

# Supervised Learning: Neural Networks

## In this module

- Introduction to neural networks for materials science
- Hands on tutorial using nanoHUB: neural networks for XX (this file)
  - Hands on tutorial using nanoHUB: neural networks for XX
    - Homework assignments

Saaketh Desai and Alejandro Strachan

[desai61@purdue.edu](mailto:desai61@purdue.edu) || [strachan@purdue.edu](mailto:strachan@purdue.edu)

School of Materials Engineering & Network for Computational Nanotechnology

Purdue University

West Lafayette, Indiana USA



# Learning objectives and prerequisites

---

After completing this lecture you will:

- Be able to create and train a neural network
- Be able to define objective functions for regression and classification tasks
- Know how to determine overfitting and underfitting in training neural networks

Pre-requisites:

- Basic Python programming
- Querying materials repositories
- Linear regression

# Launching a Jupyter tool in nanoHUB

## Machine Learning for Materials Science: Part 1

From your browser go to link: <https://nanohub.org/tools/mseml/>

**Machine Learning for Materials Science: Part 1** Collect

By [Juan Carlos Verduzco Gastelum<sup>1</sup>](#), [Alejandro Strachan<sup>1</sup>](#), [Saaketh Desai<sup>1</sup>](#)  
*1. Purdue University*

Machine learning and data science tools applied to materials science

[Edit](#)

**Launch Tool**

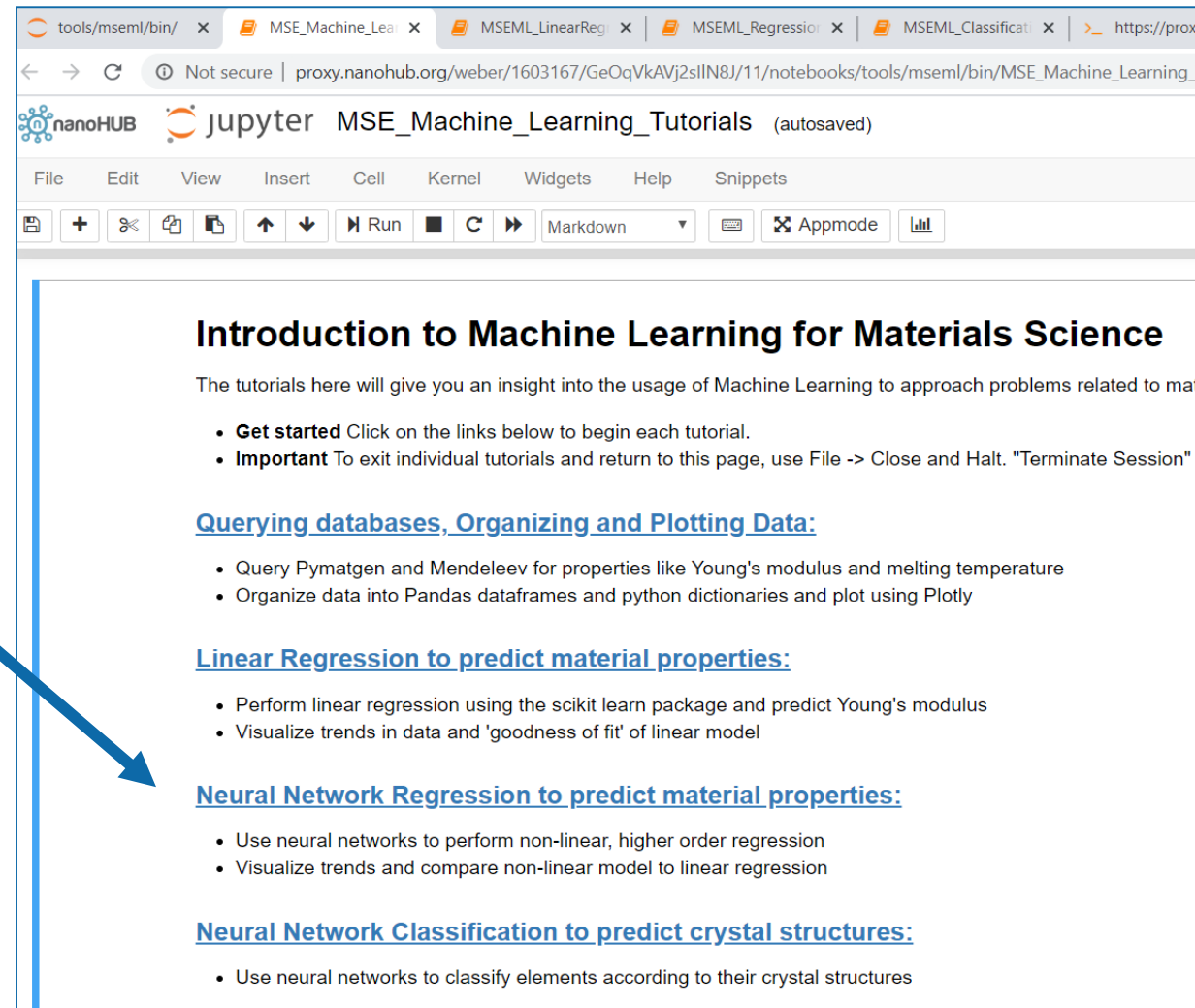
Version 1.1 - published on 25 Feb 2019  
doi:10.21981/9QJN-7N65 [cite this](#)  
Open source: [license](#) | [download](#)  
[View All Supporting Documents](#)

