Keras

## הקדמה

## התקנה ויבוא

ראשית יש להתקין את הספרייה באמצעות הפקודה הבאה. הספרייה די כבדה ולכן התקנה לוקחת קצת זמן.

Pip install tensorflow

לאחר מכן יש לייבא את הספרייה, בדרך כלל כ-tf.

import tensorflow as tf

print("TensorFlow version:", tf.\_\_version\_\_)

מודל

כדי ליצור מודל מאומן ב-Keras שניתן להריץ עליו אובייקטים ולקבל חיזוי, יש לבצע חמש פעולות:

1. יצירה – הגדרת רצף השכבות במודל.
2. קמפול – הגדרת פונקציית loss, אופטימייזר ומה רוצים למקסם.
3. אימון – נאמן את המודל באמצעות הדאטה.
4. הערכה – באמצעות השוואה לקבוצת ה-test.
5. חיזוי – נתינת אובייקט חדש וקבלת חיזוי.

## יצירת מודל

כדי ליצור מודל שהוא שרשור של שכבות נשתמש בפונקציה tf.keras.Sequential המקבלת מערך של שכבות ומחזירה מודל.

model = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=(28, 28)),

tf.keras.layers.Dense(128, activation='relu'),

tf.keras.layers.Dense(10)

])

## קמפול

באמצעות הפעלת הפונקציה compile על המודל ניתן להגדיר למודל את הרכיבים הבאים:

* אופטימייזר - באיזה שיטה לעדכן את הפרמטרים של המודל על סמך הנתונים שהוא רואה פונקציית ה-loss. יכול להיות מחרוזת עם שם האופטימייזר, או מופע של מחלקת optimizer. ניתן לראות את הסוגים השונים [בקישור](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers). מאפשרים גם להגדיר בתוכו קצב למידה ופרמטרים נוספים.
* פונקציית loss - מודד עד כמה המודל מדויק במהלך האימון. אנו רוצים מינימום של הפונקציה הזו כדי לכוון את המודל בכיוון הנכון. יכול להיות מחרוזת עם שם הפונקציית loss, או מופע של מחלקת losses. ניתן לראות את הסוגים השונים [בקישור](https://www.tensorflow.org/api_docs/python/tf/keras/losses). ניתן גם ליצור פונקציה משלנו מהצורה fn(y\_true,y\_pred).
* מדדים – רשימה של מדדים להערכת המודל במהלך אימון ובדיקות. בדרך כלל accuracy. יכולים להיות מחרוזת עם שם המדד, או מופע של מחלקת metrics. ניתן לראות את הסוגים השונים [בקישור](https://www.tensorflow.org/api_docs/python/tf/keras/metrics).

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=1e-3),

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=[tf.keras.metrics.BinaryAccuracy(),

tf.keras.metrics.FalseNegatives()])

)

## אימון

באמצעות הפעלת הפונקציה fit נאמן את המודל. במהלך אימון המודל יודפסו ה-loss וה-accuracy של המודל על ה-train עבור כל epoch. פונקציה זו מקבלת את הארגומנטים הבאים:

* x – קבוצת המידע עליו המודל מתאמן.
* y – התיוג של קבוצת האימון.
* validation\_data – קבוצת ה-validation. במידה וערך זה לא None יודפס גם ה-loss וה-accuracy על ה-validation.
* batch\_size - מספר שלם או None. מייצג את מספר הדגימות בכל עדכון הגרדיאנט. אם לא צוין, ברירת המחדל 32.
* epochs – מספר האיטרציות שעוברים על כל הדאטה כדי לאמן את המודל.
* verbose – כיצד להציג את הנתונים. 0 – מצב שקט. 1 – בר התקדמות, ברירת המחדל. 2 – מדפיס שורה אחר שורה.

History = model.fit(train\_images, train\_labels, epochs=10)

פונקציית fit מחזירה אובייקט מסוג History, שהתכונה history שלו מחזירה מילון שבו המפתחות הם מדדי loss ו-accuracy על ה-train וה-validation. הערכים הם וקטורים המכילים את כל הנתונים הרלוונטיים, מסודרים לפי סדר ה-epochs.

history\_dict = history.history

acc = history\_dict['binary\_accuracy']

loss = history\_dict['loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss, 'bo', label='loss')

plt.plot(epochs, acc, 'bo', label='acc')

plt.show()

## הערכה

באמצעות הפונקציה evaluate על המודל נוכל להעריך אותו. הפונקציה מקבלת את קבוצת ה-test ואת התיוג שלה, ומחזירה את הערך הסופי של הפונקציית loss ואת המדדים שהכנסנו למודל במהלך הקמפול.

