� · Symmetry-Preserving Neural Networks

"A molecule does not concern itself with the orientation of the coordinate axes."

1 Symmetry in physical systems

A molecule's energy doesn't change if you rotate it or translate it through space. These are *symmetries*: transformations that leave physical quantities unchanged. In the previous chapter, we saw how CNNs encode translation symmetry and GNNs encode permutation symmetry. Physical systems often have additional continuous symmetries—rotations, translations, reflections—that we can build into neural network architectures.

2 Invariance and equivariance

When we apply a symmetry transformation to the input, the output can respond in two fundamentally different ways.

Invariance. An *invariant* function produces the same output regardless of how the input is transformed:

$$f(g \cdot x) = f(x) \tag{2.1}$$

Energy is the canonical example. Rotate a molecule by any angle, translate it anywhere in space—the energy stays exactly the same. The function "doesn't see" the transformation.

Equivariance. An *equivariant* function has outputs that transform consistently with the inputs:

$$f(g \cdot x) = g \cdot f(x) \tag{2.2}$$

Forces are equivariant. Rotate a molecule, and the force vectors rotate with it. The force on atom i still points in the same direction *relative to the molecule*—but in the lab frame, it has rotated along with everything else. The function "sees" the transformation and responds accordingly.

The distinction matters for what you're predicting. Scalar quantities—energy, charge, mass, binding affinity—should be invariant. Vector quantities—forces, velocities, dipole moments—should be equivariant. Per-atom outputs that describe spatial relationships (like predicted displacements) should be equivariant.
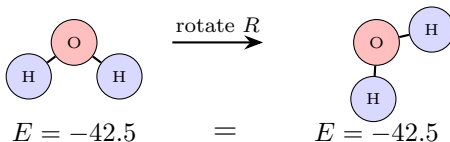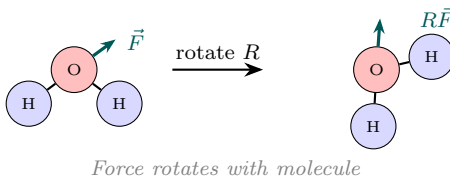
Invariance (Energy)

Equivariance (Force)

Force rotates with molecule

Figure 1. Invariance vs. equivariance. (Top) Energy is invariant: rotating the molecule doesn't change its energy. (Bottom) Force is equivariant: rotating the molecule rotates the force vectors.

3   Building in translation invariance

Translation invariance is the easiest symmetry to encode: **never use absolute positions**.

   If your input is atom positions $\{r_i\}$, don't feed raw coordinates into the network. Instead, use quantities that are intrinsically translation-invariant: *relative positions* $r_{ij} = r_i - r_j$ between pairs of atoms, or *distances* $d_{ij} = \|r_i - r_j\|$. Shift all atoms by the same vector, and these quantities don't change—the relative geometry is preserved.

   This simple prescription eliminates an entire class of failure modes. Networks trained on absolute positions can learn arbitrary correlations with the coordinate system, e.g. "molecules centered at the origin have property X.". These patterns may hold in the training set (if all molecules happen to be centered) but reflect no physics. By using only relative quantities, we make such spurious patterns impossible to learn.

4   Invariant networks: the simplest approach

The simplest way to achieve rotation invariance: **only use distances**. Distances $d_{ij} = \|x_i - x_j\|$ don't change under rotation. If your network only sees distances (never positions or relative vectors), it's automatically invariant.

