# DHOscillator_FFNN_baseline_model

March 10, 2024

[15]:
```
# import numpy, scipy, and matplotlib
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import torch
%matplotlib widget

import os
import tempfile
```

## 1 Dumped Harmonic Oscillator FFNN baseline model

In this notebook, we will train a FFNN with the standard loss function given by the MSE on the train points. The architecture, number of data point, epochs and hyperparamenters, all shared with the ../models/PINN_baseline_model.pt, will be the baseline for further improvements.

[16]:
```
# Number of epochs
n_epochs = 50000

# Batch size
batch_size = 595

# Learning rate and scheduler
lr = 0.01
factor = 0.9
patience = 200

# Model architecture
n_layers = 3
n_neurons = 20
```

[17]:
```
# Model class
class FFNN(torch.nn.Module):
    def __init__(self, n_layers, n_neurons):
```

```
        super(FFNN, self).__init__()
        layers = []
        for i in range(n_layers):
            if i == 0:
                layers.append(torch.nn.Linear(1, n_neurons))
            else:
                layers.append(torch.nn.Linear(n_neurons, n_neurons))
            layers.append(torch.nn.Tanh())
        layers.append(torch.nn.Linear(n_neurons, 2))
        self.model = torch.nn.Sequential(*layers)
    def forward(self, x):
        return self.model(x)
```

## 1.1 Load data

[18]:
```
# import data
# data are generated by "src/DHOscillator_data_gen.py"
data = np.load('../data/DHOscillator_data.npy')
X = data[:,0]
Y = data[:,1:]
```

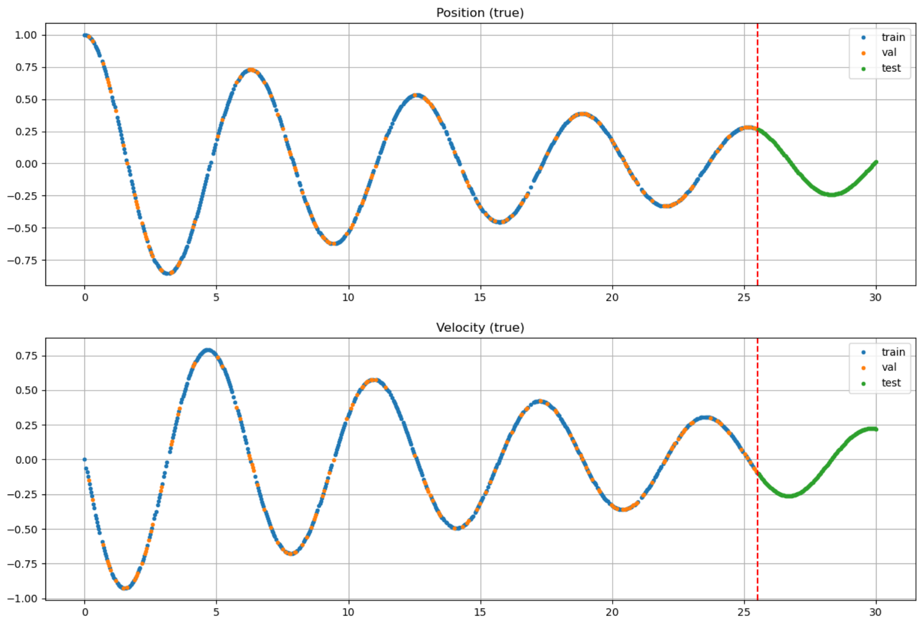
[19]:
```
def data_loader(X, Y, batch_size):
    """
    Function to load data and divide it in batches
    input: X, Y, batch_size
    output: train_X_batches, train_Y_batches, val_X, val_Y, test_X, test_Y
    """

    # divide in train, validation and test
    train_frac = 0.7
    val_frac = 0.15
    test_frac = 0.15

    train_val_X = X[:int((train_frac+val_frac)*len(X))]
    train_val_Y = Y[:int((train_frac+val_frac)*len(X)), :]
    train_X, val_X, train_Y, val_Y = train_test_split(
        train_val_X,
        train_val_Y,
        test_size=val_frac/(train_frac+val_frac),
        random_state=42
        )

    test_X = X[int((train_frac+val_frac)*len(X)):]
    test_Y = Y[int((train_frac+val_frac)*len(X)):, :]

    # convert to torch tensor
    train_X = torch.tensor(train_X, dtype=torch.float32).view(-1, 1)
```

```
    train_Y = torch.tensor(train_Y, dtype=torch.float32)
    val_X = torch.tensor(val_X, dtype=torch.float32).view(-1, 1)
    val_Y = torch.tensor(val_Y, dtype=torch.float32)
    test_X = torch.tensor(test_X, dtype=torch.float32).view(-1, 1)
    test_Y = torch.tensor(test_Y, dtype=torch.float32)

    # divide in batches train
    train_X_batches = torch.split(train_X, batch_size)
    train_Y_batches = torch.split(train_Y, batch_size)


    return train_X_batches, train_Y_batches, val_X, val_Y, test_X, test_Y
```

