



Particle Trajectory Reconstruction and Euclidian Equivariant Neural Networks

Omar Alterkait

Neutrino Physics and Machine Learning

25/06/234



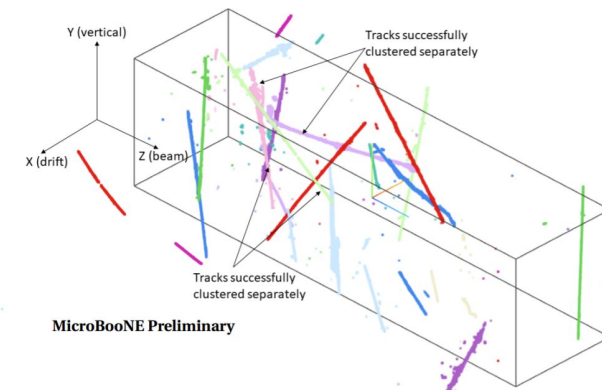
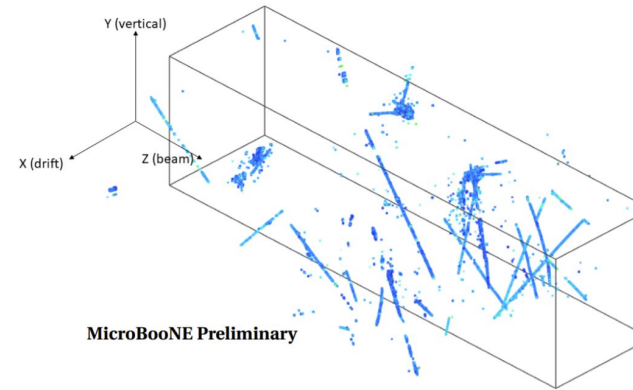
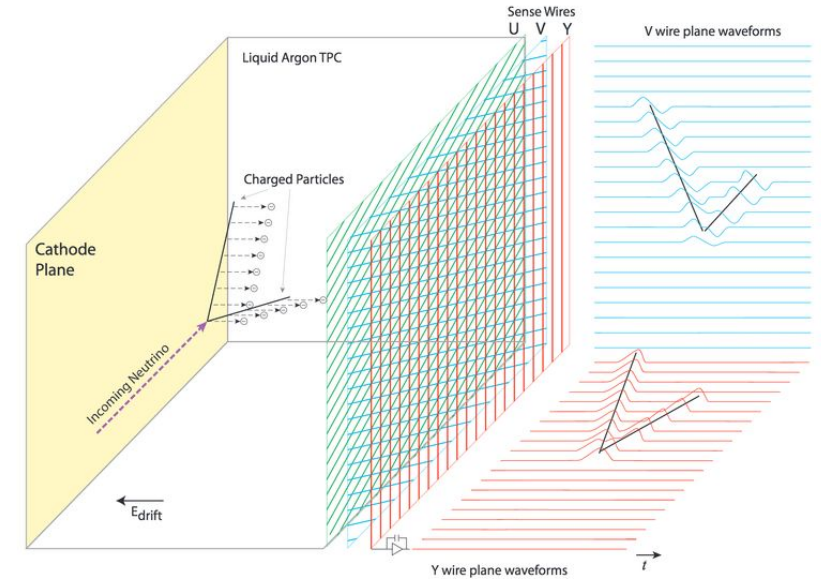
Outline

- **Introduce Track Reconstruction**
- **Start by giving a background on equivariant neural networks**
 - What is equivariance? Why is it useful?
- **Discuss Sparse Euclidean Equivariant CNNs**
 - What parts differ from normal CNNs?
- **Conclude about results and the next parts of the project**

Track Reconstruction

LArTPC Track Reconstruction

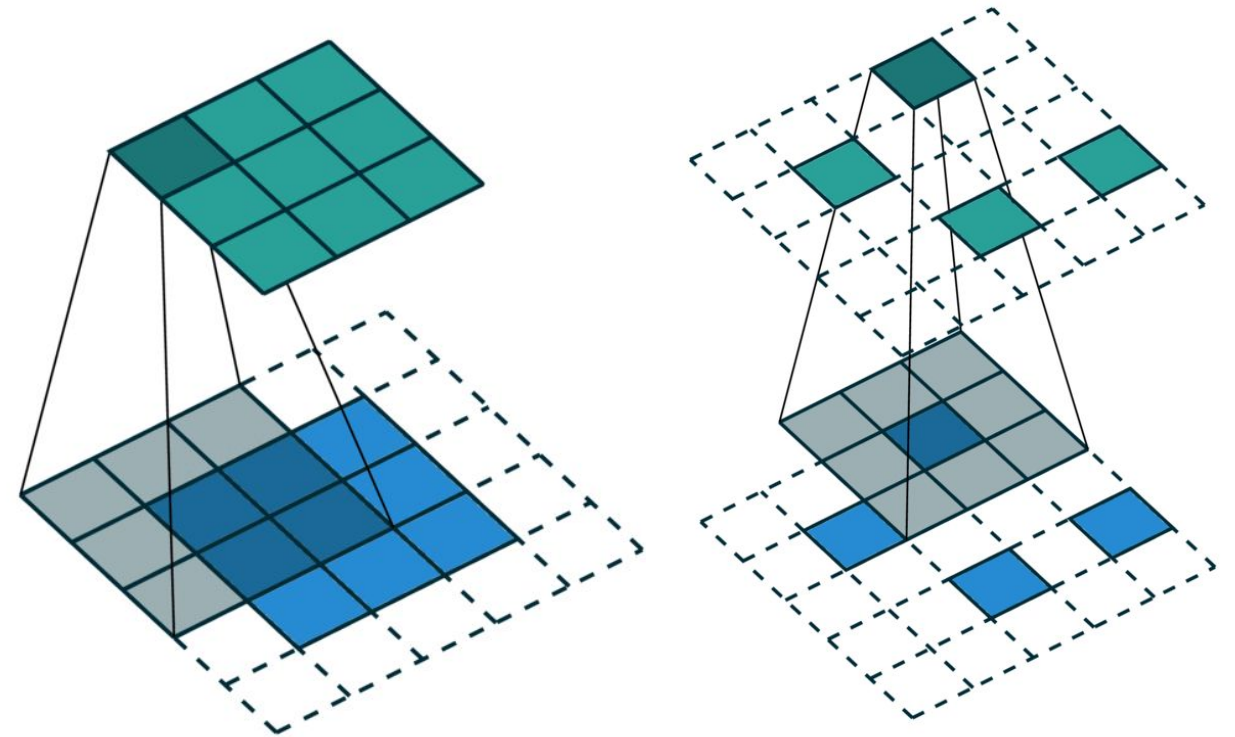
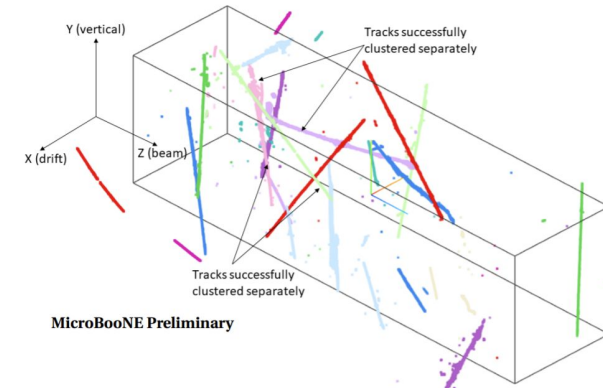
- As particles move through the detector, they leave charge behind.
- This charge is detected by wire planes
- Current ML/ non-ML tools exist to go from 2D to 3D.
- Following that, we would like to reconstruct, segment, and classify all the particles in the event.



[MICROBOONE-NOTE-1040-PUB](#)

Submanifold Convolutions

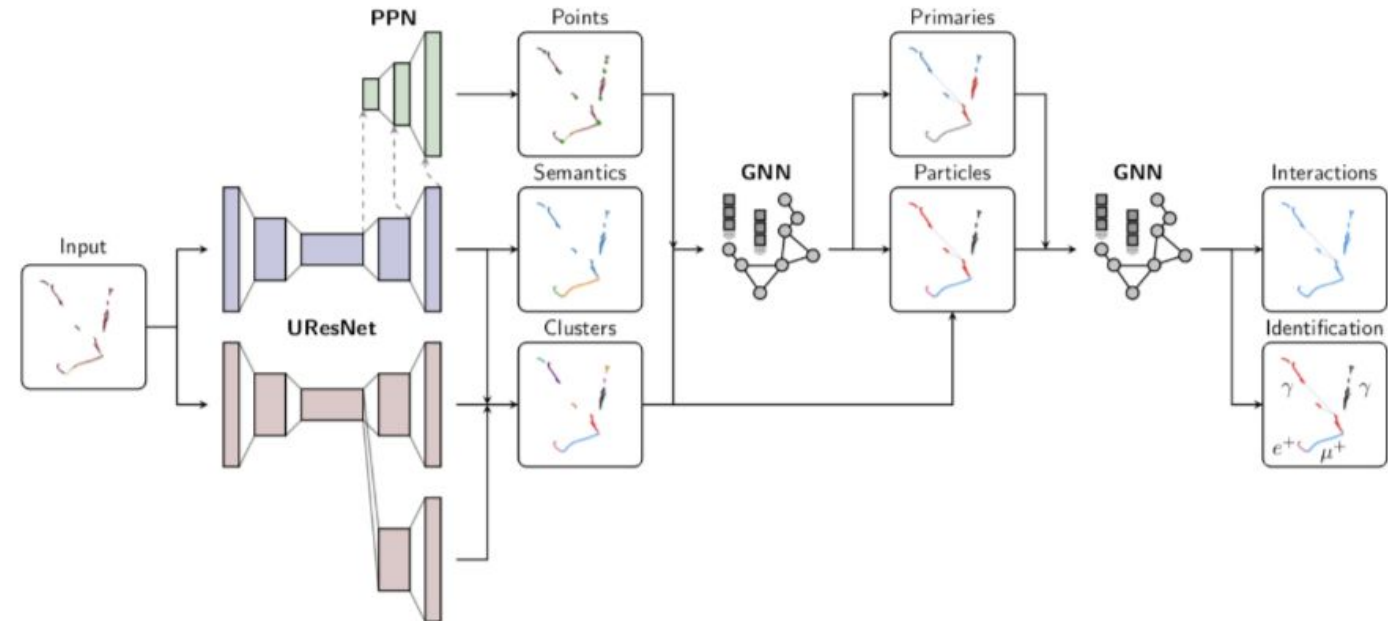
- For sparse but locally dense data, we can reduce computation by using submanifold convolutions
- This skips all empty spots during training/inference
- A package by NVIDIA called MinkowskiEngine is already employed in ML reconstruction toolchains



