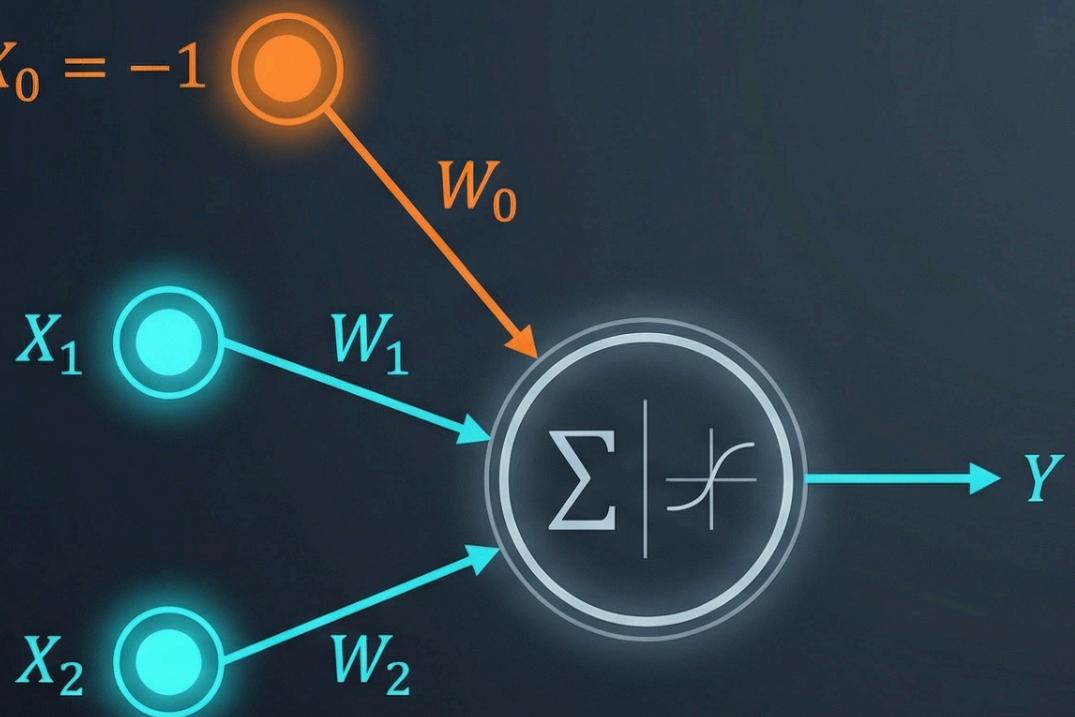


הוספת ה-Bias: העצב הנסתור

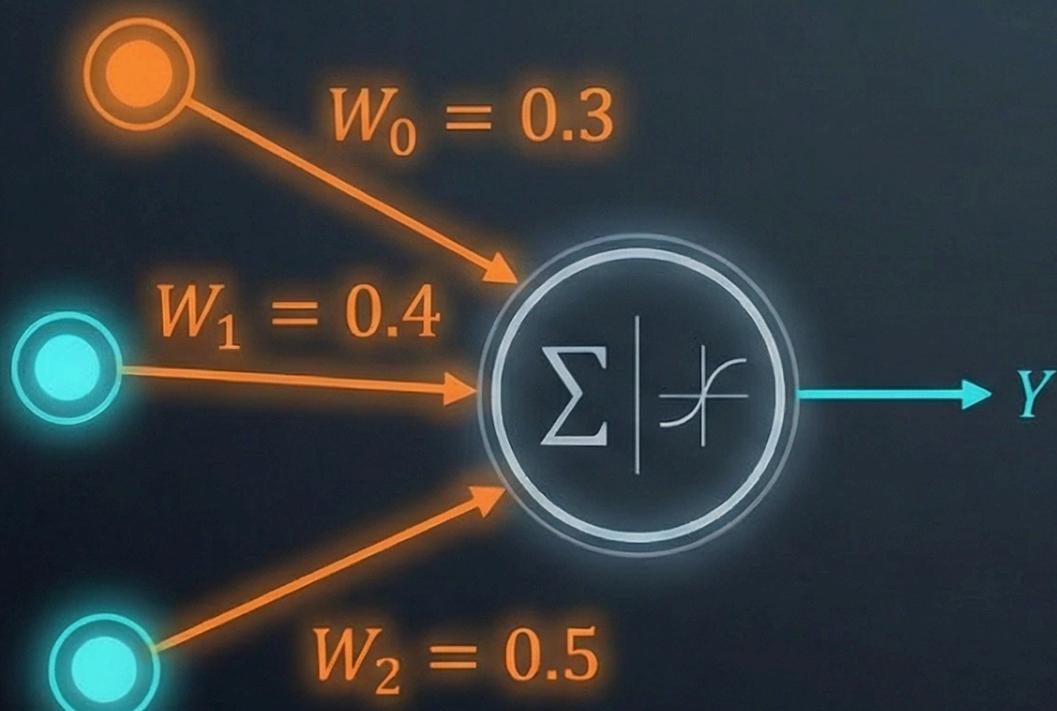


- הבועה: איך מטפלים ב-Bias (B) בצורה אחידה?
- הפתרון: הוספה קלט מלאכותי $-1 = X_0$.
- המשמעות: ה-Bias הופך לעוד משקל (W_0) בראשת.
- הנוסחה: $(AX + B)$ במקומ $(Y = \sum W_i X_i)$.

