Machine learning methods in atomistic simulations: Basics and predicting the electronic structure

CNPEM/Ilum school

Mariana Rossi, MPI for the Structure and Dynamics of Matter, Hamburg

How can ML be used in Atomistic Simulations?

- Method for data analysis that automates model building and accelerates simulations
- Method to learn from data and identify unknown patterns
  (can help interpretation)
- Method to aid “decision making” without human intervention
- Method to generate new (never seen) materials with desired properties
- **Needs data: from experiment or theory**

How can ML be used in Atomistic Simulations?

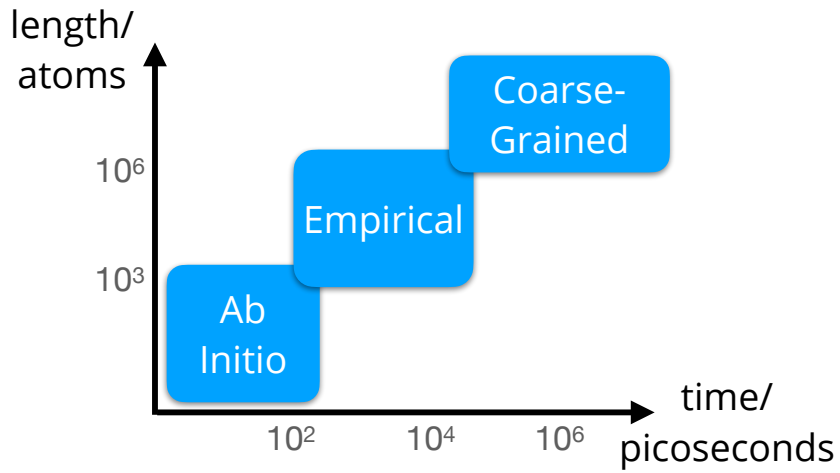
- Method for data analysis that automates model building and accelerates simulations
- Method to learn from data and identify unknown patterns
  (can help interpretation)
- Method to aid “decision making” without human intervention
- Method to generate new (never seen) materials with desired properties
- **Needs data: from experiment or theory**

The AI “Revolution” in Ab Initio Modeling

- Standard methods such as density-functional theory can:
  - Treat systems of up to 1000 atoms (normally)
  - Nuclear dynamics up to 100 picoseconds (normally)

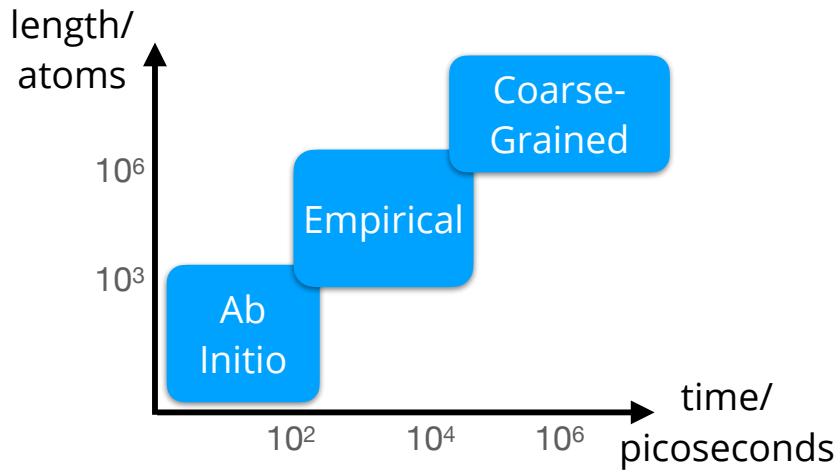
The AI “Revolution” in Ab Initio Modeling

- Standard methods such as density-functional theory can:
  - Treat systems of up to 1000 atoms (normally)
  - Nuclear dynamics up to 100 picoseconds (normally)



The AI “Revolution” in Ab Initio Modeling

- Standard methods such as density-functional theory can:
  - Treat systems of up to 1000 atoms (normally)
  - Nuclear dynamics up to 100 picoseconds (normally)



- Machine learning methods have brought “ab initio quality” calculations to 3 orders of magnitude larger in length and time scales (in the last ~7 years!)

Types of Machine Learning

Supervised

dataset

inputs/ features

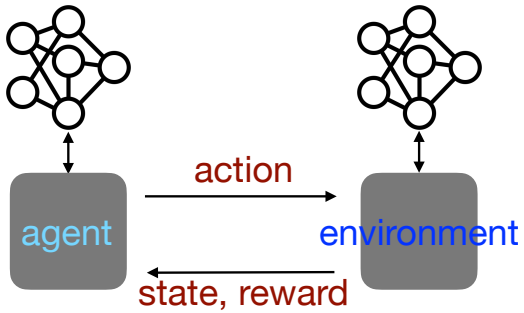
outputs/ labels

$$\mathscr{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$$

Unsupervised

$$\mathscr{D} = \{\mathbf{x}_i\}_{i=1}^{N}$$

Reinforcement



Semi-supervised, …

Basic concepts

Parametric and non-parametric models

- Parametric model:
    - Number of parameters fixed during training
    - Example: Neural network (fixed numbers of layers and nodes)
- Non-parametric model: Number of parameters increases with training data
    - More flexible, potential for better performance
    - “Curse of dimensionality”: need tricks to curb it
    - Example: Gaussian process

Supervised Learning: Regression

Concrete applications

- Regression outputs are continuous quantities that relate inputs to outputs

- In atomistic modeling:

$$\mathscr{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$$

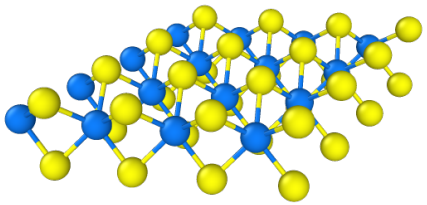
Supervised Learning: Regression

Concrete applications

- Regression outputs are continuous quantities that relate inputs to outputs

- In atomistic modeling:

$$\mathscr{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$$



Some representation of atomic structure

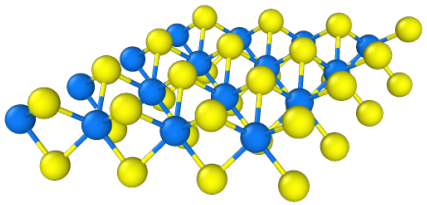
Supervised Learning: Regression

Concrete applications

- Regression outputs are continuous quantities that relate inputs to outputs

- In atomistic modeling:

$$\mathscr{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$$



Some representation of atomic structure

Property

- Total energies and forces
- Band structure
- Excitation spectrum
- Polarization
- Dielectric Responses
- Electronic density

