

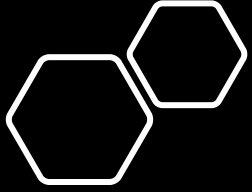
Interaction potentials: from first principles to machine learning

LJ and Morse fitted to DFT data (Xe, H), machine-learned potentials, coarse-graining

Based on a lecture by Simone Di Cataldo

Overview

1. Lennard-Jones: which physical interaction does it describe?
2. Interatomic potentials from first principles: DFT
3. Case study #1: LJ for xenon from DFT data
4. Case study #2: hydrogen, LJ fails and Morse works
5. Beyond fitted formulas: machine-learned potentials (PaiNN, MACE)
6. Second level: coarse-graining with ML (prior + neural network)



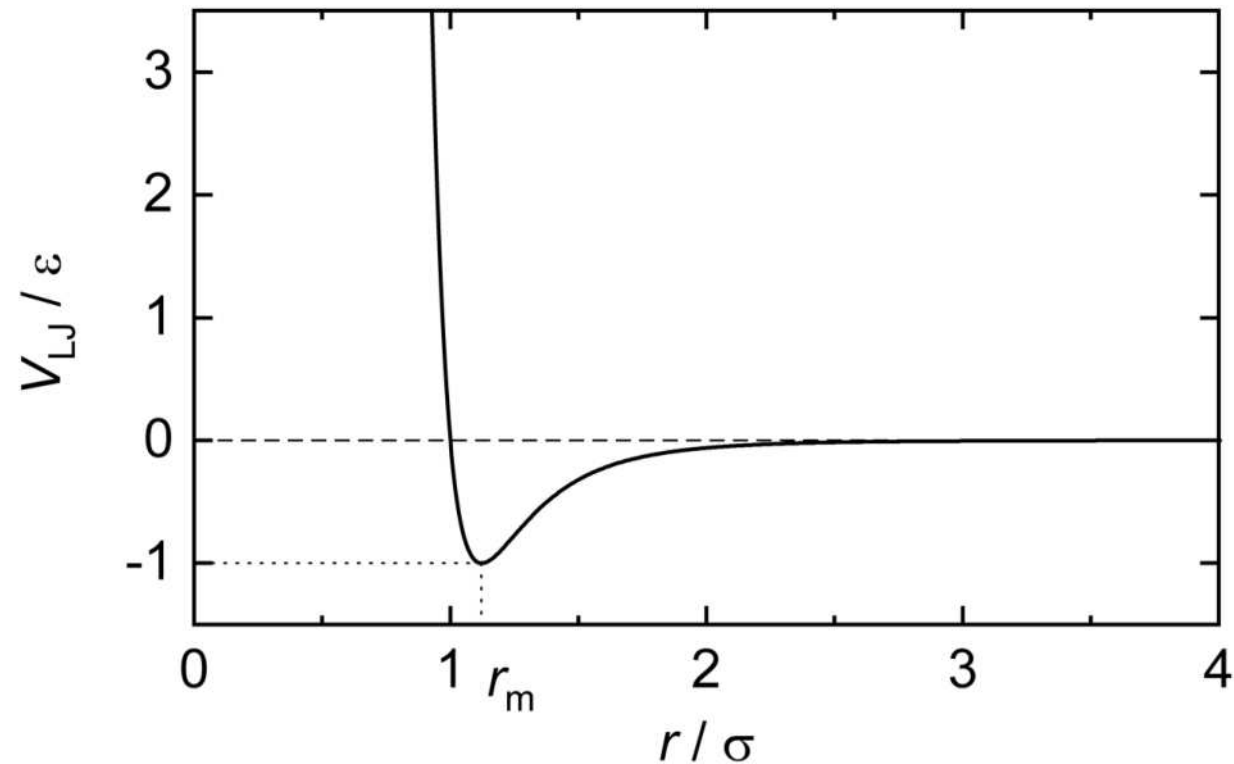
Lennard-Jones Potential

- Weakly attractive in a small region
- Strongly repulsive
- Quickly goes to zero
- **Which kind of physical interaction looks like this?**

$$V(r) = 4\epsilon \left(\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right)$$

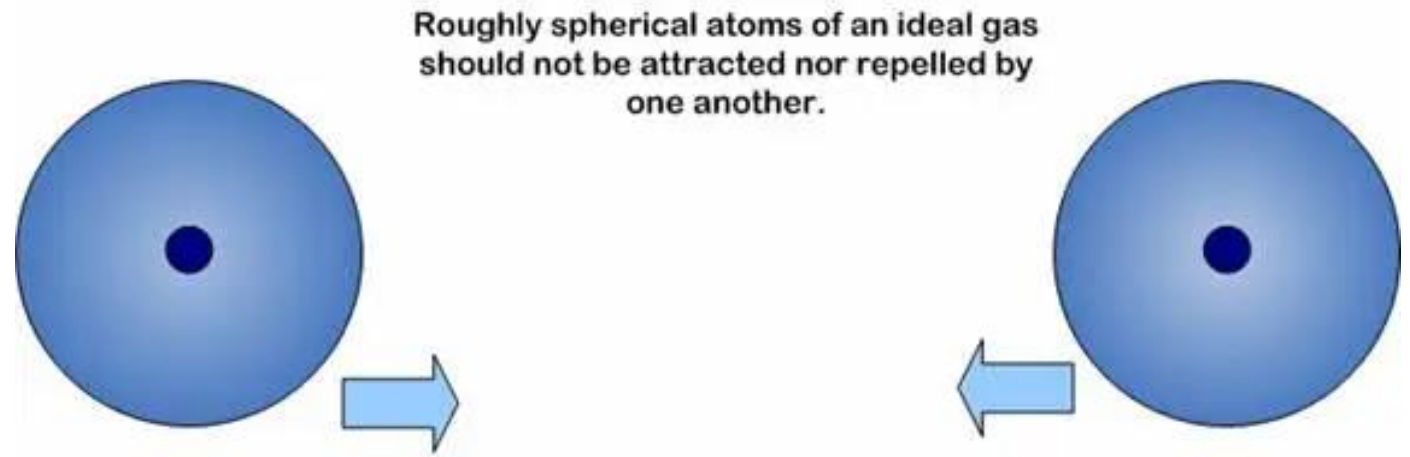
$$r_{min} = 2^{1/6} \sigma$$

$$V = -\epsilon$$



Lennard-Jones Potential

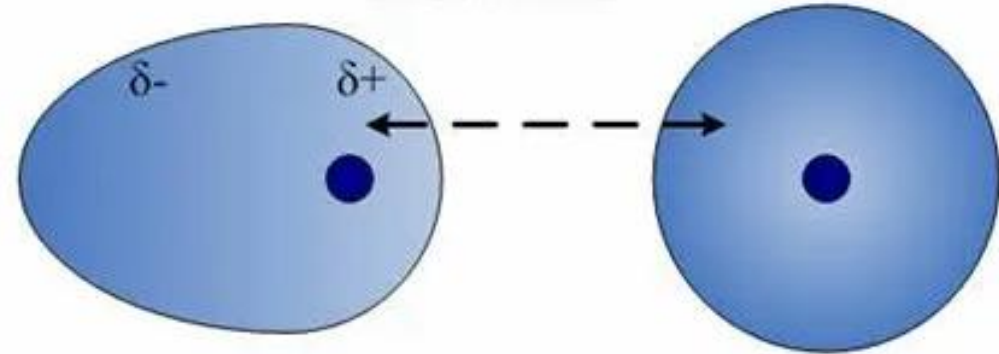
- Weakly attractive in a small region
- Strongly repulsive
- Quickly goes to zero
- **Induced-dipole with induced-dipole**



Lennard-Jones Potential

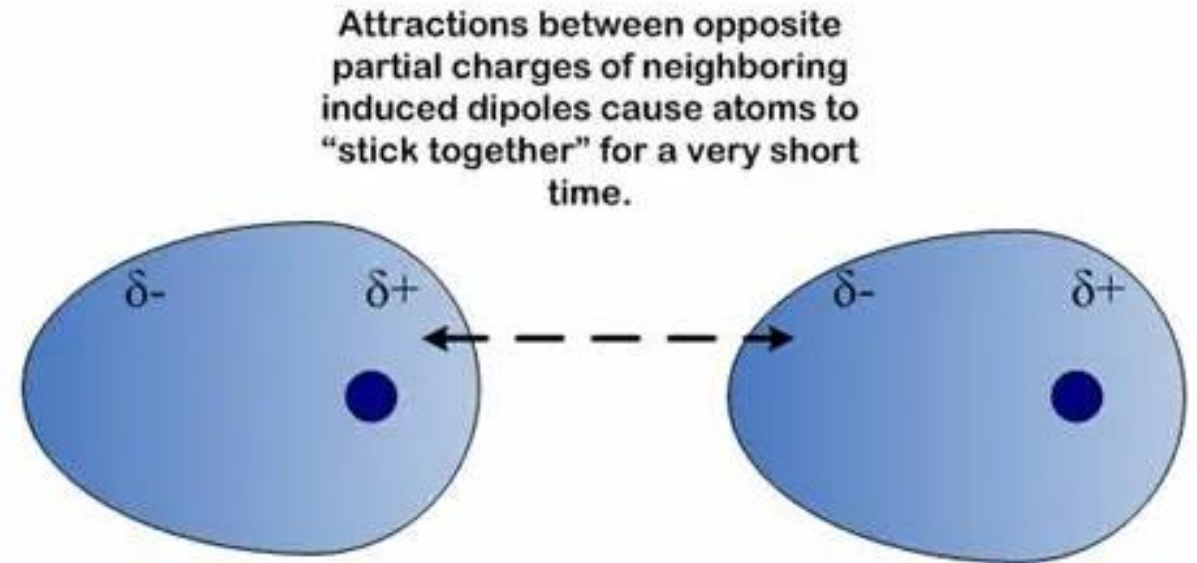
- Weakly attractive in a small region
- Strongly repulsive
- Quickly goes to zero
- **Induced-dipole with induced-dipole**

A real gas atom can have an instantaneous dipole. Partial charges on one atom cause a neighboring atom to distort due to the electrostatic attractions/repulsions of their electron clouds.



