בס"ד

הפקולטה להנדסה

המעבדה לעיבוד אותות

# הפרדת דוברים בסביבה מורעשת

# במעבדה לעיבוד אותות


אריאל נוי

יחזקאל וייס


פרויקט שנה ד' לקראת תואר ראשון בהנדסה

מנחה: מרדכי מוראדי

מנחה אקדמי: פרופ' שרון גנות

אוקטובר 2024

# תוכן עניינים

Deep learning……………………………………………………………………………1
Forward and Backward process……………………………………………………2
hyperparameters…………………………………………………………………………3
Loss functions……………………………………………………………………………5
Soft max……………………………………………………………………………………8
Gradient descent…………………………………………………………………………9
Optimizer…………………………………………………………………………………11
Neural network layers………………………………………………………………13
Activation functions…………………………………………………………………17
Dropout……………………………………………………………………………………21
Batch Normalization…………………………………………………………………22
Dataset……………………………………………………………………………………24
STFT…………………………………………………………………………………………26
הצגת הבעיה והפיתרון……………………………………………………………28
נתוני הפרויקט ומושגים…………………………………………………………32
תיאור המודל……………………………………………………………………………39
Masking Net………………………………………………………………………………40
PCL……………………………………………………………………………………………48
תוצאות……………………………………………………………………………………55
סיכום…………………………………………………………………………………………56

# :Deep learning

## למידה עמוקה

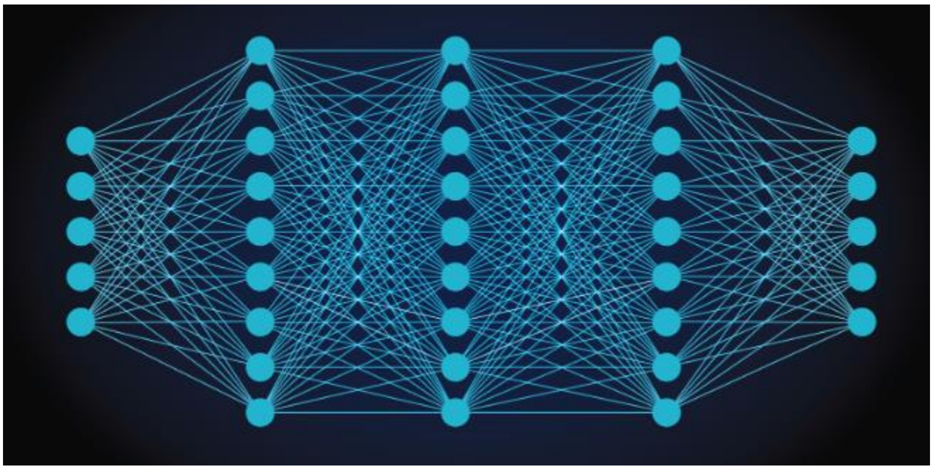
מערכת למידה עמוקה מתבססת על רשת נוירונים ,היא בעצם פונקציה מאוד גדולה ומורכבת שיש לה הרבה פרמטרים שניתן לשנות.
בשלב האימון אנחנו מכניסים המון דוגמאות מתויגות למערכת ונותנים למערכת לקבוע פרמטרים ובסוף לתת פרדיקציות עבור הדוגמאות.
בעזרת פונקציית ה loss (שעליה נרחיב בהמשך) אנו מודדים כמה טעות יש לנו בין הפרדקציות לבין התיוג האמיתי. ואז באמצעות תהליך למידה בכל איטרציה של עוד דוגמאות שעוברות במערכת, אנחנו מעדכנים את הפרמטרים עד שכל פעם ה loss קטן והמודל צודק יותר ויותר עם הפרדקציות שלו.

אחרי שסיימנו את שלב האימון עוברים לשלב הוולידציה בו בודקים שאכן הפרמטרים שנלמדו נכונים ואז בודקים את זה על דוגמאות שהמודל לעולם לא ראה - test, ובודקים שה loss הוא קטן.

למידה עמוקה היא תת-תחום של למידת מכונה המתמקד בשימוש ברשתות נוירונים מלאכותיות בעלות שכבות רבות (רב-שכבתיות). בשונה מלמידת מכונה קלאסית, בה לרוב יש צורך בהכנה ידנית של התכונות (feature engineering) שאותן המודל ילמד, למידה עמוקה מסוגלת ללמוד ולהפיק תכונות ישירות מהנתונים הגולמיים.

במילים אחרות, למידה עמוקה יכולה לגלות דפוסים ומבנים מורכבים מאוד בנתונים על ידי חיבור של שכבות רבות של נוירונים, שבאמצעותם היא מעבדת את המידע בהדרגה. ככל שהרשת "עמוקה" יותר – כלומר, ככל שיש בה יותר שכבות – כך היא יכולה ללמוד ייצוגים מורכבים יותר של המידע. זה הופך אותה למתאימה במיוחד למשימות כמו:

- **עיבוד תמונה**: זיהוי עצמים, זיהוי פנים, ועוד.
- **עיבוד שפה טבעית**: תרגום מכונה, ניתוח רגשות, יצירת טקסטים.
- **זיהוי דיבור**: זיהוי מילים, תמלול שיחות.
- **מערכות המלצה**: המלצות על מוצרים, סרטים ועוד.



**מהי רשת נוירונים (Neural Network) ?**

רשת נוירונים היא מבנה חישובי בהשראת המבנה והתפקוד של המוח האנושי. היא מורכבת מצמתים שנקראים "נוירונים", שמחוברים זה לזה באמצעות קשרים (קשתות) המיוצגים על ידי משקלים. רשתות נוירונים מורכבות משכבות של נוירונים שמטרתן לבצע חישובים הדרגתיים על הנתונים כדי להבין את הדפוסים והמבנים שבהם. רשתות נוירונים מתחלקות לשלושה סוגים עיקריים של שכבות:

1. **שכבת קלט (Input Layer):** זו השכבה הראשונה שמקבלת את הנתונים הגולמיים. כל נוירון בשכבת הקלט מייצג תכונה (feature) מסוימת מהקלט.
2. **שכבות חבויות (Hidden Layers):** אלו שכבות פנימיות שמבצעות את החישובים העיקריים, שבהן הרשת מוצאת את הדפוסים והמבנים במידע. שכבות אלו הן אלו שיוצרות את "העומק" של הרשת.
3. **שכבת פלט (Output Layer):** השכבה האחרונה שמוציאה את התוצאה הסופית של הרשת, כמו למשל סיווג לתווית מסוימת או ניבוי ערך מספרי.

כל נוירון מבצע חישוב המבוסס על הקלטים שהוא מקבל ומהווה שילוב לינארי של ערכים עם משקולות. לאחר מכן, התוצאה עוברת דרך פונקציית אקטיבציה (כגון ReLU או סיגמואיד) שמוסיפה לה אי-לינאריות. כך הרשת מסוגלת ללמוד דפוסים לא-לינאריים מורכבים.

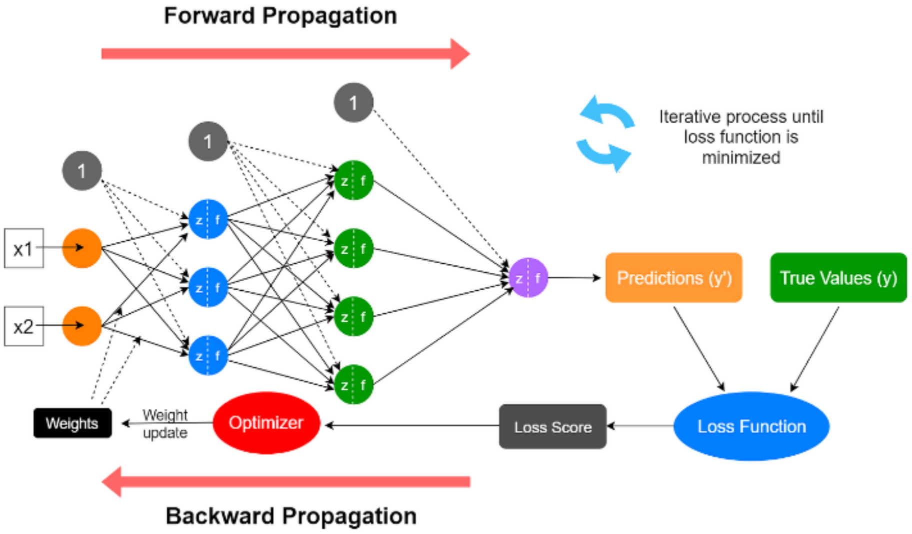
במהלך האימון, רשת הנוירונים מעדכנת את המשקלים שלה באמצעות תהליך שנקרא Backpropagation במטרה למזער את השגיאה או פונקציית ההפסד שלה. עם הזמן ודרך תהליך זה, הרשת "לומדת" לשפר את הדיוק שלה.

רשתות נוירונים מהוות את אבני הבניין של מודלים עמוקים יותר כגון רשתות קונבולוציה (CNN), רשתות חזרתיות (RNN) ועוד, שמותאמות למשימות ספציפיות בעיבוד תמונה ועיבוד רצפים, בהתאמה.

## תהליך forward ותהליך backward:

במהלך האימון של רשתות נוירונים בלמידה עמוקה, מתבצעים שני שלבים מרכזיים:

**Forward Pass** ו- **Backward Pass** שני השלבים הללו עובדים יחד כדי לעדכן את משקלי הרשת ולהפוך אותה למדויקת יותר.



**Forward Pass**

בשלב ה- Forward Pass הקלט עובר דרך הרשת מלמעלה למטה, מהשכבה הראשונה ועד לשכבת הפלט. התהליך כולל את השלבים הבאים:

1. **חישוב ערכי הנוירונים**: הקלט מוזן לשכבת הקלט, ולאחר מכן מועבר דרך כל שכבה חבויה לפי הסדר. בכל שכבה, הנוירונים מבצעים חישוב המבוסס על משקלי הקשרים ועל פונקציות האקטיבציה שלהם.
2. **פלט**: החישוב ממשיך בשכבות החבויות עד שהוא מגיע לשכבת הפלט, שבה מחושבת התוצאה הסופית. התוצאה הזו מושווית עם התוצאה הרצויה (מהנתונים המתויגים) כדי לחשב את השגיאה באמצעות פונקציית הפסד (Loss Function).

השלב הזה בעצם מאפשר לרשת "לנבא" את התוצאה על סמך המשקלים הנוכחיים.

**Backward Pass (Backpropagation)**

בשלב ה- Backward Pass הרשת מעדכנת את המשקלים שלה כדי למזער את השגיאה שחושבה

ב Forward Pass  שלב זה מתבצע בצורה של למידה חזרתית:

1. **חישוב שגיאה עבור כל שכבה**: השגיאה מחושבת בשכבת הפלט ואז מועברת אחורה לשכבות הקודמות כדי להבין איך השגיאה משפיעה על כל שכבה ושכבה.
2. **חישוב גרדיאנט (Gradient)**: בשלב זה, מחושב השיפוע של פונקציית ההפסד ביחס לכל אחד מהמשקלים. תהליך זה מתבצע באמצעות נגזרות (חישוב מתמטי) ומאפשר להבין כיצד לשנות את המשקלים כדי להקטין את השגיאה.
3. **עדכון משקלים**: לבסוף כל משקל מעודכן בהתאם לשיעור הלמידה (Learning Rate), במטרה להקטין את פונקציית ההפסד. תהליך זה עוזר לרשת "ללמוד" ולשפר את הדיוק שלה בכל איטרציה.

**דוגמה פשוטה לתהליך**

נניח שיש לנו רשת נוירונים פשוטה שמנבאת את המחיר של דירה לפי שטחה.

- ב-Forward Pass נחשב תחילה את הניבוי (מחיר הדירה) על בסיס השטח הנתון.
- לאחר מכן נשווה את הניבוי למחיר האמיתי ונחשב את השגיאה.
- ב- Backward Pass נעדכן את המשקלים בהתאם לשגיאה שנמצאה, כך שבפעם הבאה הניבוי יהיה קרוב יותר למחיר האמיתי.

השילוב בין ה- Forward Pass וה- Backward Pass באלגוריתם החזרה, מאפשר לרשת להתאים את המשקלים שלה בצורה הדרגתית, וכך להשתפר באיתור הדפוסים בנתונים שהיא מקבלת.

## Hyper parameters

**היפר-פרמטרים (Hyperparameters)** הם פרמטרים שהוגדרו מראש על ידי המשתמש ואינם מתעדכנים באופן אוטומטי במהלך תהליך האימון של המודל. היפר-פרמטרים קובעים את המבנה והאופן שבו המודל מתאמן, ויש להם השפעה משמעותית על הביצועים והיעילות של הרשת הנוירונית.

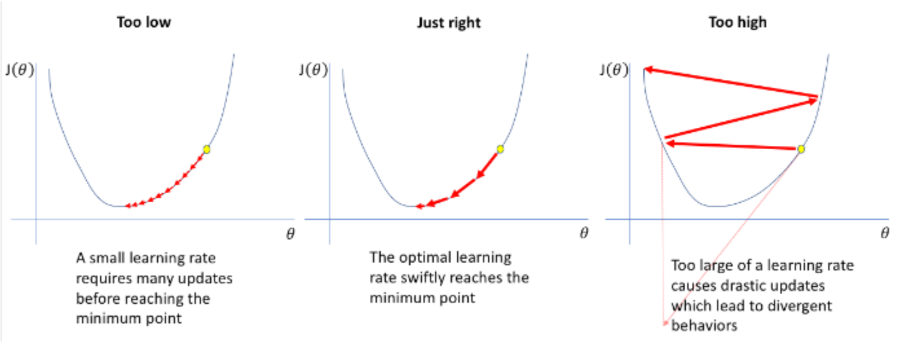
**סוגי היפר-פרמטרים בלמידה עמוקה:**

1. **היפר-פרמטרים של אדריכלות הרשת:**

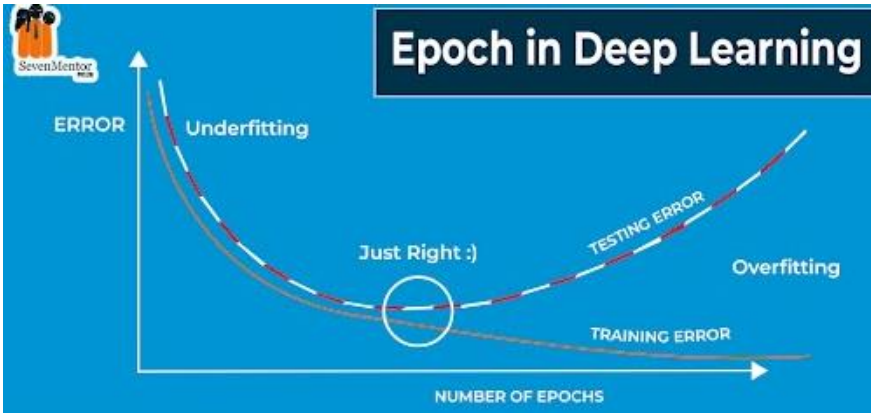
- **מספר השכבות החבויות**: מספר השכבות הפנימיות ברשת קובע את עומקה. עומק הרשת משפיע על המורכבות שלה ועל היכולת שלה ללמוד ייצוגים מורכבים.
- **מספר הנוירונים בכל שכבה**: מספר הנוירונים בשכבות החבויות קובע את רוחב הרשת. יותר נוירונים יכולים לאפשר למודל לקלוט וללמוד יותר מידע, אך עלולים גם לגרום ל overfitting.
- **סוג פונקציית האקטיביציה**: הפונקציה המיושמת בכל נוירון (כמו ReLU, סיגמואיד או Tanh) משפיעה על אופן העיבוד של האותות בכל שכבה. בחירת הפונקציה המתאימה יכולה לשפר את ביצועי המודל.

2. **היפר-פרמטרים של תהליך האימון:**

- **שיעור הלמידה (Learning Rate):** קובע את גודל הצעד שבו המשקלים מתעדכנים בכל איטרציה. ערך גבוה מדי יכול להוביל לאי-יציבות בלמידה, בעוד ערך נמוך מדי עלול להאט את תהליך הלמידה.




- **גודל (Mini-Batch Size):** גודל קבוצת הנתונים שבה משתמשים בכל איטרציה של האימון. גודל מיני-בץ' משפיע על מהירות האימון ועל הדיוק של עדכון המשקלים.
- **מספר האפוקים (Epochs):** מספר המחזורים שהמודל עובר על כל סט הנתונים במהלך האימון. מספר אפוקים גבוה יכול לשפר את הביצועים, אך עלול להוביל לעודף התאמה (overfitting) אם המודל עובר יותר מדי מחזורים.



3. **היפר-פרמטרים של אופטימיזציה:**

- **אלגוריתם האופטימיזציה:** בחירה של אלגוריתם האופטימיזציה כמו: Gradient Descent, Adam או RMSprop שמשפיע על אופן עדכון המשקלים במהלך האימון.
- **רגולריזציה:** היפר-פרמטרים כמו Dropout או L2 Regularization מונעים עודף התאמה בכך שהם מגבילים את המורכבות של המודל, מה שמוביל לשיפור בהכללה של המודל לנתונים חדשים.

4. **היפר-פרמטרים של עיבוד קדם (Preprocessing):**

- **שיעור Dropout:** קובע את אחוז הנוירונים שמתאפסים בכל שכבה במהלך האימון כדי למנוע עודף התאמה.
- **Data Augmentation:** טכניקות להרחבת מערך הנתונים על ידי יצירת וריאציות חדשות על הנתונים הקיימים, דבר המשפיע על מגוון ואיכות הנתונים שעליהם מתאמן המודל.

**חשיבות ההיפר-פרמטרים**

בחירה נכונה של היפר-פרמטרים יכולה לעשות את ההבדל בין מודל עם ביצועים טובים לבין מודל פחות מוצלח. על מנת למצוא את הערכים האופטימליים בדרך כלל משתמשים בטכניקות כמו **Grid Search** או **Random Search**, או באופטימיזציה מתקדמת יותר כמו **Bayesian Optimization**. הבחירה והכיוון של ההיפר-פרמטרים נעשים לעיתים קרובות על בסיס ניסוי וטעיה בהתאם לבעיית הלמידה המסוימת ולסוג הנתונים.

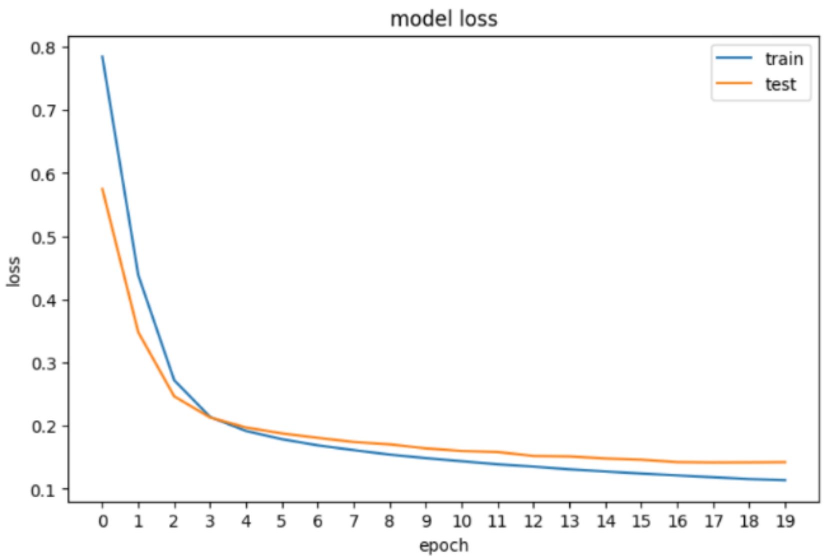
## פונקציית Loss:

**Loss** הוא מדד למידת השגיאה או הטעות של המודל בניבוי התוצאה. פונקציית ההפסד מודדת את ההבדל בין התוצאה שהמודל חזה לבין התוצאה האמיתית (התווית) בנתונים המוגדרים. המטרה המרכזית באימון מודל היא למזער את ערך ההפסד ככל שניתן, כך שהמודל ינבא בצורה מדויקת יותר.

**תפקידו של הLoss:**

בכל שלב באימון, לאחר ה- Forward Pass שבו המודל חוזה את התוצאה, מחשבים את ההפסד כדי להבין עד כמה הניבוי היה קרוב לתוצאה האמיתית:

- **שגיאה קטנה** משמעותה שהניבוי היה קרוב מאוד לתוצאה האמיתית.
- **שגיאה גדולה** מצביעה על כך שהמודל רחוק מניבוי נכון, ולכן עליו לבצע עדכונים משמעותיים במשקלים כדי להשתפר.




**סוגי פונקציות Loss**

ישנן פונקציות הפסד שונות שמתאימות למשימות שונות, והבחירה ביניהן תלויה בסוג הבעיה:

1. **הפסד ריבועי ממוצע (Mean Squared Error - MSE):**

- נפוצה בבעיות רגרסיה, מחשבת את הממוצע של ריבועי ההבדלים בין הניבוי לתוצאה האמיתית.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

כאשר יש לנו בסך הכל $N$ דגימות (כלומר $N$ זוגות של ערך אמיתי וניבוי).

כאשר עבור הדגימה ה-$i$: $y_i$ מייצג את הערך האמיתי ו- $\hat{y}_i$ מייצג את החיזוי.

2. **הפסד לוגיסטי (Log Loss):**

- בשימוש בבעיות סיווג בינאריות, ההפסד בוחן את הדיוק של ההסתברויות החזויות עבור כל אחת מהקטגוריות.

$$Log\ Loss = \frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log{(\hat{y}_i)} + (1 - y_i) \cdot \log{(1 - \hat{y}_i)}$$

כאשר יש לנו בסך הכל $N$ דגימות (כלומר $N$ זוגות של ערך אמיתי וניבוי).

כאשר עבור הדגימה ה-$i$: $y_i$ מייצג את הערך האמיתי ו- $\hat{y}_i$ מייצג את החיזוי.

3. **(Cross-Entropy Loss):**

- נפוצה בבעיות סיווג רב-קטגוריות. ההפסד מודד את מידת הדיוק של ההסתברויות החזויות עבור כל קטגוריה על בסיס ההסתברות האמיתית.

- בבעיות רב-קטגוריות משתמשים ב Categorical Cross-Entropy ובבעיות בינאריות משתמשים בבינארי.

**איך פונקציית ההפסד משפיעה על האימון?**

לאחר חישוב ההפסד מבצעים את תהליך ה- Backward Pass כדי לחשב את השיפוע של ההפסד ביחס לכל אחד מהמשקלים ברשת. השיפועים משמשים לעדכון המשקלים כדי למזער את ההפסד ולשפר את דיוק המודל. תהליך זה חוזר על עצמו באיטרציות רבות במהלך האימון, במטרה להקטין את ההפסד ולהגיע לביצועים מיטביים.

פונקציית ההפסד היא קריטית מכיוון שהיא הקובעת את האופן שבו המודל "לומד" ומתקן את עצמו. מודל שמתאמן היטב יהיה בעל ערך הפסד נמוך על הנתונים החדשים, מה שמצביע על כך שהוא מצליח ללמוד הכללות יעילות.

אנחנו בפרויקט שלנו משתמשים ב cross entropy loss:

## :Cross entropy loss

**Cross Entropy Loss** היא פונקציית הפסד המשמשת בעיקר בלמידה עמוקה וברשתות נוירונים, לצורך השוואת וקטורים ולמדידת האיבוד או השגיאה שבין התוצאה הצפויה לבין התוצאה שחוזתה על ידי המודל.

**מהו Cross Entropy?**

במהותו **Cross Entropy** (אנטרופיה צולבת) מודד את הפער שבין שתי התפלגויות הסתברות. בלמידה עמוקה נהוג להשתמש בו כאשר מבצעים סיווג קטגורי (כמו בבעיות סיווג תמונות)

Cross Entropy Loss מחשב את הפער בין הווקטור הצפוי (לדוגמה התווית האמיתית

או "One-Hot Encoding" במקרה של סיווג) לבין הווקטור הניבוי של המודל (הפלט מהשכבה האחרונה של הרשת).

**כיצד זה עובד?**

כאשר מדובר בסיווג עם מספר קטגוריות, הפלט של המודל הוא לרוב וקטור המייצג הסתברויות (למשל אחרי יישום שכבת סופטמקס), כאשר כל ערך בווקטור מייצג את ההסתברות שהנתון שייך לקטגוריה מסוימת.

