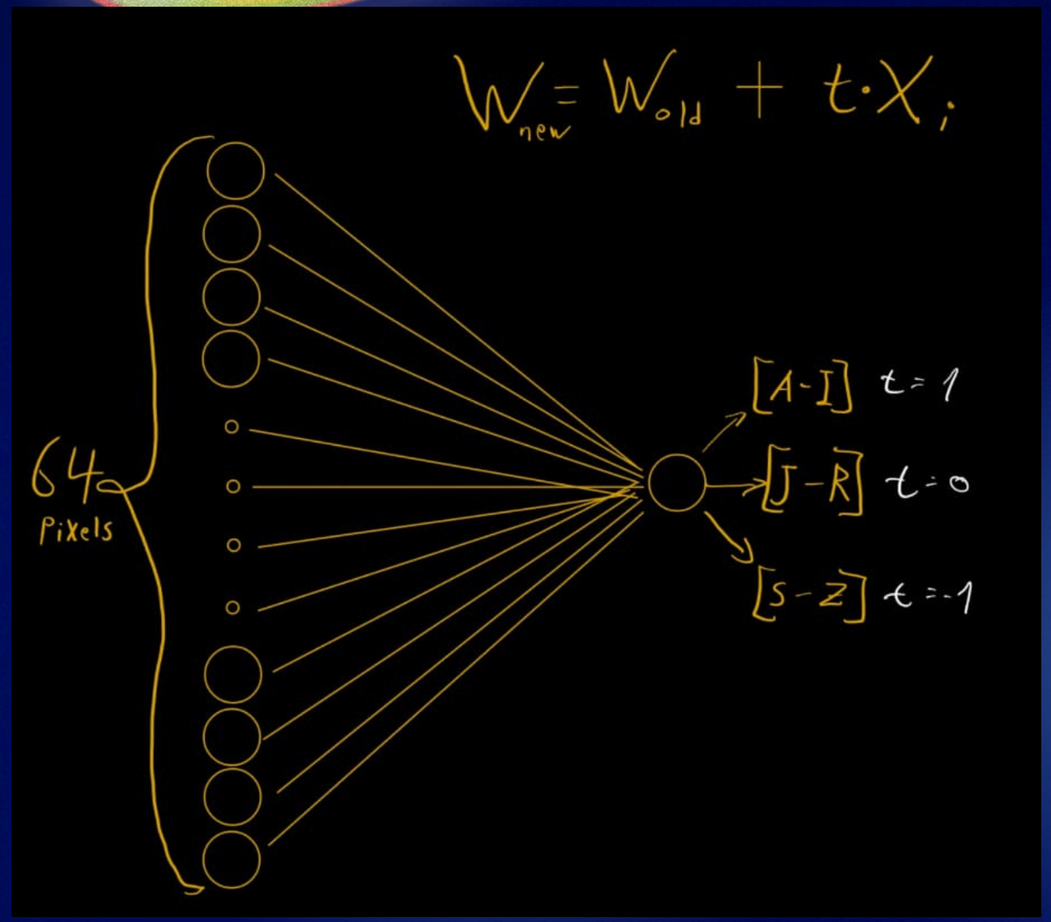
0	0	0	1	1	0	0	0
0	0	1	0	0	1	0	0
0	1	0	0	0	0	1	0
1	0	0	0	0	0	0	1
1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1

וקטור אות לדוג:

pixel64	pixel63	pixel62	pixel61	pixel60	pixel59	pixel58	pixel57	pixel56	---	pixel10	pixel9	pixel8	pixel7	pixel6	pixel5	pixel4	pixel3	pixel2	pixel1	letter
1	0	0	0	0	0	0	1			1	0	0	0	0	0	1	1	0	0	0	A

3 רשתות
של לימדת HEBB

1



קלט: וקטור באורך 64 (פיקסלים שכל פיקסל זה נוירון) שמיוצג בפורמט בינארי.
פלט: נוירון בודד שיורה:

1	הקבוצה הראשונה (A-I).
0	הקבוצה השנייה (J-R).
-1	הקבוצה השלישית (S-Z).