$$Y = AX + B \longrightarrow Y = W_0X_0 + W_1X_1 + W_2X_2$$

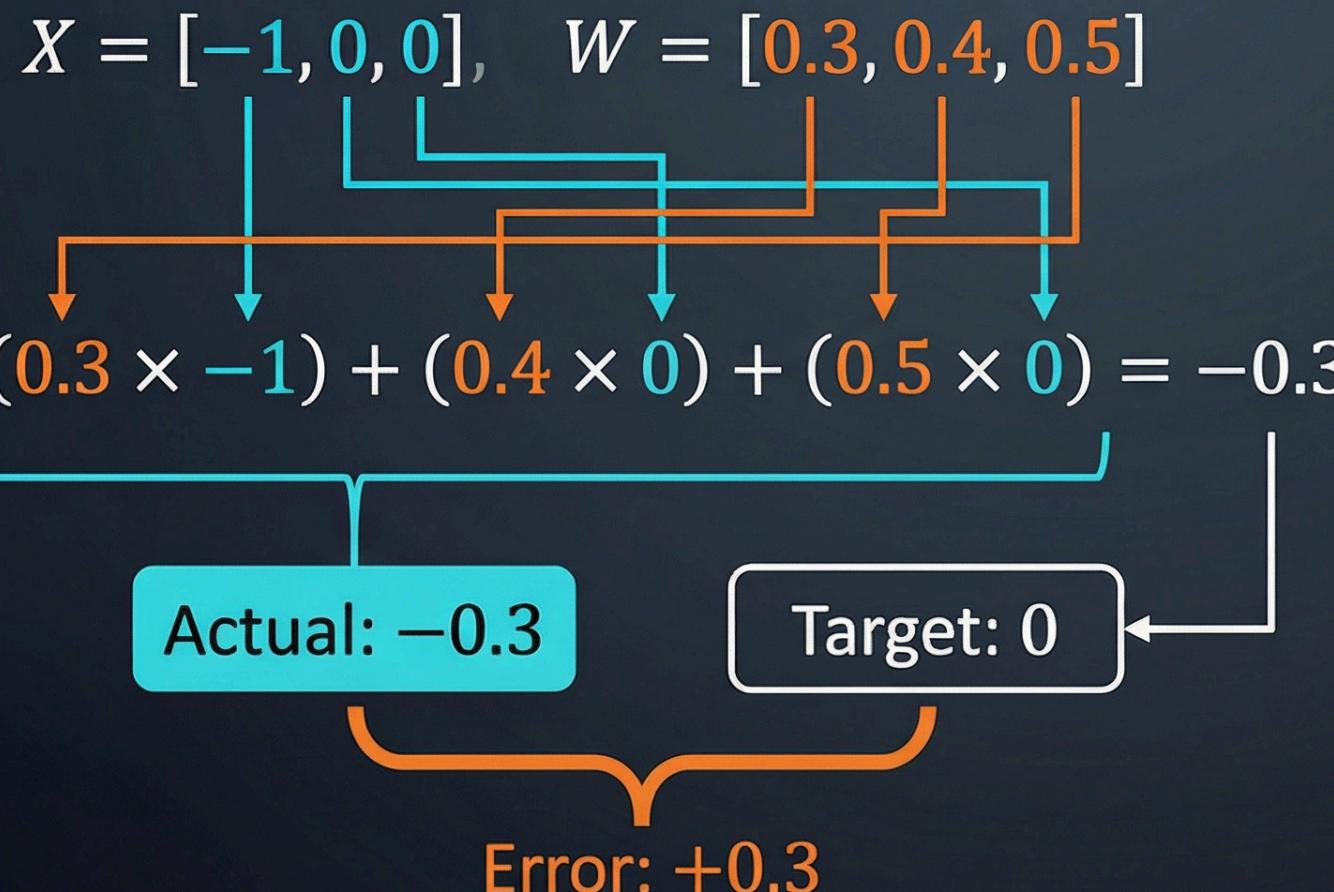
$$Y = \sum W_i X_i$$

אתחול אקראי: נקודת ההתחלה



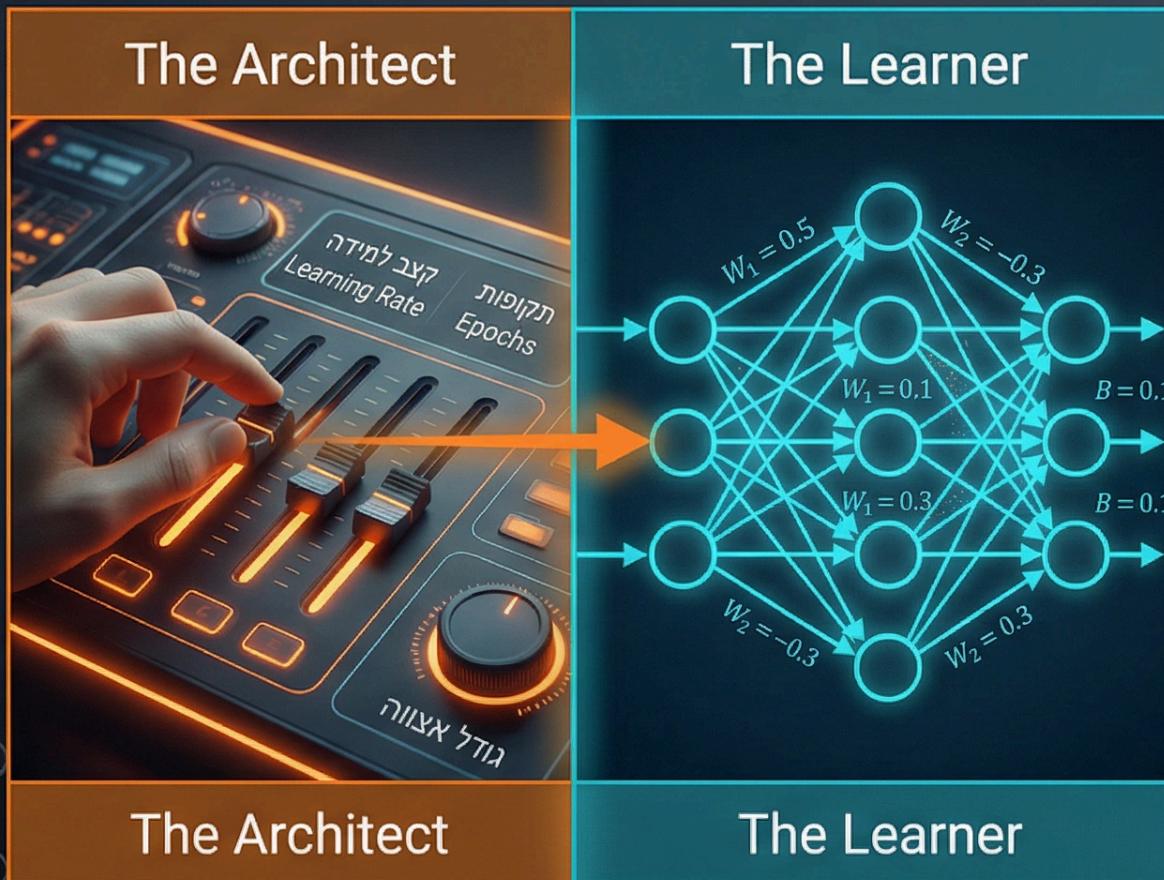
- הפעולה: הגרלת מספרים אקראיים למשקלים
- הערכים: $W_0 = 0.3, W_1 = 0.4, W_2 = 0.5$
- הסיבה: הרשת חיבת להתחיל מנוקודה כלשהי כדי ללמידה
- האנלוגיה: כמו תינוק שמתהיל ללכת בלי לדעת איר

חישוב ראשוני: המכונה בפועל (First Pass Calculation)



- הקלט: $[-1, 0, 0]$
- המשקלים: $[0.3, 0.4, 0.5]$
(Weight vector)
- הчисוב: סכום המכפלות
(Sum of products)
- התוצאה: -0.3 – (לעומת רצוי 0)
(Result vs Desired)
- השגיאה: $+0.3$
(The Error)

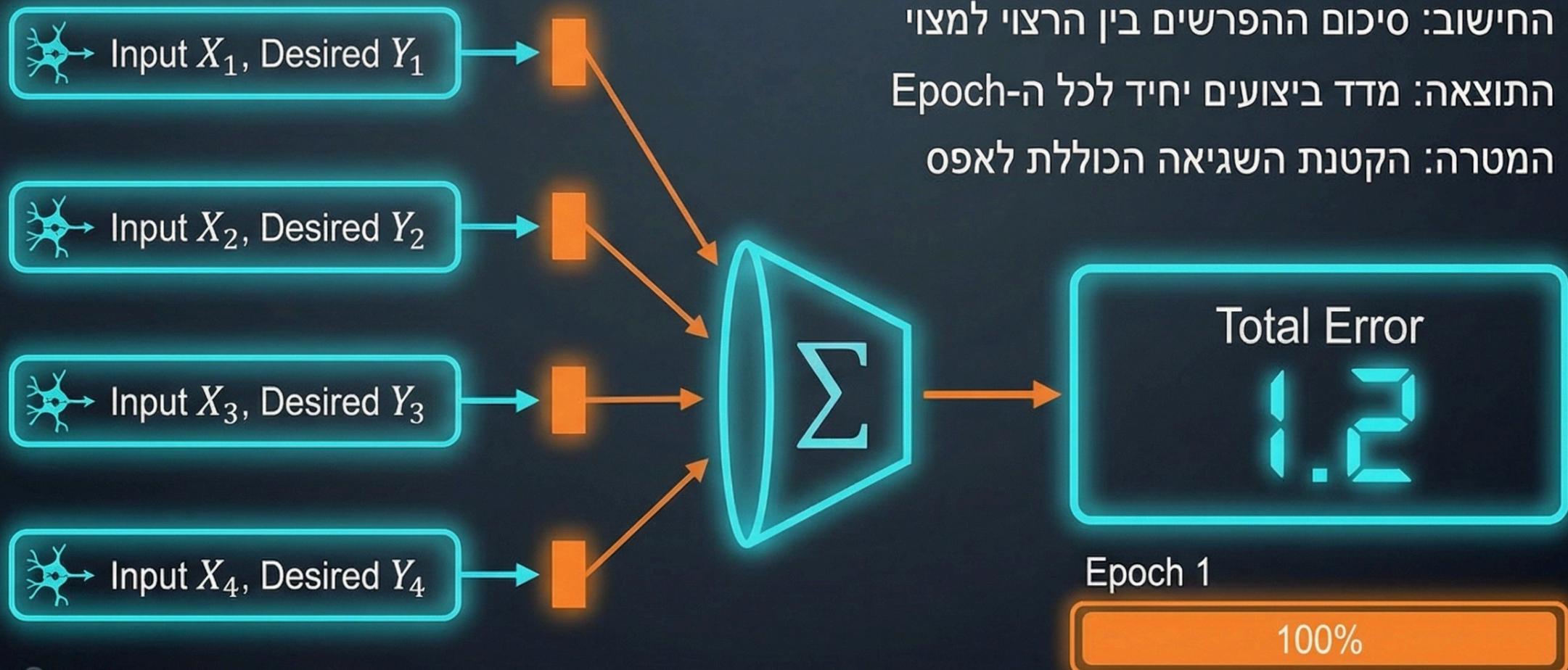
היפר-פרמטרים מול למידה: מי קובע מה?