…

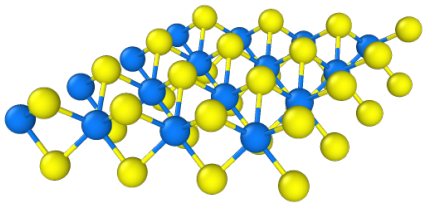
Supervised Learning: Regression

Concrete applications

- Regression outputs are continuous quantities that relate inputs to outputs

- In atomistic modeling:

$$\mathscr{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$$



Some representation of atomic structure

ML Regression Model

Estimates $\tilde{y}(\mathbf{x})$

Property

- Total energies and forces
- Band structure
- Excitation spectrum
- Polarization
- Dielectric Responses
- Electronic density

…

Supervised Learning: Regression

- There are many possible ML models to do regression
- Two of the most employed ones (in atomistic simulations) are:

Gaussian Process Regression (GPR)

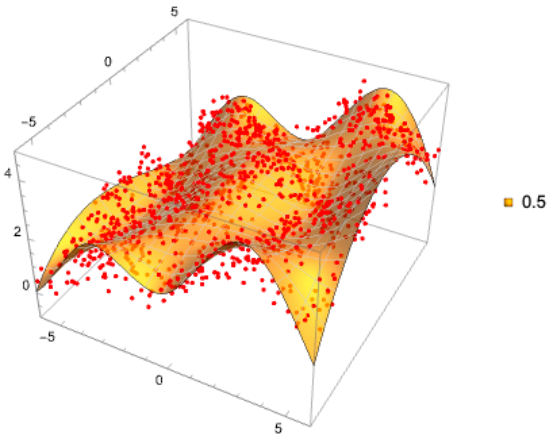
- Non-parametric model
- Types:
  - Standard
  - Symmetry-adapted (equivariant)

Artificial Neural Networks (NN)

- Parametric model
- Types:
  - “High-Dimensional" NNs
  - Graph (Message-Passing) NNs
  - Convolutional NNs
  - Equivariant or not…

Motivation and Goals

- Unknown analytical form connecting a descriptor and a property
- Simple interpolation schemes cannot deal with too many variables (high dimensions)
- Find a relationship between descriptor and property by minimising a suitable loss-function with a suitable representation of the data



GPR: weight-space view

C. E. Rasmussen & C. K. I. Williams, Gaussian Processes for Machine Learning
V. Deringer, et al., Chem Rev 121, 10073 (2021)

$$y(\mathbf{x}) \approx \tilde{y}(\mathbf{x}) = \sum_{m=1}^{M} c_m k(\mathbf{x}, \mathbf{x}_m) \qquad M \leq N$$

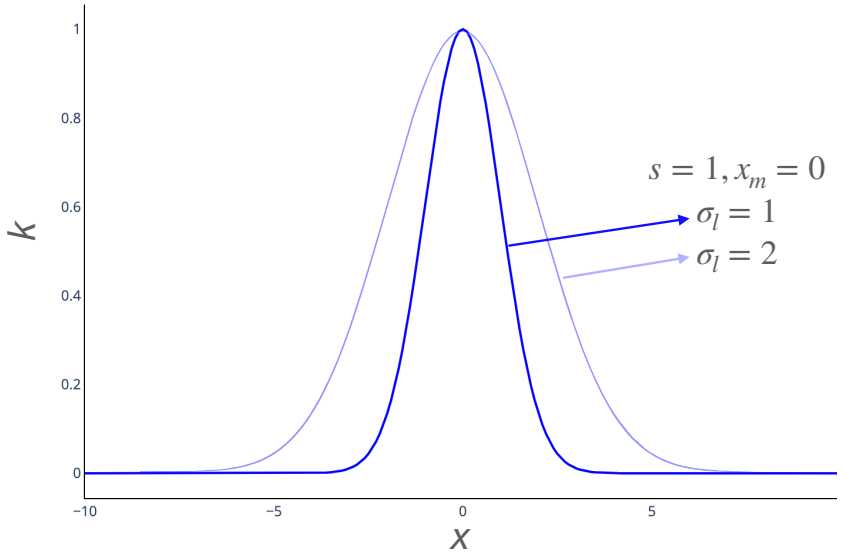
$$k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$$

- Properties:
  - $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$
  - k is positive-semidefinite (corresponding matrix has eigenvalues greater than or equal to zero)

GPR: possible kernels

- Many such kernels exist (periodic, linear, polynomial, rational, …)
- Common choice — “squared exponential”, “radial basis function (RBF)”, (a.k.a. Gaussian kernel)

$$k(\mathbf{x}, \mathbf{x}_m) = s^2 \exp\left(-\frac{|\mathbf{x} - \mathbf{x}_m|^2}{2\sigma_l^2}\right)$$



GPR: weight-space view

- Loss function is central to the fitting procedure - need to minimise it

$$\mathscr{L} = \sum_{n=1}^{N} [y_n - \tilde{y}(\mathbf{x}_n)]^2 + \eta R \qquad \tilde{y}(\mathbf{x}) = \sum_{m=1}^{M} c_m k(\mathbf{x}, \mathbf{x}_m)$$

- Common form for regularisation: Tikhonov regularisation

$$R = \sum_{m,m'} c_m k(\mathbf{x}_m, \mathbf{x}_{m'}) c_{m'}$$

GPR: weight-space view

- In matrix notation ("sparse GPR")

$$\mathscr{L} = (\mathbf{y}_N - \mathbf{K}_{NM}\mathbf{c}_M)^T(\mathbf{y}_N - \mathbf{K}_{NM}\mathbf{c}_M) + \eta\mathbf{c}_M^T\mathbf{K}_{MM}\mathbf{c}_M$$