Lennard-Jones Potential

- Weakly attractive in a small region
- Strongly repulsive
- Quickly goes to zero
- **Induced-dipole with induced-dipole**



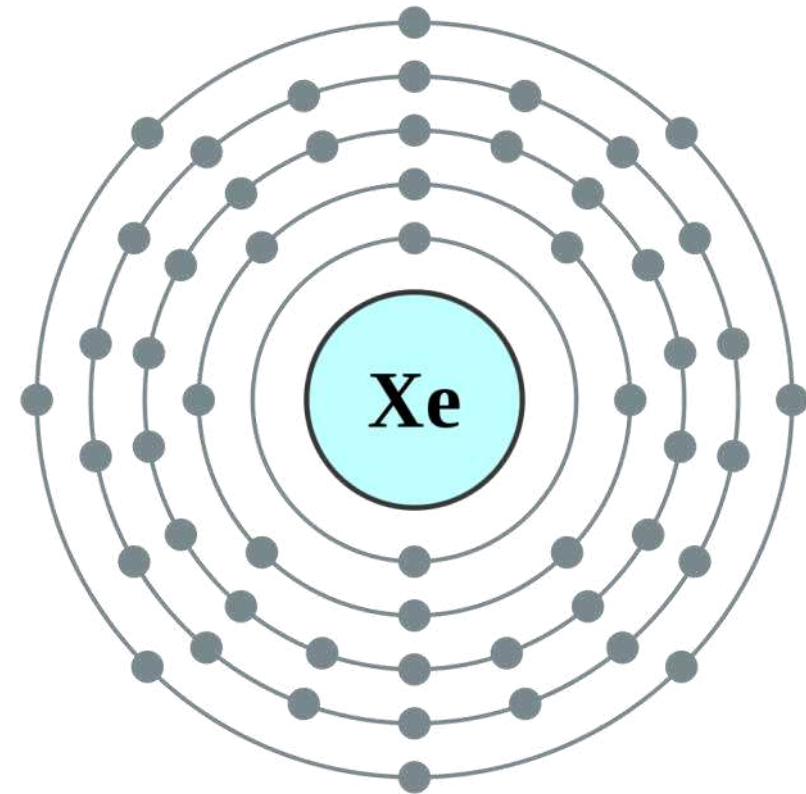
Which kind of atoms interact like this?

Case study #1: LJ Potential for Xe atoms

- To study a system of Xe atoms, we need to know the two Lennard-Jones parameters ϵ and σ
- **How can we determine them?**
 - Try to fit experiments
 - From first principles
- What are the advantages of the second approach?
 - More consistent
 - Much faster than an experiment
 - Much cheaper than an experiment
 - Experiments may not be available at all

54: Xenon

2,8,18,18,8



Interatomic potential from first principles

Many-body Hamiltonian

$$H = \underbrace{-\frac{1}{2} \sum_{i=1}^{N_e} \nabla_i^2}_{\text{electronic kinetic}} \underbrace{-\frac{1}{2} \sum_{I=1}^{N_a} \nabla_I^2}_{\text{ionic kinetic}} \underbrace{-\frac{1}{2} \sum_{i=1}^{N_e} \sum_{I=1}^{N_a} \frac{Z_I}{|\vec{r}_i - \vec{R}_I|}}_{\text{electron-ion}} + \underbrace{\frac{1}{2} \sum_{i=1}^{N_e} \sum_{j=1}^{N_e} \frac{1}{|\vec{r}_i - \vec{r}_j|}}_{\text{electron-electron}} + \underbrace{\frac{1}{2} \sum_{I=1}^{N_a} \sum_{J=1}^{N_a} \frac{1}{|\vec{R}_I - \vec{R}_J|}}_{\text{ion-ion}}$$

H_e H_I

The *electronic eigenenergies* depend parametrically on the ionic positions
This is what the interaction potential does

Born-Oppenheimer approximation:
 Decouple electronic and ionic degrees of freedom

$$H_e = -\frac{1}{2} \sum_{i=1}^{N_e} \nabla_i^2 - \frac{1}{2} \sum_{i=1}^{N_e} \sum_{I=1}^{N_a} \frac{Z_I}{|\vec{r}_i - \vec{R}_I|} + \frac{1}{2} \sum_{i=1}^{N_e} \sum_{j=1}^{N_e} \frac{1}{|\vec{r}_i - \vec{r}_j|}$$

$$H_e \psi(\vec{r}_1, \dots, \vec{r}_{N_e}) = E_e(\vec{R}_1, \dots, \vec{R}_{N_a}) \psi(\vec{r}_1, \dots, \vec{r}_{N_e})$$

Integrating out the electrons: **the first coarse-graining**

Solving this equation for more than a **few** electrons is practically impossible

Density Functional Theory

How to solve the Schrödinger's equation for many electrons

Density Functional Theory allows us to compute the **ground-state electronic energy** and charge density of a N electron system, which N up to tens of thousands of electrons.

In a nutshell, DFT is based on the construction of an auxiliary problem of *noninteracting* electrons in a fictitious external potential (v_{scf}) such as to reproduce exactly the same ground-state charge density as the interacting problem. Since in the auxiliary problem electrons are *noninteracting*, the Schrödinger's equation becomes that of a *single* electron.

$$v_{ext}(\vec{r}) = \sum_I \int \frac{Z_I}{|\vec{r} - \vec{R}_I|} d^3r'$$
$$v_H(\vec{r}) = \frac{1}{2} \int \frac{n(\vec{r}')}{|\vec{r} - \vec{r}'|} d^3r'$$

$$\left\{ \begin{array}{l} v_{scf}(\vec{r}) = v_{ext}(\vec{r}) + v_H(\vec{r}) + v_{XC}(\vec{r}) \\ [-\frac{1}{2}\nabla^2 + v_{scf}(\vec{r})]\psi_i(\vec{r}) = \varepsilon_i\psi_i(\vec{r}) \\ n(\vec{r}) = \sum_{i=1}^{N/2} |\psi_i(\vec{r})|^2 \end{array} \right.$$

Codes: Quantum ESPRESSO, Abinit, CASTEP, VASP. Here VASP gives the **total energy E(r)** of two Xe atoms at distance r.

Workflow: LJ Potential for two Xe atoms

1. Generate a set of atomic positions for two Xe atoms
2. Compute the total electronic energy for each configuration
3. Collect the data into a file with two columns: $[r_i, V(r_i)]$
4. Import the data in python and fit

Workflow: LJ Potential for two Xe atoms

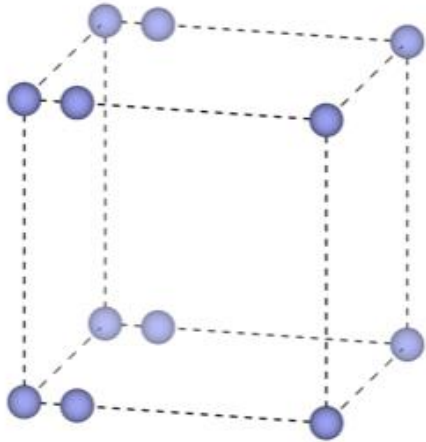
1. Generate a set of atomic positions for two Xe atoms
 - What is a sensible **range** of distances?

Atomic properties	
Oxidation states	0 , +2, +4, +6, +8 (rarely more than 0; a weakly acidic oxide)
Electronegativity	Pauling scale: 2.6
Ionization energies	1st: 1170.4 kJ/mol 2nd: 2046.4 kJ/mol 3rd: 3099.4 kJ/mol
Covalent radius	140±9 pm
Van der Waals radius	216 pm

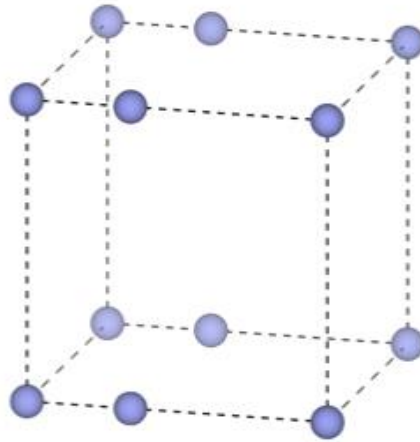
Let's visualize the atoms in the structures we just created

Why are there atoms in all corners of the cube?

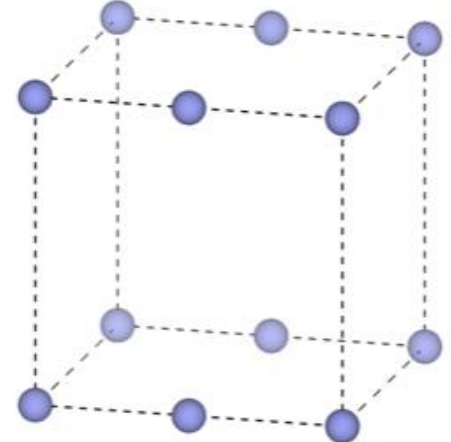
Step 0



Step 12



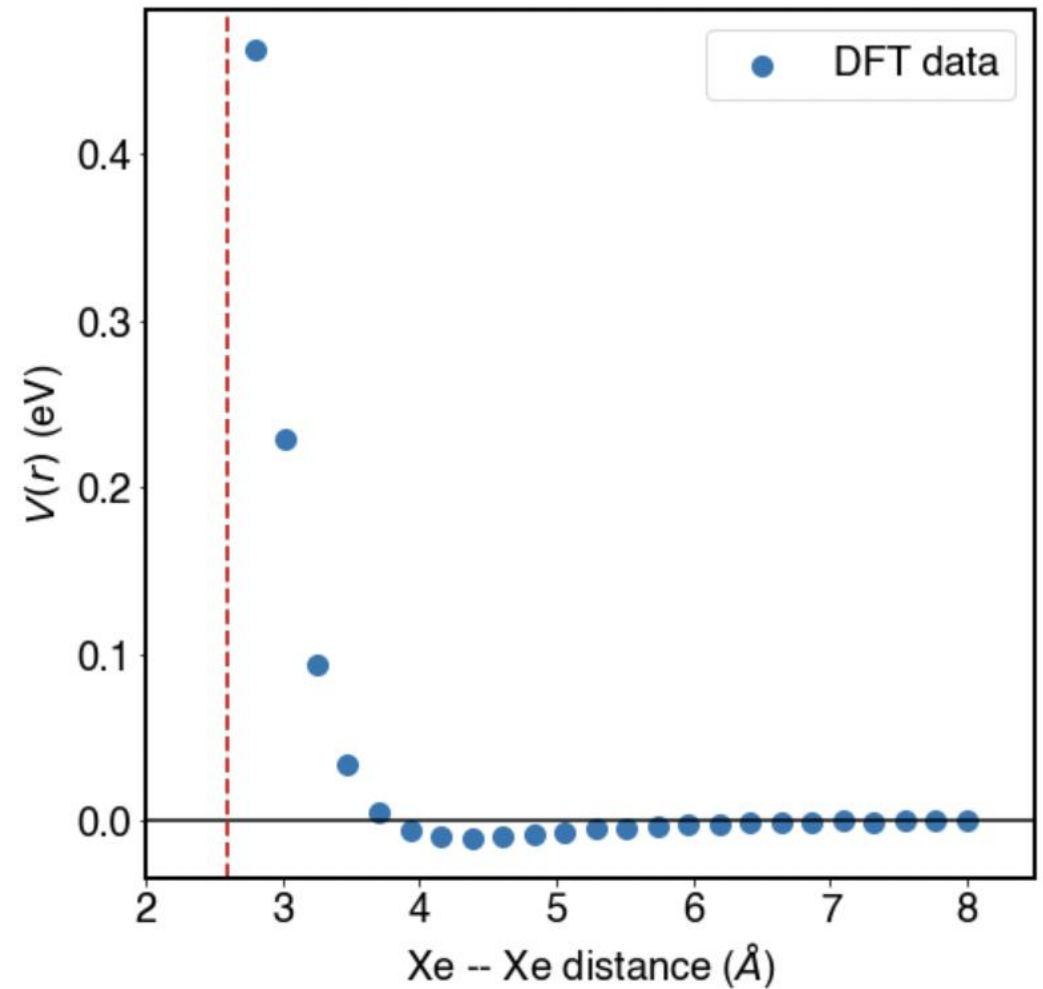
Step 23



Step 23: the second atom is halfway along the edge, $d = L/2 = 8 \text{ \AA}$ (cubic cell, $L = 16 \text{ \AA}$). Why does the scan stop there?

