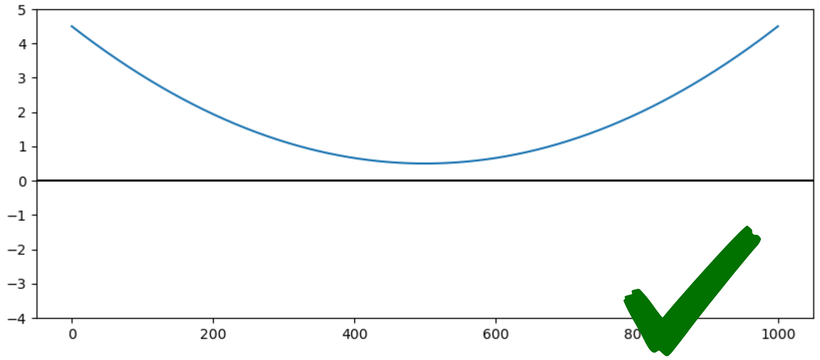
Learning Polynomial Problems with $SL(2,\mathbb{R})$-Equivariance

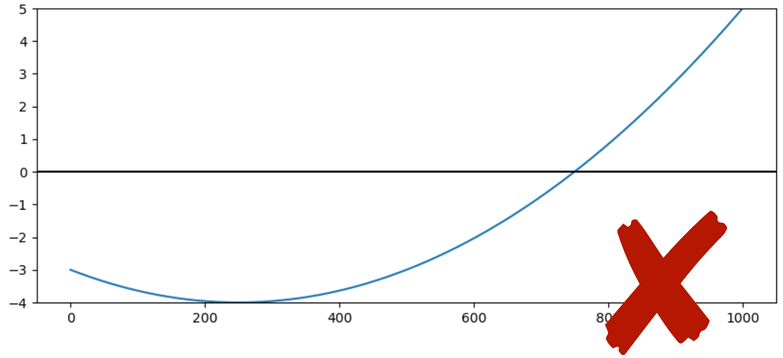
Hannah Lawrence*
MIT

Mitchell Harris*
MIT

Polynomial Positivity is Useful

Fundamental problem: is $p(\vec{x}) \geq 0$ for all $x \in \mathcal{X}$?





If so, how do you **prove** it?

Applications: robot path planning, dynamical system stability, designing nonlinear controllers

Positivity Verification is Useful

Fundamental problem**:** is $p(x, y) \geq 0$ for all x, y, **provably**?

Def. The **d-lift** $\vec{x}^{[d]} = \begin{bmatrix} y^d & xy^{d-1} & \cdots & x^d \end{bmatrix}^T$

Def. A matrix M **represents** $p(x, y)$ if $p(\vec{x}) = p(x, y) = \vec{x}^{[d]^T} M \vec{x}^{[d]}$

$$\begin{bmatrix} y^d & xy^{d-1} & \cdots & x^d \end{bmatrix} \begin{bmatrix} M_{00} & M_{01} & \cdots \\ M_{10} & M_{11} & \cdots \\ \vdots & & \\ M_{d0} & M_{d1} & \cdots \end{bmatrix} \begin{bmatrix} y^d \\ xy^{d-1} \\ \vdots \\ x^d \end{bmatrix}$$

In fact, this is a way of writing $p(\vec{x})$ as a sum of squares, $p(\vec{x}) = \sum_i q_i(\vec{x})^2$

If Q represents $p(x, y)$ and $Q \succeq 0$, then $p(x, y) \geq 0$ for all x, y by definition

Positivity Verification is Useful

Fundamental problem**:** is $p(x, y) \geq 0$ for all x, y, **provably**?

Def. The **d-lift** $\vec{x}^{[d]} = \begin{bmatrix} y^d & xy^{d-1} & \cdots & x^d \end{bmatrix}^T$

Def. A matrix M **represents** $p(x, y)$ if $p(\vec{x}) = p(x, y) = \vec{x}^{[d]^T} M \vec{x}^{[d]}$ ← constraint on antidiagonal sums of M

$$\begin{bmatrix} y^d & xy^{d-1} & \cdots & x^d \end{bmatrix} \begin{bmatrix} M_{00} & M_{01} & \cdots \\ M_{10} & M_{11} & \cdots \\ \vdots & & \\ M_{d0} & M_{d1} & \cdots \end{bmatrix} \begin{bmatrix} y^d \\ xy^{d-1} \\ \vdots \\ x^d \end{bmatrix}$$

If Q represents $p(x, y)$ and $Q \succeq 0$, then $p(x, y) \geq 0$ for all x, y by definition

Max-Determinant via SDPs

For a given positive p, many M can represent it!

Convenient choice:

$$f(p) = \text{argmax} \log\det Q \text{ such that } p(\vec{x}) = \vec{x}^{[d]^T} Q \vec{x}^{[d]} \text{ and } Q \succeq 0$$

Why?

$\rightarrow$ analytic center of the feasible region (avoiding boundary)

$\rightarrow$ has a unique solution if p is positive

Solve via convex optimization

What's wrong with SDPs?

For a given positive p, many M can represent it!

Convenient choice:

$$f(p) = \text{argmax} \log\det Q \text{ such that } p(\vec{x}) = \vec{x}^{[d]^T} Q \vec{x}^{[d]} \text{ and } Q \succeq 0$$

Why?

$\rightarrow$ analytic center of the feasible region (avoiding boundary)

$\rightarrow$ has a unique solution if p is positive

They are slow + scale poorly with degree!

What's wrong with SDPs?

For a given positive p, many M can represent it!

Convenient choice:

$$f(p) = \text{argmax} \log\det Q \text{ such that } p(\vec{x}) = \vec{x}^{[d]^T} Q \vec{x}^{[d]} \text{ and } Q \succeq 0$$

Why?

→ analytic center of the feasible region (avoiding boundary)

→ has a unique solution if p is positive

Possible acceleration: machine learning!

Machine Learning for SDPs

1. Generate many random problem instances, and solve them with a classical solver

2. Use these solved instances as **training data** for a neural network

Why could ML help?

Might seem like a lot to hope for: positivity verification is NP hard!

- Can be reduced to problems in combinatorial optimization

However, can still hope for solving only on a specific data distribution!

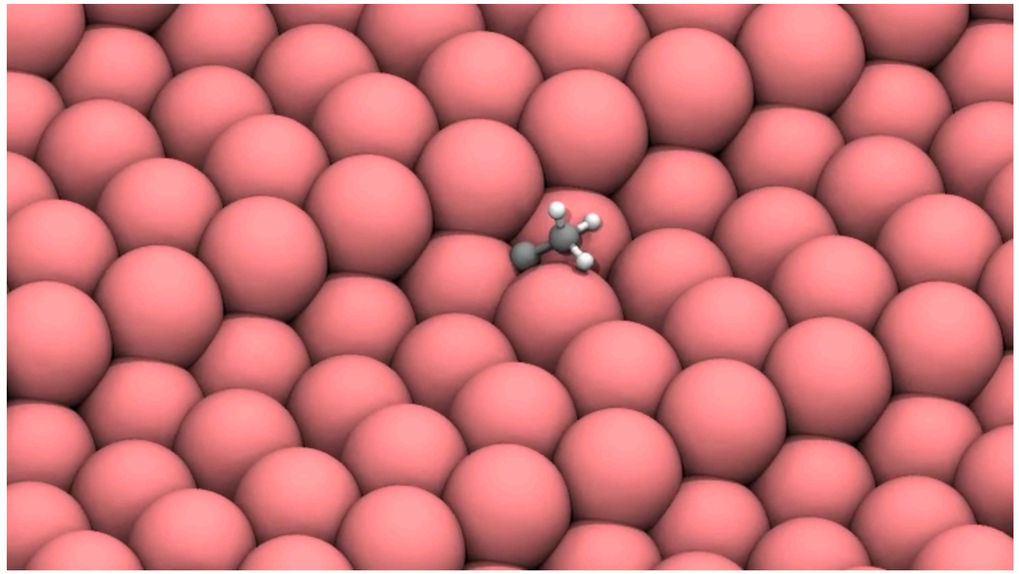
- For example, pure square polynomials $p(x) = q(x)^2$ are easier
- We don't explicitly specify the form of such a distribution, but use it as high-level intuition

Why could ML help?

Track record of ML accelerating slow computation:

- Accelerating density functional theory calculations for understanding dynamics of possible catalysts, for example

Train on expensive computations



Open Catalyst Project

Then use trained model instead of expensive computations on new data!

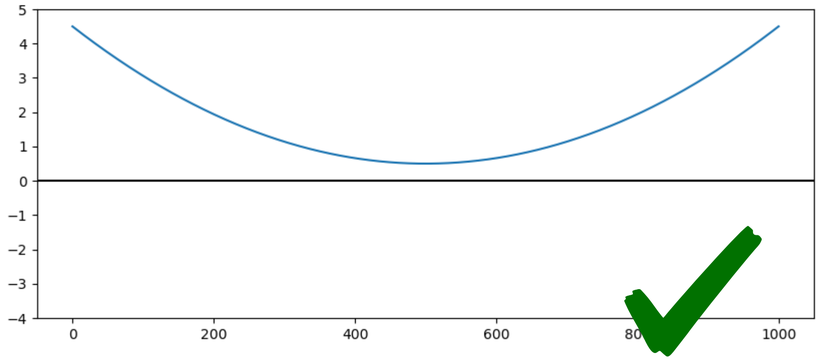
Also: can verify a proposed catalyst with regular DFT

Why could ML help?

Track record of ML accelerating slow computation:

- Accelerating **positivity verification** for **e.g. robot path planning, dynamical system stability, etc.**

Train on expensive computations from convex solvers



Then use trained model instead of expensive computations on new data!

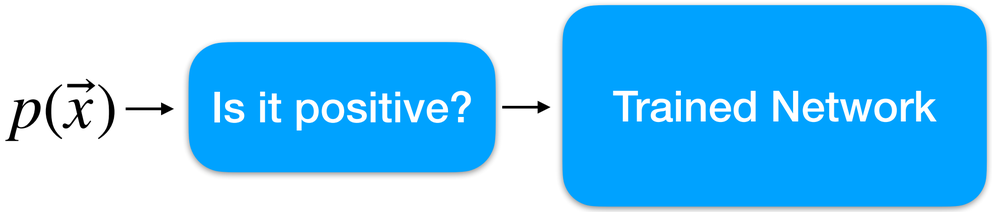
Why could ML help?

$p(\vec{x})$

Why could ML help?