TRAIN:

1. מאפס את המשקלים גם של הBIAS
2. מכפילים את הוקטור של האות על פי הקבוצה המתאימה XI*T
3. מוסיפים לוקטור המשקלים הקודם את התוצאה וככה ממשיכים עד סיום הדאטה

TEST ON THE TRAIN DATA:
לוקחים כל וקטור שמייצג אות ומכפילים בוקטור המשקלים הסופי ולפי מה שיורה לשם הוא מסווג.
גדול מ 0 לקבוצה הראשונה, קטן מאפס לקבצוה השלישית ושווה לאפס לקבוצה השנייה

עמידות לרעש ברמות שונות:
לוקחים כל אות ומוסיפים לה בצורה הסתברותית רעש בצורה של הפיכת פיקסלים באקראיות לפי 5% 10% 20% 30% (לכל פיקסל)
ומריצים 100 פעם בכל רמת רעש עבור כל אות .

TRAIN:

```python
def in_target(letter:str):
    if ord('A') <= ord(letter) <= ord('I'):
        return 1
    elif ord('J') <= ord(letter) <= ord('R'):
        return 0
    elif ord('S') <= ord(letter) <= ord('Z'):
        return -1
```

---->

```python
#אימון הרשת, אין_טארגט נותן את הטווח הרצוי של האות, קרי המטרה
def train_Hebbian(train,alpha =1):
    W = np.zeros(train.shape[1])
    for letter in train.index:
        W = W + np.dot(train.loc[letter],in_target(letter))
    return W
```

```
the final Weights vector is:
 [-5.  5.  7.  9.  9.  5.  1. -9.  3. -3.  1. -1. -1.  1. -1. -5.  7. -1.
 -5. -1. -3. -5.  3. -3. 11.  9.  5. -1.  3.  5.  7. -1. 11. -1. -9. -3.
 -1. -7.  5. -3.  9. -7. -1. -5. -3.  1. -3.  1.  5.  5.  5. -1.  5.  3.
  1. -1. -1. -1. -1. -1. -1. -1. -1. -1.  1.]
```

TEST ON THE TRAIN DATA:

```python
def test_Hebbian(test,W):
    success = 0
    for letter in test.index:
        fire = np.dot(test.loc[letter],W)
        target = in_target(letter)

        if target == fire == 0: #both are zero - J-R class
            success += 1
        elif 0 < target <= fire: #both are positive - A-I class
            success += 1
        elif fire <= target < 0: #both are negative - S-Z class
            success += 1

    return success*100/26
```

```
the success rate of the network is : 46.15%
```

```python
mod_letters = letters.copy().replace(0, -1)
```

```
the success rate of the network is : 57.69%
```

עמידות לרעש ברמות שונות:

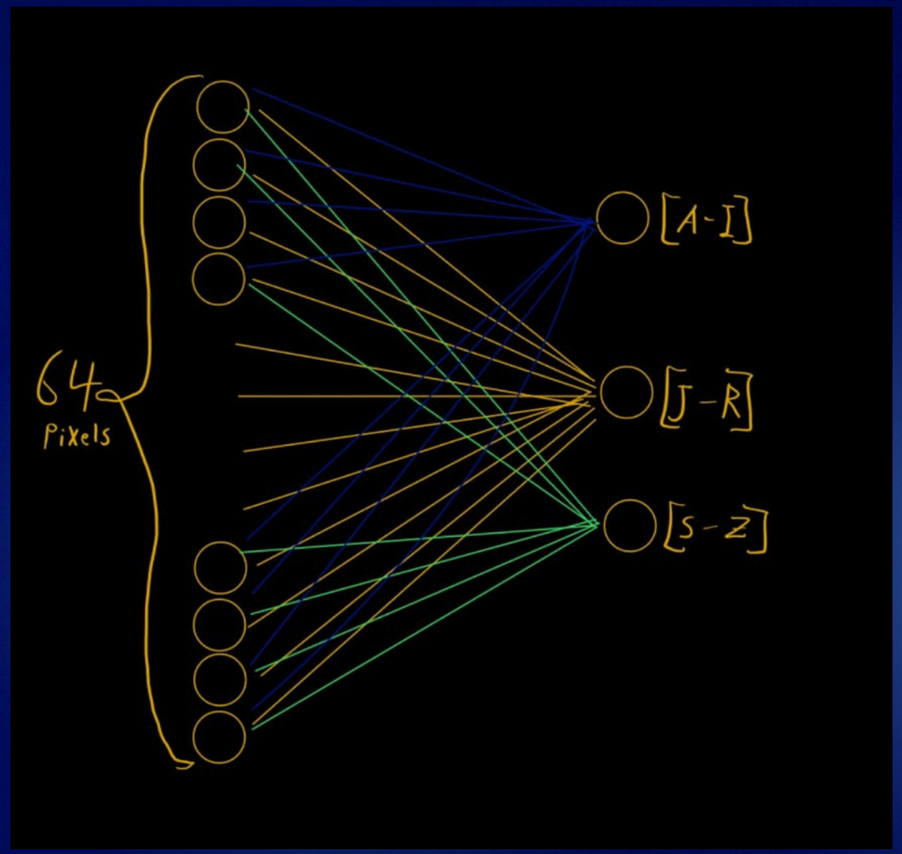
```
results of noisy data with dataset of 1's and 0's:

for noise5 the result is:   45.88461538461529
for noise10 the result is:   46.3846153846153
for noise20 the result is:   47.03846153846149
for noise30 the result is:   45.92307692307689
```

```
results of noisy data with dataset of 1's and -1's:

for noise5 the result is:   55.6923076923078
for noise10 the result is:   55.76923076923085
for noise20 the result is:   53.92307692307698
for noise30 the result is:   51.73076923076926
```

קלט: וקטור באורך 64 (פיקסלים שכל פיקסל זה נוירון) שמיוצג בפורמט בינארי.
פלט: 3 נוירונים שכל אחד יורה במידה ואת נמצאת בטווח שלו.(מצופה וקטור היחידה)



- [0, 0, 1] עבור הקבוצה הראשונה (A-I).
- [0, 1, 0] עבור הקבוצה השנייה (J-R).
- [1, 0, 0] עבור הקבוצה השלישית (S-Z).

TRAIN:

1. מאפס את המשקלים גם של הBIAS
2. עבור כל קבוצה בודקים אם האות הנבדקת אמורה להגיע אליה, מוסיפים למשקל המתאים במידה וכן, מחסירים אם לא.

TEST ON THE TRAIN DATA:

לוקחים כל וקטור שמייצג אות ומכפילים בוקטור המשקלים הסופי המתאים לקטגוריה ומסווגים לפי וקטור היחידה היוצא.

