הפקולטה להנדסה

המעבדה לעיבוד אותות

Neural Network-based Universal Sound Selector

בורר שמע מבוסס רשת נוירונים

יובל סרף

שרה איליסייב

פרויקט שנה ד' לקראת תואר ראשון בהנדסה

מנחה: יוחאי ימיני

מנחה אקדמי: פרופ' שרון גנות

אוקטובר 2021

תוכן עניינים

הבעיה שאנחנו פותרים

מבוא ללמידה עמוקה
Deep neural networks and artificial neural networks
Perceptron
תהליך ה - forward pass ל - perceptron
מ - Perceptron ל - Deep neural networks
Activation function
Sigmoid
Softmax
ReLu
Gradient descent and backpropagation
אלגוריתמים שונים ל - gradient ascent
Gradient clipping
LR scheduler
Adam
שכבות שונות ב - deep neural network
Linear
convLayer
Batch normalization
1D d-Conv
MaskGenerator
Conv-TasNet
Hyperparameters
Training algorithm
Learning rate
Batch
Epoch
Neural network structure
אימון רשת
אתחול פרמטרים
פונקציית ה - loss
חלוקת ה - dataset

הבנת הבעיה
השיטה
תקציר רצף המערכת
מבנה ה - data
פונקציית ה - Loss
צורות למידה ואבלואציה שונות
O vector
הרשת המותאמת שלנו
יישום השיטה
ארכיטקטורה
הקבצים הרלוונטים
הרכבת ה - datasets
מימוש ה - Gradient descent and backpropagation
בניית הרשת
הרצה
מהלכים השונים בפרויקט
תוצאות
Train loss
אבלואציה וקבצי שמע
הצעות לשיפור
מסקנות
Bibliography

- ## הבעיה שאנחנו פותרים

הסביבה שלנו ברוב המקרים מכילה מספר עצמים כמו אנשים מדברים, כלב נובח, צפירה של מכונית ועוד. חיי היומיום שלנו יכולים להשתפר מאוד אם נוכל לפתח מכשירי שמיעה אשר מבדילים בין אירועי שמע שונים ובוחרים את אלה שאנו רוצים להקשיב להם בהתאם למצב.
לפיכך, אנו מגדירים שתי בעיות:

1. בחירת סיגנל ספציפי והסרת שאר הסיגנלים.
2. הסרת סיגנל ספציפי ובחירה בשאר הסיגנלים.

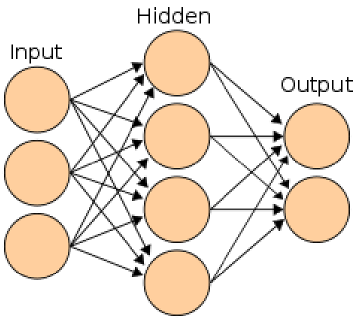
הבעיה בפרויקט זה שבאנו לפתור וליישם היא זיהוי וסיווג אירועי שמע שונים באמצעות ערוץ מיקרופון בודד מרצועת אודיו אשר מערבבת כמה מקורות סאונד יחד. גישתנו מסתמכת על רשת ניורונים הנקראת convtas-net אשר מזהה את אירוע השמע שאנו רוצים להפריד מרצועת האודיו. כלומר מנקה את כל רעשי הרקע וכך אנו מבינים באופן ברור את אירוע השמע שלנו אנו רוצים להקשיב.

• מבוא ללמידה עמוקה

בחלק זה נסביר מושגים בסיסים בהם נעשה שימוש לאורך הפרויקט :

○ Deep neural networks and artificial neural networks

רשתות נוירונים (artificial neural networks) הן מערכות למידה מפוקחות אשר בנויות מהרבה נוירונים המאורגנים בשכבות מחוברות זה לזה כאשר כל נוירון מקבל החלטות פשוטות ומעביר את אותן לנוירונים הבאים .הרשת מכילה בדרך כלל מספר רב של יחידות מידע, קלט ופלט הקשורות זו לזו, קשרים שלעיתים קרובות עוברים דרך יחידות מידע "חבויות" (Hidden Layer). יחידת הקלט מקבלת את הtraining data של המודל שאותו נרצה לאמן ויחידת הפלט אשר מייצרת חיזויים. השימוש ברשתות נוירונים נפוץ בעיקר במדעים קוגניטיביים, ובמערכות תוכנה שונות - בהן מערכות רבות של אינטליגנציה מלאכותית המבצעות משימות מגוונות - מערכת זיהוי דיבור, זיהוי תווים , זיהוי פנים ,זיהוי כתב יד, חיזוי שוק ההון, זיהוי תמונה, ניתוח טקסט ועוד.
ברשת נוירונים "רדודה" קיימות רק שלוש שכבות של נוירונים, שכבת קלט, שכבה אחת נסתרת של נוירונים ושכבת פלט:



ברשת נוירונים עמוקה (DNN) קיימות שכבות רבות, שכבת קלט, לפחות שתי שכבות נסתרות של נוירונים ושכבת פלט.
רשתות נוירונים רדודות ועמוקות מסוגלות להתמודד עם בעיות מורכבות אך ככל שמספר השכבות הנסתרות עולה, כלומר הרשת יותר עמוקה, נקבל דיוק גדול יותר.

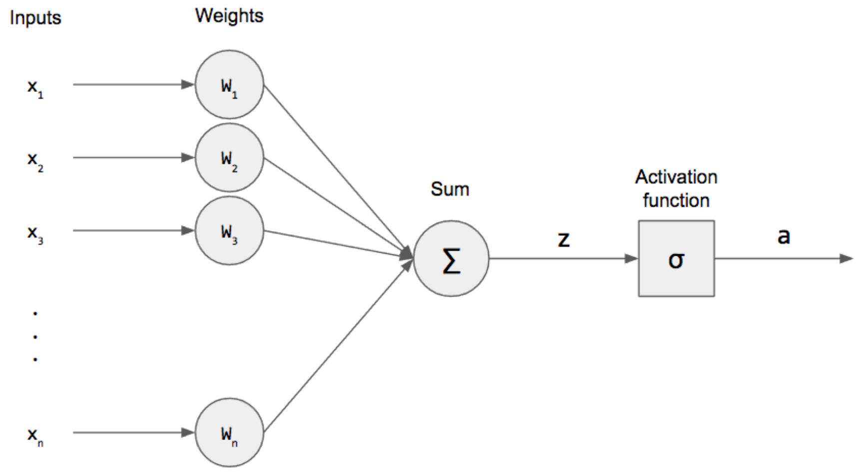
○ Perceptron

קולטן (perceptron) הוא אלגוריתם ממשפחת האלגוריתמים הלומדים. זהו אלגוריתם למידה "און ליין", משמע הוא לומד תוך כדי ריצה מדגימות שאבחן בזמן פעולתו.
הפרספטרון הוא אלגוריתם סיווג, כלומר מטרתו היא להבדיל בין סוגים שונים של דגימות אותן הוא מקבל. לדוגמה, להבחין בין גברים ונשים על פי מידת הנעל ואורך השיער.
האלגוריתם מחקה את התנהגות הנוירונים (מדמה תא עצב) במערכת העצבים, ועל-כן הוא נחשב לאחת הדוגמאות לרשת נוירונים.

○ תהליך ה - forward pass ל- perceptron

התהליך מתבצע כך:

- כניסות מוזנות לרשת אשר מוכפלות במשקולות מתאימות והן נסכמות.
- מוסיפים קבוע bias אשר מזיז לכיוון מסויים את ערכו של הפלט של כל פרספטרון.
- הסכום נכנס לactivation function (כגון: פונקציית מדרגה).
- הפלט שיוצא מהactivation function הוא המוצא לפרספטרון.



מ - Perceptron ל - Deep neural networks

לאחר שהבנו מהו הפרספטרון, נסביר עוד כמה דברים אשר מרכיבים אותה על מנת להבין באופן יותר טוב את מבנה רשת הנוירונים העמוקה (DNN).

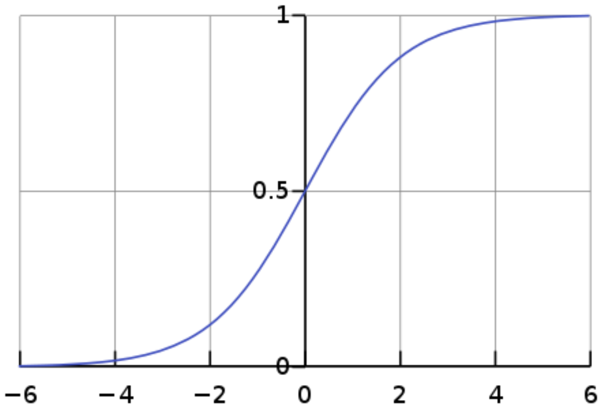
Activation function

כפי שציינו, הרשת משתמשת בactivation function, פונקציות הלוקחות את הקלט של כל נוירון או פרספטרון וקובעות את הפלט שלו. פונקציות קלאסיות המשמשות אותנו ברשתות נוירונים הן פונקציות מדרגה sigmoid, softmax, ReLu ועוד.

נפרט על סוגי הactivation functions ותפקידם:

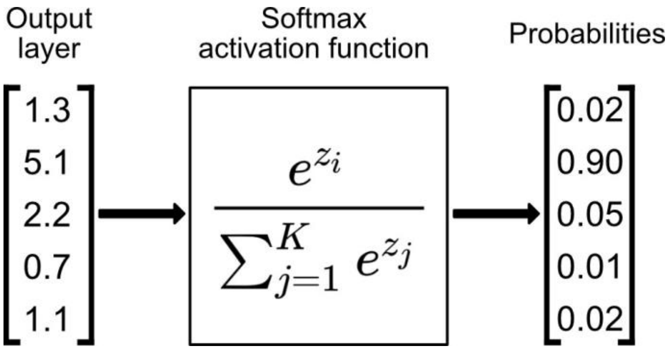
Sigmoid

פונקציית סיגמואיד היא פונקציה מתמטית, המוצא שלה יהיה בין 0 ל1. לעיתים קרובות הפונקציה מתייחסת למקרה פרטי של הפונקציה הלוגיסטית, כפי שניתן לראות באיור מטה ומתוארת על ידי הנוסחה הבאה $S(x) = \frac{1}{1+e^{-x}}$. פונקציות סיגמואיד מוגדרות על המספרים הממשיים, והן עולות מונוטונית. יתרונותיה של הפונקציה היא אי-הלינאריות שלה וטווח ערכי המוצא כפי שציינו שלא כמו בפונקציה לינארית בה הטווח אין סופי. החיסרון שלה הוא כאשר ערכי המוצא גבוהים\נמוכים נקבל נגזרת השואפת ל0 לכן רשת הלומדת על ידי הנגזרות תלמד בצורה מאוד איטית, לבעיה זו קוראים vanishing gradient.