- The analytical solution to the minimisation $\nabla_{\mathbf{c}^T}\mathscr{L} = 0$ is

$$\mathbf{c}_M = (\mathbf{K}_{NM}^T\mathbf{K}_{NM} + \eta\mathbf{K}_{MM})^{-1}\mathbf{K}_{NM}^T\mathbf{y}_N$$

- If $N = M$

$$\mathbf{c}_M = (\mathbf{K}_{NN} + \eta\mathbf{I}_N)^{-1}\mathbf{y}_N$$

$$\mathbf{y}(\mathbf{x}_*) = \mathbf{K}_{*N}(\mathbf{K}_{NN} + \eta\mathbf{I})^{-1}\mathbf{y}_N$$

GPR: function-space view

C. E. Rasmussen & C. K. I. Williams, Gaussian Processes for Machine Learning
V. Deringer, et al., Chem Rev 121, 10073 (2021)

- A Gaussian process (collection of random variables with joint Gaussian distributions) is defined by its mean function and covariance function

$$m(\mathbf{x}) = \mathbb{E}[\tilde{y}(\mathbf{x})]$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(\tilde{y}(\mathbf{x}) - m(\mathbf{x}))(\tilde{y}(\mathbf{x}') - m(\mathbf{x}'))]$$

$$\tilde{y} \sim GP(m, k)$$

- Random variables here are the values of function $\tilde{y}(\mathbf{x})$ at location $\mathbf{x}$

Predictions from GPR

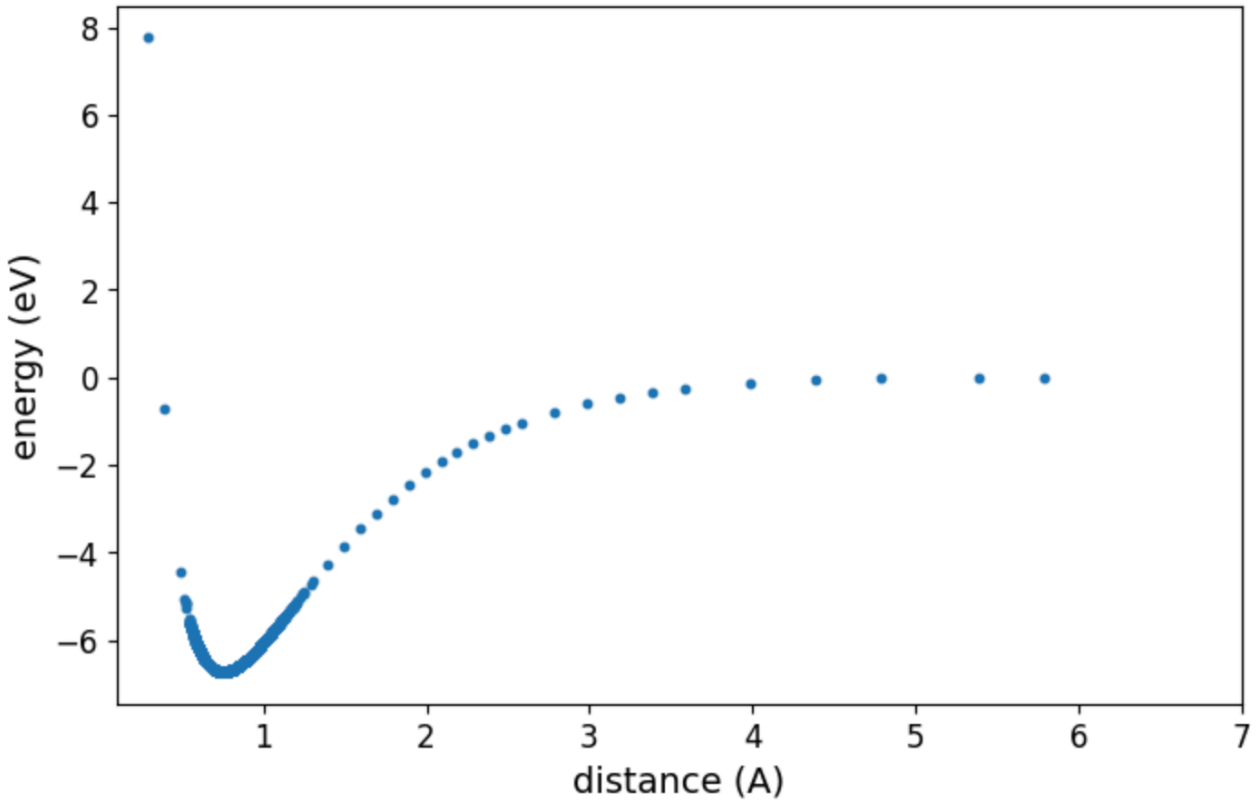
- Take a Bayesian linear regression model

$$y(\mathbf{x}) \approx \tilde{y}(\mathbf{x}) = \sum_{h=1}^{H} w_h \phi_h(\mathbf{x}) \qquad k(\mathbf{x}, \mathbf{x}') = \sigma_w^2 \sum_h \phi_h(\mathbf{x}) \phi_h(\mathbf{x}')$$

$$p(\mathbf{y}_* \,|\, \mathbf{y}_N) \sim \mathcal{N}(\mathbf{K}_{*N}(\mathbf{K}_{NN} + \sigma_\epsilon^2 \mathbf{I})^{-1} \mathbf{y}_N, \mathbf{K}_{**} - \mathbf{K}_{*N}^T (\mathbf{K}_{NN} + \sigma_\epsilon^2 \mathbf{I})^{-1} \mathbf{K}_{*N})$$

- Gives expression for variance at all points
- GP prediction does not depend on ϕ, instead only on the kernels.
- This is called the “kernel trick” in the weight-space view
- RKHS can always be recovered from the kernel definition

