מבוא לבינה מלאכותית – תרגול 5

נושאים:

- **רקע: Pycharm, Anaconda**
- **בניית רשתות נוירונים ב- TensorFlow.**
- **נספח: הקוד שבו אני השתמשתי.**

Pycharm ו- Anaconda:

- Pycharm היא סביבת פיתוח (IDE) נוחה ונפוצה בשפת Python.
- כדי להוריד חבילות שונות של Python שנעשה בהן שימוש נרחב, נוריד Anaconda, שהיא הפצה שכוללת חבילות שונות (כמו Numpy, Pandas, Matplotlib, Scikit-learn ועוד הרבה) ומאפשרת להתקין מבחר נוסף של חבילות שזמינות ב- Anaconda Cloud.
- נניח שפתחנו פרויקט חדש ב-Pycharm ונרצה לעבוד בו בעזרת חבילות שאפשר למצוא ב-Anaconda. לצורך כך, יש להגדיר את ה- interpreter של הפרויקט להיות python.exe שנמצא בתוך תיקיית Anaconda

  (Settings -> Project: (name) -> Project Interpreter -> Show All -> + -> System Interpreter

  ואחר כך קבעו את ה-Interpreter כאמור). זה יאפשר לנו להשתמש בכל הספריות ש-Anaconda סיפקה לנו.
- אם נצטרך להוריד חבילה נוספת, אפשר גם כן לעשות זאת בהגדרות הפרויקט (או להשתמש ב-terminal ב-pip install או conda install):

  Settings -> Project: (name) -> Project Interpreter -> +. חפשו את שם הפרויקט (הגדירו גרסה אם תצטרכו) ולחצו על Install Package.
- למה שמוצג בתרגול זה, הייתי צריך להוריד את החבילה tensorflow.

בניית רשתות נוירונים בעזרת TensorFlow:

- אקדים ואומר שכנראה את מה שאני מסביר, באתר של tensorflow כנראה מסבירים טוב יותר.
- את מרבית הקוד לקחתי מתוך הדוגמה המופיעה שם.
- דבר חשוב שלא מופיע שם:

```
import os

os.environ["CUDA_VISIBLE_DEVICES"] = "-1"
```

זוהי פקודה שקובעת למחשב לא להתייחס להתקני CUDA.

מה זה אומר? CUDA היא ארכיטקטורה שמאפשרת להריץ דברים על ה-GPU.

מכיוון שגרסת ה-CUDA במחשב שלכם עשויה שלא להיות מתאימה ל- tensorflow, תרצו להריץ את מודל רשת הנוירונים שלכם על CPU בלבד, ומעשית פקודה זו דואגת לכך.

<u>מה עושים בדוגמה:</u>

קודם כל, מעלים dataset מובנה בחבילה, ובודקים מהו. זהו dataset של תמונות של פרטי לבוש שכל אחד מהם מסווג לאחד מבין 10 סוגים (חולצה, נעל וכו').

דואגים לסדר את הדאטא (שהגיע כמטריצה של פיקסלים עם ערכים בין 0 ל-255, ואנחנו נרצה לחלק ב-255 כדי להכניס לרשת שלנו ערכים בין 0 ל-1).

אחר כך, בונים את המודל שלנו: מאתחלים class לרשת הנוירונים - Sequential, ומוסיפים לה שכבות (ניתן להוסיף בדרך המופיעה בדוגמה או ע"י (model.add). בשכבת הקלט ממירים את התמונה (מטריצה דו-ממדית) לווקטור שורה כדי שאפשר יהיה להכניס אותו לרשת, ובשכבת הפלט קובעים שמספר הנוירונים יהיה מתאים לסיווג שנרצה.

במקרה הזה – 10 נוירונים בשביל 10 הסוגים. הפלט מכל נוירון המתאים לסוג יהיה המובהקות שהמודל נותן לכך שהדוגמה הנכנסת היא מהסוג הזה.

בוחרים פונקציית loss ו-optimizer בעזרתו נרצה לעדכן את משקלי הרשת (לדוגמה: Stochastic Gradient Descent).

