Regression with Feed-Forward Neural Networks

Exercises in PyTorch

Marco Forgione, Dalle Molle Institute for Artificial Intelligence, USI-SUPSI, Lugano, Switzerland

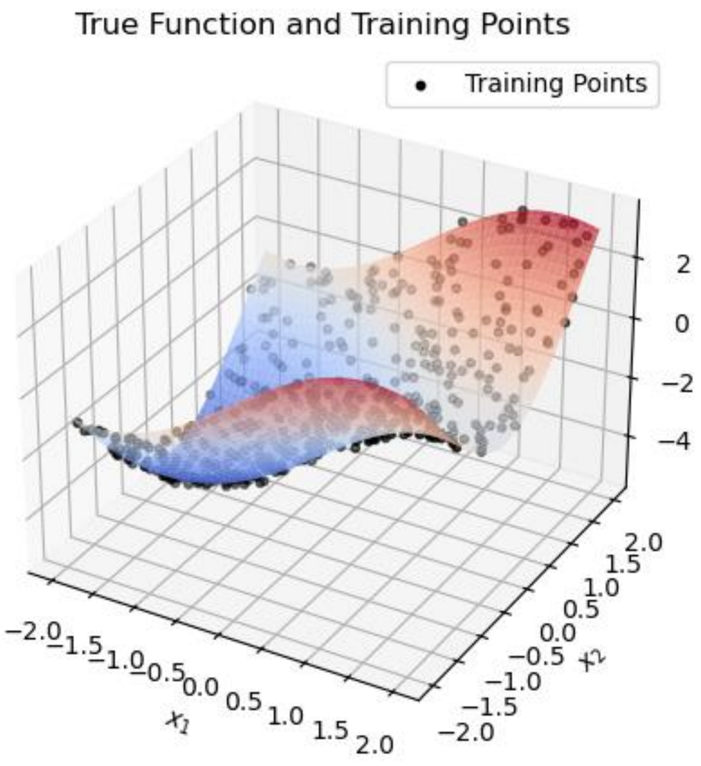
2D regression on synthetic data (synthetic_2d folder)

Consider the 2D function $f : \mathbb{R}^2 \rightarrow \mathbb{R}$

$$f(x) = 2\sin(x_1) - 3\cos(x_2)$$

$$x \in [-2, 2]^2 \subset \mathbb{R}^2$$

- Training and test datasets: 500 points uniformly sampled in the domain.
- Additive noise with standard deviation 0.1