[MinkowskiEngine](#)

Reconstruction Pipeline

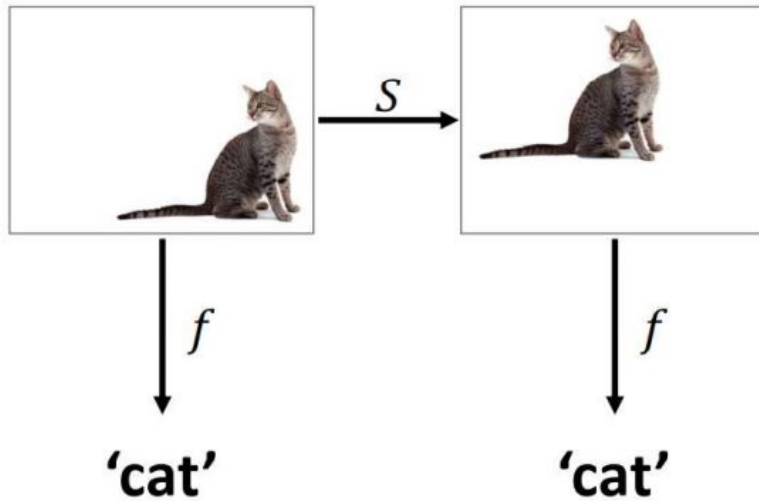
- Current ML tools are used for end-to-end reconstruction.
- These models are big and not trivial to train.
- Could we make them more efficient and robust?
- Can we reduce the amount of data we need?



Equivariant Neural Networks

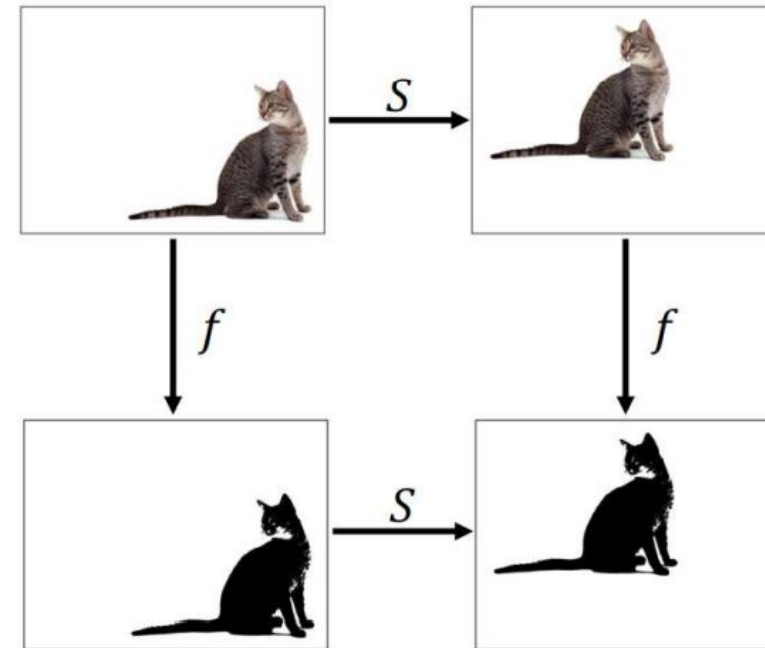
Invariance vs Equivariance

Invariance



$$f(x) = f(Sx)$$

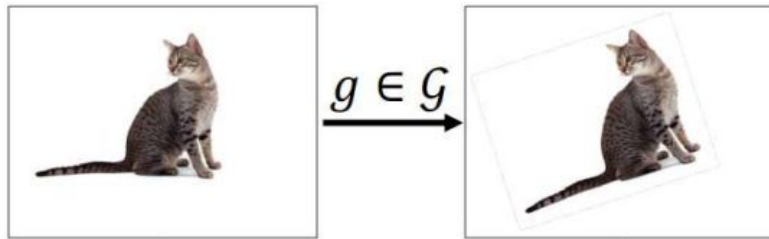
Equivariance



$$Sf(x) = f(Sx)$$

Invariance vs Equivariance

Invariance

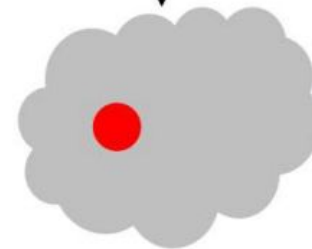
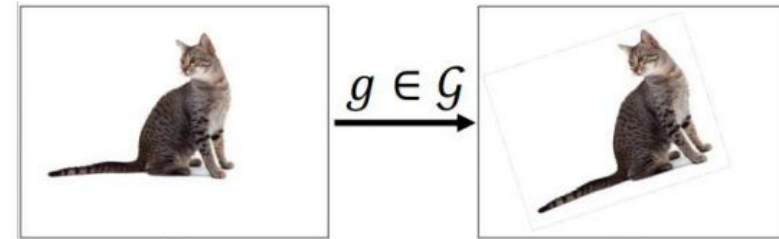


'cat'

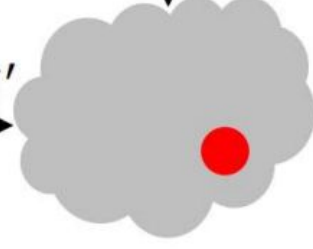
'cat'

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

Equivariance



$g' \in \mathcal{G}'$



$$g' f(x) = f(gx)$$

E(3) Equivariance

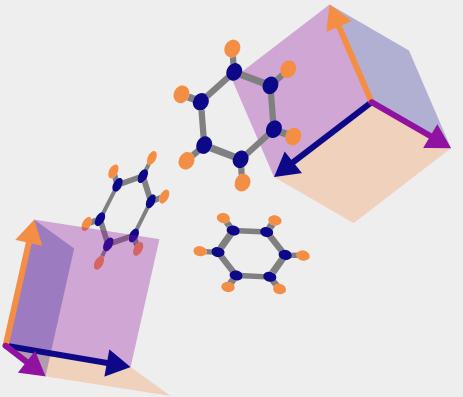
The euclidean group is a combination of all rotations and translations

$$\mathbf{E}(n) = \mathbf{T}(n) \times \mathbf{O}(n)$$

This covers the transformations we would use to convert between coordinate systems.

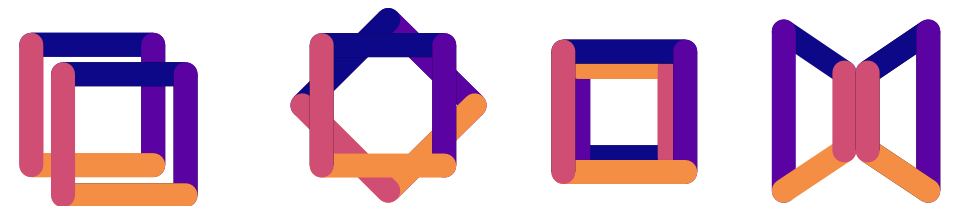
3D physical data
⇒ *Euclidean NN*

Data in 3D Euclidean space.
Freedom to choose coordinate system.