GPR with a worked example



Hyperparameter Optimisation

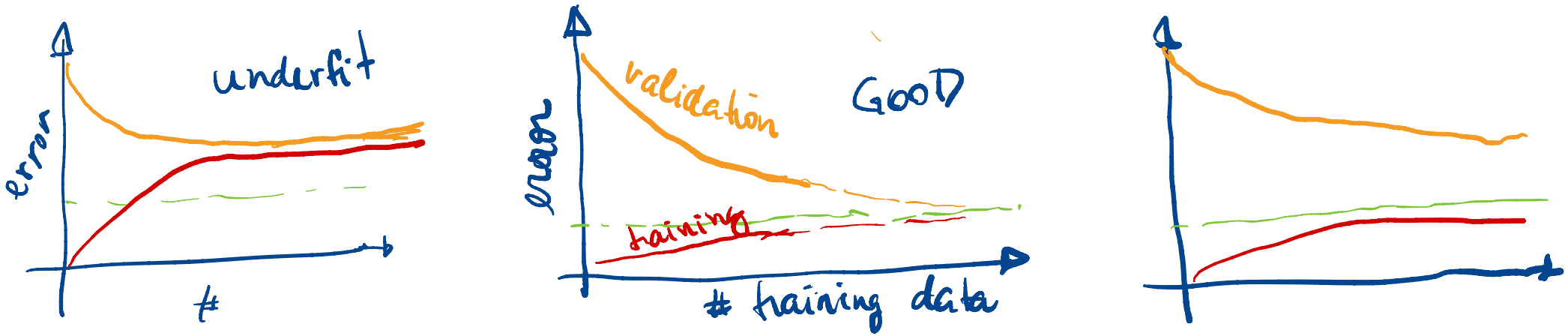
- Main hyperparameters are σ_l (squared-exponential kernel length) and η (strength of regularisation / noise in the data)
- Optimisation through maximisation of the logarithm of the marginal likelihood (evidence)

$$p(\mathbf{y}_N \,|\, \{\mathbf{x}_n\}, \sigma_l, \eta) \propto \exp\left[-\mathbf{y}^T(\mathbf{K}_{NN}(\sigma_l) + \eta\mathbf{I})^{-1}\mathbf{y}/2\right]$$

$$\log p = -\frac{1}{2}\mathbf{y}^T(\mathbf{K}_{NN}(\sigma_l) + \eta\mathbf{I})^{-1}\mathbf{y} - \frac{1}{2}\log\det(\mathbf{K}_{NN}(\sigma_l) + \eta\mathbf{I}) - \frac{N}{2}\log 2\pi$$

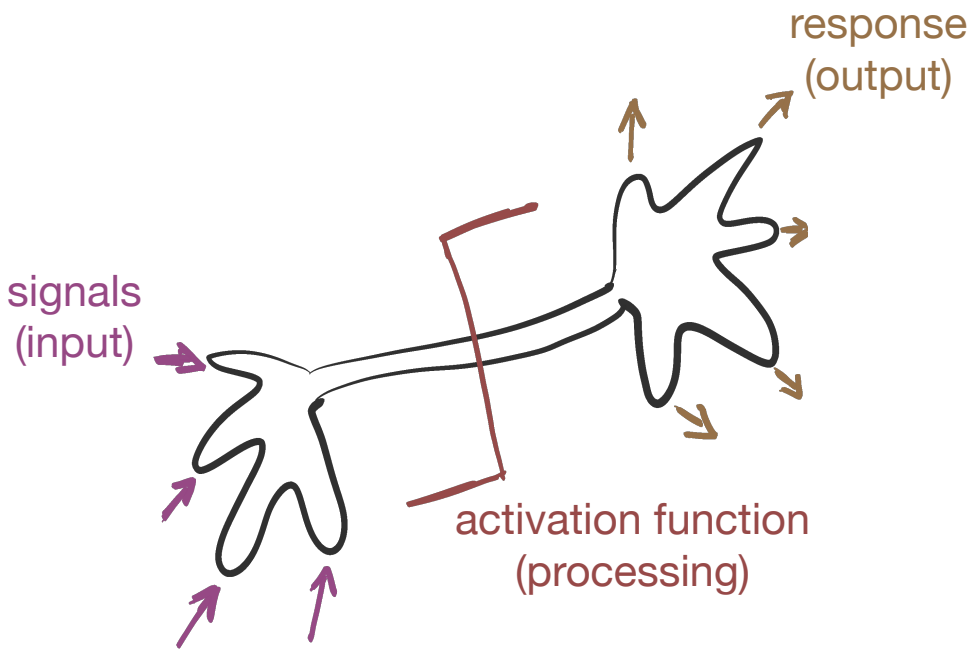
- *Often more practical*: optimise by cross-validation → divide known data into training and validation and vary hyperparameters to achieve lowest error on validation set

