**Introduction/objectives**

Instrumental indentation is good for situations where you can’t test *E* and *σy* the normal way.

Spherical indentation less accurate than tip indentation because spheres of different radii make different shapes of indentation.

A NN is a good fit for an inverse indentation problem because they are highly nonlinear.

Materials used: Al6061, Al7075, and 6 3D-printed Ti alloys.

n might be the strain-hardening exponent.

To run the code with Yanbo’s data, I will use the equation or to find C. It appears that Liu *et al*. did this identity too, based on their *Supplemental Information* and trying it on some of their data in a CSV file. I will find and by step integration of the curve, then use these to find . This will give me the information needed to find , another value used by Liu *et al.* can be found by using the first 2 (or few) points on the unloading curve to find the initial unloading slope. These data points are all available from the *Depth (nm)* and *Load (uN)* points on Yanbo’s CSV files.

In Yanbo’s CSV files, the first lone point is the start, the first set of points are just noise before placing the instrument on the material, the second are the initial rest state, the third are between yield and unloading (~2% plastic strain in the first set), and the fourth are unloading. I will need to process at least the 2nd, 3rd, and 4th sets with a python program or something.

**Training NN with fitting function data**

Dataset generated with **model.py**. Items in model.py must be hard-coded depending on what you want. Running model.py adds new points to model\_forward.xlsx.

1. NN trained with data from the fitting function: output to **Fig. SI(*A*)**, **validation\_model()**.

Error is ~50% even with large training sets for σ­y validated against fitting functions.

2. Trained with 2D FEM simulations. Dataset involved 100 different parameter combinations. 3 points with **n>0.3** and **σy/E\*≥0.3** **removed**. **Fig. SI(B)**, **validation\_FEM().** Conical indentations (harder to calculate) were used to train the NN.

3. Training using multiple geometries: **Fig. 2, validation\_FEM()**.

**Inverse Analysis Using MFNNs**

4. **n>0.3**, fitting functions (low fidelity) + 2D FEM (High). **Change datalow to ModelData and datahigh to FEMData**. **Fig. 3(*A*&*B*), validation\_mf()**.

5. Residual MFNN compared to non-residual. I’m pretty sure you find non-residual by setting **k=LeaveOneOut()** in **validation\_mf()**. This gives **Fig. S2(A&B)**.

6. Use **validation\_scaling()** for **Fig.3(C&D)**

7. Use **validation\_exp()** for **Fig.4(A&B)**.

8. Use **validation\_fitting()** for **Fig.5**? I think this is not super relevant to the paper, though.

9. Use **validation\_scaling()** for **Fig. 6**? (raw) means the 3D Berkovich data was uncorrected for radius effects, (tip) means it was corrected. **Correct (raw) to (tip) like this: raw data is P-h using one of the factors in read(self).**

10. Use **validation\_exp\_cross()** for **Fig.6**?

(mac)11. I think **Fig. 7** compares **validation\_exp\_cross()** and **validation\_exp\_cross2()**.

(PC)12. Use **validation\_exp\_cross2()** for **Fig. 8** and **Fig. S4**.

✅13. Use **validation\_exp\_cross2()** for **Fig. 9**

14. Use **validation\_exp\_cross()** and **validation\_exp\_cross\_transfer()** for **Fig. 10**.

**validation\_exp\_cross3()** and **validation\_exp\_cross\_transfer()** are only discussed at the end, but no figures.

FEMData:

1. read\_1angle: Option to include n≤0.3 in df, Scale c\* from Conical (2D) to Berkovich (3D).

2. read\_2angles: Option to change the properties in self.X for Al alloys?

ExpData:

1. read: Scale dP/dh from 3N to hm = 0.2um

2. read: Scale dP/dh from Pm to hm = 0.2um

3. read: Scale c\* from Berkovich to conical.

**Summary**

This program uses a multi-fidelity neural network (MFNN) to find the yield stress σy, the indentation elastic modulus E\*, and the stress at three plastic deformations for metal alloys. In addition to the MFNN, the paper discusses fitting functions and single-fidelity neural networks and compares these to the newer multi-fidelity method. It finds that MFNNs are much more accurate than either of these two methods. This finding is true even when the MFNN is trained using experimental data from a similar but not identical material.

