Robotics II: Control, Modeling and Learning

Laboratory 7

Inverse Kinematics

By Zarema Balgabekova

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import keras
from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
from keras.utils import np_utils
from keras import backend as K
```

The dataset of 5000 points was collected using dataset.py.

In [2]:
```
#dict.csv contains 5000 data points
data = pd.read_csv("dict2.csv", header = None, names = ["Angles", "XY"])
```

In [3]:
```
train = data['Angles'].to_numpy()
labels = data['XY'].to_numpy()
```

In [5]:
```
X = list()
Y = list()
for i in range(len(train)):
    labels[i] = labels[i].replace('   ', ' ')
    labels[i] = labels[i].replace('  ', ' ')
    labels[i] = labels[i].strip('[ ').strip(' ]')
    train[i] = train[i].strip('(').strip(')')
    result = [float(val) for val in train[i].split(',')]
    X.append(result)
    result = [float(val) for val in labels[i].split(' ')]
    Y.append(result)
```

In [15]:
```
Y = np.delete(Y, 2, 1)
```

In [6]:
```
def rmse(y_true, y_pred):
    return K.sqrt(K.mean(K.square(y_pred - y_true)))
```

Trying different regression losses

1 RMSE (Root mean squared error)

For Forward Kinematics, joint angles were used as inputs. Now, for Inverse Kinematics, we use them as outputs.

In [26]:
```
model = Sequential()
model.add(Dense(10, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss=rmse, optimizer=Adam(0.01))

X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("RMSE: %.6f" % (scores))
```

```
RMSE: 0.254764
```

We can see that RMSE is about 4 times worth for the Inverse Kinematics than for the Forward one.

2 MSE (Mean squared error)

In [27]:
```
model = Sequential()
model.add(Dense(10, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_error', optimizer=Adam(0.01))

X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)


model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSE: %.6f" % (scores))
```

```
MSE: 0.064730
```

3 MAE (Mean absolute error)

In [28]:
```
model = Sequential()
model.add(Dense(10, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_absolute_error', optimizer=Adam(0.01))

X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)


model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MAE: %.6f" % (scores))
```

```
MAE: 0.215991
```

4 MSLE (Mean squared logarithmic error)

In [29]:
```
model = Sequential()
model.add(Dense(10, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=Adam(0.01))

X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.026023
```

5 Huber loss

In [30]:
```
model = Sequential()
model.add(Dense(10, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='huber', optimizer=Adam(0.01))

X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)


model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("Huber: %.6f" % (scores))
```

```
Huber: 0.032863
```

6 Log cosh

In [31]:
```
model = Sequential()
model.add(Dense(10, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='log_cosh', optimizer=Adam(0.01))


X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("Log cosh: %.6f" % (scores))
```

```
Log cosh: 0.032310
```

Again, MSLE loss should be used.

Trying different optimizers

1 Adam

For Adam optimizer, MSLE = 0.026023.

In [32]:
```
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adadelta
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.optimizers import Adagrad
from tensorflow.keras.optimizers import Adamax
from tensorflow.keras.optimizers import Nadam
from tensorflow.keras.optimizers import Ftrl
```

2 SGD

In [33]:
```
model = Sequential()
model.add(Dense(10, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=SGD(0.01))

X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.023547
```

3 Adadelta

In [34]:
```
model = Sequential()
model.add(Dense(10, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=Adadelta(0.01))

X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.031680
```

4 RMSprop

In [35]:
```
model = Sequential()
model.add(Dense(10, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=RMSprop(0.01))

X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.028940
```

5 Adagrad

In [36]:
```
model = Sequential()
model.add(Dense(10, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=Adagrad(0.01))

X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.019792
```

6 Adamax

In [37]:
```
model = Sequential()
model.add(Dense(10, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=Adamax(0.01))

X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.023270
```

7 Nadam

In [38]:
```
model = Sequential()
model.add(Dense(10, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=Nadam(0.01))

X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.015038
```

8 Ftrl

In [39]:
```
model = Sequential()
model.add(Dense(10, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=Ftrl(0.01))

X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.024667
```

This time Nadam optimizer should be used, while Adam was used for the Forward Kinematics.

Changing layers

1 Changing the number of neurons for the first hidden layer

In [40]:
```
model = Sequential()
model.add(Dense(4, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=Nadam(0.01))


X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.014712
```

In [42]:
```
model = Sequential()
model.add(Dense(32, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=Nadam(0.01))


X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.017849
```

In [44]:
```
model = Sequential()
model.add(Dense(2, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=Nadam(0.01))


X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.015817
```

The best result is obtained with 4 neurons for the first hidden layer.

2 Adding more hidden layers

In [46]:
```
model = Sequential()
model.add(Dense(4, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(8, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=Nadam(0.01))

X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.017534
```

In [48]:
```
model = Sequential()
model.add(Dense(4, input_dim =2, activation = 'relu'))
model.add(Dense(32, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(8, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=Nadam(0.01))

X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.028296
```

Increasing the number of layers does not improve the result.

3 Decreasing the number of hidden layers

In [50]:
```
model = Sequential()
model.add(Dense(4, input_dim =2, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=Nadam(0.01))

X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.021389
```

In [51]:
```
model = Sequential()
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=Nadam(0.01))

X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.20)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.019252
```

Decreasing the number of layers also does not improve the result. So, we will keep 2 hidden layers.

Changing the size of testing data and learning rate

1 Size of testing data

In [64]:
```
model = Sequential()
model.add(Dense(4, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=Nadam(0.01))


X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.10)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.014483
```

In [62]:
```
model = Sequential()
model.add(Dense(4, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=Nadam(0.01))


X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.30)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.018688
```

Decreasing the size of testing data slightly improved MSLE. This could be because this way we have more data for training.

2 Changing learning rate

In [66]:
```
model = Sequential()
model.add(Dense(4, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=Nadam(0.1))


X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.10)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.030059
```

In [67]:
```
model = Sequential()
model.add(Dense(4, input_dim =2, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(5, activation='linear'))
model.compile(loss='mean_squared_logarithmic_error', optimizer=Nadam(0.001))


X_train, X_test, y_train, y_test = train_test_split(np.asarray(Y), np.asarray(X), test_size=0.10)

model.fit(X_train, y_train, epochs = 15, verbose = 0)
scores = model.evaluate(X_test, y_test, verbose = 0)
print("MSLE: %.6f" % (scores))
```

```
MSLE: 0.018373
```

The best result was obtained with the old learning rate of 0.01.

Results

By using the provided parameters (neural network with two hidden layers, RMSE loss, and Adam optimizer), we obtained the following loss: $\mathbf{RMSE = 0.254764}$. It was discovered that MSLE significantly decreases the loss. Trying different optimizers, we came to conclusion that Nadam optimizer gives the best performance. Furthermore, changing the number of neurons for the first hidden layer to 4 and decreasing the size of testing data also improved the result. The best loss obtained is $\mathbf{MSLE = 0.014483}$. Therefore, the result was improved by 17.6 times.

Overall, the loss and the obtained improvement are worse for the Inverse Kinematics than they were for the Forward one just as we expected.