

ML models for computed band gap of HOIPs

Huan Tran, Georgia Institute of Technology

This notebook is a part of IV. N. Tunc, Nga, T. T. Nguyen, V. Sharma, and T. D. Huan, *Proba*

<https://doi.org/10.1103/PhysRevMaterials.5.125402>, and i

The original (raw) dataset containing the computed bandpass kernel of 1,346 atomic structures predicted for 192 chemical compositions of hybrid organic-inorganic perovskites (HOIPs) is available at [C. Kim, T.D. Huan, S. Krishnan, and P. Ramprasad, *Scientific Data*, 4, 170057 (17); <https://www.nature.com/articles/sdata170057>]. Here, three fingerprinted versions of this dataset (S1, S2, and S3) will be fetched from <http://www.matsml.org/> and learned to develop 5 ML models (M1, M2, M3, M4, and M5), which are based on Gaussian Process Regression, fully connected Neural Net, and Probability Neural Net. Computations performed using matsml toolkit, available at <https://github.com/nuand/matsml.git>.

Among 5 models developed, M5 demonstrates a reasonable way to handle the aleatoric uncertainty in deep learning materials data. More details on this topic can be found in “*Probabilistic deep learning approach for targeted hybrid organic-inorganic perovskites*”, the reference mentioned above.

1. Download data

Three (fingerprinted) datasets (S1, S2, and S3) used for the work will be obtained. In fact, S2 has 2 versions, one with selector and one not.

```
from natnml.data import datasets
data = Datasets(
    S1='fr noisr S1 1dest',
```

```
matsML, v1.3.0
*****
Load requested data
Data saved in fp_hoi
Data saved in fp_hoi
```

```
Data saved in fp_hoips_S2b_1dest.csv.gz
Data saved in fp_hoips_S3_4tfp.csv.gz
```

2. Obtained datasets parameters

```
# data parameters for learning
```

```

n_train = 0.9 # 90% for training, 10% for test
sampling = 'random' # method for train/test splitting
x_scaling = 'nimma' # method for x scaling
y_scaling = 'nimma' # method for y scaling

# Dict of data parameters
data1_params = {
    'data_file': 'fp_hoips_S1_1dest.csv.gz',
    'id_col': ['ID'],
    'y_cols': ['Ymean'],
    'comment_cols': [],
    'y_scaling': y_scaling,
    'x_scaling': x_scaling,
    'sampling': sampling,
    'n_trains': n_trains,
}

data2a_params = {
    'data_file': 'fp_hoips_S2a_2dest.csv.gz',
    'id_col': ['ID'],
    'y_cols': ['Ymean', 'Ystd'],
    'comment_cols': [],
    'y_scaling': y_scaling,
    'x_scaling': x_scaling,
    'sampling': sampling,
    'n_trains': n_trains,
}

data2b_params = {
    'data_file': 'fp_hoips_S2b_1dest.csv.gz',
    'id_col': ['ID'],
    'y_cols': ['prop_value'],
    'comment_cols': ['Ymean', 'Ystd', 'hid'],
    'y_scaling': y_scaling,
    'x_scaling': x_scaling,
    'sampling': sampling,
    'n_trains': n_trains,
}