TRAIN:

```python
def train_3_Hebb(train,alpha=2):
    W = np.zeros((3,train.shape[1]))
    for weight in range(3):
        for letter in train.index:
            if weight-1 == in_target(letter):
                W[weight] = W[weight] + alpha * train.loc[letter]
            else:
                W[weight] = W[weight] - train.loc[letter]
    return W[::-1]
#W: [[S-Z],[J-R],[A-I]] ____  W[::-1] = [[A-I],[J-R],[S-Z]]
```

```
the weights Matrix of the network (65 is 64 pixels + bias)

(3, 65)
[ -7.  11.  13.  17.  17.   9.   3. -11.   1.  -7.   3.   1.  -1.   3.
  -7.  -7.   5.   1.  -9.   1.  -5.  -7.   3.  -9.  11.  15.  11.   1.
   9.  11.  13.  -5.  11.  -1.  -9.  -1.  -9.  -7.   9.  -7.   9.  -9.
  -1.  -3.  -3.  -5.  -3.   1.   5.  11.   9.   3.   9.   5.  -5.  -1.
  -1.  -1.  -1.  -1.  -1.  -1.  -1.  -1.   1.]
[-1. -7. -5. -7. -7. -3. -3. -5.  7.  5. -3. -5. -1. -3. 11. -1. 11. -5.
  3. -5.  1. -1.  3.  9. 11. -3. -7. -5. -9. -7. -5.  7. 11. -1. -9. -7.
 15. -7. -3.  5.  9. -3. -1. -9. -3. 13. -3.  1.  5. -7. -3. -9. -3. -1.
 13. -1. -1. -1. -1. -1. -1. -1. -1. -1.  1.]
[  8.  -4.  -8. -10. -10.  -6.   0.  16.  -8.   2.   0.   4.   2.   0.
  -4.   8. -16.   4.   6.   4.   4.   8.  -6.   0. -22. -12.  -4.   4.
   0.  -4.  -8.  -2. -22.   2.  18.   8.  -6.  14.  -6.   2. -18.  12.
   2.  12.   6.  -8.   6.  -2. -10.  -4.  -6.   6.  -6.  -4.  -8.   2.
   2.   2.   2.   2.   2.   2.   2.   2.  -2.]
```

TEST ON THE TRAIN DATA:

```python
def test_3_Hebb(test,W):
    results = [[] for _ in range(3)]  # Initialize list o
    for weight in range(3):
        for letter in test.index:
            ans = np.dot(W[weight],test.loc[letter])
            if ans > 1:
                results[weight].append(1)
            else:
                results[weight].append(0)
    return results
#results[[A,B,C,,,Z]--[A-I], [A,B,C,,,Z]--[J-R], [A,B,C,,
```

```
Accuracy of 1's and 0's: 38.46%

Accuracy of 1's and -1's: 38.46%
```

עמידות לרעש ברמות שונות:

```
Base Categories
results of noisy data with dataset of 1's and -1's:

for noise5 the result is:   34.76923076923077
for noise10 the result is:   34.615384615384606
for noise20 the result is:   36.038461538461526
for noise30 the result is:   34.53846153846155
```

```
Base Categories
results of noisy data with dataset of 1's and 0's:

for noise5 the result is:   41.96153846153843
for noise10 the result is:   39.3076923076923
for noise20 the result is:   35.57692307692306
for noise30 the result is:   32.46153846153847
```

שיפור האחוזים ע"י שינוי קטגוריות

```python
def in_target2(letter:str):
    first_group = ['I', 'L', 'T', 'F', 'E', 'H', 'N', 'M']
    second_group = ['O', 'Q', 'C', 'G', 'D', 'B', 'P', 'R', 'J']
    third_group = ['A', 'V', 'W', 'X', 'Y', 'Z', 'K', 'S', 'U']
```

```
Accuracy of 1's and 0's: 53.85%

Accuracy of 1's and -1's: 50.00%
```

עמידות לרעש ברמות שונות:

```
New Categories
results of noisy data with dataset of 1's and -1's:

for noise5 the result is:   44.499999999999936
for noise10 the result is:   46.38461538461535
for noise20 the result is:   46.038461538461526
for noise30 the result is:   41.15384615384613
```

```
New Categories
results of noisy data with dataset of 1's and 0's:

for noise5 the result is:   53.34615384615389
for noise10 the result is:   51.961538461538495
for noise20 the result is:   46.769230769230745
for noise30 the result is:   41.38461538461537
```