- היפר-פרמטרים (Hyperparameters) נקבעים על ידי המשתמש (User defined).
- דוגמאות: קצב למידה (Learning Rate), מתאים לעצורת (Stopping criteria).
- פרמטרים תלמידים (Learned Parameters) נקבעים במהלך המוכנה.
- דוגמאות: משקלים (Weights, W), הטיות (Biases, B).
- החלוקת: האדם קובע את החוקים, המוכנה משחקת

חישוב שגיאה כולה (Total Error Calculation) תמנון המצב

- התהילה: מעבר על כל דוגמה בדата-סט
- הчисוב: סיכום ההפרשין בין הרצוי למצוי
- התוצאה: מדד ביצועים יחיד לכל ה-Epoch
- המטרה: הקטנת השגיאה הכוללת לאפס



Step 0: Calculating Total Loss

Sample	Input (X)	Desired (Y)	Actual (\hat{Y})	Delta (δ)	Delta ²
1	[1, 0]	0	0.3	0.3	0.09
2	[1, 1]	1	0.8	-0.2	0.04
3	[0, 1]	1	0.7	-0.3	0.09
4	[0, 0]	0	0.1	0.1	0.01

Squared to ensure positive values

Loss Function
Total Loss = $\sum(\delta^2)$

$$= 0.09 + 0.04 + 0.09 + 0.01 \\ = \mathbf{0.23}$$

Why Square the Delta?
Squaring ensures all errors are positive, regardless of direction

(like hitting a target - distance matters, not direction)

$$\delta = \text{Desired} - \text{Actual}$$

שלב 0 - חישוב השגיאה הכוללת (Total Loss)

- **המצב ההתחלתי**
לאחר הגרלת משקלים (W) אקראיית, הרשות מייצרת פלטיהם עם שגיאות.
- **חישוב דلتא (δ)**
לכל דוגמה ב-Dataset, מחשבים את ההפרש בין הפלט הרצוי (Label) לפלט שהתקבל בפועל.
- **פונקציית השגיאה (Loss Function)**
השגיאה הכוללת מחושבת כסכום ריבועי כל הדeltas (δ^2).
מעלים בריבוע כדי להבטיח שהשגיאה תמיד חיובית.
- **מטרה**
כימות "הקטטרופה" שנוצרה מהනיחס הראשוני של המשקלים.

Gradient Descent

מטרה: תיקון המשקלים בעזרת פונקציית השגיאה

Calculating ΔW_0 for a Single Weight

$$\Delta W_0 = -\eta \times \sum(\delta_i \times x_{0i})$$

Step 1: Gather Data

Sample	δ_i	x_{0i}	$\delta_i \times x_{0i}$
1	0.3	1	0.3
2	-0.2	1	-0.2
3	-0.3	1	-0.3
4	0.1	1	0.1

Step 2: Sum All Products

$$= 0.3 + (-0.2) + (-0.3) + 0.1 \\ \underline{\underline{= -0.1}}$$

Step 3: Multiply by $-\eta$

$$\eta = 0.05 \text{ (Learning Rate)} \\ \Delta W_0 = -0.05 \times -0.1 \\ \underline{\underline{\Delta W_0 = 0.005}}$$

Visual Flow

Errors from
all samples



Input values
($x_0 = 1$)



Sum all
products



× Learning Rate
($\eta = 0.05$)



Result: ΔW_0

חישוב התיקון (ΔW) עבור משקל בודד

- **התהיליך:** מבוצע בנפרד עבור כל משקל בראשת (W_0, W_2 , ...). (W_0, W_1, W_2 ...).

- **פירוק הנוסחה:** התיקון (ΔW) הוא מכפלה של קבוע הלמידה (η) בסכום של ($\text{שגיאה} * \text{קלט}$) עבור כל הדוגמאות.

דוגמה לחישוב ΔW_0 :

$$\Delta W_0 = -\eta * \sum(\delta_i * x_{0i})$$

- η = קבוע למידה (למשל 0.05)

- δ_i = השגיאה עבור דוגמה i

- x_{0i} = ערך הקלט עבור דוגמה i

- **התוצאה:** ערך מספרי המיצג את ה"תוספת" למשקל.

Step 1: Updating All Weights

Weight	W_old	ΔW	Operation	W_new
w_0	0.15	+0.005	$0.15 + 0.005$	0.155
w_1	0.20	-0.010	$0.20 - 0.010$	0.190
w_2	0.25	+0.015	$0.25 + 0.015$	0.265

Old Weights
(from random init)

Corrections
(calculated via GD)

New Weights
(ready for next iter)

Weight Update Process

1. Calculate ΔW for each weight using gradient descent
2. Add ΔW to old weight (positive ΔW increases, negative decreases)
3. Replace old weights with new weights for next iteration

שלב 1 - עדכון כל המשקלים

• ביצוע התיקון

לאחר חישוב ערכי ה- ΔW לכל המשקלים ($\Delta W_0, \Delta W_1, \Delta W_2$), מבצעים את העדכון.

• הנוסחה

$$W_{\text{new}} = W_{\text{old}} + \Delta W$$

- אם ΔW שלילי, המשקל יקטן.
- אם ΔW חיובי, המשקל יגדל.

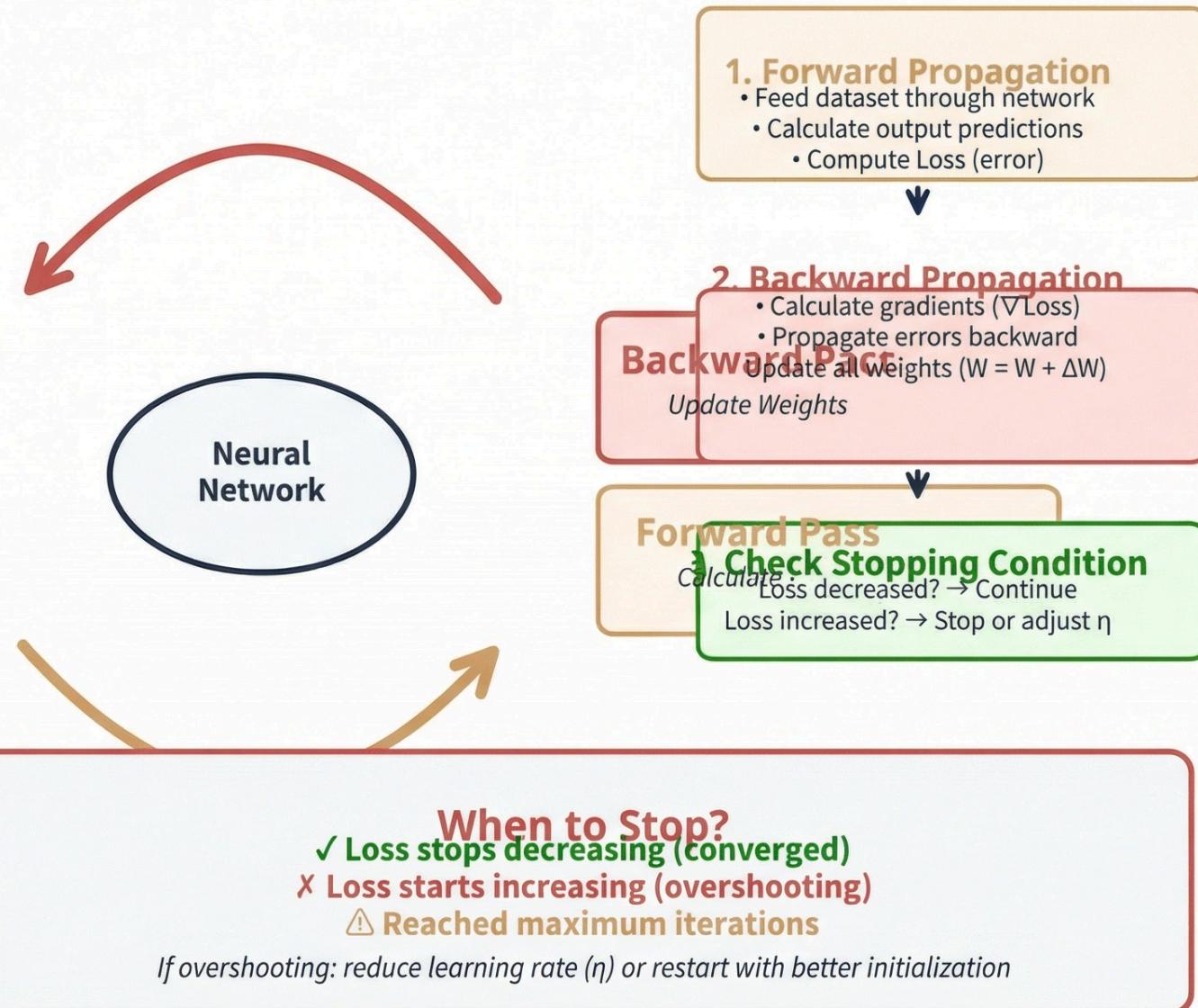
• השלב הושלם

המשקלים הישנים "נזרקים לפח", והרשות משתמשת כעת בסט המשקלים החדש והמתוקן.

• התוצאה

נוצר סט משקלים חדש (W^3, W^2, W^1, \dots) שאמור (אך לא מובטח) להוביל לשגיאה נמוכה יותר באיתרציה הבאה.

