The following are instructions how to run the code accompanying the paper: [Single particle diffusion](#) [characterization by deep learning.](#)

The network models and functions provided here are used to classify single particle anomalous diffusion trajectories and extract the Hurst exponent for Fractional Brownian Motion trajectories.

We hope that you find this code useful in your work.

Installation
- Install the Anaconda Python distribution: https://www.anaconda.com/distribution/
- Copy the provided *anomalous_diffusion_DL.yml* file to the Anaconda base folder (in Windows systems, should be *C:\Users\UserName*)
- Open Anaconda prompt from the start menu
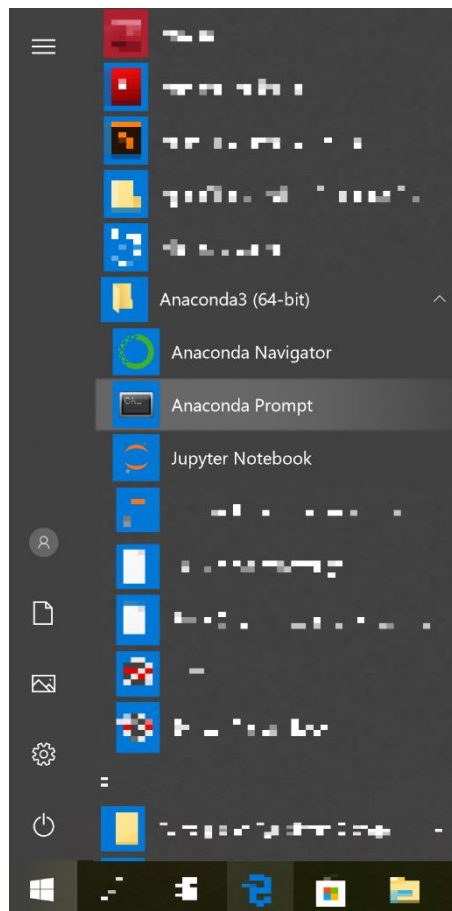
- Import the python environment from the provided yml file by entering the following in the command prompt:

```
conda env create -f anomalous_diffusion_DL.yml
```


Anaconda Prompt - conda env create -f anomalous_diffusion_DL.yml

```
(base) C:\Users\      >conda env create -f anomalous_diffusion_DL.yml
Solving environment: -
```

A new environment named anomDiffDL will be created once the process finishes.

- Activate the environment by entering:

```
conda activate anomDiffDL
```

- Install Keras into the new environment:

```
conda install -c anaconda keras-gpu
```

Note: this will install the GPU version of Keras, for the CPU version run:

```
conda install -c anaconda keras
```

- Install three more packages required for full functionality by entering the following in the same command prompt:

```
pip install matplotlib
pip install stochastic
pip install seaborn
```

Opening and editing files

- Anaconda automatically installs a Python IDE called Spyder. Every environment will have its own instance of Spyder with access to the libraries and packages of that environment.
- Open the anomDiffDL Spyder from the start menu
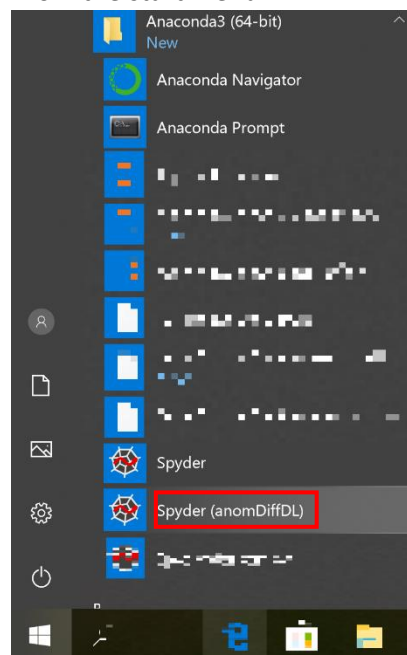
Training a network