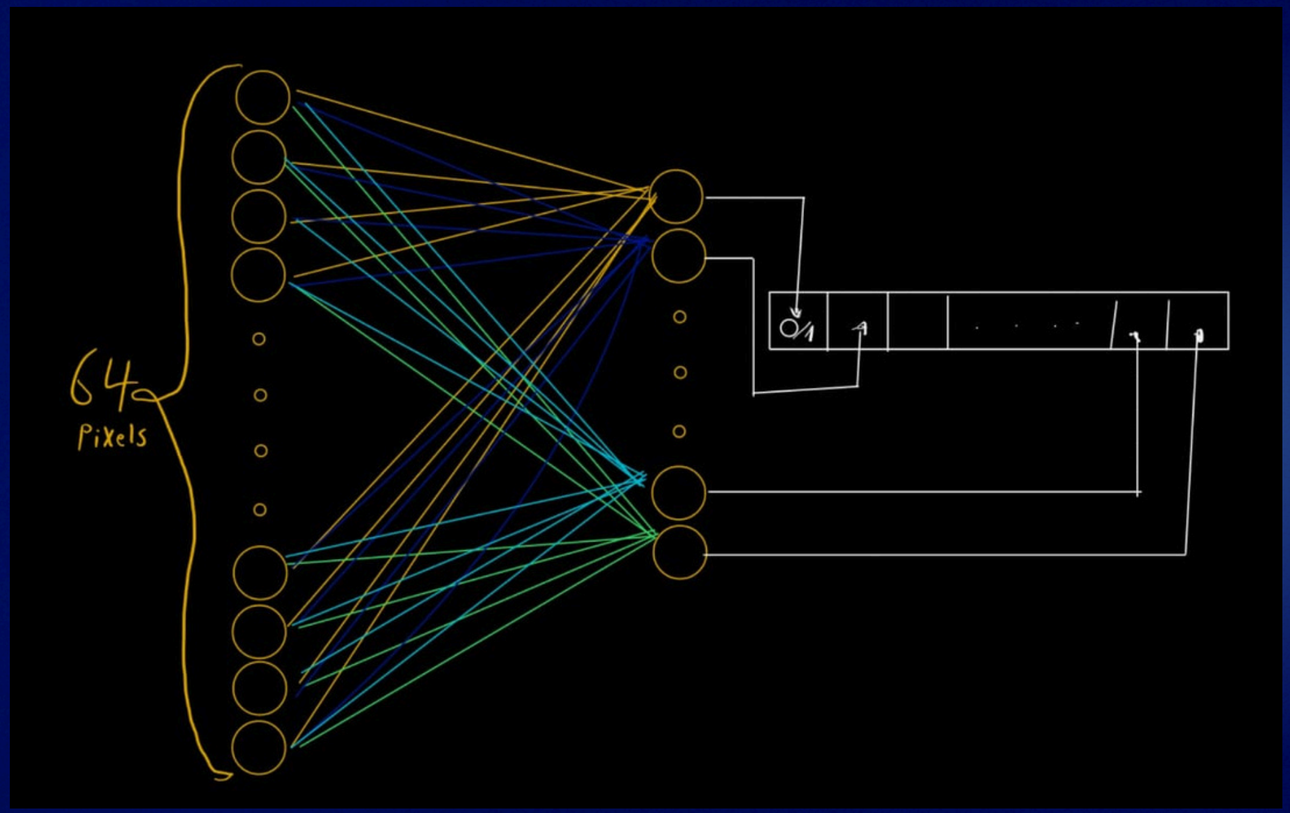
3

קלט: וקטור באורך 64 (פיקסלים שכל פיקסל זה נוירון) שמיוצג בפורמט בינארי.

פלט: וקטור היחידה באורך 26 כאשר כל מיקום מייצג אות לדוג

```
[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
```

המטרה זה שהאות שננבא שם יהיה ה 1 ובכל שאר הפיקסלים יהיה 0.