התהיליך האיטרטיבי: Forward & Backward Propagation



המחזריות: התהליך ברשות עובד בಗלים של קדימה ו'אחורה'.

הזרמת הדadata: Forward Propagation הדגוה סט, חישוב פלט ושגיאה (Loss).

חישוב נגזרות והתפשטות לאחר תיקון המשקלים (WΔ): Backward Propagation חישוב נגזרות והתפשטות לאחר תיקון המשקלים (WΔ).

תנאי עצירה: חוזה עד שהשגיאה מפסיקת לקטון (התכנסות) או מתחלת לגדול לגודל (Overshooting).

אופטימיזציה: במקרה של Overshooting, מקטינים את קצב הלמידה (η).

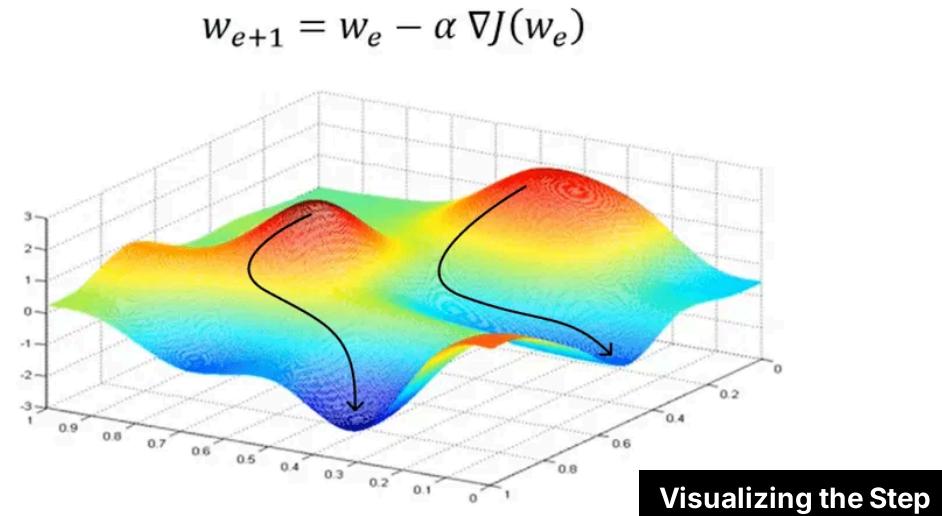
איטרציה - הצעד הבסיסי בתהיליך הלמידה

הגדירה: Iteration

צעד בודד בתהיליך הלמידה שבו הרשות מעדכנת את המשקלים
שליה על סמך השגיאה שנמדדה.

טהיליך האיטרציה:

- 1 הכפלת המשקלים ($X \times W$)
- 2 חישוב הפלט המצוי (Predicted)
- 3 חישוב ההפרש מהרצוי (Error)
- 4 חישוב השגיאה הכלולית (Total Loss)
- 5 עדכון המשקלים (Gradient Descent)

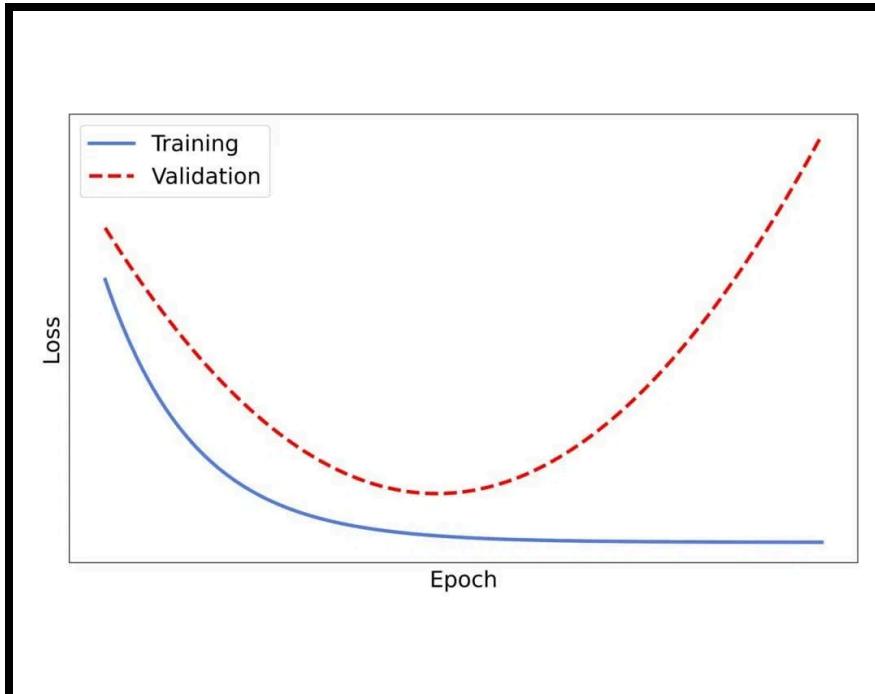


כל איטרציה היא "צעד" אחד במורד הגבעה לכיוון המינימום של פונקציית השגיאה.

דוגמה מהשיעור:

```
Iteration 0: Loss = 0.30
Iteration 1: Loss = 0.26
>> Improvement: -0.04
```

אפק (EPOCH) - מחזור למידה שלם



ציר ה-X בגרפי למידה הוא תמיד מספר האפוקים

אייזון עדין:

מעט מדי אפוקים ← Underfitting
יותר מדי אפוקים ← Overfitting

מקור: תמלול שיעור L2-NNforANDXOR_part14

הגדלה: Epoch

מעבר מלא אחד של הרשות על כל דוגמאות האימון (Training Set).

איטרציה (ITERATION)	אפק (EPOCH)
צעד בודד (Batch אחד)	מחזור שלם (כל הדאטה)
עדכון משקלים מקומי	סיום מעבר על כל המידע

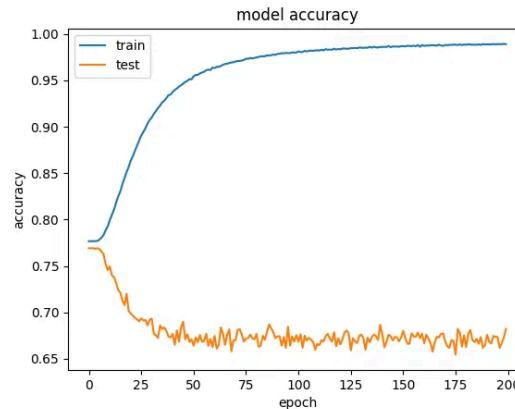
דוגמה מסכנית:

Total Data = 1000 samples
Batch Size = 100 samples

Epoch = 10 Iterations 1

גרף LOSS VS EPOCH - המוניטור של הלמידה

Typical Learning Curve



tabניות נפוצות

- ✓ ירידת חלקה: למידה תקינה
- ~~ רUIDות: Learning Rate אבוה
- עליה: באג או הຕבדרות

דוגמה מהשיעור

התוצאות הלמידה:
Epoch 0: Loss = 0.30
Epoch 1: Loss = 0.26
מסקנה: הרשת לומדת ומשתפרת.

מה רואים בגרף?

- ציר X: זמן (Epochs)
- ציר Y: שגיאה (Loss)
- המטרה: להגיע למינימום

ההבדל בין TRAINING LOSS ל VALIDATION LOSS

תרחישים אפשריים:

מצב אידיאלי

שניהם יורדים ביחד = למידה והכללה טובה.



Overfitting (שינוי)

Training יורד, אבל Validation מתחילה לעלות.



Underfitting

שניהם גבוהים ולא יורדים = הרשות לא מצליחה ללמוד.



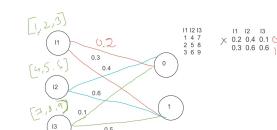
מתי עצרים? (Early Stopping)
ברגע ש-Validation Loss מפסיק להשתפר!

Training Loss

השגיאה על הדאטה שהרשות **לומדת** ממנה.
אמור לרדת באופן עקבי ככל שהאימון מתתקדם.

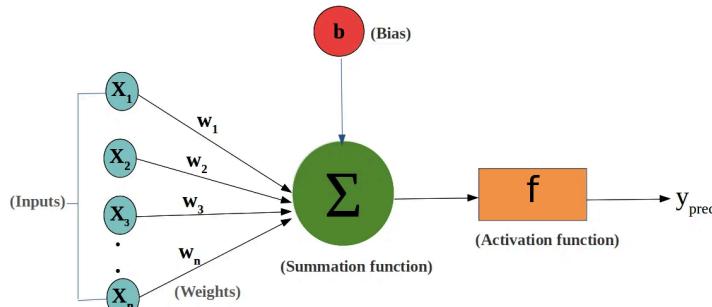
Validation Loss

השגיאה על דאטה שהרשות **לא ראתה**.
המדד האמיתי **ליקולת ההכללה** (Generalization).