- Open one of the files ending with _training.py
- In the top of the file, there are several variable definitions which control the simulated trajectories used to train the network.

```
11 batchsize = 64
12 T = np.arange(19,21,0.1) # this provides another layer of stochasticity to make the network more robust
13 steps = [500] # number of steps to generate in total
14 steps_actual = 300 # number of steps the network recieves as input out of the number of steps available
```

- batchsize is the number of training samples generated in each instance of the data generator

- T is the total time of the simulation in seconds. The data generator expects to receive an array of T values from which it randomly selects a value in each iteration. This is done to increase the variability in the generated trajectories.
- steps is the number of steps in the simulated trajectory assuming a constant Δt between time points. Similar to T, the data generator expects to receive an array of possible values from which is randomly selects one in each iteration. (When choosing value options for T and steps, please keep the options relatively close to allow the network to train properly)
- steps_actual is the number of steps the network sees out of the total trajectory length and is the trajectory length the network expects to receive. (This value should always be smaller than the minimal value in steps).

- Choose a file name for the new network model in the bottom of the script (marked in red below)

```
74          ModelCheckpoint(filepath='new_model.h5',
75                          monitor='val_loss',
76                          save_best_only=True,
77                          mode='min',
78                          save_weights_only=False)]
```

- Once all definitions and parameters have been set, run the script to generate a new network model.

Analyzing data from a file

- Open and run utils.py to load auxiliary functions to memory
- You can run all analysis functions as is with the network models provided, or replace the loaded model by changing the file name where the following comment is found anywhere in the code:
  ```
  ### change here to load a different network model
  ```
- Open and run the appropriate testing file containing the function you want to run
- All file analysis functions receive as input a .mat file, containing one matrix of N dimensional data in the following configuration:

x	y	N

Where x,y represent x,y positions in pixels and N is the trajectory serial number in ascending order, starting from 1 (i.e. all x,y values belonging to trajectory 1 will have N=1 in the third column) See example in the following image:
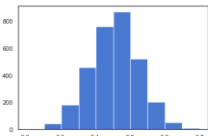
44916x3 double

	1	2	3
1	-27.2493	-59.1920	1
2	-27.6756	-58.8687	1
3	-27.6936	-59.2070	1
4	-28.0118	-59.4451	1
5	-28.0240	-59.2436	1
6	-28.1894	-59.4752	1
7	-28.3665	-59.4824	1
8	-28.2087	-59.0561	1
9	-27.8874	-59.2675	1
10	-27.7105	-59.1068	1
11	-27.4111	-59.0776	1
12	-27.5374	-59.3351	1
13	-27.3369	-59.2576	1
14	-27.8577	-58.8687	1
15	-28.0479	-59.0715	1
16	-28.3585	-59.0283	1
17	-28.2769	-59.1013	1
18	-28.2284	-59.0981	1
19	-28.3001	-58.9508	1
20	-28.2535	-59.5156	1

- You can test the operation of the file analysis functions on the included file:
  `750nm_mesh_size.mat`

  See for example the following image:


```
In [6]: prediction, NDpred = net_on_file('750nm_mesh_size.mat',0,100)

In [7]: plt.hist(prediction)
Out[7]:
(array([  3.,  42., 184., 458., 761., 871., 524., 205.,  53.,  11.]),
 array([0.20490292, 0.25436729, 0.30383165, 0.35329602, 0.40276038,
        0.45222475, 0.50168912, 0.55115348, 0.60061785, 0.65008221,
        0.69954658]),
 <a list of 10 Patch objects>)
```

Other functions

- Trained models can be tested on simulated data with various types of noise, using any of the functions which do not operate on .mat files. More information can be found in the comments above each function.
- Simulations can be generated separately using the functions in utils.py

If you find this code useful, please cite our work:

Granik, N., Weiss, L.E., Nehme, E., Levin, M., Chein, M., Perlson, E., Roichman, Y. and Shechtman, Y., 2019. Single particle diffusion characterization by deep learning. Biophysical Journal.

For any questions and comments, please contact:

naorgranik@gmail.com

yoavsh@bm.technion.ac.il