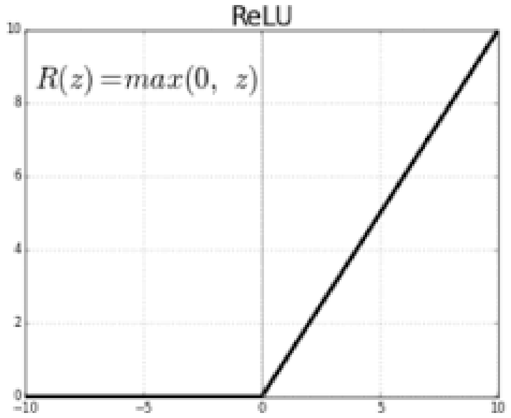
- **Softmax**

פונקציית הsoftmax לוקחת כקלט וקטור של K מספרים אמיתיים, ומנרמלת אותו להתפלגות הסתברות המורכבת מK הסתברויות ביחס לאקספונט של מספרי הקלט. לפני הsoftmax, חלק מהאיברים בוקטור יכולים להיות שליליים או גדולים מאחד והסכום שלהם לא יהיה 1. אך לאחר הsoftmax, כל אחד מהאיברים יהיה בעל ערך בטווח [1,0] וסכום כל האיברים בוקטור יהיה 1. בצורה הזו ניתן להתייחס לאיברים בוקטור כהסתברויות מכיוון שעונים על חוק ההסתברות השלמה.



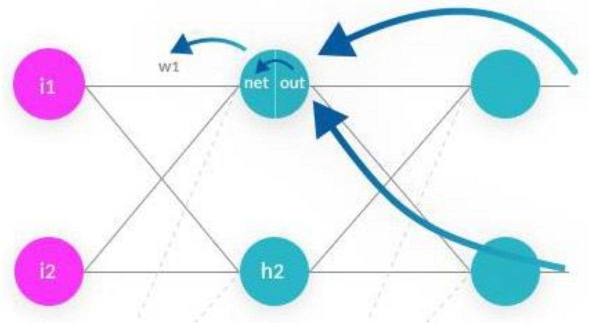
- **ReLu**

פונקציית rectified linear unit (ReLu) היא activation function הנפוצה ביותר במודלים של למידה עמוקה. הפונקציה מחזירה 0 אם היא מקבלת קלט שלילי כלשהו, אך עבור כל ערך חיובי x היא מחזירה ערך זה בחזרה. כפי שניתן לראות באיור למטה, הפונקציה לינארית בצד החיובי והטווח בה אין סופי לכן ככל הנראה קיימת הבעיה שדיברנו עליה לעיל אך הפונקציה כולה אינה לינארית וניתן להשתמש בה ברשת לאורך השכבות ברצף. כמו כן, הפונקציה הזו גורמת ליעילות ברשת כיוון שהיא מתחילה מהערך 0 וכל הציר השלילי מבוטל, כלומר חלק מהנוירונים לא פעילים עקב הגרדיאנט אשר מתאפס. נוירונים אשר נכנסו לאזור הציר השלילי של הפונקציה לא יגיבו לשינוי השגיאה ולכן לא יתרמו להקטנתה, זהו מצב אשר נקרא dying ReLU.

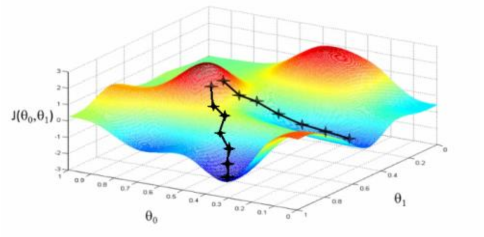


○ Gradient descent and backpropagation

לאחר כל השלבים שדיברנו עליהם לעיל, נרצה ליצור חיזוי ראשוני ולבדוק עד כמה הוא רחוק מהערך המקורי שרצינו שהרשת תחזה. נבדוק זאת על ידי פונקציית השגיאה שאותה ניתן למזער על ידי אלגוריתמים שונים. לדוגמא, נגזור את פונקציית השגיאה ונמצא ערכים למשקולות כך שנקבל שגיאה קטנה ככל האפשר.
במצב שלנו, כאשר אנו עובדים עם רשת נוירונים גדולה עם הרבה משקולות, נשתמש באלגוריתם הbackpropagation בעל יעילות חישובית טובה יותר. האלגוריתם מעביר את קלט הלימוד במורד רשת עם משקולות אקראיות ומשווה את הפלט המתקבל לפלט הרצוי בכך שהוא מחשב את הטעות עבור כל נוירון פלט.

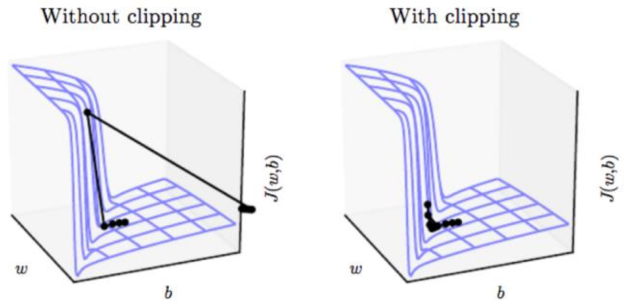


נסביר על המונח gradient descent, שיטת אופטימיזציה איטרטיבית מסדר ראשון למציאת מינימום מקומי של פונקציה. אנחנו מחשבים את כיוון הגרדיאנט מהנקודה שבה אנחנו נמצאים (משקלי הרשת) בהתאם לפונקציית ההפסד, ומתקדמים מהנקודה הנוכחית בצעד קטן בכיוון הנגדי לכיוון הגרדיאנט. הגרדיאנט הוא הכיוון בו ערך הפונקציה עולה בצורה מקסימלית, לכן הכיוון הנגדי הוא הכיוון בו הפונקציה הכי יורדת. אנו מחשבים את הגרדיאנט של פונקציית ההפסד לפי הנתונים באימון, עושים צעד מערך המשקולות הנוכחיים בכיוון הנגדי לכיוון הגרדיאנט ומעדכנים את המשקלות לערך החדש כאשר גודל הצעד נקבע לפי הlearning rate.



- אלגוריתמים שונים ל - **gradient ascent**
  - **Gradient clipping**

Gradient clipping היא טכניקה המתמודדת עם גרדיאנטים מאוד גדולים. אם אנו מקבלים גרדיאנט מאוד גדול נשנה את גודלו על מנת שיהיה קטן. בצורה מתמטית אם $\|g\| \geq c$ אז $g \leftarrow c \cdot g/\|g\|$, כאשר g הוא הגרדיאנט, $\|g\|$ הוא נורמת הגרדיאנט וc הוא היפר-פרמטר. Gradient clipping מבטיח שלוקטור הגרדיאנט g יש נורמה שלא גדולה מהערך c. הדבר עוזר לgradient descent להתנהג בצורה הגיונית על אף שפונקציית הloss אינה סדירה.



ניתן לראות בתמונה, עם clipping הפרמטרים עושים צעד עצום ועוזבים את האזור "הטוב", אך ללא clipping גודל הצעד מוגבל והפרמטרים נשארים באזור "הטוב".

  - **LR scheduler**

Learning rate scheduler מתאים את קצב הלמידה במהלך האימון על ידי הפחתת ההיפרפרמטר learning rate בהתאם ללוח זמנים שהוגדר מראש. לוחות הזמנים נפוצים של קצב למידה כוללים החלשות מבוסס זמן, החלשות אקספוננציאלית והחלשות לפי צעדים שזוהי הדרך שלנו.

  - **Adam**

ברשת שלנו נשתמש באלגוריתם adam לאופטימיזציה עם learning rate של 0.0005 וב - gradient clipping. אלגוריתם adam

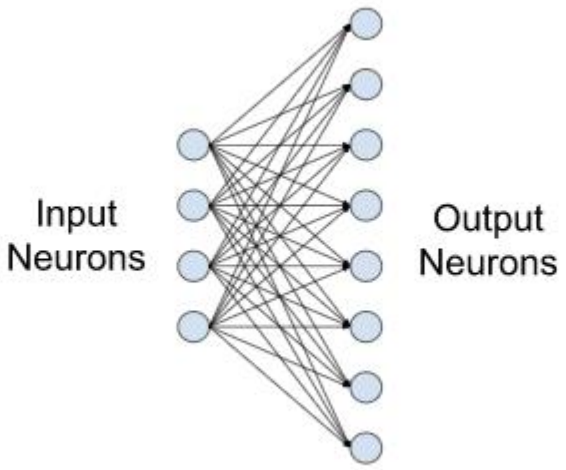
לאופטימיזציה הוא אלגוריתם שניתן להשתמש בו במקום הליך stochastic gradient descent הקלאסי לעדכון משקלי הרשת המבוסס על נתוני האימון. stochastic gradient descent שומרת על ערך קצב הלמידה (learning rate) יחיד לכל עדכוני המשקל וקצב הלמידה אינו משתנה במהלך האימון. קצב הלמידה נשמר ומותאם לכל משקל ברשת ככל שמתפתחת הלמידה.

אלגוריתם adam משלב שני יתרונות של stochastic gradient descent:

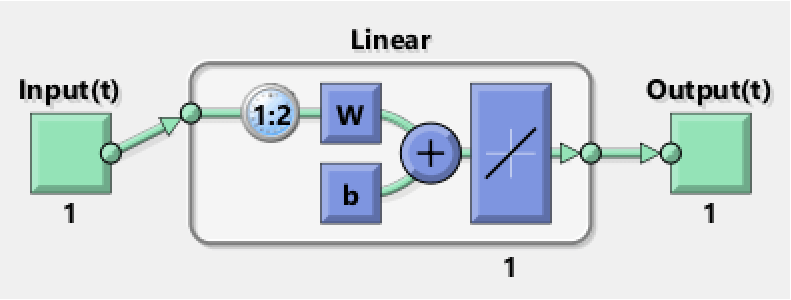
- Adaptive Gradient Algorithm
  שומר על פרמטר קצב הלמידה אשר משפר את הביצועים בבעיות sparse gradients.
- Root Mean Square Propagation
  שומר על שיעורי פרמטרי קצב הלמידה המותאמים בהתאם לממוצע המגניטודות האחרונות של הגרדיאנטים של המשקל. כלומר, האלגוריתם מסתדר היטב עם בעיות מקוונות ולא נייחות.