```

```

x_scaling : x_scaling,
'sampling': sampling,
'n_trains': n_trains,
}

```

```
data_params = {
    'data_file': 'fp_hoips_S3_4tfp.csv.gz',
    'id_col': ['ID'],
    'y_cols': ['Egap'],
    'comment_cols': [],
    'x_scaling': x_scaling,
    'y_scaling': 'none',
    'sampling': sampling,
    'n_train': 1.0,
}

3. ML Models

3a. Model M1: GPR on S1

from matml.models import GPR

# Model parameters
nfold_cv = 5 # Number of folds for cross validation
model_file = 'M1.pkl' # Name of the model file to be created
verbosity = 0
rmse_cv = False
n_restarts_optimizer = 100

model_params = {
    'nfold_cv': nfold_cv,
    'n_restarts_optimizer': n_restarts_optimizer,
    'model_file': model_file,
    'verbosity': verbosity,
    'rmse_cv': rmse_cv,
}

model = GPR(data_params=data_params, model_params=model_params)
model.train()
model.plot(pdf_output=False)

Checking parameters
all passed True

Learning fingerprinted/featured data
algorithm gaussian process regression w/ scikit-learn
kernel RBF
nfold_cv 5
optimizer fmin_l_bfgs_b
n_restarts_optimizer 100
noise_lb 0.1
noise_ub 10
rmse_cv False

Read data
data file fp_hoips_S1_idtest.csv.gz
data size 192
training size 89.6 %
test size 10.4 %
x dimensionality 32
y dimensionality 1
y label(s) ['Ymean']

Scaling x minmax
xscaler saved in xscaler.pkl

Scaling y minmax

Prepare train/test sets random

Training model w/ cross validation
cv.rmse_train,rmse_test,rmse_opt: 0 0.054013 0.084918 0.084918
cv.rmse_train,rmse_test,rmse_opt: 1 0.052865 0.060944 0.060944
cv.rmse_train,rmse_test,rmse_opt: 2 0.057390 0.053613 0.053613
cv.rmse_train,rmse_test,rmse_opt: 3 0.055018 0.061227 0.053613
cv.rmse_train,rmse_test,rmse_opt: 4 0.055892 0.053427 0.053427

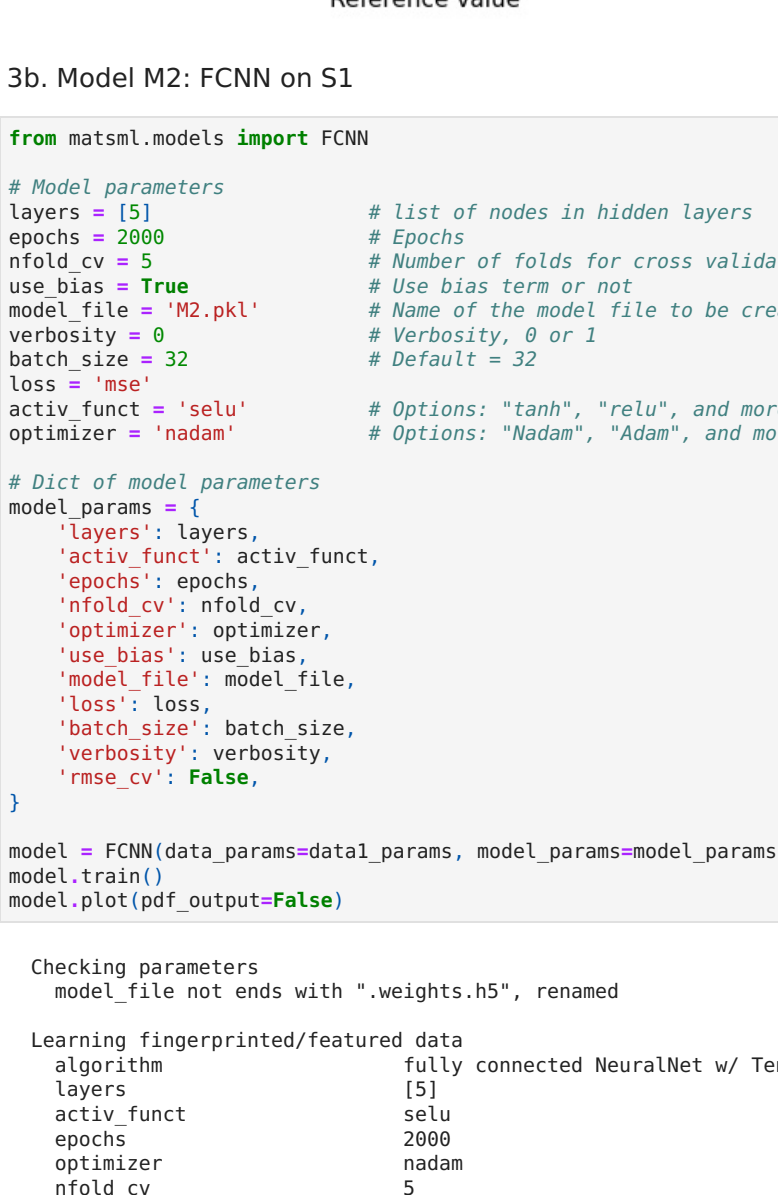
GPR model trained, now make predictions & invert scaling
unscaled y: minmax
rmse training Ymean 0.232351
unscaled y: minmax
rmse test Ymean 0.228549

Predictions made & saved in 'training.csv' & 'test.csv'
```

Plot results
training, (

A scatter plot showing the relationship between Predicted value (Y-axis) and Reference value (X-axis). The X-axis ranges from 2 to 6, and the Y-axis ranges from 2 to 6. A green diagonal line represents the ideal prediction (y=x). Red squares represent training data points, and blue circles represent test data points. Both sets of points are tightly clustered around the green line, indicating high predictive accuracy. A legend in the bottom right corner provides the following statistics:

- training, (mse & R²) = (0.232 & 0.951)
- test, (mse & R²) = (0.229 & 0.946)



```

verbosity
Read data
data file