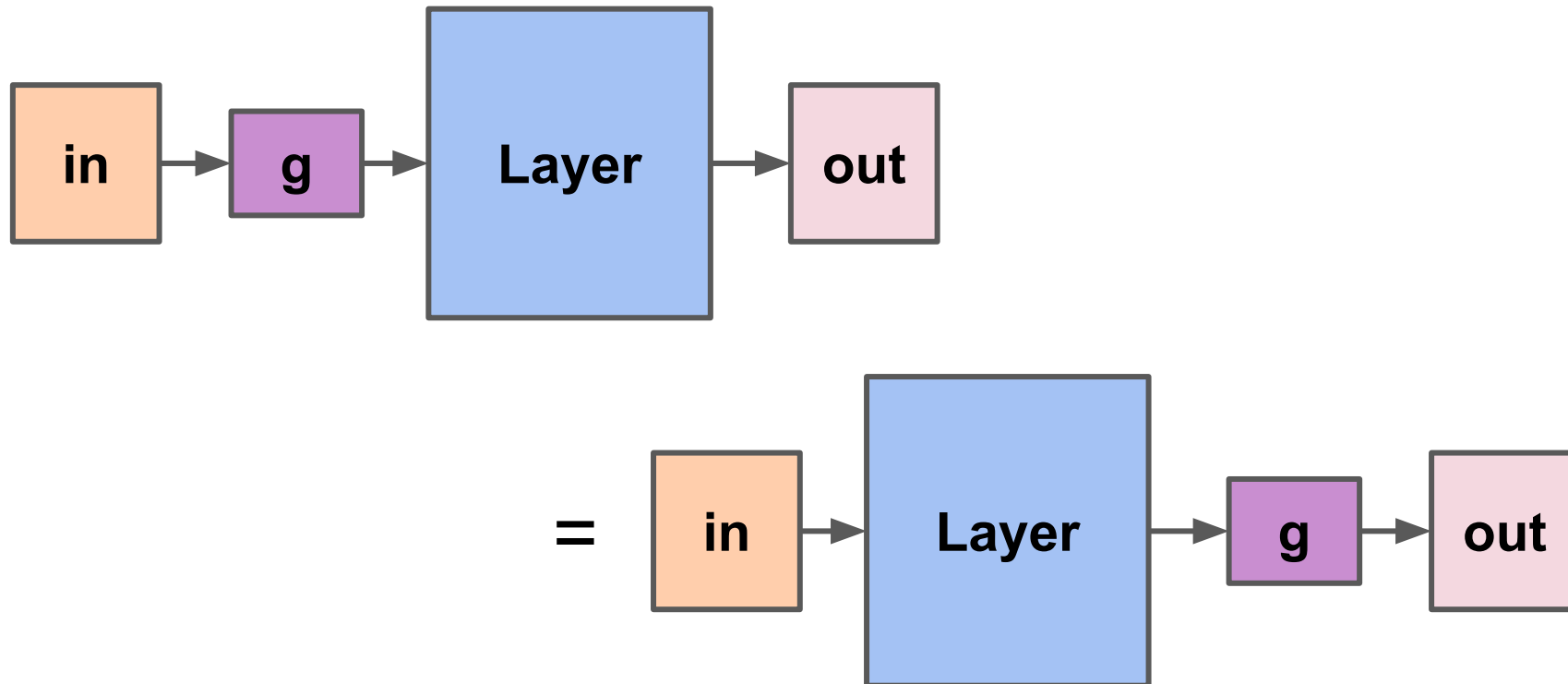
Euclidean Transformations

3D Translations 3D Rotations 3D Inversion Mirrors



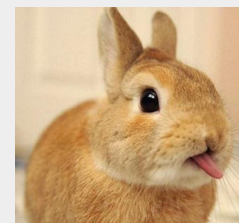
Approach 3: Equivariant Models

A g equivariant model is one where applying a group action g (in our case rotation) can be done before or after the layer is applied



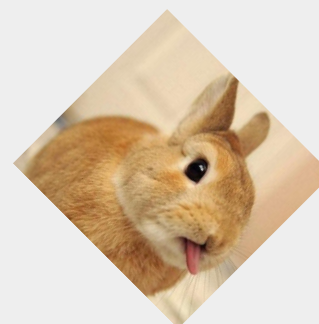
If our model learns one instance of the model, it can learn it for all different orientations

if ✓
then...



✓ translations

✓ mirrors
(rotation +
inversion)



✓ rotations

[Smidt - e3nn](#)

Equivariant Neural Networks

input learnable parameters predicted output

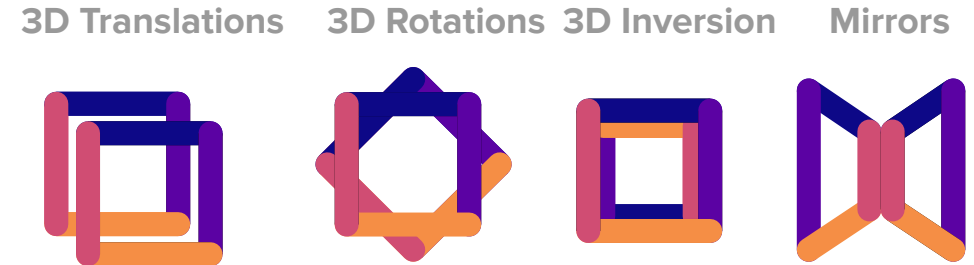
$$f(x, w) = y$$

g is an element of Euclidean symmetry

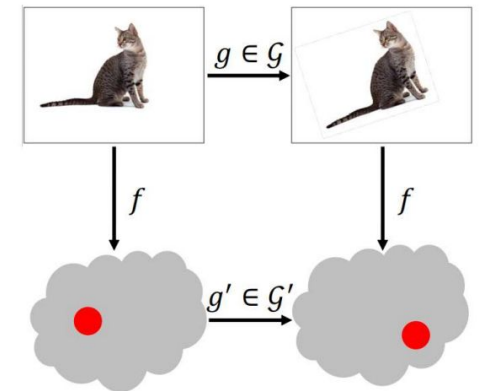
$$f(D(g)x, w) = D(g)f(x, w)$$

This restricts the possible functions to be tensors, and the only operations to be tensor algebra

Euclidean Transformations



Equivariance

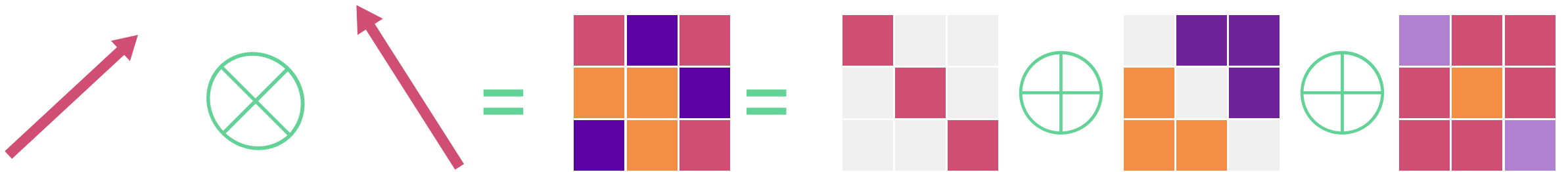


$$g' f(x) = f(gx)$$

invariant objects interact through scalar multiplication.



equivariant objects interact through tensor products



Generalizes to higher orders. Same mathematics that describes atomic interactions, e.g. selection rules in spectroscopy.

dot product
trace
invariant
L=0
1 degree of freedom

cross-product
antisymmetric
equivariant
L=1
3 degrees of freedom

symmetric
traceless
equivariant
L=2
5 degrees of freedom

e3nn

- A euclidean equivariant neural network package in jax/pytorch
- <https://github.com/e3nn/e3nn> | <https://e3nn.org> | <https://docs.e3nn.org>

e3nn

e3nn: a modular PyTorch framework for Euclidean neural networks

[View My GitHub Profile](#)

Welcome!

Getting Started

[How to use the Resources](#)

[Installation](#)

Help

Contributing

Resources

[Math that's good to know](#)

[e3nn_tutorial](#)

[e3nn_book](#)

[Papers](#)

[Previous Talks](#)

[Poster](#)

[Slack](#)

[Recurring Meetings / Events](#)

Calendar

e3nn Team

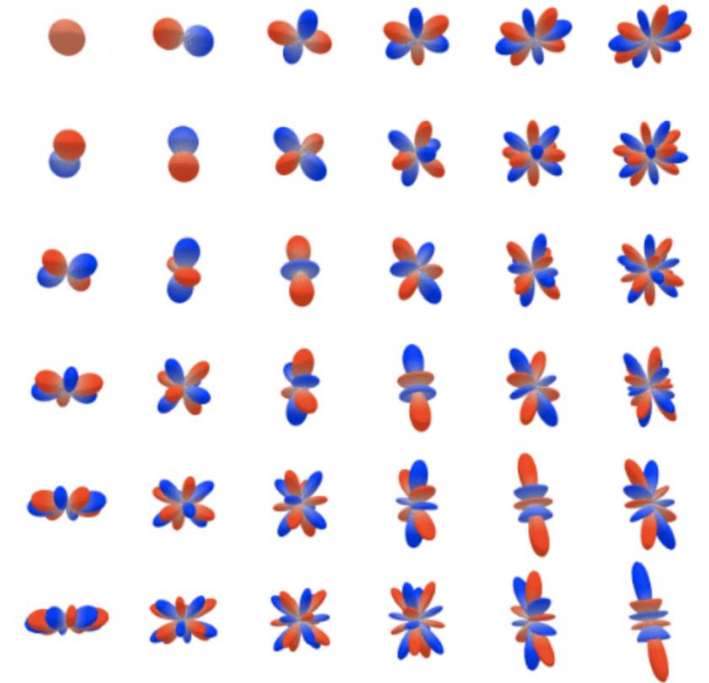
Welcome to e3nn!

This is the website for the e3nn repository

<https://github.com/e3nn/e3nn/>

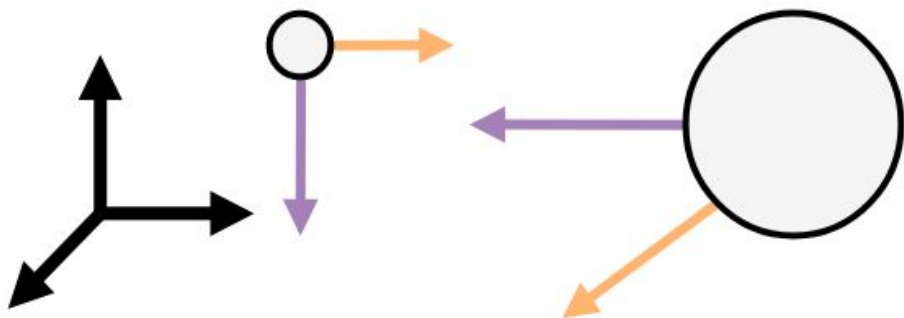
[Documentation](#)

$E(3)$ is the [Euclidean group](#) in dimension 3. That is the group of rotations, translations and mirror. e3nn is a pytorch library that aims to create $E(3)$ equivariant neural networks.