מאמנים את הרשת לפי סט האימון שלנו, וקובעים כמה פעמים עוברים על כל הדאטא שלנו. כל מעבר כזה נקרא Epoch.
אפשר לבחור שלא לאמן את המודל לפי כל הדגימות ולהשאיר כמה בצד, שעליהן מודדים (כמו קבוצת test אבל באמצע תהליך האימון) את המדדים שנגדיר (ערך ה-loss וכן ערכים אחרים, שמוגדרים ב- metrics). **הערה:** בחבילה הזאת הקבוצה נקראת validation set. בשונה מהגדרות במקומות אחרים, כאן לא משתמשים בנקודות בקבוצה זו לשום מטרה פרט להערכה כמותית של הלמידה.

אחרי שאימנו, נבדוק את ה-loss שלנו על קבוצת ה-test.
הערה: בקישור ששלחתי, ממשיכים לחפור על התחזיות שיוצאות על ה-test ואיך הן מוצגות.

תוספת שלי שלא הופיעה בדוגמה: הוספתי validation כדי שאפשר יהיה לראות אם המודל אכן לומד והאם הוא ב-overfit. לאחר האימון והתחזית, ציירתי את המדדים שמדדנו כתלות ב-Epoch. תוצאות טיפוסיות ניתן למצוא בדוגמה אחרת.

נספח – הקוד אותו הרצתי בתרגול:

```
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras
# Helper libraries:
import matplotlib.pyplot as plt
import pandas as pd
import os

os.environ["CUDA_VISIBLE_DEVICES"] = "-1"  # Important, this way TF will
run only on CPU (My GPU is not up to date).


# Our dataset is a built-in dataset.
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) =
fashion_mnist.load_data()
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
print("The shape of the array of training data: ")
print(train_images.shape)  # Images of 28 x 28 pixels.
print("We have " + str(len(train_labels)) + " training samples")  # Train
labels are integers between 0 and 9.
print("We have " + str(len(test_labels)) + " test samples")


# The original set of images have pixel values between 0 and 255.
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()

# So we will normalize them:
train_images = train_images / 255.0
test_images = test_images / 255.0


# Okay, now let's see what labels we have.
# Here the colors are black and white because we set the colormap to be
binary
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.get_cmap('binary'))
    plt.xlabel(class_names[train_labels[i]])
plt.show()

######################
# After we understood the data, let's build the neural network.
# The NN we build is sequential.
# First, the input layer needs to be flattened - the input must be a vector
and not a 28 * 28 matrix.
# After that, come the hidden and the output layers. We define number of
neurons and activation functions.
```

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])

# Sequential() is a class. we can use model = Sequential() and use the
method model.add(layer).
# Another option I didn't use here: keras.layers.Dropout(dropout_rate).

# Let's compile the model - Set the optimizer, metrics and loss function.
# Metrics are used to monitor the training and testing steps. Here we use
accuracy,
# the fraction of the images that are correctly classified.

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# In order to see a summary for our model, we can use model.summary()

# NOTE: The options of activation functions, kinds of layers (we haven't
learned about most of them),
# optimizers, loss functions, metrics, regularizations and more are in
TensorFlow's API.

######################
# Good. Now, after we have a model, we can train it.

history = model.fit(train_images, train_labels, epochs=5,
validation_split=0.2)

# Now, let's use the trained model to see how accurate we are on the test:
# What we actually do here is predicting the confidence that every test
sample is labeled as every label.
# Then, the model's guesses label will be the argmax of the confidences.
# Using our guesses and the correct data, we can compute the loss nad the
accuracy.

test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)

######################

# Let's see how the model learns as a function of the epoch we are at:
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch

plt.figure()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.plot(hist['epoch'], hist['loss'],
         label='Train Loss')
plt.plot(hist['epoch'], hist['val_loss'],
         label='Val Loss')
plt.ylim([0, 2.8])
plt.legend()
plt.savefig("Loss_NN_TF")

plt.figure()
```

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.plot(hist['epoch'], hist['acc'],
         label='Train Accuracy')
plt.plot(hist['epoch'], hist['val_acc'],
         label='Val Accuracy')
plt.ylim([0, 1])
plt.legend()
plt.savefig("Accuracy_NN_TF")
```