[20]:
```
# use the data loader to get the data, in this example we use only one batch
train_X_batches, train_Y_batches, val_X, val_Y, test_X, test_Y = data_loader(X,␣
 ↪Y, batch_size)
```

[21]:
```
# plot the position
plt.figure(figsize=(15, 10))
plt.subplot(2, 1, 1)
plt.plot(train_X_batches[0].detach().numpy(), train_Y_batches[0][:, 0].detach().
 ↪numpy(), '.', label='train')
plt.plot(val_X.detach().numpy(), val_Y[:, 0].detach().numpy(), '.', label='val')
plt.plot(test_X.detach().numpy(), test_Y[:, 0].detach().numpy(), '.',␣
 ↪label='test')
plt.grid()
plt.title('Position (true)')
plt.axvline(x=30*0.85, color='r', linestyle='--')
plt.legend()

# plot the velocity
plt.subplot(2, 1, 2)
plt.plot(train_X_batches[0].detach().numpy(), train_Y_batches[0][:, 1].detach().
 ↪numpy(), '.', label='train')
plt.plot(val_X.detach().numpy(), val_Y[:, 1].detach().numpy(), '.', label='val')
plt.plot(test_X.detach().numpy(), test_Y[:, 1].detach().numpy(), '.',␣
 ↪label='test')
plt.grid()
plt.title('Velocity (true)')
plt.axvline(x=30*0.85, color='r', linestyle='--')
plt.legend()
```

[21]: <matplotlib.legend.Legend at 0x7fcd6f82bbc0>




## 1.2 FFNN

The FFNN will be trained on data point with time < 25, discarding validation data this means 595 point of the ODE.

[22]:
```
# define the model
model_FFNN = FFNN(n_layers, n_neurons)

# define the loss function, mean squared error
loss_fn = torch.nn.MSELoss()

# define the optimizer and lr scheduler
optimizer = torch.optim.Adam(model_FFNN.parameters(), lr=lr)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min',␣
↪factor=factor, patience=patience)
```

