PyFit-FF Introduction

Speaker: J. Hickman*

Acknowledgments

G. Purja Pun**, V. Yamakov***, A. Robinson**, F. Tavazza*, Y. Mishin**

* National Institute of Standards and Technology (NIST)
Materials measurement laboratory (MML)

** George Mason University: Department of Physics

*** NASA

Supported by NRC postdoctoral fellowship

Talk Outline

- **Background**
  - Machine learning, numerical optimization, regression
  - Neural networks
  - PyTorch and automatic differentiation
- **PYFIT-FF**
  - Machine learning interatomic potentials
    - ANN, PINN,
  - Functionality and overview
- **Demonstration**
  - Installations process
  - Closer look at the code
    - Input and output files
    - README+Manual+Code details
  - Fitting example
  - Q & A

BACKGROUND

Machine learning

Algorithms

Supervised Learning

Algorithms learn a functional mapping between *pairs of known inputs and output*

→ **Regression**
curve fitting to continuous numeric variables

★ Neural networks as function interpolants
- Gaussian processes
- linear regression

→ **Classifications**
training models to identify known categories present in the data

- Logistic regression
- Classification trees (ensembles)
- Deep learning (neural networks)
- support vector machines (SVM)
- Naive Bayes algorithm

Unsupervised Learning

Algorithms find connections inherent in the data. This data has no known a priori structure *i.e. no known mapping between inputs and outputs*

→ **Clustering**

- K means
- K medians
- expectation maximization (EM)
- hierarchical clusters
- TSNE, PCA

→ **Dimensionality reduction**

- Feature selection: keep only important features
- Feature extraction: Find new features are created

Reinforcement Learning

Algorithms learn desired behaviors based on rules and some definition of desirable and undesirable outcomes

→ **The goal is to learn to choose decisions that produces the optimal outcomes**

- Examples: Robot navigation, real time decision making
- Markov decision process
  - set of nodes with penalties on each path
- Guided output based on programmed reinforcement and punishments

NIST

Numerical optimization

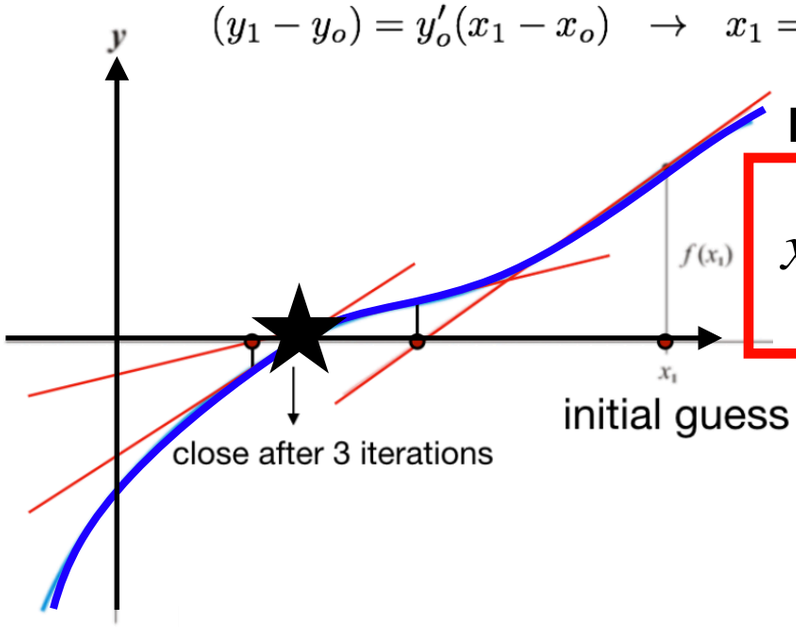
Numerical solvers:

$$f(x) = 0$$

Point slope equation of tangent line

$$(y_1 - y_o) = y'_o(x_1 - x_o) \quad \rightarrow \quad x_1 = x_o - \frac{y_o}{y'_o} \ (for\ y_1 = 0)$$

Find intercept

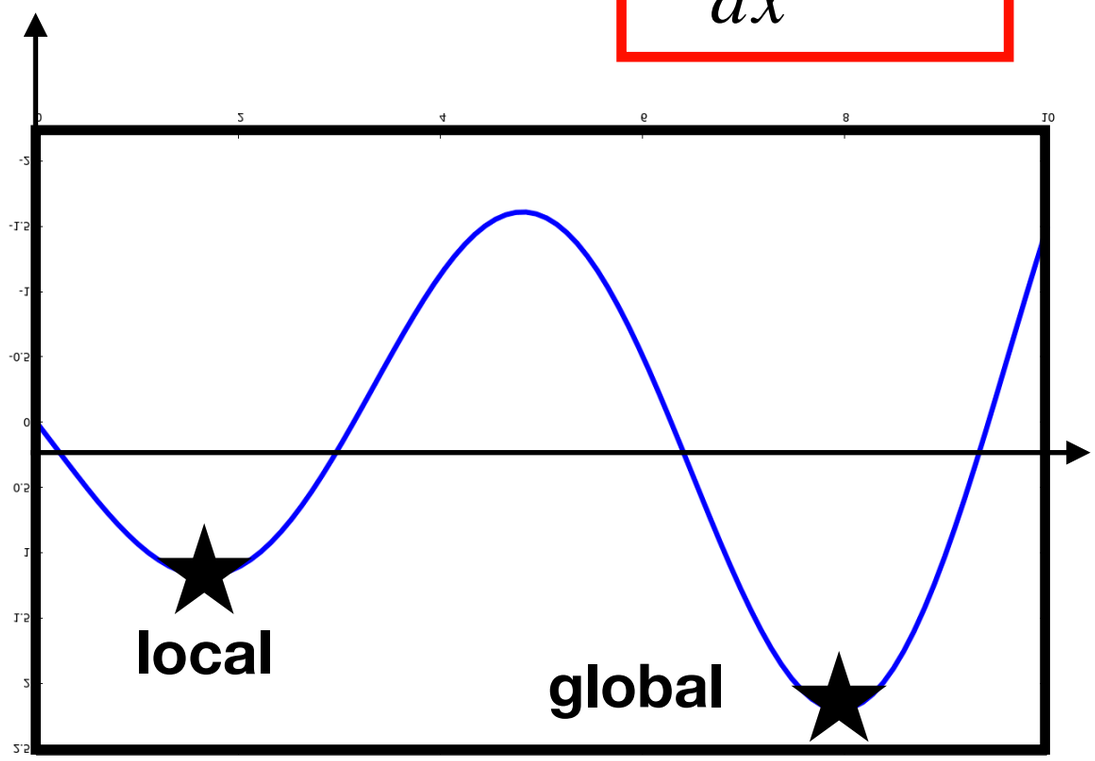


Newton's method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Numerical Optimizers:

$$min(f(x)) \rightarrow \frac{df(x)}{dx} = 0$$



Gradient based Optimizers:

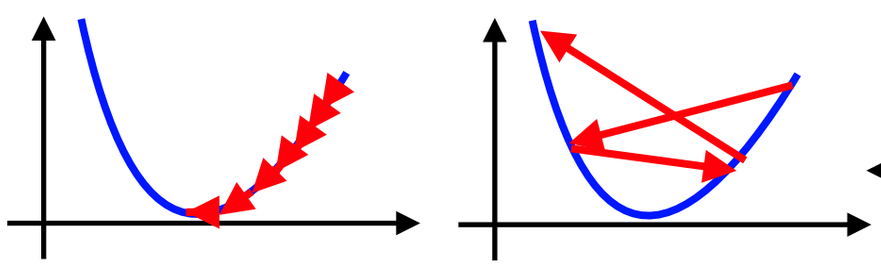
- Newton's method
- Secant method
- Gradient decent
- Davidon–Fletcher–Powell (DFP)
- Conjugate Gradient Method
- ★ Broyden–Fletcher–Goldfarb–Shanno (BFGS)

Newton's method $x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$

Gradient decent → $x_{n+1} = x_n - \lambda f'(x_n)$

Multivariable case: $\mathbf{x}_{n+1} = \mathbf{x}_n - \lambda \nabla F(\mathbf{x}_n)$

step size ("learning rate")



Regression

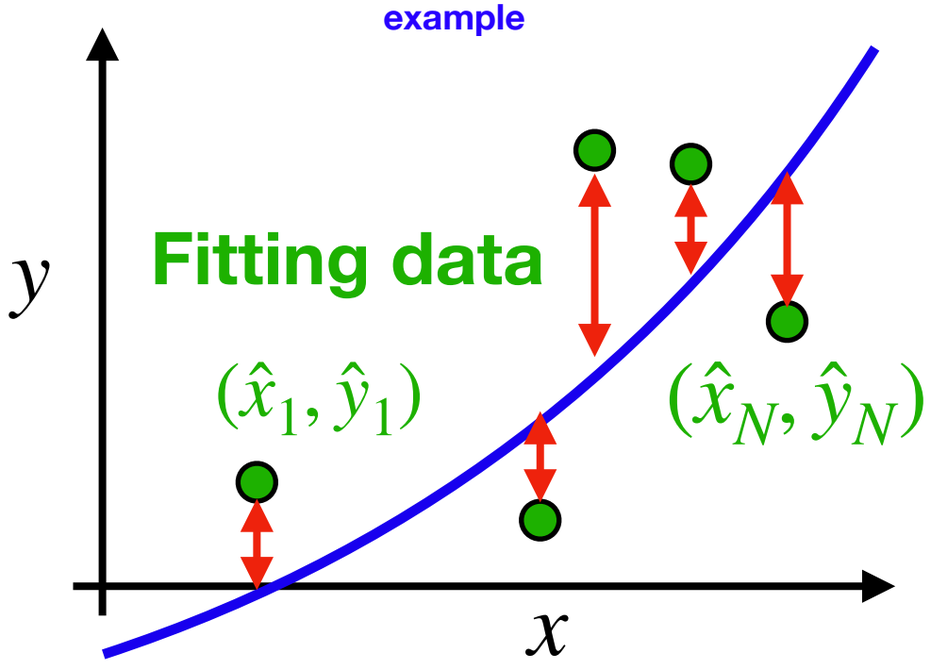
Fitting Model (interpolant)

$$y = F(x, \mathbf{w}) \rightarrow y = w_0 + w_1 x + w_2 x^2$$

example

Fitting Parameters

$$\mathbf{w} = (w_0, w_1, w_2)$$



Error Metric (objective)

RMSE:

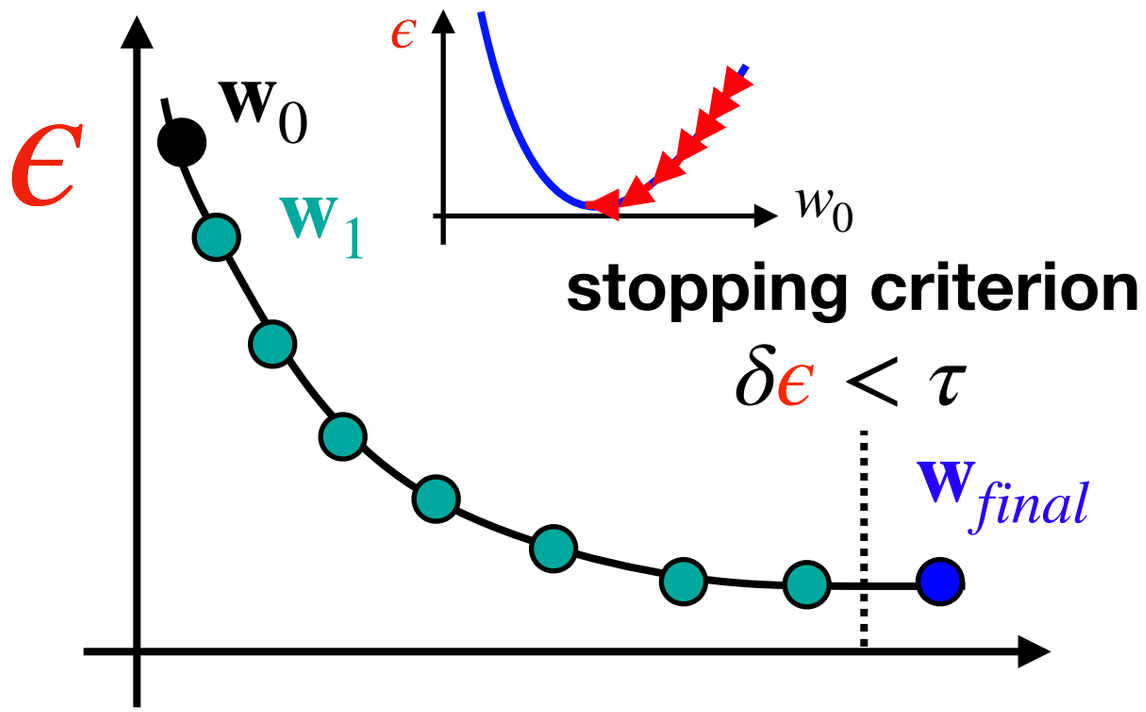
$$\epsilon(\mathbf{w}) = \sum_i \left(\frac{(y_i - \hat{y}_i)^2}{N} \right)^{\frac{1}{2}}$$