TRAIN:

1. עבור כל אות מייצרים וקטור משקלים ומאפסים אותם.
2. מכפילים את כל וקטורי האותיות בT המתאים כלומר עבור W1 נכפיל את ווקטור A ב13(הסבר יגיע) וכל השאר ב 1- וסוכמים אותם לW1 וכן גם לבאים במילים אחרות, עושים הביאן על כל נוירון בקצה \ על כל אות.
3. יווצרו לנו 26 וקטורי משקלים שעבור כל אות וקטור משקל.

TEST ON THE TRAIN DATA:

בודק עבור כל אות האם הצלחנו לנבא נכון(כלומר 1 אם מנסה לנבא את האות הנכונה או 0 אם לא. קרי- וקטור היחידה)

TRAIN:

```python
def train_26_Hebb(train,alpha =26):
    W = np.zeros((train.shape[0],train.shape[1]))
    for weight in range(26):
        for letter in train.index:
            if weight+1 == to_num(letter):
                W[weight] = W[weight] + np.dot(train.loc[letter],alpha)
            else: W[weight] = W[weight] - train.loc[letter]
    return W
```

```python
for i in range(11,16):
    W26 = train_26_Hebb(letters,alpha = i)
    TW26 = test_26_Hebb(letters,W26)
    print("i is:    ",i)
    print(test_accuracy_simple(TW26))
```

✓ 0.6s

```
i is:     11
97.92899408284023
i is:     12
98.37278106508876
i is:     13
98.66863905325444
i is:     14
95.11834319526628
i ís:     15
92.89940828402366
```

TEST ON THE TRAIN DATA:

```python
def test_26_Hebb(test,W):
    results = [[] for _ in range(26)]  # Initialize li
    for weight in range(26):
        for letter in test.index:
            ans = np.dot(W[weight],test.loc[letter])
            if ans > 1:
                results[weight].append(1)
            else:
                results[weight].append(0)
    return results
```

```
dataset of 1's and 0's with alpha = 13

Accuracy: 98.67%
Neuron C made mistakes on letters: C
Neuron F made mistakes on letters: F
Neuron G made mistakes on letters: J
Neuron I made mistakes on letters: T
Neuron L made mistakes on letters: L
Neuron M made mistakes on letters: N
Neuron N made mistakes on letters: M
Neuron P made mistakes on letters: P
Neuron Q made mistakes on letters: O
```

```
dataset of 1's and -1's with alpha = 11

Accuracy: 98.67%
Neuron C made mistakes on letters: C
Neuron F made mistakes on letters: F
Neuron I made mistakes on letters: T
Neuron L made mistakes on letters: L
Neuron M made mistakes on letters: N
Neuron N made mistakes on letters: M
Neuron O made mistakes on letters: Q
Neuron P made mistakes on letters: P
Neuron T made mistakes on letters: I
```

עמידות לרעש ברמות שונות:

```
results of noisy data with dataset of 1's and 0's;  alpha = 13:

for noise5 the result is:   97.96301775147928
for noise10 the result is:   97.33136094674559
for noise20 the result is:   94.0991124260355
for noise30 the result is:   88.353550295858
for noise50 the result is:   52.575443786982234
```

```
results of noisy data with dataset of 1's and -1's;  alpha = 11:

for noise5 the result is:   98.0118343195267
for noise10 the result is:   97.53402366863914
for noise20 the result is:   95.87130177514794
for noise30 the result is:   92.0695266272189
for noise50 the result is:   52.62869822485207
```

יתרונות של המודל:

- מודל פשוט ולא מורכב למימוש
- המודל פועל על פי כלל פשוט, מה שהופך אותו ליישומי ברמה מתמטית בסיסית.
- המודל מבוסס על עקרון פשוט ואינטואיטיבי: "נוירונים שפועלים יחד מתחזקים יחד".
- מתאים ללמידה בסיסית ואינו דורש משאבי מחשוב משמעותיים.

חסרונות של המודל:

1. לא יודע להתמודד עם נתונים מורכבים.
2. לא דקדקן (לעומת הפרספטורון)

THANK YOU