- שכבות שונות ב - **deep neural network**
  - **Linear**

שכבת Linear (או Fully connected) מחברת כל נויורון כניסה עם נוירון יציאה.
השכבות האחרונות לעיתים בהרבה מודלים הן שכבות Linear, המשלבות
הרבה כניסות לתוצאה סופית - למשל סיווג מידע למספר קטן של קבוצות.
הבעיה בכך שלפעמים הפעולה האריתמטית שהינו מבצע טיפה בזבזנית.
כך למשל נראית שכבת Linear עבור מוצא עם יותר מוצאים מאשר כניסות



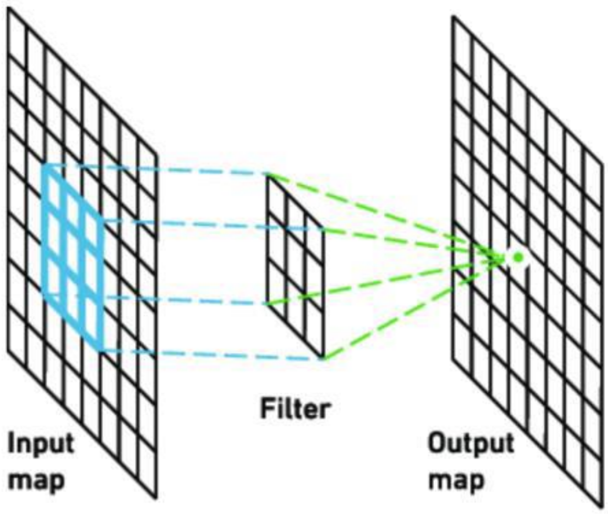
דרך נוספת להסתכל על זה באמצעות הנוסחה שזה מייצג (וקטורי):



כפי שניתן לראות גם ניתן לצרף קבוע למשוואה (אם רוצים בכך).

convLayer ■

קונבולוציה שימשה במשך זמן רב בדרך כלל בעיבוד תמונות לטשטוש וחידוד תמונות, אך גם לביצוע פעולות אחרות. CNNs אוכף דפוס קישוריות מקומי בין נוירונים של שכבות סמוכות. פרמטרי שכבת הקונבולוציה מורכבים מסט של פילטרים אשר ניתנים ללמידה. במהלך פעולת הforward pass, כפי שניתן לראות באיור מטה, מחליקים כל פילטר לגובה ולרוחב הנפח של הקלט ומבצעים מכפלה איבר באיבר של כל מסנן בכל מקום בקלט.activation map , סט פילטרים, נוצר ברגע שאנו מחלקים את המסנן לאורך ולגובה הנפח של הקלט אשר מתאר את תגובות הפילטר בכל מקום.



בכל פעם שהרשת רואה תכונה כלשהי היא לומדת את הפרמטרים של הפילטרים וכך יוצרת לכל שכבת קונבולוציה סט פילטרים מותאם. לאחר מכן, מחברים את כל הactivation maps לאורך כל עומק השכבות הללו ומקבלים מוצא מתאים.

Batch normalization

נורמליזציה היא כלי לעיבוד מקדים של נתונים המשמש להבאת הנתונים המספריים לקנה מידה משותף מבלי לעוות את צורתו. באופן כללי, כאשר אנו מכניסים את הנתונים למכונה או לאלגוריתם למידה עמוקה אנו נוטים לשנות את הערכים לסולם מאוזן. הסיבה לנורמליזציה היא בחלקה כדי להבטיח שהמודל שלנו יכול להיקבע באופן ראוי.

Batch normalizationזהו תהליך ההופך את הרשתות העצביות למהירות ויציבות יותר באמצעות הוספת שכבות נוספות ברשת עצבית עמוקה. השכבה החדשה מבצעת את סטנדרטיזציה ונורמליזציה על הקלט של שכבה המגיעה משכבה קודמת.

נורמליזציה היא תהליך הפיכת ממוצע הנתונים ל0 וסטיית התקן ל1. בשלב הראשון יש לנו את קלט הbatch משכבה h ונחשב את ממוצע הפעולות הנסתרות:

$\mu = \frac{1}{m}\sum h_i$, כאשר m הוא מספר הנוירונים בשכבה h.

לאחר מכן, נחשב את סטיית התקן של הפעולות הנסתרות: $\sigma = \left[\frac{1}{m}\sum (h_i - \mu)^2\right]^{1/2}$

לבסוף, קנה המידה והזזה החדשים חלים על הקלט, כאן נכנסים לתמונה שני פרמטרים של האלגוריתם γ ו- β . פרמטרים אלה משמשים לקנה מידה חדש (γ) והזזה (β) של הווקטור המכיל ערכים מהפעולות הקודמות.

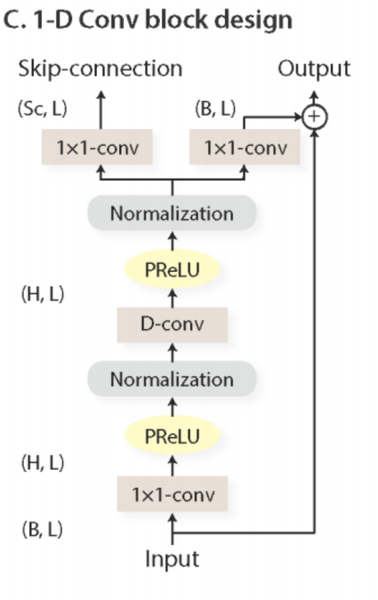
$$h_i = \gamma\, h_{i(norm)} + \beta$$

שני הפרמטרים הללו ללמידה כשלבסוף נקבל את הערכים האופטימליים של γ ו- β. מצב זה יאפשר את הנורמליזציה המדויקת של כל batch.

■ **1D d-Conv**

שכבה זו הינה אחד המרכיבים העיקריים ברשת Conv-TasNet, עליה נרחיב בהמשך.

היא נראית כך:



משמעות השם 1D d-Conv הוא one-dimensional dilated convolution block.

ניתן לראות כי יש 2 מוצאים לשכבה - דילוג (skip) ושארית (residual).

תכנון שכבה זו היא למעשה כדי שתוכל להתחבר לכאלו נוספים בהמשך (עוד על כך - ב MaskGenerator).

כדי להקטין את מספר הפרמטרים למעשה מבצעת שכבת ה d-Conv קונבולוציה על הכניסה, אך עם הרחבה (dilation) כפי שניתן לראות באיור הבא

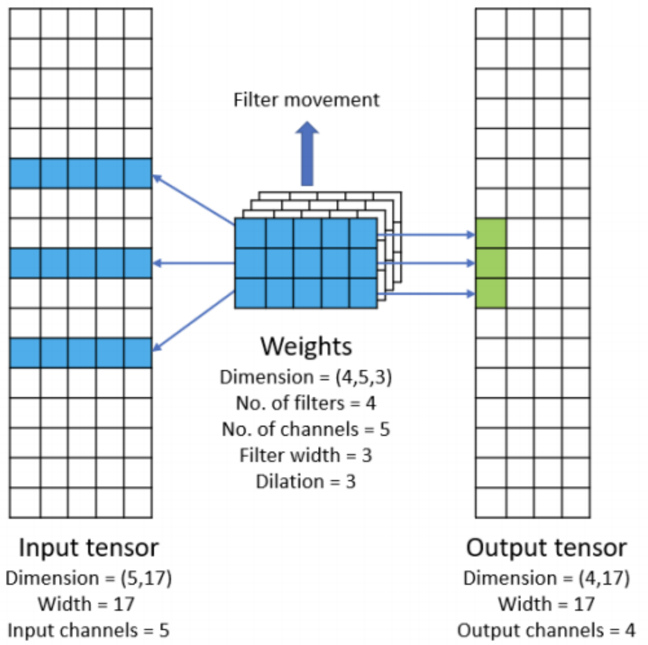


Figure 1: An example of the 1D dilated convolution layer with input channels (C) = 5, input width (W) = 17, number of filters (K) = 4, filter width (S) = 3, output width (Q) = 17, and dilation parameter (d) = 3.

לאחר מכן התוצאה עוברת ל - PReLU המכיל משקולת R המשומשת באופן הבא:

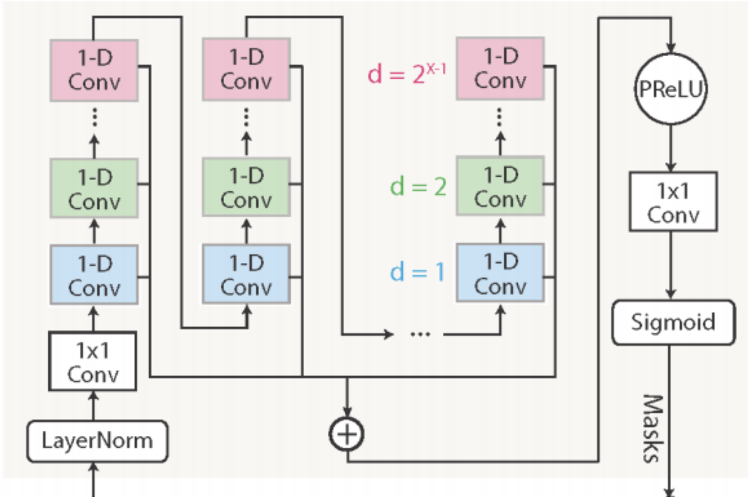
$$PReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{otherwise} \end{cases}$$

ולאחר מכן נרמול הנתונים לקראת יציאה מן השכבה.

שילוב של שימוש ב - d-Conv ורצף של שכבות מסוג זה, משיגות את האפקט של TCN - Temporal Convolutional Network בעלות נמוכה יותר של פרמטרים.

■ MaskGenerator

ציינו קודם לכן ש - Mask הינה חיבור של כמה d-Conv1 D יחדיו, וכעת נראה איור של איך דבר זה נראה



ניתן לראות ככל שעולים בשרשרת ה - d-Conv1 D אז פרמטר ההרחבה - d עולה אקספוננציאלית, וזאת כדי להבטיח יעילות לאורך הקשר לטווח ארוך של הפרדת שמע (כפי שרשת זו מתוכננת לעשות).
התוצר של ה - TCN מועבר לקונבולוציה של קרנל בגודל אחד (שגם נקרא pointwise convolution) כדי לשערך את המסכה.
ביחד עם ה - sigmoid - שהיא פונקציה לא לינארית, נקבל שיערוך ל - C וקטורים עבור C מקורות שמע שונים בהקלטה.

Conv-TasNet

ברשת Conv-TasNet, רשת הפרדת שמע מבוססת קונבולוציה בתחום הזמן. היא מבוססת למידה עמוקה על מנת לבצע הפרדת שמע מקצה לקצה. השיטה משתמשת במקודד ליניארי כדי ליצור ייצוג אופטילי של צורת גל הדיבור להפרדת כל אירוע שמע שונה. ההפרדה מושגת על ידי החלת סט של פונקציות משקלים (מסכות) לפלט המקודד. הפלט הופך לאחר מכן לצורת הגל על ידי מפענח ליניארי. המסכות משתמשות ברשת קונבולוציה זמנית (temporal convolutional network- TCN) שמורכב מבלוקי קונבולוציה של מימד אחד אשר מאפשר לרשת לעצב את יחסי התלות ארוכי הטווח של אות הדיבור תוך שמירה על גודל דגם קטן. רשת Conv-TasNet נותנת ביצועים טובים יותר משמעותית משאר שיטות המסכה הקודמות אשר מפרידות כמה אירועי שמע מקטע שמע. בנוסף, ישנם שני יתרונות גדולים לרשת מסוג זה. הרשת מתפקדת טוב כאשר מפרידים קולות מדו שיח, זאת ע״י בדיקת עיוות קול לא מוטית וכן ע״י בדיקת איכות ידנית. וכן גודל המודל של הרשת קטן יחסית מה שמאפשר את יישומה גם במערכת בה צריך את התוצאה באופן מיידי.

