Institute of Biological Information Processing
Forschungszentrum Jülich, D-52425 Jülich, Germany

Computational Soft Matter Physics

Prof. Dr. Gerhard Gompper / PD Dr. Dmitry Fedosov / PD Dr. Marisol Ripoll
g.gompper@fz-juelich.de / d.fedosov@fz-juelich.de / m.ripoll@fz-juelich.de
Universität zu Köln, Sommersemester 2023

June 13, 2023

Problem 10: Machine learning the properties of a particle in a harmonic potential

Use deep learning (DL) to extract the spring constant and the friction coefficient of a noisy oscillator from particle-displacement trajectories $x(t)$, which you can generate using the code from the previous assignment. The idea is similar to well-established image processing algorithms based on convolutional neural networks.
First, generate a number of independent particle-displacement trajectories $x(t)$ for different values of k_s and γ. You can simply choose them randomly from the intervals $k_s \in [10, 100]$ and $\gamma \in [10, 100]$. It is advisable to save these trajectories into a file, because you can use them later as training and test sets (e.g., as 70% and 30% fractions of the data) for your DL model. No need to save every time step of $x(t)$, for example, every 10 time steps would be sufficient.
Second, build a DL model by using "keras" module from tensorflow with python implementation. Below is an example of such a model with two convolutional, two max-pooling, and one dense (fully connected) layers.

```
model = keras.Sequential()
model.add(layers.Conv1D(filters=*, kernel_size=*, strides = *, activation='relu', input_shape=(*,*)))
model.add(layers.MaxPooling1D(pool_size=*))
model.add(layers.Conv1D(filters=*, kernel_size=*, strides = *, activation='relu'))
model.add(layers.MaxPooling1D(pool_size=*))
model.add(layers.Flatten())
model.add(layers.Dense(*, activation='relu'))
model.add(layers.Dense(2, activation='relu'))
```

Instead of the symbol '*' you will have to put some parameter values which define your DL model. The last layer is the output layer with two variables k_s and γ. This DL example can also be altered with more or less layers, so feel free to try. You can find more details at the tensorflow website.

To build the model, one can use

```
model.compile(loss=keras.losses.MeanSquaredError(),
optimizer=keras.optimizers.Adam(learning_rate),metrics=['accuracy'])
```

To train the model, use 70% of generated trajectories

```
history = model.fit(in_train_data, out_train_data,epochs=*,steps_per_epoch=*)
```

After that the DL model can be tested using the test data (or 30% of generated trajectories) and its predictions compared with k_s and γ values used for data generation.

```
predictions = model.predict(test_data)
```