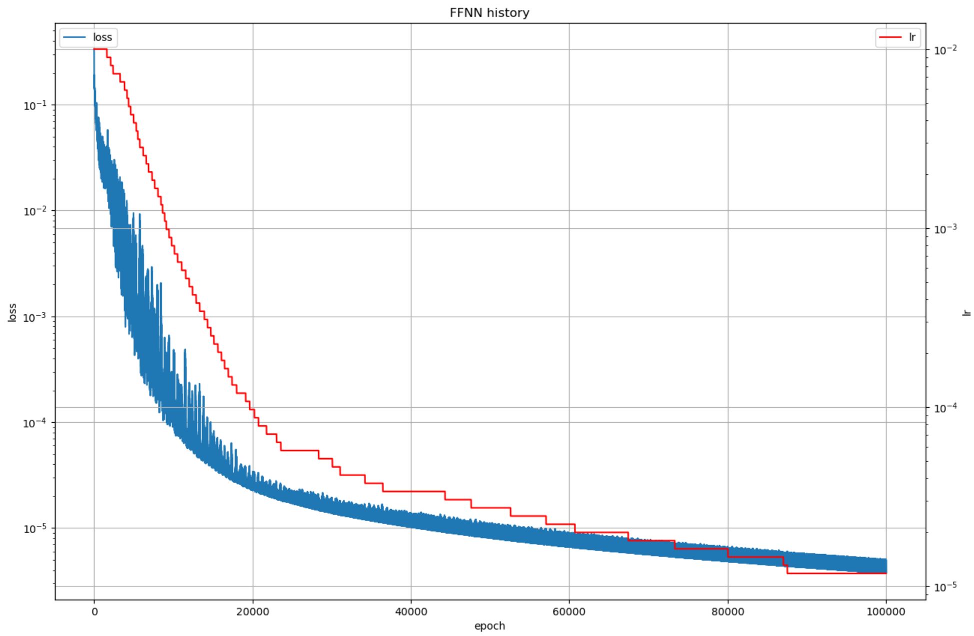
[23]:
```
%%time
history_FFNN = []

# train the model
for epoch in range(n_epochs):
    for i, (X, Y) in enumerate(zip(train_X_batches, train_Y_batches)):
        optimizer.zero_grad()
```

```
        Y_pred = model_FFNN(X)
        loss = loss_fn(Y_pred, Y)
        loss.backward()
        optimizer.step()
        optimizer.step()
        scheduler.step(loss)
        history_FFNN.append([loss.item(), optimizer.param_groups[0]['lr']])
    if epoch % 10000 == 0:
        print(epoch, loss.item())
```

```
0 0.26025626063346863
10000 2.473035237926524e-05
20000 1.0696694516809657e-05
30000 6.7767400651064236e-06
40000 4.859512955590617e-06
CPU times: user 13min 16s, sys: 7.64 s, total: 13min 24s
Wall time: 3min 21s
```

[36]:

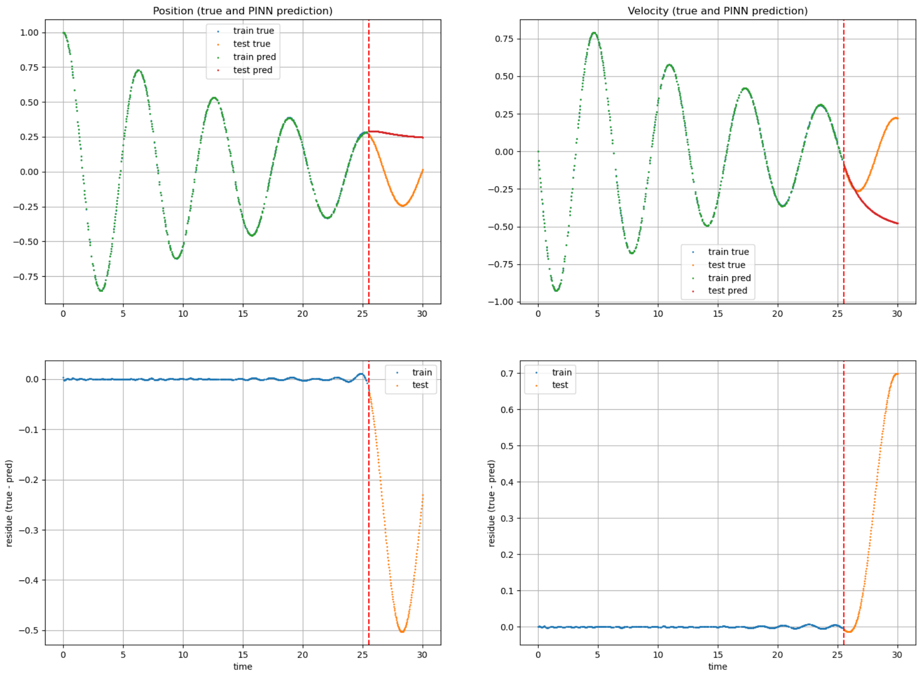
```
# plot history_FFNN loss and lr in two subplots
history_FFNN = np.array(history_FFNN)

fig, ax = plt.subplots(figsize=(15, 10))
# plot the loss
ax.plot(history_FFNN[:, 0], label='loss')
ax.legend(loc='upper left')
ax.set_yscale('log')
ax.set_xlabel('epoch')
ax.set_ylabel('loss')
plt.grid()

# plot the learning rate
ax2 = ax.twinx()
ax2.plot(history_FFNN[:, 1], label='lr', color='r')
ax2.set_yscale('log')
ax2.set_ylabel('lr')
# legend to the right
ax2.legend(loc='upper right')
plt.grid()
plt.title('FFNN history')

# save the figure
plt.savefig('../plot/DHOscillator_FFNN_baseline_history.png')
```