נניח ש:

- $Y$ הוא הווקטור הצפוי (One-Hot Encoding) בו יש ערך אחד שהוא 1 והשאר 0.
- $\hat{Y}$ הוא הווקטור שחוזה המודל, ובו ההסתברויות לכל קטגוריה.

האנטרופיה הצולבת מוגדרת כ:

$$Cross\ Entropy\ Loss = -\sum_i y_i \cdot \log(\hat{y}_i)$$

כאשר:

- $y_i$ הערך הצפוי לקטגוריה i.

- $\hat{y}_i$ הוא ההסתברות שחוזה המודל לקטגוריה i.

**למה זה חשוב?**

המשמעות של Cross Entropy Loss היא מדידה של כמה ההסתברויות שחוזה המודל קרובות לתוצאה האמיתית. אם המודל חוזה הסתברות גבוהה עבור הקטגוריה הנכונה, הערך של Cross Entropy Loss יהיה נמוך. אם הוא חוזה הסתברות נמוכה עבור הקטגוריה הנכונה, האיבוד יהיה גבוה יותר, וכך המודל יקבל חיווי לשפר את הפרמטרים שלו בהתאם.

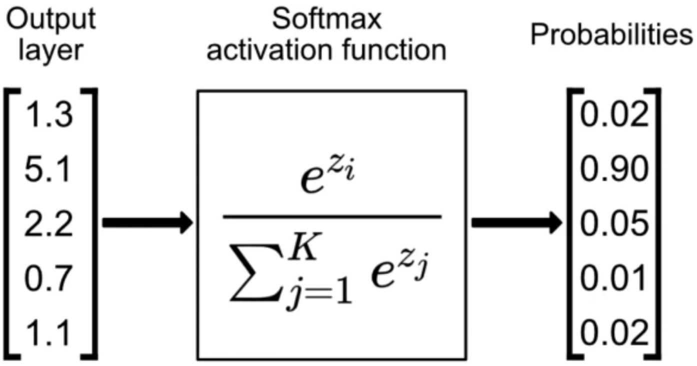
**השוואת וקטורים**

היות ושכבת Cross Entropy Loss מתבצעת על ידי השוואה בין הווקטור שחוזה המודל לווקטור התווית, היא חיונית עבור בעיות סיווג. כך, המודל לומד להתאים את ההסתברויות שלו כך שיקטין את האיבוד, ובכך ישפר את יכולתו לנבא את הקטגוריה הנכונה עבור דוגמאות חדשות.

שכבת Cross Entropy Loss מאוד נפוצה בגלל התאמתה לבעיות סיווג, והיא משמשת בלמידה עמוקה כדי להנחות את תהליך הלמידה לכיוון הנכון.

## :Soft Max

שכבת **Softmax** היא שכבת פלט נפוצה ברשתות נוירונים בלמידה עמוקה, במיוחד כאשר מדובר במשימות סיווג (classification). היא ממירה וקטור של ערכים כלשהם לווקטור של ערכים המייצגים הסתברויות, כך שסכום כל ההסתברויות יהיה שווה ל-1.




**איך פועלת שכבת Softmax ?**

נניח שיש לנו וקטור פלט מרשת נוירונים, שמכיל ערכים ממשיים (שליליים או חיוביים).

שכבת ה-Softmax ממירה את הערכים הללו להסתברויות באמצעות הנוסחה הבאה:

$$P(y_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

כאשר:

- $z_i$ הוא הערך המקורי של הנוירון i בוקטור הפלט.

- $e^{z_i}$ הוא האקספוננט של הערך, מה שמבטיח שהערך תמיד יהיה חיובי.
- $\sum_j e^{z_j}$ הוא סכום האקספוננטים של כל הנוירונים, מה שמבטיח שסכום הערכים של הוקטור יהיה 1.

**השימוש ב- Softmax בלמידה עמוקה**

שכבת ה- Softmax היא מרכיב קריטי במשימות סיווג. היא עוזרת להפוך את התוצאות של רשת הנוירונים להסתברויות, כך שניתן להבין את התוצאה במונחים של "מהי ההסתברות שכל קטגוריה היא הנכונה?" לדוגמה, במשימה של סיווג תמונה לקטגוריות שונות, Softmax יכולה לקחת את הערכים שהתקבלו בפלט ולהמיר אותם להסתברויות של השתייכות כל תמונה לכל אחת מהקטגוריות האפשריות.

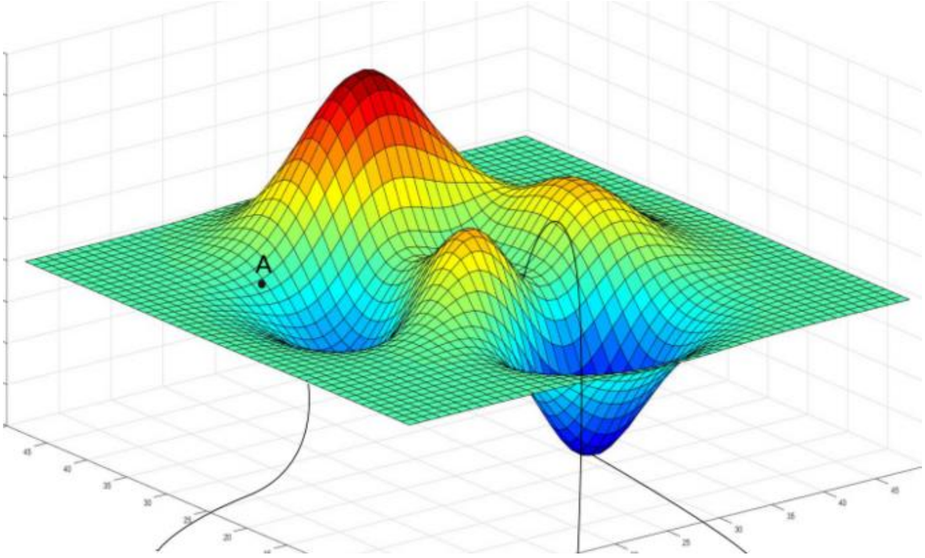
**Softmax והשוואת וקטורים**

כאשר מדובר בהשוואת וקטורים, למשל בהשוואת פלטים של רשתות נוירונים שונות, השימוש בSoftmax- יכול לסייע בהבנת הקשר בין הערכים. היות והוא ממיר כל וקטור להסתברויות, ניתן להסתכל על הפלט כמעין "הערכה" יחסית של כל קטגוריה.
למשל, אם וקטור אחד גבוה במיוחד באלמנט מסוים לאחר שכבת ה- Softmax נוכל להבין שיש הסתברות גבוהה שקטגוריה זו היא המתאימה ביותר, מה שמסייע לנו להשוות בין וקטורים ולהבין את הקשר שלהם מבחינת סיווג.

לסיכום, שכבת Softmax היא כלי חיוני לרשתות נוירונים כאשר נדרש פלט שמייצג הסתברויות, והיא משמשת בעיקר במשימות סיווג כדי לספק תובנה לגבי הסיכוי של כל קטגוריה להיות הנכונה ביותר.

## Gradient Descent:

**Gradient Descent** הוא אלגוריתם אופטימיזציה נפוץ בלמידת מכונה, המשמש להתאמת פרמטרים של מודל (כמו משקלים ברשת נוירונים) במטרה למזער את פונקציית ההפסד. המטרה היא למצוא את נקודת המינימום של פונקציית ההפסד כך שהמודל ינבא בצורה מדויקת יותר.




Gradient Descent מבצע את הפעולות הבאות:

1. **חישוב גרדיאנט (Gradient):** מחשבים את הנגזרת של פונקציית ההפסד ביחס לכל אחד מהמשקלים. הנגזרת הזו, או "הגרדיאנט," מציינת את כיוון השיפוע החד ביותר של פונקציית ההפסד.
2. **עדכון משקלים** :המשקלים מתעדכנים בכיוון הפוך לגרדיאנט כדי להקטין את הערך של פונקציית ההפסד. עדכון זה מבוצע על ידי הנוסחה:

$$w = w - \eta \cdot \nabla L(w)$$

- w מייצג את המשקלים.
- $\eta$ (שיעור הלמידה) קובע את גודל הצעד.
- $\nabla L(w)$ הוא הגרדיאנט של פונקציית ההפסד ביחס למשקלים.

תהליך זה חוזר על עצמו עד שהמודל מגיע לנקודת המינימום של פונקציית ההפסד או שהשיפור בביצועים נהיה קטן מאוד.

**וריאציות של Gradient Descent:**

**1) Batch Gradient Descent (BGD)**

בגרסה הזו של Gradient Descent החישוב של הגרדיאנט מתבצע על כל מערך הנתונים בבת אחת.

- **יתרונות**: יציב ומדויק, מאחר שהחישוב נעשה על כל מערך הנתונים.
- **חסרונות**: מאוד איטי ומשאבי החישוב גבוהים, בעיקר כאשר מערך הנתונים גדול.

**2) Stochastic Gradient Descent (SGD)**

ב- Stochastic Gradient Descent עדכון המשקלים מתבצע על בסיס דגימה אחת (או נקודת נתונים אחת) בכל פעם, ולא על כל מערך הנתונים.

- **יתרונות**: מהיר ויכול להגיע לנקודת מינימום מהר יותר. הוא גם יכול לברוח ממינימום מקומי ולמצוא פתרונות גלובליים.
- **חסרונות**: מאוד רועש ואינו יציב, כך שהתוצאה עשויה להתנדנד סביב המינימום ולא להגיע אליו בדיוק.

**3) Mini-Batch Gradient Descent**

ב- Mini-Batch Gradient Descent, מערך הנתונים מחולק למיני-בצ'ים קטנים, ועדכון המשקלים מתבצע עבור כל מיני-בץ' בנפרד.

- **יתרונות**: מאפשר איזון בין מהירות ודיוק. הוא יציב יותר מ- SGD ומהיר יותר מ- BGD.
- **חסרונות**: הבחירה של גודל המיני-בץ' משפיעה על הביצועים. גודל מיני-בץ' קטן יכול להוביל לתוצאה רועשת, בעוד גודל גדול עלול להאט את הלמידה.

**וריאציות מתקדמות של Gradient Descent**

ישנן מספר וריאציות מתקדמות המשלבות אלמנטים נוספים לשיפור הביצועים:

1. **Momentum:** מוסיף מרכיב של תאוצה, כלומר, מתחשב גם בכיוון הקודם של הגרדיאנט כדי להאיץ את ההתכנסות.

2. **Adam (Adaptive Moment Estimation):** משלב בין Momentum לבין Adaptive Learning Rate.
Adam מתחשב הן בערך הממוצע של הגרדיאנטים והן בערך הממוצע המרובע שלהם, ומאפשר עדכון משקלים חכם יותר. הוא נפוץ מאוד ונחשב ליעיל מאוד בבעיות רבות.

3. **RMS prop**: משתמש בקצב למידה שמתאים את עצמו על פי הגרדיאנטים המקומיים. האלגוריתם מתאים את עצמו באופן דינמי לאורך האימון.

הבחירה באלגוריתם Gradient Descent ובגרסה שלו תלויה בסוג הבעיה, בגודל מערך הנתונים ובמשאבי החישוב הזמינים. Gradient Descent הוא הבסיס לאימון מודלים, והוריאציות שלו מאפשרות ביצועים טובים יותר באימון מודלים מורכבים בלמידה עמוקה.

## :Optimizer

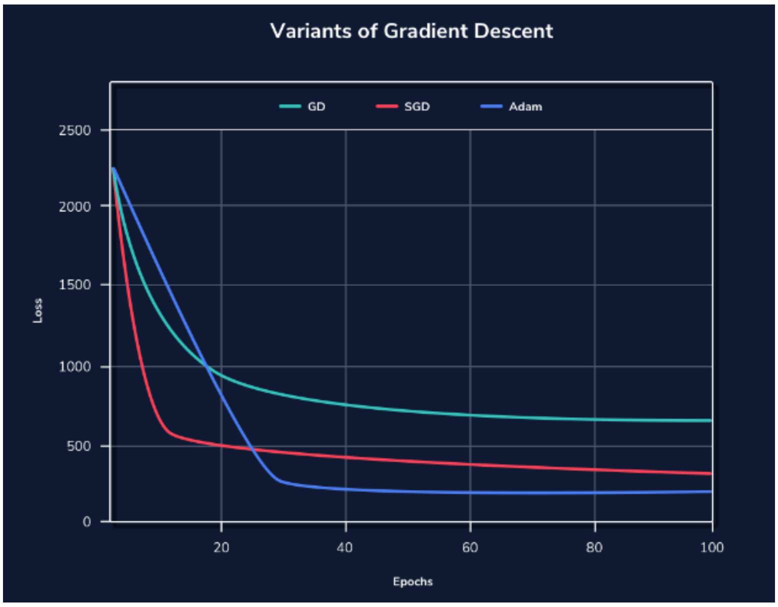
**Optimizer** הוא אלגוריתם שתפקידו לעדכן את המשקלים והפרמטרים של המודל כדי להקטין את פונקציית ההפסד, וכך לשפר את דיוק המודל. במהלך האימון, האופטימיזר משתמש במידע על הגרדיאנטים (שיפועי ההפסד) כדי לקבוע באיזו מידה ובאיזה כיוון יש לעדכן את המשקלים של הרשת.

**תפקידו של האופטימיזר**

האופטימיזר קובע את האופן שבו המודל מתקדם לקראת המינימום של פונקציית ההפסד:

- הוא קובע את כיוון השינוי של המשקלים, על בסיס השיפועים של פונקציית ההפסד.
- הוא קובע את גודל הצעד (לפי שיעור הלמידה) בכל איטרציה, ומוודא שהמודל מתקדם בצורה יציבה ויעילה.

אופטימיזרים שונים מתאימים לבעיות שונות, שכן הם שונים ביכולת שלהם להתאים את שיעור הלמידה, במידת היציבות שלהם, ובמהירות שבה הם מתכנסים לנקודת המינימום.



אנחנו משתמשים בפרויקט שלנו באופטימייזר שנרא Adam.

## Adam (Adaptive Moment Estimation)

Adam הוא אחד האופטימיזרים הנפוצים והמתקדמים ביותר בלמידה עמוקה. זהו אלגוריתם אופטימיזציה המשמש לעדכון המשקלים ברשת הנוירונים במהלך האימון.

הוא משלב בין שני רעיונות קודמים:

1. **Momentum:** רעיון שמוסיף "תאוצה" לתהליך, כך שהעדכונים מתחשבים גם בכיוון של העדכונים הקודמים, מה שעוזר להתגבר על תנודות ולהאיץ את הלמידה.
2. **Adaptive Learning Rate:** התאמה דינמית של קצב הלמידה, כך שהוא משתנה באופן אוטומטי בהתאם לשונות של הגרדיאנט.

**איך Adam עובד?**

Adam משתמש בשני מומנטים (Moments):

- **המומנט הראשון:** מחשב את ממוצע הגרדיאנטים הנוכחיים, מה שנותן תמונה על כיוון השינוי של פונקציית ההפסד.

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

- **המומנט השני:** מחשב את ממוצע הגרדיאנטים המרובעים, שמסייע להתמודד עם שינויים בתנודות בקצב הלמידה.

המשקלים מעודכנים על פי הנוסחה:

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

3. **תיקון bias:**
בגלל ש $m_t$ ו $v_t$ מאותחלים כאפס, Adam עושה bias correction להתחשב באתחול אפס.

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \widehat{v}_t = \frac{mv_t}{1 - \beta_2^t}$$

4. **עדכון משקלים:**
Adam מעדכן את המשקלים באמצעות המומנט הראשון והשני המסתגלים

$$w = w - \eta \cdot \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon}$$

**יתרונות Adam**

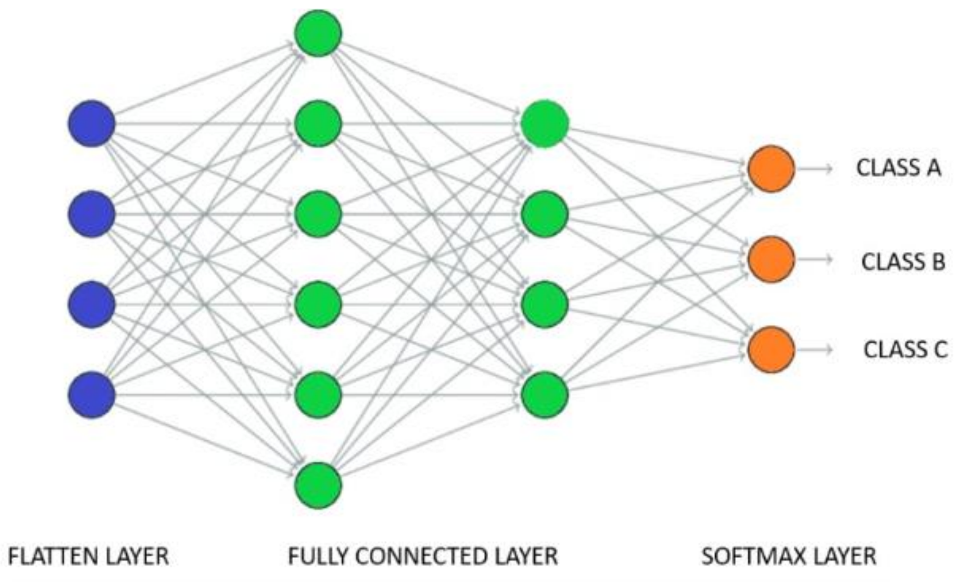
- **יציבות ודיוק:** Adam נחשב ליציב במיוחד ונוטה להתכנס במהירות.
- **התאמה עצמית:** בזכות ההתאמה האוטומטית של קצב הלמידה, Adam פועל היטב בסביבות מורכבות עם משתנים רבים.
- **מתאים לרשתות גדולות:** Adam מתמודד היטב עם נתונים גדולים ועם רשתות עמוקות.

האופטימיזר Adam מתאים למגוון רחב של בעיות למידה עמוקה, והוא פופולרי ביותר בגלל היכולת שלו לשלב בין מהירות ויציבות, וכן בגלל היכולת שלו להתמודד עם גרדיאנטים רעשניים ועם נתונים לא יציבים.

## Neural Network layers

### שכבת fully connected:

**שכבה מלאה מחוברת** (Fully Connected Layer או Dense Layer) היא שכבה נפוצה ברשתות נוירונים מלאכותיות, במיוחד בשלבים הסופיים של רשתות, כמו ב- CNN לאחר שכבות הקונבולוציה והקיטום כל נוירון בשכבה זו מחובר לכל נוירון בשכבה הקודמת, ולכן השכבה מכונה "מלאה מחוברת."




**מבנה ותפקוד של שכבה מלאה מחוברת**

בשכבה מלאה מחוברת:

1. **קשרים בין נוירונים**:

- כל נוירון מחובר לכל נוירון בשכבה הקודמת באמצעות משקל ייחודי, וכל משקל מגדיר את עוצמת הקשר בין שני נוירונים.
- הערך של כל נוירון מחושב על ידי חיבור הערכים של הנוירונים הקודמים כשהם מוכפלים במשקלים המתאימים, ולאחר מכן הוספת הטיית Bias והעברת התוצאה דרך פונקציית הפעלה (כמו ReLU או Sigmoid)

2. **חישוב היציאה (Output)**:

- התוצאה עבור כל נוירון בשכבה היא

$$f\left(\sum_i x_i \cdot w_i + b\right) = y$$

- $x_i$ הערך של הנוירון הקודם.
- $w_i$ המשקל המחובר לערך זה.
- $b$ ההטיה.

- $f$ פונקציית הפעלה – activation.

3. **פלט השכבה**:

- הפלטים מהנוירונים מתקבלים כווקטור אחד. אם יש n נוירונים בשכבה, אז הפלט יהיה בגודל של n.

**תפקיד שכבה מלאה מחוברת ברשת נוירונים**

- **אינטגרציה של מאפיינים**: בשלב זה, השכבה משקללת את כל המאפיינים שהתקבלו מהשכבות הקודמות, כדי לזהות קשרים לא-לינאריים ביניהם ולהסיק תובנות סופיות.
- **סיווג או רגרסיה**: ברוב המקרים, שכבה מלאה מחוברת מספקת את הפלט הסופי של המודל. לדוגמה: ברשת סיווג, השכבה האחרונה תהיה מלאה מחוברת עם מספר הנוירונים המתאים למספר הקטגוריות, כשהיא מספקת הסתברות לכל קטגוריה.

**דוגמה לשימוש**

ב CNN-לסיווג תמונות:

- לאחר שהשכבות הקונבולוציות והקיטום חילצו את המאפיינים מהתמונה, הפלט מועבר לשכבה אחת או יותר של Fully Connected.
- השכבה המלאה משקללת את כל המידע ומפיקה תוצאה המייצגת את הקטגוריה הסופית (כגון זיהוי אובייקט כמו "חתול" או "כלב").

**יתרונות וחסרונות**

**יתרונות**:

- **פשטות וגמישות** : השכבה קלה ליישום ויכולה ללמוד מגוון רחב של קשרים בין המאפיינים שהופקו בשלבים קודמים.
- **יעילות בסיווג**: השכבה מצוינת לסיווג בזכות יכולתה לאגד את כל המאפיינים לכדי תחזית סופית אחת.

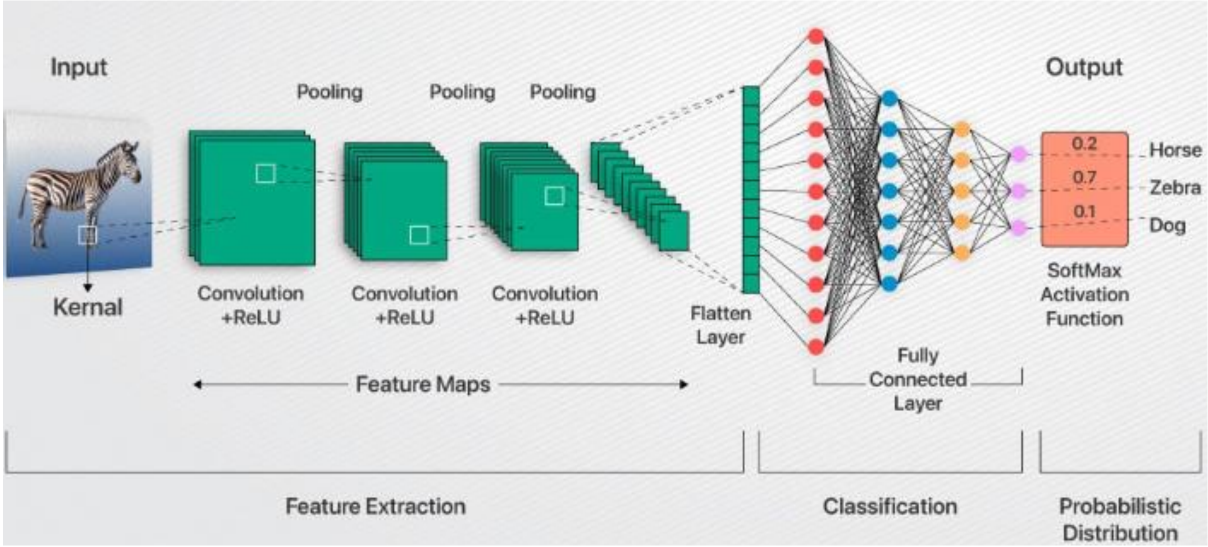
**חסרונות**:

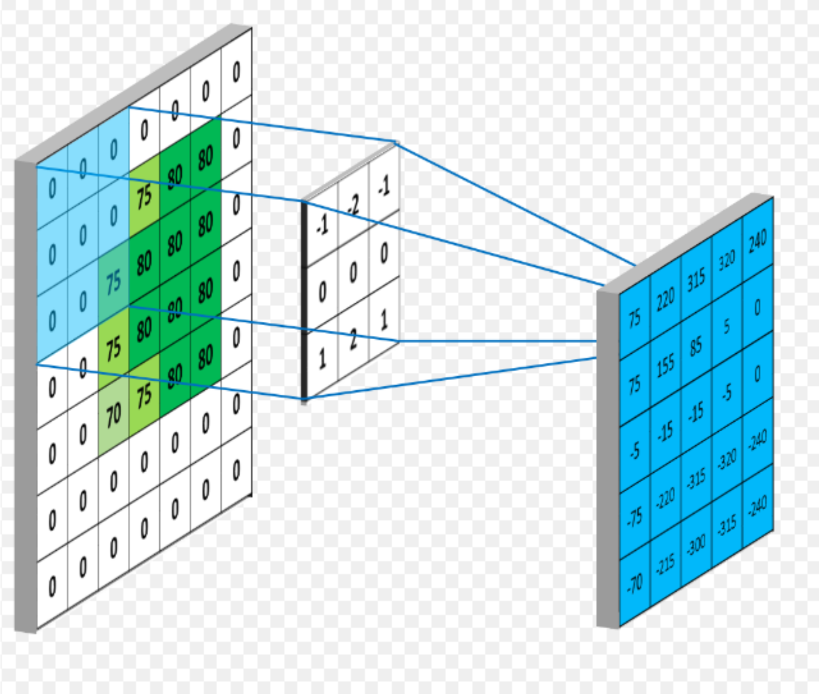
- **מספר גבוה של פרמטרים**: ככל שיש יותר נוירונים, כך יש יותר משקלים שיש לעדכן באימון, מה שעלול להוביל לעומס חישובי רב.
- **Overfitting**: השכבה עלולה לגרום להתאמת יתר, במיוחד כאשר יש יותר מדי נוירונים לעומת מספר הדוגמאות באימון.

