Commented Code

* [data.py](https://github.com/lululxvi/deep-learning-for-indentation/blob/master/src/data.py): The classes are used to read the data file. Remember to uncomment certain line in ExpData to scale dP/dh.
* [nn.py](https://github.com/lululxvi/deep-learning-for-indentation/blob/master/src/nn.py): The main functions of multi-fidelity neural networks.
* [model.py](https://github.com/lululxvi/deep-learning-for-indentation/blob/master/src/model.py): The fitting function method. Some parameters are hard-coded in the code, and you should modify them for different cases.
* [fit\_n.py](https://github.com/lululxvi/deep-learning-for-indentation/blob/master/src/fit_n.py): Fit strain-hardening exponent.
* [mfgp.py](https://github.com/lululxvi/deep-learning-for-indentation/blob/master/src/mfgp.py): Multi-fidelity Gaussian process regression.

|  |  |  |  |
| --- | --- | --- | --- |
| Modification | 1. (Fitting & FEM) | 2. (2D & 3D FEM) | 3. (FEM & real) |
| mape.append | svm(data) | *nn(data)?* | nn(data) |
| validation\_FEM | FEMData( | BerkovichData( | BerkovichData( |
| validation\_exp\_cross\_transfer | *uncomment?* | *uncomment?* | *uncomment?* |
| main | *depends* | *depends* | *depends* |
| forward\_model | cstar = 1.1957 | *cstar = 1.2370?* | cstar = 1.2370 |
| inverse\_model | cstar = 1.1957 | *cstar = 1.2370?* | cstar = 1.2370 |
| test\_inverse | d = FEMData | d = Berkovich | d = ExpData |
| test\_inverse | d = FEMData | d = Berkovich | d = ExpData |
| test\_inverse | d = FEMData | d = Berkovich | d = Berkovich |
| main | *depends* | *depends* | *depends* |
| read\_1angle | uncomment | *comment?* | *comment?* |
| read | 2nd df? | 3rd df? | 4th df? |

nn.py:

validation\_model: mape.append(svm(data))

**This appends svm (the fitting function) instead of nn (neural network). I think this needs to be uncommented for the first section.**

validation\_FEM: datafem = BerkovichData(yname)

**This uses Berkovich instead of FEM data. All 3D indentation tests were with Berkovich.**

mfnn: checker = dde.callbacks.ModelCheckpoint("model/model.ckpt", verbose=1, save\_better\_only=True, period=1000), losshistory, train\_state = model.train(epochs=30000, callbacks=[checker]), losshistory, train\_state = model.train(epochs=5000, model\_restore\_path="model/model.ckpt-28000")

**I think this was only included in the code to train it. Not sure if it will be needed.**

validation\_mf: datalow = ModelData(yname, 10000, "forward\_n"), datahigh = FEMData(yname, [70]), kf = LeaveOneOut(), mape.append(dde.utils.apply(mfgp, (data,)))

**I’m pretty sure this can also just be left alone.**

validation\_exp\_cross: cases = range(6), for train\_index in itertools.combinations(cases, 3): train\_index = list(train\_index) test\_index = list(set(cases) - set(train\_index))

validation\_exp\_cross\_transfer: Roughly half the function

**I think this half of the function was commented by mistake.**

main: Functions are uncommented depending on the application.

**These will obviously need to be adjusted depending on the application.**

model.py:

forward\_model: cstar = 1.2370

**1.1957: Conical, 1.2370: Berkovich**

inverse\_model: cstar = 1.2370

**1.1957: Conical, 1.2370: Berkovich**

test\_inverse: d = BerkovichData("Estar"), d = ExpData("B3067.csv", "Estar"), d = BerkovichData("sigma\_y"), d = ExpData("B3067.csv", "sigma\_y"), d = FEMData("n", [70])

gen\_forward: sigma\_y = random.uniform(0.03, 5.3)

**Adjust these based on which test you are using. Berkovich is for 3D FEM.**

main: print(inverse\_model(27.4e9, 0.902, 4768e3 \* 0.2 \* (27.4 / 3)\*\*0.5 \* 10\*\*(-1.5), 0.3, 0.2e-6, nu\_i=0.07, E\_i=1100e9)), test\_inverse(), test\_inverse\_dual(), gen\_inverse()

**Choose what you want to test. This is run separately from nn.py.**

data.py

read\_1angle: df = df.loc[df["n"] <= 0.3], df["dP/dh (N/m)"] \*= 1.167 / 1.128, sigma = 0.2, df["E\* (GPa)"] \*= 1 + sigma \* np.random.randn(len(df)), df["sy (GPa)"] \*= 1 + sigma \* np.random.randn(len(df))

**Scale c\* and edit the df for E\***

read\_2angles: df = df.loc[:100], self.X = df[["C (GPa)", "dP/dh (N/m)", "C (GPa)\_60", "dP/dh (N/m)\_60"]].values

**Scale c\* and edit the df for E\***

read: df["dP/dh (N/m)"] \*= 0.2 \* (df["C (GPa)"] / 3) \*\* 0.5 \* 10 \*\* (-1.5), df["dP/dh (N/m)"] \*= 0.2 \* (df["C (GPa)"] / df["Pm (N)"]) \*\* 0.5 \* 10 \*\* (-1.5), df["dP/dh (N/m)"] \*= 0.2 / df["hm (um)"], df["dP/dh (N/m)"] \*= 1.128 / 1.167

**Scale c\* and edit the df for E\***