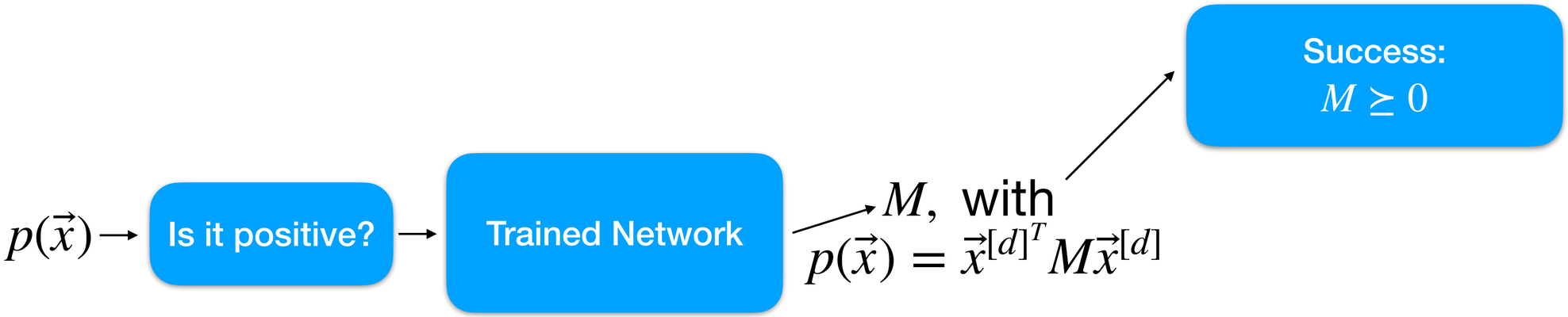
$p(\vec{x})$ → Is it positive?

Why could ML help?

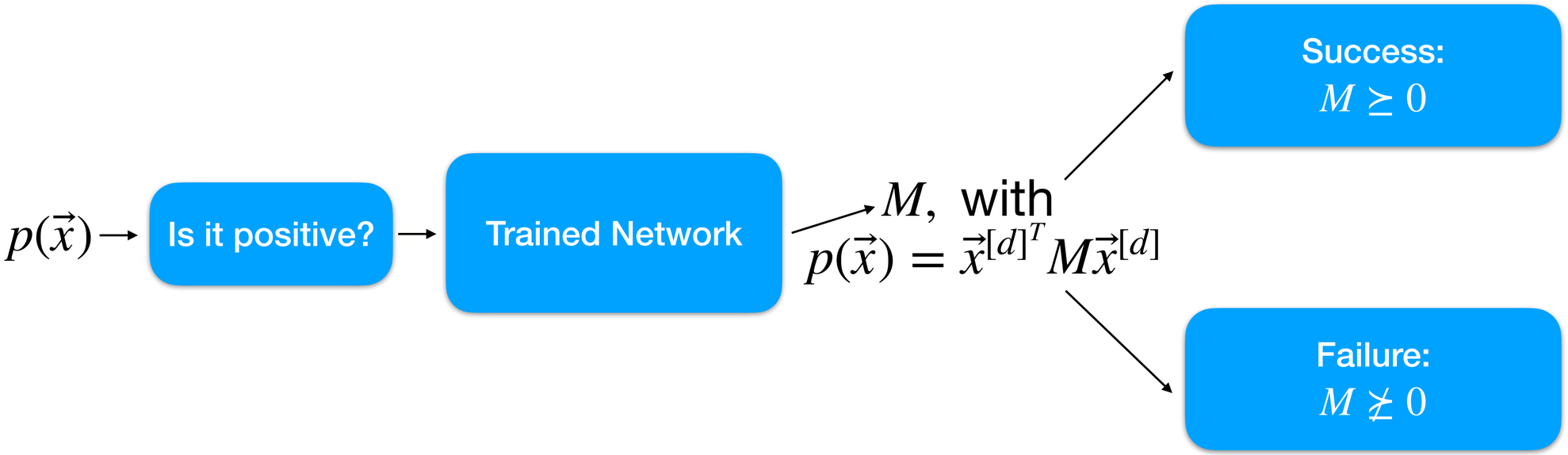


Why could ML help?

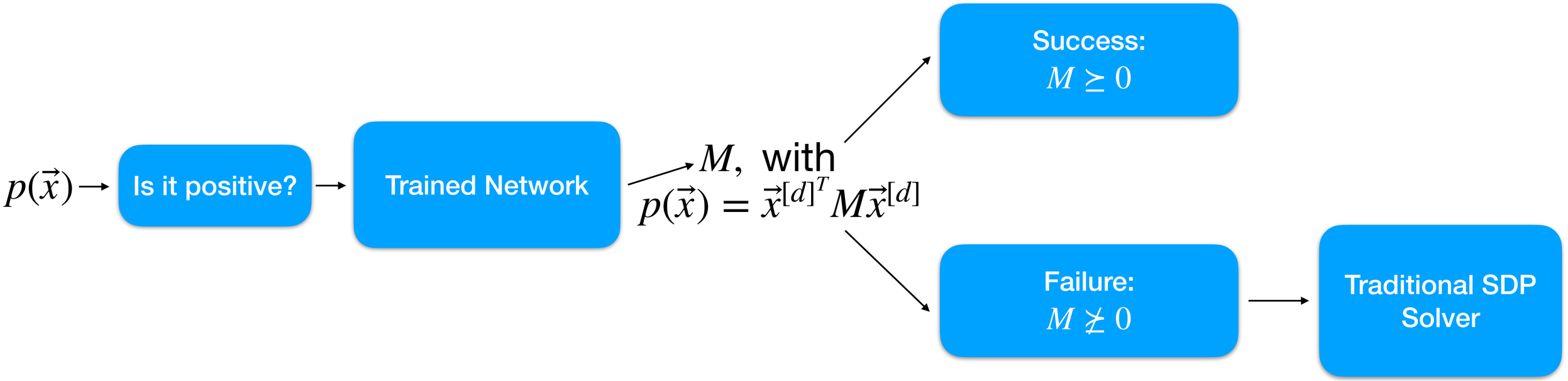
Why could ML help?



Why could ML help?



Why could ML help?

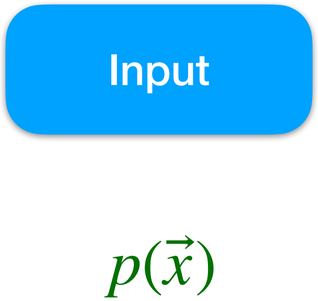


Can check your machine-learned answer!

What *kind* of ML should we try?

What *kind* of ML should we try?

Observe: the problem exhibits **symmetry**

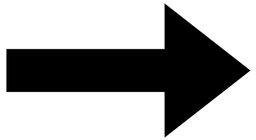


What *kind* of ML should we try?

Observe: the problem exhibits **symmetry**

$\vec{x} = (x, y)$

Input



Output

$p(\vec{x})$

Q of maximal $\log\det Q$ with

$$p(\vec{x}) = \vec{x}^{[d]^T} Q \vec{x}^{[d]}, Q \succeq 0$$

What *kind* of ML should we try?

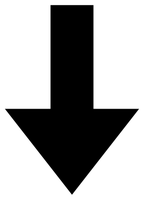
Observe: the problem exhibits **symmetry**

$\vec{x} = (x, y)$

$p(\vec{x})$

Q of maximal $\log\det Q$ with

$$p(\vec{x}) = \vec{x}^{[d]^T} Q \vec{x}^{[d]},\ Q \succeq 0$$



$p(A\vec{x})$

$A \in \mathbb{R}^{2\times 2}$

What *kind* of ML should we try?

Observe: the problem exhibits **symmetry**

$\vec{x} = (x, y)$

Input → Output

$p(\vec{x})$

Q of maximal $\log\det Q$ with

$$p(\vec{x}) = \vec{x}^{[d]^T} Q \vec{x}^{[d]},\ Q \succeq 0$$

$p(A\vec{x})$

$A \in \mathbb{R}^{2\times 2}$

$A^{[d]^T} Q A^{[d]}$ then of maximal $\log\det$

$$p(A\vec{x}) = \vec{x}^{[d]^T} A^{[d]^T} Q A^{[d]} \vec{x}^{[d]},$$

$$A^{[d]^T} Q A^{[d]} \succeq 0$$

What *kind* of ML should we try?

Observe: the problem exhibits **symmetry**

$\vec{x} = (x, y)$

Input → Output

$p(\vec{x})$

Q of maximal $\log\det Q$ with

$$p(\vec{x}) = \vec{x}^{[d]^T} Q \vec{x}^{[d]},\ Q \succeq 0$$

Here, $A^{[d]}$ is defined by $(A\vec{x})^{[d]} = A^{[d]}\vec{x}^{[d]}$

$p(A\vec{x})$

$A \in \mathbb{R}^{2\times 2}$

$A^{[d]^T} Q A^{[d]}$ then of maximal $\log\det$

$$p(A\vec{x}) = \vec{x}^{[d]^T} A^{[d]^T} Q A^{[d]} \vec{x}^{[d]},$$

$$A^{[d]^T} Q A^{[d]} \succeq 0$$

What *kind* of ML should we try?

Observe: the problem exhibits **symmetry**

$\vec{x} = (x, y)$

Input → Output

$p(\vec{x})$

Q solving a particular optimization problem

Change of variables in polynomial

$p(A\vec{x})$

$A \in \mathbb{R}^{2\times 2}$

Some linear transformation of Q

Special Linear Group: $SL(2,\mathbb{R})$

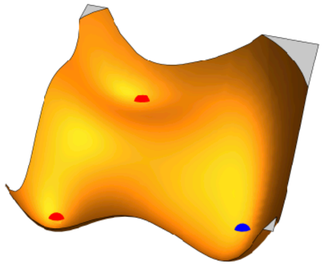
$SL(2,\mathbb{R}) = \{A \in \mathbb{R}^{2\times 2} \ : \ det(A) = 1\}$ consists of **area-preserving linear transformations**

Example: acts on bivariate polynomials as $p\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) \mapsto p\left(A \begin{bmatrix} x \\ y \end{bmatrix}\right)$

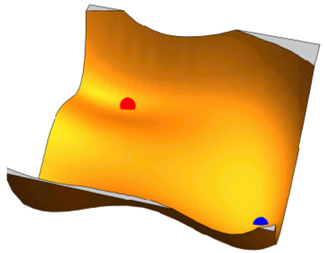
Special Linear Group: $SL(2,\mathbb{R})$

$SL(2,\mathbb{R}) = \{A \in \mathbb{R}^{2\times 2} \ : \ det(A) = 1\}$ consists of **area-preserving linear transformations**

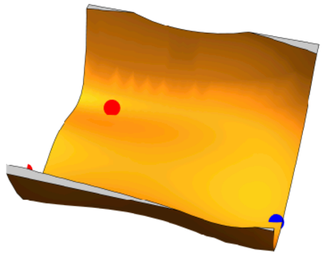
Example: acts on bivariate polynomials as $p\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) \mapsto p\left(A \begin{bmatrix} x \\ y \end{bmatrix}\right)$