GPR learning curves



Artificial Neural Networks

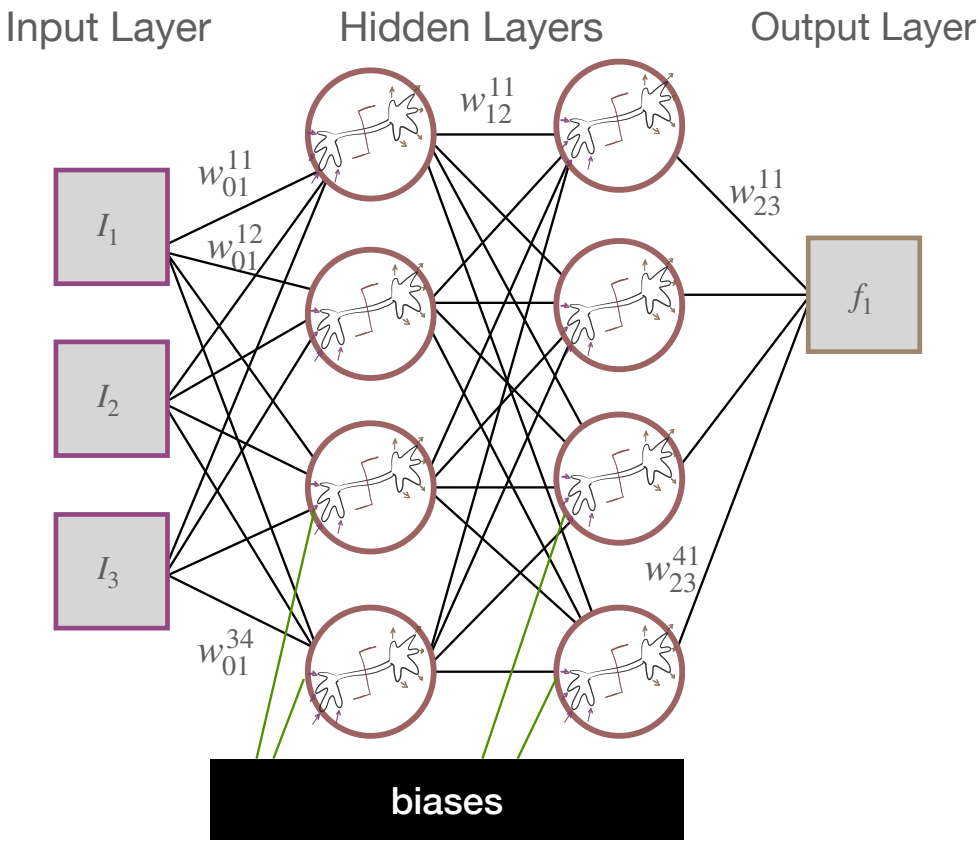
- Inspiration from biological processes



- Brain: Billions of neurons performing non-linear signal processing

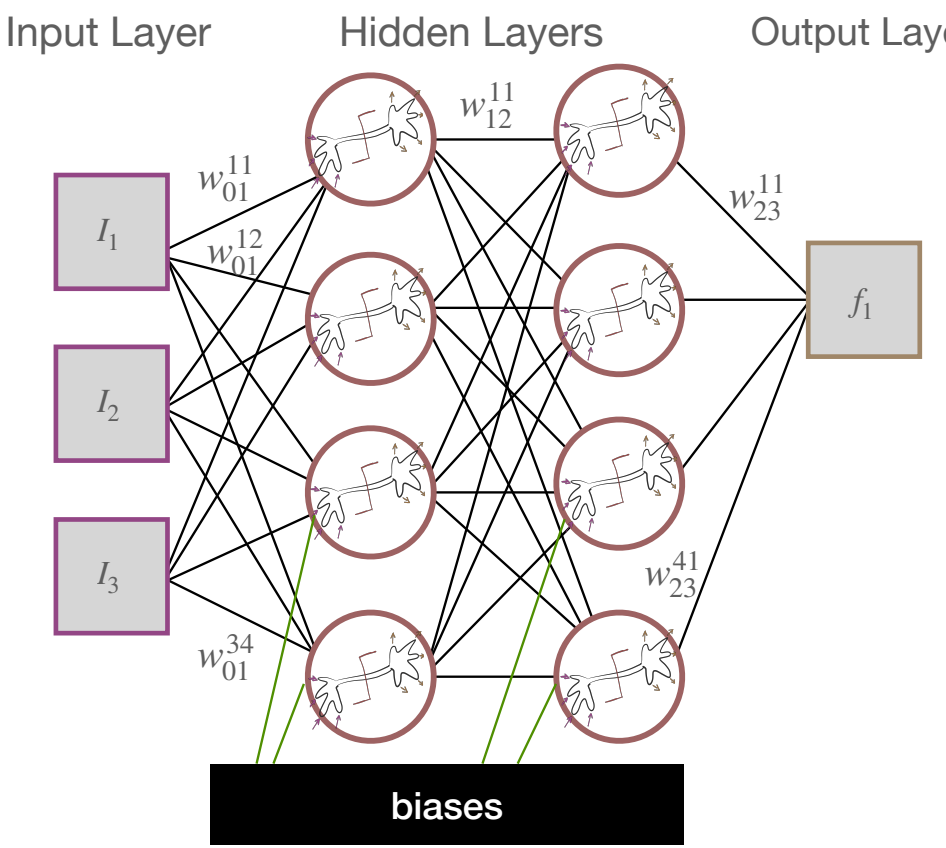
Artificial Neural Networks

- General architecture of a simple NN



Artificial Neural Networks

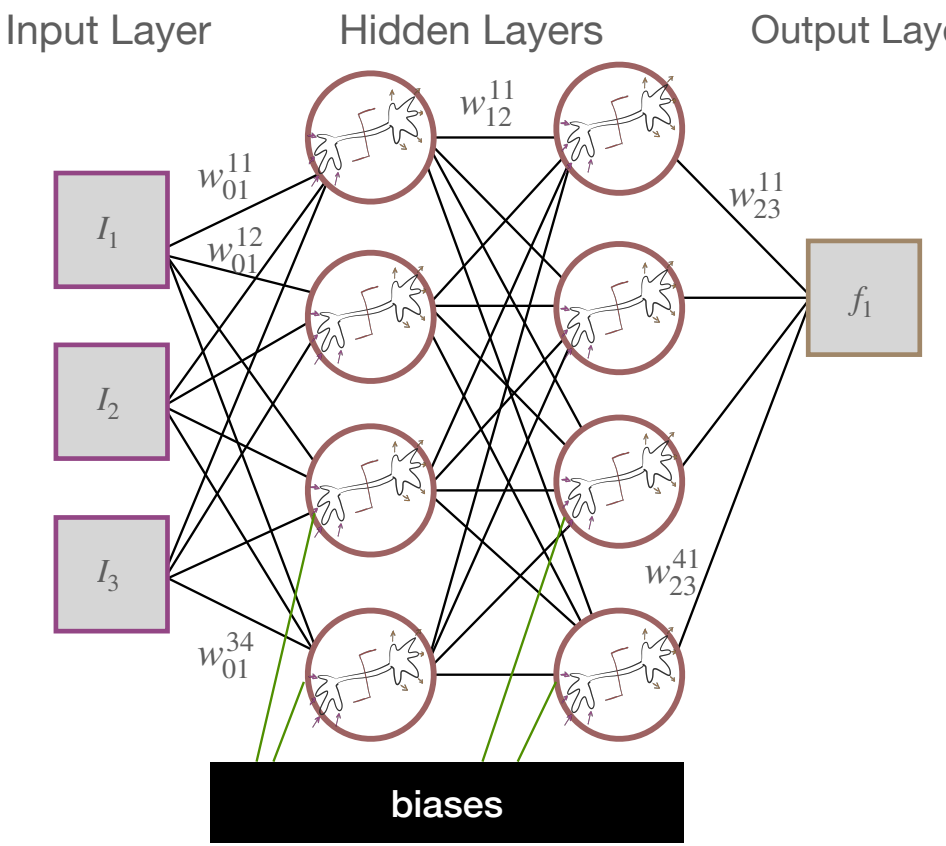
- General architecture of a simple NN



$$f_1 = b_3^1 + \sum_{k=1}^{4} w_{23}^{k1}\sigma_2\left(b_2^k + \sum_{j=1}^{4} w_{12}^{kj}\sigma_1\left(b_1^j + \sum_{i=1}^{3} w_{01}^{ij} I_i\right)\right)$$

Artificial Neural Networks

- General architecture of a simple NN

