Tutorial: Rotational equivariance constraints in NN architecture

Alex Connolly

M2LInES Winter 2023 Annual Meeting

https://m2lines.github.io/

Tutorial

1. Intro to equivariance
2. Intro to steerable CNNs
3. Pytorch baseline follow-along demo
    - Inputs one vector + one scalar
    - Outputs one ‘scalar’
4. Primer on groups and representations
5. Equivariant model follow-along demo
    - Same inputs, outputs as pytorch baseline
6. Coding activity
    - Inputs one vector + one scalar
    - Outputs one vector + one ‘scalar’

Tutorial

Scripts:
https://github.com/adconnolly/Tutorial-RotEquiv

Data:
https://g-fe3828.0da32.08cc.data.globus.org/coarse4x40104_Re900.nc
(can be downloaded from scripts instead)

Equivariance and invariance

Invariance

$$f(gx) = f(x)$$

‘Change’ the inputs, and the output is unchanged

e.g. Kinetic energy upon rotation/reflection of system

Equivariance

$$f(gx) = gf(x).$$

‘Change’ the inputs, and the output is changed in same way

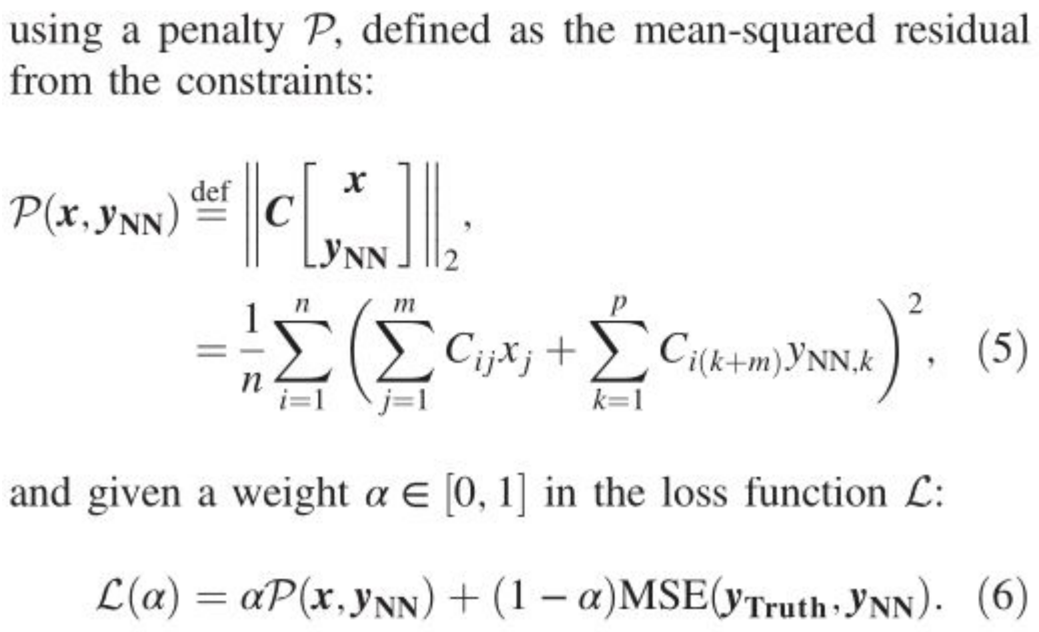
e.g. Momentum upon rotation/reflection of system

‘Change’ could mean rotate, reflect, translate, zoom-in/out, etc.