Collected DFT data

Is this what we
expected?



Fit the data points and plot

- Fit with $V(r) = 4\epsilon[(\sigma/r)^{12} - (\sigma/r)^6]$, all data with $r > 3 \text{ \AA}$
- Does the fit reproduce well our data everywhere?
- Should we be worried?
- Can we do better?

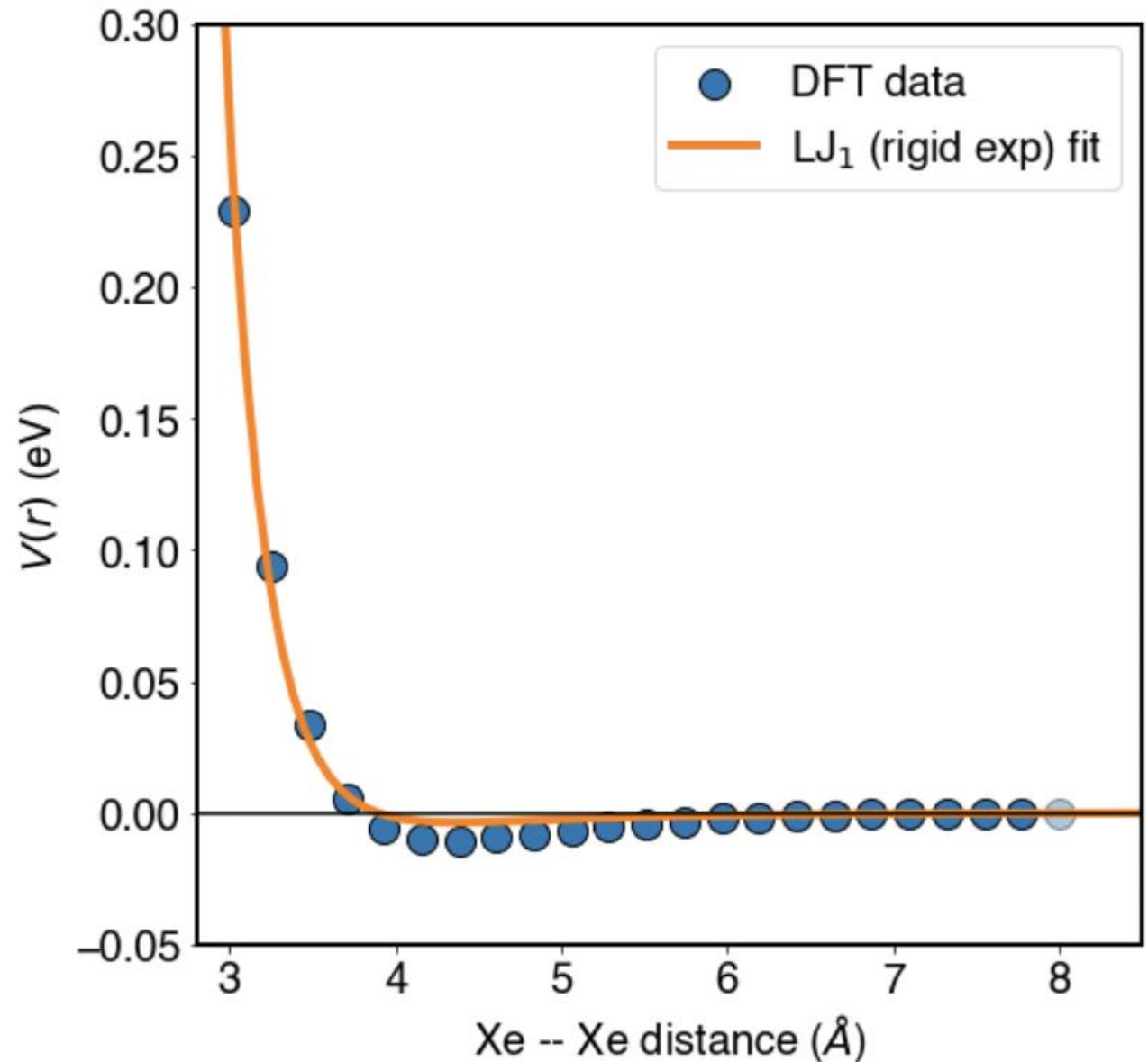
Fit results

ϵ (eV/atom): 0.0037

σ (Å): 3.8927

repulsive exponent: 12

binding distance: (Å): 4.36



Fit the data points and plot

- Fit only $r > 3.4 \text{ \AA}$ with a free repulsive exponent: $n \approx 10$ instead of 12
- Probably the best we can do with Lennard-Jones
- **fcc lattice parameter:** $6.0 \text{ \AA}$ (experiment $6.2 \text{ \AA}$)