[35]:
```
# get predictions
Y_pred_train = model_FFNN(train_X_batches[0])
Y_pred_val = model_FFNN(val_X)
Y_pred_test = model_FFNN(test_X)

# plot the position, and subplot the residue
plt.figure(figsize=(18, 13))
plt.subplot(2, 2, 1)

marker='.'
markersize=2

plt.plot(train_X_batches[0].detach().numpy(), train_Y_batches[0][:, 0].detach().numpy(), marker, label='train true', markersize=markersize)
plt.plot(test_X.detach().numpy(), test_Y[:, 0].detach().numpy(), marker, label='test true', markersize=markersize)
plt.plot(train_X_batches[0].detach().numpy(), Y_pred_train[:, 0].detach().numpy(), marker, label='train pred', markersize=markersize)
plt.plot(test_X.detach().numpy(), Y_pred_test[:, 0].detach().numpy(), marker, label='test pred', markersize=markersize)
plt.grid()

plt.title('Position (true and PINN prediction)')
```

```
plt.axvline(x=30*0.85, color='r', linestyle='--')
plt.legend()

plt.subplot(2, 2, 3)
plt.plot(train_X_batches[0].detach().numpy(), train_Y_batches[0][:, 0].detach().numpy()-Y_pred_train[:, 0].detach().numpy(), marker, label='train', markersize=markersize)
plt.plot(test_X.detach().numpy(), test_Y[:, 0].detach().numpy()-Y_pred_test[:, 0].detach().numpy(), marker, label='test', markersize=markersize)
plt.grid()
plt.ylabel('residue (true - pred)')
plt.xlabel('time')
plt.legend()
plt.axvline(x=30*0.85, color='r', linestyle='--')


# new figure for the velocity
plt.subplot(2, 2, 2)

plt.plot(train_X_batches[0].detach().numpy(), train_Y_batches[0][:, 1].detach().numpy(), marker, label='train true', markersize=markersize)
plt.plot(test_X.detach().numpy(), test_Y[:, 1].detach().numpy(), marker, label='test true', markersize=markersize)
plt.plot(train_X_batches[0].detach().numpy(), Y_pred_train[:, 1].detach().numpy(), marker, label='train pred', markersize=markersize)
plt.plot(test_X.detach().numpy(), Y_pred_test[:, 1].detach().numpy(), marker, label='test pred', markersize=markersize)
plt.grid()

plt.title('Velocity (true and PINN prediction)')
plt.axvline(x=30*0.85, color='r', linestyle='--')
plt.legend()

plt.subplot(2, 2, 4)
plt.plot(train_X_batches[0].detach().numpy(), train_Y_batches[0][:, 1].detach().numpy()-Y_pred_train[:, 1].detach().numpy(), marker, label='train', markersize=markersize)
plt.plot(test_X.detach().numpy(), test_Y[:, 1].detach().numpy()-Y_pred_test[:, 1].detach().numpy(), marker, label='test', markersize=markersize)
plt.grid()
plt.ylabel('residue (true - pred)')
plt.xlabel('time')
plt.legend()
plt.axvline(x=30*0.85, color='r', linestyle='--')

# save the figure
```

```
plt.savefig('../plot/DHOscillator_FFNN_baseline_results.png')
```




[26]:
```
# test loss
loss_test_FFNN = loss_fn(Y_pred_test, test_Y)
print('test loss:', loss_test_FFNN.item())
```

```
test loss: 0.1473110467195511
```

[27]:
```
# save the model
model_path = os.path.join('../models', 'DHO_FFNN_baseline.pt')
torch.save(model_FFNN, model_path)
```

Comment: The model do not reproduce qualitativly the beheviuor of the ODE outside the train time span.