חישובת התרגול הידני - "מי שלא נפל מסוס..."

"מי שלא נפל פעם אחת מסוס, לא נקרא שהוא רוכב סוסים. מי שלא עשה פעם אחת זאת באופן ידני... לא יכול להיות באמת איש Deep Learning".



הבנת המבנה הפנימי דורשת התנסות מעשית

למה זה קריטי?

- ✓ הבנה עמוקה של איך המשקלים משתנים
- ✓ פיתוח אינטואיציה ל- **Hyperparameters**
- ✓ יכולה "להרגיש" את המספרים ולא רק לראות אותם

המלצת: קחו דף ועט (או אקסל) ותריצו רשת פשוטה ידנית!

שתי שיטות לעדכון משקלים

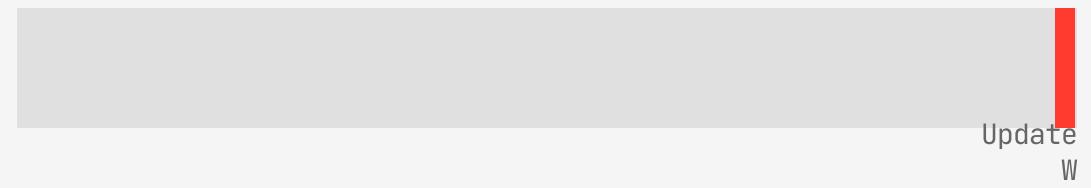
Mini-Batch .2

- מחלקים למקבצים (Batches).
- אחרי כל **Batch**: חישוב Loss ועדכן.
- עדכנים תכופים ומהירים.
- דורש ניהול זיכרון חכם.



Epoch-Based .1

- עוברים על כל הדadata-סט.
- מחשבים Loss בסוף.
- מבצעים Gradient Descent **פעם אחת**.
- עדכן איטי אך יציב.



שיטת Batch: הבעייה והפתרון

THE PROBLEM

שיטת **Batch = Epoch** היא בזבזנית וαιטית.
היא דורשת דטה-סט ענק כדי לכסות את כל מרחב המדגם,
ומעדכנת את המערכת רק בסוף התהליכי הארו.

THE SOLUTION: MINI-BATCH

לומדים "תוך כדי תנועה".
מחלקים את הדטה למקבצים קטנים.
1 → עדכון Batch 2 → W (עם W חדש) → עדכון W...
התוצאה: הטעינה מהירה יותר וניצול יעיל של המידע.

שלושת מקרי הקיצון

Batch = 1 (SGD)

R (All Samples)	Updates/Epoch
Low	Memory
Fast	Speed
Noisy	Stability

Mini-Batch

Tens~	Updates/Epoch
Medium	Memory
Balanced	Speed
Good	Stability

Batch = Epoch

1	Updates/Epoch
High	Memory
Slow	Speed
Very Stable	Stability

יתרונות וחסרונות: (Batch = 1 (SGD)

חיסרון 1: זמן חישוב

- חישוב W מחדש אחרי כל דגימה.
- תקורה (Overhead) גבואה מאד.
- $M \text{ פעולות} \times R \text{ דוגימות} = \text{המון זמן}.$

חיסרון: זיכרון

- דורש זיכרון מינימלי.
- מעלים רק דגימה אחת בכל רגע.
- מתאים ל-GPU חלש.

חיסרון 2: דגימות לדעש

- דגימה אחת שגوية (Outlier) זורקת את הרשות "מהמסלול".
- גורם לתנודתיות גבואה בתוכניות.

❖ דוגמה:

רופא כתוב 60 ק"ג במקום 60 ק"ג



Loss ענקית → עדכון W קיזוני → הרשות "משתגעת"

מתי משתמשים במה? (כללי אצבע)

Batch = Epoch

- ✓ דאטה-סט קטן
- ✓ זיכרון GPU ענק
- ✗ דאטה-סט ענק (בזבוני)

Mini-Batch (Default)

- ★ האיזון הטוב ביותר
- ✓ מתאים לרוב המקרים
- ✓ ניצול יעיל של ה-GPU

Sizes: 32, 64, 128, 256

Batch = 1 (SGD)

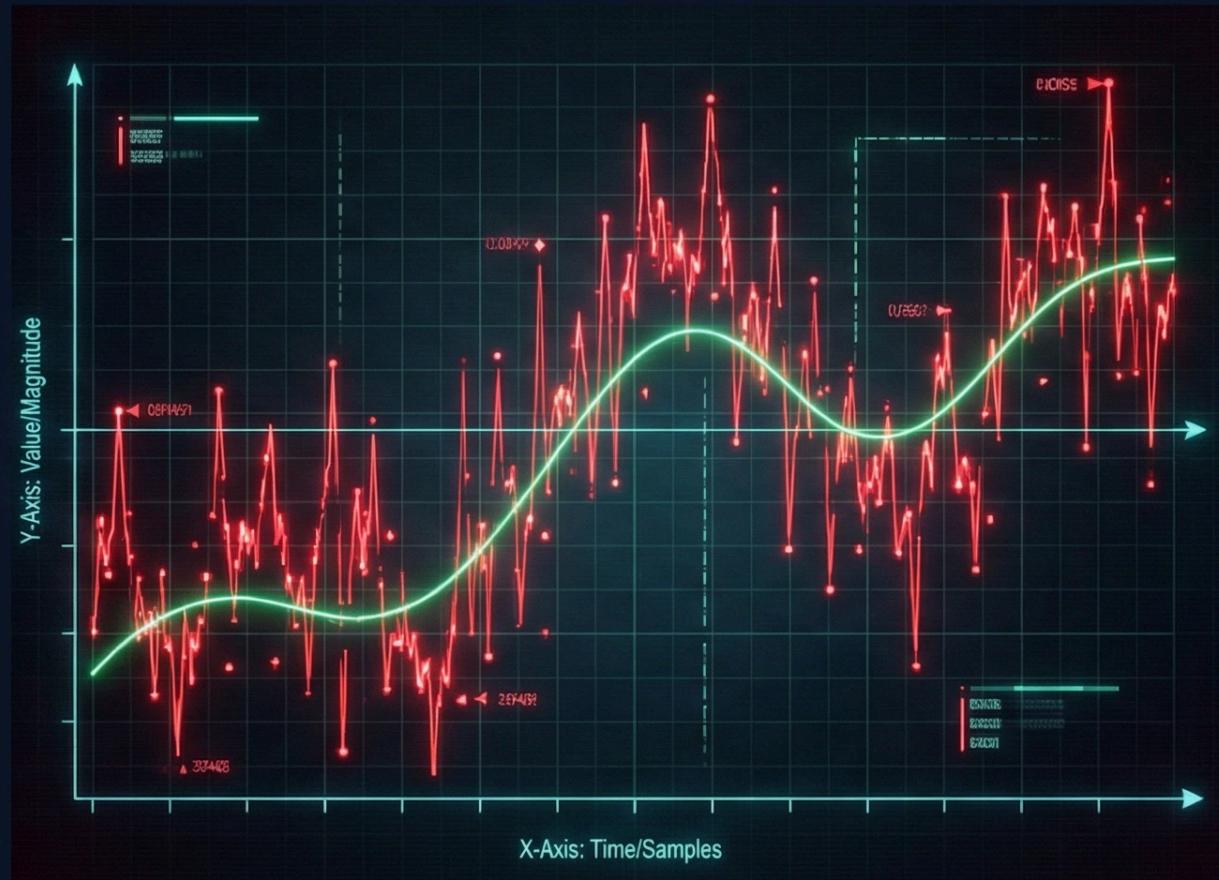
- ✓ זיכרון GPU מוגבל מאוד
- ✓ דאטה נקייה (ללא רעש)
- ✗ כישיש Outliers (רגיש!)



כל הברזל: גודל הזיכרון (GPU Memory) הוא החסם העיקרי שקבע את גודל ה-Batch.