Soft vs. hard constraints in NN

Soft: Data-augmentation, physical loss terms

From Beucler et al. 2019, "Enforcing Analytic Constraints in Neural Networks Emulating Physical Systems"



using a penalty $\mathcal{P}$, defined as the mean-squared residual from the constraints:

$$\mathcal{P}(\boldsymbol{x}, \boldsymbol{y}_{\mathrm{NN}}) \overset{\mathrm{def}}{=} \left\| \boldsymbol{C} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y}_{\mathrm{NN}} \end{bmatrix} \right\|_2 , \\ = \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{m} C_{ij} x_j + \sum_{k=1}^{p} C_{i(k+m)} y_{\mathrm{NN},k} \right)^2 , \quad (5)$$

and given a weight $\alpha \in [0, 1]$ in the loss function $\mathcal{L}$:

$$\mathcal{L}(\alpha) = \alpha \mathcal{P}(\boldsymbol{x}, \boldsymbol{y}_{\mathrm{NN}}) + (1 - \alpha) \mathrm{MSE}(\boldsymbol{y}_{\mathbf{Truth}}, \boldsymbol{y}_{\mathrm{NN}}). \quad (6)$$

Hard: Strictly enforced in the design of the NN

Weight tying/sharing for steerable CNN

Convolutional kernels can't be anything

Respect the same transformations as the inputs/outputs

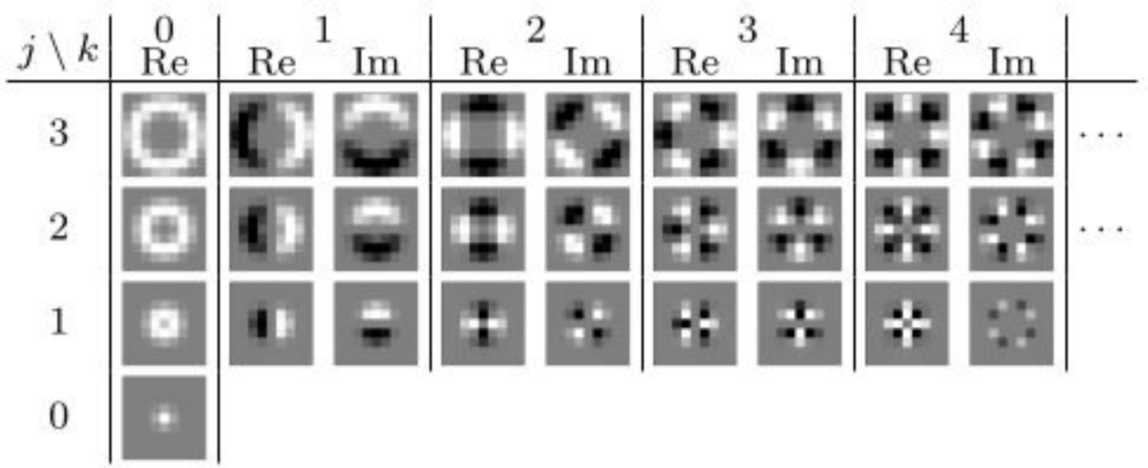


Figure 2: Illustration of the circular harmonics $\psi_{jk}(r, \phi) = \tau_j(r)\, e^{ik\phi}$ sampled on a 9×9 grid. Each row shows a different radial part j, the angular frequencies are arranged in the columns. For larger scales there are higher frequency filters not shown here.

From "Learning Steerable Filters for Rotation Equivariant CNNs"
Maurice Weiler et al. 2018