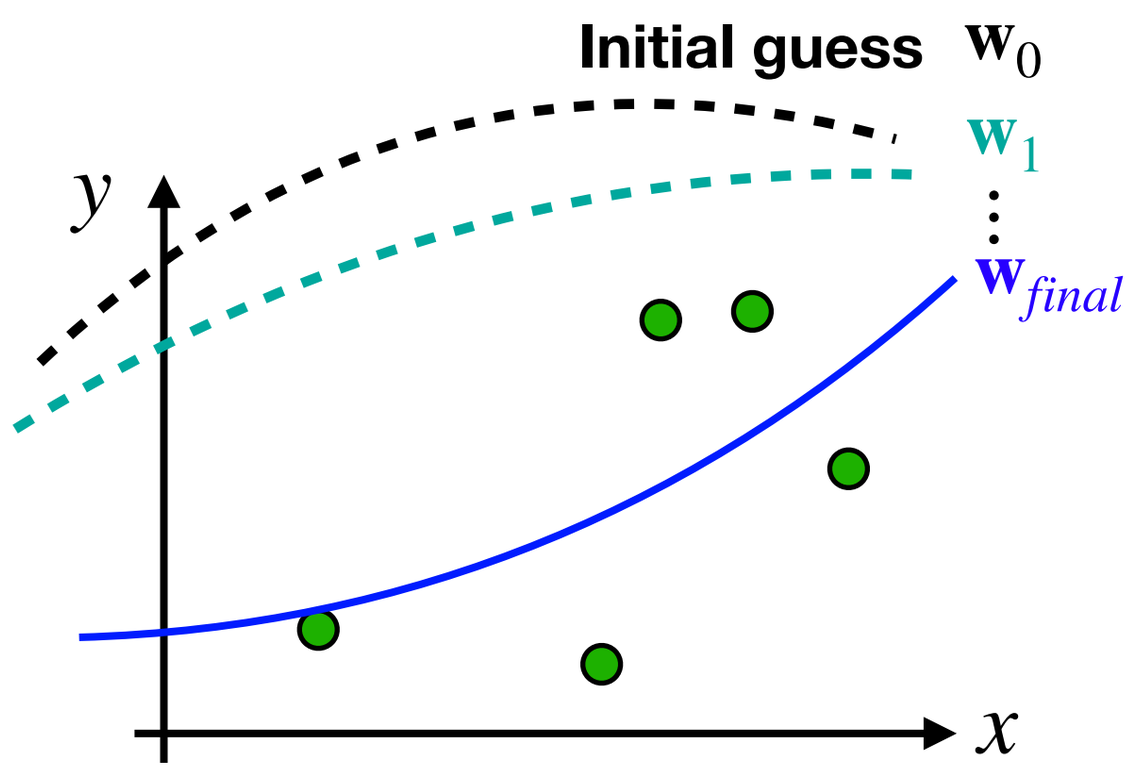
$$y_i = F(\hat{x}_i \,|\, \mathbf{w})$$

Alternatives: MAE,MSE, MEDIAN AE, ETC

Numerical optimization

minimize error
via fitting parameters

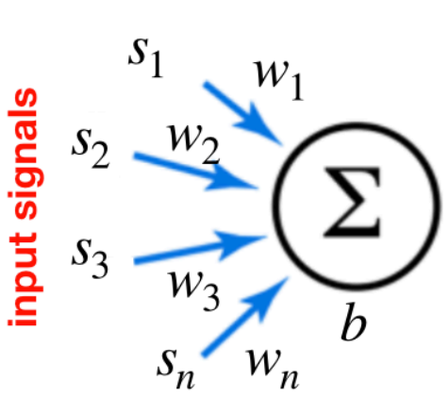




NIST

Neural networks: General idea

Artificial neuron



weighted and biased signal

$$s = \Sigma_i w_i s_i + b$$

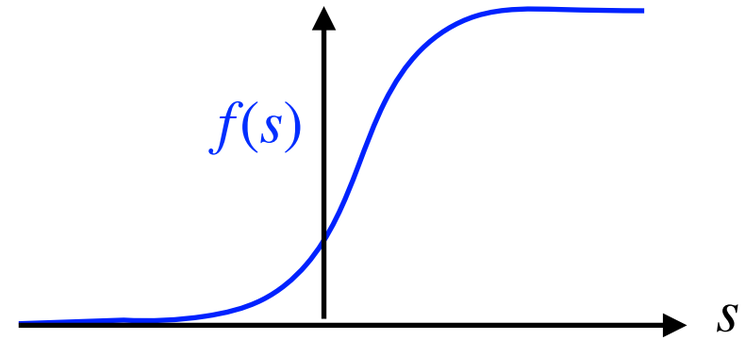
Fitting Parameters

Sigmoid Activation function

$$f(s) = \frac{1}{1 + e^{-s}}$$

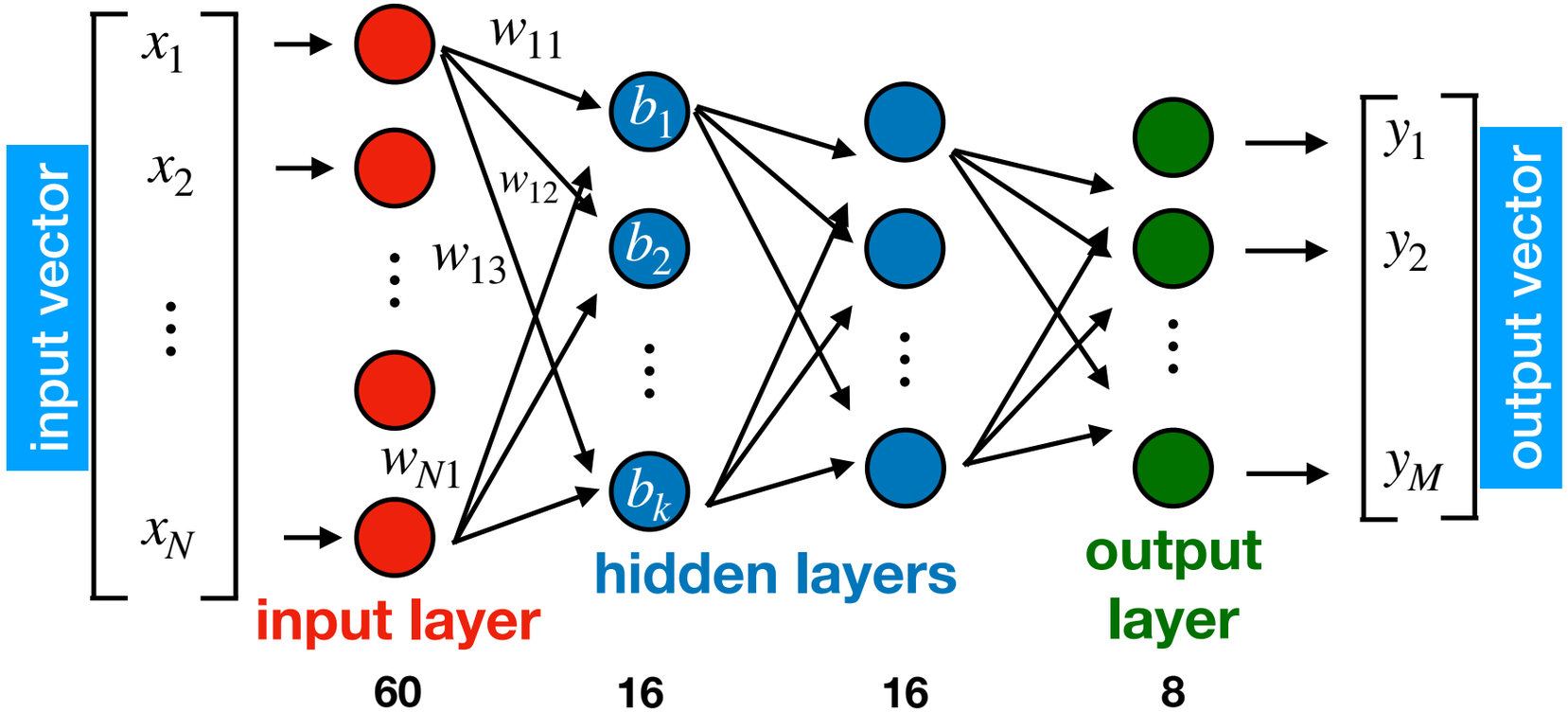
$$f(s)$$

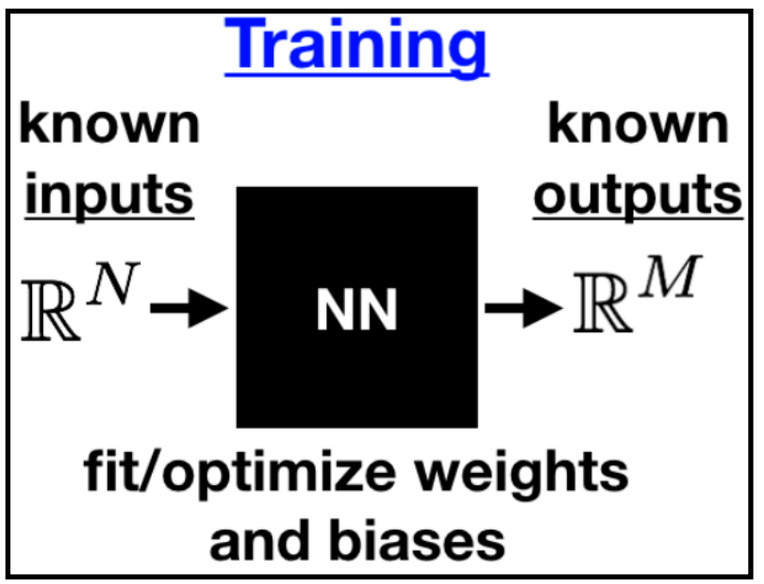
neuron output



Artificial neural network

NN=generalized high dimensional regression function





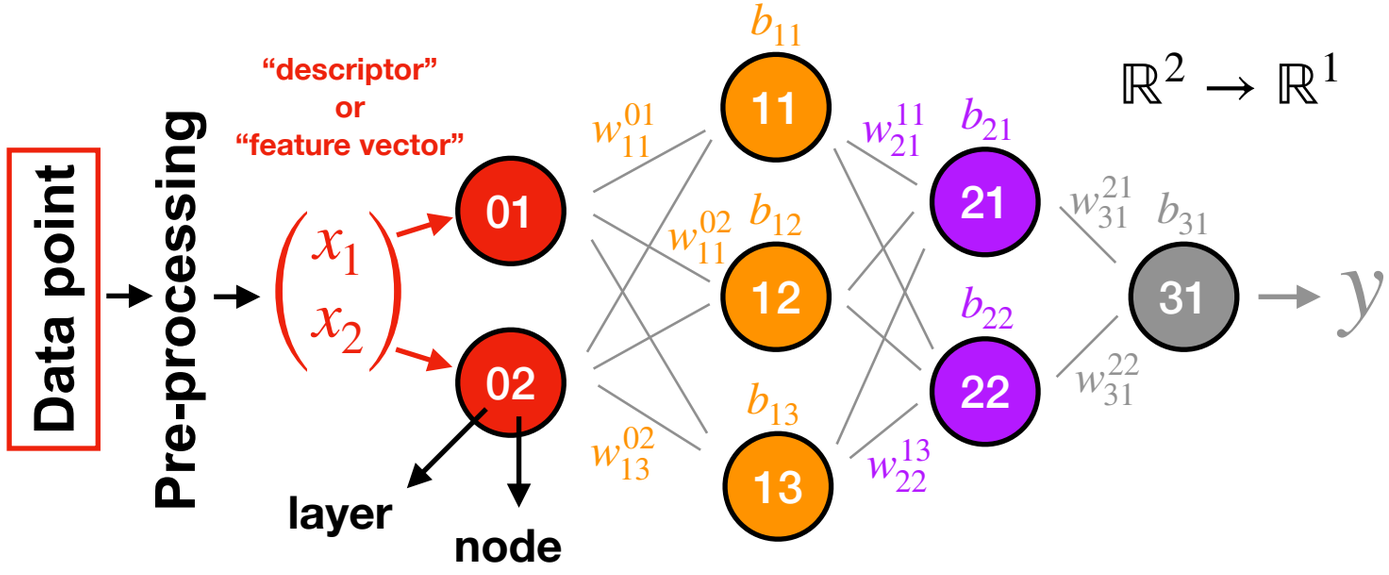
Neural network model

$$\mathbf{y} = NN(\mathbf{x}, \mathbf{w})$$

Quadratic model

$$y = F(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2$$

Neural networks: Implementation-1



NN architecture: 2-3-2-1

Number of fitting parameters:

$$N_{fit} = (3 \times 2 + 3) + (2 \times 3 + 2) + (2 \times 1 + 1) = 20$$

stack

$$\begin{pmatrix} w_{11}^{01} \\ w_{12}^{01} \\ w_{13}^{01} \\ w_{11}^{02} \\ w_{12}^{02} \\ w_{13}^{02} \\ b_{11} \\ b_{12} \\ b_{13} \\ w_{21}^{11} \\ w_{22}^{11} \\ w_{21}^{12} \\ w_{22}^{12} \\ w_{21}^{13} \\ w_{22}^{13} \\ b_{21} \\ b_{22} \\ w_{31}^{21} \\ w_{31}^{22} \\ b_{33} \end{pmatrix}$$

Activation function:

$$f(x) = \frac{1}{1 + e^x}$$

Input $\longmapsto$ $f(x)$ $\longmapsto$ $f(x)$ $\longmapsto$ **Output**

Hidden layer-1 output $\longmapsto$

$$\begin{pmatrix} O_{11} \\ O_{12} \\ O_{13} \end{pmatrix} = f\left(\begin{pmatrix} w_{11}^{01} & w_{11}^{02} \\ w_{12}^{01} & w_{12}^{02} \\ w_{13}^{01} & w_{13}^{02} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_{11} \\ b_{12} \\ b_{13} \end{pmatrix} \right)$$

apply activation function component wise

Hidden layer-2 output $\longmapsto$

$$\begin{pmatrix} O_{21} \\ O_{22} \end{pmatrix} = f\left(\begin{pmatrix} w_{21}^{11} & w_{21}^{12} & w_{21}^{13} \\ w_{22}^{11} & w_{22}^{12} & w_{22}^{13} \end{pmatrix} \begin{pmatrix} O_{11} \\ O_{12} \\ O_{13} \end{pmatrix} + \begin{pmatrix} b_{21} \\ b_{22} \end{pmatrix} \right)$$

Neural network output $\longmapsto$

$$y = (O_{31}) = \begin{pmatrix} w_{31}^{21} & w_{31}^{22} \end{pmatrix} \begin{pmatrix} O_{21} \\ O_{22} \end{pmatrix} + (b_{31})$$

Don't apply activation function on output layer

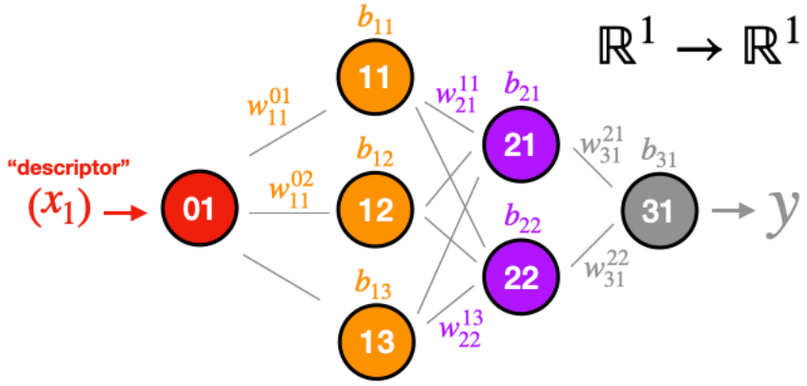
Matrix form:

$$y = \underline{\mathbf{w}}_{23} f\left(\underline{\mathbf{w}_{12}} f\left(\underline{\mathbf{w}_{01}} \mathbf{x} + \mathbf{b}_1 \right) + \mathbf{b}_2 \right) + \mathbf{b}_3$$

NIST

Neural networks: Training

Neural network fitting model



$$y = F(x, \mathbf{w})$$

$$y(x) = \underline{\mathbf{w}}_{23}\, f\left(\underline{\mathbf{w}}_{12}\, f\left(\underline{\mathbf{w}}_{01}\mathbf{x} + \mathbf{b}_1\right) + \mathbf{b}_2\right) + \mathbf{b}_3$$

Fitting parameters $\mathbf{w} =$

$$\begin{pmatrix} w_{11}^{01} \\ w_{12}^{01} \\ w_{13}^{01} \\ b_{11} \\ b_{12} \\ b_{13} \\ w_{21}^{11} \\ w_{22}^{11} \\ w_{21}^{12} \\ w_{22}^{12} \\ w_{21}^{13} \\ w_{22}^{13} \\ b_{21} \\ b_{22} \\ w_{31}^{21} \\ w_{31}^{22} \\ b_{33} \end{pmatrix}$$