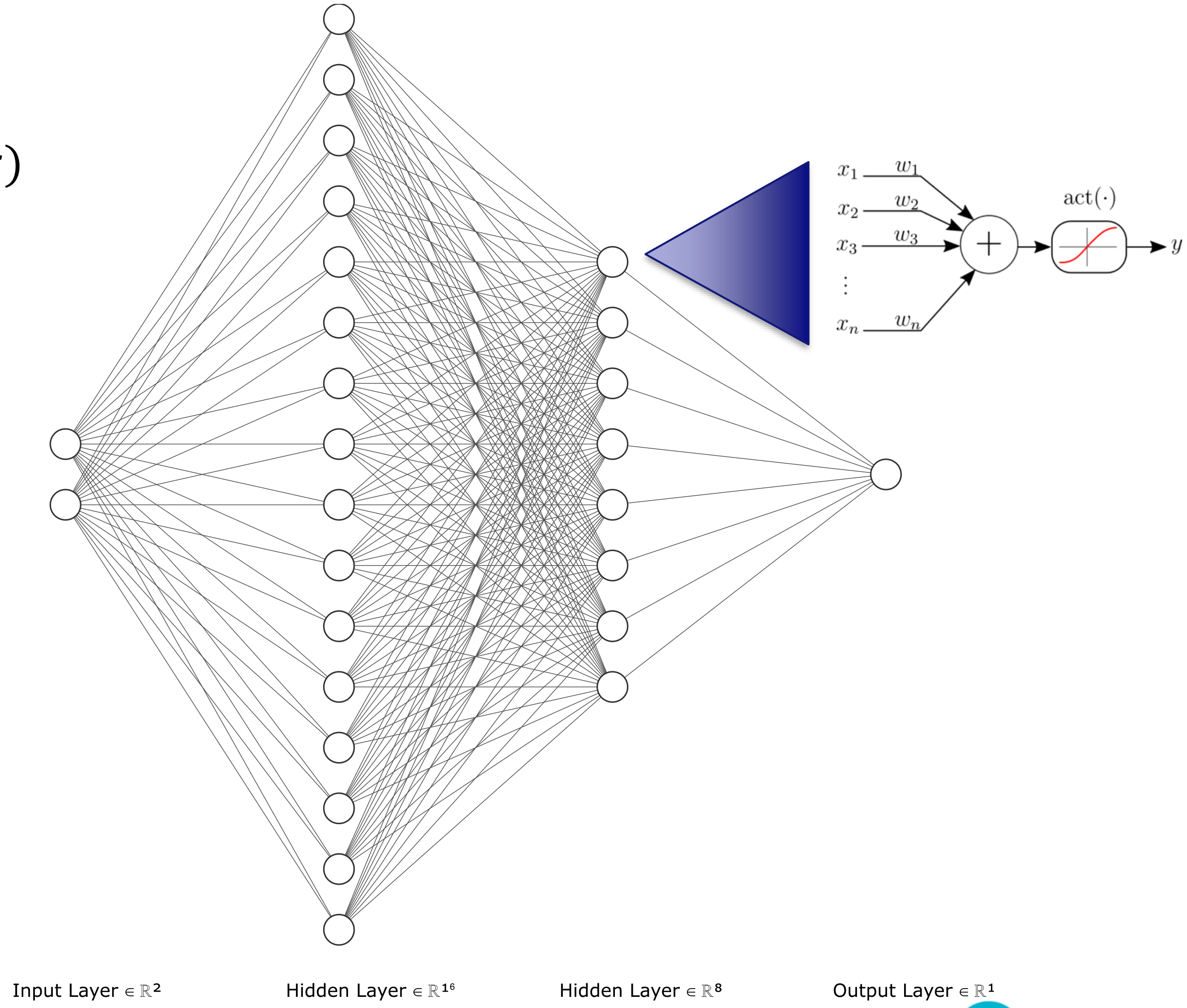
Feed-forward neural network model

- 2 inputs, 1 output - this is the structure of $f(x)$
- 2 hidden layers with [16, 8] neurons
- tanh non-linearities

```python
class FeedForwardNN(nn.Module):
    def __init__(self):
        super().__init__()
        ...

    def forward(self, x):
        ...

model = FeedForwardNN()
```

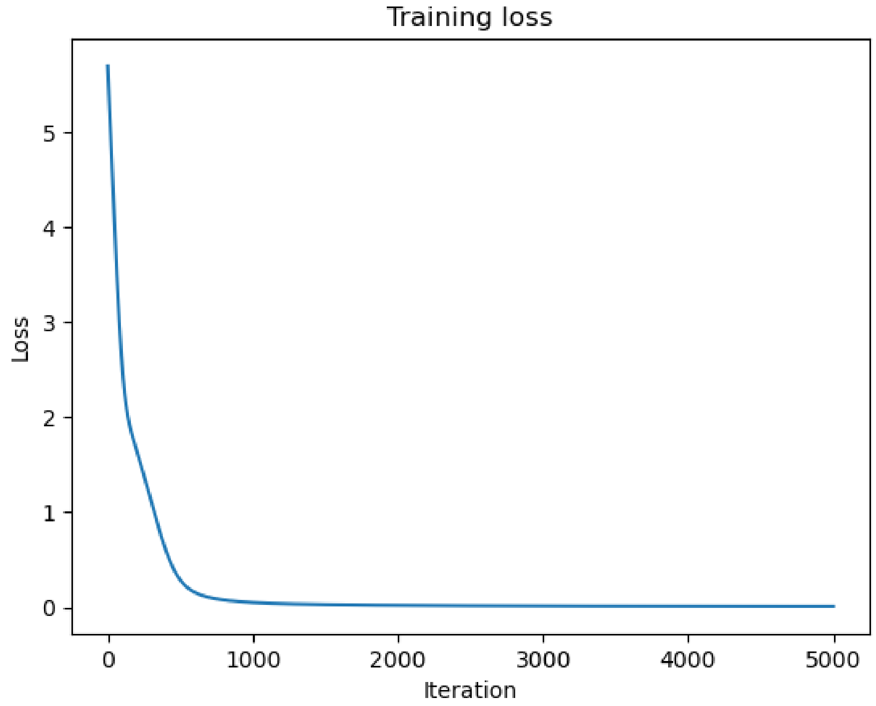


Model training

```
epochs = 100
lr = 1e-3

criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=lr)

# Optimization loop
for epoch in range(epochs):
    ...
```