○ Hyperparameters

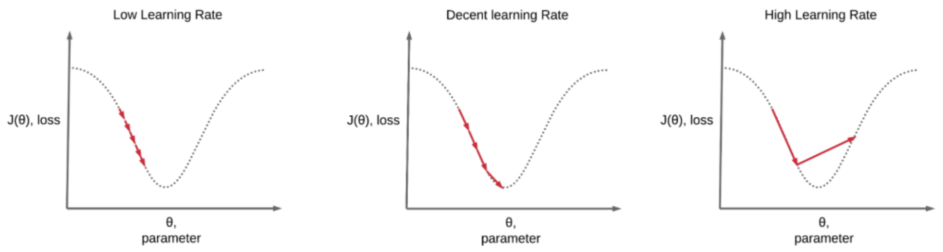
היפר-פרמטרים הם פרמטרים חיצוניים אשר נקבעים על ידי מפעיל הרשת.
הפרמטרים הללו מתחלקים ל2 קבוצות:

■ Training algorithm

היפר-פרמטרים השייכים לאימון האלגוריתם, כגון קצב למידה, epochs (מספר האיטרציות), גודל batch, מומנטום ועוד.
נרחיב על כמה היפר-פרמטרים:

- **Learning rate**

learning rate הוא אחד הפרמטרים החשובים ביותר לאימון הרשת.
הפרמטר ממזער את פונקצייה הloss של הרשת על ידי הגדלת עדכוני המשקל בצורה מדויקת כך שקצב הלמידה לא יהיה נמוך או גבוה מידי.



כפי שניתן לראות באיור לעיל, כאשר קצב הלמידה נמוך מידי נקבל אימונים איטיים עקב עדכוני משקל קטנים מאוד וכאשר קצב הלמידה גבוה מידי נקבל כי פונקצית הloss תתנהג בצורה משונה ולא כפי שאנו צריכים.

- **Batch**

הbatch אחראי על מהירות ודיוק הלמידה בצורת שקלול תמורות. כאשר הפרמטר יהיה שווה למספר קטן, הלמידה תהיה מהירה אך לא מדויקת. זאת עקב רעש המתווסף בתהליך העדכון המבוצע על ידי הגרדיאנט אשר מתבסס על dataset קטן. כאשר הפרמטר יהיה שווה למספר גדול, הלמידה תהיה איטית אך מדויקת עקב השונות הקטנה בזמן מיצוע

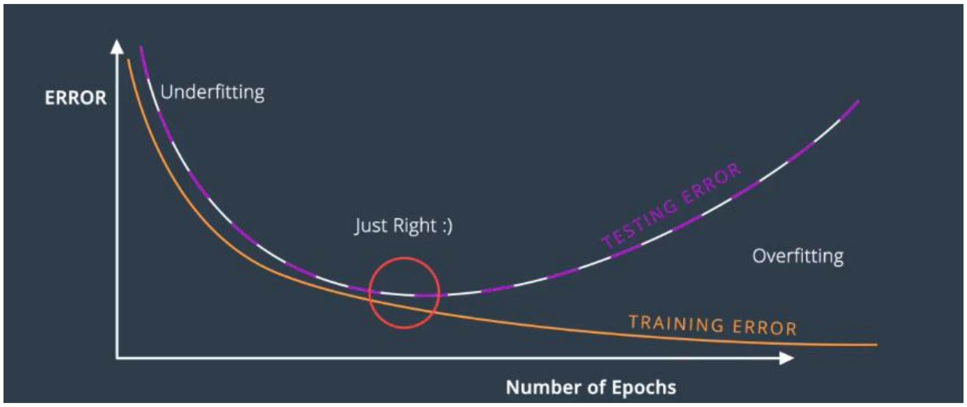
הדגימות. נמצא את ערך הbatch האופטימלי על ידי ניסוי ותהייה בתחילת תהליך האימון עד שנקבל את התוצאה הטובה ביותר. במידה והרשת סיימה לעבור על כל ה datasetוהbachים, ניתן לומר כי היא סיימה epoch.

- **Epoch**

מספר הepochים אחראי על כמות הפעמים עדכון המשקלים ברשת מתבצע בצורה לינארית. כפי שניתן לראות באיור מטה, ישנם שלושה מצבים של התנהגות הרשת

- Underfitting
- Optimal
- Overfitting

נבדוק ראשית ששגיאת האימון והולידציה יורדת שמתקבלת מהepoch שהגדרנו ונגדיל אותו עד שנגיע לשגיאה המינימלית.



- **Neural network structure**

היפר-פרמטרים השייכים למודל, כגון מספר שכבות נסתרות של נוירונים, משקלים, Activation function ועוד.

○ אימון רשת

נסביר כמה דברים הקשורים לאימון הרשת:

- **אתחול פרמטרים**

כפי שאמרנו, רשתות נוירונים עמוקות בעלות פרמטרים רבים כאשר דרך האתחול שלהם יכולה להשפיע על קצב הלמידה של הרשת שלנו ודיוקה. ניתן לאתחל לאפס את הפרמטרים האלה, כאשר גם המשקולות בכל שכבה מתאפסות. כתוצאה מכך, נקבל גרדיאנט זהה בכל שכבה של משקולות וכך גם עדכון המשקולות יהיה זהה. בדרך הזו, לא נקבל תוצאה טובה יותר מאשר אתחול הפרמטרים למספרים אקראיים קטנים.

- **פונקציית ה - loss**

פונקציית המחיר\ההפסד משמעותה עד כמה המודל שלנו צודק. ככל שהערך של פונקציית ההפסד נמוכה יותר, כך המודל "צודק" יותר. מודל "צודק" משמעו מודל הפרדיקציה במוצאו קרובה לערך המקורי שרצינו שהרשת תחזה. לכל אורך אימון המודל מטרתינו לעדכן את המשקלים ככה שהפרדיקציה שלו תהיה קרובה יותר לערך המקורי וזאת אנו עושים ע"י gradient descent. הנגזרת מלמדת אותנו על ההתנהגות (עליה\ירידה) של הפונקציה אותה גוזרים. כיוון הגרדיאנט בנקודה מצביע על הכיוון בו הפונקציה עולה ובכיוון הנגדי יורדת מהנקודה בה אנחנו נמצאים. מטרה תהליך האימון היא למצוא ערך מינימום של פונקציית ההפסד ולשם כך משתמשים בשיטות אופטימיזציה.

- **חלוקת ה - dataset**

נחלק את הdataset לשני חלקים:

- Training Dataset

המידע שאנו משתמשים בו על מנת לאמן את המודל ובאמצעותו נעדכן את המשקולות של רשת הנוירונים.

- Test Dataset

מידע הנכנס לרשת לאחר שהיא אומנה לגמרי ובודק אם המודל שלנו יפעל כראוי גם על מידע חדש שלא ראה מעולם.

- ## הבנת הבעיה

לאחר שהסברנו והרחבנו אודות תהליך הלמידה, מושגים, שיטות ועוד, נוכל כעת להרחיב על בעייתנו.
עד כה, המחקר על עיבוד האירועים האקוסטיים התמקד בעיקר על זיהוי וסיווג אירועי שמע שונים (AED) והפרדת אירועי שמע שונים (AES) לתתי קבוצות מתאימות. המחקר הצליח לזהות סיגנלים שונים אך לא הצליח לחלץ כל אחד בנפרד מקטע בעל סיגנלים רבים לכן הוא אינו עוזר לנו לפתור את הבעיה שלנו.

לאחרונה, התגלה עניין רב בשימוש ברשתות נוירונים על מנת לזהות סיגנל ספציפי ולחלץ אותו במידת הנדרש. לדוגמא מסגרת אימון קבוע בשם utterance-level permutation invariant uPIT (training) אשר פותחה תחילה לעיבוד דיבור ולאחר מכן הורחבה להפרדת אירועי שמע. גישה זו יכולה להפריד בין הסיגנלים השונים אך היא מוגבלת בכמות הסיגנלים השונים שיכולה להפריד. כתוצאה מכך, אם ישנו אירוע בעל הרבה אירועי שמע שונים שצריך להפריד, הגישה לא תדע להתמודד עם זה. נוכל להעלות את כמות אירועי השמע שנוכל להפריד בגישה אך נקבל הפסד גדול יותר באימונים השונים עקב גידול מעריכי בריבוי אפשרויות (permutation) בהפסד כשירות הPIT. כמו כן, לא ברור איזה פלט מתאים לאיזה מקור שמע שונה. לכן, שימוש ברשתות נוירונים בלבד לא עוזרת לפתור את הבעיה שלנו.

ישנה אפשרות לשלב את AED לאחר PIT (permutation invariant training) מבוסס AES, כאשר את אירוע השמע הרצוי ניקח מהפלטים. עדיין קיימות הבעיות שדיברנו עליהן: הגבלה בכמות אירועי השמע המופרדים וזיהוי לא נכון של הפלט למקור השמע השונה. על מנת לפתור זאת, נדרש לעשות AED לכל אירוע שמע בנפרד והדבר יקר מאוד.

בבורר השמע שאנו מעוניינים לבנות, אנו לא חייבים לדעת את כמות אירועי השמע המקסימלי שיש בקטע השמע הנתון, אנו מעוניינים לזהות אירוע שמע ספציפי ולדעת להפריד אותו בצורה אפקטיבית וכן בעתיד לדעת לעשות זאת בזמן אמת.

- # השיטה

לאחר שעברנו על הידע המקדים הנדרש להבנה, כעת נסביר את מימוש המערכת והחלקים השונים בה:

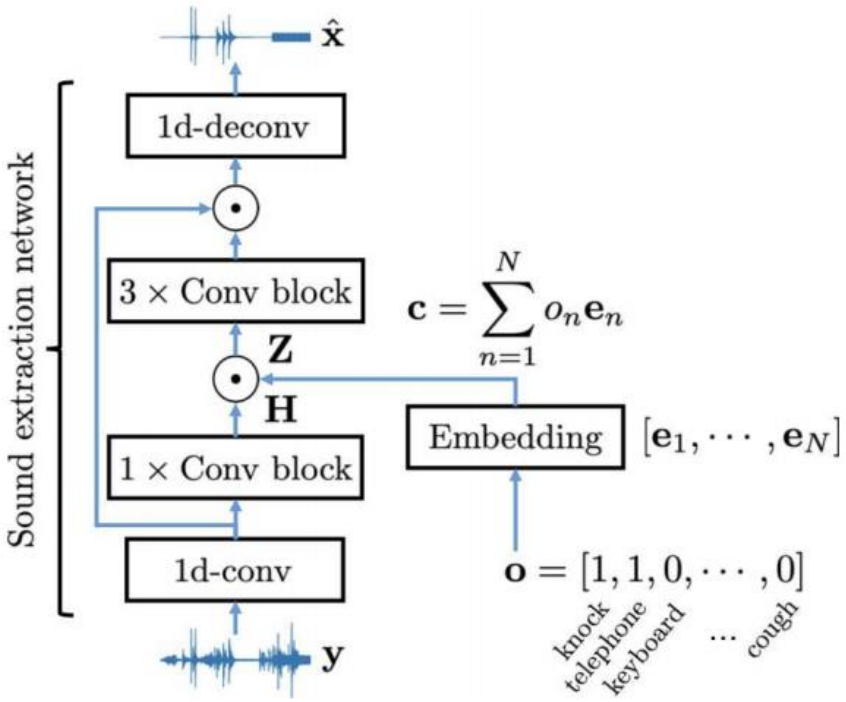
  - ## תקציר רצף המערכת

השיטה לומדת ממקור שמע אחד Y (נקרא mixure) אשר מורכב מכמה אירועי שמע (עוד על איך הוא מורכב בהמשך) ומוקטור O אשר מייצג איזה אירוע/י שמע אנו רוצים לחלץ ממנו.