→ **Numerical optimization to find best fit** →

Sine function fitting model

$$y = F(x, \mathbf{w})$$

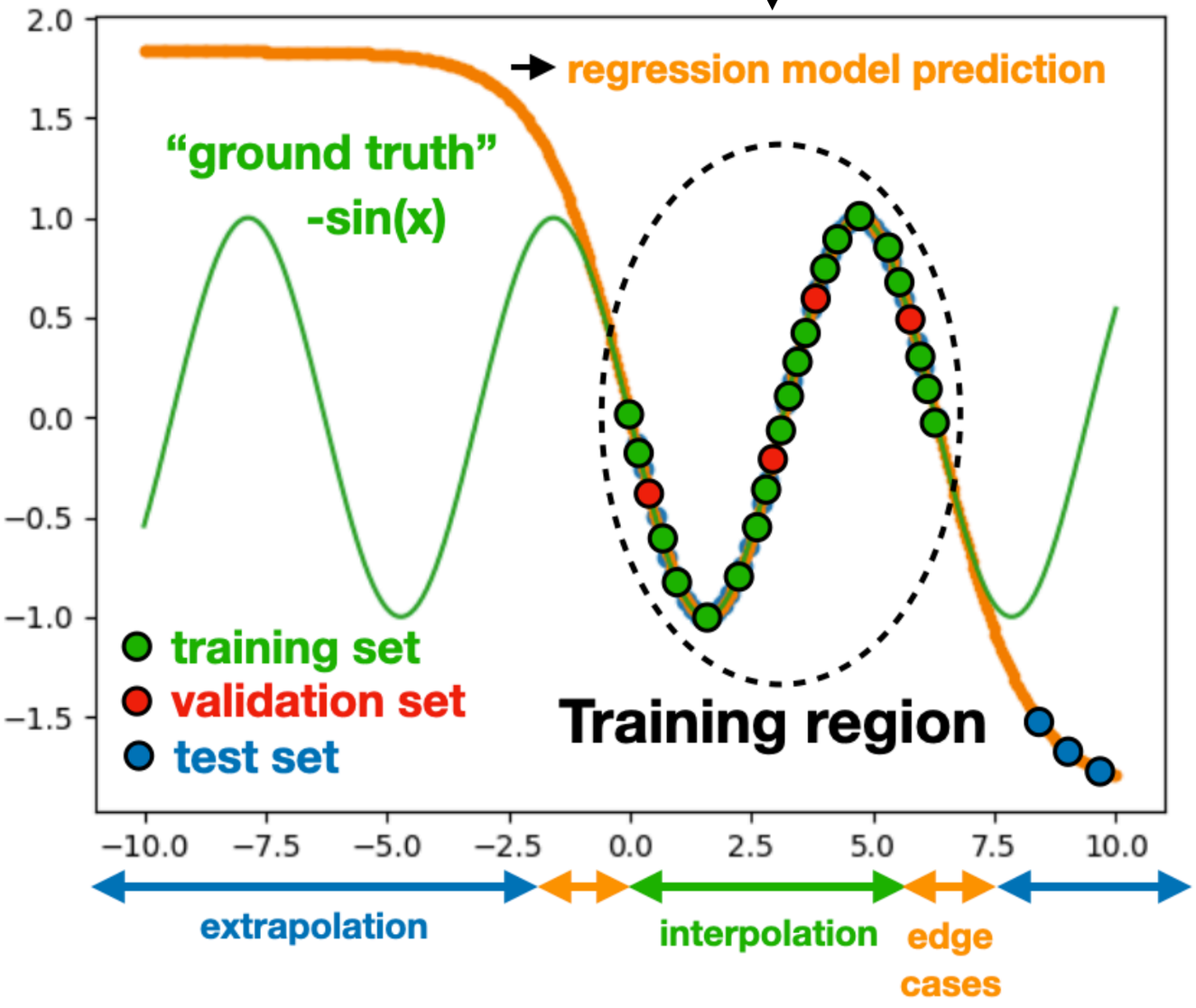
$$y(x) = A\sin(\omega(x - x_o)) + C$$

Fitting parameters $\mathbf{w} = (A, \omega, x_o, C)$

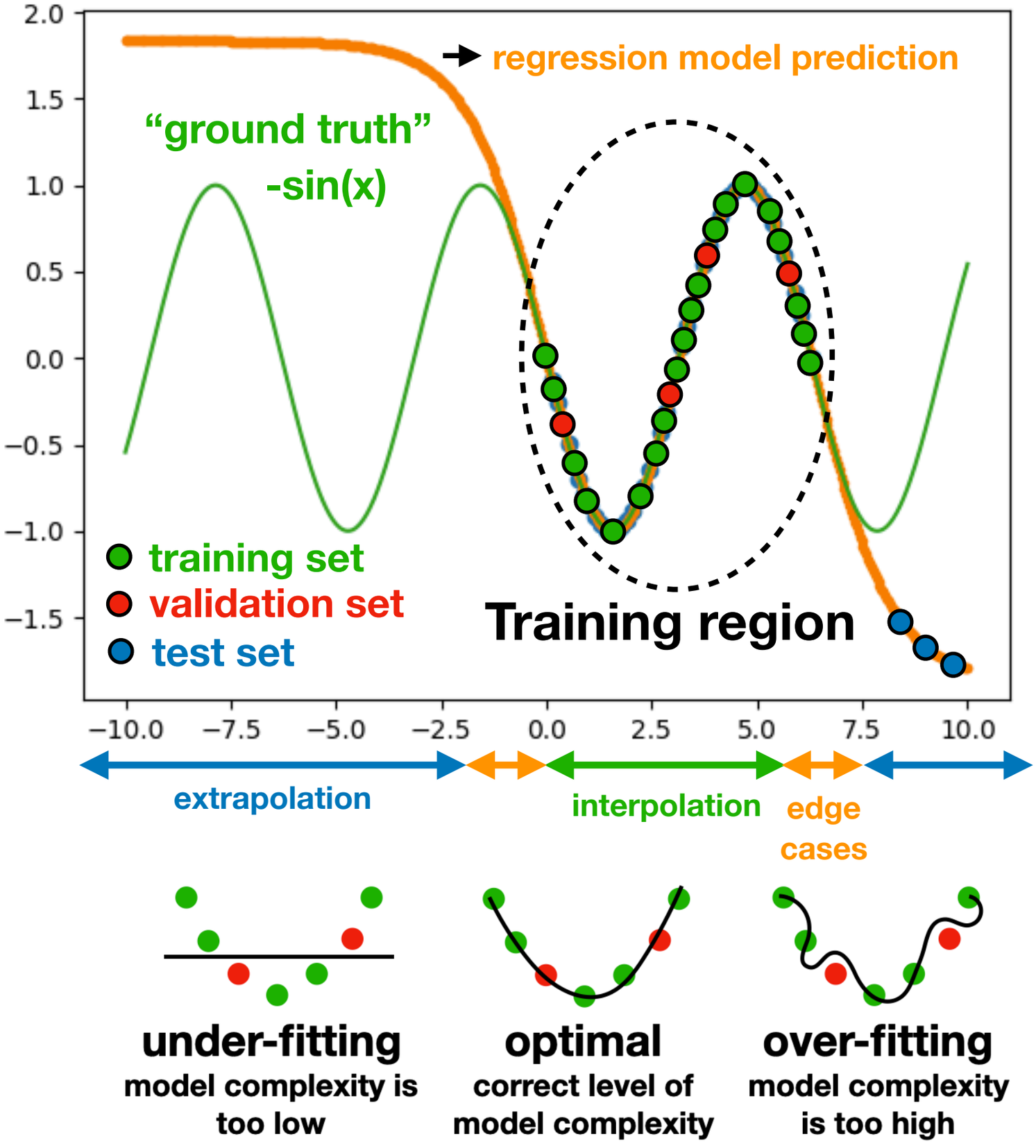
↓

Numerical optimization to find best fit

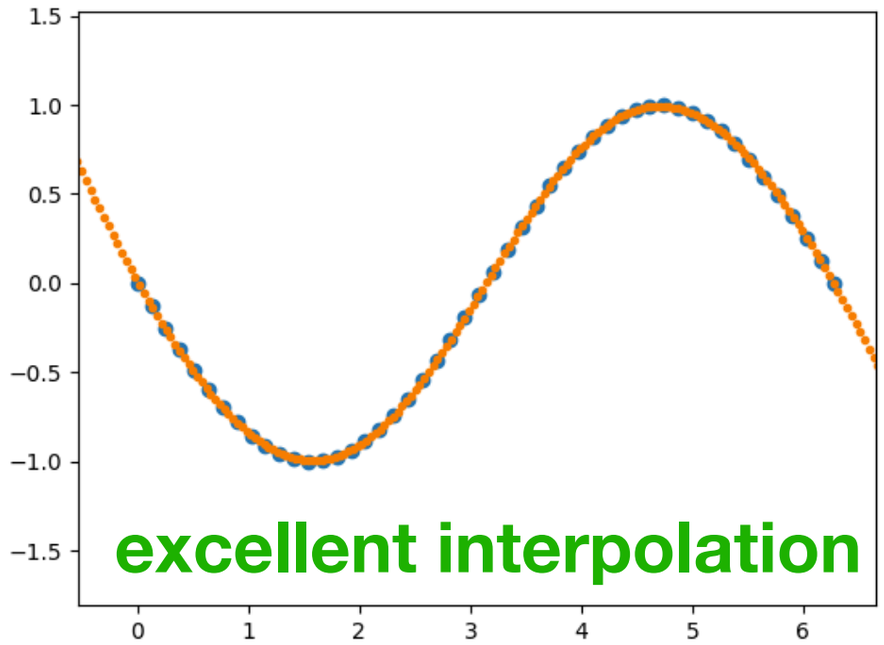
↓



Neural networks: General concerns



Train NN to reproduce provided data

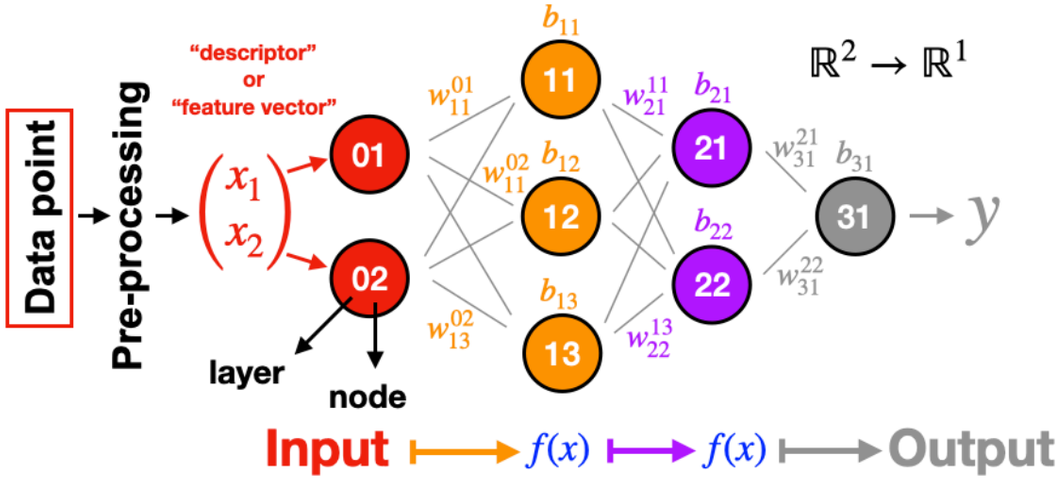


Validation data (~10-20% of the data)
-untrained but similar to training data
-tests interpolation
-used to monitor under/over fitting

Test data
-untrained but different from train and validation data
-tests extrapolation or “transferability”

Neural networks: Implementation-2

Batch implementation (multiple inputs —> multiple outputs)



Identical neural network!!

$$\begin{pmatrix} x_1 & x_2 \\ \tilde{x}_1 & \tilde{x}_2 \end{pmatrix} \longrightarrow \text{Input vector-1}, \quad \longrightarrow \text{Input vector-2}$$

Hidden layer-1:

Single input:

$$\begin{pmatrix} O_{11} \\ O_{12} \\ O_{13} \end{pmatrix} = f\left(\begin{pmatrix} w_{11}^{01} & w_{11}^{02} \\ w_{12}^{01} & w_{12}^{02} \\ w_{13}^{01} & w_{13}^{02} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_{11} \\ b_{12} \\ b_{13} \end{pmatrix} \right) \qquad f\left(\underline{\mathbf{w}}_{01}\mathbf{x} + \mathbf{b}_1\right)$$

Multiple inputs:

$$\begin{pmatrix} O_{11} & O_{12} & O_{13} \\ \tilde{O}_{11} & \tilde{O}_{12} & \tilde{O}_{13} \end{pmatrix} = f\left(\begin{pmatrix} x_1 & x_2 \\ \tilde{x}_1 & \tilde{x}_2 \end{pmatrix} \begin{pmatrix} w_{11}^{01} & w_{12}^{01} & w_{13}^{01} \\ w_{11}^{02}. & w_{12}^{02} & w_{13}^{02} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{11} & b_{12} & b_{13} \end{pmatrix} \right) \qquad f\left(\mathbf{x}\underline{\mathbf{w}}_{01}^{\mathsf{T}} + \mathbf{b}_1\right)$$

Single input:

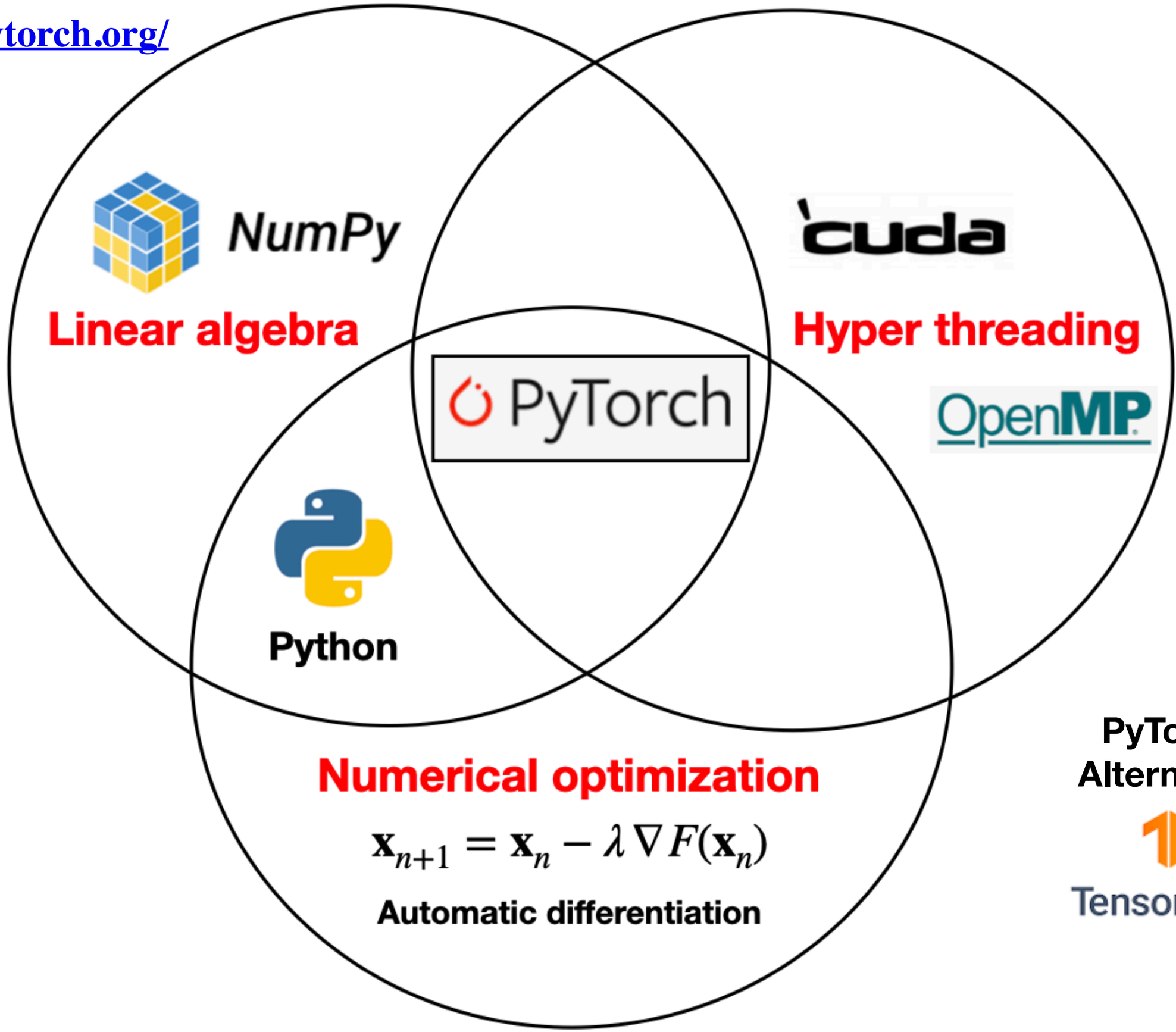
$$y = \underline{\mathbf{w}}_{23}\, f\left(\underline{\mathbf{w}_{12}}\, f\left(\underline{\mathbf{w}_{01}}\mathbf{x} + \mathbf{b}_1\right) + \mathbf{b}_2\right) + \mathbf{b}_3$$

multiple inputs:

$$\begin{pmatrix} y \\ \tilde{y} \end{pmatrix} = f\left(f\left(\mathbf{x}\underline{\mathbf{w}}_{01}^{\mathsf{T}} + \mathbf{b}_1\right)\mathbf{w}_{12}^{\mathsf{T}} + \mathbf{b}_2\right)\mathbf{w}_{23}^{\mathsf{T}} + \mathbf{b}_3$$

PyTorch Overview

https://pytorch.org/



PyTorch Alternative

TensorFlow

Automatic differentiation

Automatic differentiation

https://en.wikipedia.org/wiki/Automatic_differentiation

- **Track operations**
- **Form computational graph**
- **Differentiate via chain rule**
  - **Forward mode**
  - **Reverse mode**

AUTOGRAD PyTorch

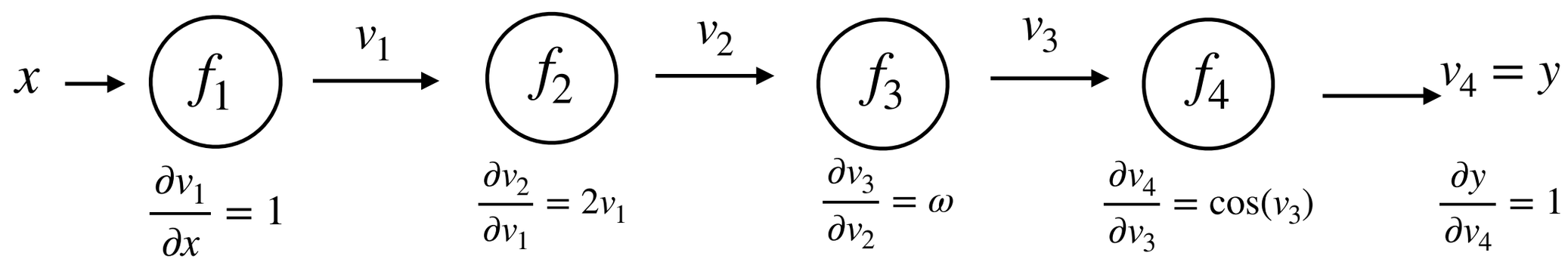
```
>>> x = torch.randn(5, 5)  # requires_grad=False by default
>>> y = torch.randn(5, 5)  # requires_grad=False by default
>>> z = torch.randn((5, 5), requires_grad=True)
>>> a = x + y
>>> a.requires_grad
False
>>> b = a + z
>>> b.requires_grad
True
```

https://pytorch.org/docs/stable/notes/autograd.html

Toy Example: inside $v_1 = f_1(x) = x$ $v_2 = f_2(v_1) = v_1^2$ $v_3 = f_3(v_2) = \omega v_2$ $v_4 = f_4(v_3) = \sin(v_3)$ out

Computational graph

$y = \sin(\omega x^2)$



Function evaluation

$$y = f_4(f_3(f_2(f_1(x))))$$

$$v_1 = f_1(x) = x$$
$$v_2 = f_2(v_1) = v_1^2$$
$$v_3 = f_3(v_2) = \omega v_2$$
$$v_4 = f_4(v_3) = \sin(v_3) = y$$

Chain rule

$$\frac{dy}{dx} = \frac{dy}{dv_4}\frac{dv_4}{dv_3}\frac{dv_3}{dv_2}\frac{dv_2}{dv_1}\frac{dv_1}{dx}$$

$$\frac{dy}{dx} = 2\omega x \cos(\omega x^2) \rightarrow y'(2)$$

evaluate numerically

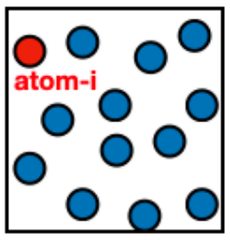
PYFIT-FF

Machine learning potentials: General

Model:

Structure (POSCAR)

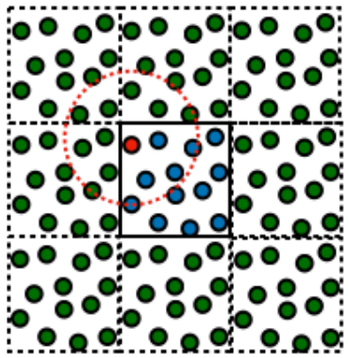
$U(\boldsymbol{r}_1, \boldsymbol{r}_2 \ldots \boldsymbol{r}_N) = ?$



N atoms

apply periodic boundary conditions

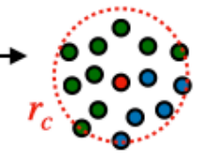
Periodic structure



periodic images

Local atomic environment

neighbor list for atom-i



$$X = \begin{pmatrix} x_1, y_1, z_1 \\ x_2, y_2, z_2 \\ \vdots \end{pmatrix}$$

relative positions of atom-i neighbors

Descriptor

local atomic "finger-print" for atom-i

$$\boldsymbol{G}_i = \boldsymbol{G}_i(X) = (G_1, G_2 \ldots G_m)$$

Regression model

fitting parameters

$$R(\boldsymbol{G}_i, \boldsymbol{w}) = e_i$$

energy of atom-i

Supercell energy

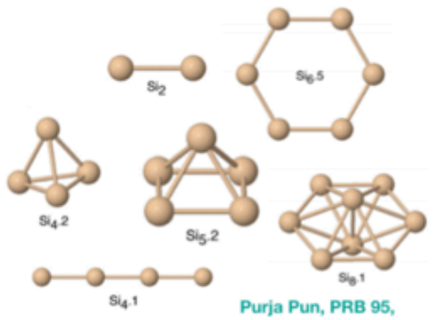
repeat for all N atoms

$$\hat{U} = \sum_i^N e_i$$

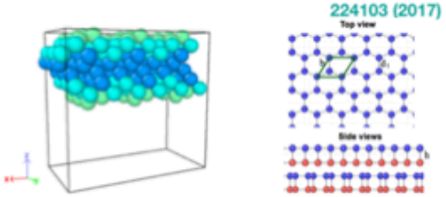
Training:

Large collection of structural prototypes

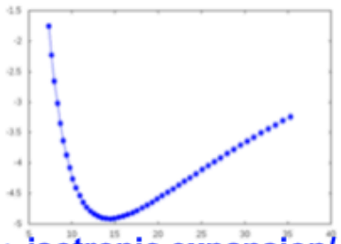
bulk (FCC, BCC, etc), defects, 2D, clusters, etc



Purja Pun, PRB 95, 224103 (2017)



structural prototype perturbations



- isotropic expansion/ compression
- Shearing
- anisotropic expansion/ compression
- frozen phonons
- thermal perturbations (AIMD)
- etc

Training

Training set	DFT energies	Model predictions
Structure-1	U_1	$\hat{U}_1$
Structure-2	U_2	$\hat{U}_2$
⋮		
Structure-M	U_M	$\hat{U}_M$

$$\hat{U}_j = \sum_i^{N_j} e_i = \sum_i^{N_j} R(\boldsymbol{G}_i, \boldsymbol{w})$$

RMSE error

$$\epsilon(\mathbf{w}) = \sum_j^M \left(\frac{\left(U_j - \hat{U}_j\right)^2}{M} \right)^{\frac{1}{2}}$$