(a) $g = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$



(b) $g = \begin{pmatrix} 1.2 & 0 \\ 0 & \frac{1}{1.2} \end{pmatrix}$



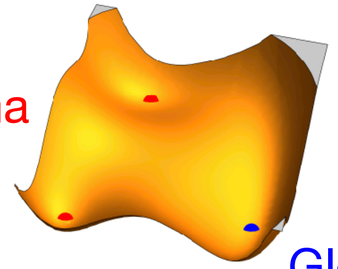
(c) $g = \begin{pmatrix} 1.4 & 0 \\ 0 & \frac{1}{1.4} \end{pmatrix}$

Special Linear Group: $SL(2,\mathbb{R})$

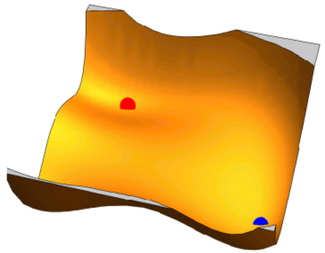
$SL(2,\mathbb{R}) = \{A \in \mathbb{R}^{2\times 2} \ : \ det(A) = 1\}$ consists of **area-preserving linear transformations**

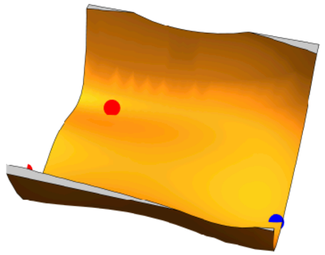
Example: acts on bivariate polynomials as $p\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) \mapsto p\left(A \begin{bmatrix} x \\ y \end{bmatrix}\right)$

Local minima



Global minimum





(a) $g = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

(b) $g = \begin{pmatrix} 1.2 & 0 \\ 0 & \frac{1}{1.2} \end{pmatrix}$

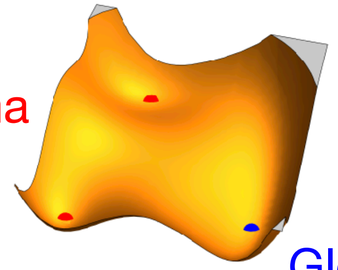
(c) $g = \begin{pmatrix} 1.4 & 0 \\ 0 & \frac{1}{1.4} \end{pmatrix}$

Special Linear Group: $SL(2,\mathbb{R})$

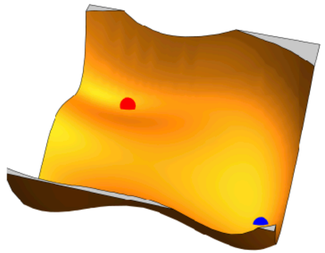
$SL(2,\mathbb{R}) = \{A \in \mathbb{R}^{2\times 2} \ : \ det(A) = 1\}$ consists of **area-preserving linear transformations**

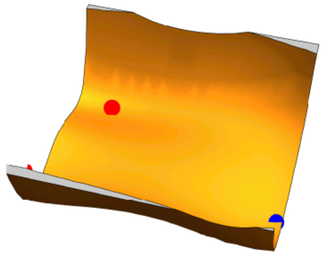
Example: acts on bivariate polynomials as $p\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) \mapsto p\left(A \begin{bmatrix} x \\ y \end{bmatrix}\right)$

Local minima



Global minimum





(a) $g = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ (b) $g = \begin{pmatrix} 1.2 & 0 \\ 0 & \frac{1}{1.2} \end{pmatrix}$ (c) $g = \begin{pmatrix} 1.4 & 0 \\ 0 & \frac{1}{1.4} \end{pmatrix}$

How do we take into account this structure?

Primer on Equivariant Learning

Primer on Equivariant Learning

Let G be a group, and let $\rho_1 : G \to GL(\mathcal{X})$ and $\rho_2 : G \to GL(\mathcal{Y})$ be group representations. A function $f : \mathcal{X} \to \mathcal{Y}$ is **equivariant** if

$$f(\rho_1(g)x) = \rho_2(g)f(x) \;\; \forall x \in \mathcal{X}, \forall g \in G$$

Primer on Equivariant Learning

Let G be a group, and let $\rho_1 : G \to GL(\mathcal{X})$ and $\rho_2 : G \to GL(\mathcal{Y})$ be group representations. A function $f : \mathcal{X} \to \mathcal{Y}$ is **equivariant** if

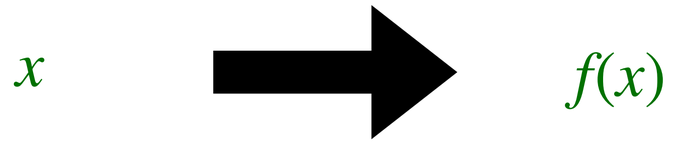
$$f(\rho_1(g)x) = \rho_2(g)f(x) \;\; \forall x \in \mathcal{X}, \forall g \in G$$

x

Primer on Equivariant Learning

Let G be a group, and let $\rho_1 : G \to GL(\mathcal{X})$ and $\rho_2 : G \to GL(\mathcal{Y})$ be group representations. A function $f : \mathcal{X} \to \mathcal{Y}$ is **equivariant** if

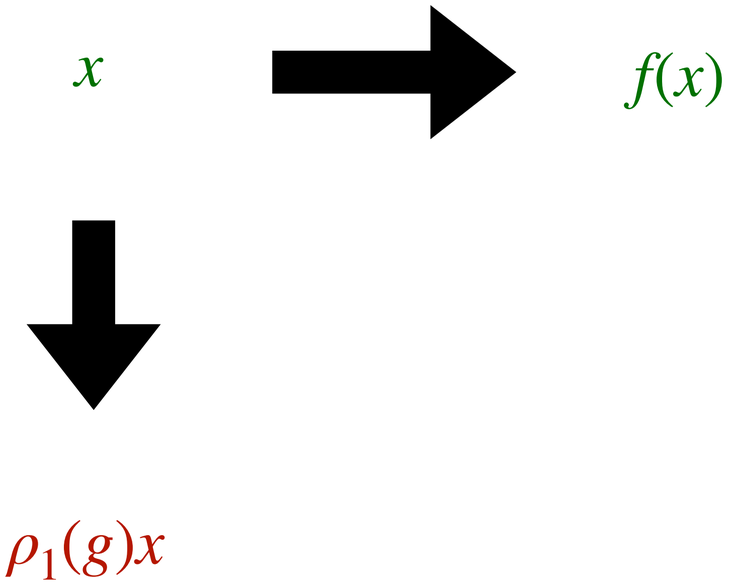
$$f(\rho_1(g)x) = \rho_2(g)f(x) \;\; \forall x \in \mathcal{X}, \forall g \in G$$



Primer on Equivariant Learning

Let G be a group, and let $\rho_1 : G \to GL(\mathcal{X})$ and $\rho_2 : G \to GL(\mathcal{Y})$ be group representations. A function $f : \mathcal{X} \to \mathcal{Y}$ is **equivariant** if

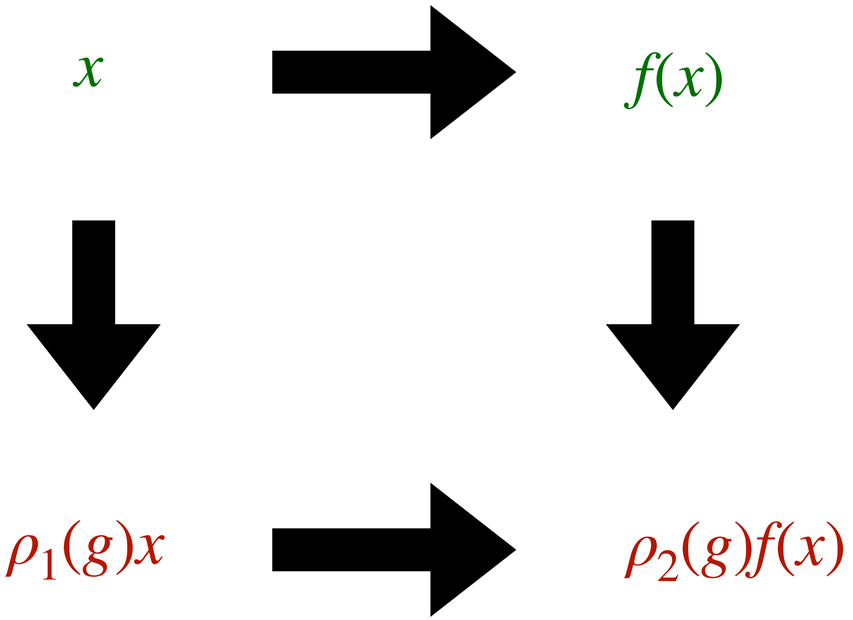
$$f(\rho_1(g)x) = \rho_2(g)f(x) \;\; \forall x \in \mathcal{X}, \forall g \in G$$



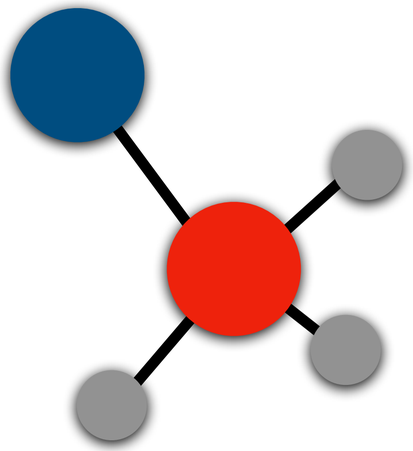
Primer on Equivariant Learning

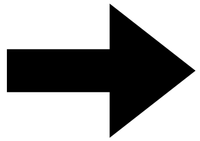
Let G be a group, and let $\rho_1 : G \to GL(\mathcal{X})$ and $\rho_2 : G \to GL(\mathcal{Y})$ be group representations. A function $f : \mathcal{X} \to \mathcal{Y}$ is **equivariant** if

$$f(\rho_1(g)x) = \rho_2(g)f(x) \;\; \forall x \in \mathcal{X}, \forall g \in G$$