המערכת פולטת את אירוע/י שמע המשוערכים, ובזמן הלמידה אנו משווים את הפלט המצוי לדרוש, ובכך מתבצעת למידת המכונה.

נפרט את שלבי רצף המערכת הלמידה:

1. אתחול ההיפר-פרמטרים.
2. הרכבת ה - data הנדרש למחזור אחד של למידה - אשר מרכיב את הוקטורים Y, O, ו - X.
3. הזנת המידע למודל הלמידה (איור למטה) עבור כל אירועי השמע שנמצאים במקור השמע.
4. חזרה לשלב 2 עד אשר עברנו על כמות הנתונים הרצויה.
5. חזרה לשלב 3 עד אשר סיימנו את כמות הepochs הרצויה - אך כעת לאותו ה - data מוזן מתוך הזיכרון ולא מורכב מחדש.
6. שמירת המודל לדיסק.



איור

באיור ניתן לראות כי בענף השמאלי מקור השמע מוזן למודל המורכב בחלקו מחלקים של ConvTasnet - תחילה encoder, ואז רצף D-Conv1, עד לdecoder.
בענף הימיני ניתן לראות כי הlabels הופכים לוקטור O (עוד על זה אחר כך) ומוזנים לשכבת Linear unbiased ומוכפלים בתוצר מן D-Conv1 הראשון בלבד.
המודל פולט X משוערך שלאחר מכן משווה מול X האמיתי.

- ## מבנה ה - data

לפני שנסביר את מבנה ה - data הסופי הנדרש נציג כמה מושגים:

- Acoustic event - AE - אירוע שמע

אירוע שמע בודד הינו הקלטה של אירוע שמע הבודד הזה, למשל הקלטת נביחת כלב, ממנו אנחנו לוקחים באופן רנדומלי בין 1.5 ל - 3 שניות כדי להרכיב את המושג הבא.
תמיד ניקח 6 אירועי שמע שונים על מנת להרכיב את מקור השמע הסופי.

- Acoustic event class - AE class - נושא אירוע השמע

נושא אירוע השמע הוא הנושא אליו שייך האירוע שמע, בדוגמא הקודמת של

הקלטת נביחת כלב אז הנושא הינו נביחת כלב.
את 6 אירועי השמע השונים נבחר מתוך בין 3 ל - 5 נושאים שונים, וזאת בהתאם לסוג הלמידה והקלט לו אנחנו מצפים (עוד על זה בנושא הבא).
זאת אומרת שתמיד יהיו לנו בין 1 ל - 2 אירועי שמע שונים בכל נושא אירוע השמע.

- Mixure - מקור שמע סופי

מקור השמע הסופי אותו נזין למערכת מורכב מאירועי שמע ונושאים בהתאם לתנאים הרנדומליים שהוצגו קודם, אך בנוסף נבחר רעש רקע רנדומלי באורך של 30 שניות, שאותו נחתוך ל6 שניות גם באופן רנדומלי.
את האירועי שמע "נלביש" על רעש הרקע כאשר לכל אירועי השמע ביחד יהיה SNR של בין $15_{[dB]}$ ל - $25_{[dB]}$.
נשתמש ב - 50,000 מקורות ללמידה ועוד 10,000 מקורות שמע מאירועי שמע זרים מהלמידה לצורך אבלואציה.

○ פונקציית ה - Loss

פונקציית הLoss שלנו מוגדרת כפונקציה של SNR

$$L = -10 \cdot log_{10}(\frac{||x||^2}{||x-\hat{x}||^2})$$

כאשר x זה כל אירועי השמע תחת נושא מסוים ו - $\hat{x}$ זה השמע המשוערך לנושא זה.

○ צורות למידה ואבלואציה שונות

דיברנו על מבנה ה - data וכעת ניתן להציג את הצורות השונות בהן נבדוק את המערכת.

- חילוץ נושא בודד - מתוך הקלטות בעלות 3 נושאי מקורות שמע כאשר נחלץ בעזרת המודל את הנושא הבודד, נריץ עליו את פונקציית הLoss שהוגדרה קודם לכן ונבצע Back propagation.

בגלל שבאותו מקור שמע יש עוד 2 נושאים נוספים, נכניס גם אותם אחד אחד

למודל ונבצע את הלמידה עבורם.

למעשה ננצל את כל הנושאים תחת אותו מקור השמע ללמידה.

- חילוץ נושא בודד - מתוך הקלטות בעלות בין 3 ל - 5 נושאי מקורות שמע

באופן דומה נעבור עבור כל מקור שמע על כל הנושאים המרכיבים אותו.

- חילוץ 2 נושאים - מתוך הקלטות בעלות בין 3 ל - 5 נושאי מקורות שמע

כאן ניתן להרכיב 2 נושאים לחילוץ בצורות שונות בהתאם לכמה נושאי מקורות שמע.

מכיוון שנדרש לבצע אבלואציה על 10,000 מקורות שמע עדיף שנשתמש ב - 2 נושאים מכל מקור שמע בלבד כדי שנבדוק מקרים כמה שיותר שונים.

- חילוץ 3 נושאים - מתוך הקלטות בעלות בין 3 ל - 5 נושאי מקורות שמע

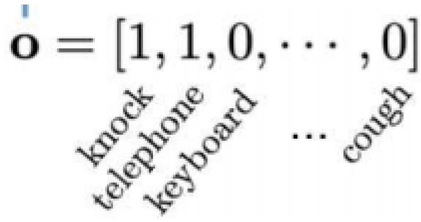
באופן דומה לחילוץ 2 נושאים.

כאשר מדובר ב3 נושאי מקורות שמע אז למעשה אנחנו מחלצים את כל האירועי השמע ומסננים את הרעש.

O vector

עד כה דיברנו על מקור השמע הסופי Y ואיך הוא נוצר, אך ראינו כי בנוסף לוקטור זה מוכנס למודל גם וקטור O.

נסתכל בקירוב על הרכב וקטור זה



מדובר בוקטור באורך N, כאשר N שווה למספר נושאי אירועי השמע הכולל במערכת שלנו (הסוגים השונים יוצגו בהמשך לאחר ניתוח ה - data).

בנוסף, איברי הוקטור מכילים 1 או 0 בלבד, כאשר אנו רוצים לחלץ נושא מסוים, האיבר במקום ב - n המתאים לאותו הנושא יהיה 1 והשאר 0.

באופן זהה וקטור זה יכיל פעמיים את האיבר 1 כאשר נרצה לחלץ 2 נושאים, ו3 פעמים כאשר נרצה לחלץ 3 נושאים.

הרשת המותאמת שלנו

עד כה הצגנו סוגי שכבות שונות, מלינארי ועד Conv-TasNet, אבל אף אחת מהן לא הזכירה את שלנו כפי שניתן לראות באיור.

הסיבה לכך היא שהרשת שלנו מורכבת מחלקים שונים הנלווים מ - Conv-TasNet וגם חלקים בסיסיים נוספים.

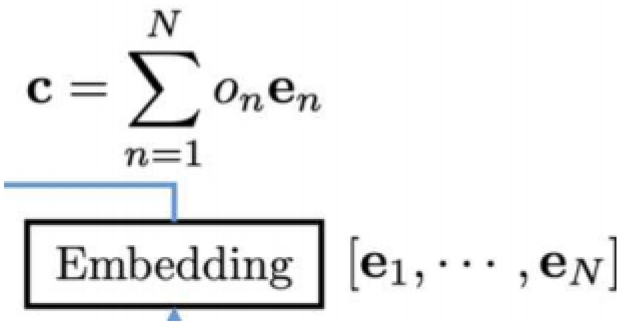
הוגדרו ההייפר-פרמטרים למודל הבאים למימוש החלקים שנלקחו מתוך ה-Conv-Tasnet ונוספים:

```
N = 256
L = 20
B = 256
H = 512
P = 3
X = 8
R = 4
D = 256
```

נעבור על החלקים השונים ברשת המותאמת שלנו.

- Embedding

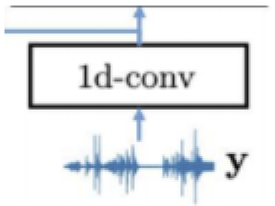
ניתן לראות שכבה זו בחלק הזה באיור



וקטור הכניסה $o \in R^{N\times 1}$, וקטור היציאה $c \in R^{D\times 1}$, לכן ניתן לממש את זה ע״י Linear layer כאשר הנוסחא שלו היא $c = A \cdot o \quad + b$.

נגדיר $A \in R^{D\times N}$(למעשה המשקולות e_n באיור), ונגדיר $b = 0$, זאת אומרת, unbiased.

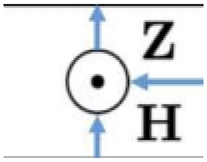
- Encoder



חלק זה הינו המקודד, נשתמש באותו המקודד מתוך ה - Conv-TasNet.

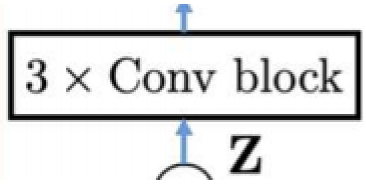
נשתמש בהייפר-פרמטרים ליצירתו.

- Elementwise product-based integration



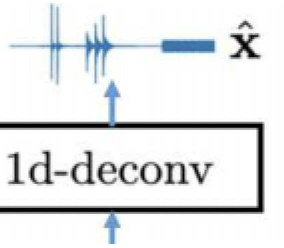
מכפלת וקטורים פשוטה.

- Mask Generator



איור זה טיפה מטעה, אך לא מדובר ב - 3 שכבות של D-Conv1 אלא מדובר ב - Mask Generator.
את המסכה נממש גם באמצעות ההייפר-פרמטרים שהוגדרו למודל, וזאת תוך כדי לקיחת שכבה ספציפית מתוך ה - Conv-TasNet.