Numerical optimization to find best fit

Potential types

Traditional potential model

Structure → physical model + parameterization → Energy

ANN potential model

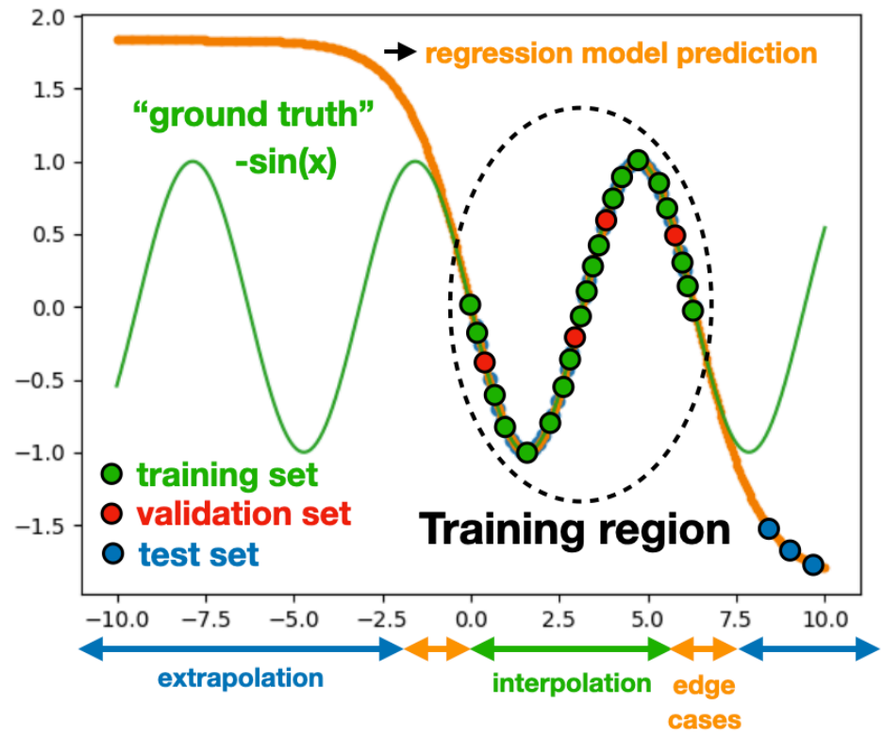
Structure → local atomic environment fingerprinting scheme → Neural network → Energy

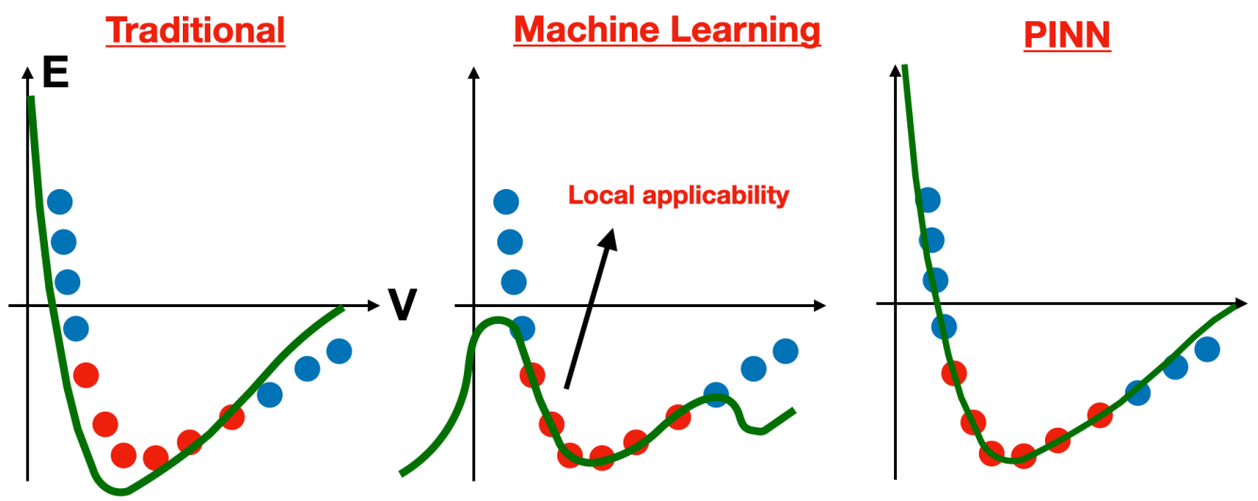
PINN potential model

Physically informed artificial neural networks for atomistic modeling of materials
GPP Pun, R Batra, R Ramprasad, Y Mishin - Nature communications, 2019

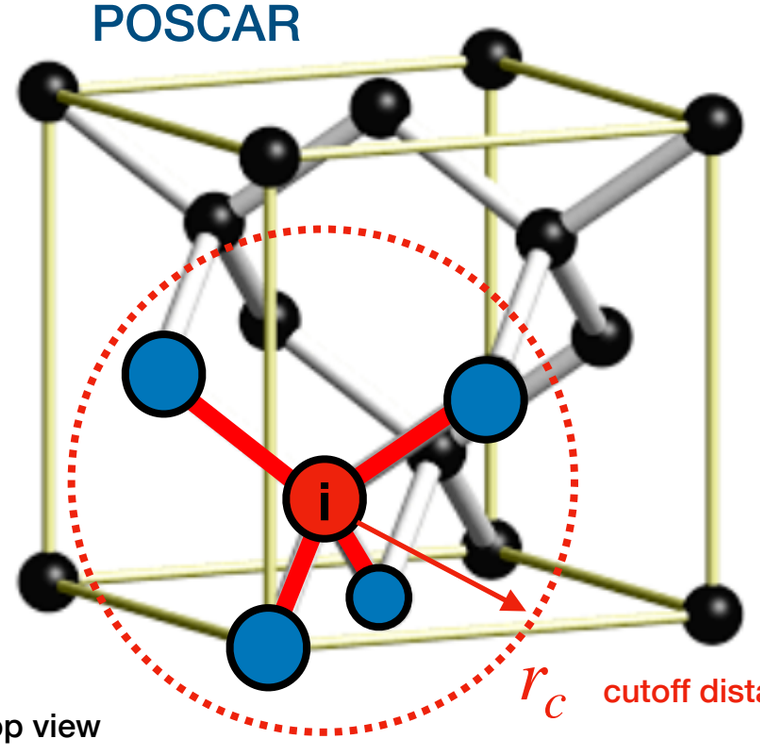
Structure → local atomic environment fingerprinting scheme → Neural → baseline physical model $A_o, B_o, \alpha_o, \beta_o, a_o, h_o, \lambda_o, \sigma_o$ + local model corrections $\boldsymbol{\delta} = \delta_A, \delta_B, \delta_\alpha, \delta_\beta, \delta_a, \delta_h, \delta_\lambda, \delta_\sigma$ → traditional potential → U

Analogy:

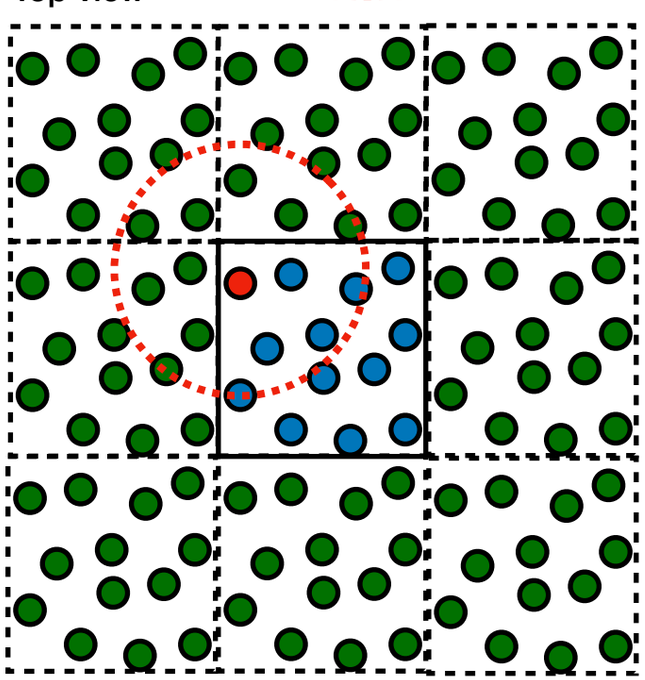


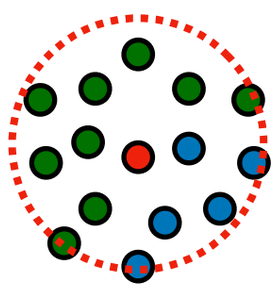


Atomic fingerprinting: Neighbor list

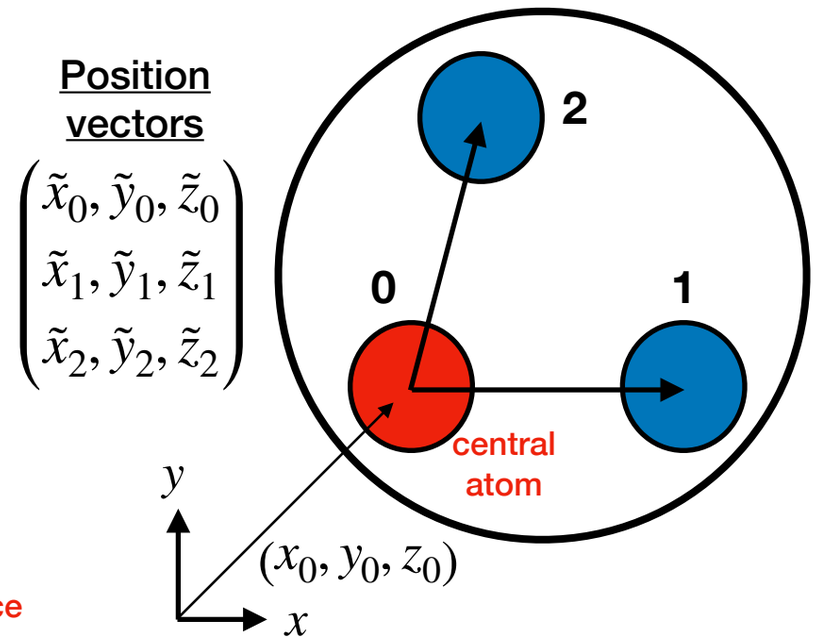


Top view





Simple Example: Single component trimer



Position vectors

$$\begin{pmatrix} \tilde{x}_0, \tilde{y}_0, \tilde{z}_0 \\ \tilde{x}_1, \tilde{y}_1, \tilde{z}_1 \\ \tilde{x}_2, \tilde{y}_2, \tilde{z}_2 \end{pmatrix}$$

Relative to central atom

$$\begin{pmatrix} \tilde{x}_1 - \tilde{x}_0, \tilde{y}_1 - \tilde{x}_0, \tilde{z}_1 - \tilde{x}_0 \\ \tilde{x}_2 - \tilde{x}_0, \tilde{y}_2 - \tilde{x}_0, \tilde{z}_2 - \tilde{x}_0 \end{pmatrix}$$

$$\downarrow$$

$$X = \begin{pmatrix} x_1, y_1, z_1 \\ x_2, y_2, z_2 \end{pmatrix}$$ Neighbor list

Pair terms: $$r_{ij} = (\sum_{rows} X^2)^{\frac{1}{2}} = \begin{pmatrix} r_{01} \\ r_{02} \end{pmatrix}$$

Three body terms: $$X_A = \begin{pmatrix} x_1, y_1, z_1 \\ x_1, y_1, z_1 \\ x_2, y_2, z_2 \\ x_2, y_2, z_2 \end{pmatrix} \quad X_B = \begin{pmatrix} x_1, y_1, z_1 \\ x_2, y_2, z_2 \\ x_1, y_1, z_1 \\ x_2, y_2, z_2 \end{pmatrix}$$