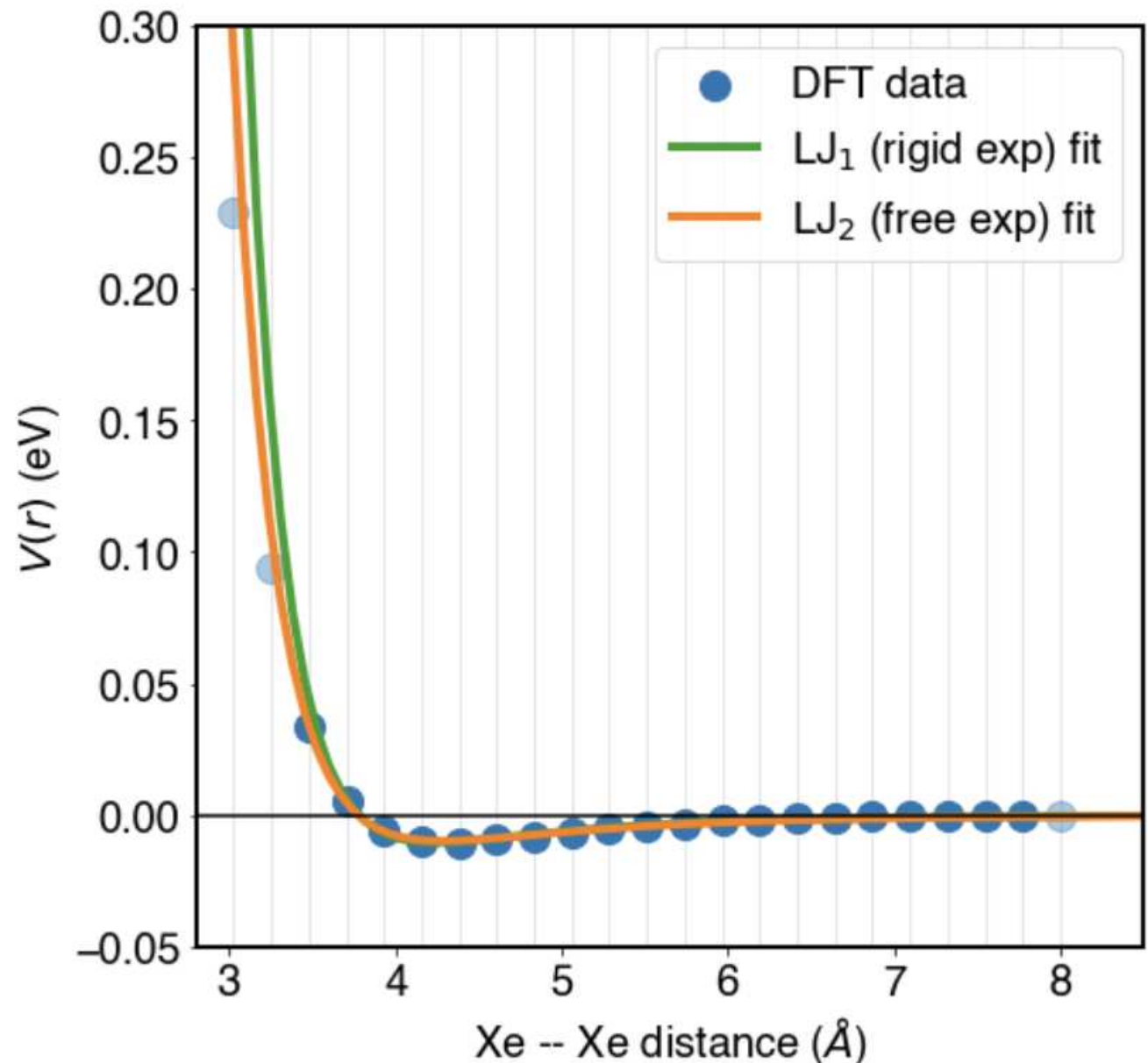
Fit results

ε (eV/atom): 0.0130

σ ($\text{\AA}$): 3.7748

repulsive exponent: 10.1

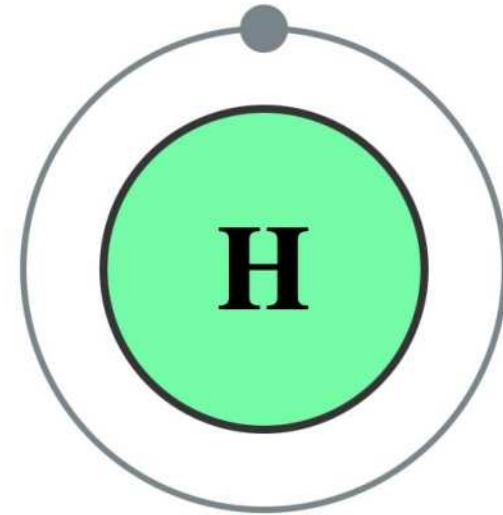
binding distance: ($\text{\AA}$): 4.23



Case study #2: LJ Potential for H atoms

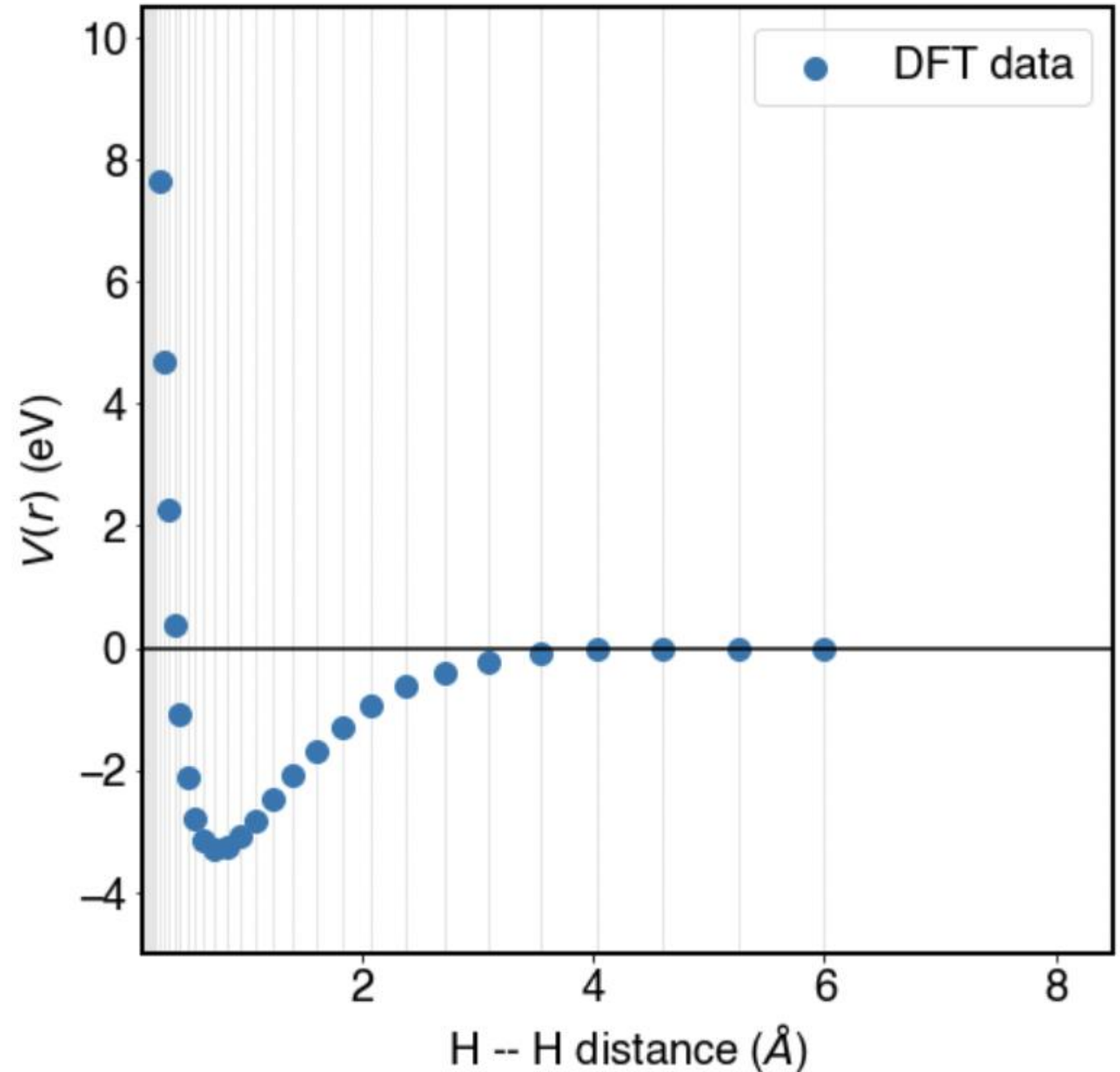
- Interatomic potential for two **hydrogen** atoms, determine ϵ and σ
- Can we do it?
- The procedure is exactly the same as before
 1. Generate a set of structures,
 2. Calculate the electronic energy with DFT
 3. Collect the data
 4. Fit

1: Hydrogen



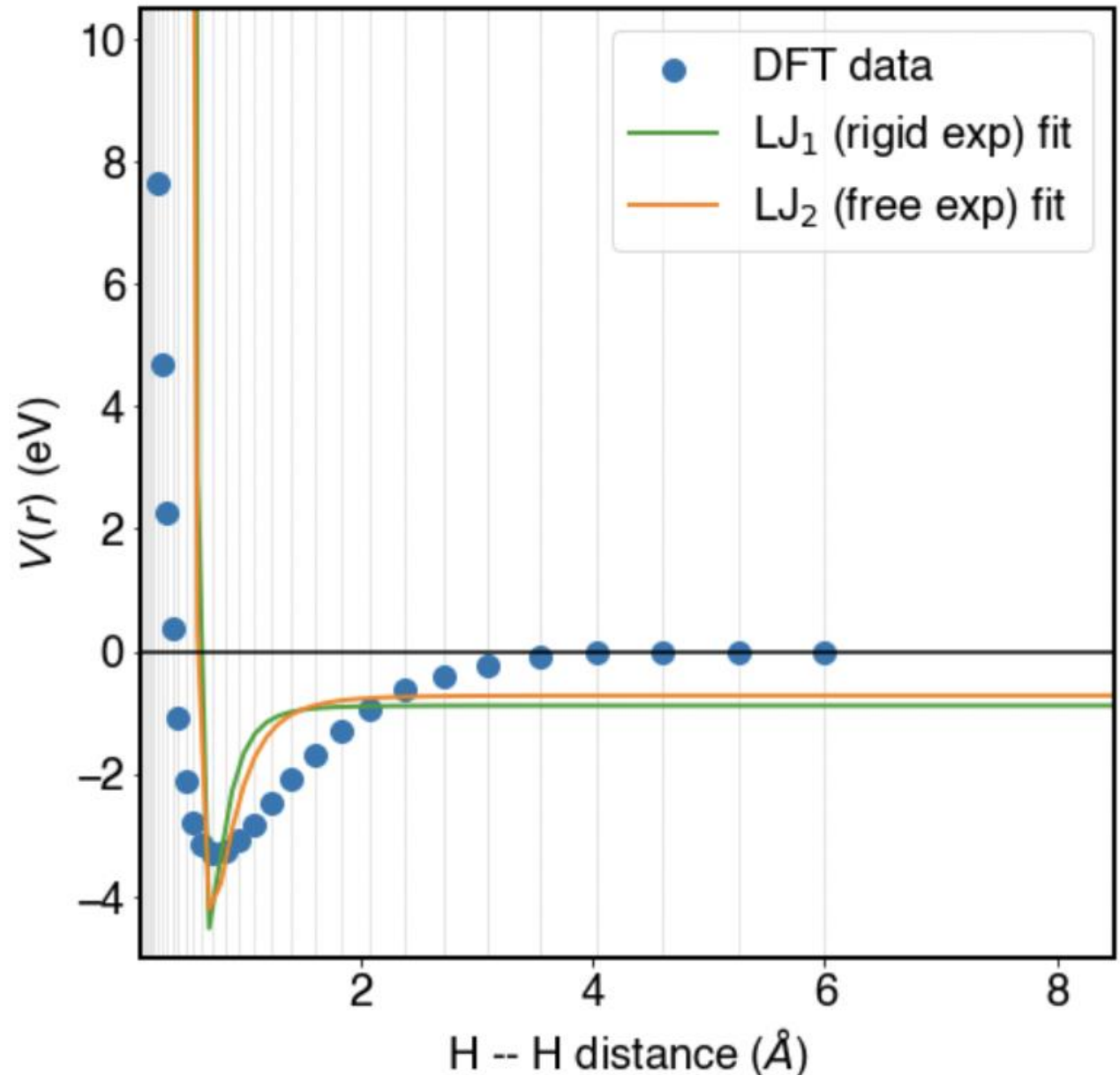
Case study #2: LJ Potential for H atoms

- How is the potential of H different from Xe?
 1. Deeper well
 2. Wider well
 3. Much shorter equilibrium distance



Case study #2: LJ Potential for H atoms

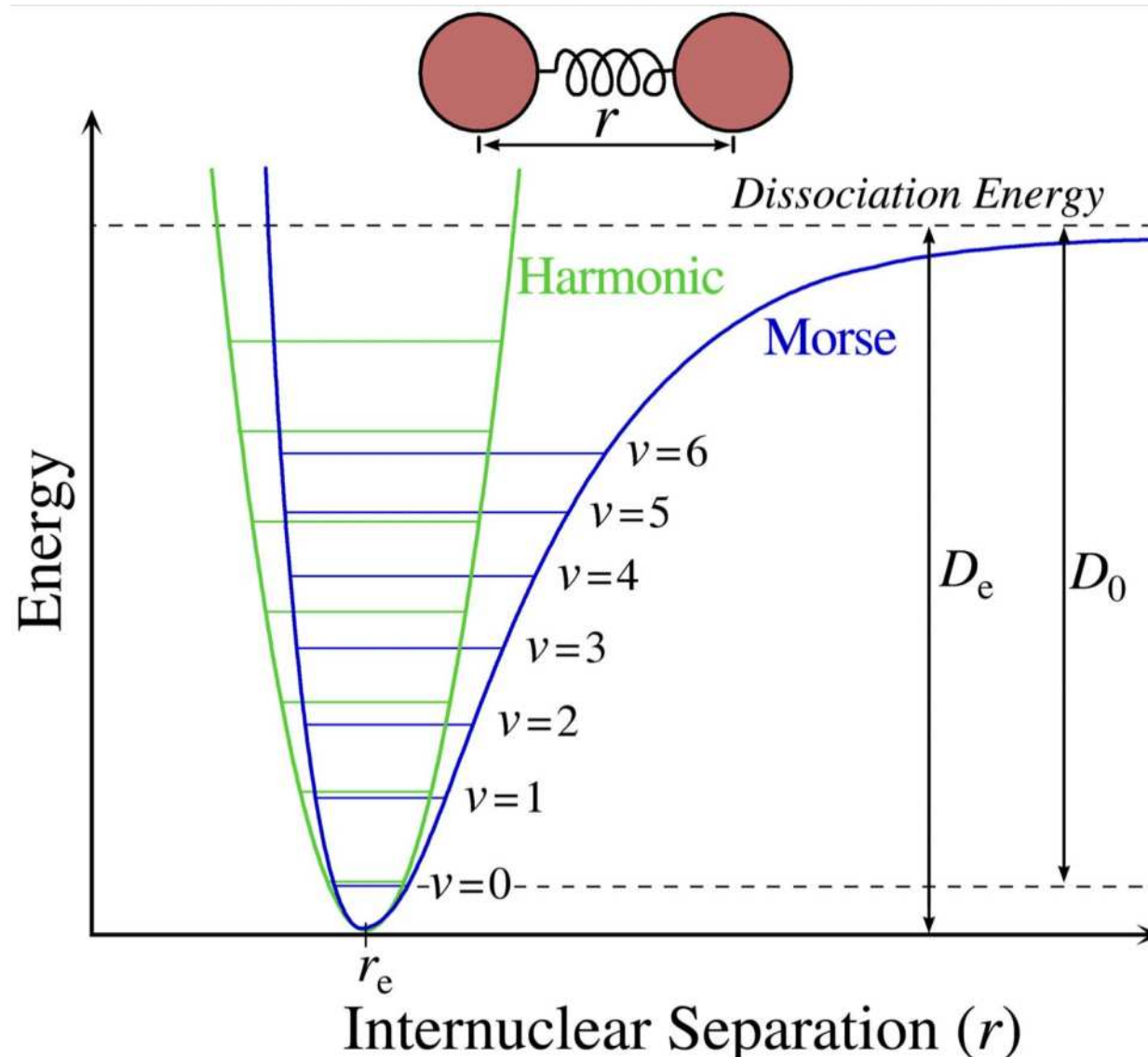
- Why is it so terrible?
 - The type of the interaction of two hydrogen atoms is **not** induced-dipole with induced dipole
 - As the two atoms get close their *electronic wavefunctions merge*, there is true Coulomb interaction, and even quantum exchange
- How to do better?