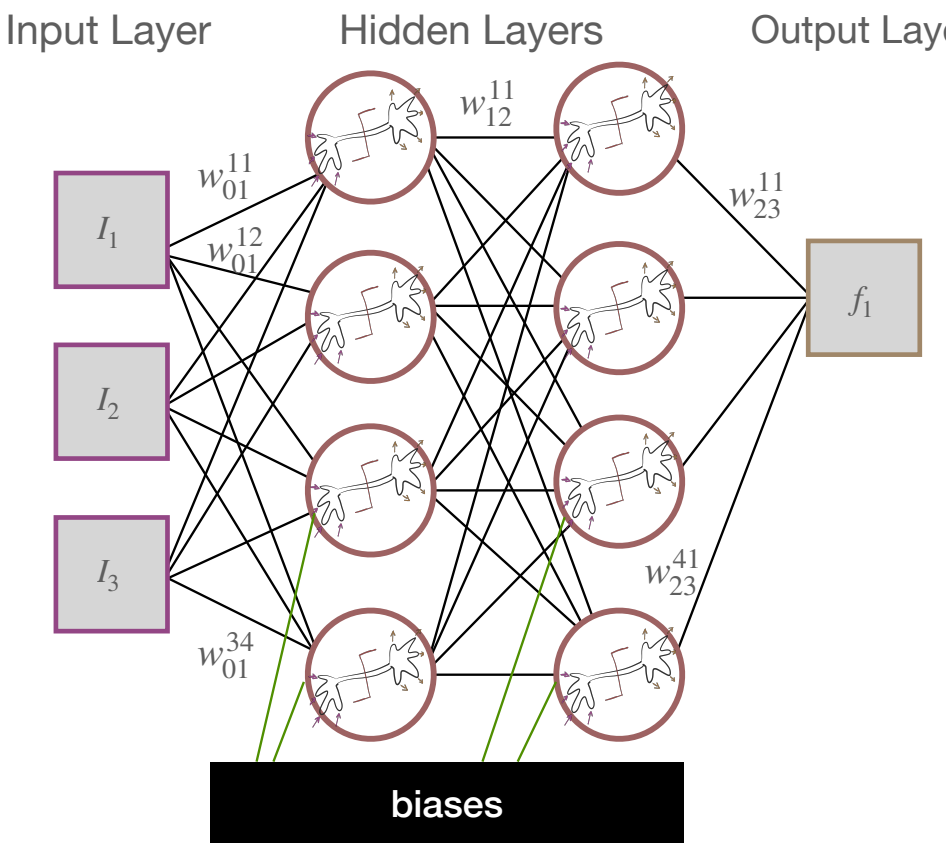


$$f_1 = b_3^1 + \sum_{k=1}^{4} w_{23}^{k1}\sigma_2\left(b_2^k + \sum_{j=1}^{4} w_{12}^{kj}\sigma_1\left(b_1^j + \sum_{i=1}^{3} w_{01}^{ij} I_i\right)\right)$$

$\{\mathbf{w}\}$ weights
$\{\mathbf{b}\}$ biases
learnable parameters
$\{\sigma\}$ activation functions
$\sigma_i(x) \rightarrow$ hyperbolic tan, sigmoid, ReLU, …

Artificial Neural Networks

- General architecture of a simple NN



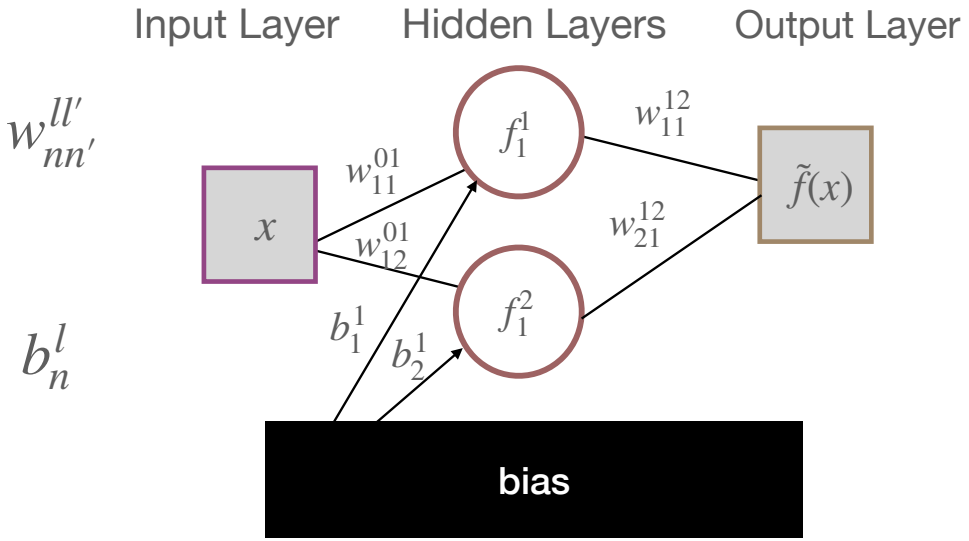
$$f_1 = b_3^1 + \sum_{k=1}^{4} w_{23}^{k1}\sigma_2\left(b_2^k + \sum_{j=1}^{4} w_{12}^{kj}\sigma_1\left(b_1^j + \sum_{i=1}^{3} w_{01}^{ij} I_i\right)\right)$$