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)

מקבלת את הארגומנטים הבאים:

* x – קבוצת המידע שעליה נבחן ונעריך את המודל. יכול להיות מערך numpy, tensor, או מילון.
* y – התיוג של קבוצת הטסט.
* batch\_size – גודל ה-batch שבו הנתונים מ-x יכנסו למודל.
* verbose – כיצד להציג את הנתונים. 0 – מצב שקט. 1 – בר התקדמות, ברירת המחדל. 2 – מדפיס שורה אחר שורה.

## חיזוי

נפעיל את הפונקציה predict על המודל כדי לקבל חיזוי עבור קבוצת אובייקטים. הפונקציה מקבלת מערך של אובייקטים x ומחזירה מערך numpy של החיזויים לכל הקלטים.

במקרה שהמודל צריך לסווג אובייקט ליותר משני מחלקות הוא מחזיר logits לכל מחלקה, ולכן יש להפעיל שכבת softmax כדי להמיר את ה-logits להסתברויות.

probability\_model = tf.keras.Sequential([model,

tf.keras.layers.Softmax()])

predictions = probability\_model.predict(test\_images)

np.argmax(predictions[0])

## פונקציות על המודל

לאחר יצירת אובייקט המודל ישנם כל מיני פונקציות שימושיות שניתן להפעיל עליו:

* summary – מדפיס טבלת סיכום של כל השכבות במודל, מספר הפרמטרים בכל שכבה, ומימד הפלט.

## שמירה וטעינה של מודל

ניתן לשמור מודל באמצעות פונקציית save.

model.save('model')

ניתן לטעון מודל באמצעות פונקציית tf.keras.models.load\_models.

reloaded = tf.keras.models.load\_model('model')

# עיבוד DATA

## העלאת קובץ

### באמצעות כתובת UML

להוריד קובץ באמצעות כתובת URL, אם אינו נמצא כבר ב-cache נשתמש בפונקציה tf.keras.utils.get\_file. הפונקציה מחזירה Path לקובץ שהורדנו. הארגומנטים של פונקציה זו הם:

* Fname - שם הקובץ. אם צוין כל ה-path בערך זה, לדוגמה /path/to/file.txt, הקובץ יישמר במיקום זה. אם שדה זה ריק, השם של הקובץ יהיה לפי הערך ב-origin.
* origin – כתובת url של הקובץ.
* extract – ערך בוליאני. אם True, מנסה לחלץ את הקובץ כארכיון, כמו tar או zip.
* cache\_subdir - מיקום לאחסון הקבצים. ברירת המחדל היא ~/.keras/.

tf.keras.utils.get\_file(

fname=None,

origin=None,

cache\_subdir='datasets',

extract=False,

…

)

## יצירת Dataset מקבצי טקסט

באמצעות הפונקציה [text\_dataset\_from\_directory](https://www.tensorflow.org/api_docs/python/tf/keras/utils/text_dataset_from_directory) ניתן לטעון נתונים מהדיסק ולהכין אותם לפורמט המתאים לאימון. הפונקציה שימושית כאשר הדאטה מסודר כך:

main\_directory/

class\_a/

a\_text\_1.txt

a\_text\_2.txt

class\_b/

b\_text\_1.txt

b\_text\_2.txt

קריאה לפונקציה text\_dataset\_from\_directory(main\_directory, …) מחזירה אובייקט tf.data.Dataset המכיל קבוצות של טקסטים מהתיקיות class\_a ו-class\_b, יחד עם התוויות 0 ו-1 (0 ל-class\_a ו-1 ל-class\_b). בנוסף מודיעה כמה אובייקטים ומחלקות נמצאו. בשלב זה רק קבצי txt נתמכים.

