

דו"ח פרויקט מסכם

Stage 1 - בחירת אלגוריתם להערכה:

מודל מס' 1 - Lookahead Optimizer:

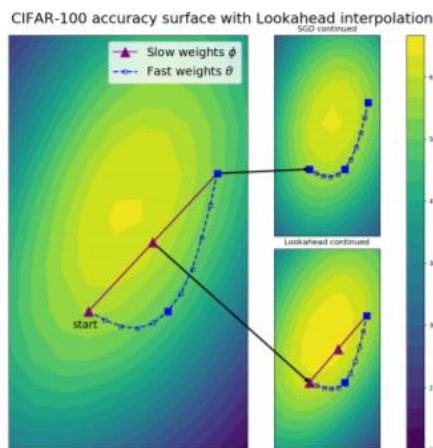
בפרויקט המסכם החלטנו להתמקד באופטימיזר lookahead. אופטימיזר lookahead היא שיטת אופטימיזציה שמשפרת את הביצועים של אופטימיזרים אחרים, שעליה היא בנויה. אנחנו בחרנו להתמקד במאמר "Lookahead Optimizer: k steps forward, 1 step back"

אשר מציג ניסויים ומימושים משלו של האופטימיזר. ניתן לראות במאמר שמבחינת ביצועים היה שיפור משמעותי מאשר מבשימוש באופטימיזר רגיל, והתוצאות מובהקות. בפרויקט אנחנו בחרנו להשתמש במימוש של Tensorflow Lookahead optimizer ובחרנו בו כמודל הראשון להשוואה.

להלן קיימים השלבים הבאים:

1. האופטימיזר מעדכן בצורה איטרטיבית שני סטים של משקלים.
2. האופטימיזר בוחר אופטימיזר פנימי.
3. הוא מעדכן את כיווני החיפוש במשקלים שנבחרו על ידי האופטימיזר הפנימי.
4. הוא מעדכן את המשקלים האיטיים בכל k צעדים בהתבסס על כיוון המשקלים המהירים ושני הסטים של המשקולים המסוכנרכנים. בצורה כזאת השיטה משפרת את יציבות הלמידה ומורידה את השונות של האופטימיזר הפנימי.

[1]



Algorithm 1 Lookahead Optimizer:

Require: Initial parameters ϕ_0 , objective function L
Require: Synchronization period k , slow weights step size α , optimizer A
for $t = 1, 2, \dots$ **do**
 Synchronize parameters $\theta_{t,0} \leftarrow \phi_{t-1}$
 for $i = 1, 2, \dots, k$ **do**
 sample minibatch of data $d \sim \mathcal{D}$
 $\theta_{t,i} \leftarrow \theta_{t,i-1} + A(L, \theta_{t,i-1}, d)$
 end for
 Perform outer update $\phi_t \leftarrow \phi_{t-1} + \alpha(\theta_{t,k} - \phi_{t-1})$
end for
return parameters ϕ

לאלגוריתם lookahead קיימות מספר יתרונות וחסרונות:

יתרונות-

- הוא משפר את יציבות הלמידה.
- מקטין את השונות של האופטימיזר הפנימי בעזרת שימוש בזיכרון זניח על מנת לעשות זאת.
- משפר את robustness של האופטימיזר הפנימי.

- מאפשר להגיע להתכנסות מהירה יותר של אלגוריתם הלמידה.
- מראה הבטחה גדולה והוכחות אמפיריות על המון סוגים של דטסטים, בנצ'מרקים, וארכיטקטורות שונות.

חסרונות-

- שימוש באופטימיזר אשר לא מסתכן טוב עם lookahead יכול לגרום לביצועים רעים, ואף גרועים יותר ממה שהיו.
- מכיוון שהlookahead משתמש באופטימיזר פנימי, הוא איטי יותר מאופטימיזרים אחרים עקב ביצועים כפולים.
- לא ניתן לבצע אופטימיזציית hyperparameters באופן אוטומטי.

Stage 2 - הצעת שיפור:

מודל מס' 2-שיפור המודל הקיים (**improved_lookAhead**):
כפי שצויין בתחילת הפרויקט, אופטימיזר lookahead מייעל ומשפר את אלגוריתמי האופטימיזרים הפנימיים שבהם הוא משתמש. נכון לעכשיו האלגוריתם משתמש באופטימיזר פנימי אחד אשר עוזר לו לקבוע את המשקלים על ידי שימוש בו. נשאלת השאלה האם lookahead יכול בעצם לשפר את עצמו על ידי שימוש בlookahead כאופטימיזר פנימי.