👤 1087 users, [detailed usage](#)  
🗉 0 Citation(s)  
💬 0 questions ([Ask a question](#))  
★ 1 review(s)  
👤 0 wish(es) ([New Wish](#))  
→ Share: [f](#) [t](#) [s](#) ...

Click on Launch Tool to begin



# Step 1: Landing Page – Notebook: Neural Network Regression

Navigate to the third link in the landing page to access the notebook















tools/mseml/bin/ x MSE\_Machine\_Lea x MSEML\_LinearReg x MSEML\_Regressio x MSEML\_Classificati x https://prox

← → ↻ Not secure | proxy.nanohub.org/weber/1603167/GeOqVkJ2sllN8J/11/notebooks/tools/mseml/bin/MSE\_Machine\_Learning\_

  MSE\_Machine\_Learning\_Tutorials (autosaved)

File Edit View Insert Cell Kernel Widgets Help Snippets

          Markdown  Appmode 

## Introduction to Machine Learning for Materials Science

The tutorials here will give you an insight into the usage of Machine Learning to approach problems related to ma

- **Get started** Click on the links below to begin each tutorial.
- **Important** To exit individual tutorials and return to this page, use File -> Close and Halt. "Terminate Session"

### [Querying databases, Organizing and Plotting Data:](#)

- Query Pymatgen and Mendeleev for properties like Young's modulus and melting temperature
- Organize data into Pandas dataframes and python dictionaries and plot using Plotly

### [Linear Regression to predict material properties:](#)

- Perform linear regression using the scikit learn package and predict Young's modulus
- Visualize trends in data and 'goodness of fit' of linear model

### [Neural Network Regression to predict material properties:](#)

- Use neural networks to perform non-linear, higher order regression
- Visualize trends and compare non-linear model to linear regression

### [Neural Network Classification to predict crystal structures:](#)

- Use neural networks to classify elements according to their crystal structures

# Step 2: Let's get some data

## 1. Getting a dataset

```
import pymatgen as pymat
import mendeleev as mendel
import pandas as pd
import numpy as np
import random
import tensorflow as tf
from tensorflow import keras
from keras import initializers
from keras.layers import Dense
from keras.models import Sequential
%matplotlib inline
import matplotlib.pyplot as plt
import sys

fcc_elements = ["Ag", "Al", "Au", "Cu", "Ir", "Ni", "Pb", "Pd", "Pt", "Rh", "Th", "Yb"]
bcc_elements = ["Ba", "Ca", "Cr", "Cs", "Eu", "Fe", "Li", "Mn", "Mo", "Na", "Nb", "Rb", "Ta", "V", "W"]
hcp_elements = ["Be", "Cd", "Co", "Dy", "Er", "Gd", "Hf", "Ho", "Lu", "Mg", "Re",
               "Ru", "Sc", "Tb", "Ti", "Tl", "Tm", "Y", "Zn", "Zr"]

elements = fcc_elements + bcc_elements + hcp_elements

random.Random(1).shuffle(elements)

querable_mendeleev = ["atomic_number", "atomic_volume", "boiling_point", "en_ghosh", "evaporation_heat", "heat_of_formation",
                     "lattice_constant", "melting_point", "specific_heat"]
querable_pymatgen = ["atomic_mass", "atomic_radius", "electrical_resistivity", "molar_volume", "bulk_modulus", "youngs_modulus",
                    "average_ionic_radius", "density_of_solid", "coefficient_of_linear_thermal_expansion"]
querable_values = querable_mendeleev + querable_pymatgen
```

	atomic_number	atomic_volume	boiling_point	en_ghosh	evaporation_heat	heat_of_formation	lattice
0	27	6.70	3143.0	0.143236	389.1	426.7	
1	69	18.10	2220.0	0.216724	232.0	232.2	
2	39	19.80	3611.0	0.121699	367.0	424.7	
3	75	8.85	5900.0	0.243516	704.0	774.0	
4	28	6.60	3005.0	0.147207	378.6	430.1	
5	67	18.70	2968.0	0.207795	301.0	300.6	
6	79	10.20	3080.0	0.261370	340.0	368.2	
7	21	15.00	3104.0	0.119383	332.7	377.8	
8	45	8.30	4000.0	0.140838	494.0	556.0	
9	74	9.53	5930.0	0.239050	824.0	851.0	

