אלמוג סרפיאן – 305069577
אביאל אביטן - 205598089

דו"ח פרויקט מסכם

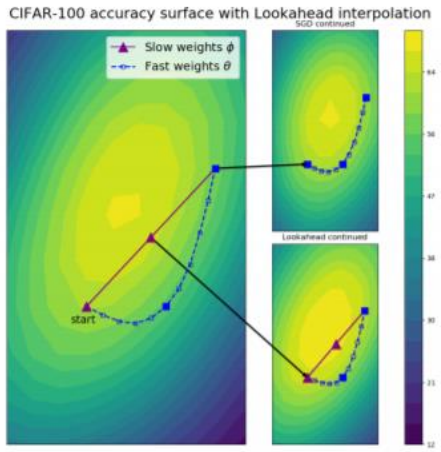
Stage 1 - בחירת אלגוריתם להערכה:

מודל מס' 1- **Lookahead Optimizer**:
בפרויקט המסכם החלטנו להתמקד באופטימייזר lookahead.
אופטימייזר lookahead היא שיטת אופטימיזציה שמשפרת את הביצועים של אופטימייזרים אחרים, שעליה היא בנויה. אנחנו בחרנו להתמקד במאמר **"Lookahead Optimizer: k steps forward, 1 step back"**
אשר מציג ניסויים ומימושים משלו של האופטימייזר. ניתן לראות במאמר שמבחינת ביצועים היה שיפור משמעותי מאשר מבשימוש באופטימייזר רגיל, והתוצאות מובהקות. בפרויקט אנחנו בחרנו להשתמש במימוש של Tensorflow לLookahead optimizer ובחרנו בו כמודל הראשון להשוואה.

ל-lookahead קיימים השלבים הבאים:

1. האופטימייזר מעדכן בצורה איטרטיבית שני סטים של משקלים.
2. האופטימייזר בוחר אופטימייזר פנימי.
3. הוא מעדכן את כיווני החיפוש במשקלים שנבחרו על יד האופטימייזר הפנימי.
4. הוא מעדכן את המשקלים האיטיים בכל k צעדים בהתבסס על כיוון המשקלים המהירים ושני הסטים של המשקולות המסוכנרכנים. בצורה כזאת השיטה משפרת את יציבות הלמידה ומורידה את השונות של האופטימייזר 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 $t = 1, 2, \ldots$ **do**
  Synchronize parameters $\theta_{t,0} \leftarrow \phi_{t-1}$
  for $i = 1, 2, \ldots, k$ **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

אלמוג סרפיאן – 305069577
אביאל אביטן - 205598089

לאלגוריתם lookahead קיימים מספר יתרונות וחסרונות:

יתרונות:

- הוא משפר את יציבות הלמידה.
- מקטין את השונות של האופטימייזר הפנימי בעזרת שימוש בזיכרון זניח על מנת לעשות זאת.
- משפר את הrobustness של האופטימייזר הפנימי.
- מאפשר להגיע להתכנסות מהירה יותר של אלגוריתם הלמידה.
- מראה הבטחה גדולה והוכחות אמפיריות על המון סוגים של דטסטים, בנצ'מרקים, וארכיטקטורות שונות.

חיסרונות:

- שימוש באופטימייזר אשר לא מסתנכרן טוב עם הlookahead יכול לגרום לביצועים רעים, ואף גרועים יותר ממה שהיו.
- מכיוון שהlookahead משתמש באופטימייזר פנימי, הוא איטי יותר מאופטימייזרים 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, אופטימייזר lookahead מייעל ומשפר את אלגוריתמי האופטימייזרים הפנימיים שבהם הוא משתמש. נכון לעכשיו האלגוריתם משתמש באופטימייזר פנימי אחד אשר עוזר לו לקבוע את המשקלים על ידי שימוש בו.
נשאלת השאלה האם הlookahead יכול בעצם לשפר את עצמו על ידי שימוש בlookahead כאופטימייזר פנימי.