$\{\mathbf{w}\}$ weights
$\{\mathbf{b}\}$ biases
learnable parameters
$\{\sigma\}$ activation functions
$\sigma_i(x) \rightarrow$ hyperbolic tan, sigmoid, ReLU, …

- ***Architecture***: Number of layers and number of nodes in each layer are hyperparameters
- ***Training***: Minimisation of loss function proportional to prediction error
- ***Loss minimisation***: stochastic gradient descent

Updating weights and biases

Forward and back propagation



$$\tilde{f}(x_i) = w_{21}^{12}\sigma(\underbrace{b_2^1 + w_{12}^{01}x_i}_{f_2^1}) + w_{11}^{12}\sigma(\underbrace{b_1^1 + w_{11}^{01}x_i}_{f_1^1})$$

$\sigma(f)$ is an activation function (e.g. sigmoid)

forward pass gives the loss

$$\mathscr{L} = \sum_i (\tilde{f}(x_i) - f(x_i))^2$$

backward pass gives the update recipe (SGD)

$$\mathbf{w} = \mathbf{w} - \eta \frac{\partial \mathscr{L}}{\partial \mathbf{w}_n}$$

learning rate

training sample index

- how much do we want to displace?
- true gradient approximated by error in one sample.
- repeat for each sample (epoch)

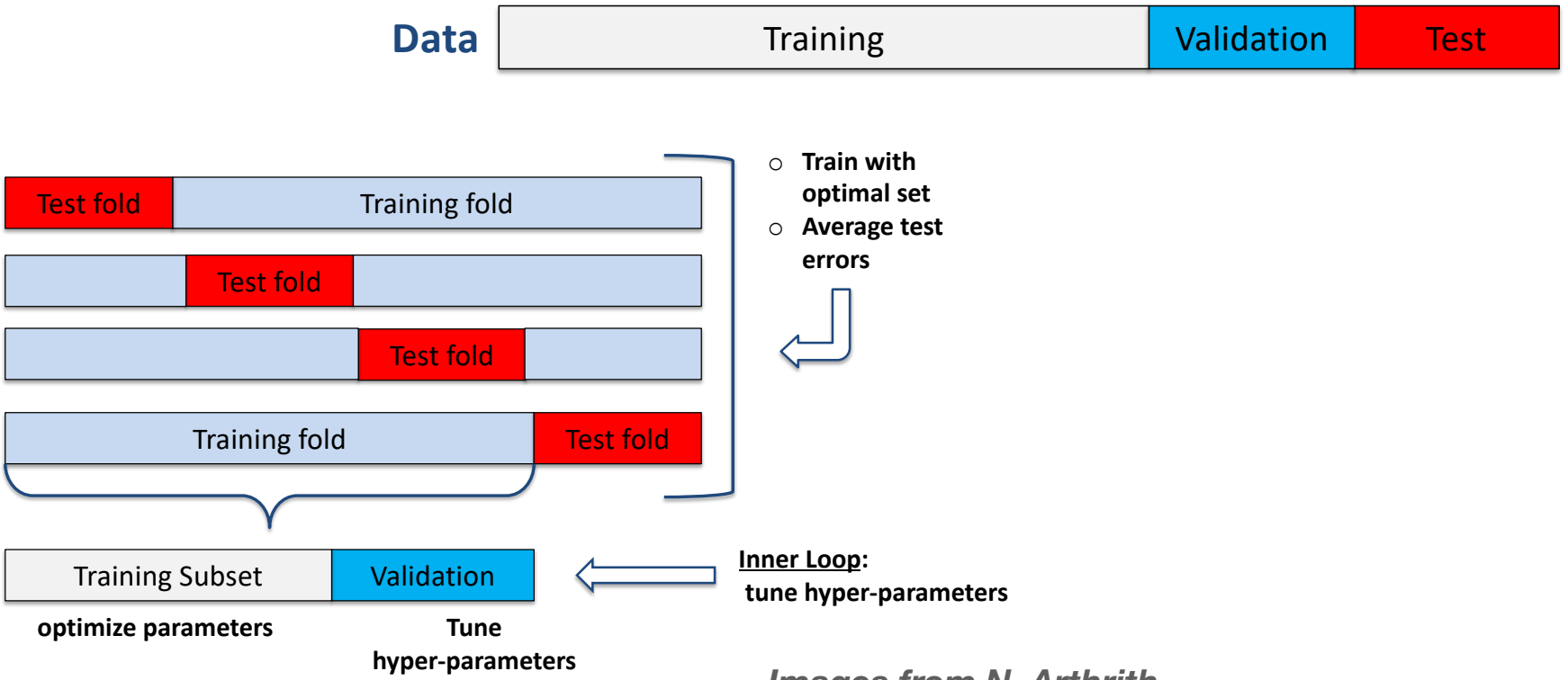
$$\frac{\partial \mathscr{L}}{\partial a_{11}^{01}} = \frac{\partial \mathscr{L}}{\partial \tilde{y}} \frac{\partial \tilde{y}}{\partial y_1^1} \frac{\partial y_1^1}{\partial a_{11}^{01}} \qquad \frac{\partial \mathscr{L}}{\partial b_1^1} = \frac{\partial \mathscr{L}}{\partial \tilde{y}} \frac{\partial \tilde{y}}{\partial y_1^1} \frac{\partial y_1^1}{\partial b_1^1}$$