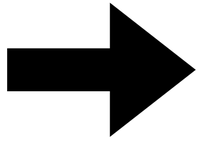


Primer on Equivariant Learning



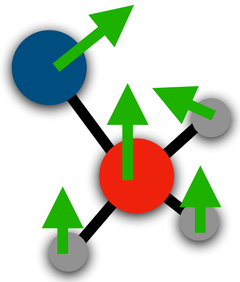


Equivariant Network

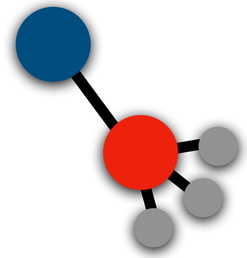


Energy e

Forces f

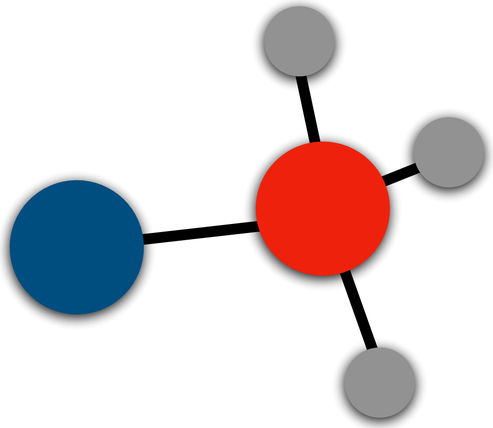


Relaxed structure



Consider predicting from an input molecular system

Primer on Equivariant Learning

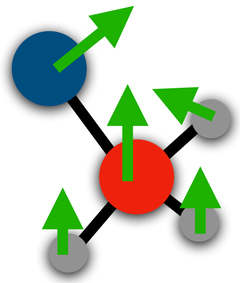


Rotate input

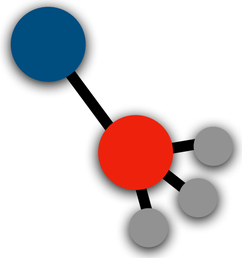
Equivariant Network

Energy e

Forces f

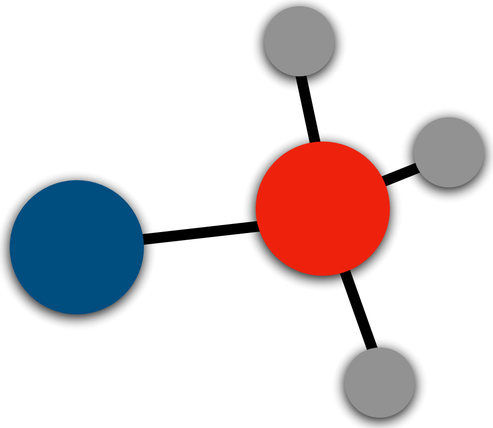


Relaxed structure



Consider predicting from an input molecular system

Primer on Equivariant Learning

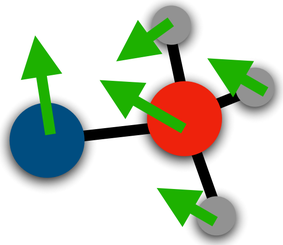


Rotate input

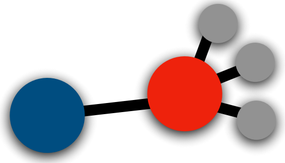
Equivariant Network

invariant
Energy e

equivariant
Forces f



equivariant
Relaxed structure

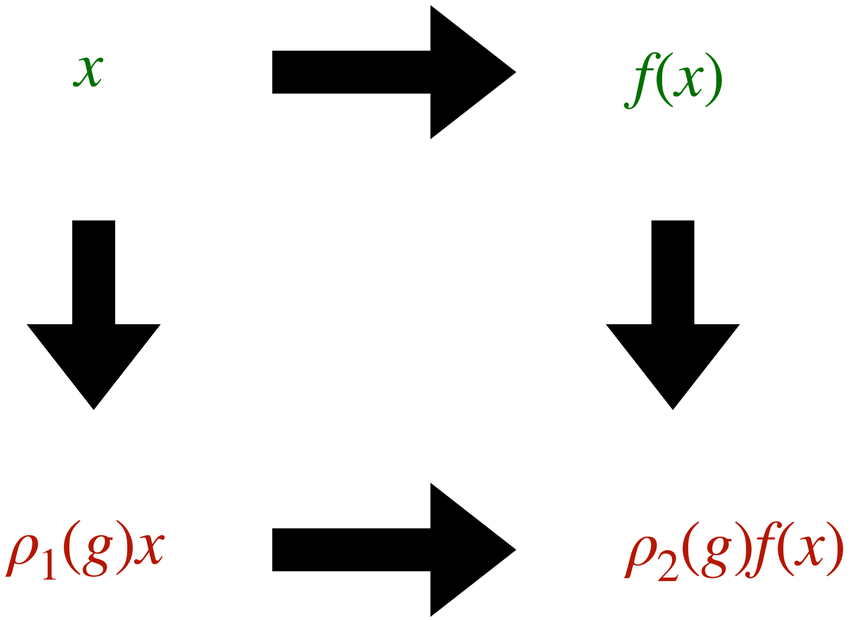


Consider predicting from an input molecular system

Primer on Equivariant Learning

Let G be a group, and let $\rho_1 : G \to GL(\mathcal{X})$ and $\rho_2 : G \to GL(\mathcal{Y})$ be group representations. A function $f : \mathcal{X} \to \mathcal{Y}$ is **equivariant** if

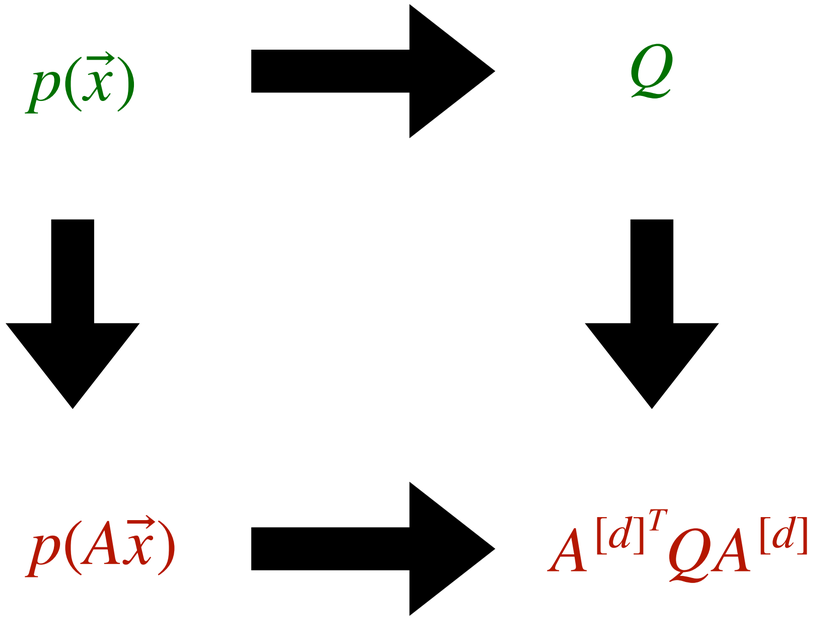
$$f(\rho_1(g)x) = \rho_2(g)f(x) \;\; \forall x \in \mathcal{X}, \forall g \in G$$



Primer on Equivariant Learning

Let G be a group, and let $\rho_1 : G \to GL(\mathcal{X})$ and $\rho_2 : G \to GL(\mathcal{Y})$ be group representations. A function $f : \mathcal{X} \to \mathcal{Y}$ is **equivariant** if

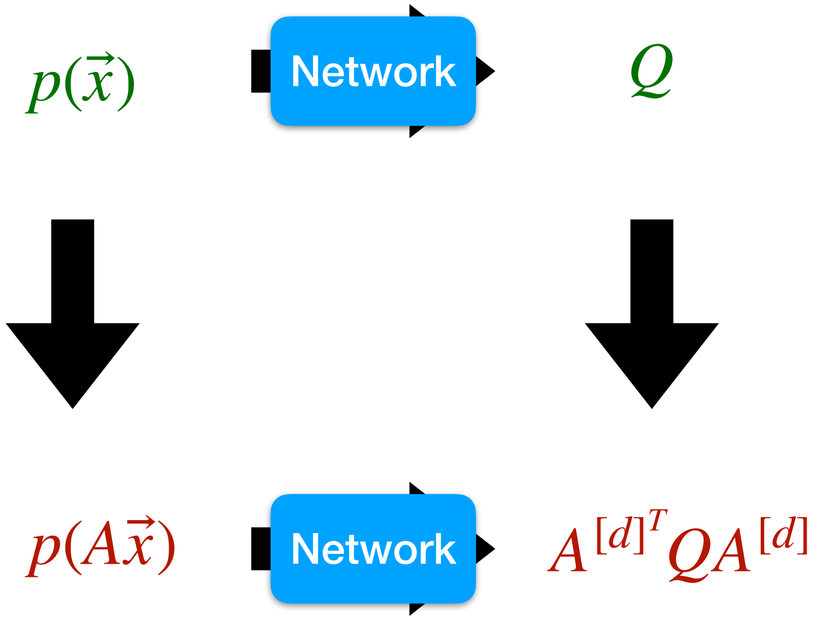
$$f(\rho_1(g)x) = \rho_2(g)f(x) \;\; \forall x \in \mathcal{X}, \forall g \in G$$



Primer on Equivariant Learning

Let G be a group, and let $\rho_1 : G \to GL(\mathcal{X})$ and $\rho_2 : G \to GL(\mathcal{Y})$ be group representations. A **network** $N : \mathcal{X} \to \mathcal{Y}$ is **equivariant** if

$$N(\rho_1(g)x) = \rho_2(g)N(x) \;\; \forall x \in \mathcal{X}, \forall g \in G$$



Equivariant Learning: Approaches

Compact groups:

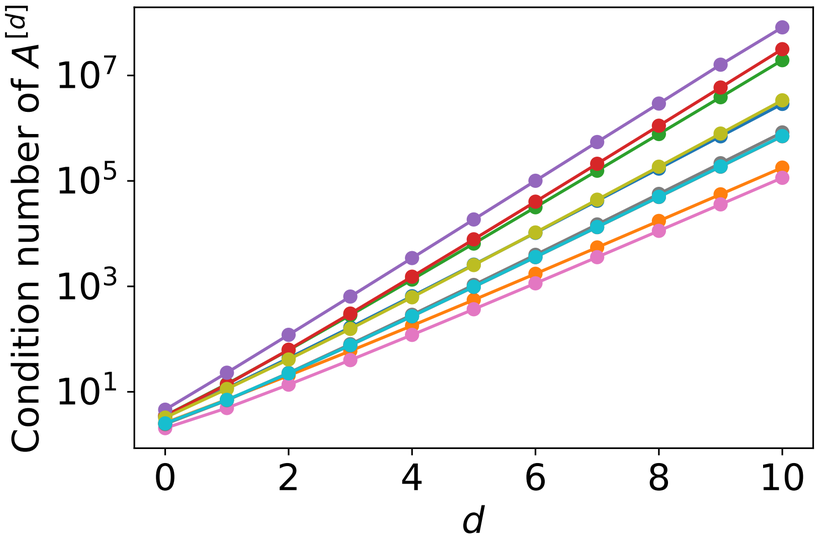
- Generalize convolutional networks to groups beyond translations
- Nonlinearities: pointwise vs tensor product