```

```

data size          192
training size      89.6 %
test size         10.4 %
x dimensionality   32
y dimensionality   1
y label(s)        ['Ymean']

Scaling x          minmax
xscaler saved in  xscaler.pkl

Scaling y          minmax

Prepare train/test sets      random

Building model          FCNN

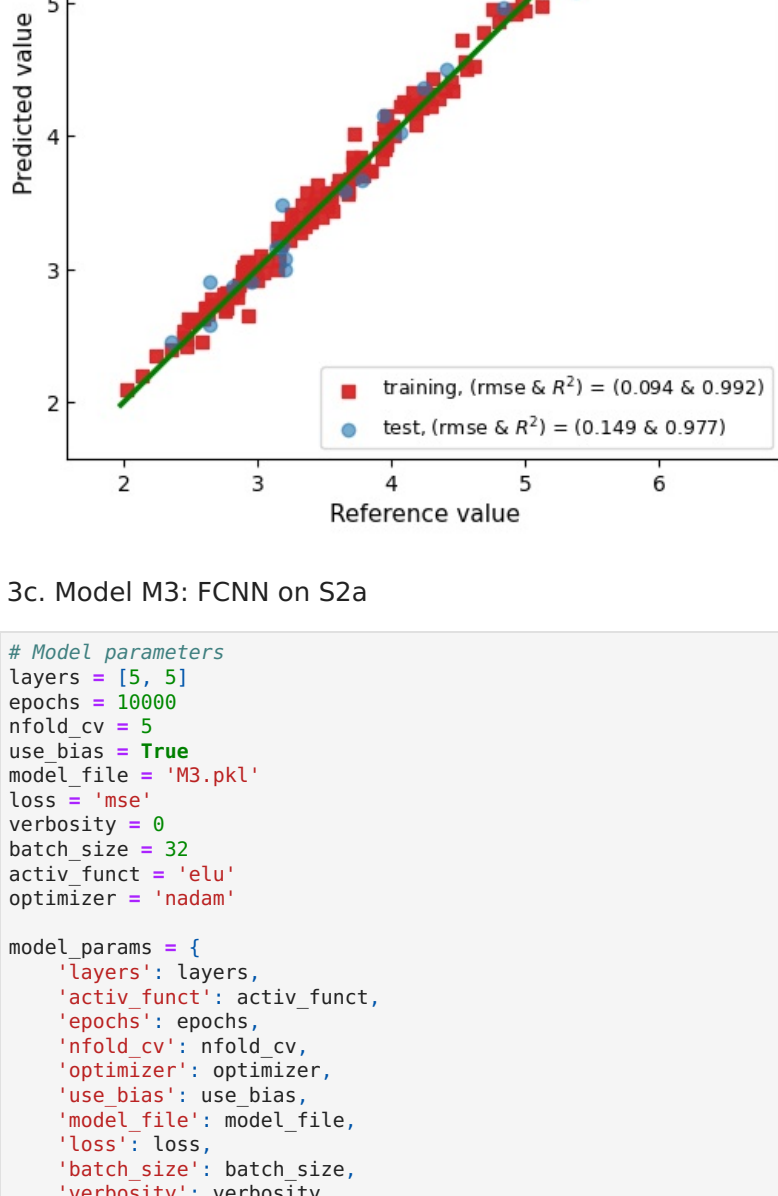
WARNING: All log messages before absl:InitializeLog() is called are written to STDERR
I0000 00:00:00.39302:201809 104287 gpu_device.cc:2020] Created device (/job:localhost/yplica:0/task:0/
GPU:0) with 4598 MB memory:  -> device: 0, name: NVIDIA GeForce RTX 2060, pci bus id: 0000:01:00.0, com
patibility: 7.5

Training model w/ cross validation

I0000 00:00:00.1765119302.948452 1042391 device_compiler.h:196] Compiled cluster using XLA! This line is log
t most once for the lifetime of the process.
5/5 [=====] -> 675us/step
2/2 [=====] -> 675us/step
cv_rmse_train_rmse_test_rmse_opt: 0. 0.826881 0. 0.629667 0. 0.629667
2/2 [=====] -> 675us/step
cv_rmse_train_rmse_test_rmse_opt: 0. 675us/step
2/2 [=====] -> 1ms/step
cv_rmse_train_rmse_test_rmse_opt: 1. 0.819770 0. 0.645732 0. 0.629667
5/5 [=====] -> 525us/step
2/2 [=====] -> 873us/step
cv_rmse_train_rmse_test_rmse_opt: 2. 0.819519 0. 0.641122 0. 0.629667
5/5 [=====] -> 543us/step
2/2 [=====] -> 739us/step
cv_rmse_train_rmse_test_rmse_opt: 3. 0.821669 0. 0.622021 0. 0.622021
5/5 [=====] -> 507us/step
2/2 [=====] -> 668us/step
cv_rmse_train_rmse_test_rmse_opt: 4. 0.816885 0. 0.60160 0. 0.622021
Optimal cvc: 3 ; optimal NET saved
FCNN trained, now make predictions & invert scaling
6/6 [=====] -> 499us/step
unscaled y: minmax
rmse training      Ymean      0.093519
unscaled y: minmax
rmse test         Ymean      0.149188
Predictions made in "training.csv" & "test.csv"

Plot results in "training.csv" & "test.csv"
training, (rmse & R2) = ( 0.094 & 0.992 )
test, (rmse & R2) = ( 0.149 & 0.977 )
showing Ymean