- Decoder



באופן זהה למקודד, נשתמש בהייפר-פרמטרים שהוגדרו על מנת ליצור את המפענח כפי שמומש ב - Conv-TasNet.

כאשר שהבנו את דרישות הטכניות מן המערכת שלנו, נעבור לצורה בה מימשנו אותה.

- # יישום השיטה

כעת נציג איך מימשנו את המערכת המבוקשת.

  - ## ארכיטקטורה

שלב ראשון בבניית מערכת היא בחירת בכלים והארכיטקטורה הרצויה לבניית הפרויקט.
השתמשנו בהרבה כלים על מנת לבצע פרויקט זה, העיקריים ביניהם היו אלו, אשר נרחיב על כל אחד בנפרד ולמה השתמשנו בו.

- Python 3.7.3 - זו שפת בה השתמשנו.
פייתון ידועה לניתוח נתונים ובכלל בתחום למידת המוכנה, לכן השתמשנו בה.
יתרה מכך, החל מגרסה 3.5 של פייתון עודכנה עם תמיכה של typing, כלי חשוב לפיתוח קוד בכמות גדולה והימנעות מבאגים.
- Poetry 1.1.5 - מדובר בכלי (CLI - Command Line Interface) לניהול פרויקט פייתון.
הכלי משפר את התקנת התלויות - dependencies, הוספת הסקריפטים וגם את הסביבה הוירטואלית של הפייתון - virtualenv.
בגלל איך שפייתון עובד, כל תלות שמותקנת ידועה לכל פרוייקטי הפייתון במחשב, מה שעלול ליצור התנגשויות ובזבוז זמן מיותר בפיתוח הקוד.
Poetry באופן דיפולטי יוצר לכל פרויקט סביבה וירטואלית - העתק ריק של הפייתון הנוכחי - ושמה מתקין את כל התלויות.
- PyTorch 1.8.1 - התלות הראשית של הפרויקט, מדובר בסיפרייה למימוש רשת נוירונים ולמידת מכונה, כפי שנדרש מאיתנו לעשות.
בנוסף הסיפרייה יודעת לנצל ביעילות את הכרטיס הגרפי של המחשב - ובכך לבצע פעילויות מתמטיות כבדות ביעילות ובזריזות.
- Github - גיטהאב הינו אתר המבוסס טכנולוגיית git, המאפשר לאחסן ולשתף קוד בקלות.
מעבר לשמירה של הקוד בענן למקרה בו ימחקו הקבצים על מחשבי האוניברסיטה (המחשבים עליהם עבדנו), ניתן היה לשתף את חלקי הקוד

בקלות בין השותפים לפרויקט, להעתיק אותו בין מחשבים שונים, לשתף את המנחה בבעיות ואף לשתף חלקים ממנו באינטרנט על מנת להתייעץ עם אנשים בבלוגים שונים.

- Github Actions - מדובר במערכת לביצוע CI-CD, והרצת מערכות אלו על מחשבים מרוחקים של github (חינמי).

CI - continuous integration - מדבר על מערכת העוזרת להכנסת קוד איכותי באופן שוטף לפרויקט.

CD - continuous deployment - מדבר על מערכת המבצעת עדכון הקוד למערכות השונות בהם הוא נמצא - חלק זה לא השתמשנו אבל ניתן היה להשתמש - (עוד על זה בהצעות לשיפור).

במערכת ה CI - השתמשנו על מנת לבקר באופן אוטומטי את איכות הקוד של הפרויקט, וכדי שתעזור לנו לשמור על איכות גבוהה

להלן קוד מערכת ה - CI שרץ על מכונות ה - Github actions

```
name: final_engineering_project

on: [push]

jobs:
test:
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v2
- uses: actions/setup-python@v2
with:
python-version: 3.8
- uses: abatilo/actions-poetry@v2.0.0
with:
poetry-version: 1.0.10
- name: Install
run: poetry install
- name: Format
run: poetry run black --check .
- name: Lint
run: poetry run flake8 .
- name: Typecheck
```

```
run: poetry run mypy --strict .
- name: Test
run: poetry run pytest ./tests
```

נעבור על החלקים השונים במערכת

- Install - מתקין את תלויות הפרויקט - דרוש בשביל התקנת כלי הבדיקה הבאים.
- Format - לוודא שכל הקבצי פייתון עונים על כללי פרמוט תקינים, למשל, רווח אחד לפני ואחד אחרי הסימן =, המסמל השמה.
- Lint - מוודא מעבר לפרמוט שהקוד עונה על מוסכמות ידועות בפייתון, למשל, משתנים צריכים להיכנס בpascal_case, בעוד שמחלקות צריכות להיכתב בCamelCase.
- Typecheck - הוזכר בקצרה קודם, שלב זה של המערכת מוודא שהקוד שלנו מכיל type hints - וזאת מכיוון שבפייתון אפשר להשתמש בtypes וגם באותו הקוד לא להשתמש, ואין לכך השפעה בכלל על הריצה של הקוד אלא רק בפיתוח.
  החלטנו להשתמש ב - typings ולכן כדי שכלי זה יהיה אפקטיבי ויעצור באגים טיפשיים - שלב זה מוודא שאכן כלי זה בשימוש בכל הקוד שלנו.
- Test - שלב זה במערכת מריץ טסטים כלשהם שכתבנו, בין unit tests ל - component tests.
  לא נרחיב על סוג טסטים אלו וטסטים בכללית, אף כי טסטים הוא כלי מאוד חשוב בפיתוח, מכיוון שקוד זה לא מתוחזק מעבר להגשת הפרויקט, לא מצאנו סיבה לכתוב לו טסטים.

■ Visual Studio Code - עורך הקוד שלנו.

מדובר בעורך טקסט קליל עם הרבה תוספות, אפשר לנו להתחבר מרחוק משרתי האוניברסיטה (באמצעות ssh), להריץ פקודות ב - terminal, לכתוב ולנהל את הקוד מול github, פרמוט אוטומטי של הקוד, ועוד הרבה יכולות מגניבות.

- הקבצים הרלוונטים

במקביל לבחירת ארכיטקטורת המערכת - אספנו את הקבצים המתאימים לבניית המערכת.

- רעשי רקע

המנחה שלנו, יוחאי, שלח לנו את קבצי רעש הרקע הדרושים לבניית ה - mixures שלנו, אותם שמרנו בענן.

מדובר ב - 30 קבצים בעלי המאפיינים הבאים, אשר כל אחד באורך של 30 שניות

Key	Value
format	1 (uncompressed PCM)
number of channel	8 (unsupported)
sampleRate	16000
byteRate	256000
blockAlign	16
bitsPerSample (bit depth)	16

- אירועי שמע מחולקים לנושאים

כפי שנדרש במאמר, השתמשנו בקבצים מתוך האתגר הבא ב - kaggle (לינקים נוספים בביבלוגרפיה).

האתר מספק צורה נוחה להורדת אירועי השמע ע״י הפקודה

```
kaggle competitions download -c freesound-audio-tagging
```

לאחר ההורדה קיבלנו 9473 אירועי שמע לצורך למידה, ועוד 1600 קבצים זהים לצורך בדיקה.

כל קובץ באורך שונה אך מאפיינים זהים

Key	Value
format	1 (uncompressed PCM)
number of channel	1 (mono)
sampleRate	44100
byteRate	88200
blockAlign	2
bitsPerSample (bit depth)	16

לכל הקבצים (גם רעשי רקע וגם אירועי השמע) שהורדנו יש גם קבצי index מסוג csv הנותנים פרטים נוספים עליהם, למשל

```
Fname,label,usage,freesound_id,license
00326aa9.wav,Oboe,Private,355125,Attribution
0038a046.wav,Bass_drum,Private,90621,Creative Commons 0
```

בדוגמא פה יש 2 שורות של אירועי השמע שכל אחד מהם שייך לנושא אחר תחת השדה .label

- הרכבת ה - **datasets**

כעת אחרי שהורדנו את קבצים המתאימים ניגשנו ליצור את ה - datasets.
Dataset הינה מחלקה בסיסית ב - PyTorch אשר עוזרת בקריאת ועיבוד הנתונים.
מה שיפה בה היא שבקוד רושמים בה דרך משיכה פריט מידע בודד, ולאחר מכן ניתן להשתמש בה על מנת למשוך ולעבד כמה פריטים במקביל ע״י batching.

יצרנו שרשרת של datasets וזאת בשביל לפצל אחריות בקוד, אף כי בשאר הקוד השתמשנו רק ב - dataset האחרון.
נרחיב כעת על ה - datasets השונים ועל פעילותם.

- NoiseDataset

Dataset המייצא פרטי מידע של רעשי רקע.
ניתן לראות בקוד את סוג המשתנים הקיימים לכל פריט מידע ב - Dataset זה

```
sample = {
"waveform": waveform,
"rate": rate,
}

return sample
```

- Waveform - וקטור המכיל את אות השמע.
- Rate - מספר המציין את מספר הדגימות לשנייה בו נדגם אות השמע.

- KaggleDataset

באופן דומה, Dataset זה מייצא פרטי מידע של אירועי השמע.
נסתכל על המשתנים המיוצאים ממנו

```
sample = {
"waveform": waveform,
"rate": rate,
"label": label,
}

return sample
```

ניתן לראות שהוספנו את פריט המידע הנוסף הרלוונטי - label - מחרוזת של נושא אירוע השמע.

- RandomDataset

פה הכנסנו את כל המספרים האקראיים הנדרשים כדי לייצר mixure לפי הנדרש לעיל.

נעבור על המשתנים המיוצאים ותפקידם כדי להבין את אחריות ה - Dataset.

```
sample = {
    "is_train": is_train,
    "noise_index": noise_index,
    "kaggle_index": kaggle_index,
    "noise_start": noise_start,
    "kaggle_start": kaggle_start,
    "kaggle_start_in_noise": kaggle_start_in_noise,
    "kaggle_length": kaggle_length,
    "events_gain": events_gain,
}

return sample
```

- Is_train - האם המשתנים הללו מתאימים לשימוש עם Dataset האימון או הבדיקה - מכיוון שעבור KaggleDataset, יש ערכים שונים.
- Noise_index - באיזה רעש רקע להשתמש ליצירת ה - mixure, מספר הנע בין 0 ל 29 (בגלל שיש לנו 30 רעשי רקע).
- Kaggle_index - וקטור המכיל 6 מספרים האקראיים שבעזרתם נבחר 6 אירועי רעש ליצירת ה - mixure.
- Noise_start - השנייה ממנה נחתוך 6 שניות מרעש הרקע ליצירת ה - mixure.

  בגלל שאורך ההקלטה של רעש הרקע היא תמיד 30 שניות, מספר זה נע בין 0 ל24 (לא בהכרח שלם).
- Kaggle_start - וקטור של 6 מספרים אקראיים בין 0 ל - 1 שאומר שמאיזה חלק יחסי בהקלטה נתחיל לחתוך ממנה את אירוע השמע.