Non-compact groups:

- Approximate equivariance via Monte-Carlo approximation of integrals
- Exact equivariance via tensor products and Clebsch-Gordan transform

Practical Difficulties of $SL(2,\mathbb{R})$

1. The induced representations can be arbitrarily poorly conditioned



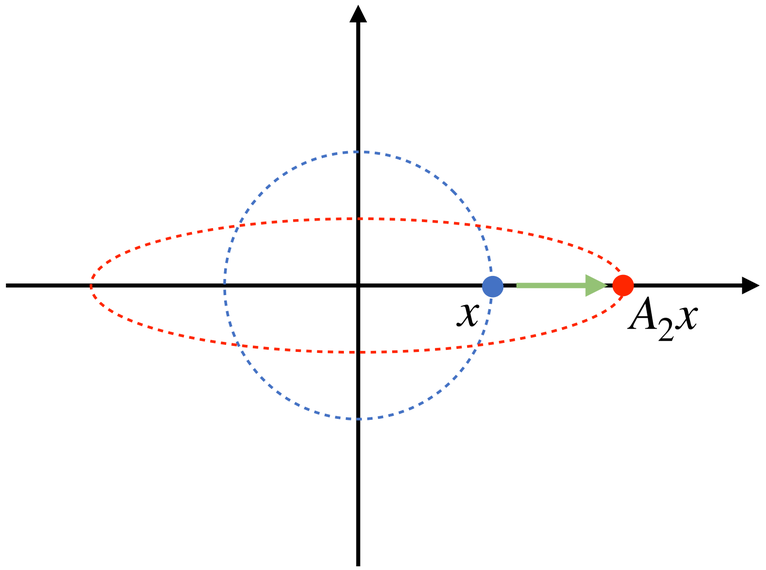
Here, each line is a randomly chosen element of $SL(2)$; d = degree of the induced representation ($d = 0$ is original 2×2 matrix)

Practical Difficulties of $SL(2,\mathbb{R})$

2. There is no finite, invariant measure over $SL(2)$, so all points in the orbit of a supported datapoint cannot be equally likely

Example:

$$x = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, A_n = \begin{bmatrix} n & 0 \\ 0 & \frac{1}{n} \end{bmatrix} \in SL(2)$$



In this example, the orbit of x is the entire x-axis, excluding the origin.

→ Think of equivariant learning as **out-of-distribution** robustness

Irreducible Reps of $SL(2,\mathbb{R})$

Def. A **group representation** is a vector space V together with a map $\rho : G \to GL(V)$ satisfying $\rho(g_1)\rho(g_2) = \rho(g_1 g_2)\ \forall g_1, g_2 \in G$

Def. An **irreducible representation** is a representation for which there does not exist an invariant subspace $W \subseteq V$ satisfying $\rho(g)w \in W$ for all $g \in G, w \in W$

The irreps are fundamental tools for understanding a group's other representations: for most groups of interest, can decompose a group representation into irreps:

$$\text{Representation } \rho(g) = U \begin{bmatrix} \text{Irrep } \rho_1(g) & & \\ & \ddots & \\ & & \text{Irrep } \rho_k(g) \end{bmatrix} U^{-1}$$

Finite-dim. Irreps of $SL(2,\mathbb{R})$

For each integer $d > 0$, the $(d+1)$-dimensional vector space of degree d binary forms ($V(d)$), is an irreducible representation of $SL(2,\mathbb{R})$.

Inputs are already in irrep spaces!

Example: $d = 3$, $10x^2y - 3y^3 \in V(3)$

Throughout, we assume a monomial basis for $V(d)$ (i.e. $x^d, x^{d-1}y, \ldots, y^d$) and represent elements of $V(d)$ as vectors of coefficients in $\mathbb{R}^{d+1}$.

Example: $10x^2y - 3y^3$ written as $\begin{bmatrix} 0 & 10 & 0 & -3 \end{bmatrix}$

What is ρ, i.e. how does $g \in SL(2,\mathbb{R})$ act on polynomials?

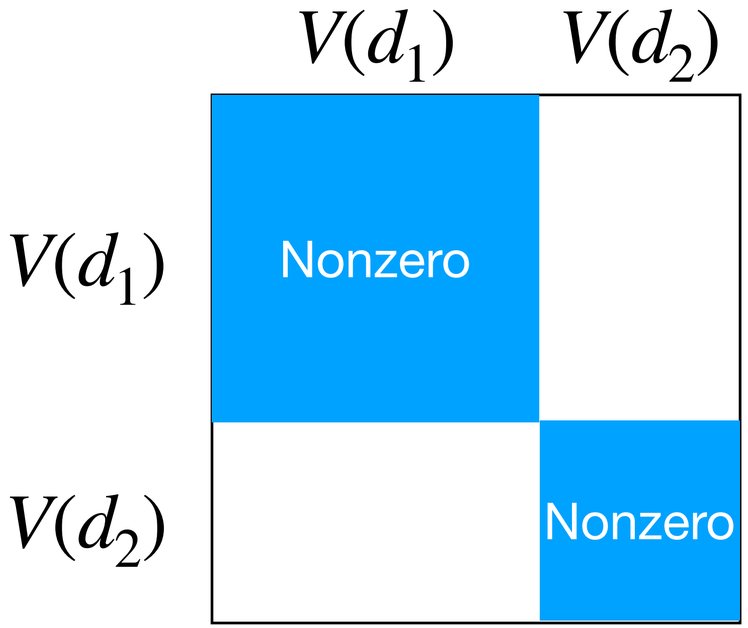
$$\rho(g)p(\vec{x}) = p(g\vec{x})$$

Linear, Learnable Layers?

Schur's Lemma → Linear Layers

Essentially: describes all linear, equivariant mappings between irrep spaces

Must send an element of an irrep space to either 0, or back to a copy of the same space!



How about a *nonlinear* layer?

Clebsch-Gordan: the transvectant

The tensor product of irreps is a representation, and therefore itself decomposes into irreps. (tensor product with self is not linear!)

For $SL(2,\mathbb{R})$, the exact decomposition is:

$$V(d_1) \otimes V(d_2) \simeq \bigoplus_{n=0}^{\min(d_1,d_2)} V(d_1 + d_2 - 2n)$$

The linear, invertible, equivariant map establishing this isomorphism

$T : V(d_1) \otimes V(d_2) \to \bigoplus_{n=0}^{\min(d_1,d_2)} V(d_1 + d_2 - 2n)$ is known as the **transvectant**

Clebsch-Gordan: the transvectant

$T : V(d_1) \otimes V(d_2) \to \bigoplus_{n=0}^{\min(d_1,d_2)} V(d_1 + d_2 - 2n)$ is the **transvectant**

Let $p(x_1, y_1) \in V(d_1)$ and $q(x_2, y_2) \in V(d_2)$. The $d_1 + d_2 - 2n$ component of $T(p \otimes q)$ is given by

$$\psi_n(p,q) := \left(\left(\frac{\partial^2}{\partial x_1 \partial y_2} - \frac{\partial^2}{\partial x_2 \partial y_1} \right)^n p \cdot q \right) \Bigg|_{\substack{x = x_1 = x_2 \\ y = y_1 = y_2}} \propto \sum_{m=0}^{n} (-1)^m \binom{n}{m} \frac{\partial^n p}{\partial x^{n-m} \partial y^m} \frac{\partial^n q}{\partial x^m y^{n-m}}$$

Clebsch-Gordan: the transvectant

$$T : V(d_1) \otimes V(d_2) \to \bigoplus_{n=0}^{\min(d_1,d_2)} V(d_1 + d_2 - 2n)$$ is the **transvectant**

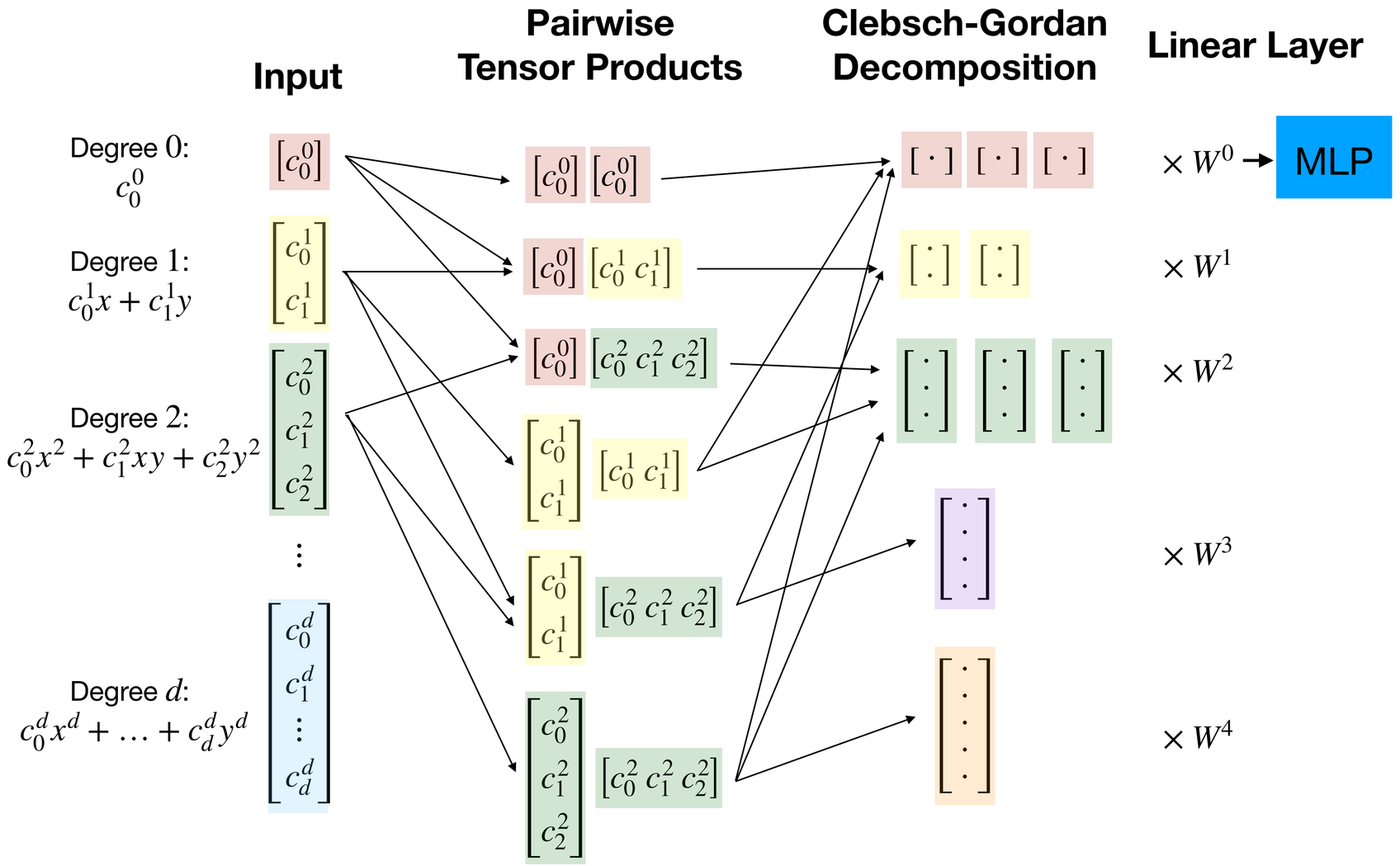
Let $p(x_1, y_1) \in V(d_1)$ and $q(x_2, y_2) \in V(d_2)$. The $d_1 + d_2 - 2n$ component of $T(p \otimes q)$ is given by

$$\psi_n(p,q) := \left(\left(\frac{\partial^2}{\partial x_1 \partial y_2} - \frac{\partial^2}{\partial x_2 \partial y_1} \right)^n p \cdot q \right) \Bigg|_{\substack{x = x_1 = x_2 \\ y = y_1 = y_2}} \propto \sum_{m=0}^{n} (-1)^m \binom{n}{m} \frac{\partial^n p}{\partial x^{n-m} \partial y^m} \frac{\partial^n q}{\partial x^m y^{n-m}}$$