מהלך הניסוי שאנחנו מציעים הוא:

1. יצירת אופטימיזר lookahead רגיל.
2. השמת אופטימיזר פנימי מוכר שהאופטימיזר בשלב מס' 1 יקבל.
3. יצירת אופטימיזר lookahead חדש אשר בו נשתמש על מנת לאמן את המודל שלנו.
4. לאופטימיזר שיצרנו בשלב מס' 3 ניתן את האופטימיזר משלב מס' 1 כקלט לשימוש כאופטימיזר פנימי.

בצורה זאת יצרנו 2 שכבות של אופטימיזציה פנימית במקום שכבה אחת אשר הוצגה במאמר המקורי. הציפיות שלנו הם שאלגוריתם lookahead יכול לשפר את הביצועים אף של עצמו ומכאן ניתן למצוא ולגלות כמה שכבות אופטימיזציה עדיין תורמות לביצועי המודל.

את השראת הרעיון שאבנו ממילות הסיכום של המאמר [1] **"Lookahead Optimizer: k steps forward, 1 step back"** we present Lookahead, an algorithm that can be combined with any standard optimization method. Our algorithm computes weight updates by looking ahead at the sequence of "fast weights" generated by another optimizer. We illustrate how Lookahead improves convergence by reducing variance and show strong empirical results on many deep learning benchmark datasets and architectures. על פי טענתם האופטימיזר אמור לעבוד עם כל אופטימיזר סטנדרטי, ולכן אין סיבה שלא יעבוד גם עם עצמו.

Stage 3 - בחירת אלגוריתם מוכר להשוואה:

מודל מס' 3-Adam optimizer(baseline_model):

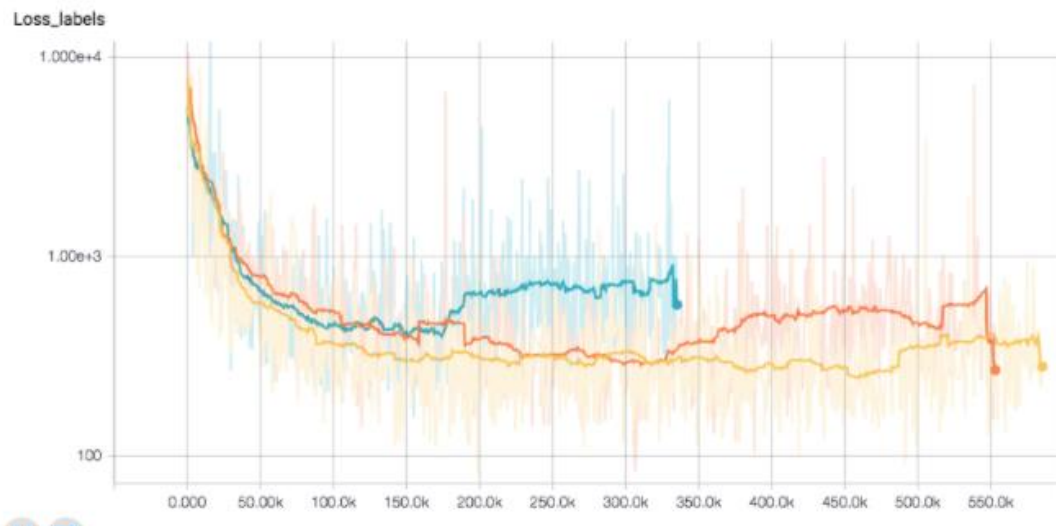
מבחינת אלגוריתם אופטימיזציה מוכר, בחרנו ב-Adam optimizer מספריית Tensorflow ו-keras.

אדם משלב את המאפיינים הטובים ביותר של אלגוריתמי AdaGrad ו-RMSProp כדי לספק אלגוריתם אופטימיזציה שיכול להתמודד עם שיפועים דלילים בבעיות רועשות.[4] אופטימיזר Adam מתחשב בהיסטוריה של ה-learning rate הקודם ובמידה ויש כמה learning rates הוא יתייחס אליהם בעזרת משקלים ממושקלים, כך שכל שה-learning rate יותר עדכני הוא יקבל משקל גדול יותר.