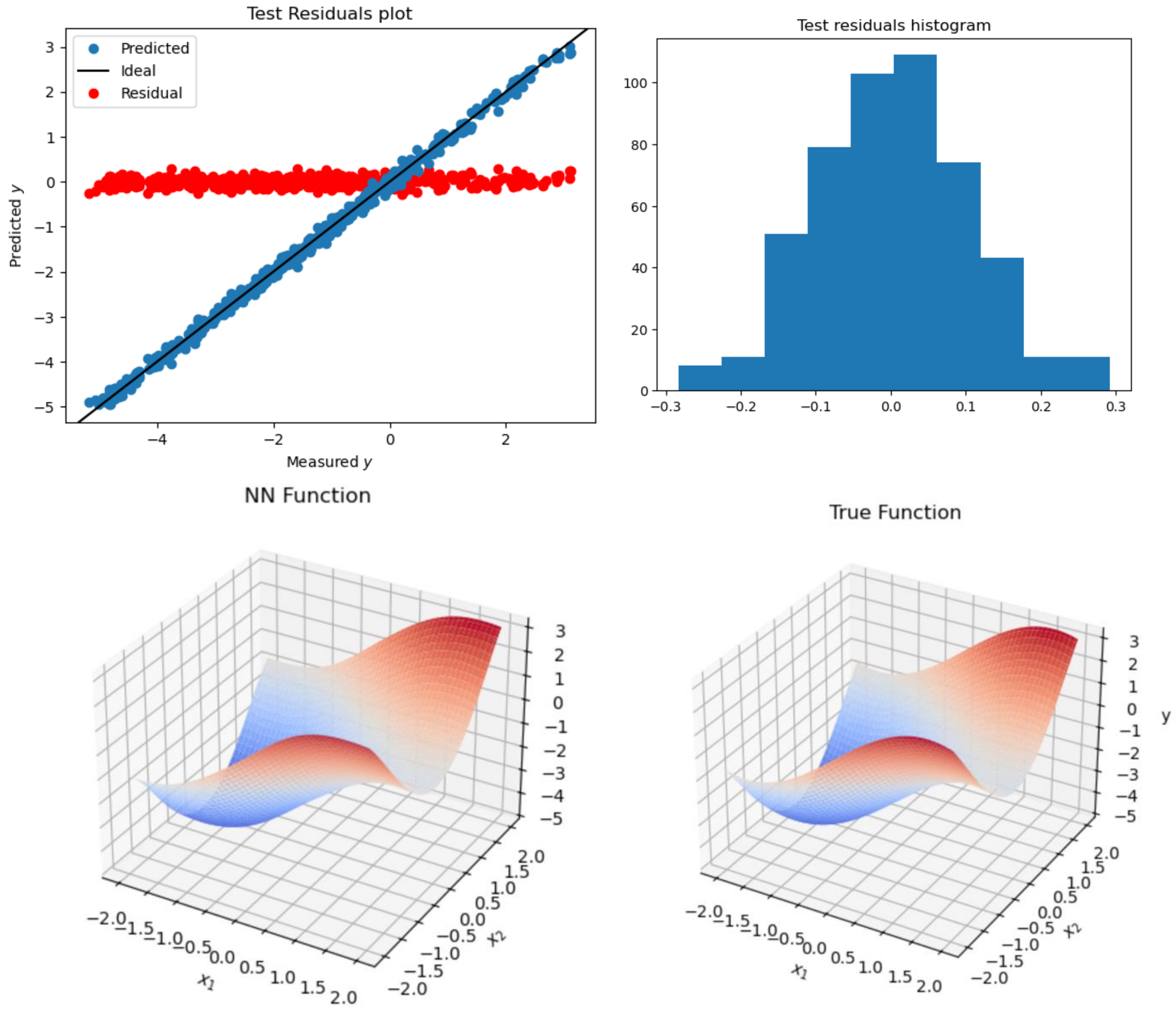
Model evaluation

It is common to inspect, on the test dataset:

- Predictions and residuals vs measured y (top left)
- Histogram of residuals (top right)

In this toy example:

- The input is only 2D. We can visualize the learned function (bottom left)
- The true function is known. We can visualize also visualize it in 2D (bottom right)

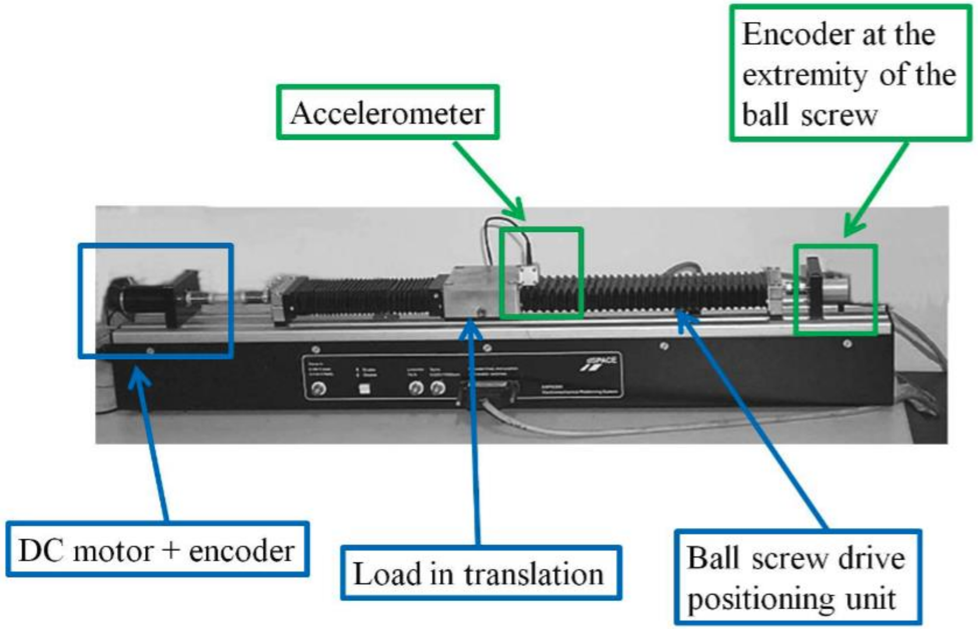


Inverse Dynamical Modeling on the EMPS benchmark (emps folder)

Real dataset from a (simple) mechanical system: the Electro-Mechanical Positioning System

- Prismatic joint, 1-DoF mechanical system

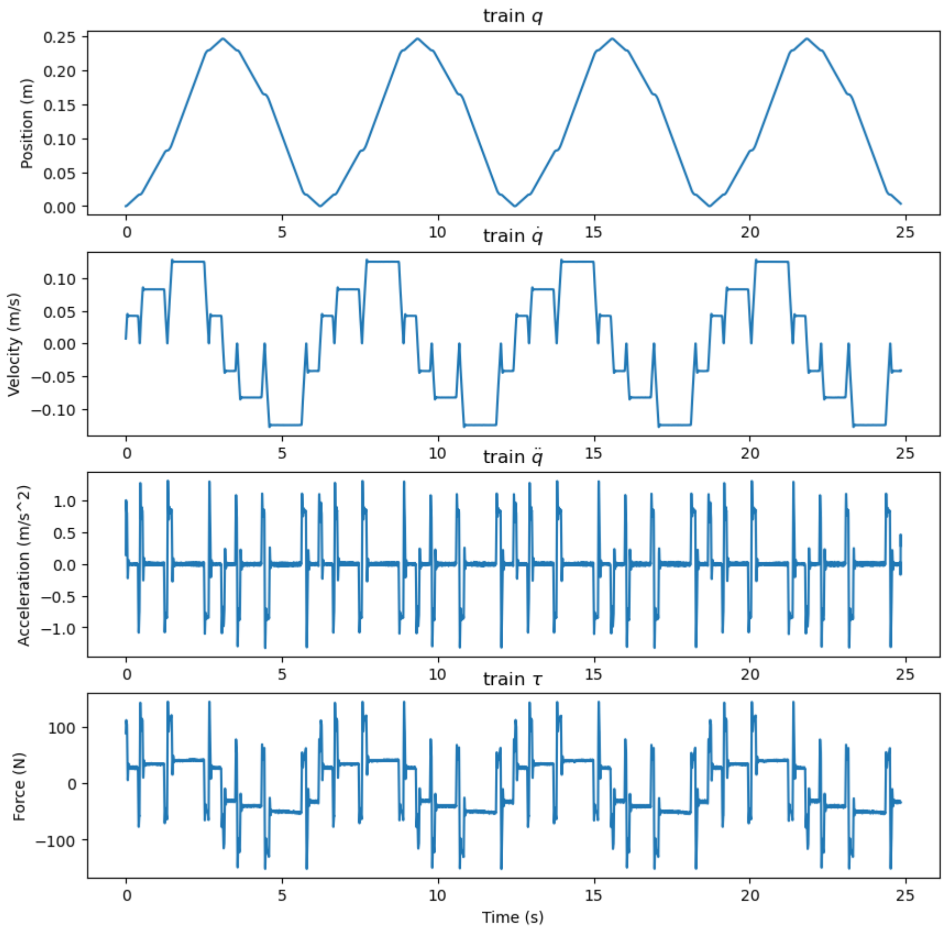
$$M\ddot{q}(t) = \tau(t) - \tau_f(t) - b$$