Input: two homogeneous polynomials

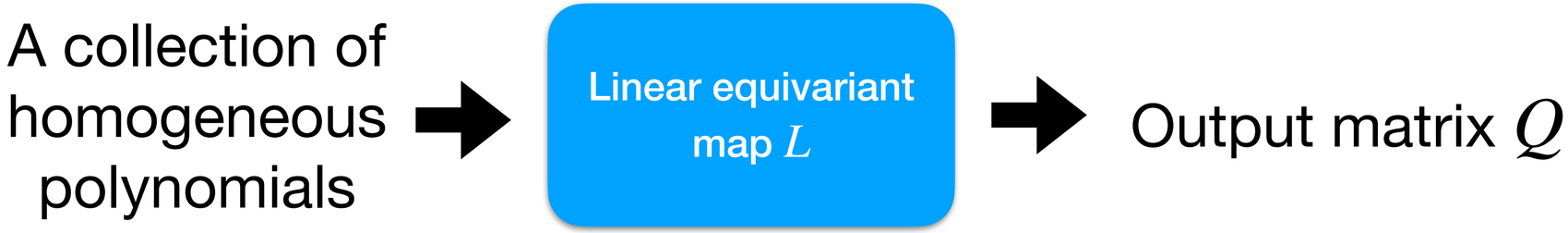
Output: another homogeneous polynomial

$SL(2,\mathbb{R})$-Equivariant Architecture



Final Layer Enforces $p(\vec{x}) = \vec{x}^{[d]^T} Q \vec{x}^{[d]}$

How to (1) go from irrep vector spaces, to the vector space of matrices? and (2) ensure that the output matrix represents the input polynomial?



Final Layer Enforces $p(\vec{x}) = \vec{x}^{[d]^T} Q \vec{x}^{[d]}$

How to (1) go from irrep vector spaces, to the vector space of matrices? and (2) ensure that the output matrix represents the input polynomial?

A collection of homogeneous polynomials → Linear equivariant map L → Output matrix Q

High-level: restrict mapping s.t. $p(\vec{x}) = \vec{x}^{[d]^T} Q \vec{x}^{[d]}$

Final Layer Enforces $p(\vec{x}) = \vec{x}^{[d]^T} Q \vec{x}^{[d]}$

How to (1) go from irrep vector spaces, to the vector space of matrices? and (2) ensure that the output matrix represents the input polynomial?

A collection of homogeneous polynomials

where degree d one is the original $p(\vec{x})$

→ Linear equivariant map L →

Output matrix Q

Our choice of L enforces that this works

Note: the PSD property is what must be learned!

How expressive is this architecture?

Universality is a standard **sanity check** for neural architectures:

Can this architecture approximate any continuous (*equivariant*) function, if given enough parameters (e.g. sufficiently large width or channels)?

How expressive is this architecture?

Universality is a standard **sanity check** for neural architectures:

Can this architecture approximate any continuous (*equivariant*) function, if given enough parameters (e.g. sufficiently large width or channels)?

The answer is typically yes, and proceeds via a polynomial approximation argument.

i.e., show that the network can approximate any (*equivariant*) polynomial to arbitrary precision, and then appeal to Stone-Weierstrass Theorem

How expressive is this architecture?

Universality is a standard **sanity check** for neural architectures:

Can this architecture approximate any continuous (*equivariant*) function, if given enough parameters (e.g. sufficiently large width or channels)?

The answer is typically yes, and proceeds via a polynomial approximation argument. ⚠

i.e., show that the network can approximate any (*equivariant*) polynomial to arbitrary precision, and then appeal to Stone-Weierstrass Theorem

How expressive is this architecture?

Universality is a standard **sanity check** for neural architectures:

Can this architecture approximate any continuous (*equivariant*) function, if given enough parameters (e.g. sufficiently large width or channels)?

The answer is typically yes, and proceeds via a polynomial approximation argument. ⚠

i.e., show that the network can approximate any (*equivariant*) polynomial to arbitrary precision, and then appeal to Stone-Weierstrass Theorem

Early work on universal equivariant architecture by Yarotsky 2018; proven for Clebsch-Gordan nets and similar architectures by Bogatskiy et al 2020

Surprise: it is **not** universal

This is a sharp contrast to previous, very analogous work!

Counterexample: max-det function

Theorem. Let $\mathcal{N}_W(p)$ denote the network output with (learned) parameters W applied to the input p. Let f be the continuous, $SL(2,\mathbb{R})$-equivariant function

$$f(p) = \text{argmax} \log\det Q \text{ such that } p(\vec{x}) = \vec{x}^{[d]^T} Q \vec{x}^{[d]} \text{ and } Q \succeq 0$$

There exists an input polynomial p, and an absolute constant $\epsilon > 0$ such that for any W, $|f(p) - \mathcal{N}_W(p)| > \epsilon$. Therefore, the architecture is not universal.

Proof outline of counterexample

Def. Call the monomial $x^k y^r$ **balanced mod** b if $k \equiv r \pmod{b}$. The polynomial p is **balanced mod** b if it is the weighted sum of monomials that are all balanced mod b.

Lemma. If f and g are balanced mod b, then $\forall n\ \psi_n(f, g)$ is balanced mod b. Similarly, the linear layer and MLP on invariants preserve balancedness.

Example: x^2y^2 and x^8 balanced mod 8, because $2 \equiv 2 \pmod{8}$ and $8 \equiv 0 \pmod{8}$. However, x^3y is not: $3 \not\equiv 1 \pmod{8}$.

Proof outline of counterexample

Proof of theorem. Let $p(x, y) = x^8 + y^8$. Since p is balanced mod 8, so too is the final network output. In particular, the degree 4 component can only contain the monomial x^2y^2. In total, the last layer of the network (for the max-log-det problem) takes as input the monomials x^8, y^8, x^4y^4, x^2y^2, and 1, all of which map to a matrix of sparsity pattern

$$\begin{pmatrix} \times & 0 & 0 & 0 & \times \\ 0 & 0 & 0 & \times & 0 \\ 0 & 0 & \times & 0 & 0 \\ 0 & \times & 0 & 0 & 0 \\ \times & 0 & 0 & 0 & \times \end{pmatrix}$$

However, the computed matrix $f(x^8 + y^8)$ does not match this sparsity pattern.

Not approximable by any polynomial!

Is this just a problem with our choice of architecture?

- Using the same techniques as Bogatskiy et al, one can show that our network can approximate any equivariant polynomial

Is it a contrived or unnatural function?

- Nope, it's the exact function we want to approximate!

Therefore, the max log det function f **cannot be approximated by equivariant polynomials**

- This suggests that any hope of getting around the issue must use a fundamentally different paradigm and proof of universality than prior work

Intuitively, what has gone wrong?

Plenty of analogous architectures, even for non-compact groups, are universal. What's different?

- The hard function f is only defined on positive polynomials, not on all inputs
- We work with real values, i.e. $SL(2,\mathbb{R})$ instead of $SL(2,\mathbb{C})$
  - Previous work relies on integration over a maximal compact subgroup, the special orthogonal group. In the complex case, $SO(2,\mathbb{C})$ and $SL(2,\mathbb{C})$ invariants are equivalent.
  - But not so in the real-valued case! E.g. $a + c$ is an invariant of the polynomial $ax^2 + bxy + cy^2$ under an $SO(2,\mathbb{R})$ action, but is not an invariant under an $SL(2,\mathbb{R})$ action.

Was this all a waste — should we throw out our knowledge of $SL(2,\mathbb{R})$?

No!

Alternative: Data Augmentation

For each training point $(p(\vec{x}), f(p(\vec{x})))$, apply a random $SL(2,\mathbb{R})$ transformation:

$$(p(\vec{x}), f(p(\vec{x}))) \mapsto (p(A^{[d]}\vec{x}), A^{[d]^T} f(p(\vec{x}))A^{[d]}) \text{ where } A \sim \mu(SL(2,\mathbb{R}))$$

Doesn't enforce equivariance strictly, but:

- Standard method for encouraging out-of-distribution robustness
- Easy to combine with any architecture

Alternative: Data Augmentation

For each training point $(p(\vec{x}), f(p(\vec{x})))$, apply a random $SL(2,\mathbb{R})$ transformation:

$$(p(\vec{x}), f(p(\vec{x}))) \mapsto (p(A^{[d]}\vec{x}), A^{[d]^T} f(p(\vec{x}))A^{[d]}) \text{ where } A \sim \mu(SL(2,\mathbb{R}))$$

What distribution μ to choose?

Restrict the condition number of A!