מהלך הניסוי שאנחנו מציעים הוא:

1. יצירת אופטימייזר lookahead רגיל.
2. השמת אופטימייזר פנימי מוכר שהאופטימייזר בשלב מס' 1 יקבל.
3. יצירת אופטימייזר lookahead חדש אשר בו נשתמש על מנת לאמן את המודל שלנו.
4. לאופטימייזר שיצרנו בשלב מס' 3 ניתן את האופטימייזר משלב מס' 1 כקלט לשימוש כאופטימייזר פנימי.

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

אלמוג סרפיאן – 305069577
אביאל אביטן - 205598089

את השראת הרעיון שאבנו ממילות הסיכום של המאמר [1] **"Lookahead Optimizer: k steps forward, 1 step back"** אשר אומרות " 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 האופטימייזר אמור לעבוד עם כל אופטימייזר סטנדרטי, ולכן אין סיבה שלא יעבוד גם עם עצמו.

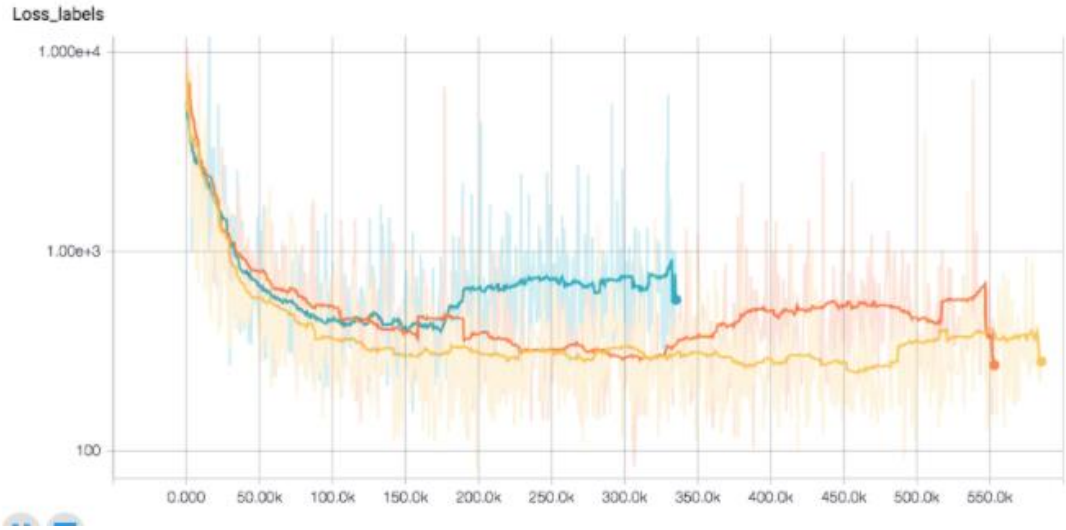
Stage 3 - בחירת אלגוריתם מוכר להשוואה:

מודל מס' 3-(**Adam optimizer(baseline_model**:
מבחינת אלגוריתם אופטימיזציה מוכר, בחרנו ב-Adam optimizer מפלטפורמת Tensorflow ו-keras. **[2]**
אדם משלב את המאפיינים הטובים ביותר של אלגוריתמי AdaGrad ו- RMSProp כדי לספק אלגוריתם אופטימיזציה שיכול להתמודד עם שיפועים דלילים בבעיות רועשות. **[4]**
אופטימייזר Adam מתחשב בהיסטוריה של ה-learning rate הקודם ובמידה ויש כמה learning rates הוא יתייחס אליהם בעזרת משקלים ממושקלים, כך שככל 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]



אלמוג סרפיאן – 305069577
אביאל אביטן - 205598089

Stage 4 - הערכה של האלגוריתמים שנבחרו:

הרצת האלגוריתמים:

- על מנת להשתמש באלגוריתמי האופטימייזרים הנ"ל יצרו רשת נוירונים פשוטה עם 2 שכבות קונבולוציה שאחריהן מגיעה שכבת pooling המצמצמות 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-categorial_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 (MaxPooling2 (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 והאופטימייזרים:
מבחינת המודל שבחרנו, השתמשנו במודל Sequential מסוג CNN אשר ייבאנו מהספרייה tensorflow.keras.model.