Weight tying/sharing for steerable CNN

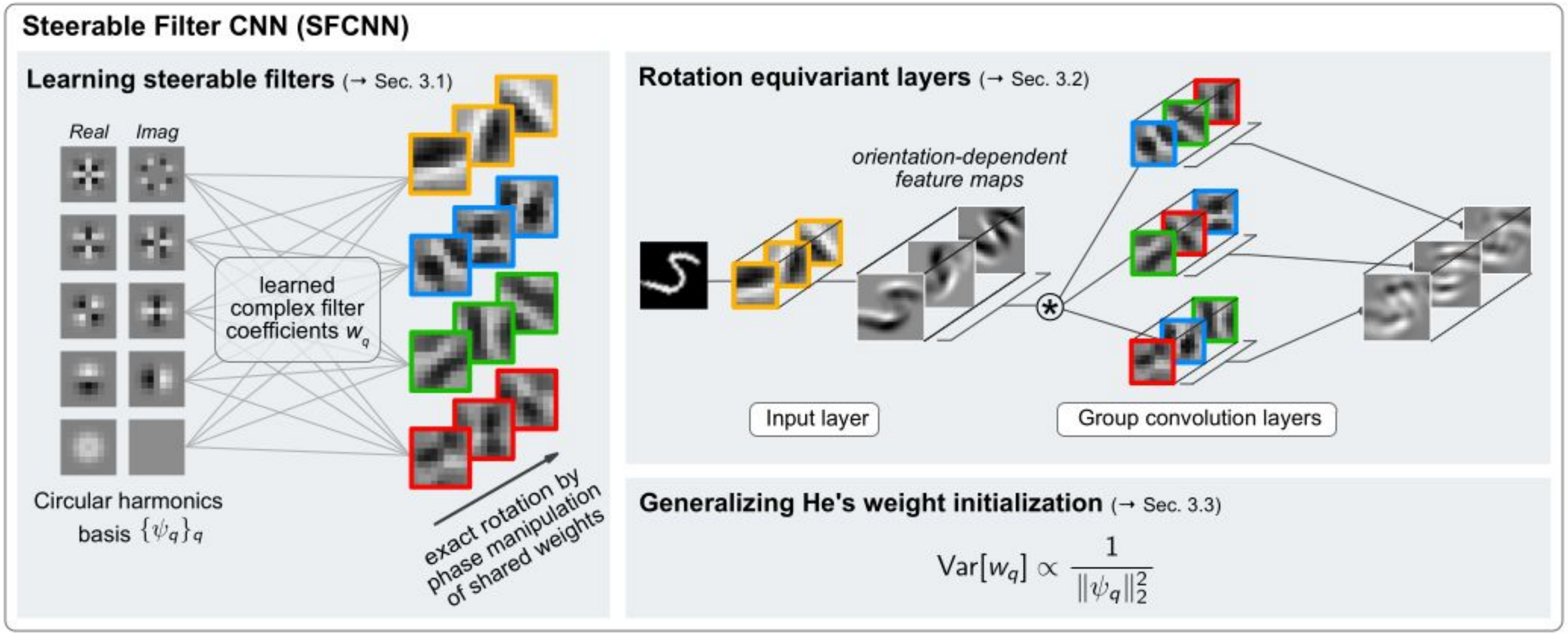
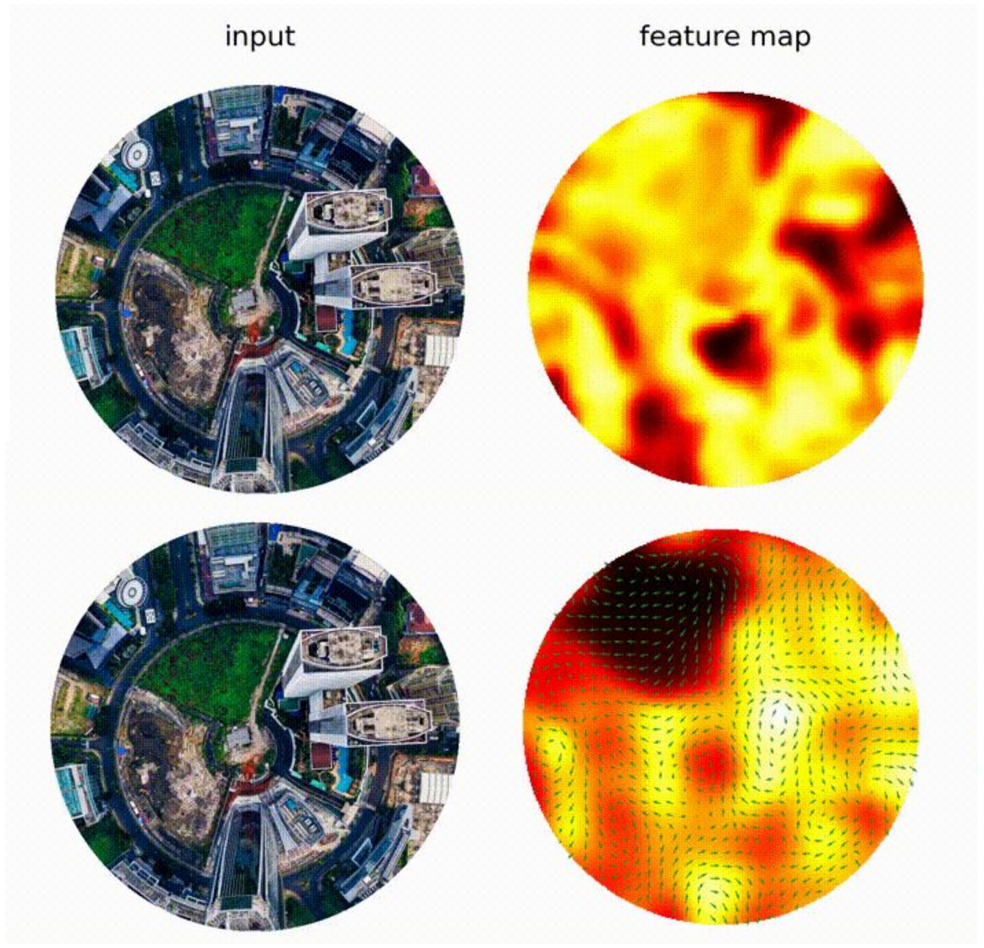