Morse Potential

- Models the *dissociation* of a diatomic molecule
- Used to describe the **anharmonic** vibrational frequencies
- **Three parameters**
 - $D_e \rightarrow$ depth of the well
 - $r_e \rightarrow$ equilibrium distance
 - $a \rightarrow$ curvature at the minimum

$$V_M(r) = D_e \left(1 - e^{-a(r-r_e)}\right)^2$$



Case study #2: Morse potential for H atoms

- Good description at all energies
- Good match with experimental H – H equilibrium distance (0.74 Å)

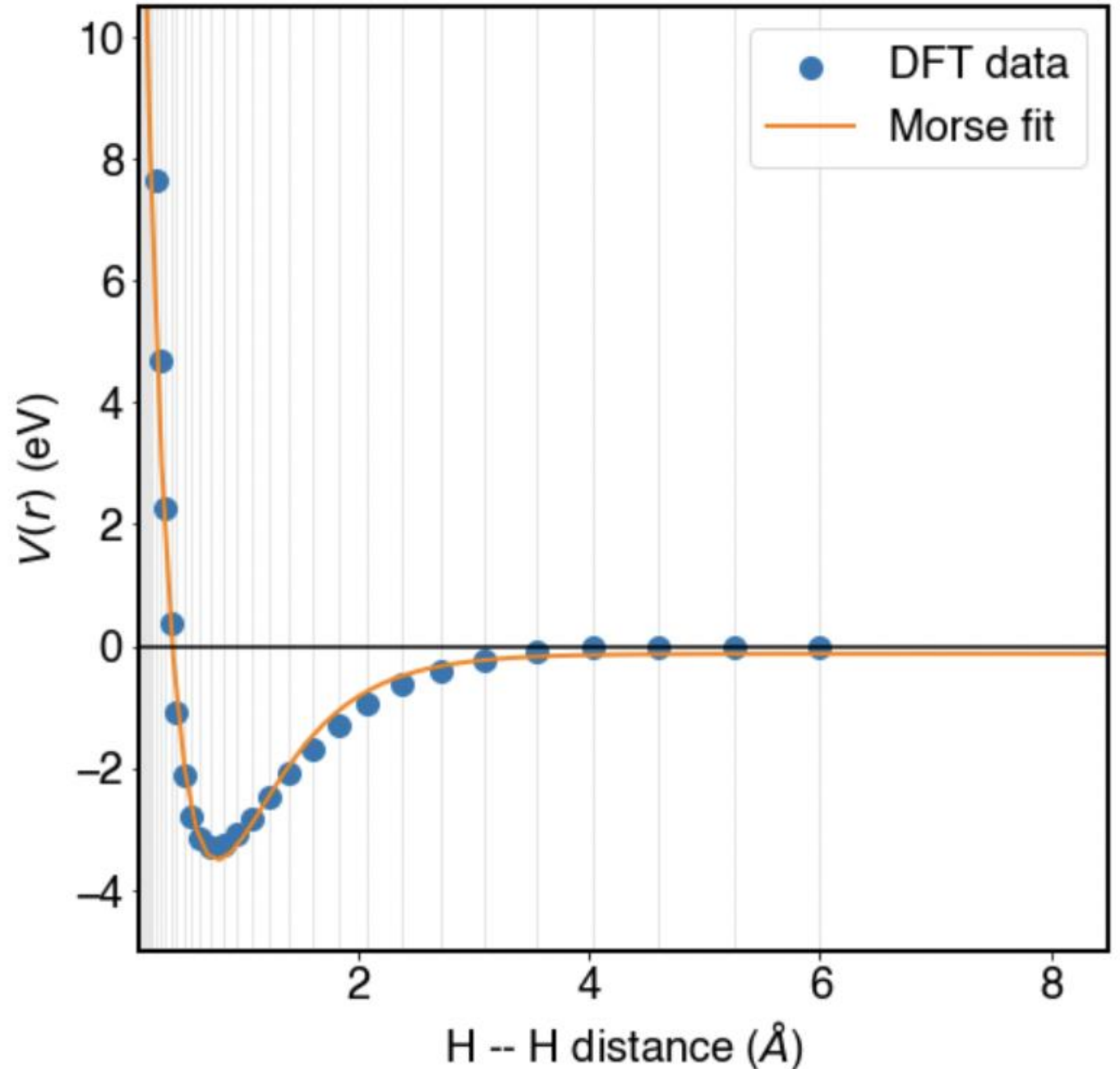
Fit results

D_e (eV/atom): 3.38

r_e (Å): 0.78

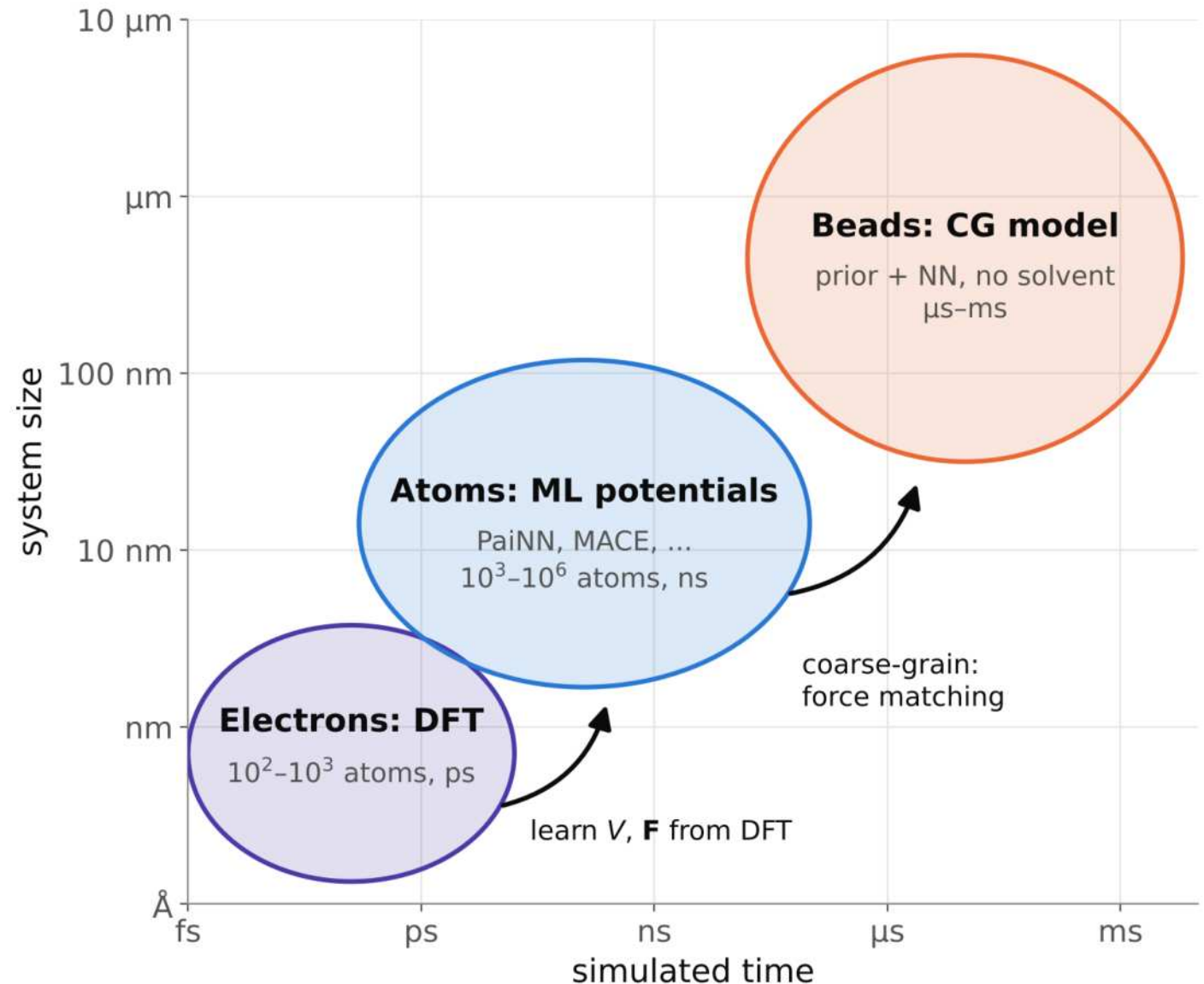
binding distance: (Å): 0.78

Shift: -3.5115



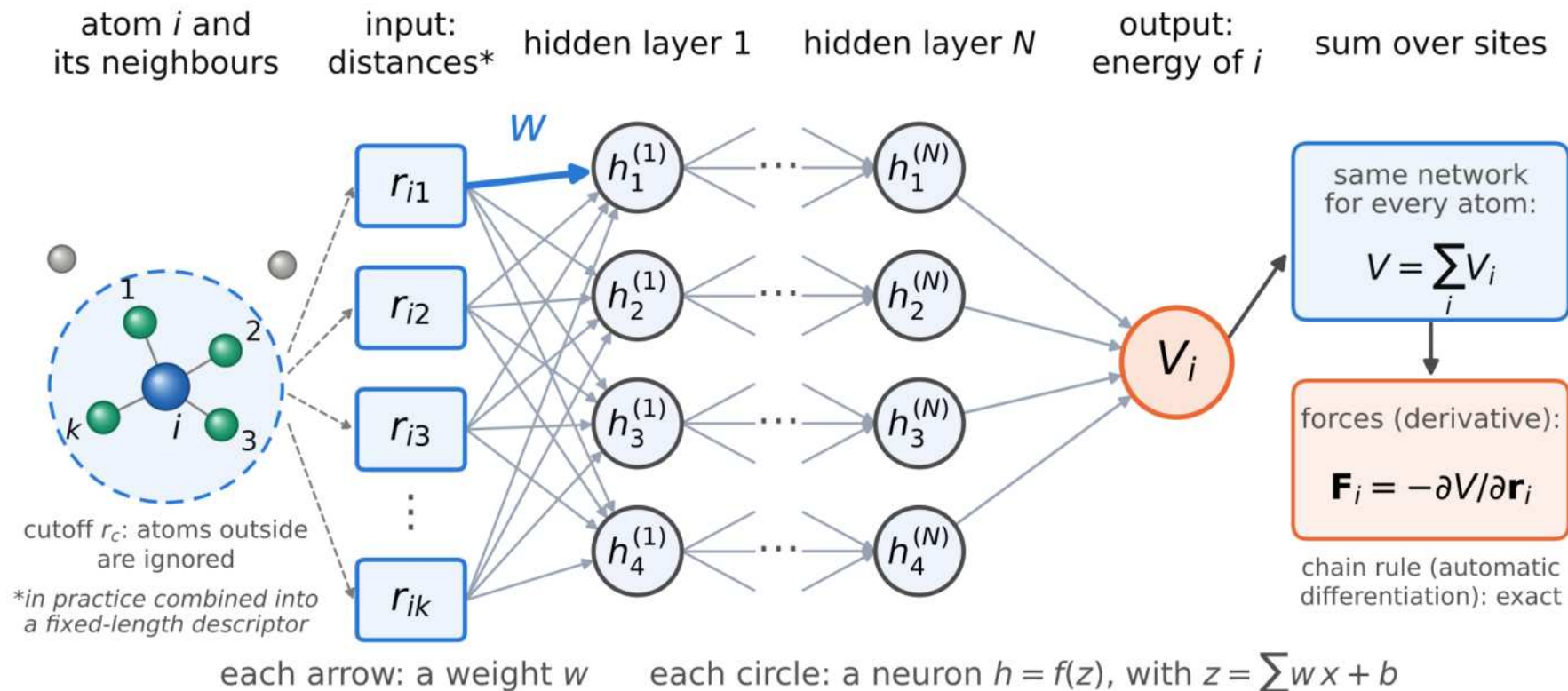
Beyond LJ and Morse: machine-learned potentials

- LJ and Morse: **a formula chosen by us**, with 2–3 parameters fitted to DFT
- Real interactions are many-body, and often we do not know the right functional form
- **Idea**: learn the energy as a function of all atomic positions directly from DFT data, with a neural network
- Each level is a **coarse-graining**: electrons $\rightarrow$ atoms gives an **energy** $V(r)$; atoms $\rightarrow$ beads gives a **free energy** (PMF), which depends on T



A neural network is just a function

Each atom: distances to its neighbours $\rightarrow$ its energy; $V = \text{sum over atoms}$; forces = derivative



$V_i = f_{\theta}(r_{i1}, r_{i2}, \dots)$: one network, many parameters θ (all the w and b), fitted to DFT

Inside the function: layers, weights and biases

Each layer is a matrix–vector product plus a bias, followed by a nonlinear function f

1 One layer

$$\underbrace{\begin{bmatrix} z_1 \\ \vdots \\ z_m \end{bmatrix}}_{\mathbf{z}} = \underbrace{\begin{bmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & & \vdots \\ w_{m1} & \cdots & w_{mn} \end{bmatrix}}_{\mathbf{W} \ (m \times n): \text{ weights}} \underbrace{\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}}_{\mathbf{x}} + \underbrace{\begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}}_{\mathbf{b}: \text{ biases}}$$

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b}) \quad f \text{ acts on each component}$$

2 The network: layers nested

$$V(\mathbf{x}; \theta) = L_{\text{out}}\left(L_3\left(L_2\left(L_1(\mathbf{x})\right)\right)\right)$$

$$L_l(\mathbf{h}) = f(\mathbf{W}^{(l)}\mathbf{h} + \mathbf{b}^{(l)}), \quad L_{\text{out}}(\mathbf{h}) = \mathbf{w}_{\text{out}} \cdot \mathbf{h} + b_{\text{out}}$$