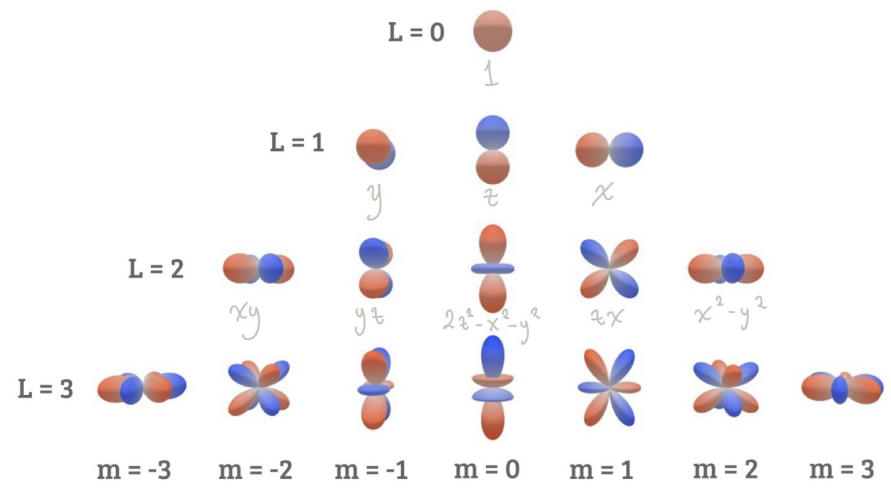
Hosted on GitHub Pages — Theme by orderedlist

The input to our network is geometry and features on that geometry.
 We categorize our features by how they transform under rotation and parity as *irreducible representations of $O(3)$* . [irreps]



```

geometry = [[x0, y0, z0], [x1, y1, z1]]
features = [
    [m0, v0x, v0y, v0z, a0x, a0y, a0z]
    [m1, v1x, v1y, v1z, a1x, a1y, a1z]
]
scalar = e3nn.o3.Irrep("0e") # L=0, even p
vector = e3nn.o3.Irrep("1o") # L=1, odd p
irreps = 1 * scalar + 1 * vector + 1 * vector
    
```



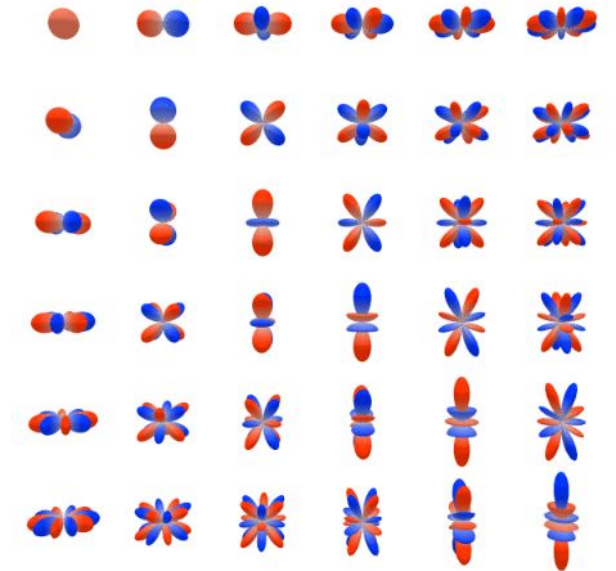
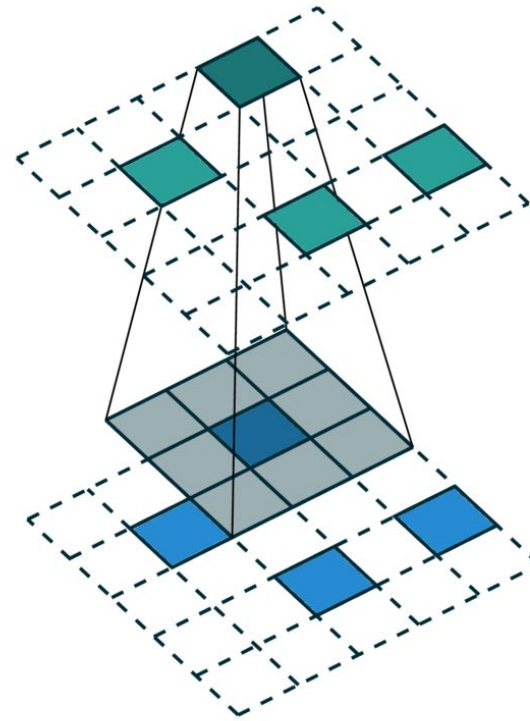
- All input, intermediate, and output data is “typed” by its transformation properties.
- All operations must respect this “typing”.

Sparse Equivariant Neural Networks

Sparse Euclidean Equivariant CNNs

Could we combine submanifold convolutions with euclidean equivariant NNs?

Yes, we can!



Recipe to make a Sparse Euclidean Equivariant CNN

- Equivariant Model Layers

Convolution

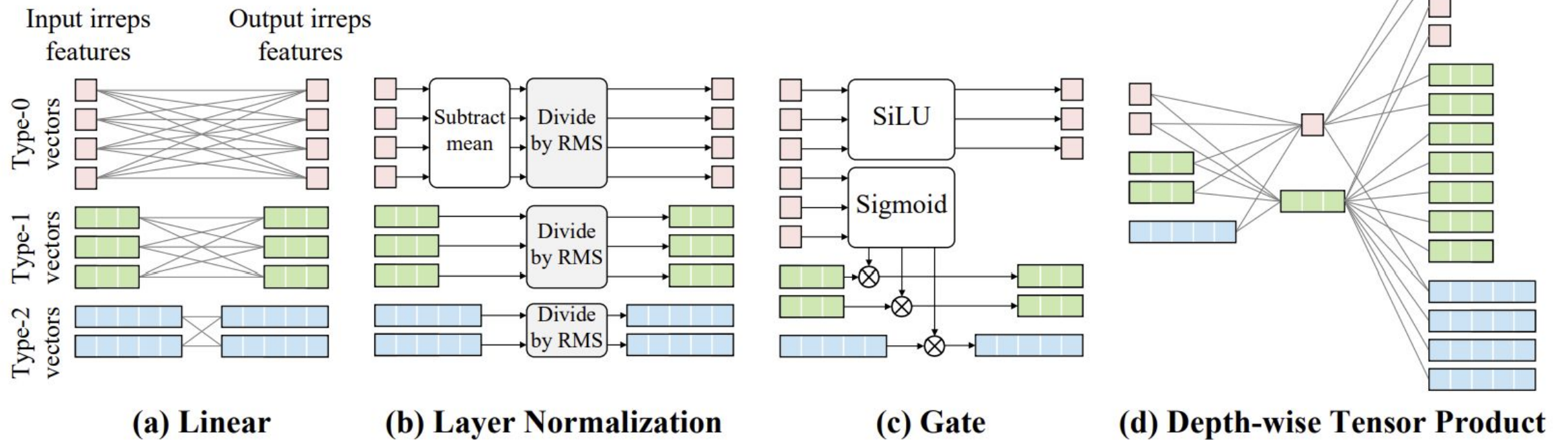
Layer Norm

Activation

Linear

Recipe to make a Sparse Euclidean Equivariant CNN

- Equivariant Model Layers



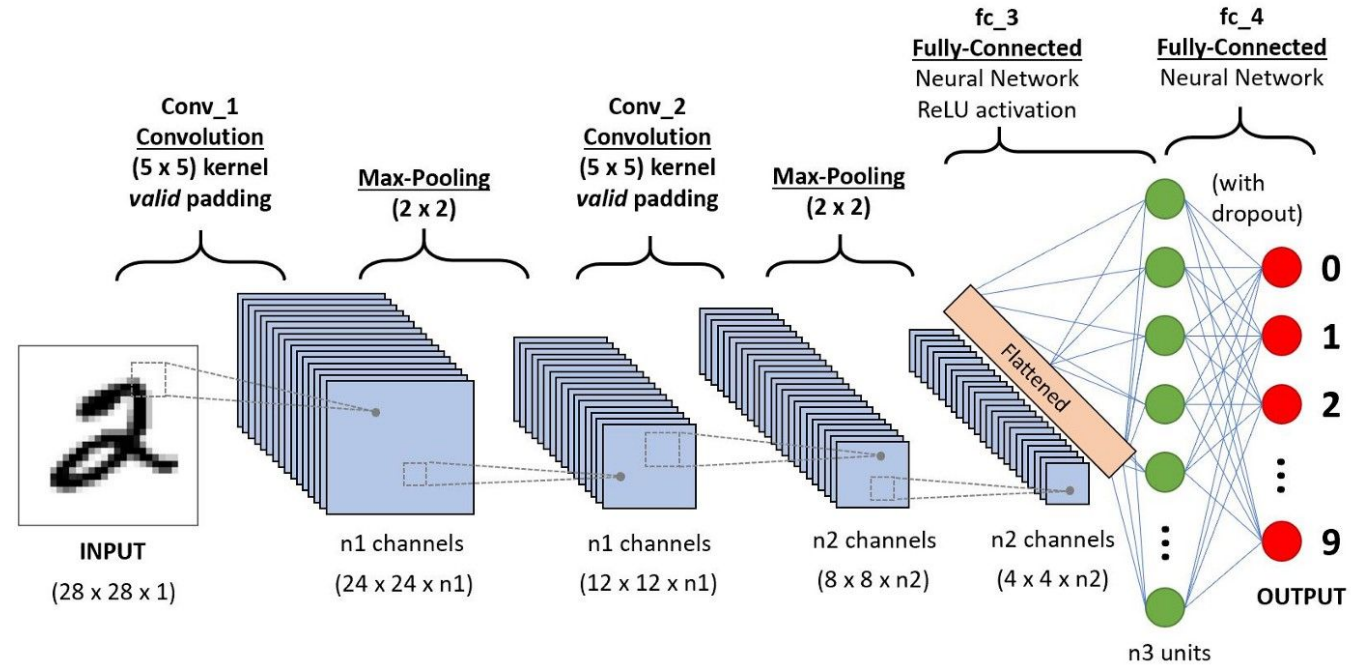
In standard image convolutions, filter depends on coordinate system.

convolutional neural networks:

Used for images. In each layer, scan over image with learned filters.