Steerable Filter CNN (SFCNN)
Learning steerable filters (→ Sec. 3.1)
Real
Imag
learned complex filter coefficients w_q
Circular harmonics basis $\{\psi_q\}_q$
exact rotation by phase manipulation of shared weights
Rotation equivariant layers (→ Sec. 3.2)
orientation-dependent feature maps
Input layer
Group convolution layers
Generalizing He's weight initialization (→ Sec. 3.3)
$\mathrm{Var}[w_q] \propto \frac{1}{\|\psi_q\|_2^2}$

Figure 1: Key concepts of the proposed Steerable Filter CNN: The filters are parameterized in a steerable function space with shared weights over filter orientations. Exact filter rotations are achieved by a phase manipulation of the expansion coefficients w_q. All layers are designed to be jointly translation and rotation equivariant. The weights w_q serve as expansion coefficients of a fixed filter basis $\{\psi_q\}_q$ rather than pixel values. Therefore, we adapt He's weight initialization scheme to this more general case which implies to normalize the basis filter energies.

e2cnn python library



Start your jupyterhub server now

- ‘large + pytorch’

Easier to implement than understand

https://github.com/QUVA-Lab/e2cnn

Follow-along demo

Create simple model with pytorch

Train with biased data, forcing winds always in x-direction

Manually rotate input scalar and vector fields,

Compare model prediction from rotated input to rotated ground truth, it will perform poorly

Use same biased data to train simple model with e2cnn

Make same comparison of predictions from rotated input to rotated ground truth, it will perform well

Coding activity: You will modify my code to make a more complicated model by adding a vector output

Pytorch baseline

Your data: Subset of your own ocean or atmospheric simulations

Inputs: Good choices might be:

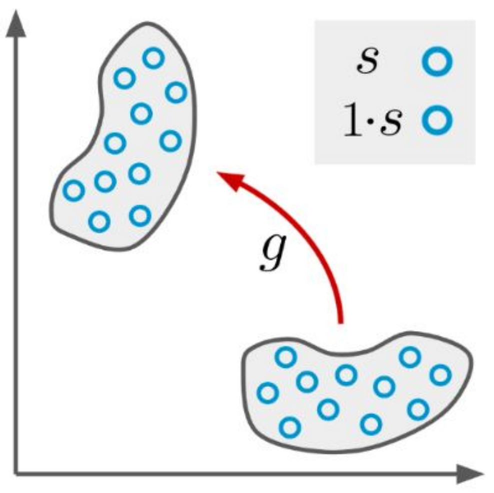
- (u,v,w) (my choice)
- (u,v, temperature/buoyancy)
- (ρu, ρv, TKE)