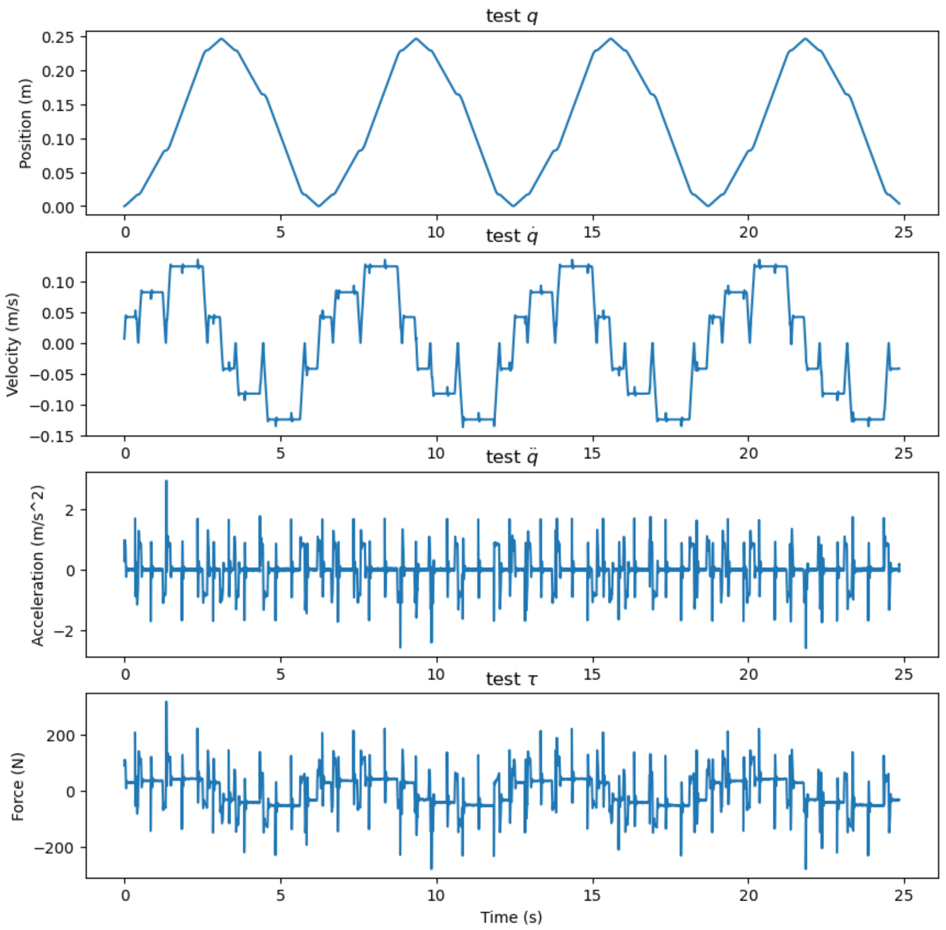


- $q(t)$: measured position (m)
- $\tau(t)$: known applied force (N)
- $\tau_f(t)$: unknown friction (N)
- M: unknown mass (kg)
- b: force measurement bias (N)