```



```

    'rmse_cv': False,
}

```

```

model = FNN(data_params=data_params, model_params=model_params)
model.plot(pdf_output=False)

Checking parameters
model_file not ends with ".weights.hs", renamed

Learning fingerprinted/featured data
algorithm          fully connected NeuralNet w/ TensorFlow
layers             [5, 5]
activation funct    elu
epochs             18000
optimizer          nadam
nfold_cv           5
verbosity          0

Read data
data file          fp_hoisps_S2a_2dest.csv.gz
data size         192
training size     89.6 %
test size        10.4 %
x dimensionality  31
y dimensionality  2
y label(s)       ['Ymean', 'Ystd'd']

Scaling x
xscaler saved in  xscaler.pkl

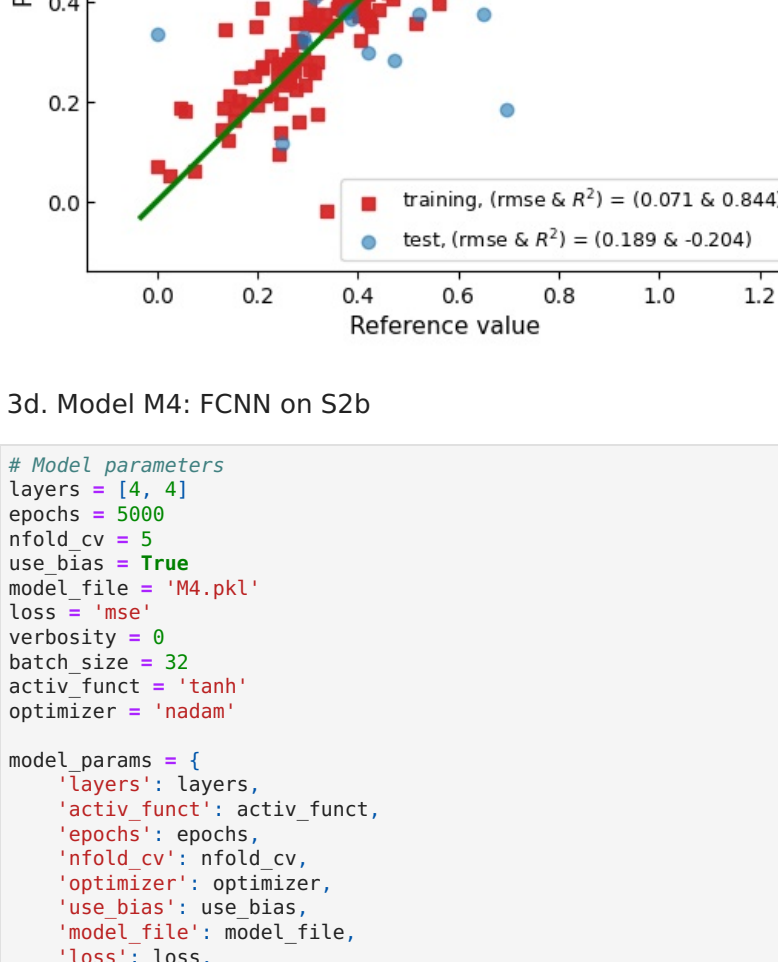
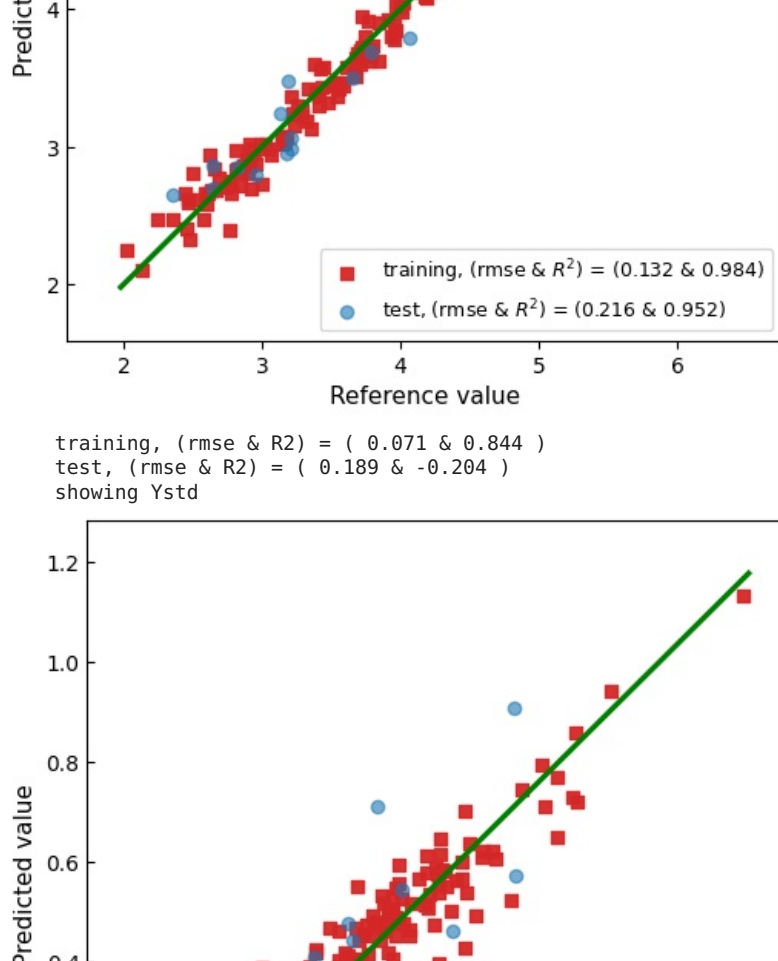
Scaling y
yminmax

Prepare train/test sets
random

Building model
FCNN
Training model w/ cross validation
5/5 [=====] 0s 576us/step
2/2 [=====] 0s 897us/step