Outputs: Good choices might be

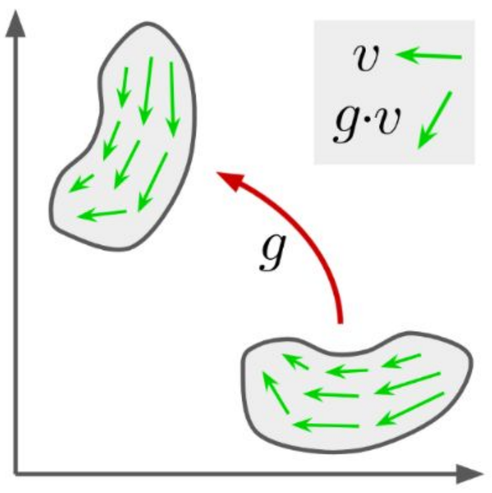
- <u'v'> (my choice, <*> = filtering, overbar)
- <w'T'> or <w' b'>
- TKE

Collocated is necessary and multiple vertical levels is better

Rotation of scalar and vector fields



scalar field



vector field

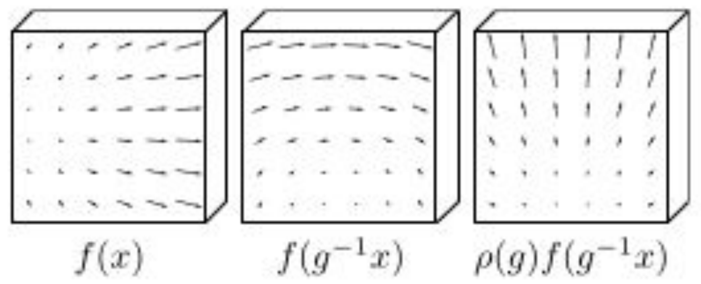


Figure 1: To transform a vector field (L) by a 90° rotation g, first move each arrow to its new position (C), keeping its orientation the same, then rotate the vector itself (R). This is described by the induced representation

"3D Steerable CNNs: Learning Rotationally Equivariant Features in Volumetric Data" Weiler et al.

Key is that there is two steps for vectors

Second step is probably more familiar, multiplication by rotation matrix

Rotation of scalar and vector fields

To rotate values to their new positions in the field, we will make use of numpy's rot90 function. For scalars, we're done

To rotate the 3-d velocity vector, (u,v,w), rotations in just the x and y directions correspond to multiplication by the following matrix:

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This leaves w component unaffected, so we'll ignore 3rd row and column in practice

“Sign” representation

For rotations by 90 degree there is an additional 1-dimensional ‘representation’ - the ‘sign’ representation

I chose, <u’v’> as my output to demonstrate this representation

Recall, upon rotation by 90 deg u -> v and v -> -u

Substitute these,

<v’(-u)’> = - <u’v’> so rotation by 90 degrees leads to sign change (hey, that’s why it’s called the sign representation!)

e2cnn python library

E(2)	Euclidean group in 2 dimensions
+	
CNN	Convolutional Neural Network

We know what CNNs are, but what is the deal with E2?

Groups

Group: A set and an operation that combines any two elements of the set to produce a third element of the set

E(2) (Euclidean group in 2 dimensions): Set of all distance preserving transformations

- Rotation
- Reflection
- Translation
- Arbitrary combinations, combined through the operation of composition

Way too abstract! Sorry, stay with me

Groups

EUCLIDE Mathematicien
Renommé par ses Élements de Geometrie
il florissoit 319 avant l'ere Ch.
a Paris chés Daumont rue St. Martin

E(2) Euclidean group combinations of

- Rotation
- Reflection
- Translation

Translational equivariance for free in CNN so let’s drop that

O(2) Orthogonal Group

- Rotation
- Reflection

And reflection messes with the handedness of the system, i.e. up becomes down, so let’s drop that too