הבעיה:>Data רועש ולא אמין

(The Problem: Noisy & Unreliable Data)



המקור: נתונים שנאספו ידנית
(למשל, ספירת מכוניות בצומת)



האיךות: טעויות אנוש, חוסר ריכוז,
כתב לא ברור



התוצאה:>Data "רועש" שקופץ
לכל הכוונים



ההשלה: אי אפשר לסמוך על כל
דוגמה בודדת



(Low Pass Filter) מASNן רעש**ים**: Batch Size



Stability & Momentum

העיקרון: Batch Size גדול משמש כASNן
ASNן תדרים נמוכים (LPF)
(Principle: Large Batch Size acts as LPF)



הפעולה: ביצוע ממוצע על קבוצת דוגמאות
(Action: Averaging a group of examples)



התוצאה: חילקת הרעש וקבלת כיוון יציב
(Result: Smoothing noise, stable direction)

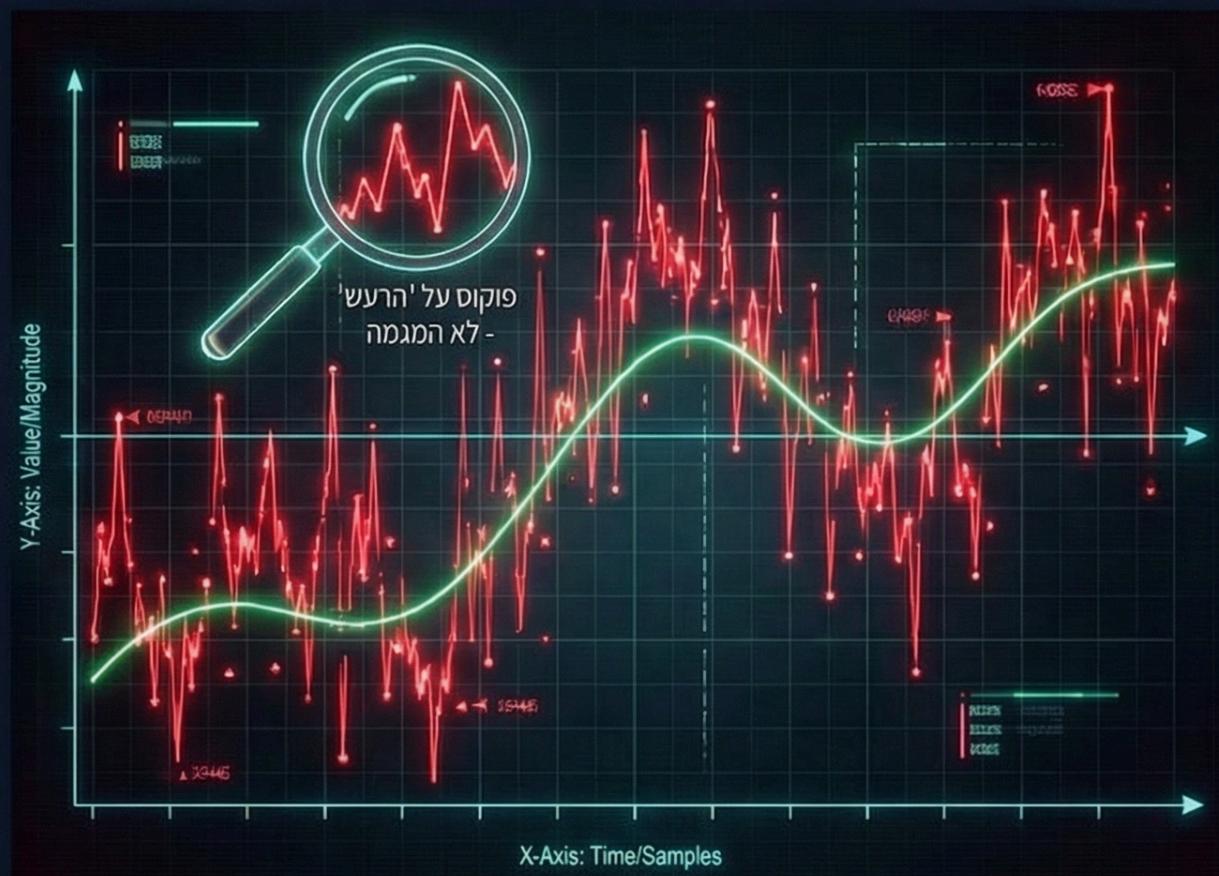


הalogיה: הזאת אונייה ענקית (יציבה)
לעומת סירה קטנה (מתנדנת)
(Analogy: Moving a huge ship vs. a small boat)



הסיכון: התאמת יתר ב-**Batches** (Overfitting) קטנים

(The Risk: Overfitting)



הבעיה: עדכון תכוֹף מדי על סמך
מעט נתוניים



התוצאה: המערכת לומדת את
"הרעש" במקום את המגמה



האנלוגיה: שינוי תשובות לבחן בלי
להבין את החומר

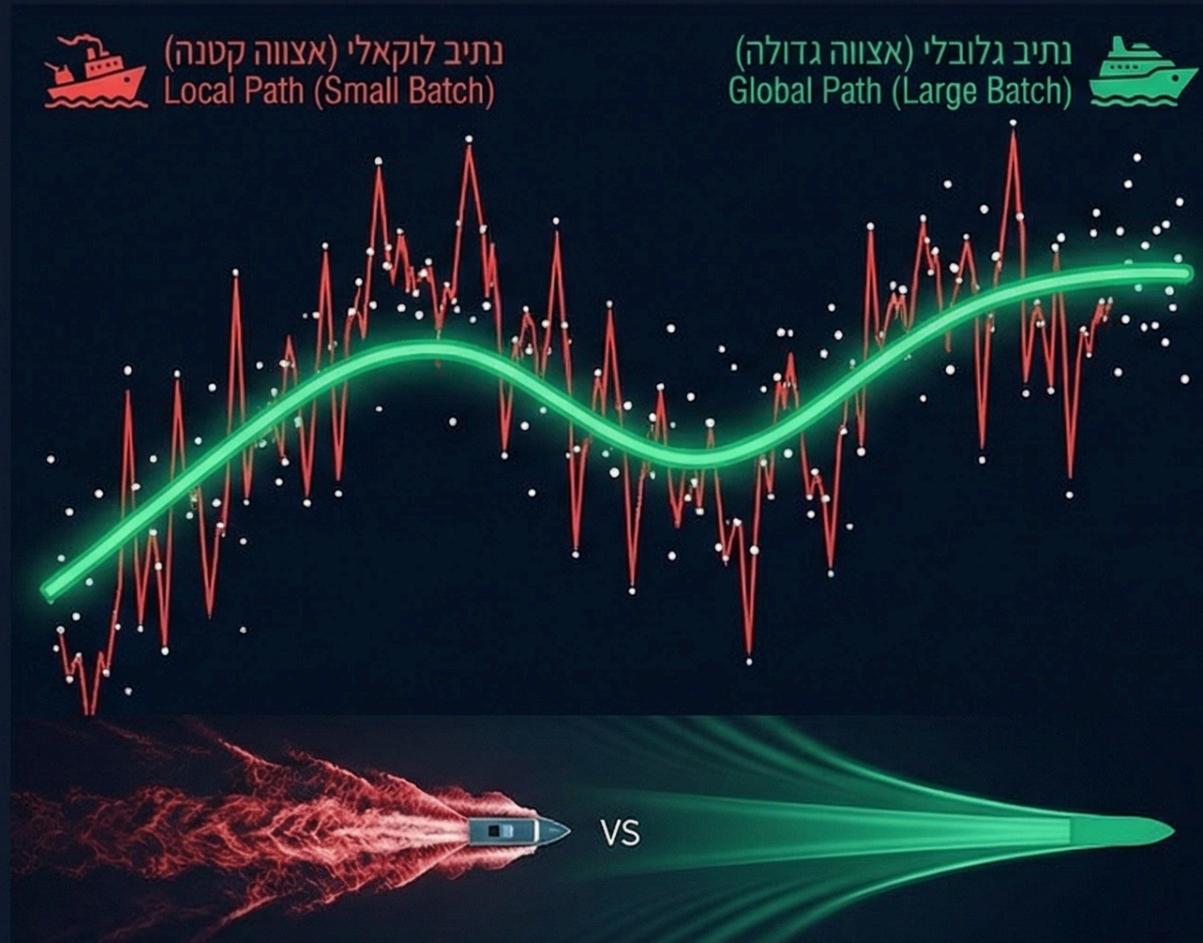


במבחן המציגות: ביצועים מעולים באימון, גראעים בננתוניים חדשים



גלוּבְּלִי מֹול לַוקָּלִי: יִצְׁبָּות הַפְּתָרוֹן

(Global vs. Local: Solution Stability)



תיקון מקומי (Local Fix): תגובה
 לכל רעש קטן



תיקון גלובלי (Global Fix): ראייה
רחבה ויציבה



ההשוואה: "רַב סָרֵן שְׁמוּעַתִּי" מִול
"חָכְמַת הַמּוֹנִים"



התוצאה: פתרון רוביסטי שנוכן
למקרה הכללי



הائزון העדין: גודל Batch, משאבים וביצועים

(The Delicate Balance)



- **Batch קטן (Small Batch):**
 - ✓ **יתרון:** ורסטיליות, בדיקה תקופה (Pro: Versatility)
 - ✗ **חיסרון:** חוסר יציבות, חישוב יקר (Con: Instability)
- **Batch גדול (Large Batch):**
 - ✓ **יתרון:** יציבות, החלקה (Pro: Stability)
 - ✗ **חיסרון:** דרישת זיכרון, החלוקת יתר (Con: Memory demand)
- **המסקנה:** פשרה (Trade-off) בהתאם למשאבים ולאמיניות הדאטה.