[2]

```
tf.keras.optimizers.Adam(  
    learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False,  
    name='Adam', **kwargs  
)
```

[3]



Stage 4 - הערכה של האלגוריתמים שנבחרו:

הרצת האלגוריתמים:

- על מנת להשתמש באלגוריתמי האופטימיזרים הנ"ל יצרו רשת נוירונים פשוטה עם 2 שכבות קונבולוציה שאחריהן מגיעה שכבת pooling המצמצמת את המימדים, ולאחר מכן השתמשנו בשכבת flatten על מנת ל"שטח" את המימדים של השכבה הקודמת ולבסוף שתי שכבות dense על מנת לחבר את הנוירונים.
- בשכבת ה-dense האחרונה השתמשנו באקטיבציה softmax כיוון שהיא מתאימה לסוג המשימה שאנו רוצים לבצע ובעצם סוכמת את כל המשקלים של הלייבלים האפשריים ל-1.
- בפונקציית ה-loss עשינו שימוש ב-categorical_crossentropy כיוון שאנו במשימת single labels.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	896
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 128)	204928
dense_1 (Dense)	(None, 3)	387
Total params: 224,707		
Trainable params: 224,707		
Non-trainable params: 0		

גרסאות המודלים-

בפרויקט השתמשנו בגרסאות הבאות עבור המודלים והאופטימיזרים:
מבחינת המודל שבחרנו, השתמשנו במודל Sequential מסוג CNN אשר ייבאנו מהספרייה tensorflow.keras.model.

אופטימיזר Lookahead:

עבור אופטימיזר Lookahead ייבאנו את הגרסה העדכנית מהספרייה tensorflow_addons.optimizers והשתמשנו בה גם למודל לפני השינוי, וגם למודל אחרי השינוי.

אופטימיזר Adam:

עבור אופטימיזר Adam (האלגוריתם המוכר) ייבאנו את הגרסה העדכנית מהספרייה
tensorflow.keras.optimizers.

הדטסטים:

- כל דאטה סט עבר עיבוד מקדים בו נירמלנו את ערכי המערכים לבין 0 ל-1 וכמו כן יצרנו גודל אחיד (23,23,3) או (23,23,1) במידה והיו בצבעי שחור-לבן, של כל התמונות על מנת שהמודל יוכל לעבוד על כולן בצורה נוחה.
- הדטסטים שהשתמשנו בעבודה נלקחו מהספרייה tensorflow_datasets וביצענו טעינה שלהם אל מערכים של numpy.
- מהדאטסטים הגדולים לקחו 1000 דוגמאות לאימון ו-500 לבדיקה, כאשר או שנלקחו מתחילת המערך או מסופו.
- שמות הדאטסטים:

1. beans
2. cifar10 (1000 and 500 from start)
3. smallnorb (1000 and 500 from start)
4. svhn_cropped (1000 and 500 from start)
5. mnist_corrupted (1000 and 500 from start)
6. mnist (1000 and 500 from start)
7. kmnist (1000 and 500 from start)
8. fashion_mnist (1000 and 500 from start)
9. horses_or_humans
10. cmaterdb
11. cmaterdb/devanagari
12. cmaterdb/telugu
13. rock_paper_scissors
14. cifar10_2 (1000 and 500 from end)
15. smallnorb_2 (1000 and 500 from end)
16. svhn_cropped_2 (1000 and 500 from end)
17. mnist_corrupted_2 (1000 and 500 from end)
18. mnist_2 (1000 and 500 from end)
19. kmnist_2 (1000 and 500 from end)
20. fashion_mnist_2 (1000 and 500 from end)

פרוטוקול ההערכה - Cross validation:

-kfold

הינה שיטת ולידציה אשר מחלקת את הנתונים בצורה אקראית לחלקים שווים (k) כך שבכל פעם האימון מבצע בדיקה על חלק אחר.

התבקשו בעבודה להשתמש בקרוס ולידיישן חיצוני בעל 10 פולדים ולבצע אימון עם הפרמטרים הטובים לאחר שביצענו 3 קרוס ולידיישן פנימי, על מנת לבצע היפרפרמטרים ולמצוא את הפרמטרים הטובים ביותר.