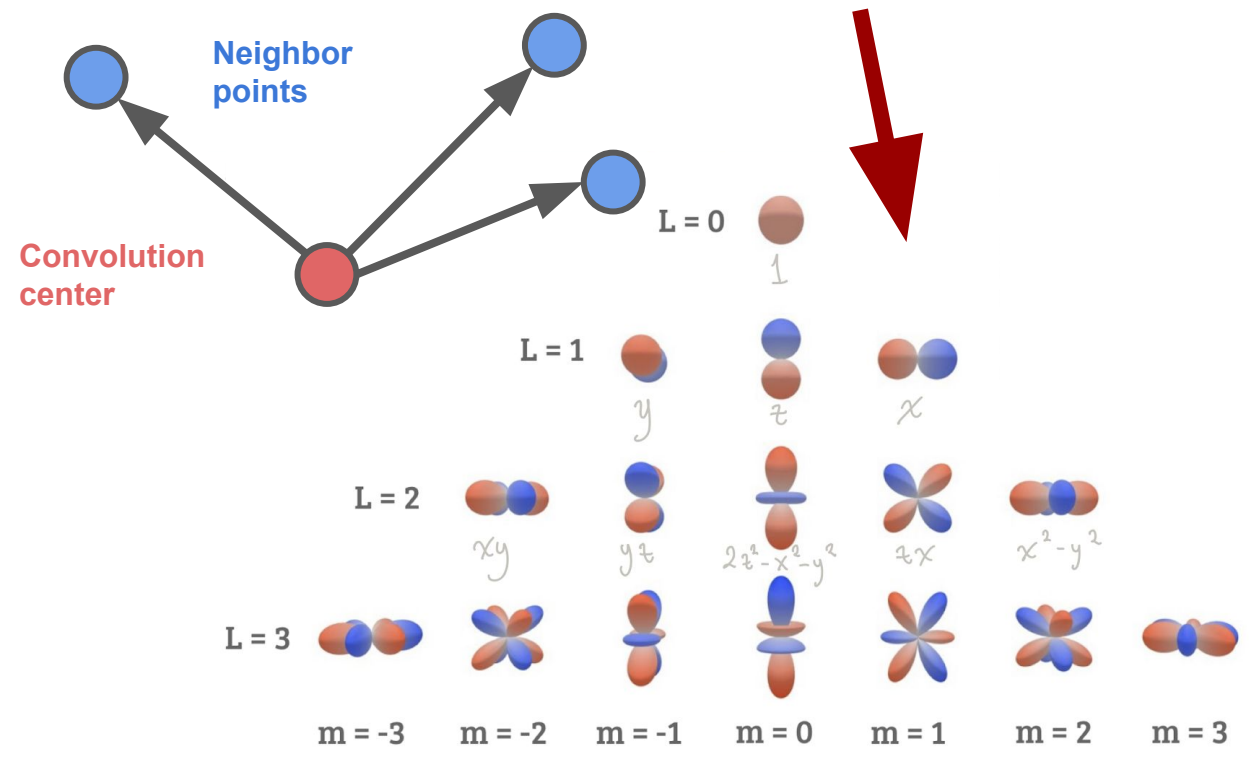
http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/



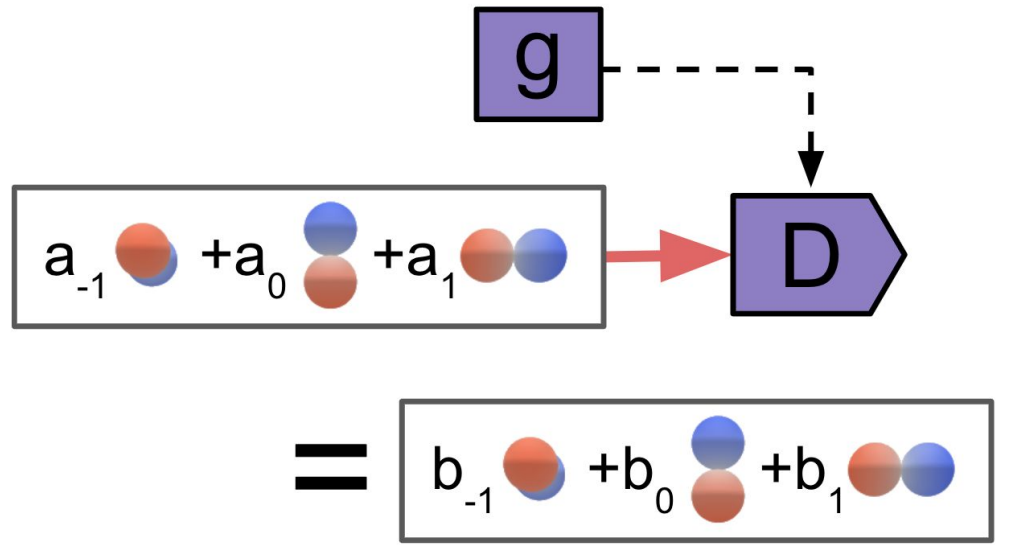
... and we require the convolutional filter to be equivariant.

Rotation equivariant convolutional filters are based on **learned radial functions** and **spherical harmonics**...

$$W(\vec{r}) = R(r) Y_l^m(\hat{r})$$



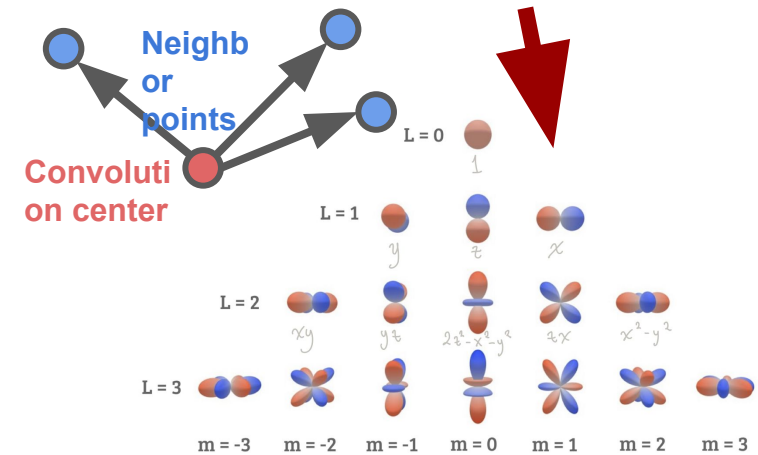
Spherical harmonics of the same L transform together under rotation **g**.



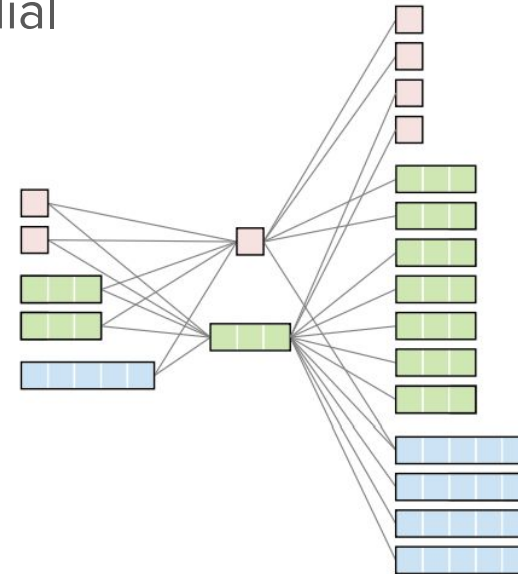
Spherical harmonics transform in the same manner as the irreducible representations of $SO(3)$.

Equivariant Convolutions on a Grid

$$W(\vec{r}) = R(r) Y_l^m(\hat{r})$$



- Instead of having an arbitrary kernel in normal CNNs, we take a combination of irreps through some learned radial function
- The convolution operations is what mixes different irreps.
- Convoluting on a grid limits our choices of radial parameterizations compared to point clouds.