לסיכום, שכבות מלאות מחוברות חשובות במיוחד במודל הסופי של רשת נוירונים, כשהן מוודאות שכל המידע שנלמד מהמאפיינים משוקלל ומניב את התחזית המדויקת ביותר.

## שכבת Convolution:

שכבת **Convolution Layer** (שכבת קונבולוציה) היא רכיב מרכזי ברשתות נוירונים קונבולוציוניות (Convolutional Neural Networks - CNNs), אשר מיועדות במיוחד לעיבוד תמונה וזיהוי תבניות. השכבה מבצעת עיבוד של תמונה או מידע ויזואלי אחר על ידי יישום מסננים ( kernels אוfilters ) החולפים על פני קטעים קטנים של התמונה, ועוזרת לרשת ללמוד מאפיינים חזותיים כמו קצוות, צורות ותבניות מורכבות יותר.



**איך זה עובד?**

שכבת הקונבולוציה פועלת על ידי יישום מסנן בגודל קבוע (למשל, 3x3 פיקסלים ) על פני קטעים של התמונה. המסנן מחליק על פני התמונה בצורה של רשת קבועה( כמו רשת של 5x5 , 3x3כדוגמה) ובכל פעם שהיא מתמקמת על קטע מסוים, הוא מבצע חישוב על ידי הכפלת הפיקסלים באזור זה במשקלים של המסנן וסכימת התוצאות. תוצאה זו נקראת "ערך קונבולוציה" והיא נשמרת במטריצה חדשה שנקראת "מפת מאפיינים " (feature map)

### תהליך החישוב

בכל צעד המסנן מייצר ערך יחיד על ידי הכפלת ערכי הפיקסלים בתמונה בערכי המשקלים של המסנן וסכימת התוצאות. לאחר מכן, המסנן עובר לאזור הבא בתמונה, ושוב מבצע את החישוב. חזרה על התהליך לאורך ולרוחב התמונה מייצרת מפת מאפיינים של כל האזורים שהתמקדו בהם.

### היפר-פרמטרים בשכבת הקונבולוציה

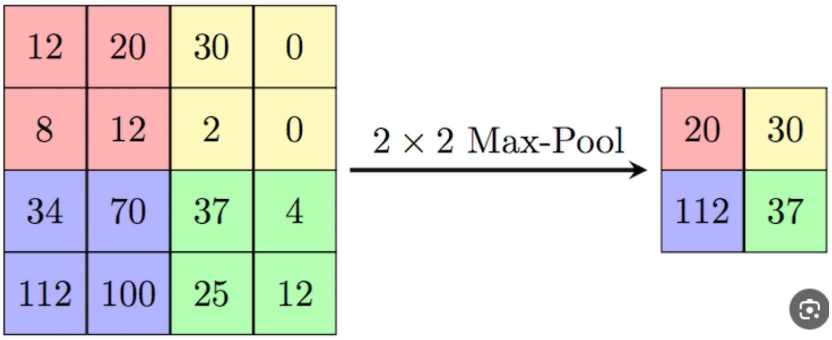
1. **גודל המסנן (Kernel Size):** גודל המסנן (למשל 3x3, 5x5) קובע את גודל השטח בו מתמקדים בכל פעם.
2. **צעד (Stride):** מגדיר בכמה פיקסלים המסנן מתקדם בכל פעם. צעד גדול יותר יגרום למפת מאפיינים קטנה יותר.
3. **ריפוד (Padding)**: מאפשר הוספת פיקסלים לתמונה המקורית על מנת לשמור על גודלה לאחר הקונבולוציה.

### למה זה חשוב?

שכבת הקונבולוציה יעילה בזיהוי מאפיינים מקומיים כגון קצוות וצורות בסיסיות, ובשכבות עמוקות יותר של הרשת, היא מאפשרת זיהוי תבניות מורכבות יותר כמו חפצים שלמים. שילוב שכבות קונבולוציה עם שכבות pooling (שכבות איגום) ושכבות Fully Connected מאפשר לרשת ללמוד ולהכליל תבניות ברמות שונות, מה שמוביל לביצועים מרשימים במשימות כמו זיהוי אובייקטים, זיהוי פרצופים, סיווג תמונות ועוד.

## שכבת Pooling:

שכבת **Pooling** היא שכבה ברשתות נוירונים, בעיקר ברשתות נוירונים קונבולוציוניות (CNN), שנועדה להפחית את גודל המידע שמועבר לשכבות הבאות מבלי לאבד מידע משמעותי. המטרה העיקרית של שכבת ה-Pooling היא לצמצם את המורכבות החישובית של הרשת, למנוע "overfitting" ולאפשר לרשת לזהות מאפיינים עיקריים של התמונה (או הנתונים) בצורה יותר עמידה לשינויים כמו מיקום, קנה מידה וסיבוב.




### איך זה עובד?

בתהליך ה-Pooling לוקחים חלון קטן (למשל גודל 2x2 או 3x3) ומעבירים אותו על פני המפה (feature map) של המאפיינים שהתקבלה מהשכבות הקונבולוציוניות. עבור כל חלון כזה מופעלת פעולה מסוימת, לרוב אחת מהבאות:

1. **Max Pooling:** עבור כל חלון, בוחרים את הערך המקסימלי. הרעיון כאן הוא לשמור את הערך החזק ביותר, שמייצג את התכונה הדומיננטית ביותר בחלון, ולמחוק את שאר הפרטים.

לדוגמה, אם בחלון 2x2 הערכים הם:

$$\begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$$

אחרי Max Pooling הערך שנשאר הוא 4, שהוא הערך המקסימלי.

2. **Average Pooling:** מחשבים את הממוצע של כל הערכים בתוך החלון ומשמרים אותו כערך מייצג.
לדוגמה, עבור אותו חלון 2x2:

$$\frac{4+3+2+1}{4} = 2.5$$

כלומר, הערך שייצג את החלון הזה הוא 2.5.

**למה זה חשוב?**

- **הקטנת ממדי הנתונים**: השכבה מפחיתה את ממדי הנתונים (למשל, התמונה) על ידי סיכום המידע בכל חלון. הדבר מאפשר לשכבות הבאות לעבד פחות נתונים ובכך להאיץ את החישובים.
- **שיפור עמידות לתזוזות**: מכיוון שהתהליך בוחר ערכים מקסימליים או ממוצעים מחלונות קטנים, הוא עוזר לרשת להיות עמידה יותר לשינויים קטנים במיקום המאפיינים בתמונה (כמו הזזה או סיבוב של אובייקט בתמונה).
- **מניעת Overfitting:** צמצום המורכבות של הרשת ועיבוד רק של מידע קריטי יותר מאפשר למנוע מצבים שבהם הרשת "לומדת" יותר מדי פרטים, שיכולים להוביל ליכולת ניבוי פחות טובה עבור נתונים חדשים.

**סיכום**

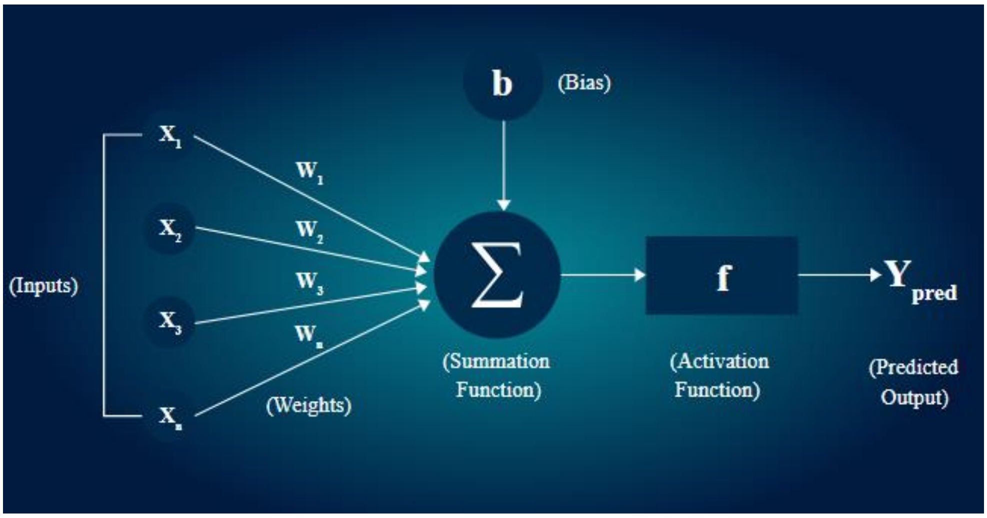
שכבת ה- Pooling היא מרכיב מרכזי ברשתות נוירונים קונבולוציוניות, והיא עוזרת לצמצם את המידע שעובר לשכבות הבאות, תוך שמירה על המאפיינים החשובים.

## שכבות Activation:

Activation Function  היא רכיב מרכזי ברשתות נוירונים המשמש לקביעת הפלט של כל נוירון. היא מבצעת עיבוד לא-ליניארי על הקלט שמגיע אליה, ומעבירה רק ערכים מסוימים קדימה לשכבה הבאה. בזכות הלא-ליניאריות שלה, פונקציית הפעלה מאפשרת לרשתות נוירונים ללמוד ולייצג קשרים מורכבים יותר.

**למה צריך פונקציות הפעלה?**

במקרה של רשת נוירונים ללא פונקציית הפעלה (בה כל שכבה היא ליניארית), כל השכבות המלאות היו מסתכמות לחישוב ליניארי אחד, וזה היה מגביל את יכולת הרשת לפתור בעיות מורכבות. פונקציות הפעלה מוסיפות רמה של מורכבות המאפשרת לרשת ללמוד קשרים לא-ליניאריים ולהיות מסוגלת להתמודד עם בעיות מורכבות יותר כמו סיווג תמונות, עיבוד שפה טבעית ועוד.




**דוגמאות לפונקציות הפעלה נפוצות**

1. **ReLU (Rectified Linear Unit):**

שכבת (Rectified Linear Unit) **ReLU** היא אחת השכבות הנפוצות ביותר ברשתות נוירונים, ותפקידה הוא להכניס **אי-ליניאריות** למערכת. היא משמשת כשכבת **הפעלה** (Activation Layer) אשר באה אחרי שכבות קונבולוציה או שכבות ליניאריות (Fully Connected Layers), כדי לאפשר לרשת ללמוד יחסים מורכבים יותר.

(אובייקט הRelu במודל מפעיל את פונקציית הRelu על כל אלמנט בטנסור הכניסה, כך שממדי הטנסור בכניסה ובמוצא הינם זהים.)

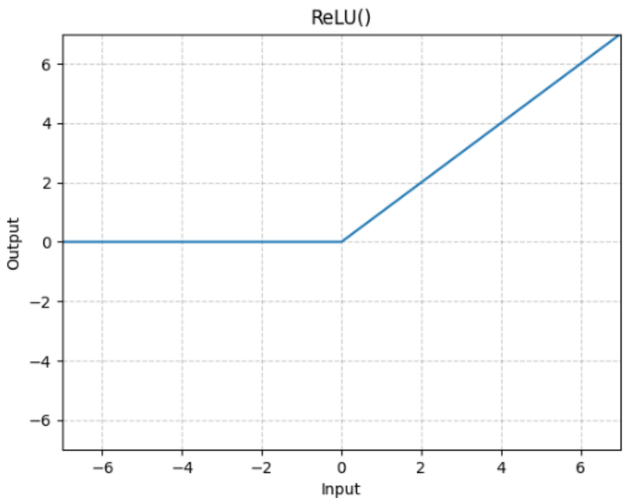
(מעצם הגדרתה של פונקציית הRelu ניתן לראות שאין בה משקלים או קבועים שהמודל יצטרך ללמוד\לזכור).

הפונקציה מוגדרת כ: $f(x) = \max\,(0, x)$

**מאפיינים של פונקציית ReLU :**

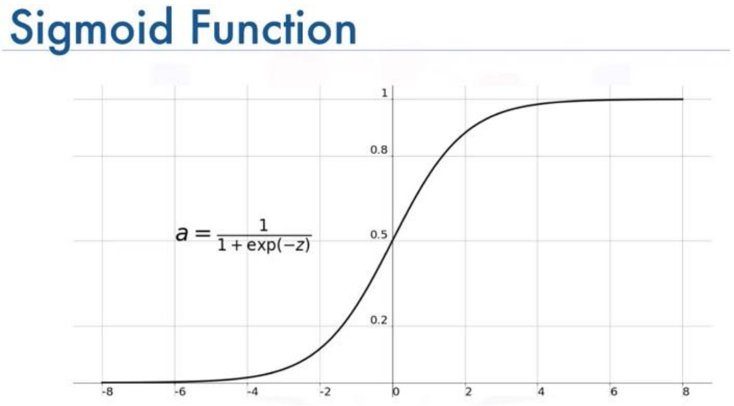
- ReLU הוא פונקציה פשוטה שמחזירה את x אם הוא חיובי, ו-0 אם הוא שלילי.
- יתרונות: מהירה לחישוב, פותרת את בעיית **ההיעלמות של הגרדיאנט**

  (Vanishing Gradient) שמאפיינת פונקציות אחרות.
- חסרונות: עלולה לגרום לבעיה שנקראת **נוירונים מתים** אם ערכים שליליים נתקעים ב-0 לאורך זמן.
- מעצם הגדרתה, פונקציית ה- ReLU פועלת על כל ערך בקלט באופן עצמאי, כך שמספר הערכים בקלט ובפלט נותר זהה (כלומר עבור טנסור, הפונקציה פועלת על כל אלמנט בנפרד, כך שממדי הטנסור בקלט ובפלט נשארים זהים).



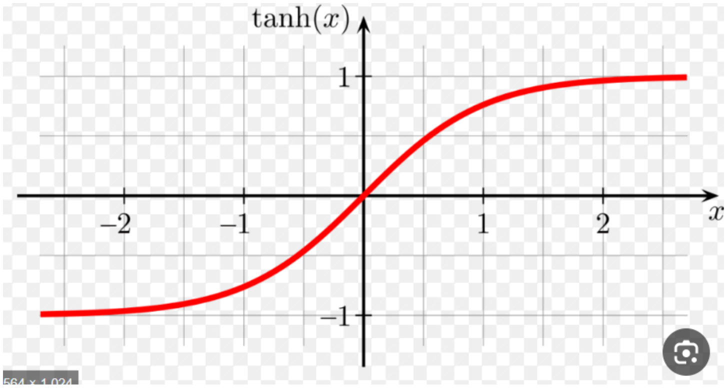

2. **Sigmoid:**

- $f(x) = \frac{1}{1+e^{-x}}$
- Sigmoid ממפה ערכים לטווח בין 0 ל1, ומתאימה במיוחד למקרים שבהם הפלט נדרש לייצג הסתברויות.
- יתרונות: מתאימה לרשתות עם פלט בינארי.
- חסרונות: גורמת להיעלמות גרדיאנט עבור ערכים מאוד גבוהים או נמוכים, דבר שמקשה על הרשת ללמוד.




3. **Tanh (Hyperbolic Tangent):**

- $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
- Tanh ממפה ערכים לטווח בין 1- ל1, ומהווה שיפור ל- Sigmoid בכך שהיא כוללת ערכים שליליים, מה שמרכז את הנתונים סביב אפס.
- יתרונות: מקטינה את הסיכון לגרדיאנטים שנעלמים, אם כי עדיין מושפעת מהבעיה.
- חסרונות: עדיין עלולה לגרום להיעלמות גרדיאנטים עבור ערכים קיצוניים.



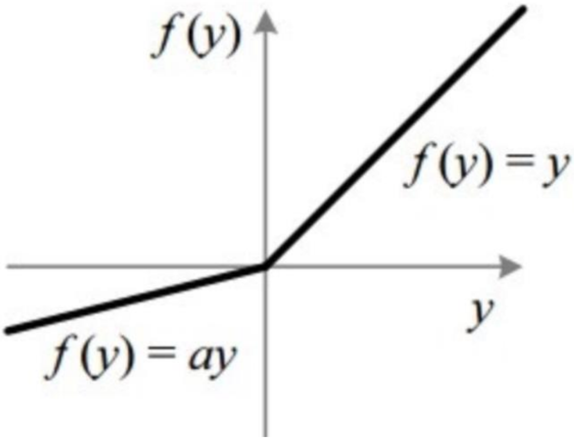

4. **Leaky ReLU:**

- $if\ x > 0 \rightarrow f(x) = x, if\ x < 0 \rightarrow f(x) = ax$
- זוהי גרסה של ReLU שבה ערכים שליליים אינם מאופסים לגמרי אלא מופחתים לפי מקדם α קטן כמו 0.01.
- יתרונות: פותר את בעיית הנוירונים המתים של ReLU על ידי כך שמאפשר לערכים שליליים לעבור, אם כי בצורה מוחלשת.

**איך בוחרים פונקציית הפעלה?**

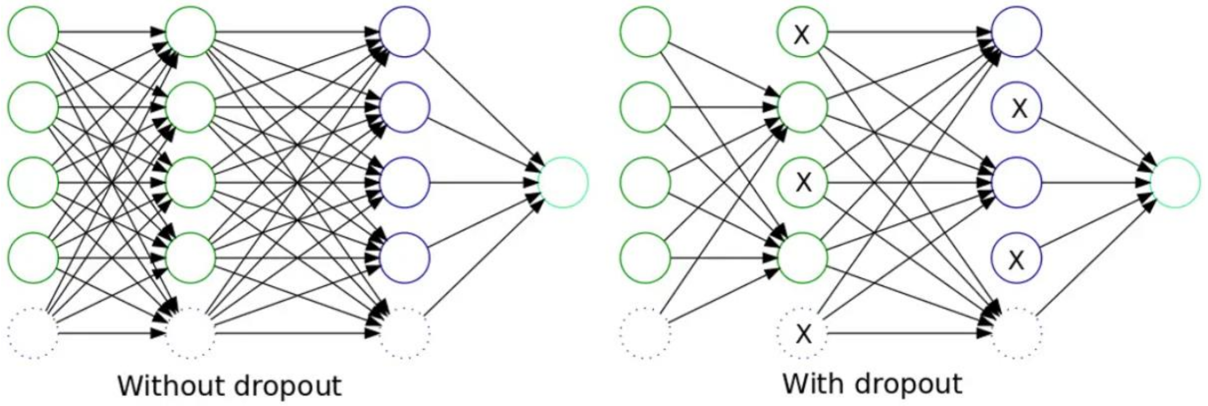
- עבור שכבות נסתרות **ReLU** וגרסאותיה (כגון Leaky ReLU) הן הבחירה הפופולרית בזכות פשטותן ויעילותן.
- עבור שכבות פלט בסיווג בינארי **Sigmoid** מתאימה.
- עבור שכבות פלט בסיווג רב-קטגוריות **Softmax** היא הבחירה המועדפת.
- **Tanh** משמשת לעיתים ברשתות שחייבות לעבוד עם נתונים המפוזרים גם במימד השלילי וגם במימד החיובי.

פונקציות הפעלה מוסיפות את האלמנט הלא-ליניארי ברשת, מה שמאפשר למודל ללמוד ממגוון רחב של תבניות ולקבל החלטות סופיות על בסיס קשרים מורכבים בין הנתונים.



## שכבת Dropout:

שכבת **Dropout** היא שכבה ברשתות נוירונים מלאכותיות שנועדה למנוע בעיית **overfitting** (התאמה יתרה), שהיא מצב שבו הרשת מתאימה את עצמה יותר מדי לנתוני האימון ולכן מתקשה להתמודד עם נתונים חדשים. שכבת ה-Dropout פועלת על ידי "כיבוי" אקראי של חלק מהנוירונים בשכבה מסוימת במהלך האימון, כלומר לא מאפשרת להם להשתתף בתהליך החישוב באותו מעבר.




**איך זה עובד?**

בכל פעם שמעבירים דוגמה של נתונים קדימה ברשת במהלך האימון:

1. שכבת ה-Dropout בוחרת באופן אקראי אחוז מסוים מהנוירונים בשכבה נתונה (למשל 20% או 50%) ומכבה אותם (כלומר, הערכים שלהם הופכים ל-0).
2. הנוירונים שכובו לא משפיעים על החישובים באותו מעבר קדימה.
3. בתהליך הבא (בכל דוגמה חדשה באימון), נבחר סט אקראי חדש של נוירונים שיכובה, כך שבכל פעם "רשת שונה" מעט מתאמנת.

לדוגמה, נניח שיש לך שכבה עם 10 נוירונים, ובשכבת Dropout מוגדרת השבתה של 50%. בכל מעבר, הרשת תבחר באופן אקראי 5 נוירונים ותכבה אותם, ותבצע את החישוב על בסיס 5 הנותרים.

**למה זה חשוב?**

1. **מניעת overfitting:** Dropout מאלצת את הרשת לא להיות תלויה מדי בנוירונים ספציפיים, כי בכל פעם קבוצה אחרת של נוירונים משותקת. בכך היא מאלצת את הרשת ללמוד ייצוגים יותר כלליים של המידע ולא להסתמך על חלקים מסוימים בלבד של הרשת. זה עוזר לרשת לבצע הכללה טובה יותר על נתונים חדשים, ולא להתאים את עצמה יותר מדי לנתוני האימון.
2. **שיפור עמידות**: על ידי "כיבוי" אקראי של נוירונים, הרשת נעשית עמידה יותר להפרעות קטנות בנתונים, מכיוון שהיא לומדת שלא להסתמך על מידע ממקור ספציפי אלא על מאפיינים רחבים יותר של הנתונים.
3. **אימון מספר רב של רשתות במקביל**: Dropout יוצר אפקט דומה לאימון מספר רב של רשתות קטנות בתוך רשת אחת גדולה, ובסוף שלב האימון מתקבלת רשת אחת ממוצעת מכל הרשתות "הקטנות" שאומנו במקביל.

**יישום בזמן בדיקה (Inference)**

חשוב לציין שבזמן הבדיקה (אחרי סיום האימון), שכבת ה Dropout-לא מכבה נוירונים. במקום זאת, כל הנוירונים פעילים, אך כדי לשמר את הסקיילינג הנכון, הערכים של כל הנוירונים שעברו אימון עם Dropout מוכפלים במספר שמייצג את ההסתברות שהנוירון היה פעיל במהלך האימון (לדוגמה, אם ב-50% מהזמן כיבו נוירונים, אז בשלב הבדיקה הערכים מוכפלים ב-0.5).

**דוגמה**

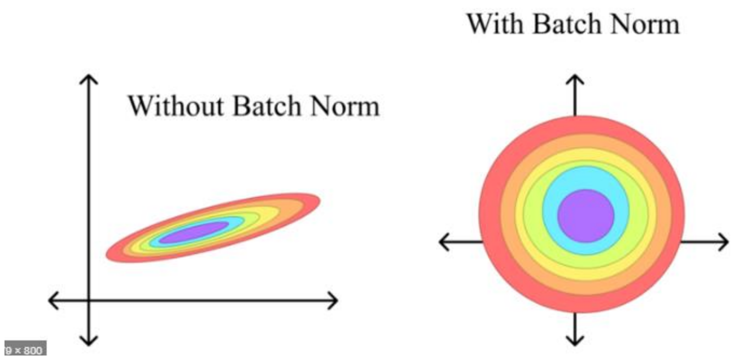
נניח שיש לנו שכבה עם 100 נוירונים, ובמהלך האימון נבחר להגדיר Dropout של 20% (כלומר, 20% מהנוירונים יכובו בכל מעבר). בכל פעם שהרשת מבצעת מעבר באימון, כ-20 נוירונים ייכבו, ורק 80 הנותרים יופעלו. זה יאלץ את הרשת ללמוד להשתמש במידע בצורה גמישה יותר, ולמנוע ממנה להיות תלויה ביותר מדי בנוירונים מסוימים.