אופטימיזציה Hyperparameters:

את ההיפרפרמטרים ביצענו באמצעות סיפריית [optuna 5]. הספרייה מאפשר ביצוע של היפרפרמטרים על רשתות נוירוניות של keras, השכבות והפונקציות השונות שהן כוללות. בעבודתנו היינו צריכים לבצע את ההיפרפרמטרים לאופטימיזר השונה בכל אלגוריתם בתוך 3 קרוס ולידיישן ולהחזיר את הממוצע הטוב ביותר של אותם פרמטרים. הרצנו את הבדיקה עם 50 טריאלס על מנת שנוכל להחזיר את הפרמטרים הטובים ביותר לאחר מספר רב של נסיונות.

באלגוריתם הראשון שנלקח מתוך המאמר lookAhead- החלטנו לחפש את הפרמטרים: optimizer - אשר מגדיר את האופטימיזר המקורי בו נעשה שימוש לחישוב ומימוש הגרדיאנט. slow_step_size - מגדיר את היחס בו האלגוריתם יעדכן את המשקולות האיטיות.

באלגוריתם השני שהינו הצעת השיפור שלנו לאלגוריתם מהמאמר בחרנו את הפרמטרים: sync_period - מגדיר משתנה אשר קובע את התיזמון של האלגוריתם בהשתלבותו עם האופטימיזר אשר הוא משתמש בו. slow_step_size - מגדיר את היחס בו האלגוריתם יעדכן את המשקולות האיטיות.

באלגוריתם השלישי שהוא האופטימיזר adam הבסיסי שבחרנו בשביל ההשוואה לאלגוריתמים הנ"ל, בחרנו את הפרמטרים: learning_rate - מגדיר את קצב הלמידה של המודל, מה שיכול להשפיע על השונות ולכן מאוד חשוב. epsilon - מגדיר מספר קבוע השומר על יציבות המספרי של האימון.

```
if self.lookAhead_model:
    hp_innerOptimizer = trial.suggest_categorical("optimizer", ["adam", "SGD"])
    hp_slowStepSize = trial.suggest_float("slow_step_size", 0.1, 0.9, log=True)
    model.compile(loss=categorical_crossentropy,
                  optimizer=Lookahead(optimizer=hp_innerOptimizer, slow_step_size=hp_slowStepSize),
                  metrics=['accuracy'])

if self.improved_model:
    hp_syncPeriod = trial.suggest_int("sync_period", 1, 10)
    hp_slowStepSize = trial.suggest_float("slow_step_size", 0.1, 0.9, log=True)
    model.compile(loss=categorical_crossentropy,
                  optimizer=Lookahead(optimizer=Lookahead(optimizer='adam'), sync_period=hp_syncPeriod,
                                      slow_step_size=hp_slowStepSize), metrics=['accuracy'])

if self.simple_model:
    hp_learning_rate = trial.suggest_float("lr", 1e-5, 1e-1, log=True)
    hp_epsilon = trial.suggest_float("epsilon", 1e-7, 1e-3, log=True)
    model.compile(loss=categorical_crossentropy,
                  optimizer=Adam(learning_rate=hp_learning_rate, epsilon=hp_epsilon), metrics=['accuracy'])
```

מטריקות להערכת ביצועי המודלים:

על מנת להעריך את ביצועי המודלים ולחשב את המטריקות מצאנו שימוש בספרייה `tensorflow.keras.metrics` אשר סייעו לנו בשליפת המטריקות בצורה יעילה בביצוע `.model.evaluate`.

המטריקות שהתבקשנו לממש פונקציות שהשתמשנו:

- מטריקת Accuracy - על מנת לחשב את דיוק המודל השתמשנו במטריקה `'Accuracy'`.
- מטריקת TPR ומטריקת FPR - חישבנו את מדדי הTPR וFPR על ידי המטריקה `tensorflow.keras.metrics.sensitivityAtSpecificity` וב- `tensorflow.keras.metrics.specifyAtSensitivity` בהתאמה.
- מטריקת Precision - עבור מטריקת Precision השתמשנו ב- `tensorflow.keras.metrics.precision`.
- מטריקת AUC - עבור מטריקת הAUC השתמשנו ב- `tensorflow.keras.metrics.AUC`.
- מטריקת השטח מתחת לעקומת Precision-recall: עבור מטריקה זאת השתמשנו ב- `tensorflow.keras.metrics.AUC(curve='PR')`.
- זמן האימון - עבור חישוב זמן האימון מדדנו את הזמן העובר מתחילת תהליך הfit במודל ועד סופו.
- זמן הסקה עבור 1000 דוגמאות - בדומה לזמן האימון חישבנו את הזמן העובר מתחילת תהליך הpredict של המודל ועד לסיום הסקתו של המודל לגבי הסיווג של 1000 דוגמאות. במידה ולא היו קיימים מספיק דוגמאות בסט הבדיקה הכפלנו את הסט את שהגענו ל1000 דוגמאות או יותר.