Groups

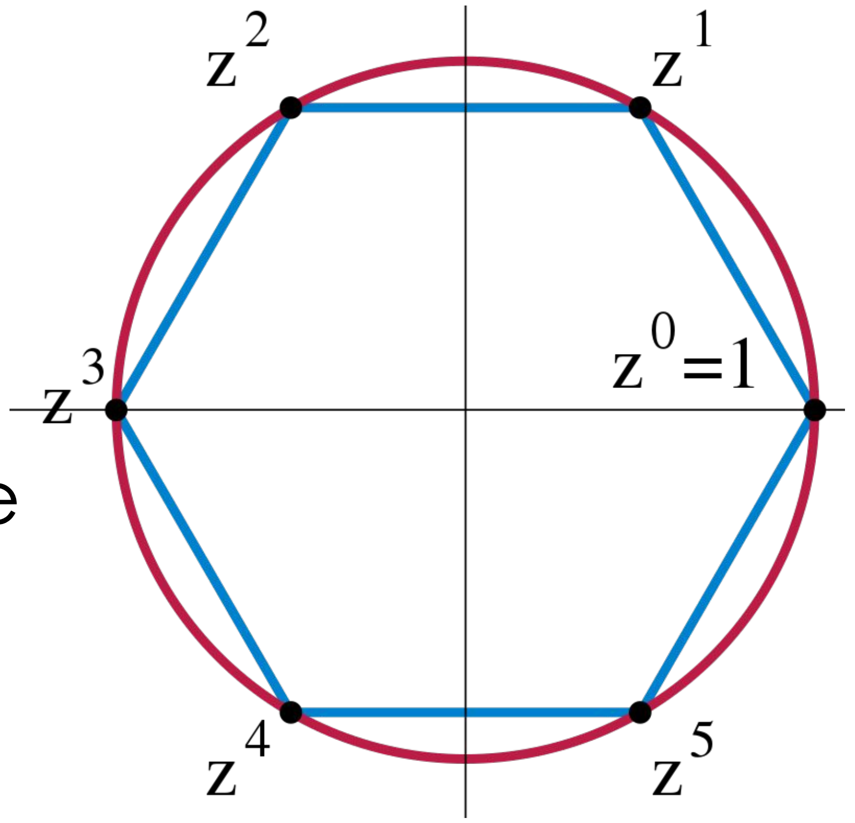
SO(2) Special Orthogonal Group in 2 dimensions

- Rotation

Almost there, but let's make it discrete

C_N Cyclic Group

- Rotations by $k2\pi/N$ for non-negative integer $k < N$



Representations

Groups are abstract, similar to Euclid's notion of a point - "A point is that which has no Part or Magnitude"

(David Hilbert's notion of a point if you insist on such rigor)

Representations makes these abstractions concrete, similar to Descartes notion of a point - "(x,y,z)"