cv_rmse_train,rmse_test,rmse_opt: 0 0.037621 0.139731 0.139731
5/5 [=====] 0s 601us/step
2/2 [=====] 0s 995us/step
cv_rmse_train,rmse_test,rmse_opt: 1 0.040474 0.070754 0.070754
5/5 [=====] 0s 532us/step
2/2 [=====] 0s 839us/step
cv_rmse_train,rmse_test,rmse_opt: 2 0.035891 0.097834 0.070754
5/5 [=====] 0s 549us/step
2/2 [=====] 0s 849us/step
cv_rmse_train,rmse_test,rmse_opt: 3 0.034440 0.088674 0.070754
5/5 [=====] 0s 617us/step
2/2 [=====] 0s 918us/step
cv_rmse_train,rmse_test,rmse_opt: 4 0.034640 0.084127 0.070754
Optimal ncv: 1; optimal NET saved
FCNN trained, now make predictions & invert scaling
6/6 [=====] 0s 568us/step
unscaled y: minmax
rmse training Ymean 0.131627
rmse training Ystd 0.071059
1/1 [=====] 0s 12ms/step
unscaled y: minmax
rmse test Ymean 0.21622
rmse test Ystd 0.189387
Predictions made & saved in 'training.csv' & 'test.csv'

Plot results in "training.csv" & "test.csv"
training: (rmse & R2) = ( 0.132 & 0.984 )
test: (rmse & R2) = ( 0.216 & 0.952 )
showing Ymean