רגסיה מול קלאסיפיקציה - ההבדל העיקרי



קלאסיפיקציה (CLASSIFICATION)

תחזית דיסקרטית - קטגוריות

השאלה: "איזה סוג?"
הפלט הוא אחת מtower קבוצה סופית.

דוגמה:
אבחן רפואי:

חוליה (1) / בריאות (0)



רגסיה (REGRESSION)

תחזית אנלוגית - מספרים רציפים

השאלה: "כמה?"
הפלט יכול להיות כל מספר על הרצף.

דוגמה:
חיזוי שער מניה לאחר:

17.8 נ

הסוד האדול - פונקציית **LOSS** חכמה

!!

"הادرת פונקציית **Loss** חכמה היא **60%** מפתרון הבעיה"

היצירתיות ב-**DEEP LEARNING**

זה לא רק מתמטיקה, זו אומנות.

חוקרים מבקרים פתרים בעיות קשות לא ע"י שינוי הרשת,
אלא ע"י הגדרה מחדש של השגיאה.

מה זה **LOSS FUNCTION**

המדד הכמותי לשגיאה בין מה שרצינו (Target) לבין מה שהרשת חצתה (Predicted).

$$\text{Loss} = \text{Distance}(\text{Target}, \text{Predicted})$$

למה תואר שני? התמחות ועומק בתחום

"דברים שרואים מפה לא רואים ממש" - כל תחום הוא עולם ומלאו.



זיהוי פנים

דатаה עצום, זמן אמת, פרטיות, התקפות סייבר.



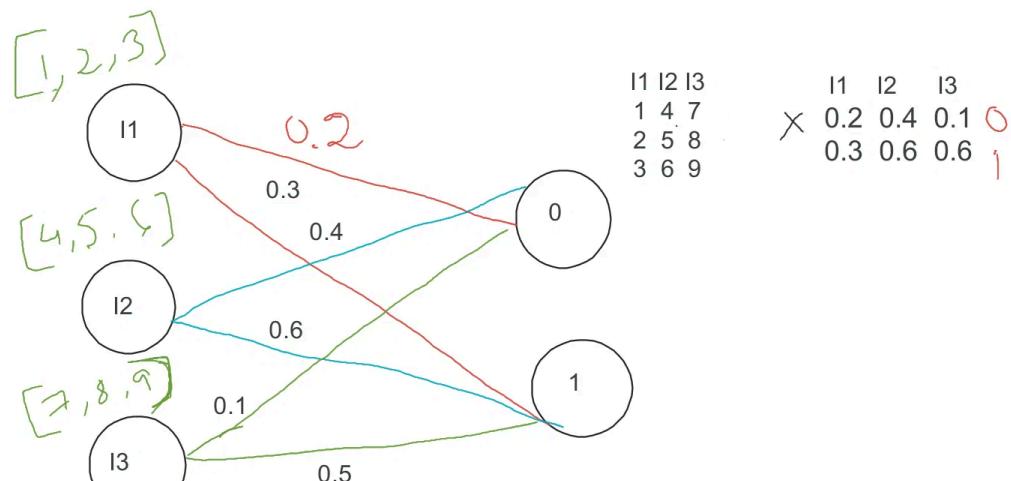
בעיות רפואיות

דיק קרייטי, דטה מועט, רגולציה, הסברתיות (Explainability).

"אנו רק התחילו, אנחנו אפילו לא נגענו בקצת של הקצה."

התקפות סייבר על DEEP LEARNING

היעד לתקיפה: המודל העצבי



התוקפים מנצלים את הרגישות של פונקציית האקטיבציה והמשקלים לשינויים זעירים בקלט.

האתגר החדש: הט夷ית המודל

FUSION ATTACK (היתוך)

ミゾג שני פרצופים לתמונה אחת.

התוצאה: המערכת מזוהה את האדם המורשה (בעל הייזה), בעוד האדם השני עובר בפועל.

ADVERSARIAL ATTACK ⚡

הוספה "רעש" בלתי נראית לעין האנושית.

התוצאה: המערכת מתבלבלת לחלוטין בסיווג, למراتות שהתמונה נראית תקינה לנו.

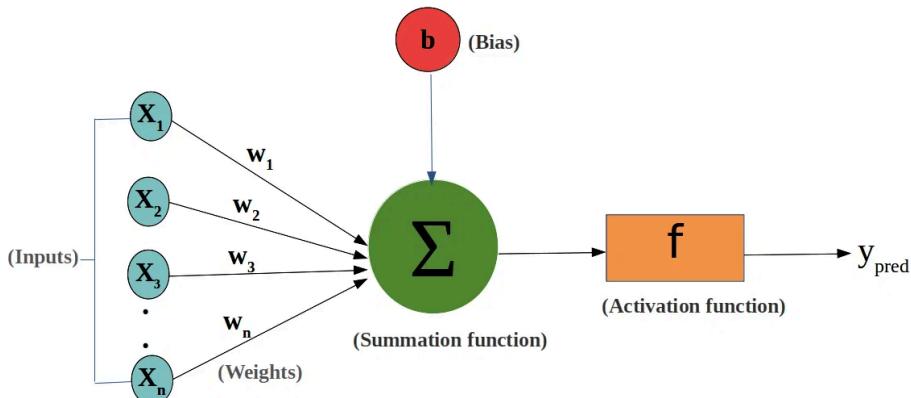
הבעיה בקורסיפיקציה עם קו לינארי

הניסיון הנאייבי

ניסיונו להתאים קו גראסיה לינארי לנקודות שמייצגות 0 ו-1.

תופעת "הקו הוז"

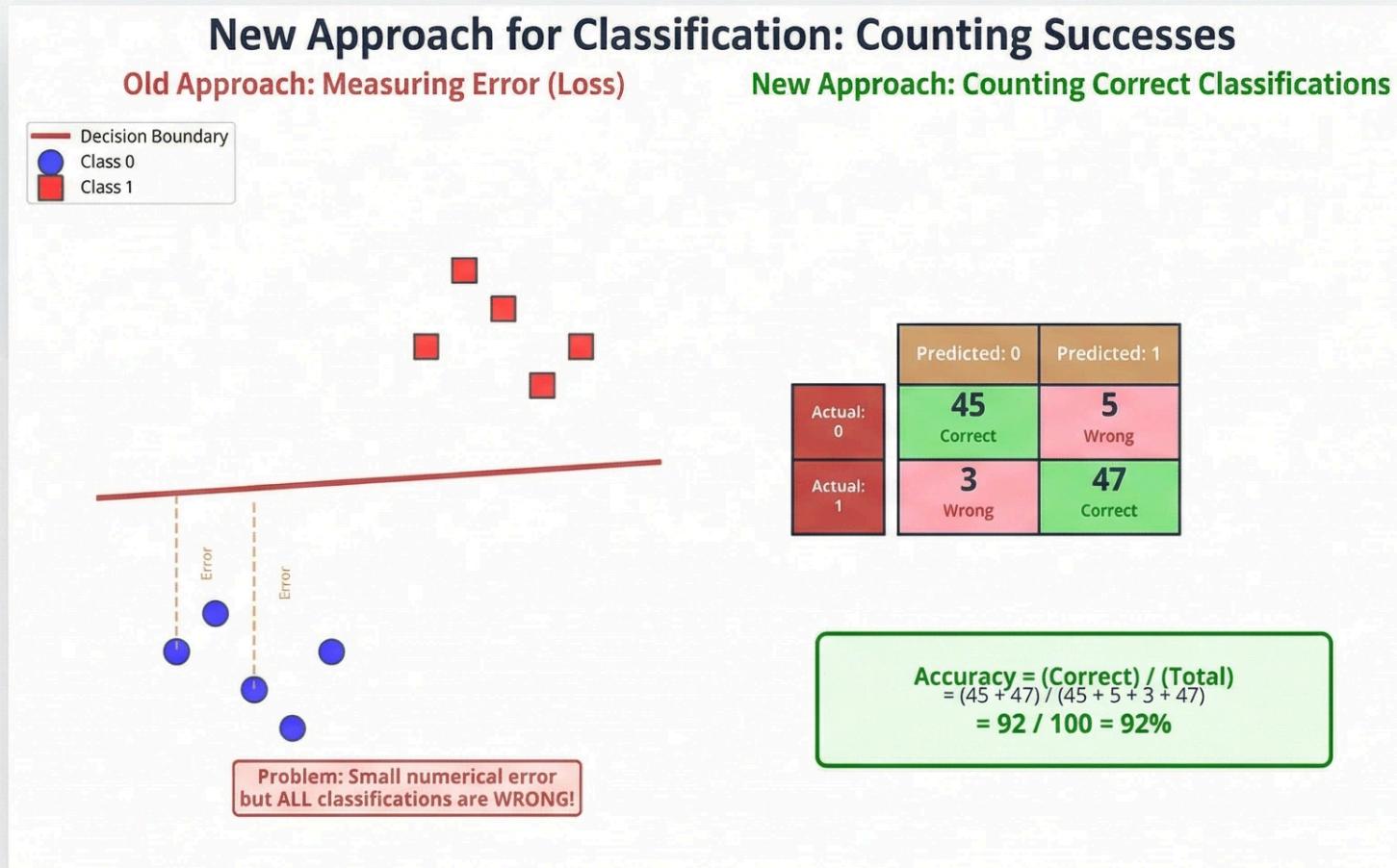
כל שימושים יותר נקודות דאטה, השיפוע של הקו משתנה ("וז").
פתרונות תלוי בכמות המדידות.