SchNet-style architecture.   Recall message passing from the previous chapter: each node i in a graph updates its features by aggregating information from neighbors $j \in \mathcal{N}(i)$. SchNet [1] makes this rotation-invariant by having messages depend only on distances:

$$h_i' = h_i + \sum_{j\in\mathcal{N}(i)} \phi(h_j) \cdot w(d_{ij}) \tag{4.1}$$

Each neighbor j sends a message $\phi(h_j)$ weighted by a learned function of distance $w(d_{ij})$. The network never sees coordinates directly—only pairwise distances. This guarantees rotation and translation invariance.
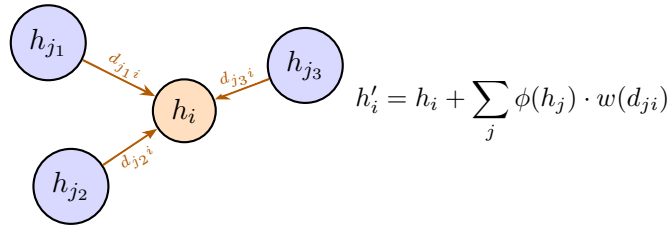

Figure 2. SchNet-style invariant message passing. Each neighbor sends a message weighted by a learned function of distance $w(d_{ji})$. The network only sees distances, never positions, ensuring rotation and translation invariance.

Encoding distances.   How should $w(d_{ij})$ depend on distance? A neural network needs a good representation of the scalar d_{ij}. The standard approach: *radial basis functions*.

Expand the distance into a set of basis functions, typically Gaussians centered at different values:

$$e_k(d) = \exp\left(-\gamma(d - \mu_k)^2\right) \tag{4.2}$$

with centers μ_k spaced from 0 to some cutoff (e.g., $\mu_k = 0, 0.5, 1.0, \ldots, 5.0$ Å). This gives a vector $[e_1(d), e_2(d), \ldots, e_K(d)]$ that the network can process with standard linear layers.

Why not just use d directly? A single scalar gives the network little to work with. The basis expansion provides a richer representation: each basis function "activates" for distances near its center, giving the network easy access to distance information at different scales.
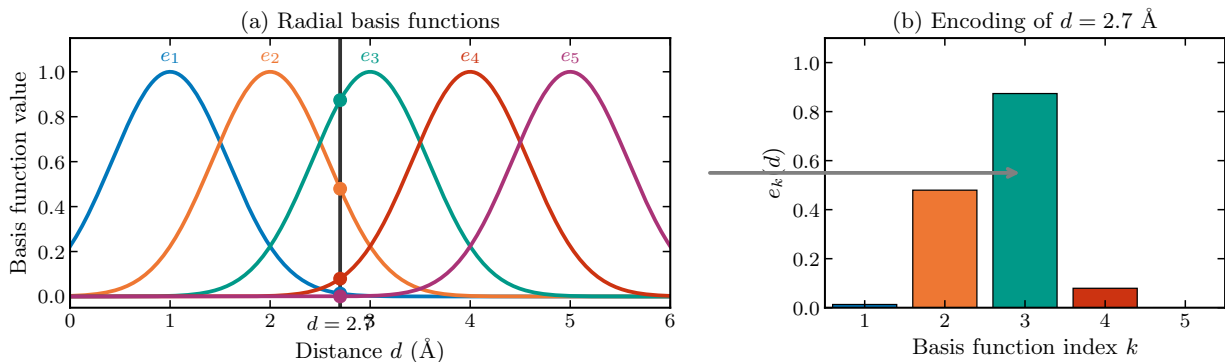

Figure 3. Radial basis function encoding. (a) Five Gaussians centered at different distances. A vertical line marks an example distance $d = 2.7$ Å; dots show where it intersects each basis function. (b) The resulting encoding: a 5-dimensional vector where each entry is one basis function's activation. One scalar (distance) becomes a vector the network can process.

Cutoff functions. Atoms far apart don't interact much. For efficiency (and physics), we restrict to neighbors within a cutoff radius r_c:

$$\tilde{w}(d) = w(d) \cdot f_{\text{cut}}(d) \tag{4.3}$$

where $f_{\text{cut}}(d)$ smoothly goes to zero as $d \to r_c$. A common choice:

$$f_{\text{cut}}(d) = \begin{cases} \frac{1}{2}\left[\cos\left(\frac{\pi d}{r_c}\right) + 1\right] & d < r_c \\ 0 & d \geq r_c \end{cases} \tag{4.4}$$

Smooth cutoffs matter: discontinuities in the energy would cause infinite forces.