Use Keras to train neural networks

Keras: <https://keras.io/>

Query Pymatgen and Mendeleev for atomic number, melting point etc.

Organize data into a Pandas Dataframe

Pandas: <https://pandas.pydata.org/>

# Step 3: Preprocess data and create network

## 2. Processing and Organizing Data

```
all_values = [list(df.iloc[x]) for x in range(len(all_values))]

# List of Lists are turned into Numpy arrays to facilitate calculations in steps to follow (Normalization).
all_values = np.array(all_values, dtype = float)
print("Shape of Values:", all_values.shape)
all_labels = np.array(all_labels, dtype = int)
print("Shape of Labels:", all_labels.shape)

# Training Set
train_values = all_values[:40, :]
train_labels = all_labels[:40, :]

# Testing Set
test_values = all_values[-7:, :]
test_labels = all_labels[-7:, :]

# NORMALIZATION

mean = np.nanmean(train_values, axis = 0) # mean
std = np.nanstd(train_values, axis = 0) # standard deviation

train_values = (train_values - mean) / std # input scaling
test_values = (test_values - mean) / std # input scaling

print(train_values[0]) # print a sample entry from the training set
#print(train_labels[0])
```

Divide data into training and testing sets

Standard Score normalization

32 = # of neurons

Activation = activation function

[https://en.wikipedia.org/wiki/Activation\\_function](https://en.wikipedia.org/wiki/Activation_function)

## 3. Creating the Model

```
model = Sequential()
model.add(Dense(32, activation='relu', input_shape=(train_values.shape[1],), kernel_initializer=kernel_init))
model.add(Dense(64, activation='relu', kernel_initializer=kernel_init))
#model.add(Dense(64, activation='relu', kernel_initializer=kernel_init))
model.add(Dense(1, kernel_initializer=kernel_init))

# DEFINITION OF THE OPTIMIZER

optimizer = optimizers.RMSprop(0.002) # Root Mean Squared Propagation

# This line matches the optimizer to the model and states which metrics will evaluate the model's accuracy
model.compile(loss='mae', optimizer=optimizer, metrics=['mae'])
model.summary()
```

Define a model with two hidden layers and an output layer: **Uncomment the line to add a third hidden layer**

Loss function: Mean absolute error

# Step 4: Train and evaluate network

## TRAINING

```
# EPOCH REAL TIME COUNTER CLASS
class PrintEpNum(keras.callbacks.Callback): # This is a function for the Epoch Counter
    def on_epoch_end(self, epoch, logs):
        sys.stdout.flush()
        sys.stdout.write("Current Epoch: " + str(epoch+1) + " Training Loss: " + "%4f" %logs.get('loss') + '\n')

EPOCHS = 2000 # Number of EPOCHS

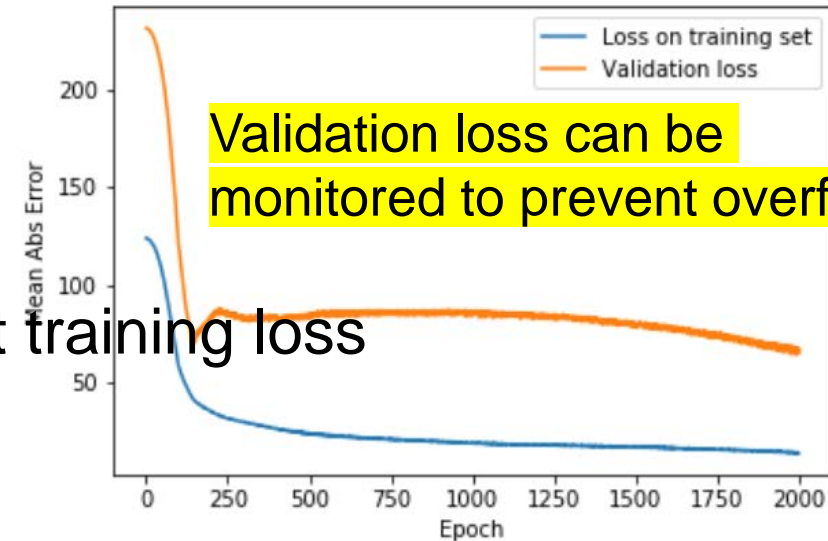
# HISTORY Object which contains how the model Learned

# Training Values (Properties), Training Labels (Known Young's Moduli)
history = model.fit(train_values, train_labels, batch_size=train_values.shape[0],
                    epochs=EPOCHS, verbose = False, validation_split=0.1, callbacks=[PrintEpNum()])

# PLOTTING HISTORY USING MATPLOTLIB

plt.figure()
plt.xlabel('Epoch')
plt.ylabel('Mean Abs Error')
plt.plot(history.epoch, np.array(history.history['mean_absolute_error']),label='Loss on training set')
plt.plot(history.epoch, np.array(history.history['val_mean_absolute_error']),label = 'Validation loss')
plt.legend()
plt.show()
```