Table 1 below says which function I think is used to create each figure in the paper. Specifics from the source code are not mentioned in the paper, and some function arguments need to be updated depending on what is being found, but I have tried to identify these thoroughly. I have tested and obtained a similar output from highlighted entries.

|  |  |  |
| --- | --- | --- |
| **Figure** | **Function** | **Notes** |
| 1 |  | Comparison between different neural networks |
| 2 | validation\_FEM | I haven’t gotten this to output reasonable values yet. |
| 3(A-B) | validation\_mf | Use ModelData as datalow and FEMData as datahigh |
| 3(C-D) | validation\_mf | Use FEMData as datalow and BerkovichData as datahigh |
| 4(A) | model.py validation\_mf  validation\_exp\_cross2 | Set **d** to the dataset you wish to model. Default Al.  Make sure you use the data file corresponding to material.  You will have to choose the self and peer data files. |
| 4(B) | validation\_mf  validation\_exp\_cross2 | Make sure you use the data file corresponding to material.  You will have to choose the self and peer data files. |
| 5 | validation\_exp  validation\_exp\_cross2  fit\_n.py | This function needs to be set for **Each Al** alloy here. Make sure you clear sigma\_y.dat before to get data. Like 9(B). Uncomment df["dP/dh (N/m)"] \*= 0.2 \* (df["C (GPa)"] / 3) \*\* 0.5 \* 10 \*\* (-1.5) for Aluminum. |
| 6 | validation\_exp\_cross2 | Uncomment df["dP/dh (N/m)"] \*= 0.2 / df["hm (um)"] in data.py for Titanium alloys only.  Uncomment df["dP/dh (N/m)"] \*= 0.2 \* (df["C (GPa)"] / 3) \*\* 0.5 \* 10 \*\* (-1.5) for Aluminum.  You need to select the correct dataset for this.  train\_size = 5 for Ti and 3 for Al. |
| 7 | validation\_exp\_cross2 | Uncomment df["dP/dh (N/m)"] \*= 0.2 / df["hm (um)"] in data.py for Titanium alloys only.  Uncomment df["dP/dh (N/m)"] \*= 0.2 \* (df["C (GPa)"] / 3) \*\* 0.5 \* 10 \*\* (-1.5) for Aluminum.  You need to select the correct dataset for this.  train\_size = 5 for Ti and 3 for Al. |
| 8 | validation\_exp  validation\_exp\_cross2 | Uncomment df["dP/dh (N/m)"] \*= 0.2 / df["hm (um)"] in data.py. Use validation\_exp for training\_size = 0. |
| 9(A) | validation\_exp\_cross2 | train\_size = 5. Uncomment df["dP/dh (N/m)"] \*= 0.2 / df["hm (um)"] in data.py. Change dataexp1 to B3067 for peer. |
| 9(B) | validation\_exp\_cross2  fit\_n.py | This function needs to be called for **Ti** alloys here. Make sure you clear sigma\_y.dat before to get data |
| 10 | validation\_exp  validation\_exp\_cross2 | Uncomment df["dP/dh (N/m)"] \*= 0.2 / df["hm (um)"] in data.py for Titanium alloys only.  Uncomment df["dP/dh (N/m)"] \*= 0.2 \* (df["C (GPa)"] / 3) \*\* 0.5 \* 10 \*\* (-1.5) for Aluminum.  You need to select the correct dataset for this.  train\_size = 5 for Ti and 3 for Al. |
| S1(A) | validation\_model | Append svm(data)? |
| S1(B) | validation\_model | Append nn(data)? |
| S2 | validation\_exp\_cross | Uncomment the training for residual connection. |
| S3 | validation\_model? | Like Figure S5, I’m not sure where to find the data. |
| S4 | validation\_exp  validation\_exp\_cross2 | Like figure 8. Uncomment df["dP/dh (N/m)"] \*= 0.2 / df["hm (um)"] in data.py. |
| S5(A-C) | validation\_model? | Append svm(data)? Not sure where to find data. |
| S5(D-F) | validation\_model? | Append nn(data)? Not sure where to find data. |

The code had several factors that needed to be hard-coded. For example, performing validation\_exp\_cros2 on titanium alloys for figure 8 required that the last two lines of ExpData in data.py be uncommented to scale c\* and dP/dh to correct units.