$\bullet\bullet\bullet$ $\bullet\bullet\bullet$

Neural Network learning curves

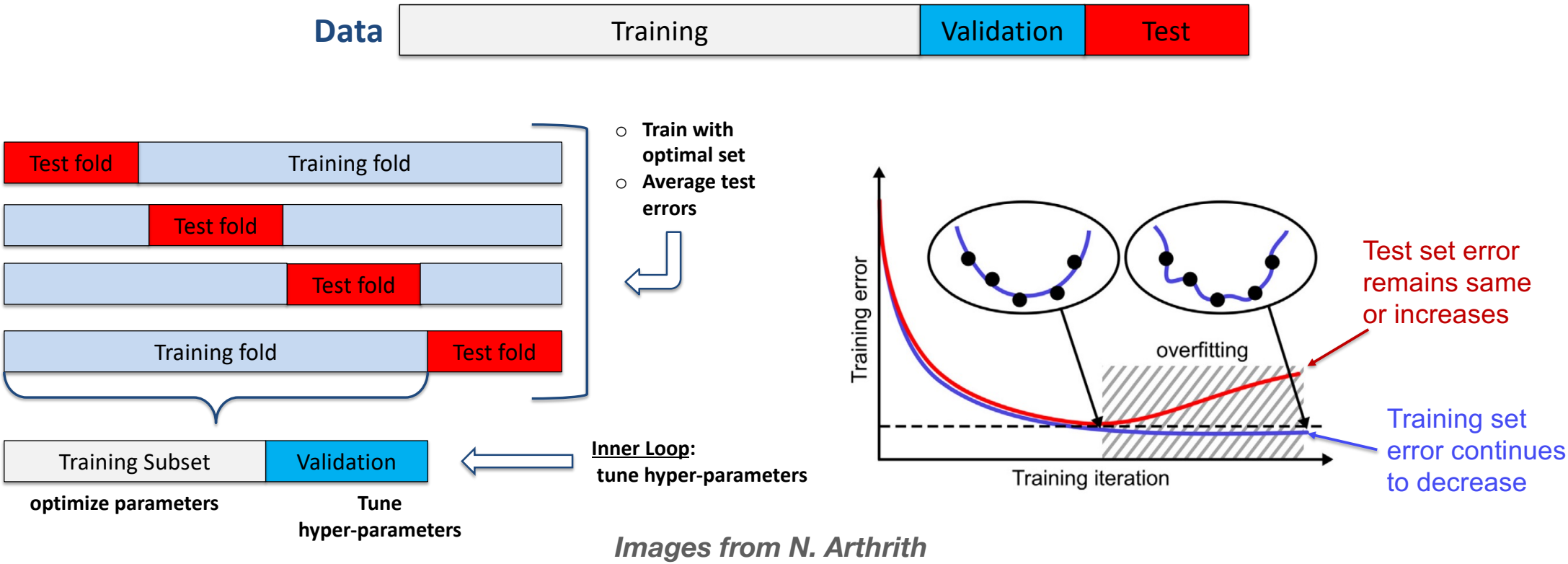
Images from N. Arthrith

Neural Network learning curves



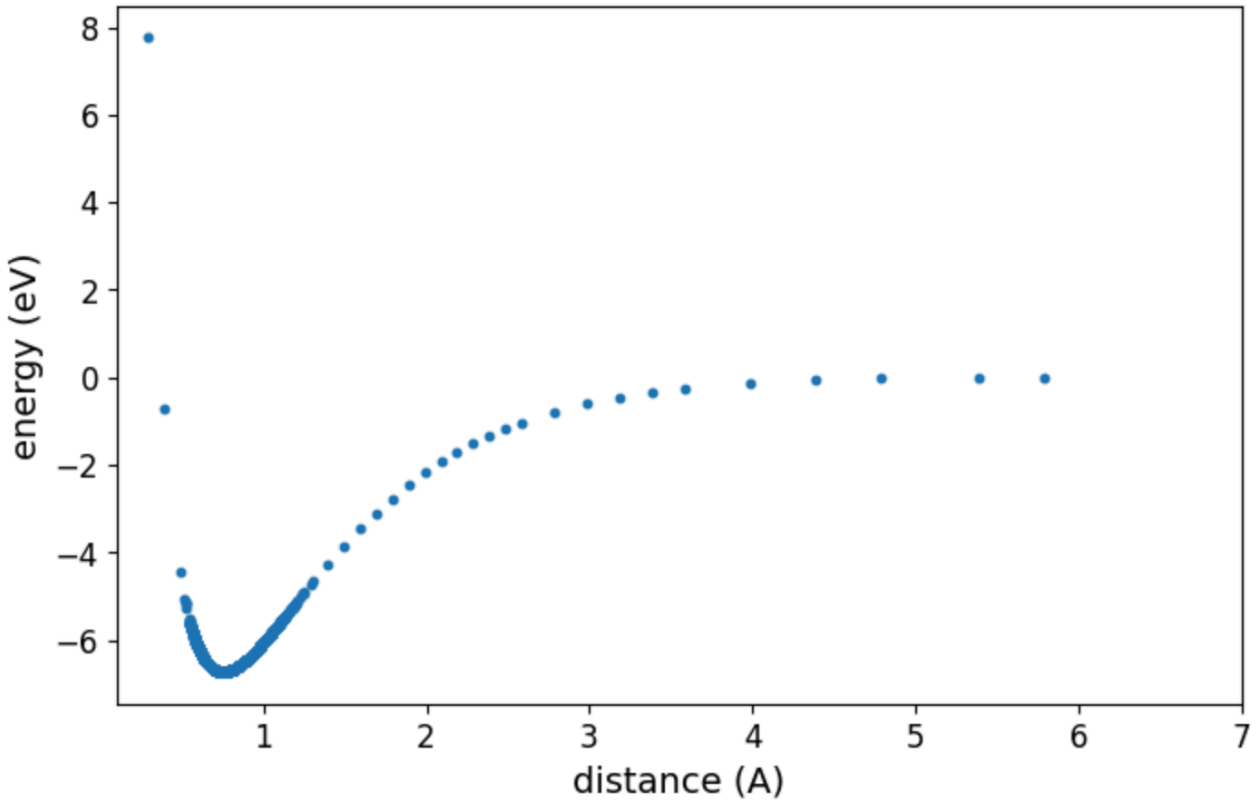
Images from N. Arthrith

Neural Network learning curves



Images from N. Arthrith

NN with a worked example



The End!

Thank you!