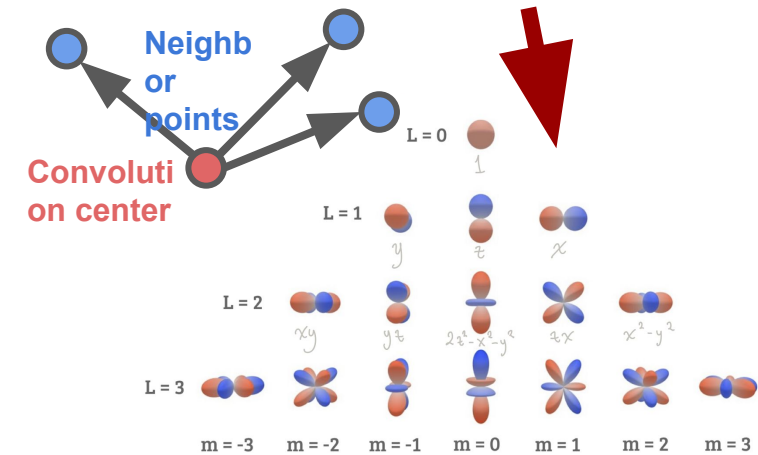


(d) Depth-wise Tensor Product

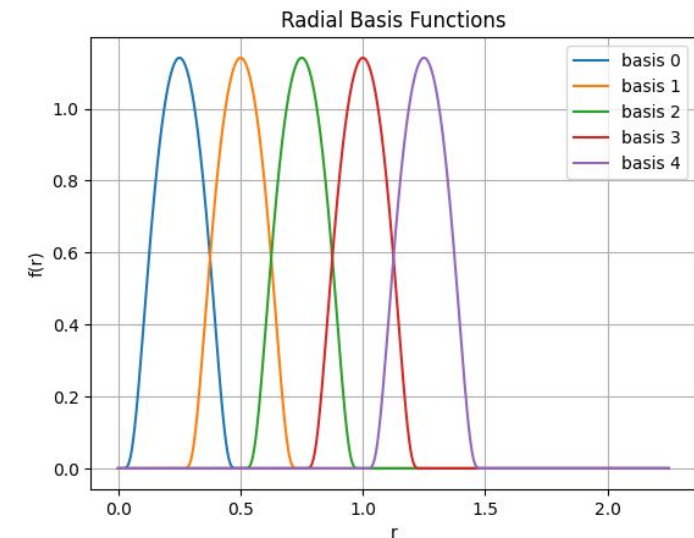
Convolution

Equivariant Convolutions on a Grid

$$W(\vec{r}) = R(r) Y_l^m(\hat{r})$$

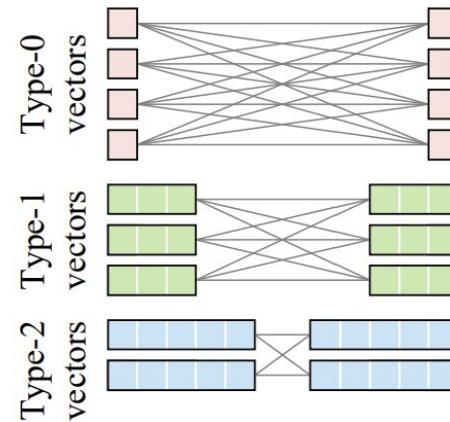


- We evaluate our 3x3 voxel kernel using a pre-set basis which gives us an embedding on the kernel grid.
- The weight of our convolution is given by (num_radial_basis, irreps_out.dim)
- Using this, we can now do a full equivariant convolution on a grid:
 - Define embedding on the grid
 - Place input irrep features on those voxel points weighted by the embeddings.
 - Do depthwise tensor product with spherical harmonics to mix different irreps

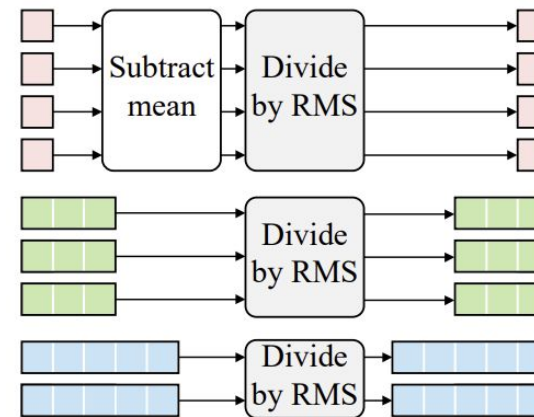


Recipe to make a Sparse Euclidean Equivariant CNN

- For channels of the same irrep, we can just use a fully connected network between them
- For layer normalization, we only subtract means from scalars. We then normalize by dividing over the RMS of all the same irreps



Linear

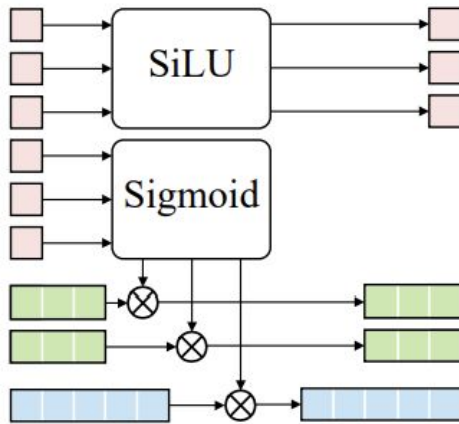


Layer Norm

Gate Activation

Activation

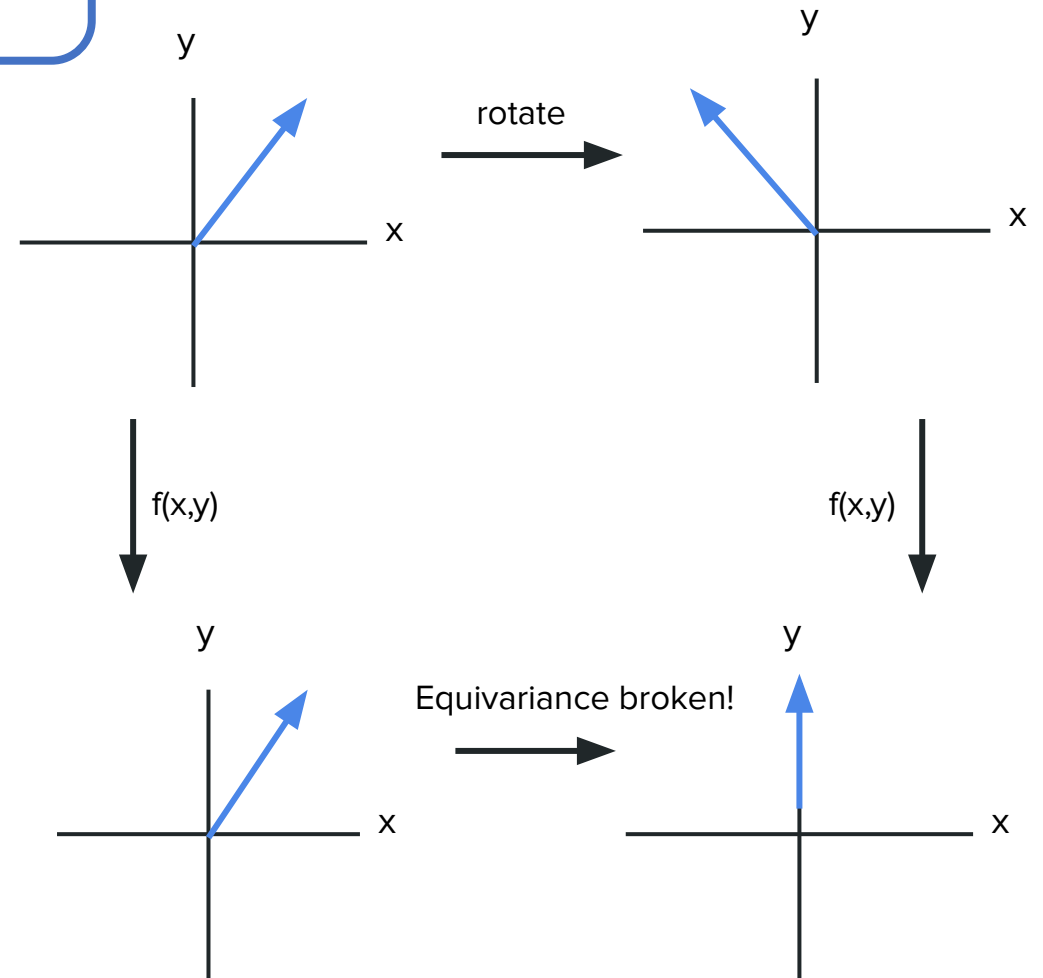
- We cannot simply take the non-linearity of an equivariant function as that breaks the equivariance
- We can take non-linearity of scalars but not anything higher order
- To solve this, we take the non-linearity of a scalar times the higher order irrep



(c) Gate

$$\left(\bigoplus_i \phi_i(x_i) \right) \oplus \left(\bigoplus_j \phi_j(g_j)y_j \right)$$