```



```
'batch_size': batch_size,
'verbosity': verbosity,
'rmse_cv': False,
```

```

model = FCNN(data_params=datap2b_params, model_params=model_params)
model.train()
model.plot(pdf_output=False)

Checking parameters
model_file not ends with ".weights.hs", renamed

Learning fingerprinted/featured data
algorithm fully connected NeuralNet w/ TensorFlow
layers [4, 4]
activ_funct tanh
epochs 5800
optimizer nadam
nfold_cv 5
verbosity 0

Read data
data file fp_hoips_S2b_1dtest.csv.gz
data size 384
training size 89.8 %
test size 10.2 %
x dimensionality 53
y dimensionality 1
y label(s) ['prop_value']

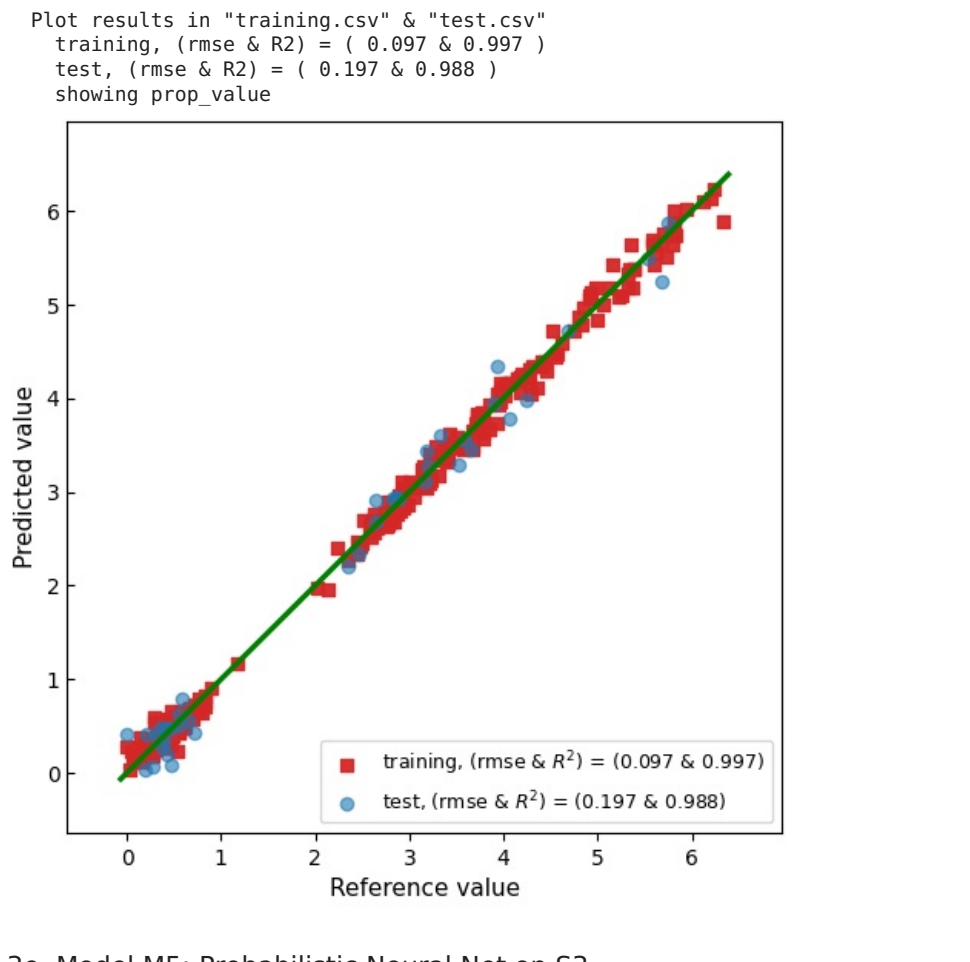
Scaling x
xscaler saved in xscaler.pkl

Scaling y
minmax

Prepare train/test sets random

Building model FCNN
Training model w/ cross validation
9/9 [=====] - 0s 459us/step
3/3 [=====] - 0s 630us/step
cv_rmse_train,rmse_test,rmse_opt: 0 0.052984 0.087306 0.087306
3/3 [=====] - 0s 452us/step
cv_rmse_train,rmse_test,rmse_opt: 0 0.0570us/step
cv_rmse_train,rmse_test,rmse_opt: 1 0.041848 0.104289 0.087306
9/9 [=====] - 0s 651us/step
cv_rmse_train,rmse_test,rmse_opt: 0 0.052us/step
cv_rmse_train,rmse_test,rmse_opt: 2 0.040184 0.110252 0.087306
9/9 [=====] - 0s 486us/step
cv_rmse_train,rmse_test,rmse_opt: 3 0.0687us/step
cv_rmse_train,rmse_test,rmse_opt: 3 0.042021 0.098261 0.087306
9/9 [=====] - 0s 526us/step
cv_rmse_train,rmse_test,rmse_opt: 4 0.0754us/step
cv_rmse_train,rmse_test,rmse_opt: 4 0.043746 0.070754 0.070754
optimal ncv: 4 ; optimal NET saved
FCNN trained, now make predictions & invert scaling
11/11 [=====] - 0s 494us/step
unscaled y: minmax
rmse training selector1 prop value 0.113456
rmse training selector2 prop value 0.076638
2/2 [=====] - 0s 895us/step
unscaled y: minmax
rmse test selector1 prop value 0.200466
rmse test selector2 prop value 0.193418
Predictions done & saved in 'training.csv' & 'test.csv'