model.fit(...) trains the network



Plot training loss

## TESTING

```
[loss, mae] = model.evaluate(test_values, test_labels, verbose=0)
print("Testing Set Mean Absolute Error: {:.2f} GPa".format(mae))

Testing Set Mean Absolute Error: 34.38 GPa

test_predictions = model.predict(test_values).flatten()
```

model.evaluate(...) evaluates the network

model.predict(...) makes predictions for a given dataset



# Step 4: Train and evaluate network

## TRAINING

```
# EPOCH REAL TIME COUNTER CLASS
class PrintEpNum(keras.callbacks.Callback): # This is a function for the Epoch Counter
    def on_epoch_end(self, epoch, logs):
        sys.stdout.flush()
        sys.stdout.write("Current Epoch: " + str(epoch+1) + " Training Loss: " + "%4f" %logs.get('loss') + '\n')

EPOCHS = 2000 # Number of EPOCHS

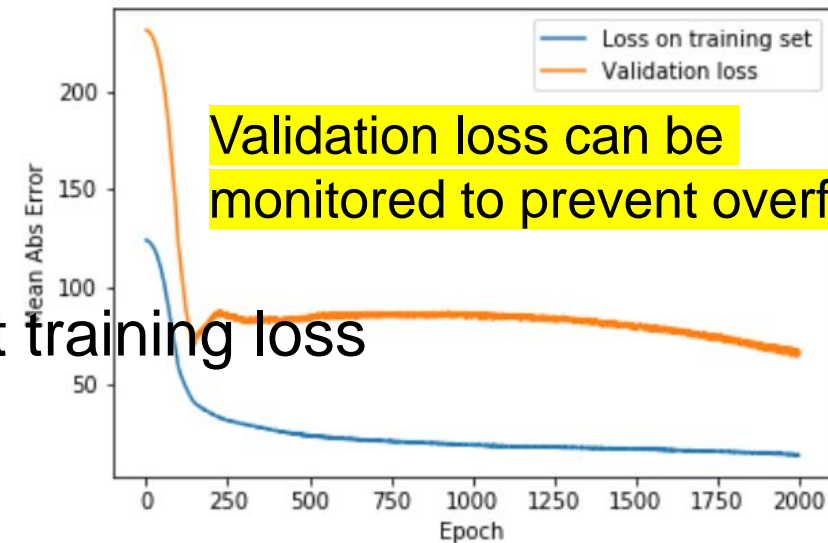
# HISTORY Object which contains how the model Learned

# Training Values (Properties), Training Labels (Known Young's Moduli)
history = model.fit(train_values, train_labels, batch_size=train_values.shape[0],
                    epochs=EPOCHS, verbose = False, validation_split=0.1, callbacks=[PrintEpNum()])

# PLOTTING HISTORY USING MATPLOTLIB

plt.figure()
plt.xlabel('Epoch')
plt.ylabel('Mean Abs Error')
plt.plot(history.epoch, np.array(history.history['mean_absolute_error']),label='Loss on training set')
plt.plot(history.epoch, np.array(history.history['val_mean_absolute_error']),label = 'Validation loss')
plt.legend()
plt.show()
```

model.fit(...) trains the network



Plot training loss

## TESTING

```
[loss, mae] = model.evaluate(test_values, test_labels, verbose=0)
print("Testing Set Mean Absolute Error: {:.2f} GPa".format(mae))

Testing Set Mean Absolute Error: 34.38 GPa

test_predictions = model.predict(test_values).flatten()
```

model.evaluate(...) evaluates the network

model.predict(...) makes predictions for a given dataset



# Plot results