ניתן לראות את כל המטריקות(חוץ מהמטריקות הקשורות לזמן הריצה) בזמן קומפילציית ובניית המודל.

```
if self.LookAhead_model:
    model.compile(loss=categorical_crossentropy,
                  optimizer=Lookahead(optimizer=study.best_params['optimizer'],
                                       slow_step_size=study.best_params['slow_step_size']),
                  metrics=['accuracy', tf.keras.metrics.SensitivityAtSpecificity(0.5),
                           tf.keras.metrics.SpecifyAtSensitivity(0.5), tf.keras.metrics.Precision(),
                           tf.keras.metrics.AUC(), tf.keras.metrics.AUC(curve='PR')])

if self.improved_model:
    model.compile(loss=categorical_crossentropy,
                  optimizer=Lookahead(optimizer='adam',
                                       sync_period=study.best_params['sync_period'],
                                       slow_step_size=study.best_params['slow_step_size']),
                  metrics=['accuracy', tf.keras.metrics.SensitivityAtSpecificity(0.5),
                           tf.keras.metrics.SpecifyAtSensitivity(0.5), tf.keras.metrics.Precision(),
                           tf.keras.metrics.AUC(), tf.keras.metrics.AUC(curve='PR')])

if self.simple_model:
    model.compile(loss=categorical_crossentropy,
                  optimizer=Adam(learning_rate=study.best_params['lr'],
                                epsilon=study.best_params['epsilon']),
                  metrics=['accuracy', tf.keras.metrics.SensitivityAtSpecificity(0.5),
                           tf.keras.metrics.SpecifyAtSensitivity(0.5), tf.keras.metrics.Precision(),
                           tf.keras.metrics.AUC(), tf.keras.metrics.AUC(curve='PR')])
```

Stage 5- מבחני מובהקות סטטיסטית- מבחן Friedman ומבחן Post-hoc:

לאחר הרצת המודלים, רצינו לבחון אם קיים הבדל מובהק בין האלגוריתמים, או שהם מספיק זהים על מנת לקבוע שאין הבדל. לצורך בדיקה זאת ביצענו את מבחן Friedman על התוצאות כאשר המדד שבחרנו להשוואה הוא מדד AUC. על מנת לבצע את מבחן פרידמן השתמשנו ב-friedmanchisquare מהספרייה stats של scipy. הסטטיסטי שקיבלנו מתוצאות המבחן הוא 33.25 וה-p-value שקיבלנו הוא 0.002.

```
FriedmanchisquareResult(statistic=33.25, pvalue=6.023573837886477e-08)
```

מכיוון שה-p-value קטן מ-0.05, דחינו את השערת האפס והגענו למסקנה שאכן יש הבדל בין האלגוריתמים ולכן נמשיך למבחן Post-hoc.

עבור מבחן Post-hoc השתמשנו בפונקציה posthoc_nemenyi_friedman מהספרייה scikit_posthocs אשר מקבלת את התוצאות ממבחן פרידמן ומוציאה כפלט מטריצה אשר מציגה את ההבדלים בין הקבוצות.