Replicate and tile the neighbor list

```
Xij=np.tile(nbl,n).reshape(n*n,3)
Xik=np.tile(nbl,(n,1)).reshape(n*n,3)
```

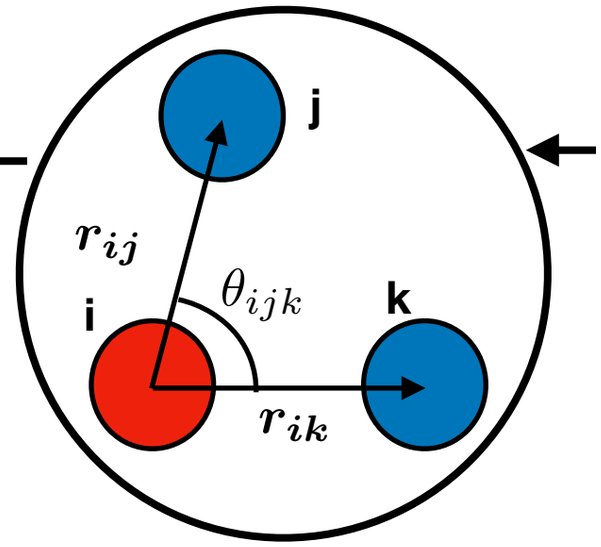
```
rij=(((Xij**2.0).sum(axis=1))**0.5).reshape(1,n*n); #sum accros row
rik=(((Xik**2.0).sum(axis=1))**0.5).reshape(1,n*n);
cos_ijk1=((Xij*Xik).sum(axis=1).reshape(1,n*n))/rij/rik;
```

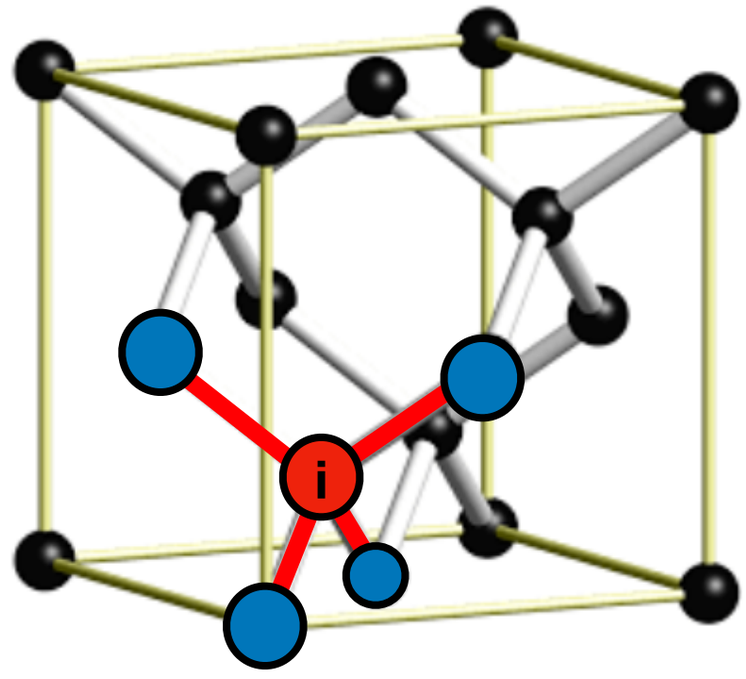
Atomic fingerprinting: LSP Calc

Physically informed artificial neural networks for atomistic modeling of materials GPP Pun, R Batra, R Ramprasad, Y Mishin - Nature communications, 2019

Structure parameters:

$$G_i^{m,n} = \sum_{j,k} P_m(cos(\theta_{ijk}))f(r_{ij})f(r_{ik})$$





Angular term:

$$P_m(cos(\theta)) \qquad m = 0,1,2,4,6$$

(Legendre polynomials)

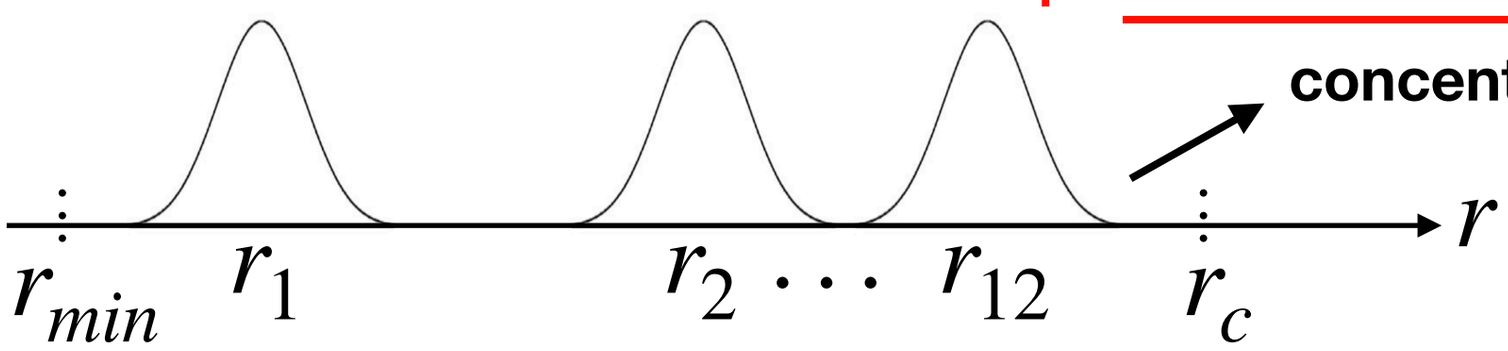
Radial term:

$$f(r) = \frac{1}{r_n} e^{-\frac{(r-r_n)^2}{\sigma^2}} f_c(r)$$

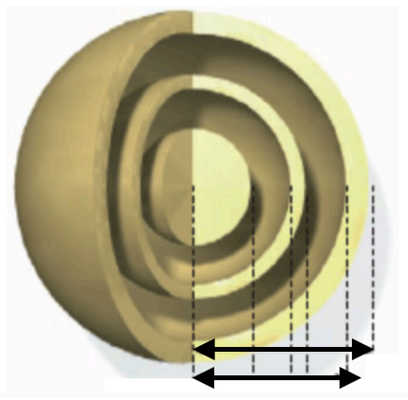
$f_c(r)$ = Cutoff function

choose 12 values for r_n

Gi's act as "fingerprints" of the local atomic environment



concentric shells



$$f_c(r) = \begin{cases} \frac{(r-r_c)^4}{d^4+(r-r_c)^4} & r \le r_c \\ 0, & r \ge r_c \end{cases}$$

PyFit Functionality

★ **Current functionality**

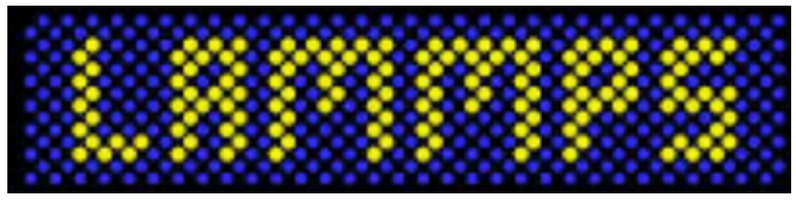
- Single component mathematical ANN
  - local atomic environment descriptors developed by Purja-Pun and Mishin

- **Beta version**
  - Single component PINN training
    - local atomic environment descriptors developed by Purja-Pun and Mishin

- **Future plans**
  - Add Behler-Parrinello LSP for the case of single component ANN
  - Multi-component ANN training
  - Multi-component PINN training

- **Implementation** → ParaGrandMC: PGMC
  Parallel Grand Canonical Monte Carlo Simulation Code
  https://software.nasa.gov/software/LAR-18773-1
  V. Yamakov

↓ (in progress)



G. Purja Pun, J. Chapman

Workflow

Pre-training

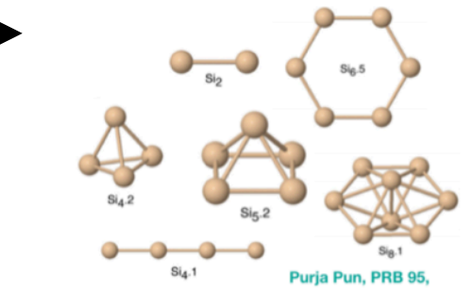
DFT Data Generation

- generated using VASP before training
- K-Point and ENCUT convergence
- Literature comparison
  - properties+values

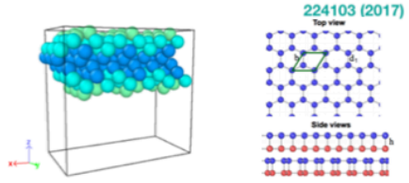
Structural sampling

Large collection of structural prototypes

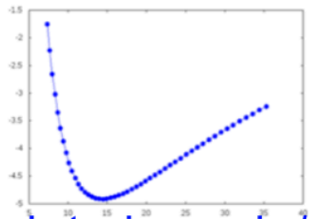
bulk (FCC, BCC, etc), defects, 2D, clusters, etc



Purja Pun, PRB 95, 224103 (2017)



structural prototype perturbations



- isotropic expansion/ compression
- Shearing
- anisotropic expansion/ compression
- frozen phonons
- thermal perturbations (AIMD)
- etc

DFT

Data set

Structure-1	U_1
Structure-2	U_2
⋮	
Structure-M	U_M

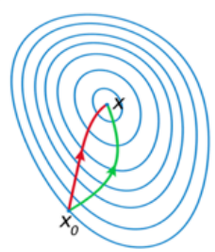
PyFit

Pre-processing

- atomic environment fingerprinting
- error/consistency checks
- matrix construction

Training loop

L-**BFGS** (quasi-Newton method)



Training set	DFT energies	Model predictions
Structure-1	U_1	$\hat{U}_1$
Structure-2	U_2	$\hat{U}_2$
⋮		
Structure-M	U_M	$\hat{U}_M$

PyTorch

RMSE error $\epsilon(\mathbf{w}) = \sum_j^M \left(\frac{\left(U_j - \hat{U}_j\right)^2}{M} \right)^{\frac{1}{2}}$ → Minimize

stopping criterion

Interatomic potential

DEMO

Dependency Installation

Operating systems


✓


✓

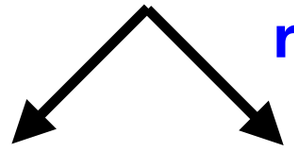

?