- Kaggle_start_in_noise - וקטור של 6 מספרים אקראיים בין 0 ל 1 - שאומר באופן יחסי איפה למקם את אירוע השמע החתוך בתוך שמע הרקע.
- Kaggle_length - וקטור של 6 מספרים אקראיים בין 1.5 ל 3 - שאומר מה יהיה אורך כל אירוע שמע בנפרד.
- Events_gain - מספר אקראי בין 15 ל 25 שאומר מה יהיה הגבר כל אירועי השמע ביחס לרעש הרקע.

■ MixureDataset

זה הוא ה - Dataset האחרון המרכיב את המידע, המשתמש בכל ה - Datasets שקדמו לו.

לאחר החיתוכים והחיבורים המתאימים, הוא מייצא את המשתנים הבאים

```
sample = {
    "is_train": is_train,
    "waveform": waveform_gpu,
    "events": [event["waveform"] for event in class_events],
    "labels": labels,
}

return sample
```

נעבור על המשתנים אלו

- Is_train - האם מדובר ב - mixure הנוצר מאוסף אירועי השמע של האימון או הבדיקה.
- Waveform - וקטור של השמע של ה - mixure.
- Events - מערך של 6 וקטורים של אירוע השמע המרכיבים את ה - mixure.

  צריך אותם לצורך אימון הרשת.
- Labels - מערך של 6 מחרוזת של הנושאים של אותם אירועי השמע, לבניית ה - O-Vector בהמשך.

■ TrainDataset / TestDataset

נסתכל על ה - Dataset האחרון בפרויקט

```
sample = {
    "waveform": mixure_sample["waveform"],
```

```
"events": [
    {
        "waveform": event_wav,
        "o_vector":
self._o_vector_utility.get_o_vector_by_label(
            events_labels[i]
        ),
    }
    for i, event_wav in enumerate(events_wavs)
],
}

return sample
```

למעשה מדובר בהתאמה של הנתונים היבשים לקראת למידה, פה אנחנו למעשה ממירים את ה - Labels ל - O-Vector, בנוס, כאשר מבצעים למידה ובדיקה שונות (כפי שהורחב בצורות למידה ואבלואציה שונות).
למעשה המילה event כעת תופסת אצלינו משמעות בפרויקט של צורת למידה מסויימת, ולא אירוע שמע.

○ מימוש ה - Gradient descent and backpropagation

תחילה יצרנו את האובייקטים הדרושים

```
model = Model(
o_vector_length=o_vector_utility.get_vector_length(),
).to(gpu_device)

optimizer = torch.optim.Adam(
model.parameters(),
lr=5e-4,
)
scheduler = torch.optim.lr_scheduler.StepLR(
optimizer,
step_size=step_size,
gamma=0.2,
)
```

כפי שנדרש במאמר והוזכר קודם לכן, השתמשנו בAdam algorithm לאופטימיזציה,
בנוסף השתמשנו בשיטה בסיסית לדעיכת קצב הלמידה - StepLR.
לאחר מכן העברנו את האובייקטים הללו ומשתנים נוספים ל Solver.py, וזאת על מנת לקבע את לוגיקת הלימוד לקובץ אחד.
נעבור על תהליך הלמידה

```
for epoch in range(self._epoch_size):
for (i, batch) in enumerate(self._data):
y = batch["waveform"]
for event in batch["events"]:
x = event["waveform"]
o = event["o_vector"]
x_pred = self._model(y, o)
mse_loss = mean(self._criterion(x_pred, x), [1, 2])
loss = mean(10 * log10(mse_loss))

self._optimizer.zero_grad()
loss.backward()
torch.nn.utils.clip_grad.clip_grad_norm(
self._model.parameters(),
self._clip_value,
)
```

```
self._optimizer.step()
self._scheduler.step()
print("Finished training!")
```

ניתן לראות כי מתבצעות כאן כמה לולאות וזה בהתאם להייפר-פרמטרים של תהליך הלמידה

- Epoch - לולאה על מספר החזרות הנדרש ללמידה
- Data - לולאה על ה - mixures - אבל ב - batches כפי שהוגדר בהייפר-פרמטרים.
- Events - לולאה על ה events המוגדרים לאותו mixure (או קובץ של mixures כפי שהוגדר קודם).

עבור כל אחד מאלו נבצע את תהליך הלמידה, שלבסוף נעדכן את המשקולות לפי Adam, וגם נפחית את תהליך הלמידה בעזרת StepLR.

○ בניית הרשת

בעת עברנו למימוש הרשת שלנו, כפי שהוגדר בחלק של הרשת המותאמת שלנו.

תחילה יצרנו חתימה לפונקציות וניסינו להבין כניסה ומוצא.

```
class Model(nn.Module):
    def __init__(
        self,
        o_vector_length: int,
    ) -> None:
        super().__init__()

    def forward(self, y: torch.Tensor, o: torch.Tensor) -> torch.Tensor:
        output = y
        return output
```

קיבלנו למודל את אורך הוקטור o בשביל ליצור את שכבת ה - embedding, וזה הוא הפרמטר היחיד הנדרש.

אחרי שהשתמשנו במודל ריק זה, ודיבגנו את גודל הקלטים (y ו - o) והפלטים (output), יכולנו לגשת למימוש עצמו.

יצירת השכבות השונות הייתה די פשוטה באמצעות מימוש של ה - ConvTasNet ע״י torchaudio.

```
from torchaudio.models.conv_tasnet import ConvBlock, MaskGenerator

class Model(nn.Module):
    def __init__(
        self,
        o_vector_length: int,
    ) -> None:
        super().__init__()

        self._N = 256
        self._L = 20
        self._B = 256
        self._H = 512
        self._P = 3
        self._X = 8
        self._R = 4
        self._D = 256
```

```
self._enc_stride = self._L // 2

self._encoder = torch.nn.Conv1d(
    in_channels=1,
    out_channels=self._N,
    kernel_size=self._L,
    stride=self._enc_stride,
    padding=self._enc_stride,
    bias=False,
)
self._decoder = torch.nn.ConvTranspose1d(
    out_channels=1,
    in_channels=self._N,
    kernel_size=self._L,
    stride=self._enc_stride,
    padding=self._enc_stride,
    bias=False,
)

self._before_o = ConvBlock(
    io_channels=self._B,
    hidden_channels=self._H,
    kernel_size=self._P,
    padding=1,
    no_residual=True,
)

self._mask_generator = MaskGenerator(
    input_dim=self._N,
    num_sources=1,
    kernel_size=self._P,
    num_feats=self._B,
    num_hidden=self._H,
    num_layers=self._X,
    num_stacks=self._R,
)

self._embedding = nn.Linear(
    in_features=o_vector_length,
```

```
out_features=self._D,
bias=False,
)
```

ניתן לראות כי הגדרנו את היפר-פרמטרים של הרשת במודל ולא קיבלנו אותם מחוץ למודל - לכן הם hard-coded.
זה בסדר למימוש שלנו, אבל אם נראה יש אפשרות לקבל אותם מבחוץ ע״י שינוי זריז בקוד.
השתמשנו בשכבת ה - MaskGenerator ובשכבת ה - DConv1 (בקוד - ConvBlock) מתוך המימוש של torchaudio.
בנוסף יצרנו שכבות נוספות עם דגשים הבאים

- Encoder - כפי שמומש ב - ConvTasNet.
- Decoder - כפי שמומש ב - ConTasNet.
- Before_O - ריווח (padding) שווה ל - 1 וזאת מכיוון שגודל הקרנל הוא 3 (P=3) ואנחנו רוצים אותו גודל כניסה ויציאה לשכבה הזאת.
- Embedding - כפי שצוין קודם, השתמשנו ב - Linear אך עם bias=Flase.

כעת עדכנו את פונקציית ה - forward

```
def forward(self, y: torch.Tensor, o: torch.Tensor) -> torch.Tensor:
    c = self._embedding(o)
    c_row, c_column = c.size()
    c3d = c.reshape(c_row, c_column, 1)

    encoded = self._encoder(y)
    _, skip = self._before_o(encoded)
    H = skip
    Z = H * c3d
    mask = self._mask_generator(Z)
    masked = mask.squeeze(1) * encoded
    output = self._decoder(masked)

    return output
```

קוד דומה לכך קיים ב - ConvTasNet, וכך ידענו כיצד להרכיב את המסכה (mask) עם הקידוד
(encoded).

- הרצה

כעת כשכתבנו את עיקר הקוד, רצינו שיהיה בידינו את האופציה להריץ את הפרויקט תחת כמה תנאים שונים

- שיהיה ניתן לראות פרמטרים שונים במהלך הלמידה.
- שיהיה אפשר לשלוט על ההיפר-פרמטרים של הלמידה (לא של המודל).
- שיהיה אפשר להריץ את זה ברקע ולשמור את הלוג לקובץ (ככה נוכל להפיק גרפים אחר כך).
- דברים זהים עבור שלב הבדיקה.

החלטנו לכתוב הפרויקט כסוג של תוכנת CLI - Command Line Interface - וזאת מכיוון שקל לאתחל אותו עם פרמטרים שונים, לא צריך גישה לשינוי לאחר התחלת ריצה, ובטרמינל של linux נוכל להריץ פקודה זו ברקע ולהפיק את הלוגים לקובץ, למשל ע״י הפקודה הבא

```
poetry run python -u -m final_engineering_project --train-enable --train-override-model --train-step-size 7000 --train-print-progress-every 1 --train-save-model-every 1 --train-epoch-size 8 --train-batch-size 8 --train-size 56000 > train_output_17.log &
```

- מהלכים השונים בפרויקט

השתמשנו בסיפריית ה - argparse - המגיעה כחלק מהגירסא שלנו של פייתון כדי להגדיר פרמטרים ל - CLI.
הטבלה הבאה ניתן לראות את סוגי הפרמטרים שהוגדרו ומה תפקידם

שלב	שם	ערך דיפולטי	הסבר
יצירת data, מדובר בשלב שיוצר את ה -	--data-enable	False	האם להפעיל את שלב יצירת ה - data
	--data-train-size	100	מספר ה - mixures ליצור לטובת שלב

mixures לדיסק, וזאת על מנת לשמוע אותם וגם להשתמש בהם לאימון ולבדיקה, חוסך את היצירה שלהם בזמן אמת.			האימון
	--data-test-size	10	מספר ה - mixures ליצור לטובת שלב הבדיקה
	--data-min-mixure	3	מספר נושאי אירועי השמע ליצירת ה - mixures, למעשה מדובר בטווח שעבור כל mixure יוגרל בנפרד. עוד על כך ב[צורות למידה ואבלואציה שונות]
	--data-max-mixure	3	
	--data-print-progress-every		כל כמה mixures להדפיס לוג מסויים. בשלב היצירה זה רק כמה זמן לקח לכל x שורות. אם פרמטר זה לא ניתן לתוכנית אז לא ידפיס כלל
אימון המודל	--train-enable	False	האם להפעיל כלל את שלב אימון המודל
	--train-use-fs	False	האם לקרוא את ה - mixures מתוך הדיסק לא לייצר אותם לבד
	--train-min-mixure	3	אם יוצרים את ה - Data בשביל הזה (ז״א --train-use-fs הוא False) אז זה מדבר על אופן
	--train-max-mixure	3	