We want an inverse dynamical model (IDM): $q(t), \dot{q}(t), \ddot{q}(t) \rightarrow \tau(t)$

Training and test datasets



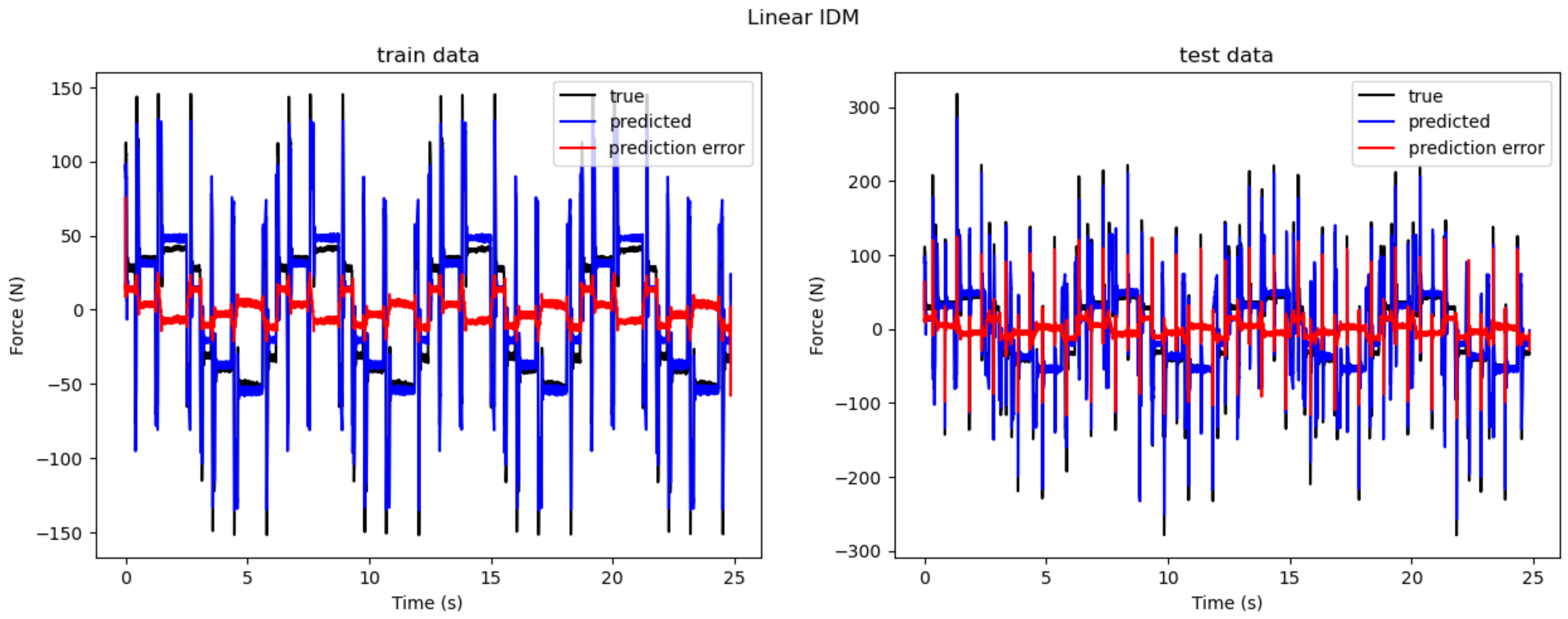


Linear model

We assume a linear friction model: $\tau_f = -F_v \dot{q}(t)$

Then, the IDM is: $\tau(t) = M\ddot{q}(t) + F_v \dot{q}(t) + b$. We can fit the IDM with a **linear regression**:

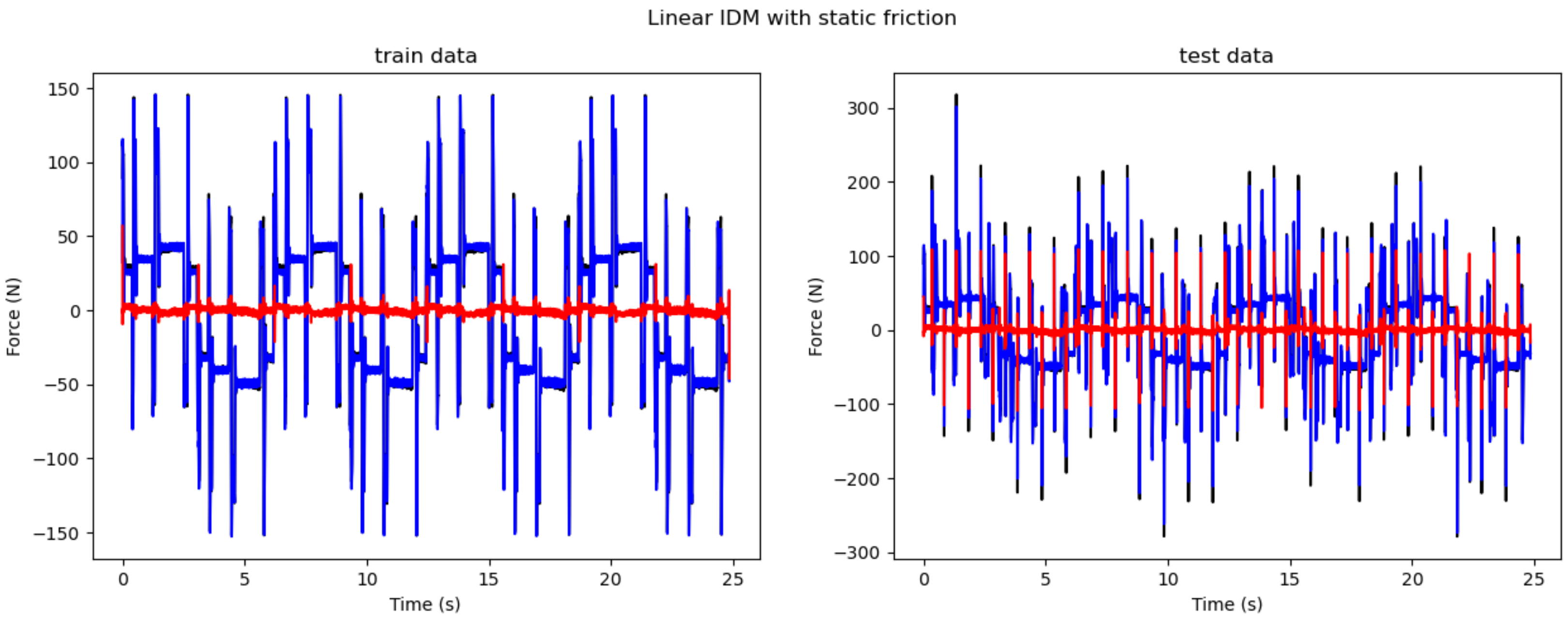
$$\tau(t) = \phi(t)\theta, \qquad \phi(t) = [\ddot{q}(t)\; \dot{q}(t)\; 1], \qquad \theta = [M\; F_v\; b]^\top$$



Linear model with static friction

We use a more sophisticated friction model: $\tau_f(t) = -F_v\dot{q}(t) - F_c\text{sign}(\dot{q}(t))$

$$\tau(t) = \phi(t)\theta, \qquad \phi(t) = [\ddot{q}(t)\;\dot{q}(t)\;\text{sign}(q(t))\;1], \qquad \theta = [M\;F_v\;F_c\;b]^\top$$