Required dependencies

Python 3.x

NumPy

recommended



Manual install

sudo pip3 install torch torchvision numpy

CONDA®

README.md

1) Install conda: https://docs.conda.io/projects/conda/en/latest/user-guide/install/

2) Create conda environment and install dependencies

```
conda deactivate                          # exit current conda enviroment if one is activated
conda create -n TORCH3.7 python=3.7       # create new conda enviroment named TORCH3.7
conda activate TORCH3.7                   # activate the TORCH3.7 enviroment
conda install -c pytorch pytorch          # install the pytorch in the TORCH3.7 enviroment
conda update --all && conda clean -all    # update and clean
```

3) Clone PyFit demo from GitHub

```
git clone https://github.com/jfh3/PYFIT-FF
```

OR https://github.com/jfh3/PYFIT-FF

NIST

Contents

https://github.com/jfh3/PYFIT-FF

GitHub

PYFIT-FF

User manual

PYFIT-Manual.pdf

doc

examples

ANN → Si ANN potential fitting example

TORCH-EXAMPLES → various instructive PyTorch examples

Source code

src

data.py — dataset class

defaults.json — Default parameters

neural.py — neural network class

pyfit.py — Main script

reader.py — File reader subroutines

util.py — Miscellaneous subroutines

writer.py — File writer subroutines

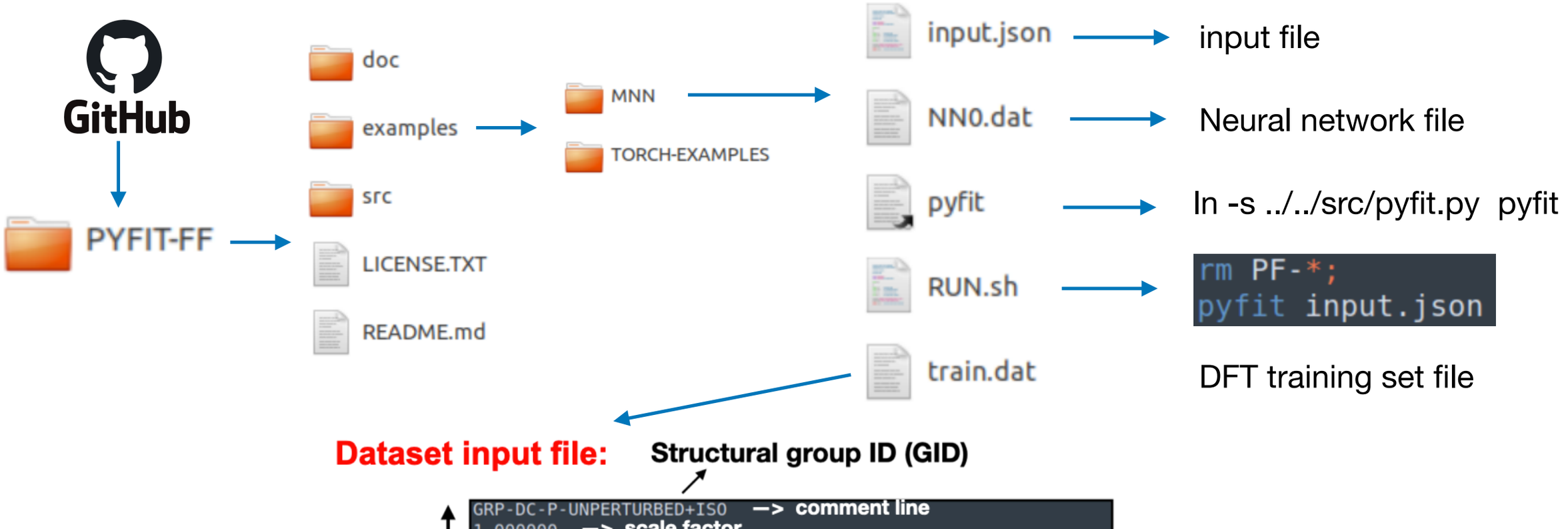
NIST license

LICENSE.TXT

README.md

download instructions

Example directory



Neural network input file

Potential type flag: ANN=5

optional LSP shift (default=0)

activation function flag

$0 \rightarrow s(x) = \dfrac{1}{1+e^{-x}}$

$1 \rightarrow \left(\dfrac{1}{1+e^{-x}}\right) - 0.5$

$$f_c(r) = \begin{cases} \dfrac{(r-r_c)^4}{d^4+(r-r_c)^4} & r \le r_c \\ 0, & r \ge r_c \end{cases}$$

cut-off parameters

i_{rand} W_{max} r_c d σ ← 5 0.0 1

1 → Number of elements

Si 28.0855 → Chemical symbol and weight

0 0.250000 4.500000 1.000000 2.000000

2 0 1 → N_{LG} O_1 $O_2 \ldots$

N_{r_o} r_1 $r_2 \ldots$ ← 2 2.0 3.0

0 0 1 1 1 1 1 1 1 1 → Reference parameters

N_{layer}, N_{input}, H_1 H_2 N_{output} ← 3 2 1

w{11}—>{21}

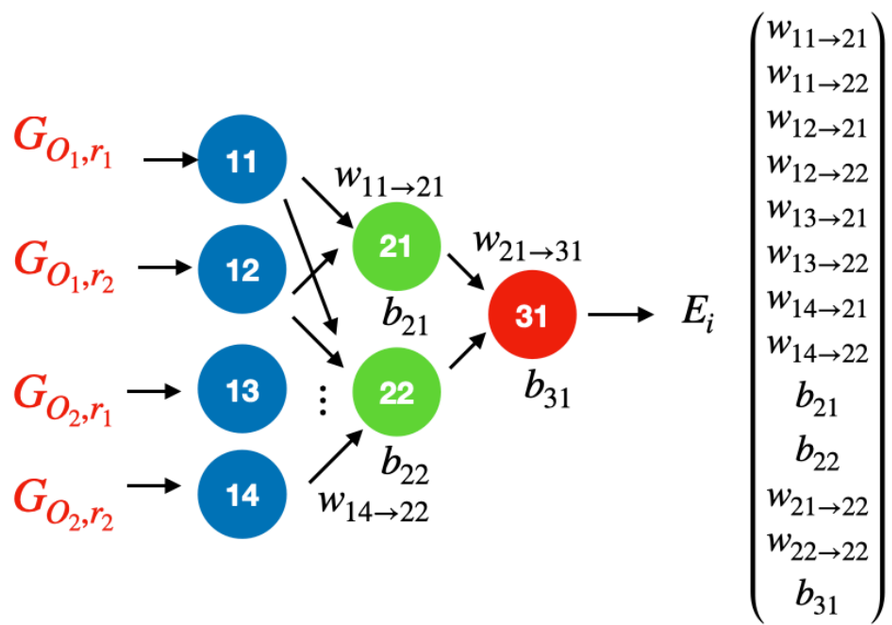
w{11}—>{22}

w{12}—>{21}

⋮

list of NN weights and bias terms

Neural network ordering



Input file

PYFIT-FF/src/defaults.json

```
{
"train_RMSE"            :       true,
"rmse_tol"              :       0.00001,
"dump_poscars"          :       false,
"u_shift"               :       0.795023,
"ramp_LR"               :       true,
"mid_ramp"              :       10,
"LR_o"                  :       0.01,
"LR_f"                  :       0.10,
"lambda_Lp"             :       0.00001,
"LP"                    :       2.0,
"constrain_WB"          :       0.0,
"test_set_tags"         :       ["GRP-BCC-P-UNPERTURBED+ISO"],
"n_rand_GIDS"           :       0,
"fix_rand_seed"         :       false,
"exclude_from_test"     :       ["DC","ISOLATED","LIQ"],
"save_every1"           :       10,
"save_every2"           :       100,
"lbfgs_max_iter"        :       20,
"train_edges"           :       true,
"pot_type"              :       "NN",
"pot_file"              :       "NN1.dat",
"dataset_path"          :       "train.dat",
"max_iter"              :       100000,
"rmse_dU"               :       0.0,
"rmse_stop"             :       0.002,
"fraction_train"        :       0.8,
"re_randomize"          :       true,
"weight_selector"       :       ["DC","AIMD","Wurt"],
"default_weight1"       :       1.0,
"default_weight2"       :       0.0,
"mod_weight1"           :       1.0,
"mod_weight2"           :       0.0,
"use_cuda"              :       false,
"dynamic_NN"            :       false,
"start_fresh"           :       false,
"try_n_times"           :       1,
"lambda_E1"             :       1.0,
"lambda_dU"             :       1.0,
"lambda_L1"             :       0.0,
"cnst_final_bias"       :       true,
"final_bias"            :       0.0,
"write_lsp"             :       true,
"normalize_gi"          :       false,
"normalize_by_ro"       :       false,
"normalize_ei"          :       false
}
```

PYFIT-FF/examples/MMN/input.json

```
{
"pot_type"              :       "NN",
"pot_file"              :       "NN0.dat",
"dataset_path"          :       "train.dat",

"u_shift"               :       0.795023,

"test_set_tags"         :       ["GRP-BCC-P-UNPERTURBED+ISO"],
"fraction_train"        :       0.8,

"dump_poscars"          :       false,

"ramp_LR"               :       true,
"mid_ramp"              :       10,
"LR_o"                  :       0.01,
"LR_f"                  :       0.10,

"lambda_Lp"             :       0.00001,
"LP"                    :       2.0,

"train_RMSE"            :       true,
"rmse_tol"              :       0.00001,
"max_iter"              :       100000,
"rmse_stop"             :       0.002,
"save_every1"           :       10,
"save_every2"           :       100,

"weight_selector"       :       ["DC","AIMD","Wurt"],
"default_weight1"       :       2.0,

"train_edges"           :       true,

"use_cuda"              :       false,

"pot_type"              :       "NN",
"pot_file"              :       "NN1.dat",
"dataset_path"          :       "train.dat",
"rmse_dU"               :       0.0,
"re_randomize"          :       true
}
```

Output files

```
(MBP3.7) james@james-VirtualBox:~/HOME/SRC/PY-FIT/PYFIT-FF/examples/MNN$ ls
input.json      PF-e_vs_V-no_dft-0.dat    PF-e_vs_V-test-100.dat   PF-e_vs_V-validate-0.dat    PF-LSP.dat     PF-stats-test.dat      pyfit
NN0.dat         PF-e_vs_V-no_dft-100.dat  PF-e_vs_V-train-0.dat    PF-e_vs_V-validate-100.dat  PF-NN-0.dat    PF-stats-train.dat     RUN.sh
PF-err-log.dat  PF-e_vs_V-test-0.dat      PF-e_vs_V-train-100.dat  PF-log.dat                  PF-NN-100.dat  PF-stats-validate.dat  train.dat
```

- e_vs_v **files:**
  - As the name suggests these files contain volume vs energy data for the various structures. Each dataset, i.e. training, validation, and test, will get its own file. The naming convention is , for example 00-e_vs_v-test-1000.dat. The first column is the volume per atom in the structure (V/N), the second column is the energy per atom DFT energy (E_{DFT}/N).
- **err file:**
  - This file is a simple columnar text file which reports various terms of the objective function as function of the iteration step. This file only reports metrics associated with the training set, metrics associated with other data sets are reported in the stat files described below. The frequency at which the values are written is controlled by the save_every parameter in the input.json file, see sec.3.4.2. If a particular term in the objective function is not being used then it will be reported as a zero in the file.
- **stat files:**
  - These files describe the statistical state of the training process and contain various error metrics of the various data sets; training, validation, test.
- **NN files:**
  - The code periodically writes the current state of the neural network to a "NN" file using the naming convention "PREFIX-NN-STEP.dat" where STEP refers the training iteration,

Run Example