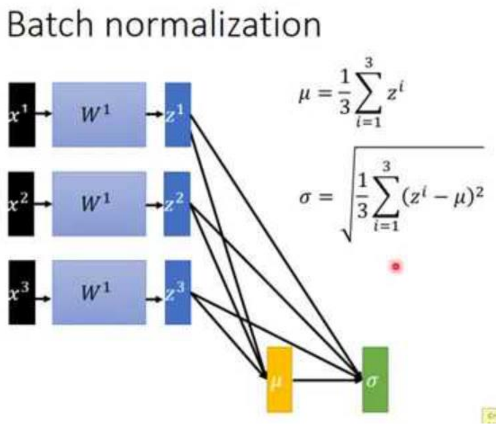
**סיכום**

שכבת Dropout היא כלי יעיל להפחתת overfitting ולהגדלת היכולת של רשת נוירונים לבצע הכללות. היא עושה זאת על ידי כיבוי אקראי של נוירונים במהלך האימון, ובכך מאלצת את הרשת ללמוד תבניות ומאפיינים רחבים יותר מהנתונים.

## שכבת Batch normalization:

שכבת **Batch Normalization** היא שכבה ברשתות נוירונים שמטרתה להאיץ את תהליך האימון ולשפר את הביצועים והיציבות של הרשת. היא עושה זאת על ידי נרמול של הקלטים לכל שכבה כך שההתפלגות שלהם תהיה אחידה יותר, תוך שמירה על יכולת למידה של המאפיינים החשובים. שכבה זו פותרת בעיות של שינויים דרסטיים בהתפלגות הנתונים בכל מעבר בשכבות הרשת, תופעה שנקראת **Internal Covariate Shift**.



**איך זה עובד?**

בכל שכבה במהלך האימון של הרשת Batch Normalization, מנרמלת את הפלט של השכבה הקודמת לפי חישוב של הממוצע והסטיית התקן של קבוצת הנתונים (batch) הנוכחית, כך שהנתונים יופצו סביב ממוצע 0 וסטיית תקן 1. זה מאפשר לשמור על יציבות ההתפלגות של הפלט של כל שכבה לאורך האימון.

תהליך ה Batch Normalization-כולל כמה שלבים:

1. **חישוב ממוצע וסטיית תקן**: עבור כל מאפיין בקבוצת נתונים (batch) נחשב ממוצע $\mu$ וסטיית תקן $\sigma$.
2. **נרמול הקלטים**: הקלטים עוברים נרמול לפי הנוסחה:
$$\hat{x_i} = \frac{x_i - \mu}{\sigma + \epsilon}$$

   כאן $\epsilon$ הוא מספר קטן מאוד שנועד למנוע חלוקה באפס.
3. **טרנספורמציה נוספת (סקיילינג והיסט)**: לאחר הנרמול, מחילים שני פרמטרים הניתנים ללמידה $\beta, \gamma$: כך שניתן לשנות את הסקייל (scale) וההיסט (shift) של הנתונים במידת הצורך:
$$\beta + \gamma x_i = y_i$$

   זה מאפשר לרשת לשמר גמישות מסוימת בתהליך הלמידה ולהחזיר את ההתפלגות למבנה מקורי אם זה יעיל יותר.

**למה זה חשוב?**

1. **האצת האימון**: Batch Normalization מאפשר למידה יציבה יותר, ולכן אפשר להשתמש בערכי קצב למידה (learning rate) גבוהים יותר מבלי לחשוש שהרשת "תתפרק" (כשתוצאה מחישובים עלולים להתפוצץ או להתכווץ מהר מדי).
2. **מניעת "Internal Covariate Shift"**: בשכבות רבות של רשת נוירונים, ההתפלגות של הקלטים משתנה עם כל שכבה. זה מכביד על תהליך האימון, כי כל שכבה צריכה להתמודד עם התפלגות חדשה של נתונים. בכל מעבר של Batch Normalization מנרמל את הנתונים בכל שלב, כך שההתפלגות נשארת יציבה יותר בין שכבות הרשת.
3. **רגולריזציה נוספת**: יש הטוענים ש- Batch Normalization גם מוסיף אפקט של רגולריזציה (הפחתת overfitting) מאחר שבאופן דומה ל-Dropout יש שונות בין נורמליזציה של כל אצווה (batch), דבר שמקשה על הרשת "להתאים יתר על המידה" לנתונים מסוימים.
4. **תלות נמוכה יותר באתחול משקלים**: נרמול הנתונים מאפשר לרשת ללמוד מהר יותר גם אם הערכים ההתחלתיים של המשקלים רחוקים מהערכים האופטימליים.

**יישום בזמן בדיקה (Inference)**

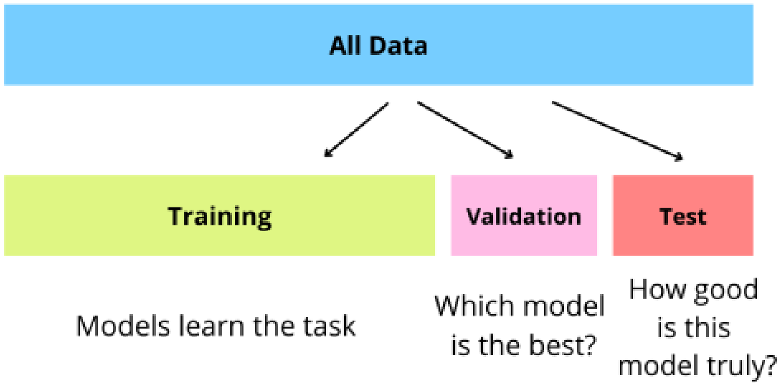
בזמן הבדיקה (שלב ה- inference) כאשר מריצים נתונים חדשים דרך הרשת, אין אפשרות לחשב את הממוצע וסטיית התקן על כל batch ולכן במקום זה משתמשים בערכים ממוצעים שנלמדו במהלך האימון. הערכים הללו נשמרים ומשמשים לנרמול הקלטים החדשים.

**דוגמה:**

נניח שיש לנו batch עם 100 דוגמאות שכל אחת מהן מכילה מאפיינים שונים.
שכבת ה- Batch Normalization תחשב את הממוצע וסטיית התקן של כל מאפיין בכל batch, תבצע נרמול של הקלטים כך שיהיו בעלי התפלגות סטנדרטית (ממוצע 0 וסטיית תקן 1), ואז תוסיף את פרמטרי γ ו β - כדי להתאים את ההתפלגות החדשה ללמידה.

## Dataset

**Dataset** (מערך נתונים) הוא אוסף של דוגמאות (דאטה), אשר משמש לאימון, בדיקה והערכה של מודל למידת מכונה. כל דוגמה במערך הנתונים מכילה את המידע הדרוש כדי לאמן את המודל לבצע משימה מסוימת, למשל, תמונות עם תוויות זיהוי, נתונים מספריים עם תוצאה רצויה וכו'. בחלוקה נפוצה של מערך נתונים, הוא מחולק לשלושה חלקים: **Train, Validation** ו- **Test** כל אחד מהם משמש שלב אחר בתהליך פיתוח המודל.




**חלוקה ל Train, Validation ו- Test**

1. **Train Set:**

- מטרת ה- Train Set היא לשמש את המודל לצורך למידה.
- במהלך שלב זה, המודל רואה את הדוגמאות בנתונים האלה שוב ושוב ומעדכן את הפרמטרים הפנימיים (המשקלים) שלו כדי ללמוד את היחסים בין הקלט לפלט.
- זהו החלק הגדול ביותר במערך הנתונים, לרוב הוא מהווה בין 60% ל-80% מהנתונים.
- לדוגמה, אם מפתחים מודל שמנבא אם תמונה היא כלב או חתול, כל דוגמה ב- Train Set תהיה תמונה ותווית המצביעה אם יש בה כלב או חתול.

2. **Validation Set:**

- מטרת ה- Validation Set היא להעריך את ביצועי המודל במהלך האימון ולכוונן את ההיפר-פרמטרים של המודל (כמו קצב למידה, אדריכלות הרשת, שכבות, Dropout וכדומה).
- המודל אינו מתעדכן ישירות מול נתוני ה- Validation,הוא משמש רק כדי לוודא שלאחר מספר סבבים של אימון, המודל לא עובר **overfitting** לנתוני האימון.
- זהו מדגם קטן יחסית מהנתונים, לרוב מהווה בין 10% ל-20% ממערך הנתונים.

- בשלב זה, נבדקים הביצועים של המודל על דאטה חדש, שלא שימש אותו ישירות לאימון. המודל בעצם רואה נתונים חדשים, אבל לא מתעדכן על פיהם.
- לאחר בדיקת הביצועים על ה- Validation Set ניתן לכוון את ההיפר-פרמטרים בהתאם לתוצאות.

3. **Test Set**:

- מטרת ה- Test Setהיא להעריך את הביצועים האמיתיים של המודל בסוף תהליך האימון והכוונון.
- נתוני ה- Test הם נתונים שהמודל לא ראה בכלל בתהליך האימון, כך שמדובר בהערכה אובייקטיבית על ביצועי המודל.
- ה- Test Set נשמר בצד במהלך כל תהליך האימון והכוונון, ומשתמשים בו רק אחרי שהמודל סיים להתאמן ולהתכוונן.
- הוא מספק הערכה כיצד המודל צפוי לתפקד במציאות (Production) כשיפגוש נתונים חדשים לגמרי.
- כמו ה- Validation Set, ה- Test Set מהווה לרוב בין 10% ל-20% ממערך הנתונים.

**למה מחלקים את מערך הנתונים?**

1. **מניעת Overfitting:** אם היינו משתמשים בכל הנתונים לצורך אימון בלבד, המודל היה עשוי "לזכור" את הנתונים בצורה טובה מדי (במיוחד אם יש מספיק פרמטרים), כלומר להתאים יתר על המידה לנתוני האימון ולא לבצע הכללה טובה על נתונים חדשים. באמצעות חלוקה ל- Validation

   ול- Test Set ניתן להעריך את היכולת של המודל לבצע הכללה על נתונים חדשים ולא "לזכור" רק את מה שראה באימון.

2. **כוונון היפר-פרמטרים**: ה- Validation Set מאפשר למדוד את הביצועים של המודל תוך כדי תהליך האימון ולכוונן פרמטרים חשובים מבלי להשתמש ב- Test Set זה עוזר לשפר את המודל מבלי לפגוע באובייקטיביות של מדידת הביצועים הסופית.
3. **הערכת ביצועים אובייקטיבית:** Test Set נותן הערכה אובייקטיבית לביצועי המודל. מאחר והוא לא שימש בשום צורה בתהליך האימון או הכוונון, הוא מהווה מדד טוב לביצועי המודל על נתונים חדשים במציאות.

**דוגמה לחלוקה:**

נניח שיש לנו מערך נתונים של 10,000 דוגמאות. ניתן לחלק אותו כך:

- 70% מהנתונים (7,000 דוגמאות) ישמשו לאימון. (Train Set)
- 15% מהנתונים (1,500 דוגמאות) ישמשו לאימות. (Validation Set)
- 15% מהנתונים (1,500 דוגמאות) ישמשו לבדיקה סופית. (Test Set)

**סיכום**

החלוקה ל- Train, Validation ו- Test היא שיטה חיונית בלמידת מכונה שמבטיחה שהמודל יתאמן, יתכוונן וייבדק בצורה נכונה על נתונים חדשים. חלוקה זו מונעת בעיות כמו overfitting מאפשרת כוונון היפר-פרמטרים ומספקת הערכה אמינה על ביצועי המודל בעולם האמיתי.

# הקדמה מתמטית:

## התמרת STFT:

התמרת פוריה לזמן קצר (STFT - Short-Time Fourier Transform) היא טכניקה לניתוח אותות המשתנים בזמן, שמאפשרת לראות את התוכן התדרי של האות לאורך הזמן. זו גרסה מורחבת של התמרת פוריה הקלאסית, שמאפשרת להתמודד עם אותות שהתכונות התדריות שלהם משתנות בזמן.

### הבעיה עם התמרת פוריה רגילה

בניתוח התמרת פוריה רגילה, אנחנו לוקחים אות בזמן (אות מאופיין ע"י אמפליטודות לאורך ציר הזמן) וממירים אותו לאות במרחב התדרים (אות מאופיין ע"י תדרים ואמפליטודות של כל תדר). אולם, התמרת פוריה רגילה לא מתייחסת לזמן שבו כל תדר מופיע באות – כלומר, היא נותנת מידע תדרי כללי על כל האות, אבל לא על הזמן שבו התדרים משתנים או מופיעים.

### איך עובדת התמרת פוריה לזמן קצר?(STFT)

STFT פותרת את הבעיה הזו בכך שהיא מחלקת את האות לחלונות זמן קטנים, ובתוך כל חלון זמן כזה היא מבצעת התמרת פוריה רגילה. כך אנחנו מקבלים מידע על התדרים שנמצאים באות בכל רגע נתון בזמן.

השלבים של STFT הם:

1. **חלוקת האות לחלונות זמן קצרים**: תחילה בוחרים חלון (בדרך כלל פונקציה מתמטית כמו חלון האן או חלון גאוסי) שאותו "מחליקים" על פני האות עם חפיפה מסוימת. בכל פעם מתבצעת חיתוך קטן של האות בתוך חלון זמן קצר.
2. **ביצוע התמרת פוריה על כל חלון זמן**: בכל חלון, מבצעים התמרת פוריה על אותו קטע קטן של האות, מה שמספק מידע תדרי עבור אותו חלון.
3. **הזזת החלון על פני האות**: מזיזים את החלון לאורך ציר הזמן ומבצעים שוב את התמרת פוריה על כל חלון חדש. בצורה זו מתקבל רצף של תוצאות שמראות איך התוכן התדרי משתנה לאורך זמן.

### התוצאה של STFT

הפלט של STFT הוא פונקציה שתלויה בשני משתנים – זמן ותדר. זהו למעשה ספקטרום תדרים שמשתנה עם הזמן, ונקרא **ספקטרוגרמה** (Spectrogram). הספקטרוגרמה היא ייצוג תלת-ממדי שבו:

- הציר האופקי, ציר X, מייצג את הזמן.
- הציר האנכי, ציר Y, מייצג את התדרים.
- והעוצמה (אמפליטודה) של התדרים בכל רגע נתון מיוצגת על ידי עוצמת הצבע או גובה ערכים על ציר נוסף.

$$STFT\{x(t)\} = X(t,w) = \int_{-\infty}^{\infty} x(\tau)w(\tau - t)e^{-jw\tau}d\tau$$

כאשר:

- $x(t)$ הוא האות המקורי.
- $w(\tau - t)$ היא פונקציית החלון הממוקמת סביב הזמן
- $w$ הוא התדר הזוויתי (ביחידות רדיאנים לשנייה).
- $e^{-jw\tau}$ הוא המרכיב האקספוננציאלי של התמרת פוריה.

# הצגת הבעיה והפתרון:

## Speech Separation - מבוא

משימת *speech separation* היא אתגר מרכזי בעיבוד אותות קוליים ובלמידה עמוקה. המטרה שלה היא "לחלץ" קולות של דוברים שונים מתוך הקלטה אחת שבה כמה דוברים מדברים בו-זמנית, כך שבסיום התהליך ניתן יהיה להאזין לכל דובר בנפרד כאילו הוא מוקלט בלעדית. אתגר קשה יותר הוא מצב שבו מספר אנשים משוחחים זה עם זה ברקע של רעש סביבתי (למשל, בבית קפה או חדר ישיבות), והמטרה היא להפריד את הדוברים ולהסיר את רעשי הרקע.

**הסבר על האתגר**

כדי להבין לעומק את האתגר הטמון במשימה הזו, חשוב להכיר כמה היבטים מרכזיים:

1. **עיבוד אותות קוליים**: בהקלטה עם דוברים רבים, האות המוקלט הוא תוצאה של סכום הקולות של כל הדוברים. מכיוון שהדוברים מדברים בו-זמנית, התדרים שלהם מתמזגים ומתקבל אות יחיד שמכיל "ערבוב" של כל הדוברים.
2. **מאפייני קול שונים**: לכל דובר יש מאפיינים קוליים שונים (כגון גובה קול, טון, קצב דיבור ועוד) שמאפיינים את הקול האינדיבידואלי שלו. הבעיה היא לזהות את המאפיינים האלו גם כשהם "מתערבבים" יחד.
3. **רעשים סביבתיים**: לעיתים קרובות הקלטות כוללות לא רק קולות דיבור אלא גם רעשים לא-רלוונטיים (כמו רעשי רקע) שמקשים עוד יותר על התהליך.

**גישות לעיבוד והפרדת קול**

השיטות הקלאסיות של עיבוד אותות להפרדת דוברים פותחו הרבה לפני עידן הלמידה העמוקה. אלו שיטות המשתמשות בעיקר במודלים מתמטיים ואותות קוליים, תוך ניצול הבדלים בתדרים, שלבים ומיקומים של מקורות הקול. הנה כמה מהשיטות המרכזיות:

### 1. סינון מרחבי (Beamforming)

בשיטה זו נעשה שימוש במערך של מיקרופונים כדי "למקד" את קליטת הקול לכיוון מסוים. הרעיון הוא ליצור סיגנל מוגבר מכיוון הדובר הרצוי ולהפחית את הצלילים שמגיעים מכיוונים אחרים.

- **שיטת Delay-and-Sum Beamforming**: בשיטה זו מתבצעת התאמה בין העיכוב של הקול שמגיע מכל מיקרופון לפי המיקום המשוער של הדובר. על ידי כך ניתן לחזק את האות מהכיוון הרצוי ולהפחית את הקולות מכיוונים אחרים.
- **Adaptive Beamforming**: גרסה מתקדמת שמנסה למצוא את כיווני הקולות השונים באופן אוטומטי על ידי חישוב הסתברויות לכיוון כל קול ועידכון הדגשים בהתאם.

### 2. הפרדת מקורות עיוורת (Blind Source Separation - BSS)

שיטה זו מתייחסת למצב שבו אין מידע מוקדם על מיקומם של הדוברים או על מספרם, ומטרתה להפריד בין מקורות הקול השונים.

- **ניתוח רכיבים עצמאיים (ICA - Independent Component Analysis)**: זוהי טכניקה שמשמשת להפרדת מקורות כאשר כל מקור נחשב כעצמאי. השיטה מנסה למצוא מטריצה שתפריד את מקורות האותות לערוצים בלתי-תלויים אחד בשני, כך שהאות הסופי יהיה קרוב ככל האפשר לאותות המקוריים של כל דובר.

3. **מסנני Wiener ומסננים אחרים**

שיטה זו מתבססת על יצירת מסנן אשר "מנחש" את האות הרצוי (הקול של דובר מסוים) לפי תכונות של האות שנמצא במרחב התדרים.

- **מסנן Wiener**: מסנן זה משתמש בהסתברויות על מנת לנחש את הרכב התדרים של הקול הרצוי ולהפחית רעשים ו"קולות מתחרים". מסנן זה נחשב למסנן אופטימלי בתנאים מסוימים.
- **Spectral Subtraction**: מסנן נוסף שבו מניחים שניתן למדוד את רעשי הרקע בתדרים מסוימים ולחסר אותם מהאות. טכניקה זו משמשת בעיקר להפחתת רעשי רקע, אך היא עשויה לשפר גם את ההפרדה של קולות הדוברים.

4. **שיטות עיבוד מבוססות זמן-תדר (Time-Frequency Masking)**

בשיטות אלה נעשה שימוש בהמרה של אות השמע לתחום הזמן-תדר (למשל על ידי ניתוח-Short). – STFT Time Fourier Transform הגישה הזו יעילה בזיהוי הבדלים בדפוסי תדר בין דוברים שונים:

- **Binary Time-Frequency Masking**: בשיטה זו יוצרים "מסיכה" בינארית לכל דובר על פני ציר הזמן-תדר, שמפרידה את התדרים השייכים לדובר אחד וממסכת את התדרים ששייכים לדוברים האחרים.
- **Wiener-Like Masking**: בשיטה זו מחילים מסיכה רכה לפי מידת ההסתברות של כל תדר להיות שייך לדובר מסוים. זה מאפשר דיכוי פחות אגרסיבי של דוברים אחרים ומשפר את איכות הצליל.

5. **שיטת Subspace**

שיטה זו מבוססת על פירוק האותות למרחבי תת-מרחב (Subspaces), כך שניתן למפות כל מקור קול לתת-מרחב משלו ולהפריד אותו מהמרחב הכולל.

- **Principal Component Analysis (PCA)**: בשיטה זו מחפשים מרחבים אורתוגונליים שמפרידים את האותות לאזורים שאינם תלויים זה בזה. PCA משתמשת בשונות של האותות כדי למצוא את הכיוונים המובהקים של כל מקור קול, אך היא לא אפקטיבית בהפרדת דוברים שאינם עצמאיים לחלוטין זה מזה.

6. **שיטות סינון מבוססות קורלציה בין מיקרופונים (Cross-Correlation)**

שיטה זו מבוססת על חישוב הקורלציה בין אותות שנקלטו על ידי מיקרופונים שונים, כאשר ההנחה היא שכל מיקרופון קולט את האות בזווית שונה ובזמן שונה.

- **Generalized Cross-Correlation (GCC)**: חישוב ההיסט (ההבדל בזמנים) שבו מתקבל מקסימום בקורלציה מסייע לקבוע את מיקום מקור הקול, ומאפשר בידוד של הדובר מכיוונים שונים לפי העיכוב.

**אתגרים בשיטות הקלאסיות**

בעוד שהשיטות הקלאסיות הן פורצות דרך בתחומן, הן מתמודדות עם כמה אתגרים מרכזיים:

- **ריבוי דוברים ורעשי רקע חזקים**: בשיטות רבות יש קושי להפריד דוברים כאשר הם קרובים בתדרים שלהם או בתכונות הקוליות.

- **תלות במידע מקדים:** חלק מהשיטות דורשות ידע מקדים על מיקום המיקרופונים או על תכונות של הדוברים.
- **איכות הפרדה משתנה**: שיטות מסוימות נותנות תוצאות פחות איכותיות בסביבות עם תנאים משתנים או עם רעשי רקע משמעותיים.

שיטות אלו, אף שהן מוגבלות יחסית ללמידה עמוקה, עדיין מספקות בסיס טוב להבנה של הפרדת קולות ונמצאות בשימוש ביישומים מסוימים, בעיקר כשאין מספיק נתונים לאימון רשתות נוירונים או כשדרוש פתרון בזמן אמת עם משאבים מוגבלים.

בעבר, נעשו ניסיונות להפריד בין דוברים בשיטות של עיבוד אותות קלאסי, אך שיטות אלו התקשו בהתמודדות עם הסביבה ה"בלתי יציבה" של קולות דיבור מורכבים.

במהלך השנים האחרונות, פותחו כמה גישות בלמידה עמוקה כדי להתמודד עם המשימה הזו, והן כוללות:

1. **Deep Clustering**: בשיטה זו, רשת נוירונים מנסה ללמוד מיפוי של מקטעי דיבור שונים למרחב מיוחד שבו מקטעי דיבור מאותו דובר "קרובים" זה לזה ומקטעי דיבור מדוברים שונים "רחוקים" זה מזה. לאחר מכן, ניתן לקבץ את המקטעים לכל דובר וליצור הפרדה.
2. **Permutation Invariant Training (PIT)**: שיטה זו מתגברת על בעיה הנובעת מכך שקשה לסווג עבור כל נקודת זמן את הדוברים השונים. בשיטה זו מאמנים רשת שמנסה להפריד את הדוברים ולוקחת בחשבון את כל האפשרויות של שיוך מקטעים לדוברים, כך שנמצא התוצאות הטובות ביותר.
3. **TasNet (Time-Domain Audio Separation Network):** גישה זו פועלת ישירות במרחב הזמן במקום בתדרים, בניגוד לרוב השיטות הקודמות שעשו שימוש בייצוג התדר של האות TasNet . התגלתה כיעילה מאוד עבור הפרדת דוברים עם שיפור ניכר בביצועים.
4. **Conv-TasNet**: גרסה מתקדמת של TasNet שמבוססת על רשתות קונבולוציה והיא משפרת את TasNetבכך שהיא מאפשרת למידה של מבנים מקומיים בקול, ובכך מציעה ביצועים טובים יותר גם בסביבות רועשות ומורכבות.