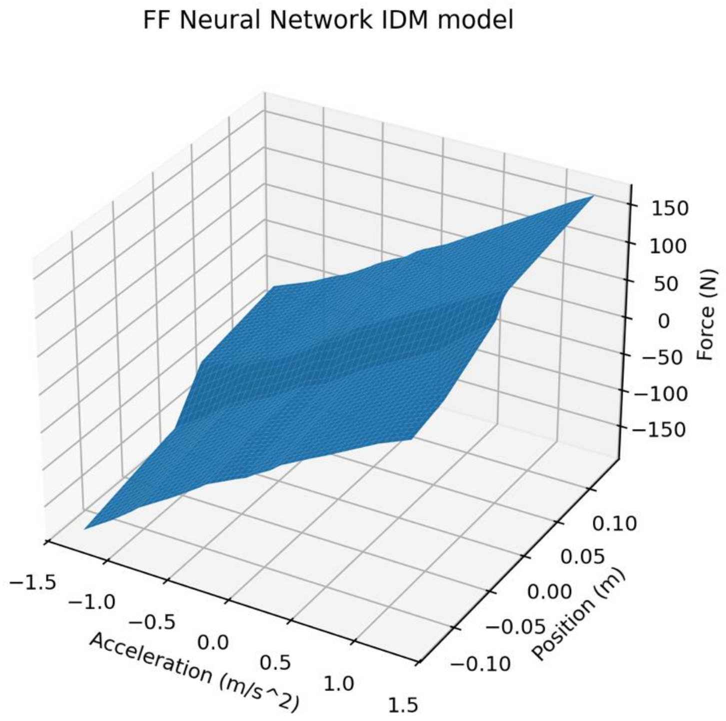


Feed-forward neural network

We ignore all physics and fit a feed-forward neural network instead: $\tau(t) = \mathrm{FF}(\ddot{q}, \dot{q})$

```
in_dim = train_X_torch.shape[1] # 2
out_dim = 1
batch_size = 128
hidden_size = 32
lr = 5e-4

friction_net = torch.nn.Sequential(
    torch.nn.Linear(in_dim, hidden_size),
    torch.nn.ReLU(),
    torch.nn.Linear(hidden_size, hidden_size),
    torch.nn.ReLU(),
    torch.nn.Linear(hidden_size, 1)
)
```



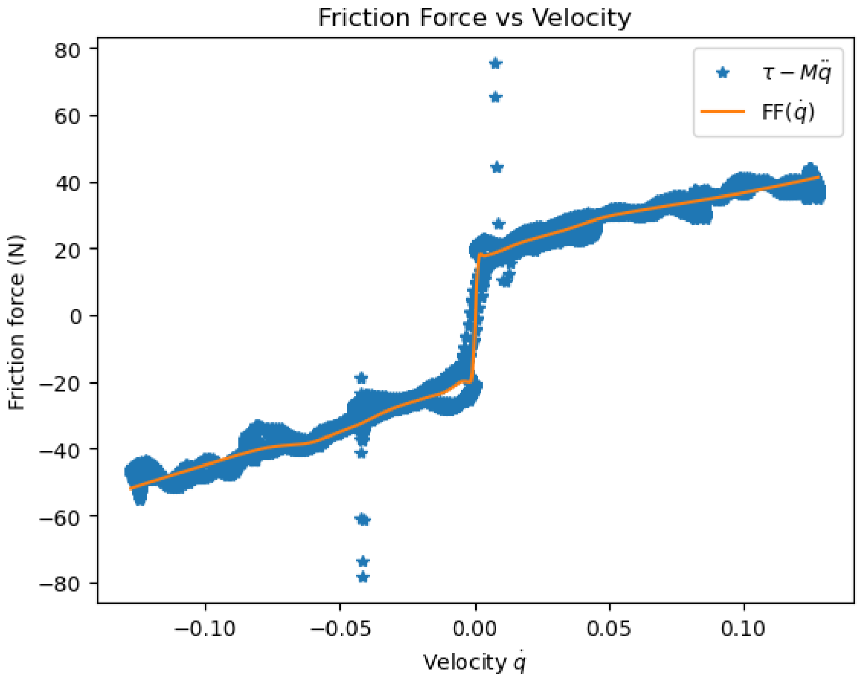
Train it yourself!

Physics-inspired neural network

We trust Newton's law, but nothing else: $\tau(t) = M\ddot{q} + \mathrm{FF}(\dot{q})$

```
class CustomIDM(nn.Module):
    def __init__(self, n_q=1, hidden_size=32):
        self.nq = n_q
        super(CustomIDM, self).__init__()
        self.inertia_net = nn.Linear(n_q, self.nq, bias=False)
        self.friction_net = nn.Sequential(
            nn.Linear(n_q, hidden_size),
            nn.GELU(),
            nn.Linear(hidden_size, hidden_size),
            nn.GELU(),
            nn.Linear(hidden_size, 1)
        )

    def forward(self, x):
        inertia = self.inertia_net(x[:, :self.nq])  # Linear in \ddot q
        friction = self.friction_net(x[:, self.nq:])  # Non-linear in \dot q
        return inertia + friction
```



Train it yourself!