הארגומנטים של פונקציה זו הם:

* + - directory – path למיקום של ה-data.
    - batch\_size - מספר שלם או None. מייצג מספר הדגימות בכל עדכון של הגרדיאנט. אם לא צוין ברירת המחדל 32.
      * validation\_split – כמה אחוז מה-train ישמש לקבוצת ה-validation.
    - subset – משמש רק אם validation\_split מוגדר. מגדיר האם האובייקט המוחזר הוא “training” או “validation”.
    - shuffle - האם לערבב את הנתונים. ברירת מחדל היא true. אם מוגדר כ-false, ממיין את הנתונים בסדר אלפאנומרי.
    - seed – ערך אקראי עבור ערבוב הדאטה.

בעת שימוש בארגומנטים validation\_split ו-subset, צריך לציין סיד אקראי, או לציין shuffle=False, כך שלפיצול לא יהיה חפיפה.

raw\_train\_ds = tf.keras.preprocessing.text\_dataset\_from\_directory(

'data/train',

validation\_split=0.2,

subset='training',

seed=42)

raw\_val\_ds = tf.keras.preprocessing.text\_dataset\_from\_directory(

'data/train',

validation\_split=0.2,

subset='validation',

seed=42)

## נרמול דאטה

ניתן להוסיף שכבה למודל שמטרתה לנרמל את הדאטה שנכנס אליה.

normalizer = tf.keras.layers.Normalization()

normalizer.adapt(np.array(train\_features))

# שכבות

אבן הבניין הבסיסית של רשת נוירונים היא השכבה. רוב הלמידה העמוקה מורכבת משרשור שכבות. בראש התוכנית יש לייבא את השכבות באמצעות הפקודה:

from tensorflow.keras import layers

### Embedding

ניתן להשתמש בשכבה זו רק כשכבה ראשונה במודל. לוקחת כל ערך בדאטה והופכת אותם לווקטור הטמעה בגודל קבוע. וקטורים אלה נלמדים כאשר המודל מתאמן. שכבה זו מוסיפה מימד לדאטה, אם מימד הדאטה בקלט היה אזי מימד הדאטה בפלט הוא . שכבה זו מקבלת את הארגומנטים הבאים:

* input\_dim - גודל אוצר המילים של האינדקסים + ​​1.
* output\_dim – גודל הוקטור המוטמע.
* input\_length - אורך כל אובייקט בקלט, כאשר הוא קבוע. ארגומנט זה נדרש אם מתכוונים לחבר שכבות Flatten ואז Dense (אחרת לא ניתן לחשב את גודל הוקטורים המוטמעים).

### Flatten

הופך את הקלט ממערך רב ממדי למערך חד ממדי. לשכבה זו אין פרמטרים ללמוד, אלא רק מעצבת מחדש את הנתונים.

### Dense

מייצגת שכבת fully connected. מקבלת את:

* יחידות: מספר הנוירונים בפלט של שכבה זו.
* פונקציית אקטיבציה. לדוגמה 'relu'.
* פרמטרים נוספים.

tf.keras.layers.Dense(

units,

activation=None,

…

)

### Dropout

שכבת ה-Dropout מאפסת אחוז מסוים מהקלטים משכבה קודמת באופן אקראי. אחוז הקלטים שיתאפסו מתקבל כארגומנט לשכבה. שכבה זו עוזרת במניעת התאמת יתר. כניסות שאינן מוגדרות ל-0 מוכפלות ב, כך שהסכום על כל השכבה אינו משתנה. שכבת ה-Dropout פועלת רק כאשר האימון מוגדר ל-True כדי שלא יתאפסו ערכים במהלך שלב ה-test.

### Softmax

שכבת Softmax שאותה נשים אחרונה במודל קלסיפיקציה שיש בו יותר משתי מחלקות. הופכת את כל ה-logits שהמודל פלט להסתברויות שסכומן 1. משמעות כל ערך שנפלט משכבה זו היא מה ההסתברות שהקלט שייך לכל מחלקה i.

tf.keras.layers.Softmax(

axis=-1, \*\*kwargs

)

### GlobalAveragePooling1D

### TextVectorization

שכבה זו מאפשרת לבצע פעולות עיבוד בסיסיות על דאטה של מחרוזות. העיבוד של כל מחרוזת מכיל את השלבים הבאים:

* Standardization - בדרך כלל שינוי לאותיות קטנות והורדת סימני פיסוק.
* חלוקת כל דוגמה לתת-מחרוזות (בדרך כלל מילים).
* שילוב מחדש מחרוזות משנה ל-tokens (בדרך כלל N-grams)
* שיוך אינדקס ייחודי מסוג int לכל token.
* המרת כל מחרוזת לוקטור אינדקסים של ה-tokens שבו, או לוקטור המכיל ערכי float המייצגים מידע על ה-tokens.