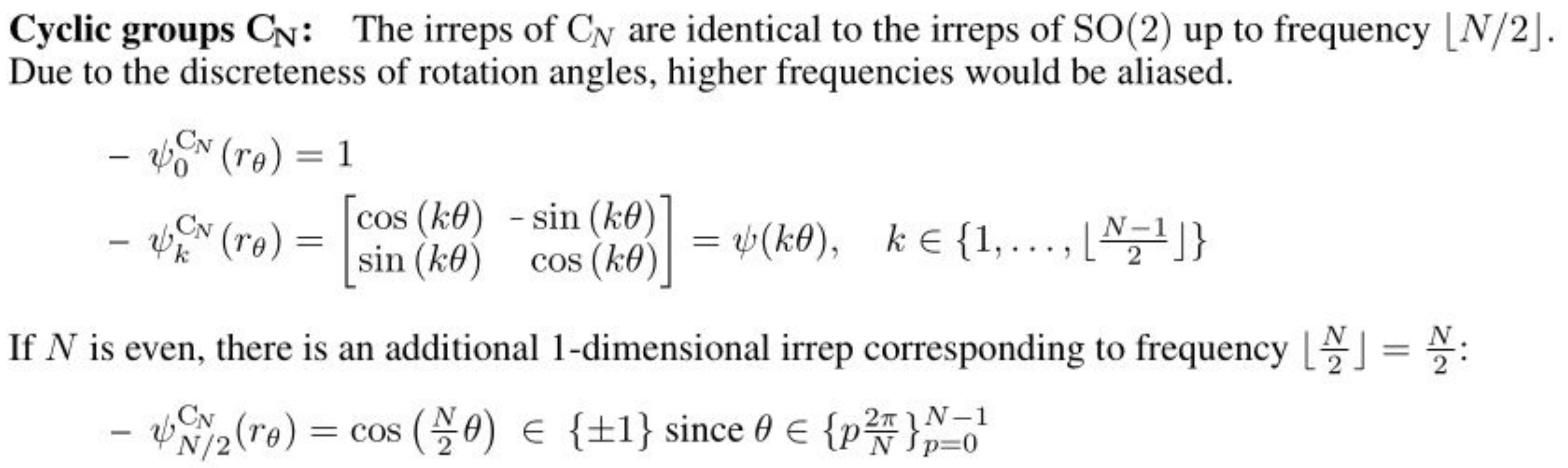
The representations are composed from irreducible representations, or irreps

Frans Hals - Portret van René Descartes, the father of analytic geometry

Representations

These geometric abstractions, rotation, reflection, etc., need some representation, in order for the groups to 'act'

For the groups associated with rotations, SO(2) and C(N), these representations are familiar, e.g. rotation matrices



Cyclic groups $\mathrm{C_N}$: The irreps of C_N are identical to the irreps of $\mathrm{SO}(2)$ up to frequency $\lfloor N/2 \rfloor$. Due to the discreteness of rotation angles, higher frequencies would be aliased.

- $\psi_0^{\mathrm{C}_N}(r_\theta) = 1$
- $\psi_k^{\mathrm{C}_N}(r_\theta) = \begin{bmatrix} \cos(k\theta) & -\sin(k\theta) \\ \sin(k\theta) & \cos(k\theta) \end{bmatrix} = \psi(k\theta), \quad k \in \{1, \dots, \lfloor \frac{N-1}{2} \rfloor\}$

If N is even, there is an additional 1-dimensional irrep corresponding to frequency $\lfloor \frac{N}{2} \rfloor = \frac{N}{2}$:

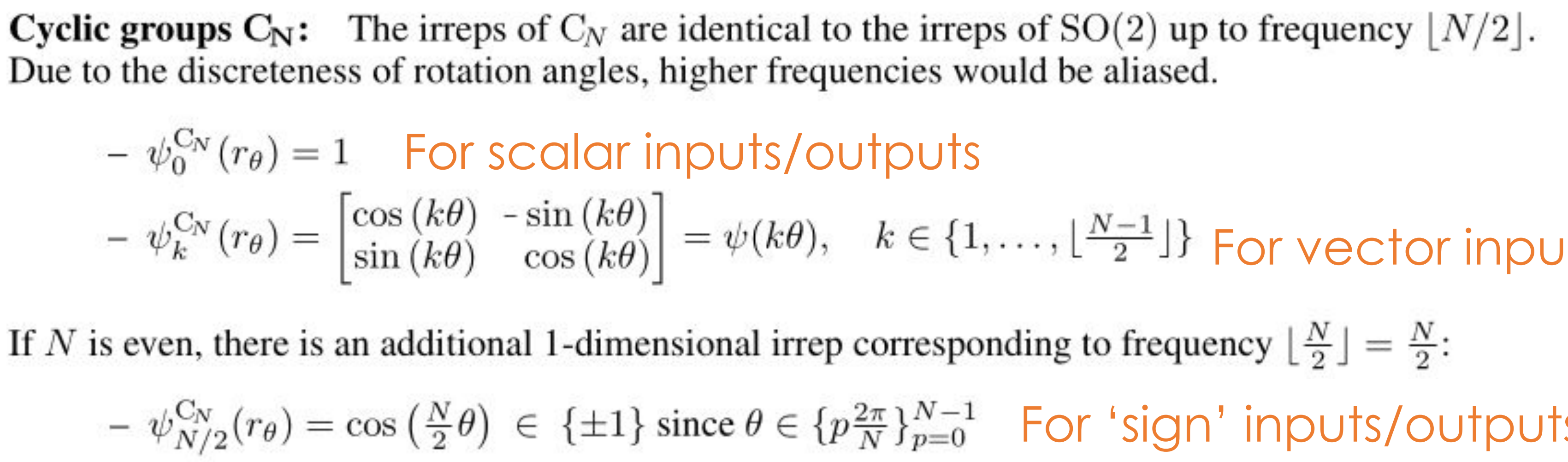
- $\psi_{N/2}^{\mathrm{C}_N}(r_\theta) = \cos\left(\frac{N}{2}\theta\right) \ \in \ \{\pm 1\}$ since $\theta \in \{p\frac{2\pi}{N}\}_{p=0}^{N-1}$