5 Equivariant networks: when you need vectors

Invariant networks predict scalars. But what if you need *vector* outputs—forces, velocities, dipole moments? These should rotate with the input, and in this case we need to encode *equivariance*.

Scalar and vector features. The key idea is to maintain two types of features at each node: *scalar features* $h_i \in \mathbb{R}^d$ that are invariant to rotation, and *vector features* $\vec{v}_i \in \mathbb{R}^{3 \times d}$ that rotate with the input.

How can we combine these without breaking equivariance? Scalars are flexible—you can add them, multiply them, pass them through any nonlinearity you like. Vectors are more constrained: you can add vectors and scale them by scalars, but you *cannot* pass them through elementwise nonlinearities like ReLU. Applying $\text{ReLU}(v_x, v_y, v_z)$ component-wise would break rotation equivariance—the result depends on how the coordinate axes are oriented.

The allowed operations form an algebra: inner products of vectors produce scalars (the dot product is rotation-invariant), and scalars times vectors produce vectors (scaling doesn't break equivariance). These constraints may seem limiting, but they're enough to build powerful architectures.

The EGNN approach. EGNN [2] is a simple and effective equivariant architecture. Each node has an embedding $h_i \in \mathbb{R}^d$ and a coordinate $x_i \in \mathbb{R}^3$. The layer updates both:

$$m_{ij} = \phi_e(h_i, h_j, \|x_i - x_j\|^2, a_{ij}) \qquad \text{(scalar edge messages)} \tag{5.1}$$

$$x_i' = x_i + \sum_{j \neq i}(x_i - x_j)\,\phi_x(m_{ij}) \qquad \text{(coordinate update)} \tag{5.2}$$

$$h_i' = \phi_h\left(h_i, \sum_j m_{ij}\right) \qquad \text{(feature update)} \tag{5.3}$$

Why is this equivariant? The construction is careful. Edge messages m_{ij} depend only on *squared distances* $\|x_i - x_j\|^2$, which are rotation-invariant—no directional information leaks in. The coordinate update uses *relative positions* $(x_i - x_j)$, which are vectors that rotate with the input: if you rotate all coordinates by R, these vectors rotate by R too. The update direction is then scaled by a learned *scalar* $\phi_x(m_{ij})$—and scalar times vector gives a vector that rotates correctly. Rotate all input coordinates by a matrix R, and all output coordinates rotate by the same R. The network is E(3)-equivariant by construction.

Handling velocities.   For dynamical systems, particles have velocities in addition to positions. Velocities are vectors—they rotate with the coordinate system just like positions. The original EGNN paper [2] introduces a velocity-aware variant for learning dynamics. The key modification replaces the coordinate update with:

$$x_i' = x_i + v_i \cdot \phi_v(h_i) + \sum_{j \neq i}(x_i - x_j)\,\phi_x(m_{ij}) \tag{5.4}$$

where ϕ_v is a learned scalar function of the node features. The velocity vector v_i contributes to the position update, scaled by a learned factor—this maintains equivariance because scalar times vector transforms correctly. The velocity itself can be updated similarly, using the same equivariant pattern as coordinates. This formulation lets EGNN learn particle dynamics (like charged particles interacting via Coulomb forces) while respecting rotational symmetry.
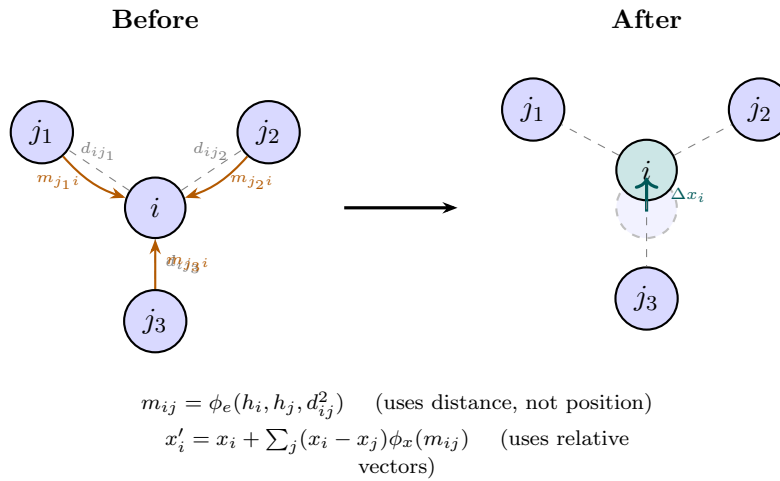