parameters θ = all the $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$:
a nested function with many parameters

e.g. 10 inputs, 3 layers of 64 neurons, 1 output:

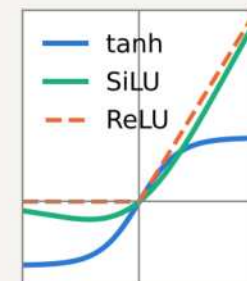
$$704 + 4160 + 4160 + 65 = 9\,089 \text{ parameters}$$

3 Why the nonlinear f ?

without f the layers collapse into one linear map:

$$\mathbf{W}^{(2)}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)} = \underbrace{\mathbf{W}^{(2)}\mathbf{W}^{(1)}}_{\mathbf{W}'}\mathbf{x} + \underbrace{\mathbf{W}^{(2)}\mathbf{b}^{(1)} + \mathbf{b}^{(2)}}_{\mathbf{b}'}$$

→ just a linear fit: f is what makes the function flexible



smooth f (tanh, SiLU)
→ smooth forces

$\mathbf{F} = -\nabla V$: chain rule,
as in training

ReLU has a kink:
forces would jump

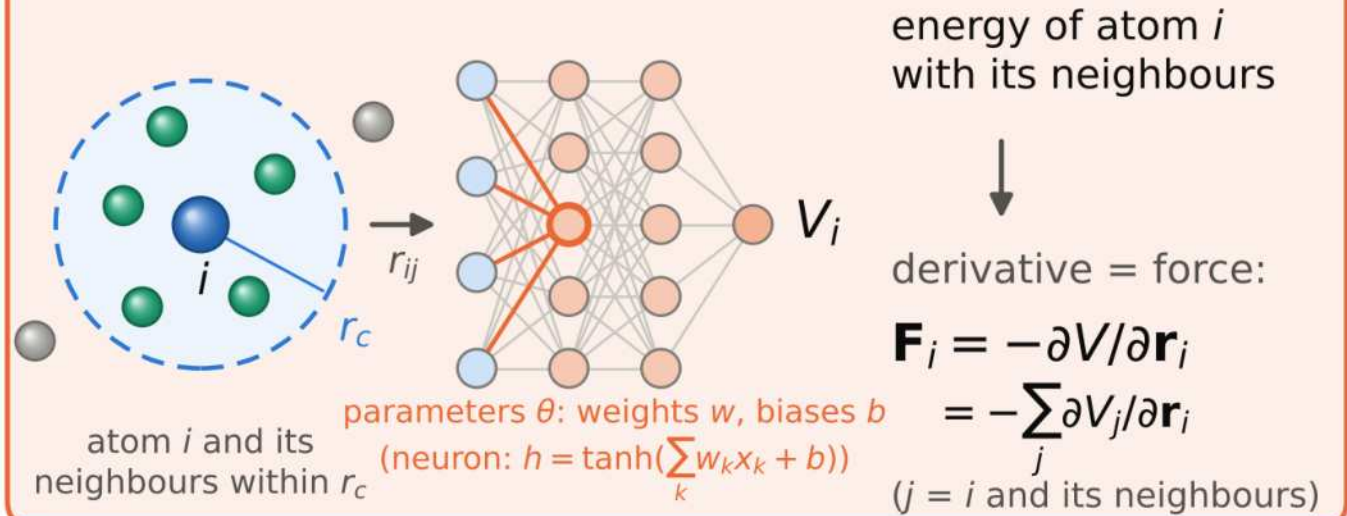
What is a neural-network potential?

- Just a **function** of the atomic positions, with parameters θ (**weights and biases**), that returns the **potential energy**
- Atom i and its neighbours within r_c $\rightarrow$ the network gives V_i : the **energy of atom i with its neighbours**
- Its derivative is the **force** on the atom: $\mathbf{F}_i = -\partial V / \partial \mathbf{r}_i$
- Training = **fitting** θ to DFT energies and forces, as we fitted ϵ and σ for LJ

Lennard-Jones: a function with 2 parameters

$$r \rightarrow 4\epsilon[(\sigma/r)^{12} - (\sigma/r)^6] \rightarrow V(r) \quad \text{parameters: } \epsilon, \sigma$$

Neural network: atom i + its neighbours $\rightarrow$ energy



Training = fitting, as for LJ (many more parameters)

$$\min_{\theta} \sum_{\text{configs}} \left[(V_{\theta} - V_{\text{DFT}})^2 + \lambda \sum_i |\mathbf{F}_{\theta,i} - \mathbf{F}_{\text{DFT},i}|^2 \right]$$

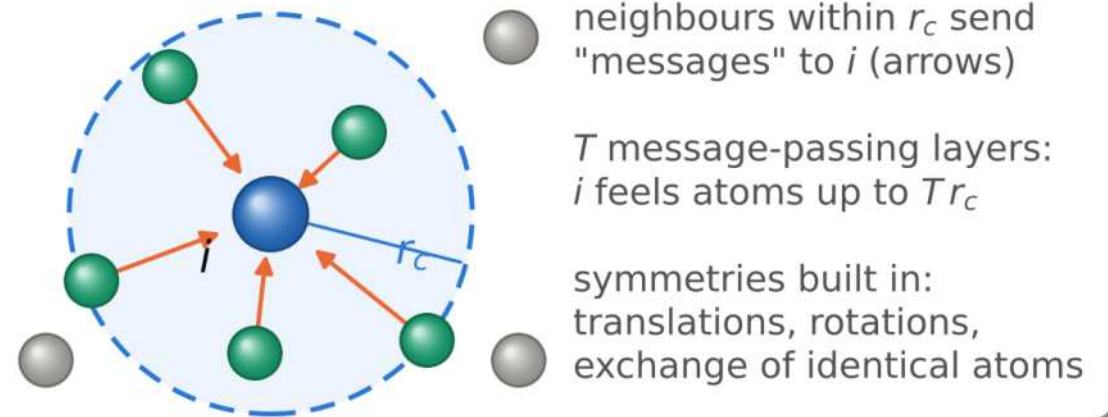
How a machine-learned potential works

- Potential energy V = sum of atomic terms V_i ; each depends on the **neighbours within a cutoff r_c**
- Trained on DFT **energies and forces** of many configurations
- Forces $\mathbf{F} = -\nabla V$ by **automatic differentiation**: conservative. Predicting $\mathbf{F}$ directly is faster, but not conservative (and MC needs V anyway)
- Symmetries built in: **PaiNN** (2021), **NequIP**, **MACE** (2022)
- Cost $\propto N$ (**DFT**: $\propto N^3$): 10^3 – 10^6 atoms for ns (DFT: 10^2 – 10^3 atoms, ps)
- Per atom and step: **10^5 – $10^7\times$ faster than DFT**, 10^2 – $10^3\times$ slower than Lennard-Jones