**שלבים בבניית מודל להפרדת דוברים**

1. **ייצוג האותות**: בשלב הראשון לרוב נעשית המרה של הקלט הגולמי לייצוג תדר-זמן (כמו STFT – Short-Time Fourier Transform, ) שבו ניתן להבחין בדוברים השונים לפי התדרים השונים שלהם.
2. **למידה**: בשלב זה מאמנים את הרשת הנוירונית על הקלט כדי שתוכל לזהות דוברים וליצור הפרדה מדויקת ביניהם. סוג הרשת והאובדן המתאים (למשל Loss של PIT ) תלוי בגישה שבה משתמשים.
3. **שחזור האות**: לאחר ההפרדה, יש צורך לשחזר את האותות המקוריים כדי להחזיר אותם למצב שבו ניתן להאזין להם בנפרד.
4. **שיפור ואופטימיזציה:** לאחר שהושגה הפרדה ראשונית, ניתן להשתמש בשיטות נוספות (למשל, רשתות נוספות או פוסט-פרוסיסינג) כדי לשפר את האיכות של ההפרדה.

**אתגרים נוספים ויישומים**

**אתגרים:**

1. **ריבוי דוברים ודמיון קולות**: הפרדת דוברים דומים (כמו אחים או אנשים עם קול דומה) היא בעיה קשה במיוחד, ודורשת תכונות מיוחדות לזיהוי דקויות.
2. **דובר בלתי מוכר**: לעיתים הרשת מאומנת לזהות דוברים מסוימים, אך עשויה להתקשות בהפרדת קולות מדוברים חדשים שלא נתקלה בהם קודם לכן.
3. **רעש סביבתי**: רעשי רקע חזקים או משתנים יכולים לגרום לרשת להפריד את הרעשים יחד עם הקולות ולפגום באיכות ההפרדה.

**יישומים:**

1. **שיחות וידאו ושיחות טלפון**: שיפור איכות השמע עבור משתמשים בשיחות וידאו או שיחות קוליות מרובות משתתפים.
2. **זיהוי דובר (Speaker Recognition):** הפרדת דוברים מאפשרת לזהות כל דובר בנפרד ביישומים כמו פתיחת מכשירים באמצעות קול או אבטחה קולית.
3. **מכשירים שמיעה**: מכשירי שמיעה יכולים להפריד את הקולות הרלוונטיים מהסביבה הרועשת ולהגביר את הדיבור של הדוברים הרצויים.

משימת ההפרדה של דיבור היא אתגר משמעותי, אך הפיתוחים בעיבוד אותות ובלמידה עמוקה הופכים את התחום ליותר ויותר אפקטיבי ויישומי.

## מאמר - Speech Separation

המאמר שלנו מתעסק במשימה של הפרדת דוברים בסביבה מורעשת.
מטרת משימת הפרדת דוברים היא להפריד ערבוב של אותות דיבור חופפים.
לאחרונה פותחו מספר רשתות נוירונים בשביל המשימה והגיעו להישגים מרשימים כאשר מדובר בערבוב נקי של אותות דיבור (בלי רעש). אך יש אתגר גדול להפריד דוברים בסביבה מורעשת מכיוון שלרעש יש תחום רחב של תדרים שמתערבב על הקול האנושי.

במשימת הפרדת דוברים בסביבה מורעשת, השיטה המרכזית מבוססת מסיכות, אך שיטה זו היא פגיעה לרעש שנוסף לדובר הרצוי.

פתרון אינטואיטיבי הוא להשתמש ב speech enhancement כתהליך קודם להפרדה כדי לבטל את הרעש מה-מיקס, למרות השפור הקטן, נוצרת בעיה של over-suppression שבו מודל
ה- speech enhancement יסיר בהכרח מידע מועיל כאשר מבטלים רעש.

המטרה היא להפריד בין 2 הדוברים ולהפריד אותם מהרעש ובעצם לקבל שחזור איכותי של ההקלטות שאיתם התחלנו.

לא כמו השיטות הקלאסיות של עיבוד אותות להפרדת דוברים, המאמר שלנו משתמש בשיטות של למידה עמוקה ורשתות נוירונים לפתור את הבעיה.

החידוש במאמר הוא שכדי לטפל בבעיה של הרעש, אנו נתייחס אל הרעש כעל עוד דובר עצמאי של המערכת שעליו אנו צריכים פרדיקציה

### נתוני הפרויקט

**ייצור הדאטא:**

ייצרנו בסך הכל 30,0000 דוגמאות מתוכם 22,500 דוגמאות אימון train, 5400 דוגמאות validation ו- 2100 דוגמאות עבור test.
את הקלטות הדוברים אנו לוקחים ממאגר נתונים LibriSpeech.
את הקלטות הרעש אנו לוקחים ממאגר נתונים !WHAM המכיל הקלטות רעשים מהרחוב, בתי קפה ומסעדות.

**הנתונים:**

- יש לנו הקלטות של דוברים (דובר אחד ודובר שתיים) ויש לנו הקלטות של הרעש
- מייצרים חדר בעל מידות אורך,רוחב וגובה שניתנות לשינוי.
- ישנו מיקרופון אחד בחדר.
- מעמידים 2 דוברים במיקומים ספציפים בחדר ביחס לקירות החדר וביחס למיקרופון.
- קובעים את כמות הרעש שיש בחדר Signal-to-Noise Ratio (SNR)
- קובעים את רמת הדהוד שיש בחדר RT60 – Reverberation Time
- קובעים את רמת חפיפת הדיבור בין 2 הדוברים.

אנחנו מייצרים datasets של train, validation, test שנוכל להכניס כקלט למודל.

לפני שנתחיל בתיאור המודל שבנינו, נרחיב כאן על את הגדלים השונים בהם השתמשנו בהמשך המאמר. לעיתים נציין בנוסף לשמו של הגודל אותה ציינו גם את שמו המלא. בנוסף נרחיב כאן על מושגים שונים ועל האלמנטים הבסיסיים השונים שהשתמשנו במודל שלנו.

$\boldsymbol{pickle}$ : זהו האלמנט הבסיסי שבאמצעותו אנו מאמנים את המודל (בכל איטרציה אנו מכניסים למודל מספר פיקלים כגודל הבאצ'). כל אחד מהאלמנטים הנ"ל כולל בתוכו 4 הקלטות: דובר אחד, דובר שתיים ,רעש , MIX.

$\boldsymbol{Dataset}$ : אוסף כל הפיקלים שאנו משתמשים בהם לאימון המודל.

$\boldsymbol{epoch}$ : זהו מעבר בודד של כל הפיקלים שנמצאים ב$dataset$ דרך המודל, כלומר $epoch$ בודד מסתיים לאחר שהמודל ראה את כל הפיקלים שב $dataset$.

$\boldsymbol{B = Batch}$ : תת קבוצה של ה$dataset$ כלומר אוסף של מספר פיקלים שאנו מעבירים דרך המודל במקביל, כלומר במקום להכניס למודל כל פעם פיקל בודד, אנו מכניסים בכל פעם מספר פיקלים ביחד (מטעמי יעילות).

$\boldsymbol{Iteration = Step}$ : בכל איטרציה פרמטרי המודל מתעדכנים וזאת לאחר שהעברנו $batch$ בודד דרך המודל.

$\boldsymbol{f = Sampling\ frequency\ [Hz] = \left[\frac{1}{sec}\right]}$ : זהו תדר הדגימה כלומר ספר הפעמים שהסיגנל נדגם בשנייה אחת.

$\boldsymbol{f \cdot t = T\ [samples]}$ : מספר הדגימות שדגמנו עבור סיגנל באורך $t$ שניות.

$\boldsymbol{F}$ : מספר הקרנלים השונים שהופעלו על ידי אלמנט ה *conv1D* שבתוך ה *encoder* (יורחב על כך בהמשך הספר).

$\boldsymbol{L}$ : אורך הסיגנל לאחר ביצוע ה *conv1D* שבתוך ה *encoder* (השתנה מ*T* לאורך *L* , יורחב על כך בהמשך).

$\boldsymbol{\tau}$ : הוא פרמטר המייצג את השפעת הטמפרטורה (אצלינו נקבע ל0.07 )

<u>conv1D (קונבולוציה חד מימדית)</u>

הקונבולוציה מוגדרת על ידי הפרמטרים הבאים (לעיתים ציינו מלבד שמו המלא של הפרמטר גם את הסימון של אותו הפרמטר במודל, בסוגרים מופיע הגודל שנתנו עבור הconv1D שבתוך ה encoder)

$\underline{(1)input\ channels:}$ מספר ערוצי הכניסה (לשם הבנה לתמונה בשחור לבן יהיה ערוץ כניסה בודד, ולתמונה צבעונית יהיו 3 ערוצי כניסה ($RGB$))

$\underline{(140)F \equiv \text{out_channels}:}$ כמה פילטרים (=קרנלים) שונים אנחנו רוצים להפעיל על ה- input.

$\underline{(32)win:}$ זהו למעשה גודל החלון (מילים שקולות קרנל,פילטר) שאותו אנו רוצים להעביר על סיגנל הכניסה. ה- kernel הוא מטריצה חד מימדית שהאלמנטים שלה נקראים משקלים, והם יתעדכנו בכל epoch.

בקונבולוציה חד מימדית הקרנל הוא חד מימדי, הוא אמור "לדלות את כל המידע מה *input*" ולכן כמובן עומקו של הקרנל יהיה כגודלו של *input channels*.

$\underline{(16)stride:}$ גודל הקפיצה של ה kernel

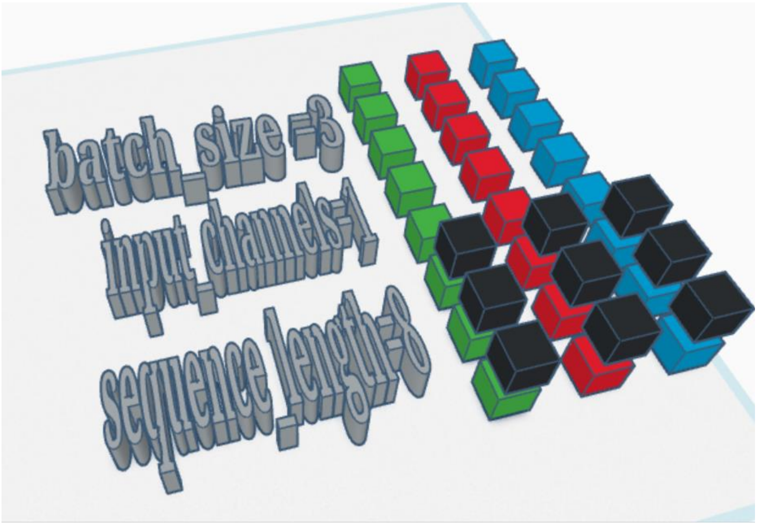
$\underline{(0)\text{Padding size}:}$ הוספת אפסים בתחילת ובסוף סיגנל הכניסה, הריפוד נעשה כדי למנוע כיווץ יתר של מימד הסיגנל עקב הקונבולציה.

גדלי הכניסה ומוצא הconv1 : $[B,1,T] \rightarrow conv1D \rightarrow [\mathrm{B},\mathrm{F},\mathrm{L}]$

אורכו של L נתון לפי הנוסחה:

$$L = \left\lfloor \frac{T - \text{kernel size} + \text{padding} \times 2}{\text{stride}} \right\rfloor + 1$$

להלן תמונה שממחישה את פעולת ה *conv1D* :




כפי שניתן לראות בתמונה, יש לנו שלושה סיגנלים בצבעים כחול, ירוק ואדום, ומעליהם שלושה קרנלים שונים בצבע שחור.

במהלך ביצוע הקונבולוציה, בכל פעם מתבצעת מכפלה איבר-איבר בין אלמנטי הקרנל לאלמנטים המתאימים בסיגנל. לאחר המכפלה, מתבצעת סכימה של הערכים שהתקבלו. לאחר מכן, הקרנל "גולש" לאורך הסיגנל ומבצע שוב מכפלה וסכימה, וכך הלאה עד שהקרנל מגיע לסוף הסיגנל. גודל ההתקדמות של החלון נקבע לפי הערך של ה- *stride*.

CONV2D

(הערה: את הקונבולציה הדו ממדית ניתן לעשות כמובן גם בצורה מקבילית על כמה כניסות שונות (כלומר עבור מקרה שיש בו Batch . לשם פשטות הסברנו את אופן פעולת הקונבולציה עבור מקרה שבו Batch=1.)

עבור batch בגודל 1 הקונבלציה פועלת על קלט שממדיו הם: $(x, y, input\ channels)$

הקונבולוציה הדו מימדית מוגדרת לפי הפרמטרים הבאים:

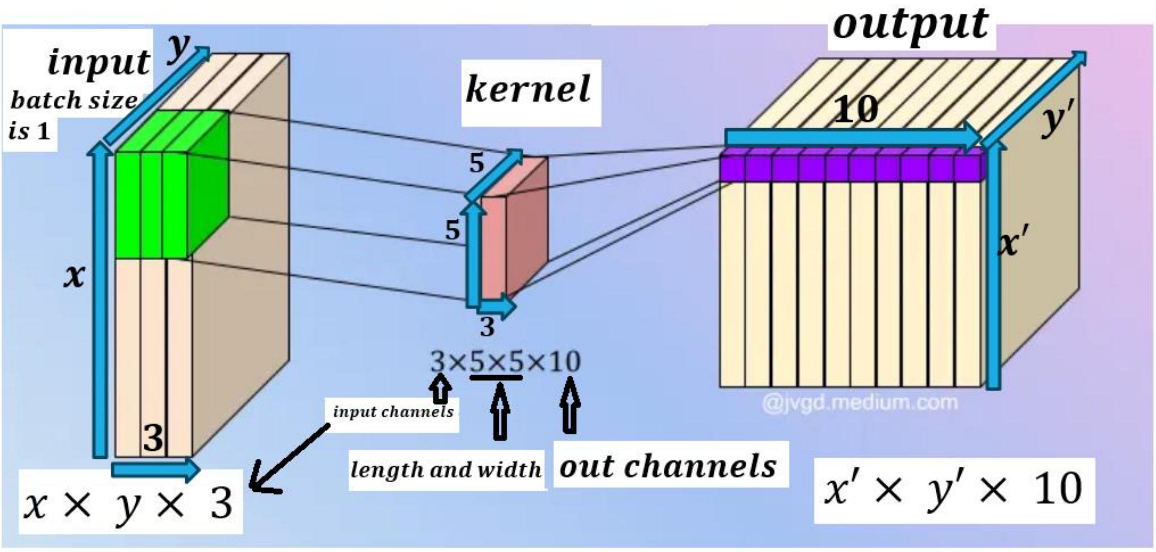
$input\ channels\ , out_channels$ מוגדרים כמו שהוגדרו ב- CONV1D.

$: win$ זהו גודל של הקרנל ממדיו הם באופן כללי: $(input\ channels, a, b)$.

$: stride$ זהו גודל הקפיצה של החלון, בקונבולציה דו ממדית זוהי גודלה של הקפיצה שנבצע תמיד במהלך תזוזת הקרנל (כיווני תנועת הקרנל הם לאורך 2 הממדים האחרונים (*x* ו- *y* )).

: Padding size כמות האפסים שאנו מוסיפים במעטפת הקלט כדי למנוע כיווץ יתר של הקלט עקב ביצוע הקונבולציה.

ניתן לראות המחשה לקונבולציה באיור למטה:




כפי שניתן לראות באיור הפעלנו 10 קרנלים שונים במהלך ביצוע הקונבולוציה, בכל פעם מתבצעת מכפלה איבר-איבר בין אלמנטי הקרנל לאלמנטים המתאימים בסיגנל. לאחר המכפלה, מתבצעת סכימה של הערכים שהתקבלו. לאחר מכן, הקרנל "גולש" לאורך הסיגנל ומבצע שוב מכפלה וסכימה, וכך הלאה עד שהקרנל כסיים לעבור על כל ה*INPUT*. גודל "הקפיצה" של החלון נקבע לפי הערך של ה– *stride*.

<u>ייצוג הSTFT במודל</u>

במודל שלנו כיננו את מוצא ה encoder "ייצוג ה STFT של סיגנל" הכניסה ל encoder ,

בחלק זה נרצה לתת הסבר לכך.

נתבונן על מוצא ה encoder:

$$x_n \equiv [B,1,T] \rightarrow \ encoder \rightarrow h_{x_n} \equiv [\mathrm{B},\mathrm{F},\mathrm{L}]$$

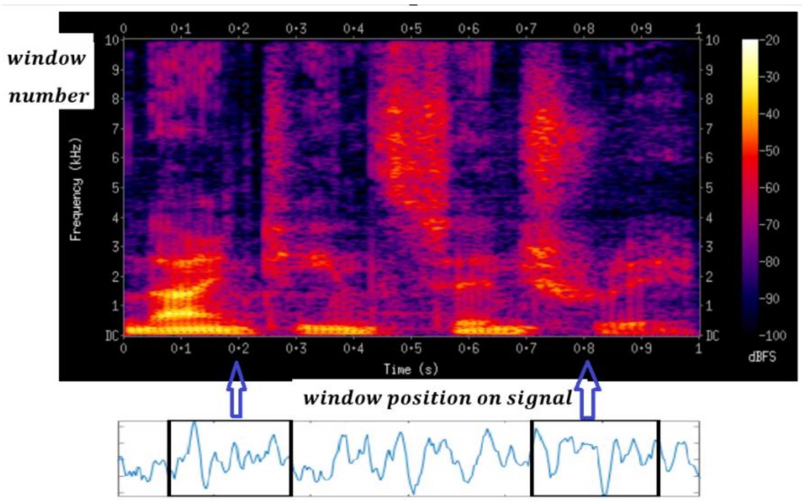
הטנסור $h_{x_n}$ הוא "מעין תצורת *stft* של הסיגנל $x_n$ ".

הסיבה לכך היא שבהינתן *batch* כלשהי- אזי עבור כל אחת מהקלטות ה*mix* (שהם במישור הזמן) הפעלנו חלון "שנע" על גבי הסיגנל בציר הזמן , מספר הקפיצות שהחלון מבצע עד שהוא מגיע לסוף הסיגנל, הוא למעשה מספר הדגימות שיש לנו בציר הזמן ב*STFT* (לפי הפרמטרים אצלינו זה *L* ) וזה עבור חלון (=קרנל) מסויים שהפעלנו וזה למעשה מייצג לנו פס תדר בודד.

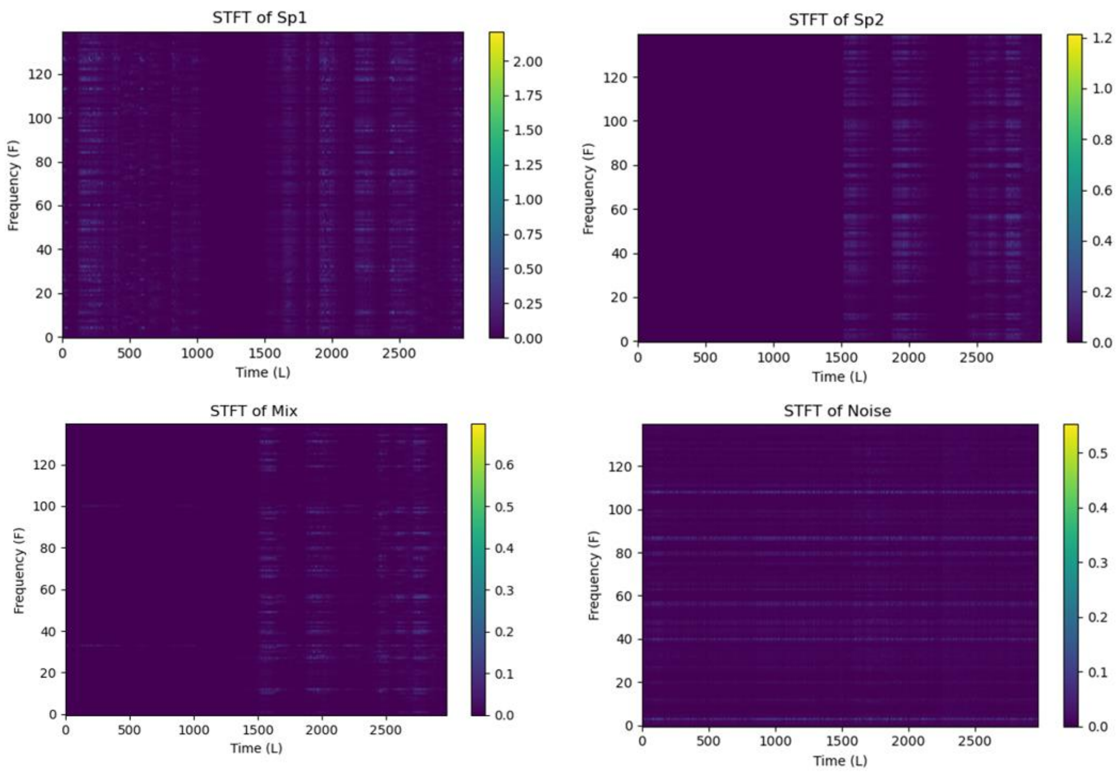
כלומר חלון בודד שאנו מחליקים על הסיגנל מייצג לנו פס תדר מסוים, ועל ידי קידום החלון אנחנו יכולים לראות: כיצד "התדר" משתנה בזמן.

אנו חוזרים על תהליך זה עבור חלונות שונים בעלי אותם מימדים, ובצורה כזו מקבלים פסי תדר נוספים (בסך הכל *F* תדרים)

(<u>הערה</u>: כמובן שהייצוג שקיבלנו לאחר שהסיגנל עבר ב *encoder* הוא לא באמת *STFT* כי הוא לא ממש את הנוסחה עבורה, אבל מבחינה רעיונית הוא די דומה). ניתן לראות המחשה לכך בתמונה למטה:




בנוסף מצורף כאן לשם המחשה ייצוגי *STFT* "אמיתיים" (כלומר לא *STFT* אמיתי אלא במובן שהסברנו בסעיף זה, כלומר ציר התדר אינו מייצג ציר תדר של *STFT* רגיל אלא של *STFT* במובן שהסברנו) של דובר אחד ושתיים, ה*MIX* ושל הרעש. שנוצרו לאחר מעבר הסיגנלים דרך ה *encoder* שקיבלנו אצלינו בפרויקט.




<u>*Cosine similarity*</u>

*Cosine similarity* מוגדר לפי:

$$\boldsymbol{A} * \boldsymbol{B} \equiv \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \cdot \sqrt{\sum\limits_{i=1}^{n} B_i^2}}$$

כאשר:

- A ו – B הם שני הווקטורים שנרצה להשוות ביניהם.
- $A \cdot B$ הוא המכפלה הסקלרית של שני הווקטורים.
- ||A|| ו - ||B|| הם האורכים (הנורמות) של הווקטורים.

ה *Cosine similarity* מקבל ערכים בין מינוס 1 ל1 .

כאשר 1 אומר שהוקטורים מצביעים לאותו כיוון

מינוס אחד אומר שהוקטורים בכיוונים מנוגדים

0 אומר שהוקטורים מאונכים זה לזה.

**למה משתמשים ב- Cosine Similarity?**

1. **אי-תלות בגודל הווקטורים**: Cosine Similarity מתמקדת בזווית בין הווקטורים ולא בגודלם. כך היא מודדת את הדמיון בכיוון של הווקטורים ולא את גודלם האבסולוטי. זה יתרון משמעותי כשעובדים עם נתונים נורמליים.
2. **קלות חישובית**: חישוב הדמיון מבוסס בעיקרו על מכפלה סקלרית, מה שהופך אותו למהיר יחסית לעיבוד ברשתות נוירונים.
3. **רלוונטי במיוחד לעיבוד שפה ולתמונות**: Cosine Similarity יעיל להשוואת משמעויות, כך ששני אובייקטים דומים (כמו מסמכים או תמונות) יהיו קרובים יותר במונחי זווית, גם אם הם שונים באורך הווקטור.