	0	1	2
0	1.000000	0.00100	0.033252
1	0.001000	1.000000	0.003310
2	0.033252	0.00331	1.000000

קבוצה מס' 0- אלגוריתם lookahead ללא השיפור.
קבוצה מס' 1-אלגוריתם lookahead עם השיפור.
קבוצה מס' 2-אלגוריתם adam המוכר.
ככל שתוצאת החיתוך בין האלגוריתמים גבוהה יותר כך הם יותר דומים.

	lookahead	improved lookahead	adam
lookahead	1	0.001	0.033252
improved lookahead	0.001	1	0.00331
adam	0.033252	0.00331	1

לפי התוצאות ניתן לראות שקיימים הבדלים מהותיים בין כל האלגוריתמים, אך ההבדל הגדול ביותר קיים בין האלגוריתם lookahead לפני השיפור, והאלגוריתם אחרי השיפור. מתוצאות אלו ניתן להסיק מספר דברים:

- lookahead שונה מהאלגוריתם המוכר.
- השינוי שביצענו באלגוריתם lookahead הוא משמעותי, בין אם זה משפר את הביצועים ובין אם מוריד אותם.
- אלגוריתם lookahead הכי דומה לאלגוריתם adam מקבוצת האלגוריתמים שבחנו.

Stage 6 - מסקנות:

- במהלך הניסוי ביצענו השוואה בין 3 אלגוריתמים שונים.
- האלגוריתם הראשון היה ממומש בהשראת וברעיון המאמר **"Lookahead Optimizer: k steps forward, 1 step back"**.
- האלגוריתם השני היה הצעת השיפור שלנו שבה השתמשנו באופטימיזר שלקוח מהמאמר כאופטימיזר אשר באמצעותו אופטימיזר LookAhead יבצע את העידכון למשקולות האיטיות.
- האלגוריתם השלישי שבחרנו לשם השוואה הינו אופטימיזר 'Adam' אשר הינו מוכר ויעיל מאוד בעבודתו עם רשתות נוירונים.
- לאחר הרצה של הניסוי בו עבדנו עם 20 דאטסטים שונים בכל אלגוריתם וניתוח מטריקות הביצועים לכל דטסט הגענו למסקנה כי ברוב הפעמים השינוי שבוצע באלגוריתם אכן משפר את התוצאות. בנוסף ראינו שהאלגוריתם המספק ברוב המקרים את התוצאות הכי פחות טובות הוא אלגוריתם adam, למרות שהוא האלגוריתם הכי מוכר.
- כפי שנאמר בתחילת הפרויקט, נשאלת השאלה האם אלגוריתם lookahead יכול לשפר את עצמו. מהתוצאות שקיבלנו התשובה היא חיובית, ופותחת אפשרות למחקר עד כמה רבדים של אופטימיזציה ניתן להוסיף עד שהשיפור לא יהיה מובהק.
- מצורף pseudo codes של האלגוריתמים, ניתן לראות כי הפסאודו קוד של האלגוריתם עם השיפור וללא השיפור הם זהים, מכיוון שהרעיון עצמו הוא זהה, אך המחשב מאחורי האופטימיזר הפנימי שאותו הוא מקבל היא שונה ולכן התוצאה הסופית שונה גם כן.

פסאודו קוד עבור lookahead ועבור השינוי:

[1]

Algorithm 1 Lookahead Optimizer:

Require: Initial parameters ϕ_0 , objective function L
Require: Synchronization period k , slow weights step size α , optimizer A
for $t = 1, 2, \dots$ **do**
 Synchronize parameters $\theta_{t,0} \leftarrow \phi_{t-1}$
 for $i = 1, 2, \dots, k$ **do**
 sample minibatch of data $d \sim \mathcal{D}$
 $\theta_{t,i} \leftarrow \theta_{t,i-1} + A(L, \theta_{t,i-1}, d)$
 end for
 Perform outer update $\phi_t \leftarrow \phi_{t-1} + \alpha(\theta_{t,k} - \phi_{t-1})$
end for
return parameters ϕ

פסאודו קוד עבור אופטימיזר adam:

[6]

Require: α : Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0 : Initial parameter vector
 $m_0 \leftarrow 0$ (Initialize 1st moment vector)
 $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
 $t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
 $t \leftarrow t + 1$
 $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
 $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
 $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
 $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
 $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
 $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

References:

קישור לגיט שבו קיים הקוד ותוצאות הניסוי בקובץ CSV -

<https://github.com/almogs575/Final-Project-Applied-Computational-Learning.git>

[1]

<https://arxiv.org/pdf/1907.08610.pdf>

[2]

https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam

[3]

<https://nuk.manisaskincare.pw/adam-optimizer-tensorflow.html>

[4] <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/> Adam oprimizer

[5] <https://optuna.org/>

[6] <https://arxiv.org/pdf/1412.6980.pdf>