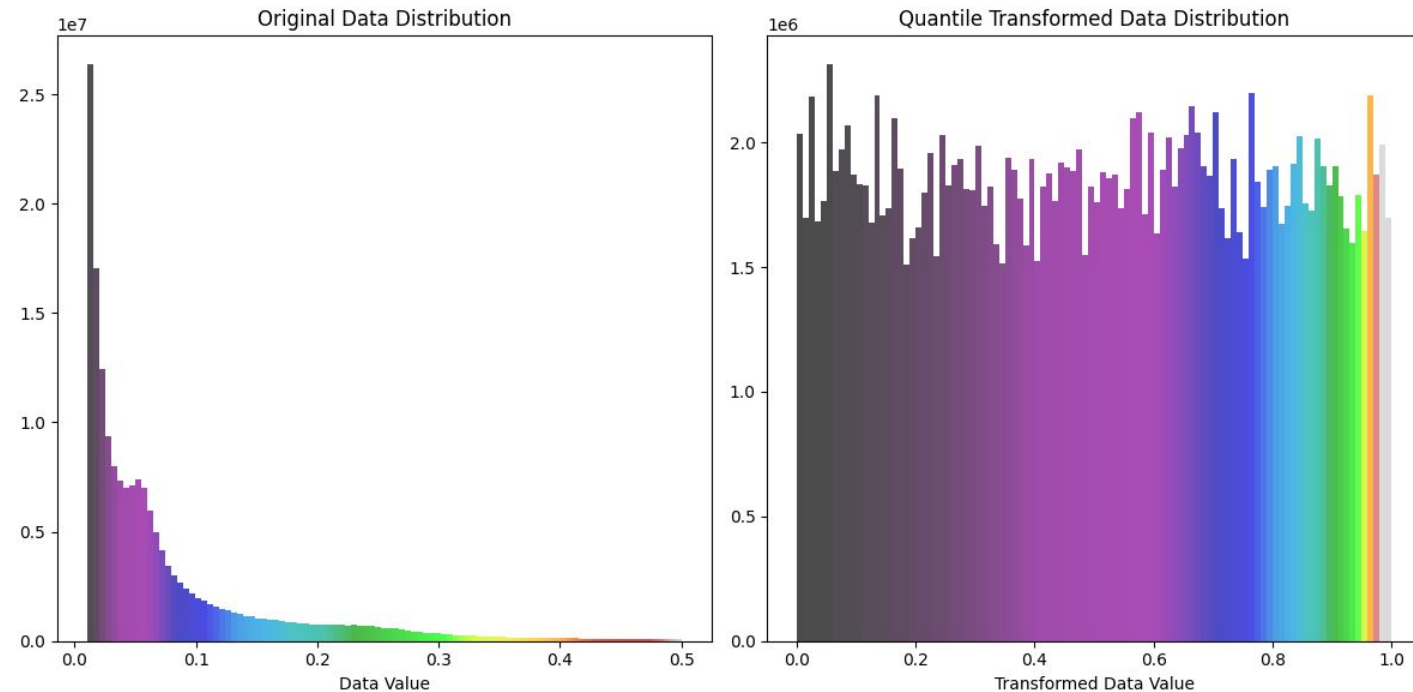
$$f(x,y) = [\text{ReLU}(x) , y]$$



Putting it all together

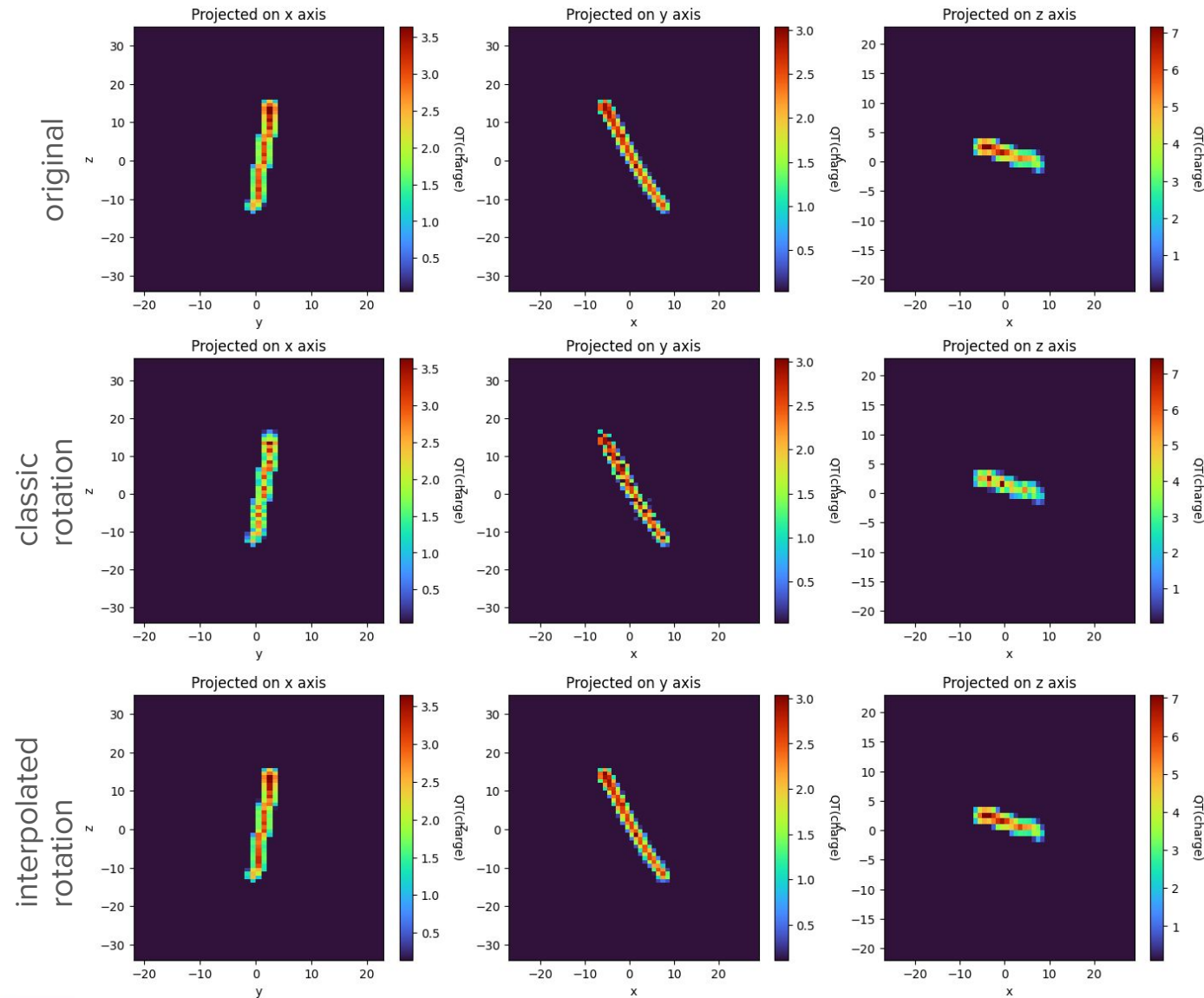
Quantile Transform

- Need some way to normalize our input data's charge
- This was originally used for training purposes. We run over the statistics of our distribution and then map it such that the i th-quantile is transformed to $i/100$.



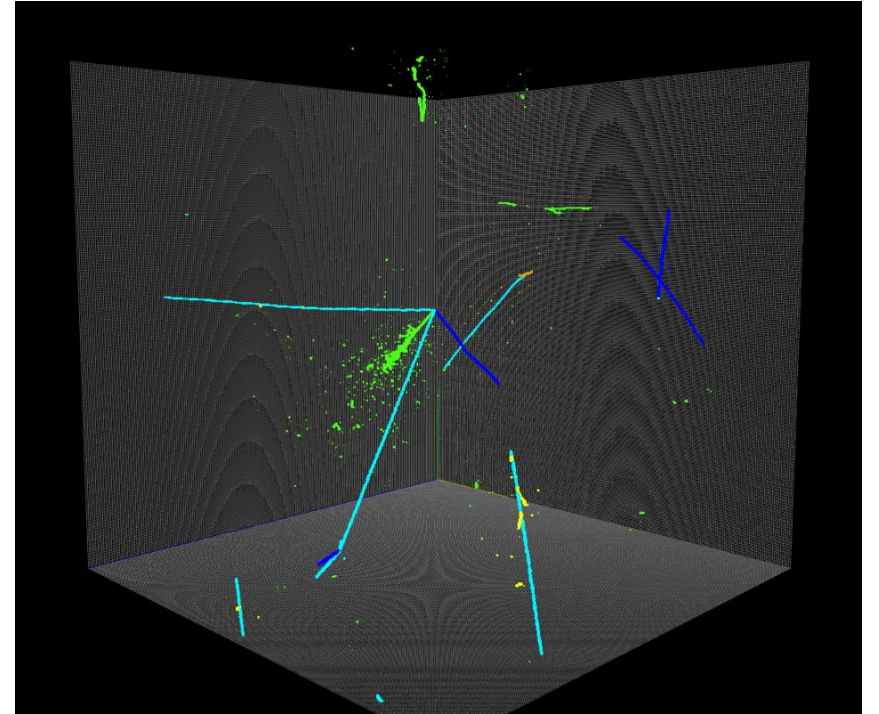
Rotations on a Grid

- Doing rotations on points clouds is simple. We can rotate and unrotate without any loss of generality.
- For voxels, we have to snap the points on the grid somehow. This introduces nonzero error at the beginning
- Only snapping to the nearest point creates holes
- Solve this by doing interpolation with a gaussian kernel (slight smoothing) before snapping