1 Training data from DFT

many configurations $\{\mathbf{r}\}$: DFT energies V and forces $\mathbf{F}$

2 Each atom sees only its neighbours

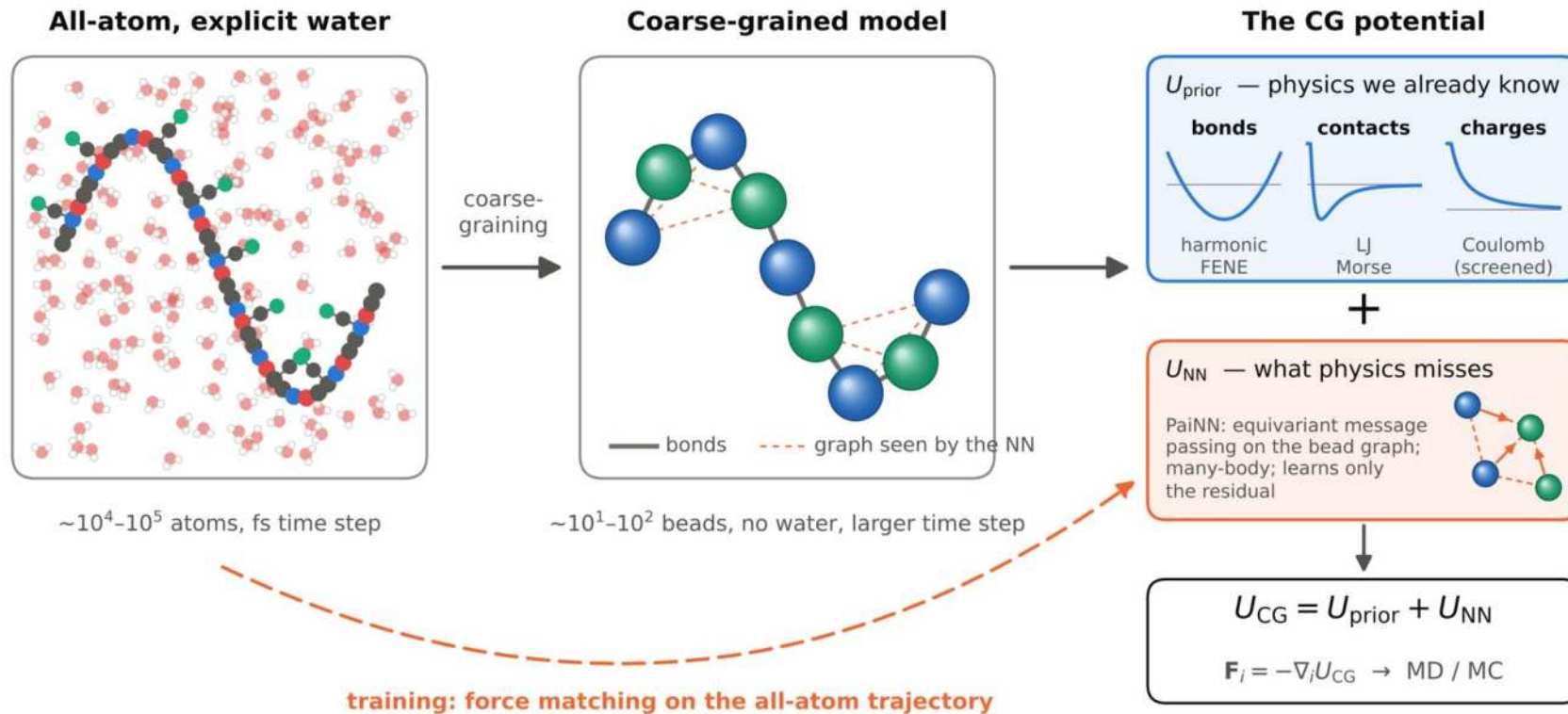


3 Neural network $\rightarrow$ energy $\rightarrow$ forces

$$V = \sum_i V_i^{\text{NN}} \quad \mathbf{F}_i = -\nabla_i V$$

Second level: coarse-graining with ML

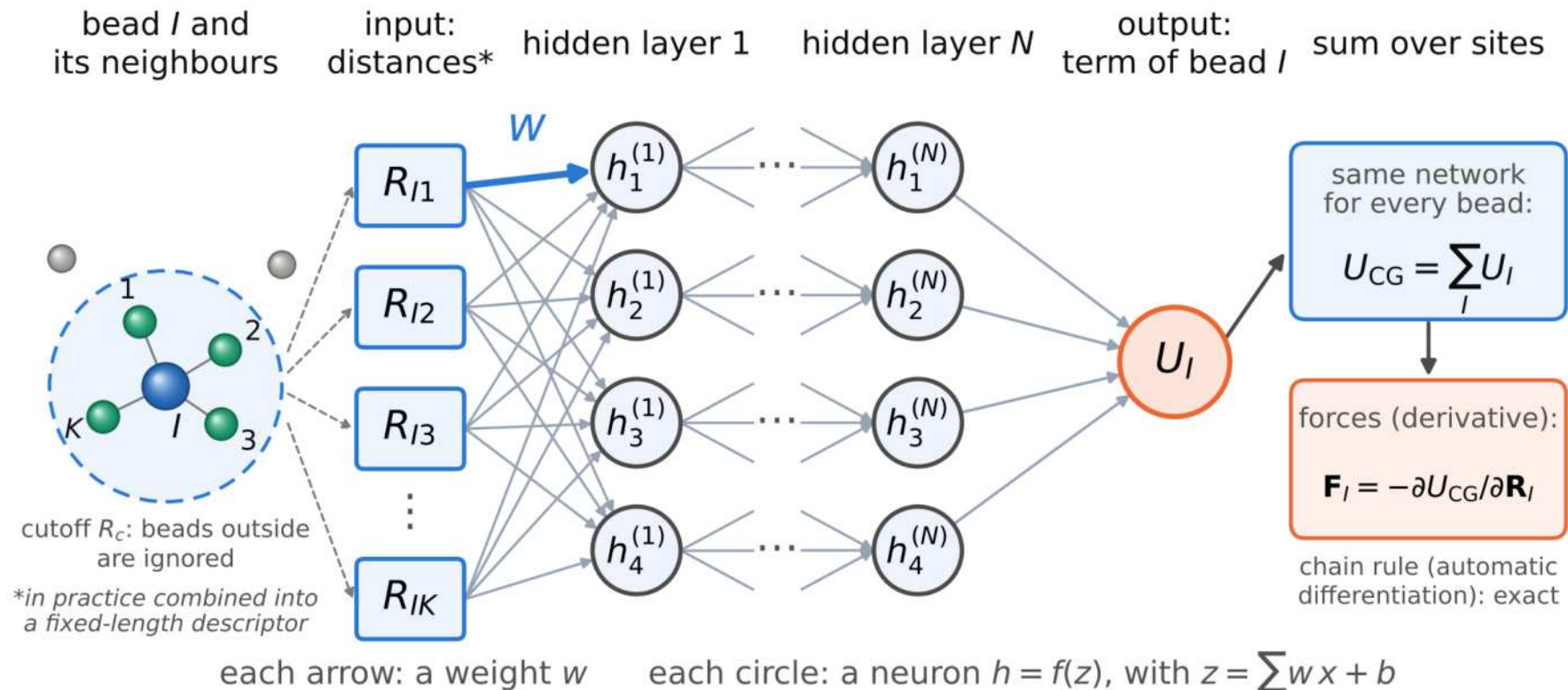
From atoms to beads: remove the solvent, keep the physics, learn the rest



Speed-up vs all-atom: fewer particles (no solvent, several atoms per bead) $\times$ longer time step (2 $\rightarrow$ 10–40 fs) $\times$ smoother landscape $\rightarrow$ **typically 10^2 – $10^4\times$ faster** (but CG time $\neq$ real time)

The CG potential is a function too

Each bead: distances to its neighbours → its term; the CG potential (a free energy) is their sum

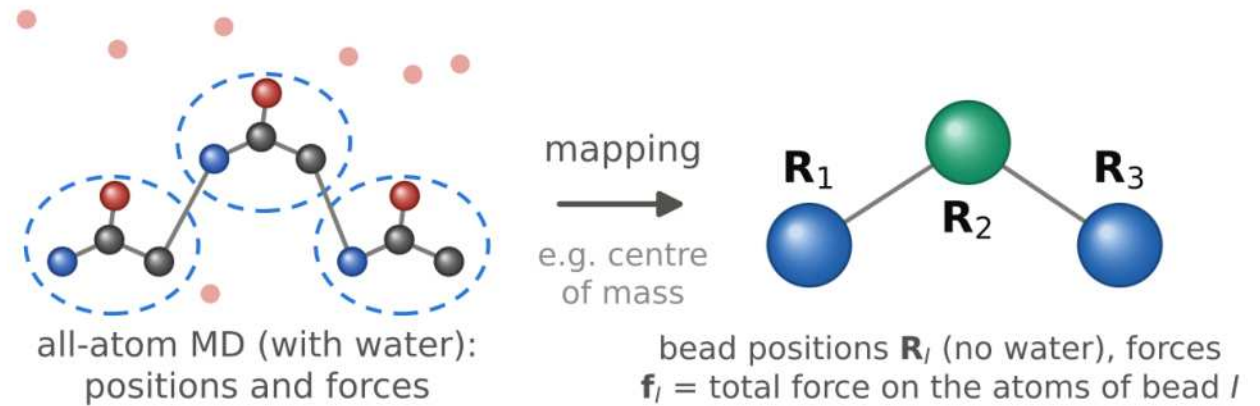


$U_I = f_{\theta}(R_{I1}, R_{I2}, \dots)$: **one network for every bead, θ fitted by force matching**

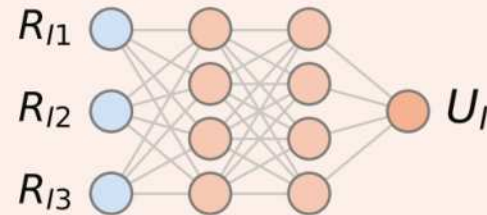
A neural-network CG potential

- Same idea one level up: a **function of the bead positions** $\mathbf{R}_1, \dots, \mathbf{R}_n$ (through the distances R_{IJ}), with parameters θ
- $U_{\text{CG}} = U_{\text{prior}} + U_{\text{NN}}$; its derivative is the **force on each bead**
- θ is fitted so that these forces match the all-atom forces mapped onto the beads (**force matching**)
- The result approximates the PMF: a **free energy**, it depends on T

1 Map each all-atom snapshot onto beads



2 Each bead: a function of its distances



same network for every bead i :

$$U_{\text{CG}} = U_{\text{prior}} + \sum_i U_i$$

forces on the beads: $\mathbf{F}_i = -\nabla_i U_{\text{CG}}$

3 Training: force matching

$$\min_{\theta} \left\langle \sum_i |\mathbf{F}_i(\theta) - \mathbf{f}_i|^2 \right\rangle_{\text{AA}}$$