אופטימייזר הLookahead:
עבור אופטימייזר הLookahead ייבאנו את הגרסה העדכנית מהספרייה tensorflow_addons.optimizers והשתמשנו בה גם למודל לפני השינוי, וגם למודל אחרי השינוי.

אלמוג סרפיאן – 305069577
אביאל אביטן - 205598089

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

<u>הדאטסטים</u>:

- 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.
- בהרצת האימון השתמשנו ב-to_categorial(one hot encoding) על ה-y_train וה-y_test על מנת לשנות את התצורה של המערכים על מנת שביצוע האימון יהיה יותר יעיל.
- הדטסטים שהשתמשנו בעבודה נלקחו מהספרייה tensorflow_datasets וביצענו טעינה שלהם אל מערכים של numpy. **[7]**
- מהדאטסטים הגדולים לקחו 1000 דוגמאות לאימון ו500 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)**

אלמוג סרפיאן – 305069577
אביאל אביטן - 205598089

פרוטוקול ההערכה- **Cross validation**:

Cross validation הינה שיטת ולידציה אשר מחלקת את הנתונים בצורה אקראית לחלקים שווים(k) כך שבכל פעם האימון מבצע בדיקה על חלק אחר.
התבקשנו בעבודה להשתמש בקרוס ולידיישן חיצוני בעל 10 פולדים ולבצע אימון עם הפרמטרים הטובים לאחר שביצענו היפרפרמטרים בתוך קרוס ולידיישן פנימי בעל 3 פולדים.
בעבודה השתמשנו באובייקט KFold הלקוח מהספרייה sklearn **[8]**
דוגמא מהקוד למימוש הcross validation:

```
inter_kfold = KFold(n_splits=3, shuffle=True)

# K-fold Cross Validation model evaluation
acc_per_fold = []

for train_in, test_in in inter_kfold.split(self.X_train, self.y_train):
    model.fit(self.X_train[train_in], to_categorical(self.y_train[train_in]),
              validation_data=(self.X_train[test_in], to_categorical(self.y_train[test_in])), batch_size=128,
              epochs=3)

    scores = model.evaluate(self.X_test, to_categorical(self.y_test), verbose=0)
    acc_per_fold.append(scores[1] * 100)

return mean(acc_per_fold)
```

אופטימיזציית **Hyperparameters**:

את ההיפרפרמטרים ביצענו באמצעות ספריית **optuna [5]** הספרייה מאפשר ביצוע של היפרפרמטרים על רשתות נוירונים של keras, השכבות והפונקציות השונות שהן כוללות.
בעבודתנו ביצענו את ההיפרפרמטרים לאופטימיזרים השונים בכל אלגוריתם בשימוש של 3 קרוס ולידיישן, כך שבסופם החזרנו את הממוצע (של אותו טריאל) של דיוק המודל באמצעות ערכי הפרמטרים שנבדקו ולבסוף לאחר הרצה של 50 טריאלס החזרנו את הערכים הטובים ביותר לכל הפרמטרים שהניבו את התוצאות הטובות ביותר.
על מנת להשתמש בספריה נאלצנו לממש את פונקציית objective. בפונקציה זו הגדרנו את המודל והגדרנו טווחי מקסימום ומינימום או ערכים שונים לפרמטרים שביצענו עליהם אופטימיזציה. הפונקציה אשר מפעילה את objective הינה study. ניתן להגדיר את study לפי מטרת הניסוי, כך שנוכל להגדיר אם מטרתנו היא מיקסום התוצאות או להביא למינימום האפשרי.

באלגוריתם הראשון שנלקח מתוך המאמר lookAhead- החלטנו לחפש את הפרמטרים:
optimizer - אשר מגדיר את האופטיימזר המקורי בו נעשה שימוש לחישוב ומימוש הגרדיאטס.
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 האופטימייזר 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 **[9]** 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.specifityAtSensitivity בהתאמה.
- מטריקת Precision- עבור מטריקת Precision השתמשנו ב- tensorflow.keras.metrics.precision.