$$m_{ij} = \phi_e(h_i, h_j, d_{ij}^2) \quad \text{(uses distance, not position)}$$
$$x_i' = x_i + \sum_j (x_i - x_j)\phi_x(m_{ij}) \quad \text{(uses relative vectors)}$$

Figure 4. EGNN message passing. Scalar messages depend only on distances (rotation-invariant). Coordinate updates use relative position vectors $(x_i - x_j)$, which rotate with the input. The combination ensures the network is E(3)-equivariant.

Beyond vectors: spherical harmonics.— Scalars and vectors aren't the only geometric objects. Think about what happens when you rotate the coordinate system:

- A **scalar** (like energy) doesn't change at all.

- A **vector** (like a force) rotates with the coordinates—its three components mix together according to the rotation matrix R.

- A **quadrupole** (like the moment of inertia tensor) transforms in a more complicated way—it has five independent components that mix under rotation.

This pattern continues: there's a hierarchy of geometric objects labeled by $\ell = 0, 1, 2, \ldots$, with $2\ell + 1$ components each. These are the *irreducible representations* of the rotation group.

> **Spherical harmonics** $Y_\ell^m(\hat{r})$ provide a concrete basis for each level. You may know them from quantum mechanics: $\ell = 0$ is an s-orbital (spherical), $\ell = 1$ gives p-orbitals (three lobes along x, y, z), $\ell = 2$ gives d-orbitals (five cloverleaf patterns), and so on. They form a complete basis for any angular pattern on a sphere.
>
> More advanced equivariant networks—NequIP [3], MACE [4], Allegro [5]—represent features as collections of spherical harmonic components at each atom. The key operation is the *tensor product*: combining two spherical harmonic features (say, $\ell = 1$ and $\ell = 1$) produces features at multiple orders ($\ell = 0, 1, 2$). The coefficients governing this combination are called *Clebsch-Gordan coefficients*—fixed numbers from representation theory, not learned parameters.
>
> The payoff is these networks capture directional interactions that scalar/vector architectures miss, like the angular dependence of chemical bonds or the anisotropy of crystal environments. The cost: mathematical and computational complexity. For a comprehensive introduction, see [6].

6   Forces from energy: equivariance for free

Suppose you train an invariant network to predict energy $E(\{x_i\})$. The forces are gradients:

$$F_i = -\nabla_{x_i} E \tag{6.1}$$

If E is rotation-invariant, then F_i is automatically rotation-equivariant.

Working through it: Let R be a rotation matrix. Invariance means $E(R\{x_i\}) = E(\{x_i\})$. Differentiate both sides with respect to x_i:

$$R^\top \nabla_{Rx_i} E = \nabla_{x_i} E \tag{6.2}$$

So $\nabla_{Rx_i} E = R\nabla_{x_i} E$. The gradient at the rotated position equals the rotated gradient at the original position. That's equivariance.

The practical payof is that we can train a simple invariant network on energies and get forces for free by automatic differentiation. The forces are then exactly equivariant. This is how most modern machine learning potentials work: learn E, differentiate to get F, run molecular dynamics. More generally: **symmetries of a function imply symmetries of its derivatives**. Build invariance into your energy predictor, and equivariance of forces follows automatically.