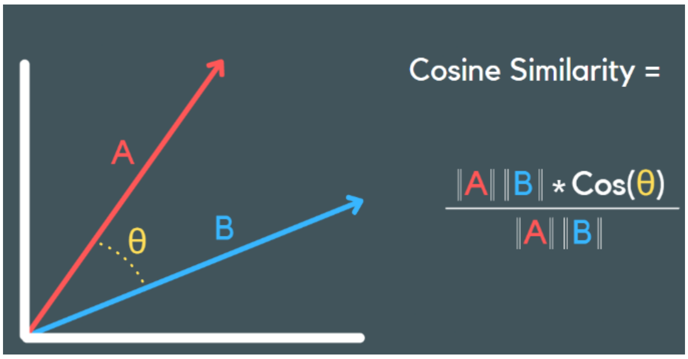
**איך Cosine Similarity משתלב ברשתות נוירונים?**

- **שכבת השוואה**: Cosine Similarity יכולה להיות שכבת הסיום במודלים שמטרתם למצוא דמיון בין אובייקטים, כמו מודלים לזיהוי פנים או מערכות חיפוש. השכבה מחושבת על ידי לקיחת תכונות שמופקות מווקטורים במימד גבוה (בדרך כלל, פלטים של שכבות Fully Connected ).
- **אימון המודל**: במקרים כאלה, המודל מאומן כך שוקטורים של אובייקטים דומים יפנו בכיוון דומה. לדוגמה: בתהליך זיהוי פנים, המודל מאומן כך שתמונות של אותו אדם ייצרו וקטורים שכיוון הזווית ביניהם יהיה קטן ככל האפשר.

**שימוש בפועל:**

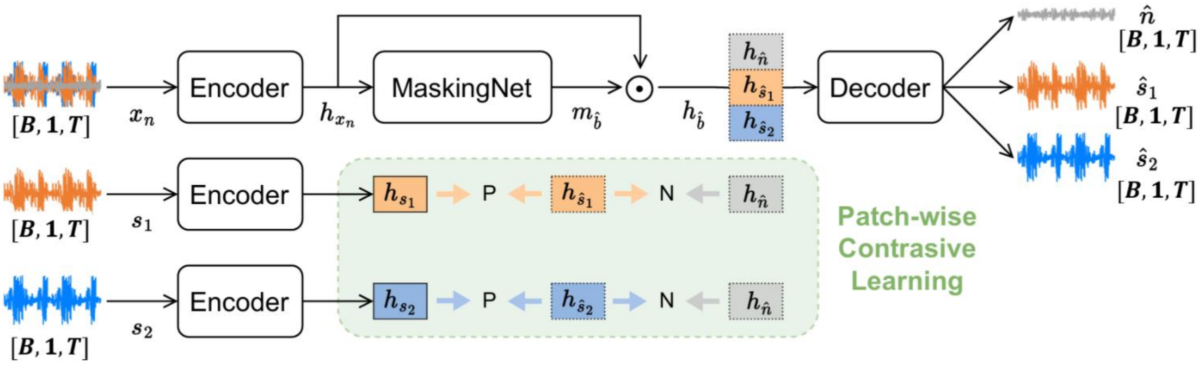
נניח שיש לנו שתי תמונות שברצוננו להשוות ביניהן. המודל יפיק וקטור תכונות עבור כל תמונה, והשכבה האחרונה תשתמש ב- Cosine Similarity כדי לחשב את הדמיון בין שני הוקטורים. אם הדמיון גבוה (קרוב ל- 1), אז ניתן להסיק שהשתי התמונות דומות מאוד.

לסיכום, Cosine Similarity היא כלי חשוב בעולם למידת המכונה המאפשר להשוות בין וקטורים מבלי להתחשב בגודלם, אלא רק בכיוון שלהם, והיא מהווה שכבת סיום יעילה לרשתות נוירונים המתמחות בזיהוי והבנת דמיון בין אובייקטים.



# המערכת:

המערכת שלנו בנויה מ2 מודלים:
1) מודל (PCL) Patch-wise Contrastive Learning עם loss משלו.
2) מודל Masking Net עם loss משלו.




## הקלט

ישנם 2 קלטים שנכנסים למערכת:

1) הקלטות נקיות של דובר ראשון, דובר שני, והרעש ( $s_1, s_2, n \in R^{B\times1\times T}$) נכנסות ל encoder של מודל ה PCL.
2) הקלטת MIX - הקלטה בודדת שמכילה 2 דוברים ורעש על אותה ההקלטה, מסומן בתור $x_n \in R^{B\times1\times T}$ נכנסת ל encoder של מודל Masking Net.

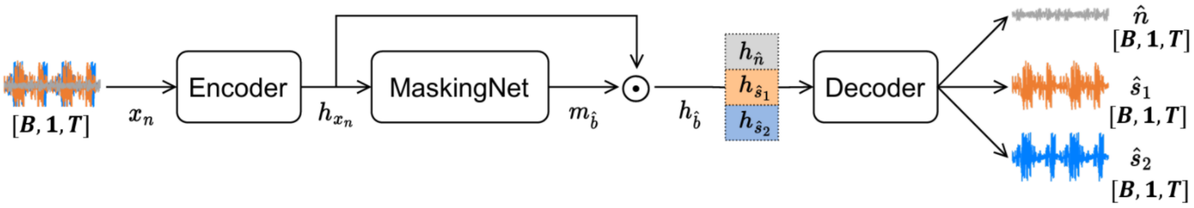
הערה: 2 הרשתות משתמשות ב encoder זהה (כלומר עם אותם משקלים בדיוק)

## הפלט

רשת Masking Net מוציאה 2 פלטים:
1. שלושה "שערוכי STFT " עבור דובר ראשון, דובר שני והרעש. ( $h_{\hat{s}_1}, h_{\hat{s}_2}, h_{\hat{n}} \in R^{B\times1\times F\times L}$ ) שמשמשים כקלט עבור רשת ה *PCL*.
2. בנוסף הרשת מוציאה בתור פלט שלושה שערוכים (במישור הזמן) עבור הקלטת הדובר הראשון, הדובר השני והרעש. ($\hat{s}_1, \hat{s}_2, \hat{n} \in R^{B\times1\times T}$)

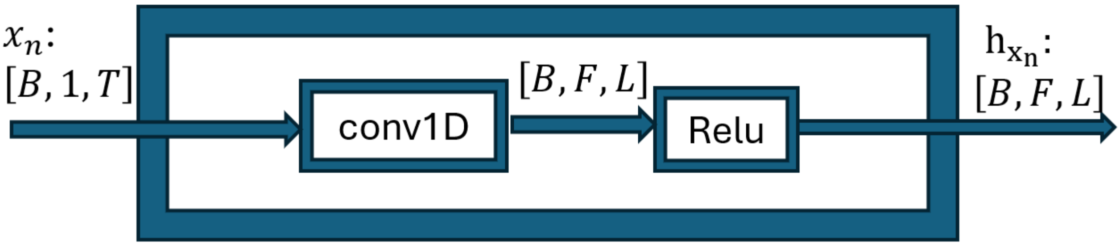
## מודל Masking Net:

רשת ה Masking Net הינה הרשת:




ראשית אנו מכניסים את סיגנל ה MIX ($x_n \equiv mix \in R^{B\times1\times T}$) דרך ה encoder כנראה באיור למטה:

**Encoder:**

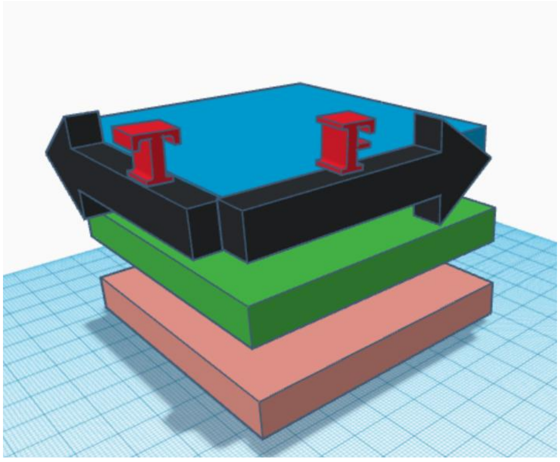


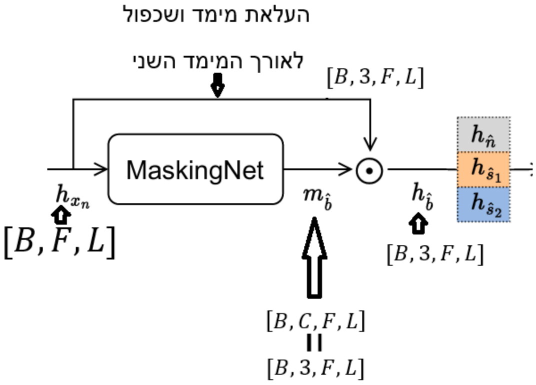

הסיגנל עובר conv1D ולאחר מכן Relu (ניתן לראות הרחבה בעמודים: 18,34)

הסיגנל $h_{x_n}$ הוא למעשה "תצוגת" STFT של סיגנל $x_n$ (ניתן לראות הרחבה בעמוד: 36)

"ייצוג הSTFT" של הסיגנל $x_n$ ($h_{x_n}$) הוא הקלט לרשת ה *Masking Net* .

מוצא הרשת: $m_{\hat{b}} \in R^{B\times3F\times L}$, הוא "טנסור משקלים" שכולל בתוכו 3 מטריצות משקלים ("3 מסכות") שכל אחד מהן גודלה כגודלו של $h_{x_n} \in R^{B\times F\times L}$ . כל אחת מן המסכות מוכפלת איבר איבר ב$h_{x_n}$, ובצורה כזאת נקבל 3 שערוכים לתצוגות הSTFT עבור דובר אחד, דובר שתיים והרעש. ( $h_{\hat{s}_1}, h_{\hat{s}_2}, h_{\hat{n}} \in R^{B\times1\times F\times L}$ ).
(הבחירה איזה מסיכה מתאימה לאיזה דובר, אינה טרוויאלית ונעשית באמצעות פרמוטציות עליהם נרחיב בהמשך).



כפי שניתן לראות באיור

הרשת מקבל קלט $h_{x_n} \in R^{B\times F\times L}$ (כאן באיור מתוארת דוגמה עבור B=3), ניתן לראות שהכניסה היא למעשה 3 "תוצאות של STFT (עם ציר "זמן T וציר "התדר" F) עבור כל אחד מסיגנלי הMIX שבתוך הbatch

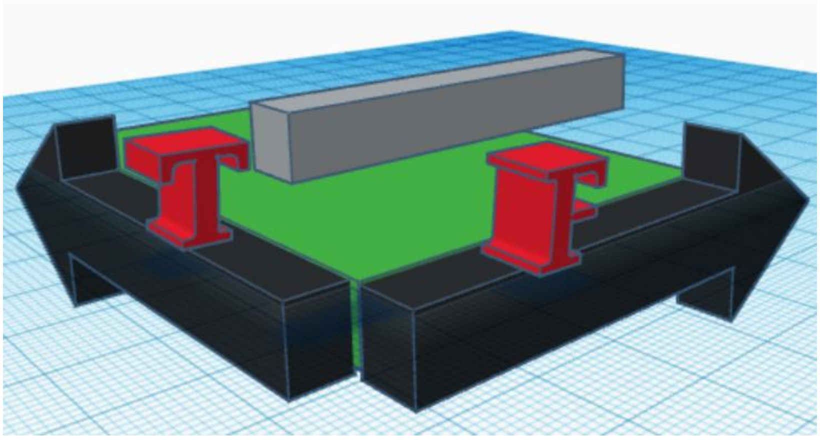
הפלט של רשת הMASKING הוא $m_{\hat{b}}$ שמייצג לנו את שלושת המסכות שיוכפלו איבר איבר במטריצת הSTFT המתאימה להם .

רשת הMASKING בנויה לפי ארכיטקטורת TCN,(Temporal Convolutional Network )

כל הקונבולציות החד מימדיות השונות המתבצעות בתוך הרשת מוגדרות כך שמימד הכניסה שלהם ( input dim) הוא בדיוק כאורכו של ציר התדר (אורכו של ציר התדר משתנה לאורך המעבר ברשת, כתוצאה מהמעבר באלמנטים השונים (אורכו יכול להיות: $F, Bn\ dim, hidden\ dim$ )).

כלומר כיוון תנועת החלון היא לאורכו של ציר הזמן "במישור *STFT*" .

כפי שנראה כאן באיור:



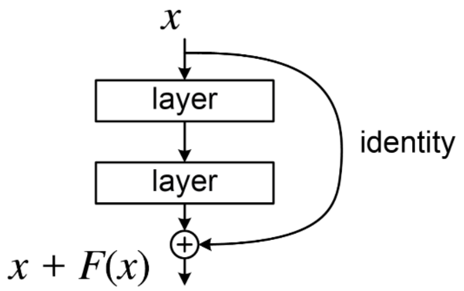
הסיבה לכך היא שבצורה כזאת אנחנו תופסים קשרים ותלויות שונות שתלויות בזמן.

כלומר עיקר המידע שלנו יהיה מרוכז לאורכו של ציר הזמן,ואותו אנו מנסים לתפוס.

זאת בניגוד לשיטות אחרות המשתמשות בייצוג *STFT* אמיתי לשם ביצוע ההפרדה, שם מעבר לאורכו של ציר התדר מספר לנו אילו תדירויות של הסיגנל דומיננטיות בכל זמן בהקלטה.אצלינו לעומת זאת ציר התדר הוא פחות אינפורמטיבי ולכן כיוון תנועת הקרנל היא כפי שציינו.

רשת ההפרדה מבוססת על ארכיטקטורת *TCN (Temporal Convolutional Network)* והיא מתאפיינת בתכונות הבאות:

1. חיבורים עוקפים בין שכבות שונות *(skip connections)*.




בעזרת החיבורים העוקפים ניתן להתגבר על 2 בעיות מרכזיות

<u>בעיית הגראדינט הנעלם:</u> *(Vanishing gradient)*. בעיה זו נובעת מכך שעבור רשת בעלת שכבות רבות , חישוב כל אחד מהגראדינטים הינו (לפי כלל השרשת) מכפלה של הרבה משקלים אילו באילו, ועבור משקלים קטנים מ1 , ורשת עם שכבות רבות , המכפלה תדעך במהירות לאפס, ובכך עדכון המשקלים יהיה איטי מאוד. בעזרת חיבורים עוקפים בין השכבות השונות אנו יוצרים "קיצור דרך" עבור הגראדינטים של כל שכבה כך שהם יוכלו לעקוף את השכבה ובכך להימנע מהכפלה במספר גדול של משקלים מה שיוביל לדעיכת הגראדינט.

<u>בעיית שכבות לא אינפורמטיביות:</u> ייתכנו שכבות מסוימות במודל שמסיבות מסוימות אינם מועילות למודל, והיה עדיף שלא יופיעו במודל, במקרה כזה המודל ירצה עבור השכבה הבעייתית ללמוד משקלים שייתנו את מיפוי היחידה, כלומר משקלים שיגרמו למוצא ולכניסת השכבה להיות זהים.

למידת המשקלים הנ"ל היא לא משימה פשוטה, וזאת מאחר שיש שכבות רבות במודל שמיוצגות על ידי פונקציה מורכבת ולכן מציאת משקלים שיגרמו למודל "לדלג כל שכבה אחת" תהיה לא פשוטה.

הוספת חיבורים עוקפים עוזרת להתגבר על בעיה זו בכך שהרשת יכולה לאפס את המשקלים של השכבה הבעייתית ועל ידי כך לממש את מעבר הזהות (ניתן לעזור לרשת לעשות זאת על ידי הוספת נרמול לפונקציית ההפסד שמעודד את הרשת לבחור משקלים נמוכים , על ידי הענשה בפונקציה ההפסד כאשר לוקחים משקלים גבוהים
(נרמול מקובל הוא *L2*: $Loss_{total} = Loss_{original} + \lambda \sum W^2$ )

2. שימוש בקונבולוציה מדוללת *(Dilated Convolutions)*.

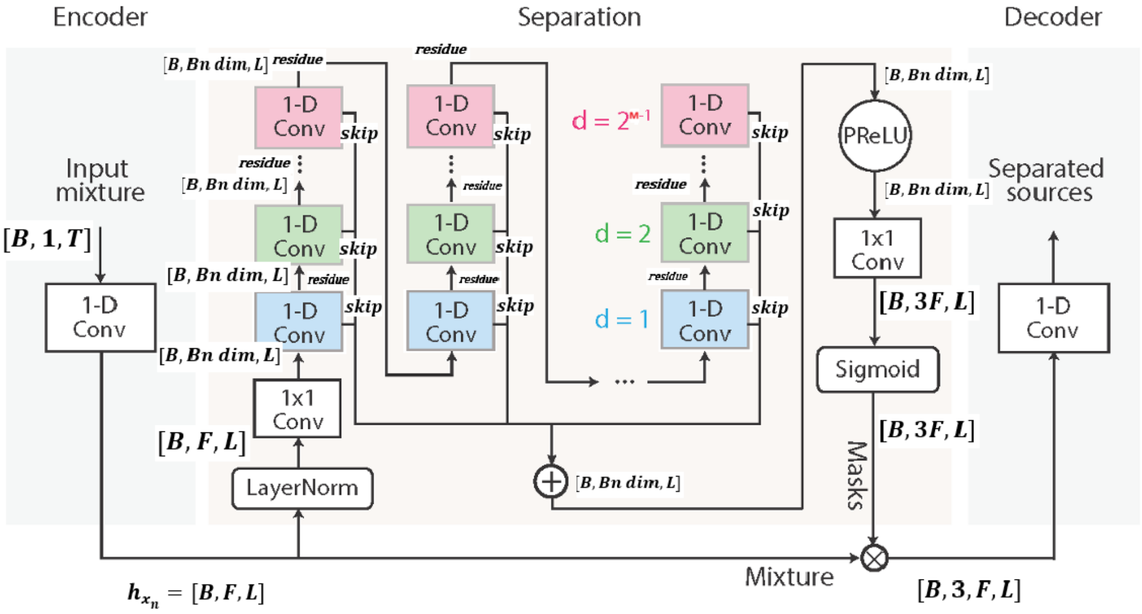
הרשת משתמשת בקונבולציה חד מימדית עבור קרנלים המרופדים באפסים בין כל 2 משקולות של החלון , דבר זה מאפשר למודל לקלוט **תלויות ארוכות טווח** באות הקלט בצורה יעילה יותר.

השימוש בדילול מאפשר לכל שכבה ברשת לכסות קטע גדול יותר של האות הנקלט, מה שמאפשר למודל "לראות" חלקים רחבים יותר של האות, מבלי צורך להוסיף שכבות נוספות (דבר שיכול להגדיל את המורכבות החישובית וזמן האימון).

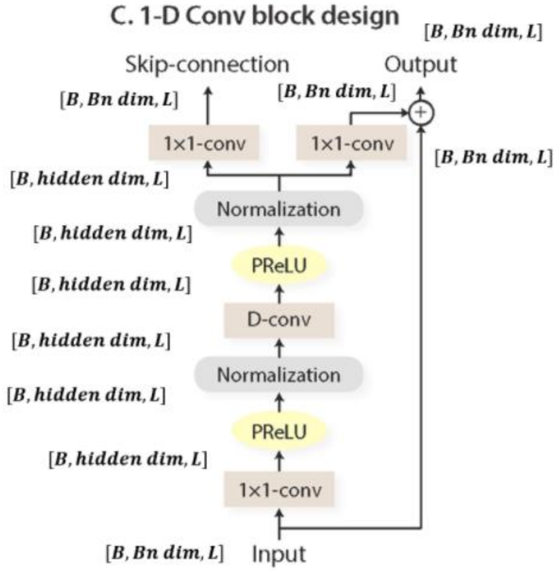
בנוסף על ידי שימוש בדילולים שונים הרשת יכולה ללמוד דפוסים ארוכי וקצרי טווח של האות.

3. שימוש בשכבות נורמליזציה וברכיבים לא לינארים: (ניתן לראות הרחבה בעמודים: 17,22).

מבנה רשת ההפרדה מוצג בתמונה הבאה:




כאשר כל אחד מבלוקי $Conv\ 1D$ נראה כך:




הכניסה והמוצא של הרשת מסומנים כך:

*input dim*= זהו מימד הכניסה של הכניסה לרשת ה *Masking* שהוא למעשה אורך "ציר התדר" שזה בדיוק מספר הפילטרים שהופעלו ברכיב הencode לפני הכניסה לרשת ה *Masking*.

(באיור הכניסה לרשת מסומנת ב $h_{x_n} \in \mathbb{R}^{B\times F\times L}$ ו, *input dim=F* )

*Output dim* = זהו גודל השווה ל $C \cdot F = 3F$ הסיבה לכך היא שאנו מנסים לשערך 3 שערוכים של "תצוגות *STFT*" עבור סיגנל דיבור של דובר אחד, דובר שתיים והרעש.

אנחנו יודעים ש "*STFT*" של כל אחד משלושת הסיגנלים הינו במימדים של $[B, F, L]$ ומכאן שאנחנו רוצים חיזוי של 3 הסיגנלים אנו מעוניינים למעשה ב3 ייצוגי *STFT* ולכן גודלו של *Output dim* הוא חייב להיות *3F*. הדאגה לכך שמימד התדר יהיה באורך הרצוי נעשה על ידי הרכיב $Conv\ 1 \times 1$

שנמצא סמוך למוצא הרשת.

בכניסה לרשת מתבצע נרמול.

ולאחר מכן קונבולוציה חד מימדית רגילה. ואז אנו נכנסים לשרשרת מחזורית של בלוקי $1 - D\ conv$

כאשר הרשת בנוייה משרשורים של הבלוקים הנ"ל ,כאשר *M* הבלוקים  משוכפלים אחד אחרי השני  בסך הכל *R* פעמים .

נציין שכפי שניתן לראות באיור רצף של *M* הבלוקים הוא זהה לחלוטין לרצפי ה*M* בלוקים האחרים, אבל *M* הבלוקים עצמם אינם זהים זה לזה (מבחינת הפעולות שמבצעים הם כן זהים אבל הפעולות מתבצעות עם פרמטרים שונים).

*M*  מציין את גודל ה *LAYER* כלומר כמה בלוקי  $1 - D\ conv$ יש בכל שרשור

*R*  מציין את גודל ה *STACK* כלומר כמה פעמים שכפלנו כל שרשור.

כל אחד מבלוקי $1 - D\ conv$ מוגדר לפי הפרמטרים הבאים:

*Bn dim* קובע את מימד מימד התדר שיהיה בכניסה ובמוצא של כל אחד מבלוקי $1 - D\ conv$

*Hidden dim* קובע את מימד התדר שיהיה בתוך בלוק ה$1 - D\ conv$

(הפרמטרים הללו שולטים על אורכו של מימד התדר בכל שלב . (דבר זה נעשה על ידי קביעת מספר הפילטרים (=החלונות השונים) שיופעלו על ידי כל אחת מהקונבולציות החד ממדיות)).

בתוך כל אחד מהשכפולים, בלוקי ה$1 - D\ conv$ נבדלים זה מזה במבנה הקרנל של הקונבולציות החד מימדיות שבתוך הבלוק.

מספר האלמנטים של הקרנל אומנם נשאר זהה אבל בין כל 2 משקלים של החלון יש ריפוד בכמות של $2^{d-1}$ אפסים  $\{\ d \in [0, M - 1]\ \}$ הנקבעת לפי מיקום הבלוק בשרשת

וזאת כפי שניתן לראות כאן בטבלה עבור: $dilation \equiv d = 1$ (אצלינו ה *stride =1* )

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| סיגנל ← | *X* | *X* | *X* | *X* | *X* | *X* | *X* |
| קרנל בלוק 1 ← | | | | | 3 | 2 | *1* |
| קרנל בלוק 2 ← | | | 3 | 0 | 2 | 0 | 1 |
| קרנל בלוק 3 ← | 3 | 0 | 0 | 2 | 0 | 0 | 1 |

ניתן לראות גם שככל שהקרנל "מתרחב" נדרש לרפד את סיגנל הכניסה באפסים מ2 צידיו בכמות הולכת וגדלה של אפסים.