```



```
from natsml.models import PrFCNN
```

```

layers = 5
epochs = 200
nfold cv = 10
use_bias = True
model_file = 'M5.pkl'
loss = 'mse'
verbosity = 0
batch_size = 32
activ_func = 'selu'
optimizer = 'nadam'

model_params = {
    'layers': layers,
    'activ_func': activ_func,
    'epochs': epochs,
    'nfold_cv': nfold_cv,
    'optimizer': optimizer,
    'use_bias': use_bias,
    'model_file': model_file,
    'loss': loss,
    'batch_size': batch_size,
    'verbosity': verbosity,
    'rmse_cv': False,
}

model = PrfCNN(data_params, model_params=model_params)
model.train()
model.plot(pdf_output=False)

```

```

Checking parameters
WARNING: "negloglik" must & will be used for loss

Learning fingerprinted/featured data
algorithm      Probabilistic NeuralNet w/ TensorFlow-Probability
layers        5
activ_func    selu
epochs        200
optimizer     nadam
loss          negloglik
nfold_cv      5

Read data
data file      fp_hoisps_53_4tfp.csv.gz
data size     1346
training size  100.0 %
test size     0.0 %
x dimensionality  221
y dimensionality  1
label(s)      ['Egap']

```

```
Scaling x                               minmax
xscaler saved in                       xscaler.pkl

Scaling y                               none

Prepare train/test sets                 random
Building model                         PrFCNN
Training PrFCNN w/ cross validation

34/34 [=====] - Bs 456us/step
9/9 [=====] - Os 49us/step
cv_rmse train_rmse test_rmse opt: 0.629529 0.608153 0.608153
34/34 [=====] - Bs 456us/step
```

```

9/9 [=====] - 0s 4830step
cv_mse_train, mse_test, mse_opt: 1.0 5.8991step 0.600550
3/4 [=====] - 0s 4830step
cv_mse_train, mse_test, mse_opt: 1.0 5.8991step 0.600550
3/4 [=====] - 0s 5440step
cv_mse_train, mse_test, mse_opt: 2.0 6.62281 0.64302 0.60153
3/4 [=====] - 0s 4605step
cv_mse_train, mse_test, mse_opt: 2.0 6.62281 0.64302 0.60153
3/4 [=====] - 0s 6205step
cv_mse_train, mse_test, mse_opt: 3.0 6.63289 0.64304 0.60153
3/4 [=====] - 0s 4385step
cv_mse_train, mse_test, mse_opt: 0.0 5.4115step
9/9 [=====] - 0s 4.63288 0.605294 0.605294
Optimal cvc: 4

```

```

PRF> trainB, h0d make predictions & invert scaling
unscaled y: none      Egap      0.41921
rmse training
Predictions made & saved in "training.csv"

Plot results in "training.csv" & "test.csv"
showing Egap

```

```
import io
import requests
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
%matplotlib inline

# read the trained data
```

```

# pd.read_csv('training.csv')

# Trained data contain 10 of 1346 cases, but we need the name of the organic cations A,
# cation B, and anions X also. They can be obtained here
sum_url = 'https://huntsid.github.io/data/getsum17_comp.csv'
mapping = pd.read_csv(io.StringIO(requests.get(sum_url).content.decode('utf-8'))))

# We will plot 16 compositions ASn13 of 16 cations A and Sn for B and I for X.
organics = [
    'Acetuedinium', 'Amonium', 'Azetidinium', 'Butylammonium',
    'Diethetlammonium', 'Ethylammonium', 'Formamidinium', 'Guanidinium',
    'Hydrazinium', 'Hydroxylammonium', 'Imidazolium', 'Isopropylammonium',
    'Methylammonium', 'Propylammonium', 'Tetraethylammonium'
]

```

```

    'Trimethylammonium'
]
cations = ['Sn']
anions = ['I']

comps = [
    (organic, cation, anion)
    for organic in organics
    for cation in cations
    for anion in anions
]

```

```
# DataFrame will be extracted from pred for identifying
bandgap_strs = pd.DataFrame(columns=['id', 'organic_cat', 'bandgap'])
bandgap_comp = pd.DataFrame(
    columns=['cid', 'mean_comput', 'std_comput', 'mean_pred', 'std_pred'])

# for each of 16 compounds, extract needed data
for cid in range(len(comps)):
    comp = comps[cid]
    sel_rows = mapping[
        [mapping['organic'] == comp[0]]
        & [mapping['cation'] == comp[1]]
        & [mapping['anion'] == comp[2]]
    ]
    rows = sel_rows[0:16]
    bandgap_strs = bandgap_strs.append(
        pd.DataFrame(
            columns=['id', 'organic_cat', 'bandgap'],
            data=[
                [cid, 'organic_cat', bandgap_strs['bandgap']]
            ]
        ),
        ignore_index=True
    )
    bandgap_comp = bandgap_comp.append(
        pd.DataFrame(
            columns=['cid', 'mean_comput', 'std_comput', 'mean_pred', 'std_pred'],
            data=[
                [cid, bandgap_comp['mean_comput'], bandgap_comp['std_comput'],
                 bandgap_comp['mean_pred'], bandgap_comp['std_pred']]
            ]
        ),
        ignore_index=True
    )
```

```

sel_ids = list(sel_rows['ID'])
sel_pred = pred['sel_rows']['ID'].isin(sel_ids)
sel_pred.reset_index(drop=True, inplace=True)

bandgap_comp.loc[(len(bandgap_comp) ==
cid,
np.mean(sel_pred['Egap']),
np.std(sel_pred['Egap']),
sel_pred.at[0, 'nd_Egap'],
sel_pred.at[0, 'nd_Egap_err'])
]