7   Why encode symmetry?

Why go through the trouble of building symmetry into architectures? After all, neural networks are universal approximators—with enough data, shouldn't they learn any symmetry implicitly? The advantage of encoding symmetry is largest with smaller datasets (less data to learn the symmetry implicitly), larger symmetry groups (more transformations to cover by augmentation), and stricter symmetry requirements (when approximate invariance isn't good enough—e.g., forces must be exactly equivariant for stable molecular dynamics).

Data augmentation: the alternative. Instead of encoding symmetry in the architecture, we can *augment* the training data: for each molecule, include rotated copies. The network sees all orientations and (hopefully) learns that orientation doesn't matter.

This works, but has costs—training on N rotations multiplies compute by N. The network learns *approximate* invariance—close to exact, but not perfect.

Data efficiency. An equivariant network doesn't need to see every rotation of every molecule—it *knows* that rotations preserve energy. A standard network must learn this from augmented data, requiring many more examples. When labeled data is scarce (common in science, where each label may require expensive simulations or experiments), equivariance can be the difference between a useful model and an overfit one.

Figure 5 illustrated this: equivariant networks (using higher-order spherical harmonic features, $L \geq 1$) achieve lower error at every training set size, with steeper improvement as data increases (more favorable scaling).
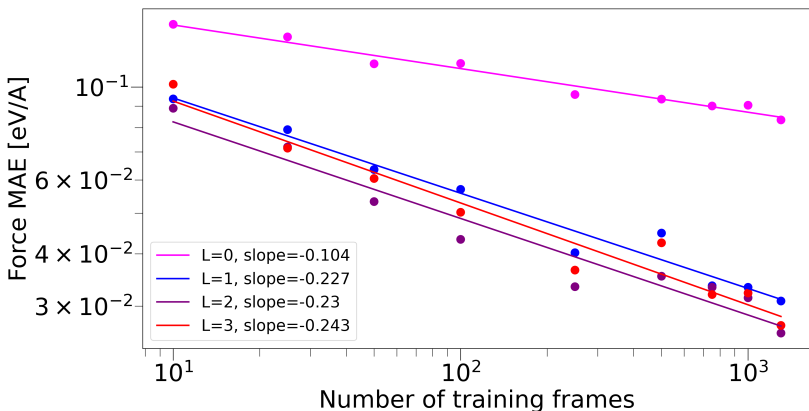

Figure 5. Data efficiency of equivariant networks. Force prediction error vs. training set size for water molecules. $L = 0$ uses only scalars (invariant); $L \geq 1$ includes vector and higher-order equivariant features. Equivariant networks achieve lower error at all training set sizes. Figure from [3].

Can augmentation catch up? With enough epochs, yes—but at a cost. Recent scaling studies [7] show that training non-equivariant networks with data augmentation can eventually match equivariant performance, but only when training for many more epochs on the same data. In the small-data regime where you'd train for thousands of epochs anyway, augmentation closes the gap. In the large-data regime where each sample is seen only once or a few times, equivariant networks maintain their advantage.

Compute efficiency. Even with infinite data, equivariant networks remain more efficient. At any fixed compute budget—measured in floating-point operations—equivariant models outperform non-equivariant ones trained with augmentation (Figure 6). Both model classes follow power-law scaling: double your compute, and the loss decreases by a predictable factor. But equivariant models maintain a consistent advantage across all tested compute budgets—roughly a factor of two in this benchmark. The symmetry constraint focuses learning capacity on the