הפרמטרים (Weight & Bias) הופכים ללא יציבות כאשר מנסים לכפות גראסיה על בעיית קרסיפיקציה.

מסקנה: ב-(Big Data) Ai
אפשר להסתמך על פתרון שתלו依 בגודל המדגם.

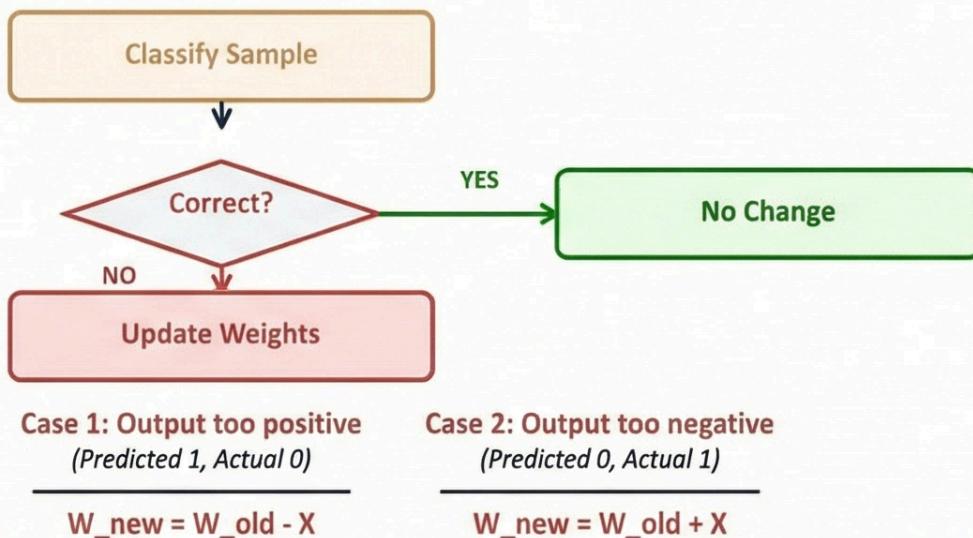
גישה חדשה לסיווג – ספירת הצלחות



- **הבעיה בגישה הקודמת:** בבעיות סיווג (Classification), מדידת גודל השגיאה (Loss) יכולה להיות מטעה.
- **פתרונות:** במקום למדוד "כמה טעינו", סופרים "כמה פעמים צדקנו".
- **המדד החדש:** Accuracy – אחז הדוגמאות שהרשות סיווגה נכון.
- **המטרה:** למקסם את ה-Accuracy במקום למזער את השגיאה המספרית.

אלגוריתם העדכון הdiskreti

Discrete Update Algorithm for Classification



Intuition: "Pull" the decision boundary toward the correct side

- If output too high → subtract input vector to reduce weights
 - If output too low → add input vector to increase weights
- This is a discrete approximation of gradient descent*

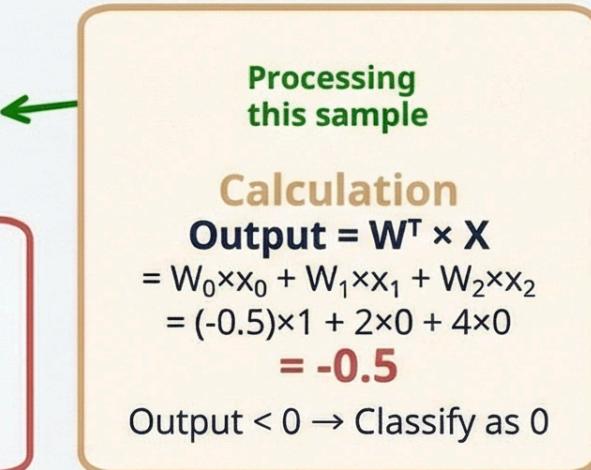
- **השראה:** קירוב של שיטת הנגזרות (Descent) למקרה הדיסקרטי (סיווג).
- **הכלל:** אם הסיווג נכון - לא משנהים את המשקלים. אם הסיווג שגוי - מבצעים תיקון.
- **נוסחת התיקון פשוטה:**
 - אם הפלט היה חיובי מדי (למשל 1 במקום 0):
 $W_{\text{new}} = W_{\text{old}} - X$
 - אם הפלט היה שלילי מדי (למשל 0 במקום 1):
 $W_{\text{new}} = W_{\text{old}} + X$
- **היגיון:** "מוסכים" את קו ההפרדה לכיוון הנכון על ידי הוספה/חסרה של וקטור הקלט עצמו.

דוגמה: שלב 1 – סיווג נכון

AND Truth Table

x_0	x_1	x_2	Output
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Initial Weights
 $W = [-0.5, 2, 4]$



CORRECT CLASSIFICATION!

Predicted: 0 | Actual: 0

No update needed - weights stay the same

- התקנה: בעית AND, משקלים התחלתיים (W), וקטור קלט ראשון (למשל $[0, 0, 1]$).
- чисוב: $X^T \cdot W$. לדוגמה: $(-0.5) \times 1 + (2 \times 0) + (4 \times 0) = -0.5$.
- חלה: התוצאה שלילית (0), ולכן הסיווג הוא 0.
- בדיקה: הפלט הרצוי (Label) הוא 0. הסיווג נכון!
- מסקנה: מכיוון שהסיווג נכון, לא מבצעים שום שינוי במשקלים (W).

דוגמה: שלב 2 - סיווג שגוי ותיקון

AND טבלת אמת

x_0	x_1	x_2	Output
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

משקלים הנוכחיים
 $W = [-0.5, 2, 4]$
(לא שינוי שלב 1)

чисוב

$$\text{Output} = W^T \times X$$
$$= W_0 \times x_0 + W_1 \times x_1 + W_2 \times x_2$$
$$= (-0.5) * 1 + 2 * 0 + 4 * 1$$
$$= 3.5$$

Output > 0 \rightarrow Classify as 1



סיווג שגוי!

חزو: 1 | בפועל: 0

עדכן נדרש - יש לתקן את המשקלים

- המשר: לוקחים את הדוגמה הבאה מה-Dataset- משקלים.
- чисוב: $\text{Output} = W^T \times X$.
- לדוגמה: $\text{Output} = W^T \times X = (-0.5 * 1) + (2 * 0) + (4 * 1) = 3.5$.
- החלטה: התוצאה חיובית (> 0), ולכן הסיווג הוא 1.
- בדיקה: הפלט הרצוי (Label) הוא 0. הסיווג שגוי!
- מסקנה: צריך לעדכן את המשקלים. מכיוון שההתוצאה הייתה חיובית מדי, נחסיר את וקטור הקלט מהמשקלים.

דוגמה: שלב 3 - ביצוע העדכון

Component	W_{old}	X	Operation	W_{new}
w_0	-0.5	1	$-0.5 \rightarrow$	-1.5
w_1	2	0	$2 \rightarrow$	2
w_2	4	1	$4 \rightarrow$	3

- **הפעולה:** ביצוע תיקון המשקלים לפי הכלל $X - W_{old} = W_{new}$ (מכיוון שהפלט היה חיובי מדי).
- **פירוט החישוב:**
 - W_0 מתחדכן: $-0.5 - 1 = -1.5$
 - W_1 מתחדכן: $2 - 0 = 2$ (לא שינוי)
 - W_2 מתחדכן:
- **התוצאה:** סט המשקלים החדש הוא [-1.5, 2, 3].
- **המשר התהלייר:** כעת עוברים לדוגמה הבאה ב-Dataset, ומשתמשים במשקלים המעודכנים הללו.