$\Rightarrow U_{\text{CG}} \approx \text{PMF}$
(a free energy: depends on T)

Physics prior + neural network

The potentials of today become the prior; the network learns only the correction

$$U_{\text{CG}} = U_{\text{prior}} + U_{\text{NN}}$$

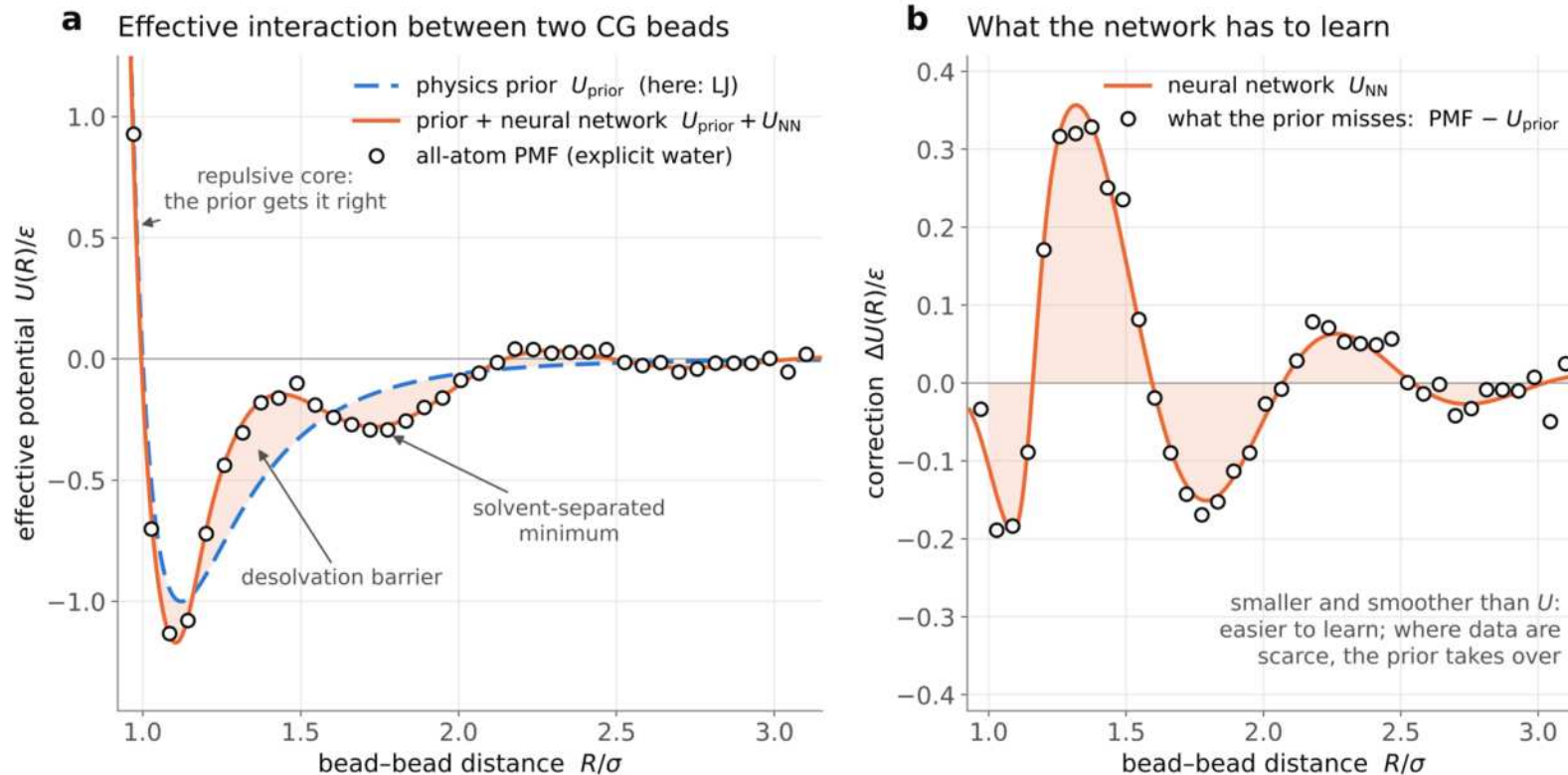


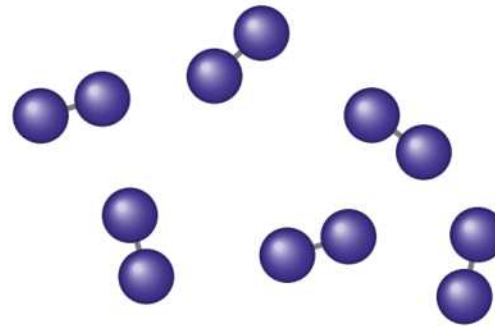
Illustration: in the real model U_{NN} is a many-body function of all bead positions (PaiNN), trained on all-atom forces.

- U_{prior} : physics we know: harmonic and FENE bonds, LJ / Morse contacts, Coulomb
- U_{NN} : what the prior misses (solvent, many-body effects)
- Trained by **force matching** on all-atom forces
- Where data are scarce, the prior keeps the model physical
- Research example: CG models of DNA G-quadruplexes with a PaiNN correction

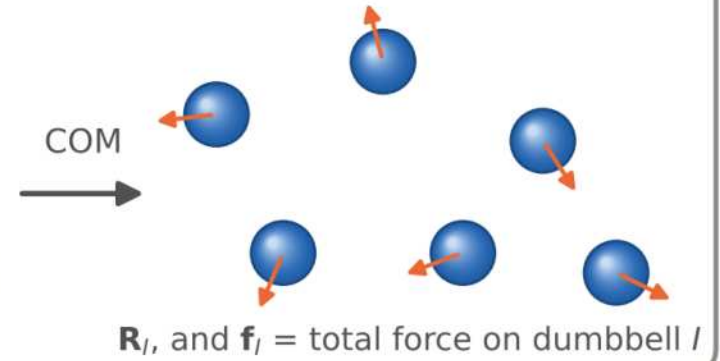
A possible final project: dumbbells → ML CG model

- Simulate a fluid of **LJ dumbbells** with your MC code (C++)
- Map each dumbbell to its **centre of mass**; store bead positions and total forces
- Train $U_{CG} = U_{prior} + U_{NN}$ in **PyTorch** by force matching; export the weights
- Read them in **your C++ code** (vector and matrix classes): MC needs only U , so just the forward pass. Compare with the dumbbells: $g(R)$, pressure, speed-up, transferability

1 MC of LJ dumbbells (C++)



2 COM mapping

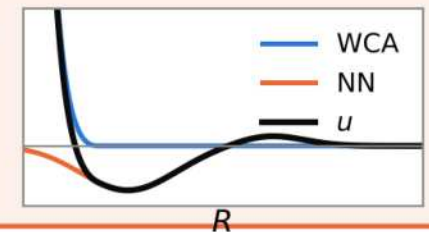


3 PyTorch: train the CG pair potential

$$U_{CG} = \sum_{i < j} u(R_{ij}), \quad u(R) = u_{WCA}(R) + u_{NN}(R; \theta)$$

network input: only the COM distance R_{ij}

force matching on the stored MC snapshots



4 back to C++: your vector and matrix classes

read the weights, $h = f(\mathbf{W}\mathbf{x} + \mathbf{b})$ layer by layer: MC needs only U_{CG}

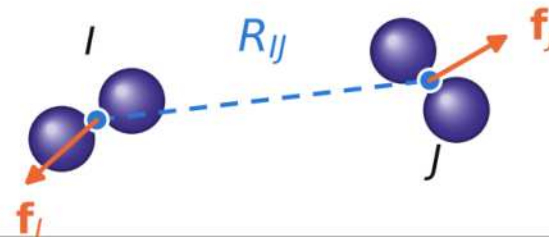
5 compare $g(R)$, pressure, speed-up vs dumbbells;

change density or T : is U_{CG} transferable?

Final project: what goes into the network

- Input: only the **distance** R_{IJ} between the centres of mass of two dumbbells: a single number
- The **same network** for every pair within R_c , then the sum: $U_{CG} = \sum u(R_{IJ})$
- Training data: $\mathbf{R}_I$ and $\mathbf{f}_I$ of every molecule in each saved snapshot
- Forces of the model by **autograd**, compared with $\mathbf{f}_I$ (force matching). Full details: final project XXII

1 From your MC: each saved snapshot

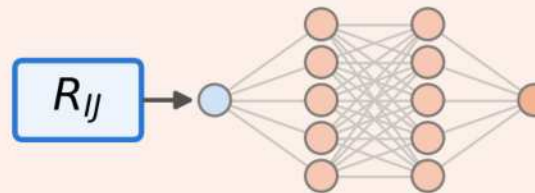


for every molecule I save:

$\mathbf{R}_I$ centre of mass

$\mathbf{f}_I$ total force from the other molecules

2 The network sees only the distance R_{IJ}



MLP $_{\theta}$: 1 $\rightarrow$ 32 $\rightarrow$ 32 $\rightarrow$ 1

$$u(R) = u_{\text{WCA}}(R) + s(R) \text{MLP}_{\theta}(R)$$

$s(R)$: smooth switch to 0 at R_c , e.g.

$$\frac{1}{2}[1 + \cos(\pi R/R_c)] \text{ or } \frac{1}{2}[1 + \tanh(\frac{R_s - R}{\delta})]$$

(tanh: cut at $R_c = R_s + 5\delta$); same network for every pair $I < J$ with $R_{IJ} < R_c$; no coordinates

3 Sum over pairs, forces, loss

$$U_{CG} = \sum_{I < J} u(R_{IJ}) \quad \rightarrow \quad \mathbf{F}_I = -\partial U_{CG} / \partial \mathbf{R}_I \quad (\text{autograd})$$

$$\text{loss} = \left\langle \sum_I |\mathbf{F}_I - \mathbf{f}_I|^2 \right\rangle_{\text{snapshots}}$$

$\rightarrow$ update θ (Adam)
never zero: orientations

Conclusions

1. First-principles calculations based on Density Functional Theory can be used to calculate quite accurately the parameters of a given interatomic potential
2. The choice of the potential is very important, it must capture the right physics of the interaction
 - Lennard Jones with noble gases is great, but don't use it for a binding molecule!
3. First-principles calculations correctly reproduce the interaction at the quantum level, it's up to us to model them correctly
4. We're not restricted to DFT for the quantum mechanical part (e.g. QMC, Coupled Cluster)
5. Machine-learned potentials (PaiNN, MACE, ...) learn the interaction from DFT data without fixing a functional form; a physics prior plus a neural network brings the same idea to coarse-grained models