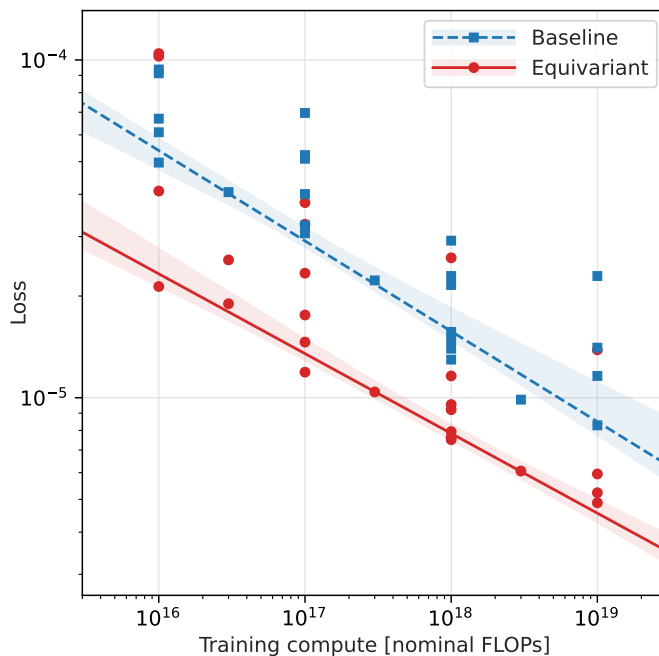degrees of freedom that matter, rather than wasting capacity re-learning that rotations preserve energy.

Figure 6. Compute scaling of equivariant vs. non-equivariant transformers on a rigid-body dynamics task. Both follow power-law scaling, but the equivariant model (red) consistently outperforms the non-equivariant baseline (blue) at every compute budget. The advantage persists even with large-scale training. Figure from [7].

When augmentation wins. If the symmetry is only approximate, strict equivariance may be too strong. Images have gravity—up and down aren't equivalent. Molecules in solvent or on surfaces break full rotation symmetry. In these cases, learning an approximate symmetry from data may be more appropriate than enforcing an exact one. Augmentation also applies to any architecture without redesigning the model, which matters when leveraging pretrained models or established codebases.

Equivariance as a building block. The architectures in this chapter are not standalone models but composable components. Diffusion models for molecules use equivariant score networks to ensure the learned distribution respects rotational symmetry. Machine learning potentials use equivariant networks to predict forces that transform correctly—critical for conserving angular momentum in long simulations. The design principles transfer beyond molecules to any domain with geometric structure: point clouds, meshes, robotics, physical simulations.

References

[1] Kristof T Schütt, Pieter-Jan Kindermans, Huziel E Sauceda, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. SchNet: A continuous-filter convolutional neural network for modeling

quantum interactions. In *Advances in Neural Information Processing Systems*, volume 30, pages 992–1002, 2017.

[2] Víctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) equivariant graph neural networks. In *International Conference on Machine Learning*, pages 9323–9332. PMLR, 2021.

[3] Simon Batzner, Albert Musaelian, Lixin Sun, Mario Geiger, Jonathan P Mailoa, Mordechai Kornbluth, Nicola Molinari, Tess E Smidt, and Boris Kozinsky. E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature Communications*, 13(1):2453, 2022.

[4] Ilyes Batatia, Dávid Péter Kovács, Gregor NC Simm, Christoph Ortner, and Gábor Csányi. MACE: Higher order equivariant message passing neural networks for fast and accurate force fields. In *Advances in Neural Information Processing Systems*, volume 35, pages 11423–11436, 2022.

[5] Albert Musaelian, Simon Batzner, Anders Johansson, Lixin Sun, Cameron J Owen, Mordechai Kornbluth, and Boris Kozinsky. Learning local equivariant representations for large-scale atomistic dynamics. *Nature Communications*, 14(1):579, 2023.

[6] Alexandre Duval, Simon V Mathis, Chaitanya K Joshi, Victor Schmidt, Santiago Miret, Fragkiskos D Malliaros, Taco Cohen, Pietro Liò, Yoshua Bengio, and Michael Bronstein. A hitchhiker's guide to geometric GNNs for 3D atomic systems. *arXiv preprint arXiv:2312.07511*, 2023.

[7] Johann Brehmer, Sönke Behrends, Pim de Haan, and Taco Cohen. Does equivariance matter at scale? *Transactions on Machine Learning Research*, 2025.