Representations

See the subscript on Psi below?

That is the input of e2cnn.gspaces.Rot2dOnR2(N).irrep(k)

which you will need to specify in your model init



Cyclic groups $\mathrm{C_N}$: The irreps of C_N are identical to the irreps of $\mathrm{SO}(2)$ up to frequency $\lfloor N/2 \rfloor$. Due to the discreteness of rotation angles, higher frequencies would be aliased.

- $\psi_0^{\mathrm{C}_N}(r_\theta) = 1$ For scalar inputs/outputs
- $\psi_k^{\mathrm{C}_N}(r_\theta) = \begin{bmatrix} \cos(k\theta) & -\sin(k\theta) \\ \sin(k\theta) & \cos(k\theta) \end{bmatrix} = \psi(k\theta), \quad k \in \{1, \ldots, \lfloor \frac{N-1}{2} \rfloor\}$ For vector inputs/outputs

If N is even, there is an additional 1-dimensional irrep corresponding to frequency $\lfloor \frac{N}{2} \rfloor = \frac{N}{2}$:

- $\psi_{N/2}^{\mathrm{C}_N}(r_\theta) = \cos\left(\frac{N}{2}\theta\right) \in \{\pm 1\}$ since $\theta \in \{p\frac{2\pi}{N}\}_{p=0}^{N-1}$ For 'sign' inputs/outputs, example in my notebook

Representations

The features of hidden layers do not relate to these familiar geometric representations, so don’t have irreps

Instead, they have the ‘regular’ representations

The bases of this representation, for the cyclic group, is

ϕ	0	$\frac{\pi}{2}$	π	$\frac{3\pi}{2}$
$\rho_{\text{reg}}^{C_4}(\phi)$	$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$	$\begin{bmatrix}0&0&0&1\\1&0&0&0\\0&1&0&0\\0&0&1&0\end{bmatrix}$	$\begin{bmatrix}0&0&1&0\\0&0&0&1\\1&0&0&0\\0&1&0&0\end{bmatrix}$	$\begin{bmatrix}0&1&0&0\\0&0&1&0\\0&0&0&1\\1&0&0&0\end{bmatrix}$

i.e. permutation matrices

My (potentially wrong) understanding is that every neuron is actually N values and their N permutations for C(N)

Follow-along demo conclusions

Wow, that was a blast!

Thanks for all the enthusiastic participation

Discussion

My ideas on interesting discussion topics:

Are there other symmetries, other than horizontal rotation, relevant to geophysical problems for which we could use the e2cnn, escnn, or other libraries for steerable NNs?

Does coriolis break this symmetry?

Are steerable CNNs worth the trouble?

- I hope I have demonstrated that it isn't too hard to implement, but data augmentation actually works pretty well, and we can't have a vector on a staggered grid, so idk, maybe not worth it

When do we need these hard constraints and when can data augmentation, physical loss terms, or other 'soft' constraints suffice?