Experiments: Methods

Model Name	Description
mlp-aug-rots	MLP with rotation augmentations
mlp-aug-a-b	MLP with augmentations by $g^{[d]}$ and g has condition number κ s.t. $a \leq \kappa \leq b$
SO2Net-aug-a-b	$SO(2, \mathbb{R})$ equivariant network with data augmented by $g^{[d]}$ as above
SO2Net	$SO(2, \mathbb{R})$ equivariant architecture with no data augmentation
SL2Net	$SL(2, \mathbb{R})$ equivariant architecture with no data augmentation
MLP	multilayer perceptron with no data augmentation

Experiments: Synthetic Data

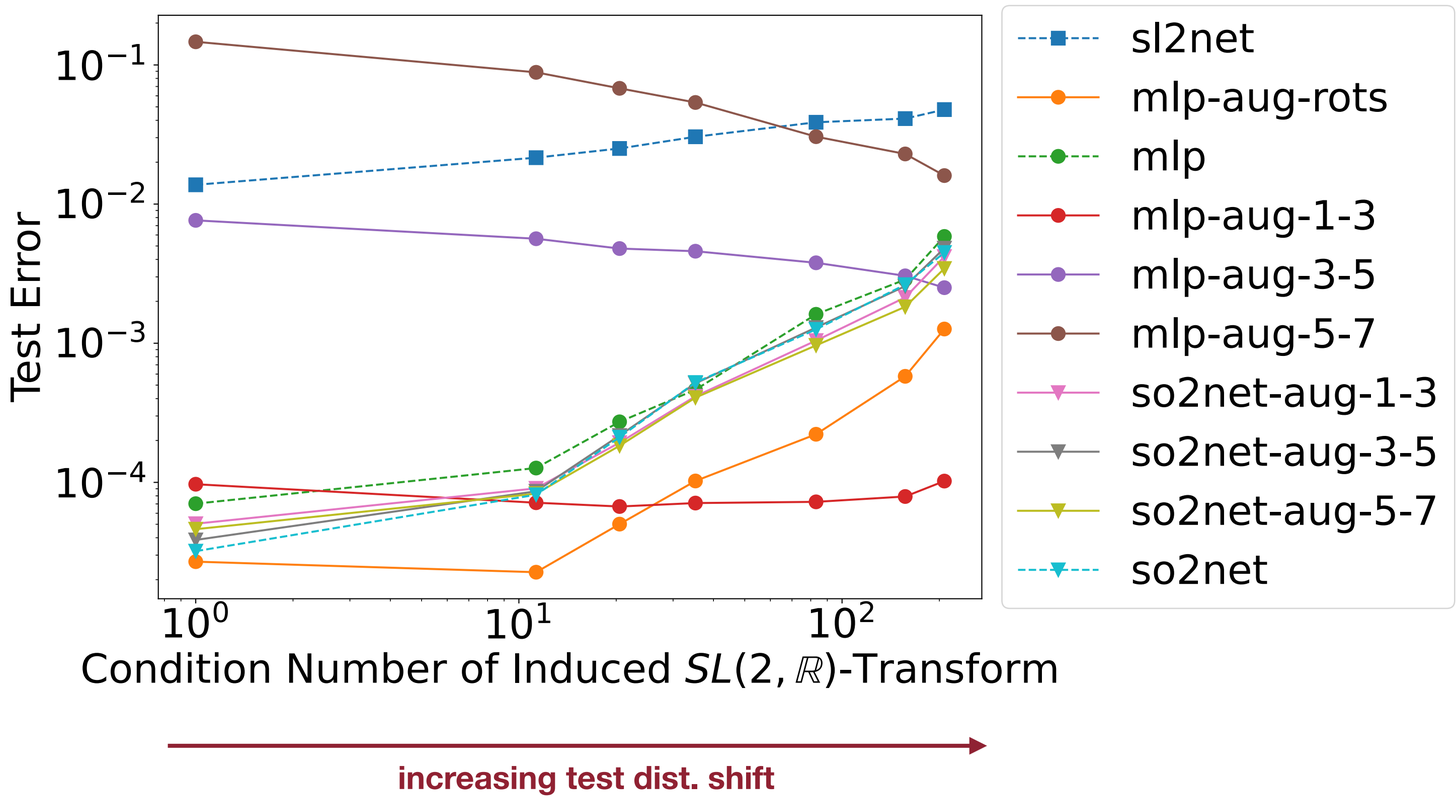
Data generation: **only** positive polynomials

$$\text{Sample } U \in \mathbb{R}^{d+1 \times d+1} \text{ via } U_{ij} \sim_{iid} N(0,1)$$

$$\text{Set } p(\vec{x}) = \vec{x}^{[d]^T}(U^T U + 10^{-8} I)\vec{x}^{[d]}, \text{ which is positive}$$

Labels: computed using the Mosek optimizer

Experiment Results



MLP with data augmentation by well-conditioned elements of $SL(2,\mathbb{R})$ is best!

MLP is faster than a convex solver!

Degree	6	8	10	12	14
MLP NMSE	9.5e-6	6.0e-5	2.9e-5	2.3e-5	1.1e-5
MLP times (min)	0.062	0.082	0.17	0.22	0.29
SCS NMSE	2.7e-5	6.2e-5	1.2e-4	2.7e-4	1.2e-3
SCS times (min)	3.50	4.94	9.11	18.8	37.4

SCS: a first order solver (O'Donoghue et al 2016)

NMSE: normalized mean squared error, with respect to the ground truth labels computed by the second order solver, Mosek (ApS, 2022)

These times are estimates for 5,000 *test* examples on a single **CPU**

→ **The trained MLP has a much faster forward pass than a solver, at same error!**

Conclusions

- Machine learning is a promising avenue for accelerating polynomial problems
  - Even a simple architecture can harness patterns in a dataset not used by traditional convex solvers
- Non-compact group equivariance can be very subtle, despite analogies to compact groups!
  - Any approach to complete $SL(2,\mathbb{R})$-equivariance must enable non-polynomial approximations
- Data augmentation is currently the best “equivariant” option for $SL(2,\mathbb{R})$

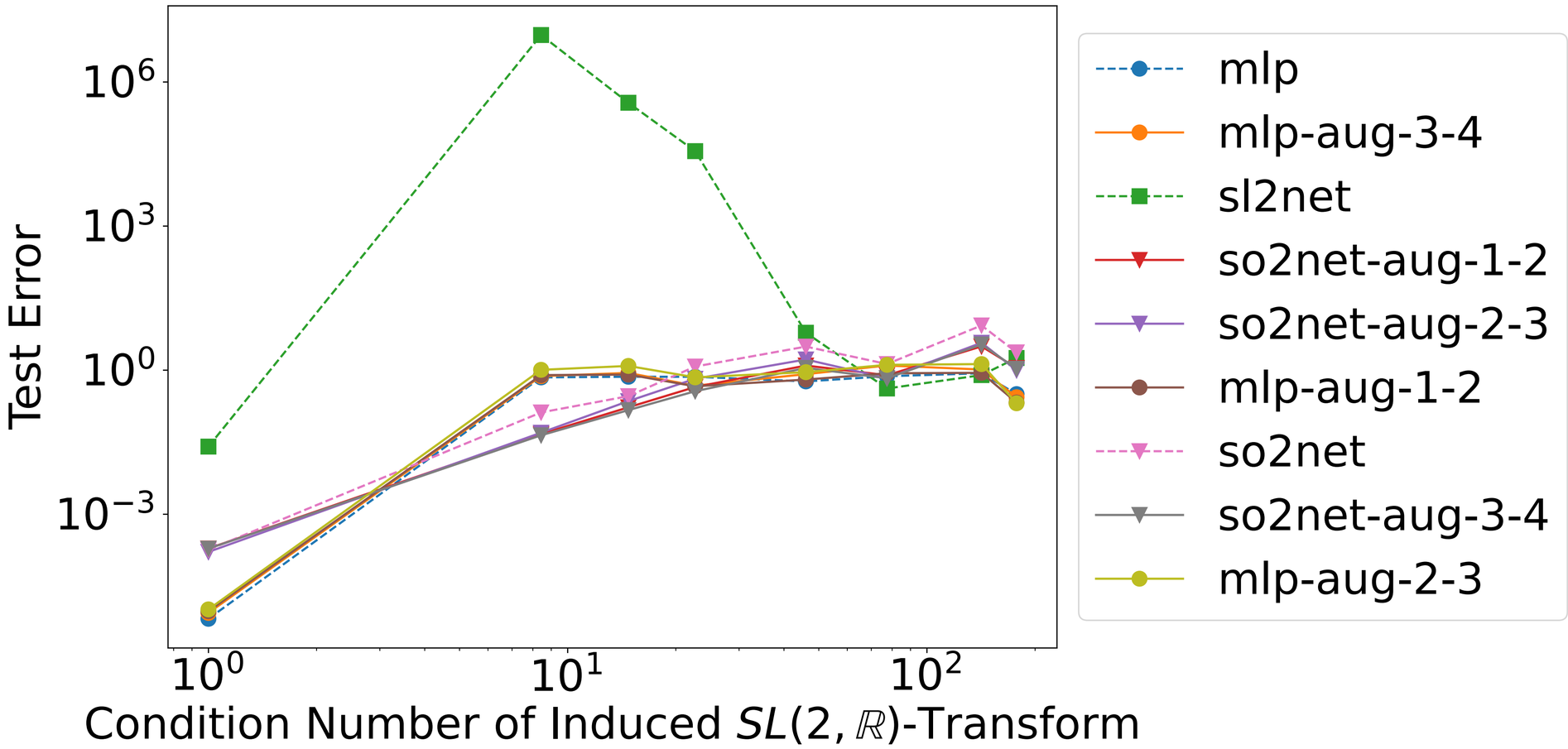
Future Directions

- Incorporating $SL(2,\mathbb{R})$ structure **without** loss of expressivity
  - For invariance: use separators, e.g. separating invariants or bilipschitz invariants
  - Compute some non-polynomial equivariant feature and input to network?
- Applications!
  - Non-synthetic data — e.g. replace solver calls in robotics application
  - Multivariate extension with $SL(d,\mathbb{R})$ (easy for data augmentation!)

Thanks! Questions?