```

```
for idx, eq in zip(listsel_pred['ID'], listsel_pred['Eqag']):
    bandgap_strs.loc[(len(bandgap_strs)) = (cid, idx, comp[0], eq)]

# Make figure
fig, ax = plt.subplots(figsize=(8, 6), frameon=True)
plt.subplots_adjust(
    left=0.12, bottom=0.21, right=0.98, top=0.98, wspace=0, hspace=0
)

plt.rcParams['font.size'] = 16
plt.rcParams['font.family'] = 'serif'
plt.rcParams['font.serif'] = 'serif'
plt.rcParams['font.style'] = 'normal'
plt.rcParams['font.weight'] = 'normal'
plt.rcParams['font.variant'] = 'normal'
plt.rcParams['font.family'] = 'serif'
plt.rcParams['font.serif'] = 'serif'
plt.rcParams['font.style'] = 'normal'
plt.rcParams['font.weight'] = 'normal'
plt.rcParams['font.variant'] = 'normal'
```

```
plt.tick_params('x', which='both', bottom=True, top=True, labelbottom=True)
plt.tick_params('y', which='both', left=True, right=True, labelleft=True,
                direction='in', labelleft=True, length=5)

ax.set_ylim([0.0, 4.0])

plt.tick_params('x', which='both', direction='in', labelsize=12, top=True)
plt.tick_params('y', which='both', direction='in', labelsize=12, right=True)
ax.set_xticks(np.arange(0, 16, 1))
ax.set_yticks(labels['organisms'], rotation=35, ha='right')
```

```
plt.ylabel(r"$\Sigma_{\nu}(\nu)$ (eV)", color="black", fontsize=18)

ax.scatter(
    bandgap_strs['cid'], bandgap_strs['bandgap'],
    color='royalblue', alpha=0.75, zorder=3,
    label='computed data'
)

ax.errorbar(
    bandgap_comp['cid'], bandgap_comp['mean comput'],
    yerr=bandgap_comp['std comput'], capsize=3,
    color='darkgoldenrod', alpha=0.5, zorder=3,
)
```

```

markersize=5, face='s')
label=r'computed SE \(\rm g\)^{-1}(\rm mean) \pm E \(\rm g\)^{-1}(\rm std)\$'

ax.plot(
    bandgap_comp['cid'], bandgap_comp['mean_pred'],
    color='tab:red', linewidth=2,
    label=r'predicted SE \(\rm g\)^{-1}(\rm mean)\$'
)

ax.fill_between(
    bandgap_comp['cid'],
    bandgap_comp['lower_bound'], ~ bandgap_comp['std_upper']
)

```

```
bandgap_comp['mean_pred'] + 2 * bandgap_comp['std_pred'],
color='#2ca02c', alpha=0.15,
label=r'predicted SE_{\rm g}^{(2)}({\rm mean} \pm 2 {\rm SE}_{\rm g}^{(2)}({\rm std})')
)

handles, labels = ax.get_legend_handles_labels()
handles, labels = zip(*sorted(zip(labels, handles), key=lambda t: t[0]))
ax.legend(handles, labels, loc='lower right', fontsize=13)

<matplotlib.legend.Legend at 0x1533718e3390>
```

Figure 10 is a plot showing the error E_g (eV) versus the number of layers N . The plot compares computed data (blue dots) with a predicted range (green shaded area). The computed data points are generally within the predicted range, which is bounded by $E_g^{\text{mean}} \pm E_g^{\text{std}}$ (orange line) and $E_g^{\text{mean}} \pm 2E_g^{\text{std}}$ (green area). The error E_g decreases as N increases, with the predicted range narrowing around $N=10$.

Ammonium Salt	Predicted E_{mean} (eV)
Acetammonium	0.4
Ammonium	0.4
Azidammonium	0.4
Butylammonium	0.4
Dimethylammonium	0.4
Ethylammonium	0.4
Formammonium	0.4
Guanidinium	0.4
Hydroxylammonium	0.4
Imidazolium	0.4
Isopropylammonium	0.4
Methylammonium	0.4
Propylammonium	0.4
Tetramethylammonium	0.4
Trimethylammonium	0.4

Fig. 1. Electronic band gap E_g (circles) computed for the predicted atomic structures of $\text{AsSn}_{1-x}\text{Sb}_x$ 16 HOIP formulas corresponding to 16 organic cations A. For each formula, the mean and standard deviation of E_g i.e., $E_g^{\text{predicted}}$ and E_g^{std} are given by dark golden squares and associated errorbars. Predicted E_g^{mean} is given in red while the shaded area indicates the 95-percent confidence interval ($E_g^{\text{mean}} \pm 2\sigma_{E_g}$) of the predictions using the probabilistic model developed in this work.