מומלץ בשלב האימון להשתמש בשכבה זו על ה-data מחוץ למודל. ולבסוף, כדי להסיק חיזוי על מידע חדש, ליצור מודל חדש שמשלב בין שכבה זו למודל שאימנו.

ניתן לקרוא לשיטת adapt() על שכבה זו המקבלת dataset. פעולה זו תנתח את ה-dataset, תקבע את התדירות של ערכי מחרוזת בודדים ותיצור מהם 'אוצר מילים'. אוצר המילים הזה יכול להיות בגודל בלתי מוגבל או להיות מוגבל (לפי ערך max\_tokens). אם יש יותר ערכים ייחודיים בקלט מאשר גודל אוצר המילים המרבי, המונחים השכיחים ביותר ישמשו ליצירת אוצר המילים. חשוב להשתמש רק בקבוצת ה-train בקריאה ל-adapt מפני ששימוש ב-test יגרום להדלפת מידע.

צריך להזין לשכבה את הנתונים הבאים:

* max\_tokens - הגודל המרבי של אוצר המילים עבור שכבה זו.
* standardize – ברירת המחדל היא "lower\_and\_strip\_punctuation". ניתן גם להכניס אובייקט callable, כמו פונקציה, שמקבל ומחזיר מחרוזת. כל מחרוזת תישלח לאובייקט זה ותוחלף בתוצאה.
* split – כיצד לפצל כל מחרוזת. ברירת המחדל היא לפי “whitespaces”. ניתן גם להכניס אובייקט callable.
* ngrams – עבור מספר ייצור ngrams עד גודל מספר זה. אם none לא מפצל ל-N-grams וכל מילה היא token.
* output\_mode - מפרט אופציונלי לפלט של השכבה. הערכים יכולים להיות "int", "multi\_hot", "count" או "tf\_idf".
* output\_sequence\_length – אם בארגומנט קודם שלחנו int, אזי יגרום לשכבה לרפד או לקצץ אורך כל וקטור (המייצג מחרוזת) בדיוק לערך זה.

max\_features = 5000 # Maximum vocab size.

max\_len = 4 # Sequence length to pad the outputs to.

vectorize\_layer = tf.keras.layers.TextVectorization(

max\_tokens=max\_features,

output\_mode='int',

output\_sequence\_length=max\_len)

vectorize\_layer.adapt(text\_datasets)

באמצעות הפעלת השיטה get\_vocabulary() על אובייקט השכבה ניתן להחזיר מילון של כל האוצר מילים. במילון זה המפתחות הן האינדקסים והערכים הם ה-tokens.

# מודלים

## רגרסיה ליניארית

נבנה מודל פשוט של רגרסיה ליניארית, עם שכבת נורמליזציה.

normalizer = tf.keras.layers.Normalization(axis=-1)

normalizer.adapt(np.array(train\_features))

linear\_model = tf.keras.Sequential([

normalizer,

layers.Dense(units=1)

])

נקמפל את המודל עם אופטימייזר adam, ופונקציית loss של mae.

linear\_model.compile(

optimizer=tf.keras.optimizers.Adam(learning\_rate=0.1),

loss='mean\_absolute\_error'

)

נאמן את המודל.

history = linear\_model.fit(

train\_features,

train\_labels,

epochs=100,

verbose=0,

validation\_split = 0.2

)

נבנה גרף המציג את ה-loss על קבוצות ה-train ו-validation עם התקדמות האימון.

plt.plot(history.history['loss'], label='loss')

plt.plot(history.history['val\_loss'], label='val\_loss')

plt.ylim([0, 10])

plt.xlabel('Epoch')

plt.ylabel('Error [MPG]')

plt.legend()

plt.grid(True)

נעריך את איכות המודל על קבוצת ה-test.

test\_results['linear\_model'] = linear\_model.evaluate(

test\_features, test\_labels, verbose=0

)

## רשת נוירונים לבעיית רגרסיה