			יצירת ה - Data
	--train-override-model	False	האם להמשיך לאמן את המודל או ליצור חדש ולדרוס את הישן
	--train-size	100	מספר ה - mixures ליצירה (אם יוצרים)
	--train-epoch-size	1	כמה חזרות לעשות באימון על המידע
	--train-batch-size	8	גודל ה - batch לצורך יעילות
	--train-step-size	100	כל כמה צעדים להקטין את ה - LR
	--train-save-model-every		כל כמה חזרות בלולאת הלמידה לשמור את המודל
	--train-print-progress-every		כל כמה חזרות בלולאה להדפיס את פרטי הלמידה האחרונה (טוב ליצירת דוחות לאחר מכן)
אבלואציה של תוצר המודל	--test-sample	False	האם להפעיל את שלב הבדיקה של תוצר המודל, יוצר קבצים ל - fs, עוד על כך בתוצאות
אבלואציה של המודל	--test-enable	False	האם להפעיל כלל את שלב האבלואציה של

			המודל
	--test-size	100	כמה mixures ליצור לבדיקה (במידה שלא משתמשים ב - fs)
	--test-batch-size	8	גודל ה - batch לצורך יעילות
	--test-print-progress-every		כל כמה חזרות בלולאה להדפיס את פרטי האבלואציה האחרונה (טוב ליצירת דוחות לאחר מכן)

בעזרת פקודות אלו, יכולנו להריץ את הפרויקט עבור מקרים שונים ובכך לקבל את התוצאות הדרושות.

• תוצאות

נעבור על הגרפים והפקודות השונות שקיבלנו בזמן הלמידה והאבלואציה:

○ Train loss

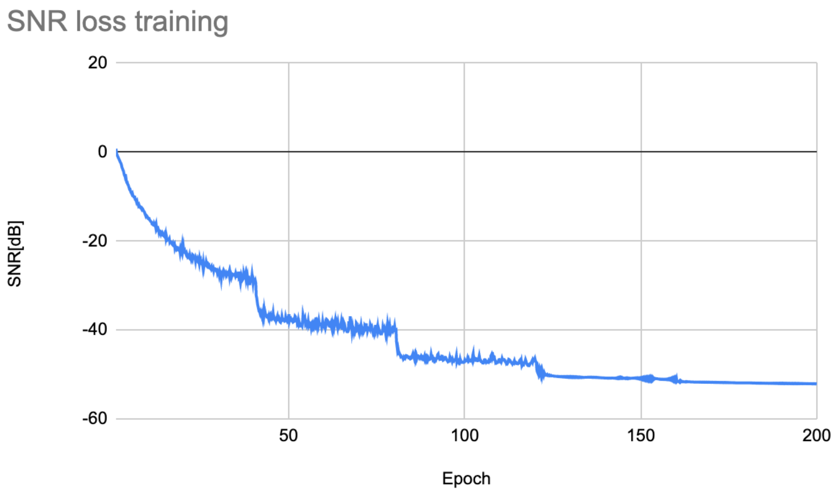
ביצענו אימון על 50 אלף mixures (מתחלק ב - 8 אז אין בעיה) על פני 200 epochs כנדרש במאמר בעזרת הפקודה הבא

```
poetry run python -u -m final_engineering_project --train-enable --train-override-model --train-step-size 250000 --train-print-progress-every 1 --train-save-model-every 1 --train-epoch-size 200 --train-batch-size 8 --train-size 50000 > train_output.log &
```

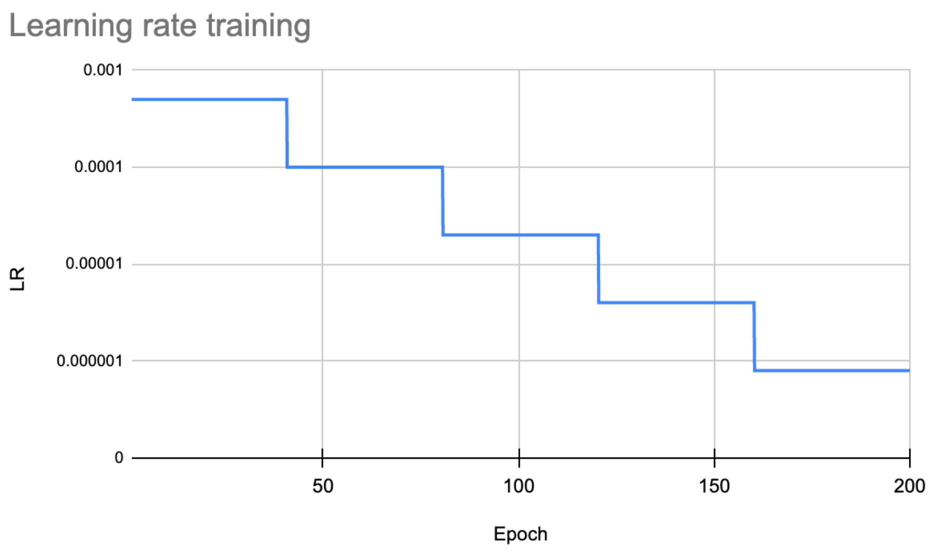
כאשר הסתכלנו על שורות מתוך קובץ האימון ראינו .

```
trained 2579/6250 of batches, epoch 1/200, loss is -8.131072044372559, norm is 0.004999999577166657, lr is 0.0005, this batch took 3.4566004276275635 seconds.
trained 2580/6250 of batches, epoch 1/200, loss is -3.9551329612731934, norm is 0.004999999204879026, lr is 0.0005, this batch took 3.431574583053589 seconds.
```

יצאנו את התוצאות לאקסל ובנינו את גרף הלמידה הבא



כדי להבין את המדרגות שנמצאו כאן יצרנו גם גרף של קצב הלמידה הדועך, פרט שגם מיוצא בלוגים של הלמידה.



ניתן לראות כי הרשת אכן למדה בעזרת המידע של הלמידה, אך בו זמנית עלינו לבדוק אם אכן מדובר בלמידה ואיכותית ולא overfitting ל - Data הלימוד (השונה מ - Data לאבלואציה).

זאת נבדוק בחלקים הבאים.

- אבלואציה וקבצי שמע

יצרנו קבצי שמע בעזרת הפקודה הבא

```
poetry run python -u -m final_engineering_project --test-sample > test_sample_output.log &
```

אשר קובץ האבלואציה פלט את השורה הזו לגבי פרטי האבלואציה

תכלנו על שורות מתוך קובץ האימון ראינו

```
loss is -54.325048670859, this batch took 3.4566004276275635 seconds.
```

מדובר ב - SNR עבור batch שלם, ניתן לראות כי ה - SNR עבור הקלטות אלו נורא נמוך, כעת נקשיב לאחד ה - mixures.

- Mixure
  - המקור - AE class - Squeak
  - פלט של הרשת - AE class - Squeak
  - המקור - AE class - Oboe
  - פלט של הרשת - AE class - Oboe
  - המקור - AE class - Electric piano
  - פלט של הרשת - AE class - Electric piano

הקלטות אלו ועוד נמצאות בתיקיית הקלטות אבלואציה של המודל.

- **הצעות לשיפור**
  - הרשת די כבדה לריצה ותופסת משאבים רבים בעת הלימוד, לדוגמא עבור ריצה על gpu בודד ו - batch של 8

```
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:0B:00.0 Off |                    0 |
| N/A   49C    P0   297W / 300W |  22272MiB / 32510MiB |     99%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

ניתן לראות כי תופסת בסביבות ה - GiB20 מזיכרון כרטיס הגרפי.

לכן לייעול מהירות המערכת ניתן להשתמש בכמה gpus בו זמנית כפי שמוגדר במדריכים של pytorch.

עדכון:

הרצנו את הפקודה הבאה

```
print("Let's use", torch.cuda.device_count(), "GPUs!")
```

וקיבלנו את ההדפסה הבאה

```
er/extensions/ms-pytho
e 500 --train-epoch-si
Let's use 8 GPUs!
(final-engineering-pro
```

לא תהיה להריץ את המערכת הנוכחית על כמה gpu

  - CD - Continous deployment

הוזכר מעט בארכיטקטורה של הפרויקט, CD איכותי לפרויקט שכזה יכול להתבטא בכמה צורות.

- בעת הכנסת קוד לפרויקט, להריץ אימון קטן על הפרויקט לראות אם הוא בטווח המצופה.
- את הלוגים לעלות אוטומטית ל - google spreadsheet שמה יצרנו פירסור וגרפים לסוג ה - logs השונים.
- לאחר אימון, לבדוק את המודל מול הקלטות של data הבדיקה ולעלות את התוצאות לתיקיית הקלטות אבלואציה של המודל.

כל אלו מהלכים שעשינו באופן ידני כל פעם ששינינו מעט את קוד הפרויקט, אז CD איכותי לפרויקט יעשה מהלכים אלו בשבילינו אוטומטית.

- ניתן לראות כי התכנסות תחילה של תהליך הלמידה לוקח מספר רב של epochs, ורק לאחר מכן צובר תאוצה.
  דרך טובה לייעול המערכת תהיה שמירה של מודל התחלתי או אתחול המשקולות בצורה כזו שתייעל את תהליך הלמידה.

- ## מסקנות

בפרויקט זה פתרנו ויישמנו זיהוי וסיווג אירועי שמע שונים באמצעות ערוץ מיקרופון בודד מרצועת אודיו אשר מערבבת כמה מקורות סאונד יחד.

הצלחנו לנקות את כל רעשי הרקע וכך הבנו באופן ברור את אירוע השמע שלו אנו רוצים להקשיב.

- **Bibliography**

  - המאמר שהיה עלינו לממש - Listen to What You Want: Neural Network-based Universal Sound Selector
  - הפרויקט בgithub
  - תיקיית הקלטות אבלואציה של המודל
  - כללי tyings בהם השתמשנו בפרויקט
  - Freesound Dataset Kaggle 2018 corpus - FSD
  - REVERB challenge corpus - REVERB
  - Gentle Introduction to the Adam Optimization Algorithm for Deep Learning
  - רשת עצבית מלאכותית – ויקיפדיה
  - סיגמואיד (מתמטיקה) – ויקיפדיה
  - Softmax function
  - מאמר בנושא RELU
  - מאמר בנושא gradient descent
  - What is Gradient Clipping?. A simple yet effective way to tackle… | by Wanshun Wong
  - Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning
  - Linear layer - MATLAB linearlayer
  - Linear/Fully-Connected Layers User Guide :: NVIDIA Deep Learning Performance Documentation
  - מאמר בנושא D d-Conv1
  - מאמר בנושא Conv-TasNet
  - מאמר בנושא batch normalization