אלמוג סרפיאן – 305069577
אביאל אביטן - 205598089

- 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.SpecificityAtSensitivity(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=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.SpecificityAtSensitivity(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.SpecificityAtSensitivity(0.5), tf.keras.metrics.Precision(),
                           tf.keras.metrics.AUC(), tf.keras.metrics.AUC(curve='PR')])
```

אלמוג סרפיאן – 305069577
אביאל אביטן - 205598089

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 והp-value שקיבלנו הוא 0.002.

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

מכיוון שה p-value קטן מ0.05, 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 מהספריה scikit_posthocs אשר מקבלת את התוצאות ממבחן פרידמן ומוציאה כפלט מטריצה של p-values אשר כל תוצאה שלה מייצגת את הp-value של קבוצה מסוימת להיות שונה מקבוצה אחרת. הכוונה היא, שבמידה והערך קטן מהאלפא הקבוצות שונות.
בניסוי שלנו בחרנו את אלפא להיות 0.05 ולכן ניתן לראות כי כל הקבוצות שונות סטטיסטית אחת מהשניה.

	0	1	2
0	1.000000	0.00100	0.033252
1	0.001000	1.00000	0.003310
2	0.033252	0.00331	1.000000

דוגמא מהקוד למבחן פרידמן ולמבחן פוסט הוק:

```
def friedman_test(self):
    self.create_AUC_list("results.csv")
    results = stats.friedmanchisquare(self.auc_adam, self.auc_lookAhead, self.auc_improved_lookAhead)
    print(results)
    if results[1] < 0.05:
        return False
    return True

def hoc_test(self):
    data = np.array([self.auc_lookAhead, self.auc_adam, self.auc_improved_lookAhead])
    hoc = sp.posthoc_nemenyi_friedman(data)
    print(hoc)
```

אלמוג סרפיאן – 305069577
אביאל אביטן - 205598089

קבוצה מס' 0- אלגוריתם הlookahead ללא השיפור.
קבוצה מס' 1-אלגוריתם הlookahead 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:

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

אלמוג סרפיאן – 305069577
אביאל אביטן - 205598089

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 באופטימייזר שלקוח מהמאמר כאופטימייזר אשר באמצעותו אופטימייזר 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 המחשבה מאחורי האופטימייזר הפנימי שאותו הוא מקבל היא שונה ולכן התוצאה הסופית שונה גם כן.

אלמוג סרפיאן – 305069577
אביאל אביטן - 205598089

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 $t = 1, 2, \ldots$ **do**
  Synchronize parameters $\theta_{t,0} \leftarrow \phi_{t-1}$
  for $i = 1, 2, \ldots, k$ **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 אופטימייזר 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)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

לסיכום, בפרויקט הכרנו את אופטימייזר lookahead אשר משפר ביצועים של אופטימייזרים אחרים עליהם הוא נבנה. הlookahead אופטימייזר יכול לשפר גם את התוצאות של עצמו ברוב המקרים אך בחלק מהמקרים הביצועים יכולים גם לרדת. ניתן לבצע עבודה עתידית ולנסות לשפר את האלגוריתם עוד יותר על ידי הוספה של עוד אופטימייזרים פנימיים ולראות מתי התוצאות מתכנסות לכדי תוצאות זהות. השתמשנו באלגוריתם אדם לאופטימציה על מנת באמת להשוות את התוצאות, וראינו שאף אחד מהאלגוריתמים אינו דומה לשני.
ניתן לראות את תוצאות המודלים ושל כל אופטימייזר בקובץ CSV אשר נמצא בפרויקט.

References:

קישור לגיט שבו קיים הקוד ותוצאות הניסוי בקובץ **CSV** –

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

[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/**

[5] **https://optuna.org/**

[6] **https://arxiv.org/pdf/1412.6980.pdf**

[7] https://www.tensorflow.org/datasets/catalog/overview

[8] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

[9] https://www.tensorflow.org/api_docs/python/tf/keras/metrics