וזאת כדי שלאחר ביצוע הקונבולוציה הסיגנל לא יתקצר.

התועלת בביצוע הקונבולציות בצורה הנ"ל היא שאנו יכולים לתפוס תלויות ארוכות טווח שלסיגנל הכניסה , וזאת מכיוון שהקרנל מכסה טווחים גדולים יותר.

כפי שניתן ל ראות מוצא כל אחד מבלוקי $1-D\ conv$ מתפצל ל2 מוצאים: (המוצאים כמובן שווים)

*skip, residue*

*residue* מעביר את מוצא ה$1\ D\ conv$ לכניסה של הבלוק הבא אחריו (מסומן בתור *residu* )

*skip* מעביר את מוצא הבלוק מיוחד המבצע בסופו של דבר סכימה של כל מוצאי ה$1\ D\ conv$

(ה*skip* מתפקד כחיבור עוקף בן שכבות שונות עליו כבר הרחבנו לפני כן.)

בסופו של דבר לאחר מעבר ברשת ה *Masking* אנחנו מקבלים טנסור בגודל $[B, 3F, L]$

שאותו אנחנו מפצלים למעשה ל3 טנסורים נפרדים שגודל כל אחד מהם הוא $[B, F, L]$

כל אחד מ3 הטנסורים הללו הוא "מסיכה" שאותו נכפיל בייצוג ה*STFT* של ה*MIX*

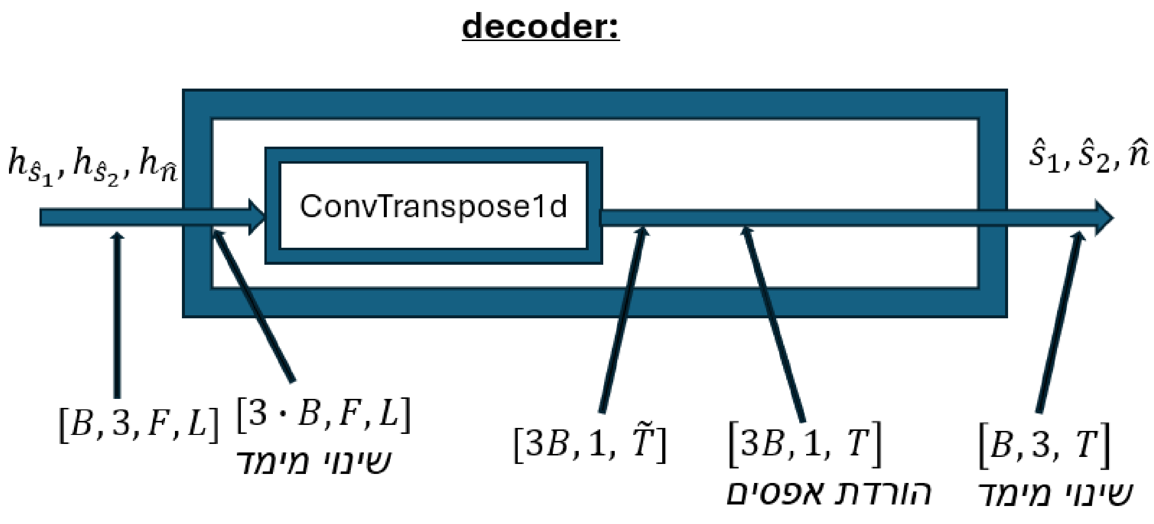
כלומר אנחנו כופלים את ייצוג ה*STFT* של ה*MIX* ב3 מסכות שונות (הכפילה מתבצעת לאורך ממדים 1 ו2 איבר איבר ( *B* נחשב מימד מספר 0 )

ובצורה כזאת מקבלים 3 ייצוגי *STFT* חדשים ( $h_{\hat{s}_1}, h_{\hat{s}_2}, h_{\hat{n}}$ ) שתפקידם לייצג את ה *STFT* המשוערך של דובר אחד, דובר שתיים ושל הרעש. וזאת על ידי זה שנכפיל את ייצוג ה*STFT* של הסיגנל ה *MIX* ב3 מטריצות משקלים שונות (=שלושת המסכות).

כדי לשייך כל אחת מ3 המסכות לאיזה ייצוג *STFT* היא שייכת השתמשנו "בפרמוטציות" (נרחיב על כך בהמשך).

כעת כדי לקבל את השערוך לדוברים ולרעש (במישור הזמן) $(\hat{h}_{s_1}, \hat{h}_{s_2}, \hat{n})$ נעביר כל אחד מייצוגי ה*STFT* דרך ה *decoder* לקבלת הסיגנל במישור הזמן

המעבר ב *decoder* מתבצע כפי הנראה באיור:




כעת בעזרת 3 הסיגנלים המשוערכים: $\hat{s}_1, \hat{s}_2, \hat{n}$ ו3 הסיגנלים היודעים לנו: $s_1, s_2, n$ (*grand truth* )

נחשב את *(Scale-Invariant Signal-to-Distortion Ratio) Loss SI-SDR* שמוגדר כך (כאן *C=2* כי יש לנו 2 דוברים):

$$\mathcal{L}_{si-sdr} = \frac{1}{C+1}\sum_{b=1}^{C+1} -10\log_{10}\left(\frac{\left\|\frac{\langle \mathbf{y}_b,\hat{\mathbf{y}}_b\rangle}{\|\mathbf{y}_b\|^2}\mathbf{y}_b\right\|^2}{\left\|\frac{\langle \mathbf{y}_b,\hat{\mathbf{y}}_b\rangle}{\|\mathbf{y}_b\|^2}\mathbf{y}_b - \hat{\mathbf{y}}_b\right\|^2}\right)$$

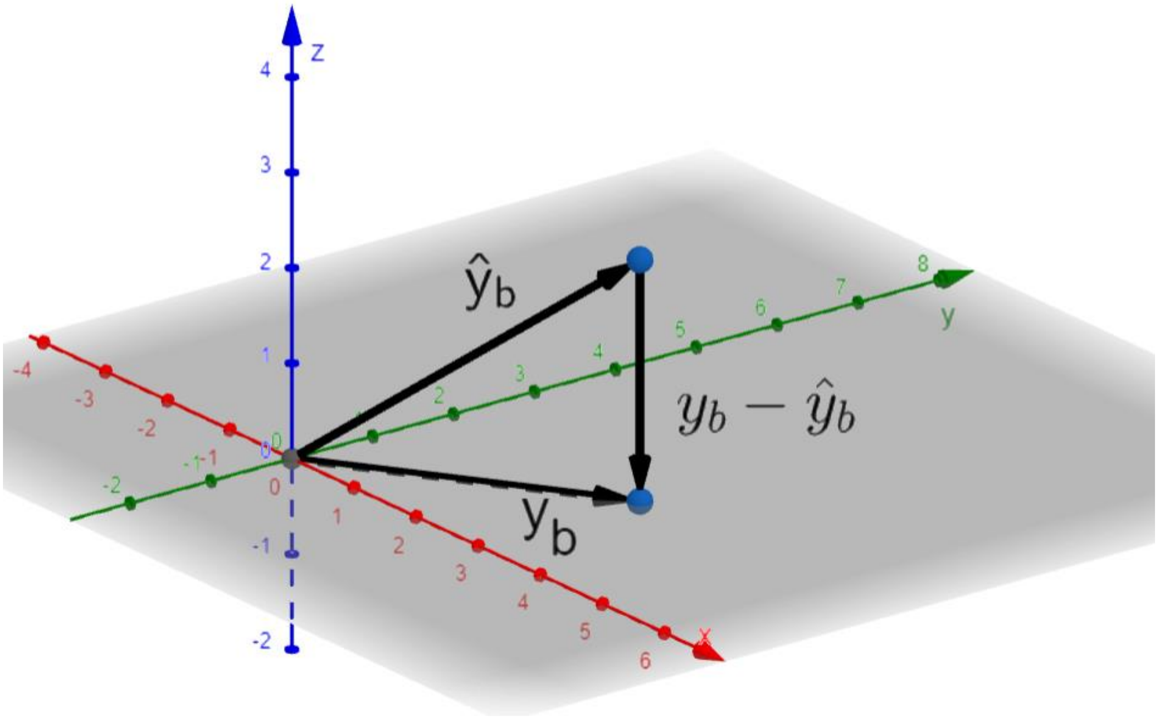
כאשר: $\boldsymbol{y_b} \in R^T$ זה סיגנל האודיו האמיתי ($\boldsymbol{y_1}, \boldsymbol{y_2}, \boldsymbol{y_3}$ זה סיגנל האודיו של דובר אחד דובר שתיים ושל הרעש).

$\boldsymbol{\hat{y}_b} \in R^T$ זה סיגנל האודיו המשוערך שיקבלנו במוצא ה *decoder* ($\boldsymbol{\hat{y}_1}, \boldsymbol{\hat{y}_2}, \boldsymbol{\hat{y}_3}$ זה סיגנל האודיו המשוערך של דובר אחד דובר שתיים ושל הרעש).

כדי להבין מה בדיוק מבטאת ה *Loss* נסתכל על המרכיבים שלה:

$\frac{<y_b, \hat{y}_b>}{||y_b||^2}\boldsymbol{y_b}$ הוא ההיטל האורטוגנלי של הוקטור $\boldsymbol{\hat{y}_b}$ על הוקטור $\boldsymbol{y_b}$.

נמחיש זאת בצורה גרפית עבור מקרה שבו $\boldsymbol{y_b}, \boldsymbol{\hat{y}_b} \in R^3$ , נציין שאצלנו המימד של הוקטורים גבוהה בהרבה ולכן לא ניתן להמחיש זאת מבחינה גרפית אך הרעיון זהה.




לפי הגדרה ההיטל האורתוגנלי מקיים: $<\boldsymbol{\hat{y}_b}, \boldsymbol{\hat{y}_b} - \boldsymbol{y_b}> = 0$. כאשר:

$\frac{<y_b,\hat{y}_b>}{||y_b||^2}y_b$ הוא ההיטל האורתוגנלי של של הוקטור $\hat{y}_b$ על הוקטור $y_b$, והערכים שהוא יכול לקבל בסופו של דבר יהיו: $\zeta \in \mathbb{R}$, $\zeta \cdot y_b$ , ניתן להבין זאת על ידי זה שנכתוב את הוקטור $\hat{y}_b$ כצירוף לינארי של 2 וקטורי יחידה, שאחד מהם מקביל לוקטור $y_b$ ($e_{y_b||}$) והשני מאונך לו ($e_{y_b\perp}$)

$$\frac{<y_b,\hat{y}_b>}{||y_b||^2}y_b = \frac{<\gamma e_{y_b||},\alpha e_{y_b||}+\beta e_{y_b\perp})>}{||\gamma e_{y_b||}||^2}\gamma e_{y_b||} = \frac{\gamma\cdot\alpha+0}{\gamma^2}\gamma e_{y_b||} = \alpha e_{y_b||}$$

ניתן לראות מהפיתוח שעשינו שההיטל האורתוגנלי של הוקטור $\hat{y}_b$ על הוקטור $y_b$ (ששווה לפי הגדרה ל: $\frac{<y_b,\hat{y}_b>}{||y_b||^2}y_b$ ). מייצג למעשה את "החלק של הוקטור $\hat{y}_b$ (שזה בעצם $\alpha \in \mathbb{R}$ , $\alpha e_{y_b||}$ ) ששייך לוקטור $y_b$ ".

<u>הערה</u>: נציין שבתמונה שהוספנו זה מתאים עבור מקרה שבו $\alpha = ||y_b||$, כמובן שבפועל $\alpha \in \mathbb{R}$ וההטלות שנראה מבחינה גרפית יראו שונות (יראו כמו: $\alpha \in \mathbb{R}$ , $\alpha e_{y_b||}$ )

כעת נסתכל על המכנה: $\frac{<y_b,\hat{y}_b>}{||y_b||^2}y_b - \hat{y}_b$ זה למעשה "הווקטור המאונך שבתמונה"

אם כן ניתן להבין שמקסום ה *Loss SI SDR*

שקול לכך שוקטור השערוך $\hat{y}_b$ קרוב ככל הניתן לוקטור האמיתי $y_b$ וזאת במובן שרוצים

שנורמת וקטור השגיאה (הוקטור המאונך בתמונה) יהיה כמה שיותר קטן

ושנורמת ההיטל האורתוגנלי של של הוקטור $\hat{y}_b$ על הוקטור $y_b$ יהיה כמה שיותר גבוהה.

כמובן כפי שניתן לראות הכפלנו את הביטוי של ה *Loss SI SDR* במינוס 1 ולכן כעת השקול יהיה למזער את הביטוי הנ"ל.

ה *target* שאנחנו הגדרנו היה: $s_1, s_2, n$ לפי הסדר הזה.

במוצא ה *decoder* קיבלנו 3 שערוכים לסיגנלים הנ"ל $\hat{s}_1, \hat{s}_2, \hat{n}$ . אולם אנו לא יודעים איזה שערוך מתאים לאיזה סיגנל.

נסמן את 3 השערוכים שקיבלנו בשמות: $estimated\ 0, estimated\ 1, estimated\ 2$

נסמן זאת בקיצור בטבלה באמצעות 0,1,2

למעשה קיימים $3! = 6$ תשובות לשאלה איזה שערוך מתאים לאיזה סיגנל. המוצגים כאן בטבלה למטה:

| | $n$ | $s_2$ | $s_1$ |
|---|---|---|---|
| פרמוטציה 1 | 2 | 1 | 0 |
| פרמוטציה 2 | 1 | 2 | 0 |
| פרמוטציה 3 | 2 | 0 | 1 |
| פרמוטציה 4 | 1 | 0 | 2 |
| פרמוטציה 5 | 0 | 2 | 1 |
| פרמוטציה 6 | 0 | 1 | 2 |

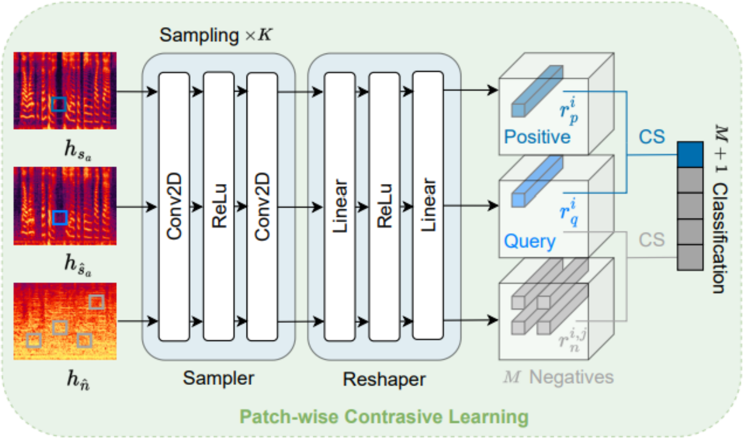
כעת חישבנו עבור כל אחת מ6 הפרמוטציות את *Loss SI SDR*, ואז בחרנו עבור את הפרמוטציה עבורה *Loss SI SDR* (המופיע למטה) היה המינימלי:

$$\mathcal{L}_{si-sdr} = \frac{1}{C+1}\sum_{b=1}^{C+1} -10\log_{10}\left(\frac{\left\|\frac{\langle \mathbf{y}_b,\hat{\mathbf{y}}_b\rangle}{\|\mathbf{y}_b\|^2}\mathbf{y}_b\right\|^2}{\left\|\frac{\langle \mathbf{y}_b,\hat{\mathbf{y}}_b\rangle}{\|\mathbf{y}_b\|^2}\mathbf{y}_b - \hat{\mathbf{y}}_b\right\|^2}\right)$$

בצורה כזאת הצלחנו למעשה לשייך את 3 שערוכי הסיגנלים שבמוצא ה *decoder* למה מייצג שערוך של דובר אחד, דובר שתיים והרעש.

וגם לקבוע איזה ייצוג *STFT* מייצג איזה *STFT* של הסיגנל המקורי.

**<u>מודל PCL:</u>**




**קלט שנכנס למערכת:**

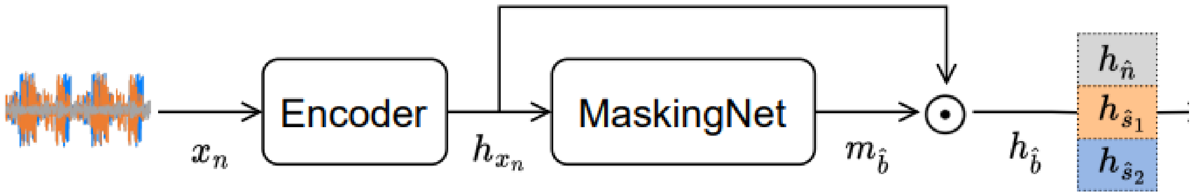
לרשת הPCL נכנסים 5 קלטים:

<u>2 ייצוגי STFT "אמיתיים":</u> $h_{s_1}, h_{s_2}$ שמתקבלים על מעבר של כל אחד מסיגנלי הדוברים הנקיים:

$s_1, s_2 \in R^{B\times1\times T}$ דרך ה encoder

$$s_1, s_2 \in R^{B\times1\times T} \rightarrow \text{encoder} \rightarrow h_{s_1}, h_{s_2} \in R^{B\times F\times L}$$

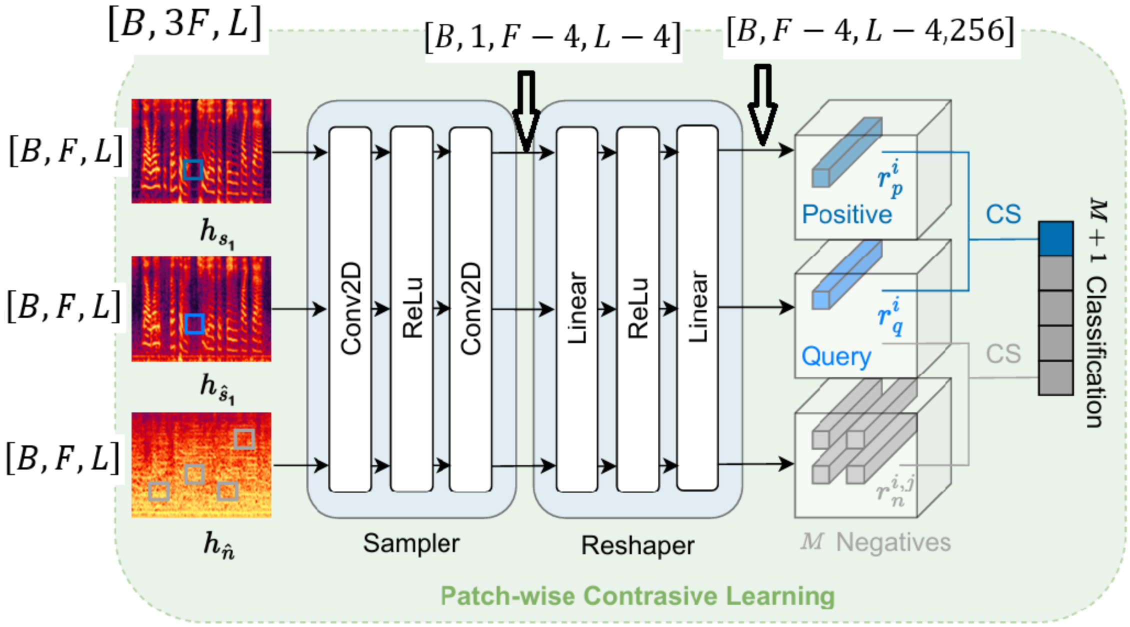
<u>3 ייצוגי STFT "משוערכים":</u> $h_{\hat{s}_1}, h_{\hat{s}_2}, h_{\hat{n}} \in R^{B\times F\times L}$ שמתקבלים בעזרת רשת ה *Masking Net* (כפי שנראה באיור:




בכל איטרציה מערכת ה PCL עושה שימוש ב5 הכניסות שציינו, כאשר כל הפעולות עבור דובר אחד ודובר 2 הינם זהות ומתבצעות במקביל. לשם נוחות, כדי לראות את אופן פעולת הרשת נתמקד בפעולת רשת ה PCL עבור דובר מספר אחד בלבד .

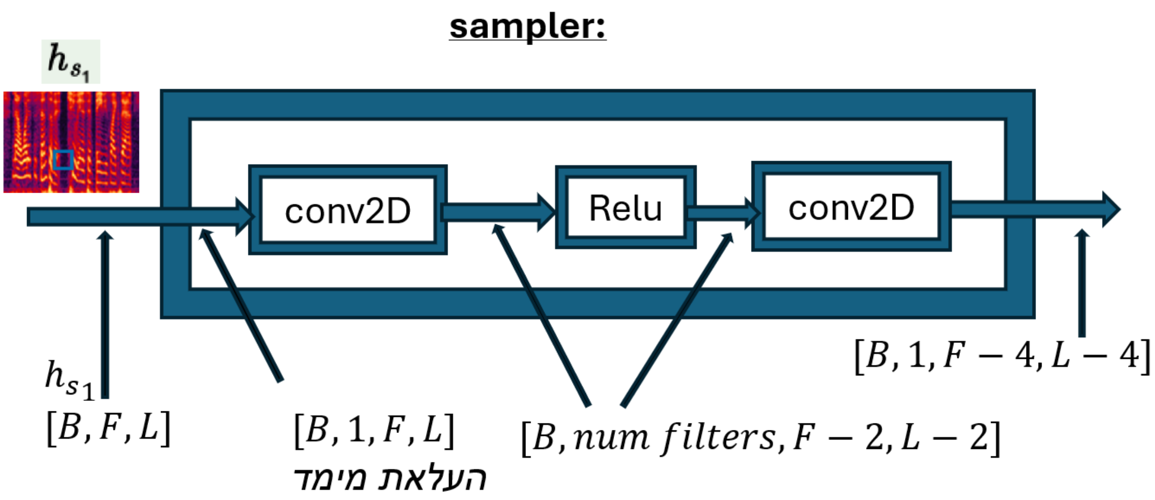
אם כן הרשת מקבלת כקלט את הכניסות: $h_{s_1}, h_{\hat{s}_1}, h_{\hat{n}} \in R^{B\times F\times L}$

כל אחד מהטנסורים עובר דרך רכיב ה *sampler* ורכיב ה *reshaper* כפי שנראה באיור:




כעת נפרט בפירוט על המעבר ב *sampler* וב *reshaper* התמונות למטה מתוארות עבור מעברו של: $h_{s_1} \in R^{B\times F\times L}$ , אולם המעבר כמובן יהיה זהה עבור כל אחד מהכניסות האחרות שמקבל רשת ה*PCL*.

נסתכל על המעבר ב*SAMPLER*:

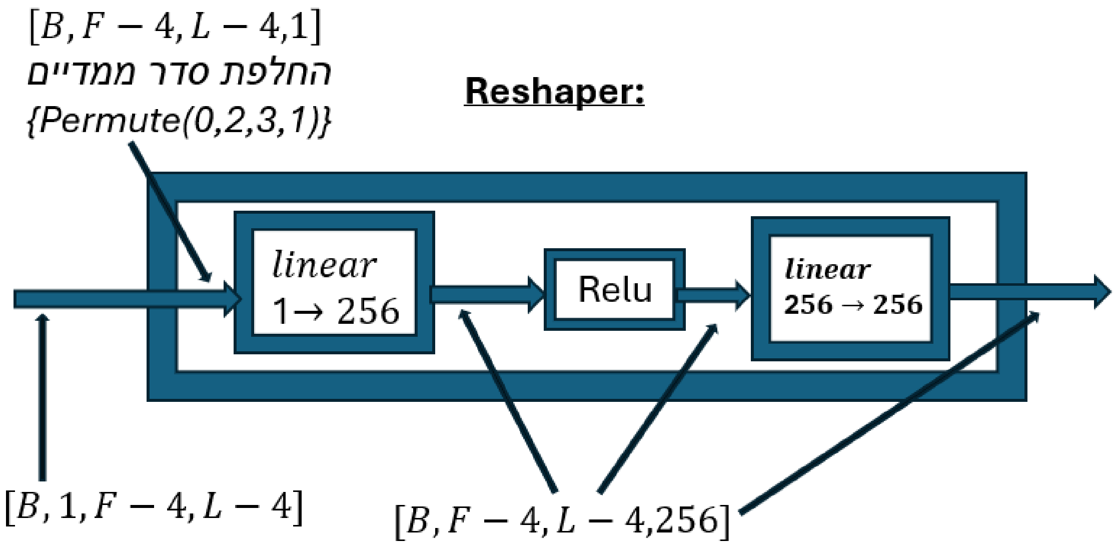



באיור מודגם מעבר של  עובר קונבולוציה דו מימדית עבור *input channels = 1* עבור 32 פילטרים שונים, וגודל חלון 3 על 3 . לאחר מכן הוא עובר דרך פונקציית אקטיבציה לא  ליניארית *ReLU*

ולאחר מכן עוד קונבולצייה דו מימדית עבור,*input channels = 32*  עבור פילטר בודד וגודל חלון 3 על 3.