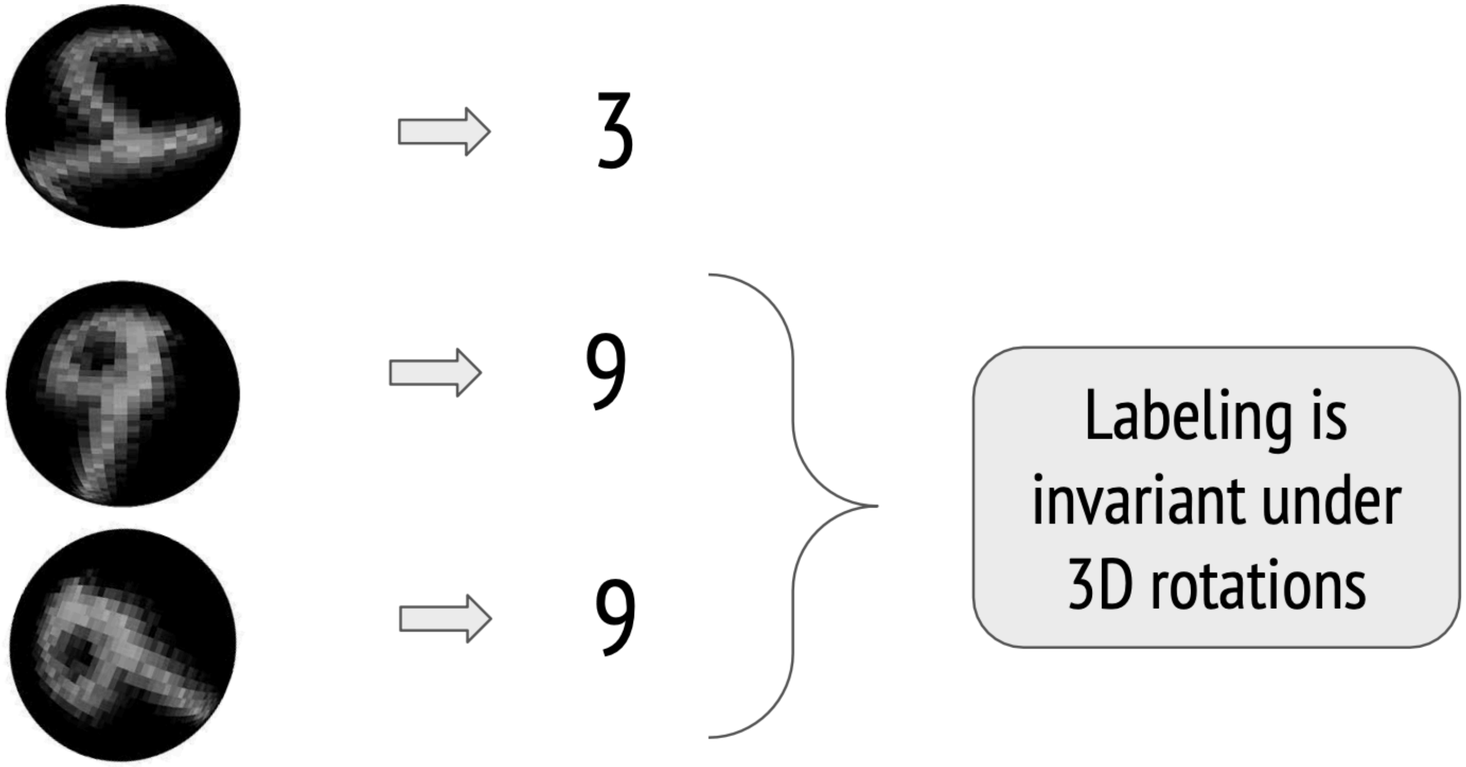
arXiv:2312.02146

Experiments: Poly Min for Spherical Code Bounds



Background: group invariance can be a useful inductive bias

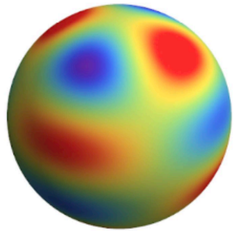
Example: "rotated MNIST on the sphere"



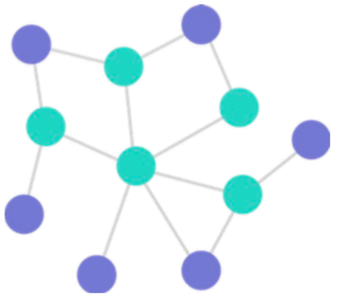
Source(s): Cohen, Taco S., et al. "Spherical cnns." *arXiv preprint arXiv:1801.10130* (2018).
Image source(s): https://openreview.net/pdf?id=Hkbd5xZRb

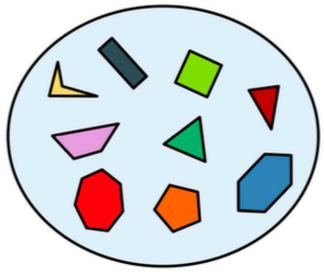
Background: group symmetries

$SO(3)$

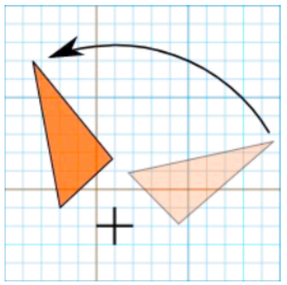


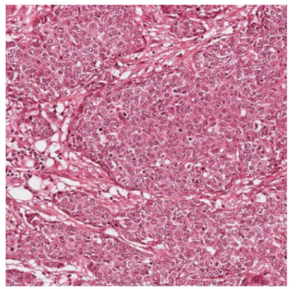
S_n





SE_2





Main idea: enforce group symmetries via architecture of network

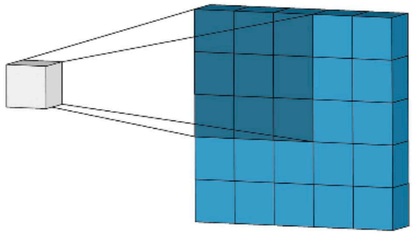
How to build such an architecture?

Source(s): Cohen, Taco S., et al. "Spherical cnns." *arXiv preprint arXiv:1801.10130* (2018).
Image source(s): https://openreview.net/pdf?id=Hkbd5xZRb

Group convolutions are equivariant

Familiar convolution (1D translation group): $f, g : \mathbb{R} \to \mathbb{R}$

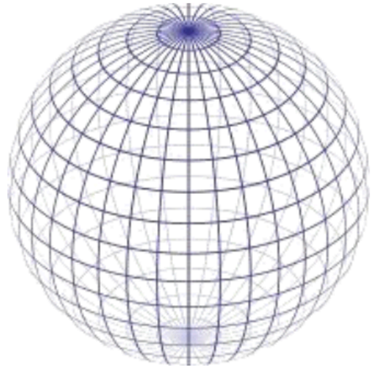
$$(f * g)(x) = \int f(x-y)\, g(y)\, dy$$



Convolution with respect to a **general** group: $f, g : G \to \mathbb{R}$

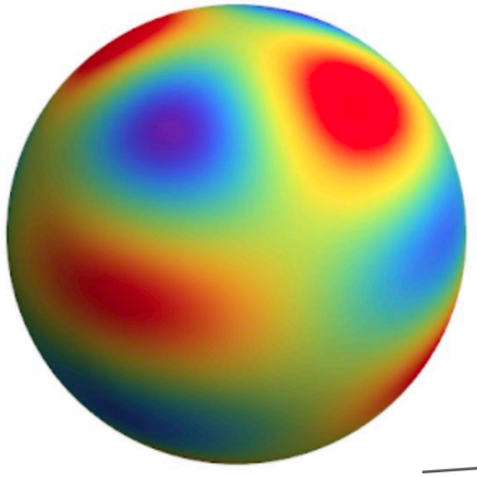
$$(f * g)(u) = \sum_{v \in G} f(uv^{-1})\, g(v)$$

**Can be generalized to functions on spaces on which G acts*



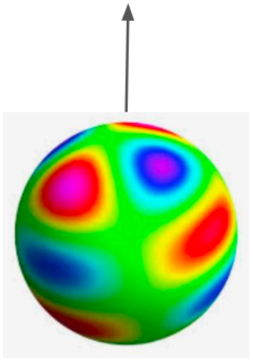
To build a G-equivariant neural net, one can **compose convolutions and non-linearities** as in an ordinary CNN

Example: Spherical CNNs



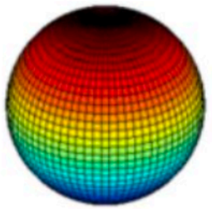
Input: function on a sphere

At each layer: convolve them and apply pointwise nonlinearity



Learned filter (function on sphere or or rotation group)

Input to next layer



Repeat with a new filter for each layer

Spherical CNNs by Cohen, Geiger, Kohler, and Welling

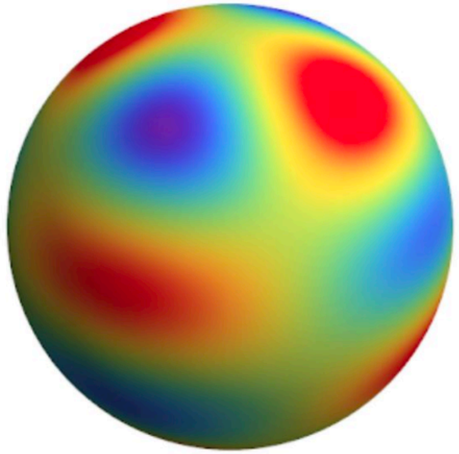
Tool: Fourier Analysis on Compact Groups

Classical	Group
$f : \mathbb{R} \to \mathbb{C}$	$f : G \to \mathbb{C}$
$\rho_k = e^{ik\theta} : \mathbb{R} \to \mathbb{C}$	$\rho_l : G \to \mathbb{C}^{S_l \times S_l}$
$\hat{f}(k) = \int_{\mathbb{R}} e^{ik\theta} f(\theta) d\theta$	$\hat{f}(l) = \int_G \rho_l(g) f(g) dg$

Tool: Fourier Analysis on Compact Groups

Classical	Group
$f(\theta) = \int_{\mathbb{R}} e^{-ik\theta} \hat{f}(k) dk$	$f(g) = \sum_{l=0}^{\infty} \mathrm{tr}[\hat{f}(l)\rho_l(g^{-1})]$
$\{\rho_k\}$ form basis for $L^2(\mathbb{R})$	$\{\rho_l\}$ matrix entries form basis for $L^2(G)$
$\widehat{f * h}(k) = \hat{f}(k)\hat{h}(k)$	$\widehat{f * h}(l) = \hat{f}(l)\hat{h}(l)$ matrix multiplication!

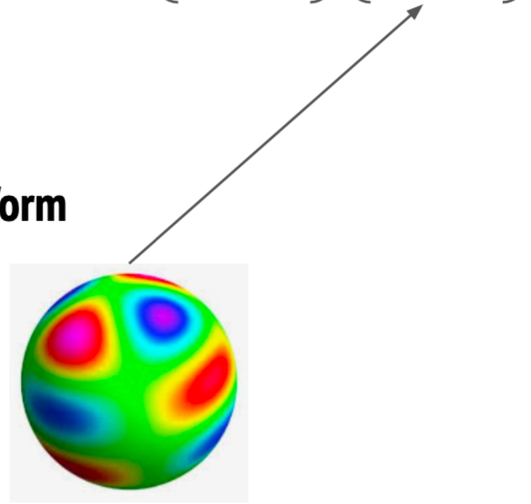
Convolutions in Fourier space, in practice



Input: function on a sphere

SO(3)-transform

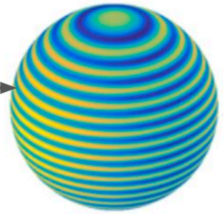
Matrix multiplication



SO(3)-transform

Learned filter (function on sphere or rotation group)

Inverse SO(3)-transform



Pointwise nonlinearity

Input to next layer