בסך הכל מוצא ה *sampler* הוא  $[B, 1, F-4, L-4]$ .

כעת נסתכל על ה *reshaper* :




ה reshaper לוקח את הפלט של ה *sampler* כקלט אליו.
הוא בנוי מ 3 שכבות:
1) שכבת fully connected
2) שכבת פונקציית ReLU
3) שכבת fully connected נוספת

## PCL loss

כדי להסביר כיצד ה *Loss PCL* עובד, **נתעלם בהסבר מימד 0 (מימד ה*BATCH*)** ניתן כמובן להתעלם ממנו בהסבר כי הוא בסך הכל אומר שהפעלות מתבצעות במקביל בצורה זהה עבור כל אחד מרכיבי ה*BATCH* ולכן כמובן נתעלם בהסבר ממימד ה*BATCH*

בסך הכל עבור הקלטים לרשת ה*PCL* : $h_{s_1}, h_{s_2}, h_{\hat{s}_1}, h_{\hat{s}_2}, h_{\hat{n}} \in R^{B\times F\times L}$

קיבלנו בהתאמה לאחר מעבר רכיב ה *sampler* וה *Reshaper*

את המוצאים הבאים: $P_{s1}, P_{s2}, Q_{\hat{s}_1}, Q_{\hat{s}_2}, N_{\hat{n}} \in R^{B\times F-4\times L-4\times 256}$

כאשר *P* הוא ה*positive* (ייצוג תלת מימדי של הדובר האמיתי) , *Q* הוא ה*query* (ייצוג תלת מימדי של הדובר המושערך) ו*N* הוא ה*negative* (ייצוג תלת מימדי של הרעש המשוערך).

מבחינה אינטואיטיבית אנו רוצים שעבור כל אחד מין הדוברים "שה*query* שלו יהיה כמה שיותר קרוב ל *positive* שלו ורחוק ככל הניתן ה*negative* שלו".

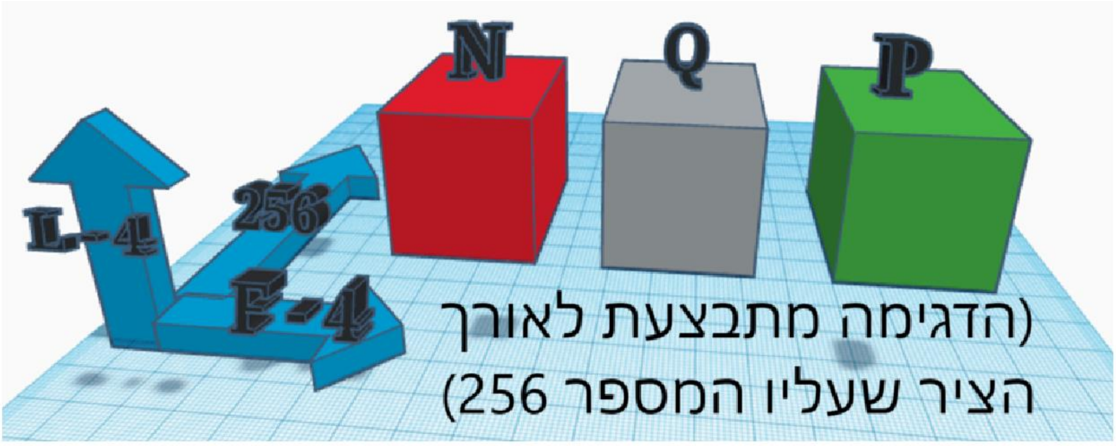
כדי לעשות זאת:

דגמנו את את ה *K query* פעמים במיקומים רנדומלים על פני המימד האחרון שלו (כל וקטור דגום אורכו 256)

דגמנו את את ה *K positive* פעמים על פני המימד האחרון שלו (כל וקטור דגום אורכו 256) באותם מיקומים כמו שדגמנו את *query*.

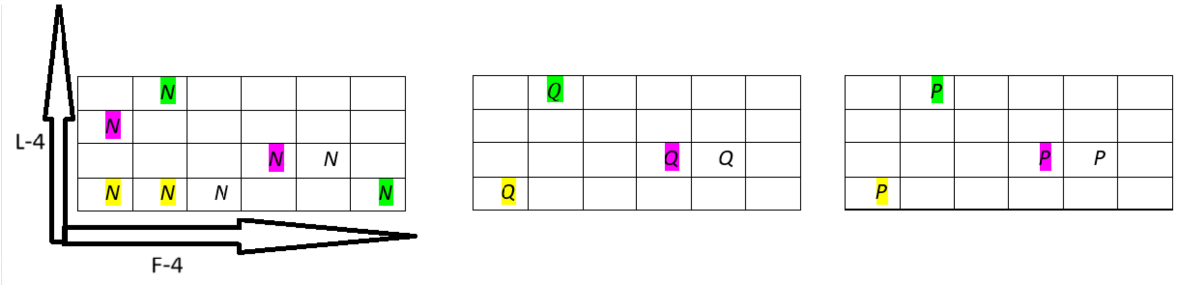
דגמנו את ה*negative* $k \cdot m$ פעמים , (כל וקטור דגום אורכו 256) כאשר מיקום האינדקסים הינו זהה למיקום הדגימות *K query* פעמים, ובשאר המקרים הוא במיקום שונה (וכמובן נגדם עבור מיקומים רנדומלים).

ניתן לראות המחשה לדגימות שביצענו באיור באיור (עבור *Q,P,N* של **אחד מ2n** הדוברים , כמובן יש לבצע את התהליך הנ"ל גם עבור הדובר השני).




ומיקומי הוקטורים (הנדגמים (כתלות במיקום הצירים *"F-4"* ו"*L*-4" . הם כפי שמופיע באיור:

למשל עבור ערכי $k = 4, m = 2$ , כאשר *Q,P,N* מייצגים את המיקום עבורו דגמנו את הוקטור ב*query* , *positive* וה*negative* בהתאמה.)




בסך הכל לאחר ביצוע הדגימה מקבלים את הוקטורים הבאים: (שמיקום הדגימה שלהן הינו באופן שהסברנו ואורכו של כל אחד מהם 256).

$$\left\{r_Q^{i,t} \mid i \in [1,k] \;\; t \in [1,2]\right\} \;\; \left\{r_P^{i,t} \mid i \in [1,k] \;\; t \in [1,2]\right\}$$

$$\left\{r_N^{j,t} \mid j \in [1,M] \;\; t \in [1,2]\right\}$$

וה *Loss PCL* מחושב על ידי הנוסחה הבאה:

$$\boldsymbol{loss}_{\text{pcl}} = \frac{1}{2}\sum_{t=1}^{2}\sum_{i=1}^{k} -\ln\left(\frac{e^{\frac{r_Q^{i,t} * r_P^{i,t}}{\tau}}}{e^{\frac{r_Q^{i,t} * r_P^{i,t}}{\tau}} + \sum_{j=1}^{M} e^{\frac{\left(r_Q^{i,t} * r_N^{j,t}\right)}{\tau}}}\right)$$

כאשר $*$ מייצג *Cosine similarity* המוגדר לפי: (ראה הרחבה בעמוד: 37).

ה*Loss PCL* מחושב עבור כל דובר (*t=1,2* )

כעת ניתן הסבר לנוסחת *Loss PCL* , הנוסחה הנ"ל היא למעשה *Cross-entropy loss* (ראה הרחבה בעמוד: 7).

כידוע *Cross-entropy loss* מוגדרת בצורה הבאה:

$y_t \in [0,1]$ מייצג את ה*LABEL* (ה*TAREGETS* ) האפשריים

ו $p(y_t|x_t;w)$ מייצג את ההסתברות שהמודל יכריע $y_t$ בהינתן המשקולות של הרשת, וקלט כלשהו $x_t$.

אצלינו הקלט $x_t$ הוא אחד מן הערכים:

$$x_t \in \left\{\frac{r_Q^{i,t} * r_P^{i,t}}{\tau} \;\; , \frac{\left(r_Q^{i,t} * r_N^{j,t}\right)}{\tau} \mid i \in [1,k] \;\; t \in [1,2], j \in [1,M]\right\}$$

כלומר $x_t$ מתקבל מביצוע של *Cosine similarity* בין 2 וקטורים כאשר וקטור אחד תמיד יהיה מה*query* והוקטור השני יהיה תמיד מה *positive* או ה *negative*.

כאשר נבצע *Cosine similarity* בין *query* ל*positive* התוצאה האידאלית תהיה 1 וזאת מאחר שמטרתנו כפי שציינו לפני כן היא שה *query* וה *positive* יהיו דומים ככל הניתן.

כאשר נבצע *Cosine similarity* בין *query* ל*negative* התוצאה האידאלית תהיה 0 וזאת מאחר שמטרתנו כפי שציינו לפני כן היא שה *query* וה *negative* יהיו שונים ככל הניתן.

כמובן שבפועל הקלט $x_t$ בסבירות גבוהה לא יהיה 1 או 0 אלא איבר בטווח בין מינוס 1 ל1 .

אבל אנו יודעים מה*TARGET* הרצוי 0 או 1 אליו נרצה להגיע).

כעת נרצה לתת הסתברות עבור כל אחד מהאיברים שבסט הבא:

$$x_t \in \left\{ \frac{r_Q^{i,t} * r_P^{i,t}}{\tau} \;, \frac{r_Q^{i,t} * r_N^{j,t}}{\tau} \mid i \in [1,k] \;\; t \in [1,2], j \in [1,M] \right\}$$

ואת ההסתברות $p(y_t|x_t;w)$ נגדיר על ידי הפעלת פונקציית הsoft max באופן הבא:

עבור:

$$x_t \in \left\{ \frac{r_Q^{i,t} * r_P^{i,t}}{\tau} \mid i \in [1,k] \;\; , j \in [1,M], t \in [1,2] \right\}$$

$$p(y_t|x_t;w) \in \left\{ \frac{\exp\left\{\frac{r_Q^{i,t} * r_P^{i,t}}{\tau}\right\}}{\exp\left\{\frac{r_Q^{i,t} * r_P^{i,t}}{\tau}\right\} + \sum_{j=1}^{M} \exp\left\{\frac{r_Q^{i,t} * r_N^{j,t}}{\tau}\right\}} \right\}$$

עבור

$$x_t \in \left\{ \frac{r_Q^{i,t} * r_N^{j,t}}{\tau} \mid i \in [1,k] \;\; , j \in [1,M], t \in [1,2] \right\}$$

$$p(y_t|x_t;w) \in \left\{ \frac{\exp\left\{\frac{r_Q^{i,t} * r_N^{j,t}}{\tau}\right\}}{\exp\left\{\frac{r_Q^{i,t} * r_P^{i,t}}{\tau}\right\} + \sum_{j=1}^{M} \exp\left\{\frac{r_Q^{i,t} * r_N^{j,t}}{\tau}\right\}} \mid i \in [1,k] \;\; , t \in [1,2] \right\}$$

ה- *Cross-entropy loss* מוגדר לפי:

$$\boldsymbol{loss}_{\mathrm{pcl}} = \frac{1}{2} \sum_{t=1}^{2} \sum_{i=1}^{k} -\ln\big(p(y_t|x_t;w)\big) \cdot y_t$$

וזאת עבור:

$$x_t \in \left\{ \frac{r_Q^{i,t} * r_P^{i,t}}{\tau} \;, \frac{r_Q^{i,t} * r_N^{j,t}}{\tau} \mid i \in [1,k] \;\; t \in [1,2], j \in [1,M] \right\}$$

**ניתן לכתוב ביטוי קומפקטי יותר** כי בחלק מהמקרים הסכימה מתאפסת (עבור $y_t = 0$ )

ונקבל:

$$\boldsymbol{loss}_{\mathrm{pcl}} = \tfrac{1}{2} \textstyle\sum_{t=1}^{2} \sum_{i=1}^{k} -\ln\big(p(y_t|x_t;w)\big)$$

וזאת עבור

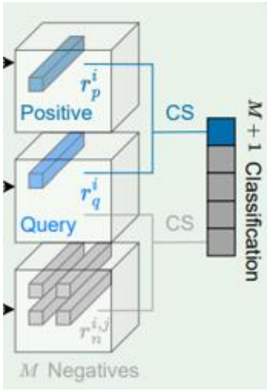
$$x_t \in \left\{\frac{r_Q^{i,t} * r_P^{i,t}}{\tau} \mid i \in [1,k] \;\; , j \in [1,M], t \in [1,2]\right\}$$

אם נציב את הביטוי שהגדרנו עבור ההסתברות נקבל בדיוק את ה *LOSS* שה*PCL* מנסה למזער.

כלומר את

$$\boldsymbol{loss}_{\mathrm{pcl}} = \frac{1}{2}\sum_{t=1}^{2}\sum_{i=1}^{k} -\ln\left(\frac{e^{\frac{r_Q^{i,t} * r_P^{i,t}}{\tau}}}{e^{\frac{r_Q^{i,t} * r_P^{i,t}}{\tau}} + \sum_{j=1}^{M} e^{\frac{\left(r_Q^{i,t} * r_N^{j,t}\right)}{\tau}}}\right)$$

כלומר כפי שניתן לראות מהנוסחה ,כדי לחשב את ה LOSS PCL עבור 2 דוברים אנו נדרשים עבור כל דובר בנפרד לחשב M+1 מכפלות של Cosine Similarity, ויש לבצע זאת K פעמים ( כמספר הוקטורים אנו מעוניינים לדגום מה Query.) ניתן לראות המחשה לדבר זה באיור:



## תוצאות:

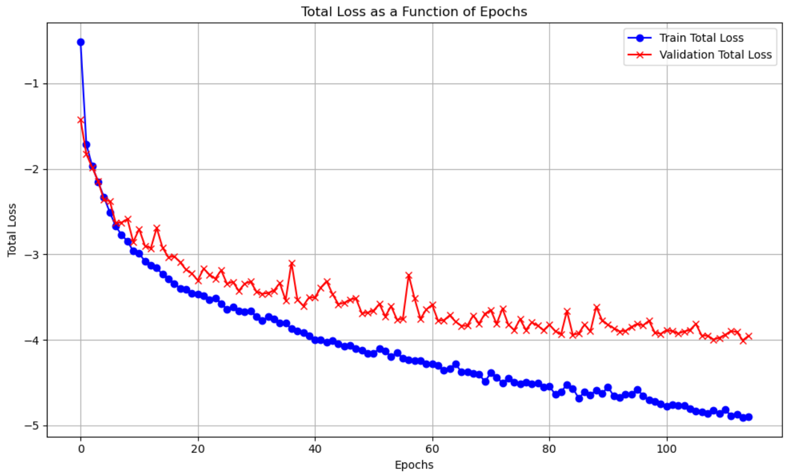
פונקציית ההפסד המשוקללת:

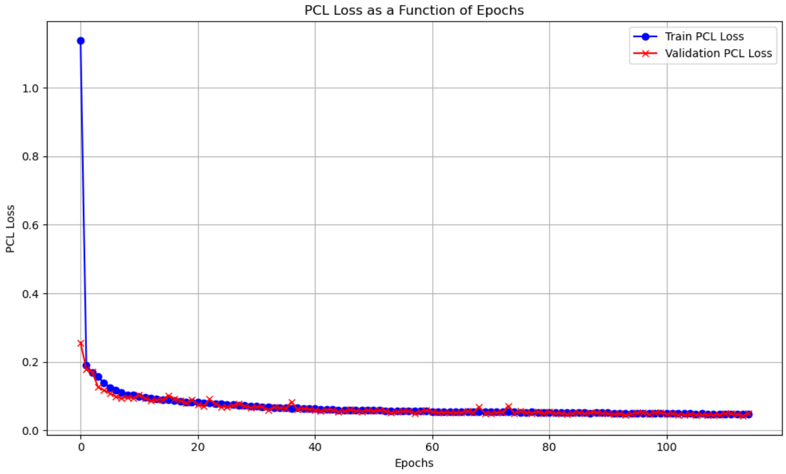
$L_{total} = L_{si-sdr} + \lambda \cdot L_{pcl}$ כאשר $\lambda = 1$

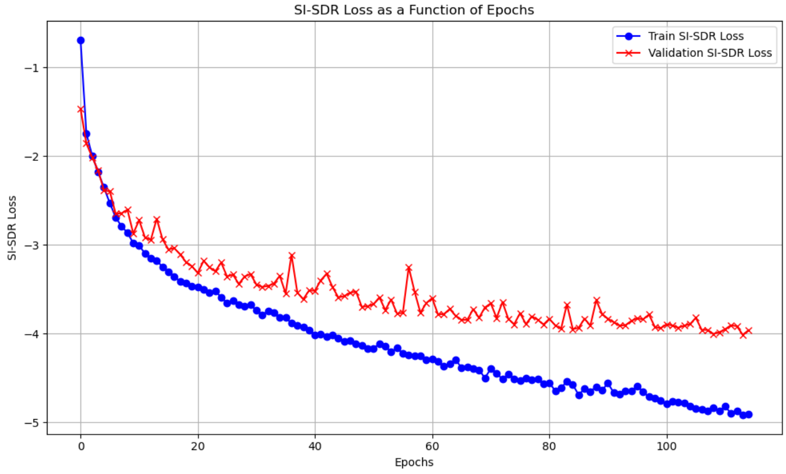
פונקציית ההפסד המרכזית הינה $L_{si-sdr}$ מכיוון שכפי שהסתברנו לפני כן היא מראה לנו במישור הזמן ההקלטה המשוערכת דומה להקלטה האמיתית (במובן שהסברנו).

תפקיד ה $L_{pcl}$ הוא לסייע להורדת *loss si-sdr*.

הרצנו 115 אפוקים, להלן ערכי ה *loss* השונים עבור *Training loss* וה*validation loss*



- הרצנו על 115 *epochs*
- ניתן לראות כי יש תהליך למידה, ה *loss* של 2 המודלים יורדים.
- כפי שניתן לראות, אין *overfitting*.
- $total\ loss = loss_{sisdr} + \lambda * loss_{pcl}$
- עבור מודל *PCL*, ניתן לראות שאין הפרש בין ה *train* ל *validation* ולכן ה *PCL* עושה הכללה בצורה טובה.
- אם עושים יותר מדי *epochs* אז יש סכנה ל *overfitting*.

## סיכום:

- בנינו מודל של למידה עמוקה שמבצע את משימת הפרדת דוברים בסביבה מורעשת
- אימנו את המודל ושמרנו את הפרמטרים שלו.
- נתקלנו בבעיות שהמודל לא למד, דיבגנו את הקוד וניסינו לפתור את כל הבעיות
- ככל שנביא יותר דוגמאות למודל כך יש לו יכולת יותר גדולה ללמוד ולהביא תוצאות טובות יותר
- מטרת המאמר הייתה לשפר יכולת ההפרדה של מערכת ה masking net ע"י שימוש בטכניקת Patch wise contrastive learning ובהתייחסות אל הרעש כעוד פלט.

## נספחים:

```python
class pcl_Model(nn.Module):
    def __init__(self):
        super(pcl_Model, self).__init__()
        self.encoder_num_filters = Freq
        self.sampler_num_filter = Freq

        # Define encoder layers
        #self.encoder_conv1d = nn.Conv1d(1, self.encoder_num_filters, kernel_size=H)
        self.enc_dim = Freq
        self.win = int(16000*2/1000)
        self.stride = self.win // 2
        self.encoder_conv1d = nn.Conv1d(1, self.enc_dim, self.win, bias=False, stride=self.stride)
        self.encoder_relu = nn.ReLU()
        print('pcl parameters',self.enc_dim,self.win,self.stride)


        # Define sampler layers
        self.sampler_conv2d_1 = nn.Conv2d(1, self.sampler_num_filter , kernel_size=(H, H))
        self.sampler_conv2d_2 = nn.Conv2d(self.sampler_num_filter,1, kernel_size=(H, H))

        # Define reshaper layers
        self.reshaper_linear1 = nn.Linear(1, 256)
        self.reshaper_linear2 = nn.Linear(256, 256)
        self.reshaper_relu = nn.ReLU()
```

**קוד של ה encoder**

```python
def encoder(self, x):
    x = x.float()
    x = self.encoder_conv1d(x)
    encoded = self.encoder_relu(x)
    print('encoded',encoded.shape)
    return encoded
```

**קוד של ה sampler**

```python
def sampler(self, encoded):
    x = encoded  # 4 32 607
    x = x.float()
    if x.dim() == 3:
        x = x.unsqueeze(1)  # [4, 1, 32, 607]
    x = self.sampler_conv2d_1(x).to(device) # 4 32 30 605
    x = self.encoder_relu(x)    # 4 32 30 605
    sampled = self.sampler_conv2d_2(x)    # 4 1 28 603
    print('sampler',sampled.shape)
    return sampled # 4 1 28 603
```

**:Reshaper**

**קוד של ה reshaper**

```python
def reshaper(self, sampled):     # 4 1 28 603
    x = sampled.permute(0, 2, 3, 1) # 4 28 603 1
    x = x.float()
    x = self.reshaper_linear1(x)    #4 28 603 256
    x = self.reshaper_relu(x)     #4 28 603 256
    reshaped = self.reshaper_linear2(x)  #4 28 603 256
    print('reshaped',reshaped.shape)
    return reshaped
```

```python
class pcl_loss(nn.Module):
    def __init__(self, k,m, t):
        super(pcl_loss, self).__init__()
        self.k = k
        self.t = t
        self.m = m
        self.cel = nn.CrossEntropyLoss()

    def select_k_random_indices(self, Query, positive, noise):
        print('query',Query.size())
        print('positive',positive.size())
        print('noise',noise.size())
        b, f, t, e = Query.size()

        vec_reshape_Q = Query.reshape(b, -1, e)
        vec_reshape_P = positive.reshape(b, -1, e)
        vec_reshape_N = noise.reshape(b, -1, e)

        indices = torch.randperm(f * t)[:self.k]
        indices_noise = torch.randperm(f*t)[:self.k*self.m]
        vec_sampled_Q = vec_reshape_Q[:, indices, :]
        vec_sampled_P = vec_reshape_P[:, indices, :]
        vec_sampled_N = vec_reshape_N[:, indices_noise, :]
        return vec_sampled_Q, vec_sampled_P, vec_sampled_N
```

קוד לשימוש ב Cosine Similarity:

```python
def cosine_similarity(self, Query, positive, noise,original_query):
    b, k, e = Query.size()
    QP_sim = F.cosine_similarity(Query, positive, dim=2)
    Query = Query.repeat(1, self.m, 1)  # Repeat each row m times
    QN_sim = F.cosine_similarity(Query, noise, dim=2)
    return QP_sim, QN_sim
```

קוד ל pcl loss:

```python
def forward(self, Query, positive, noise):
    vec_sampled_Q, vec_sampled_P, vec_sampled_N = self.select_k_random_indices(Query, positive, noise)
    QP_sim, QN_sim = self.cosine_similarity(vec_sampled_Q, vec_sampled_P, vec_sampled_N,Query)
    QP_sim = QP_sim / self.t
    QN_sim = QN_sim / self.t
    QP_sim = QP_sim.unsqueeze(1)
    QN_sim = QN_sim.unsqueeze(1)
    #QN_sim = QN_sim.view(QN_sim.size(0), self.k, self.k)
    y_hat = torch.cat((QP_sim, QN_sim), dim=-1)
    #y_hat = torch.rand_like(y_hat)
    zeros_tensor = torch.zeros_like(QN_sim)
    ones_tensor = torch.ones_like(QP_sim)
    y = torch.cat((ones_tensor, zeros_tensor), dim=-1)
    #y = torch.rand_like(y_hat)
    pcl_loss = self.cel(y_hat[0,0,:], y[0,0,:])
    return pcl_loss
```