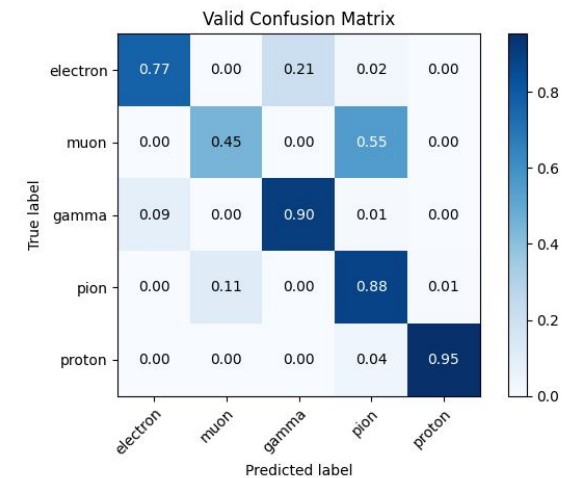
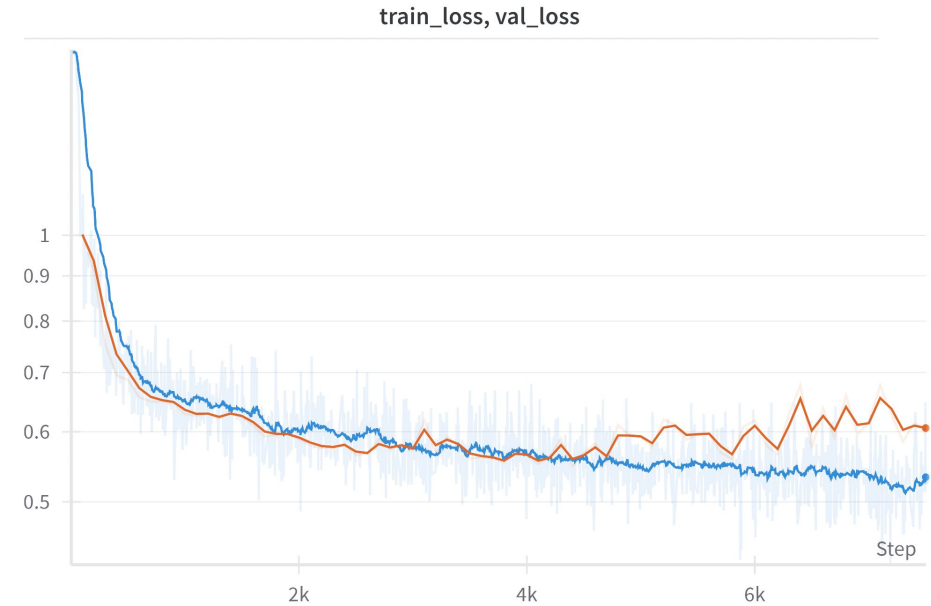
Training

- A public dataset (PILArNet) is used for the training/test data
- The equivariant model is trained on a baseline of single particle classification. This is an invariant task (easier for both the equivariant and the non-equivariant model to learn).
- Baseline: ResNet50 (46M parameters)
- Equivariant: (1M parameters)
- Time for forward and backward is on the same order (even with all the tensor products)



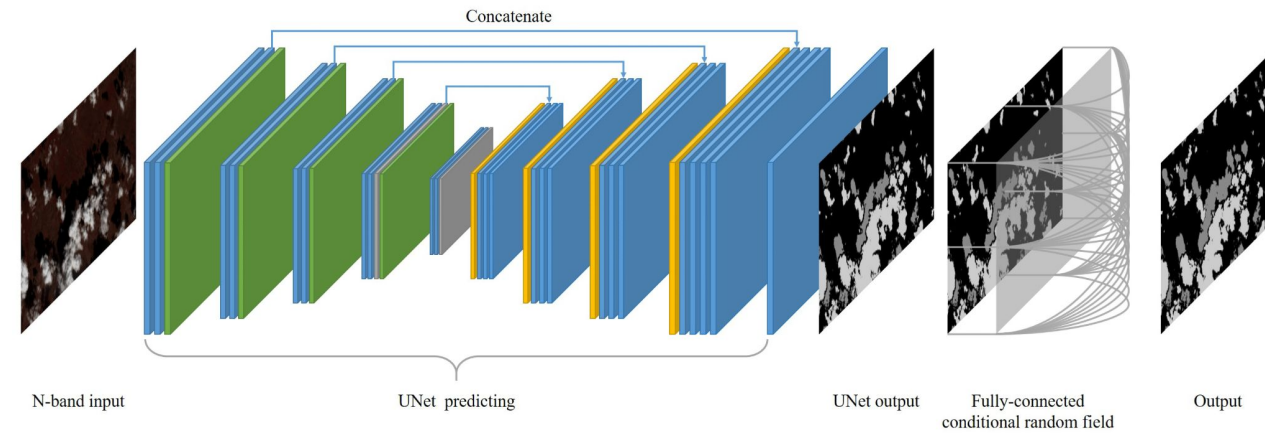
Results

- The equivariant model is capable of training.
- Training is limited now by model choices. Need more hyperparameter tuning to bridge the gap with the baseline.
- This data only contains single particles. Harder to differentiate pions and muons



Next Steps

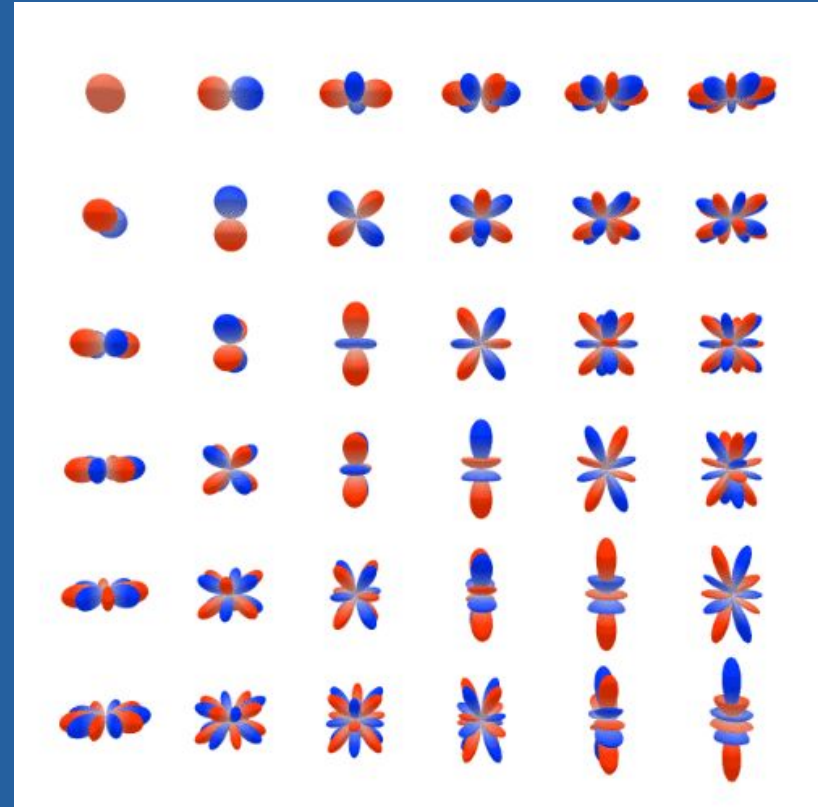
- Currently dissecting the model layer by layer to find where the equivariance problems are happening
- Try to go to a full UNet and test equivariance instead of invariance
- Instead of only semantic segmentation, try harder tasks such as keypoint detection



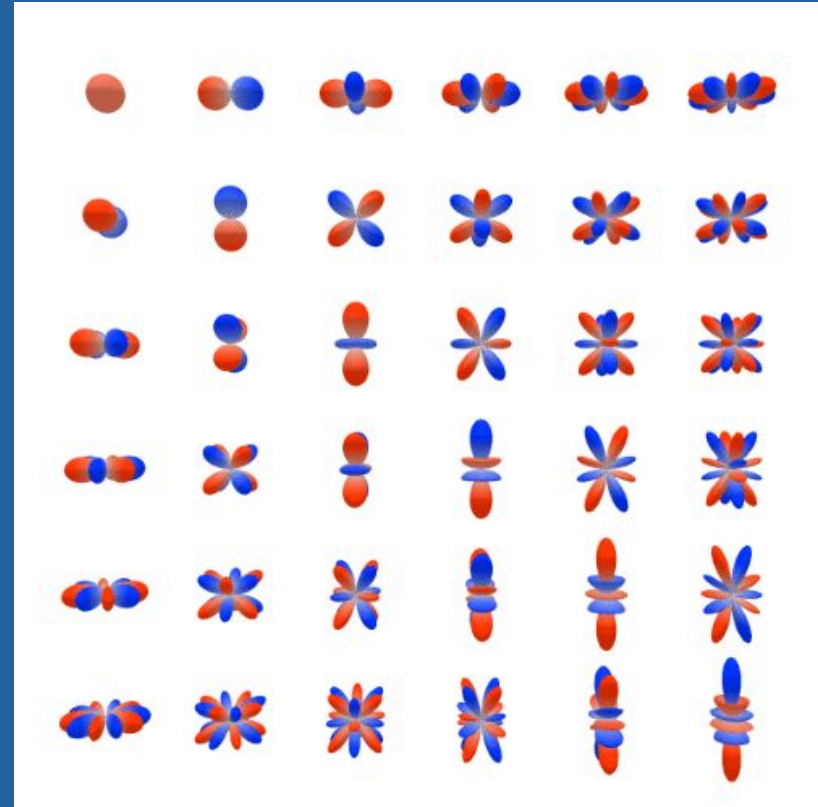
Conclusion/ Outlook

- Track Reconstruction is a difficult problem that could use new innovation
- Equivariant Neural Networks is a surging field with many possible implementations/improvements
- I have successfully built a sparse euclidean equivariant convolutional neural networks, and the next steps are to

Questions



Backup Slides

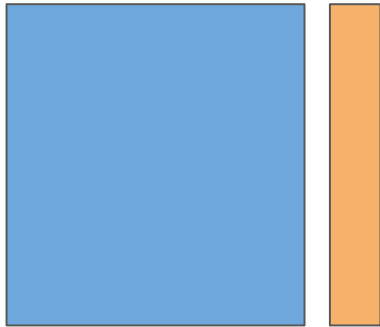


Neural networks are specially designed for different data types.
Assumptions about the data type are built into how the network operates.

W

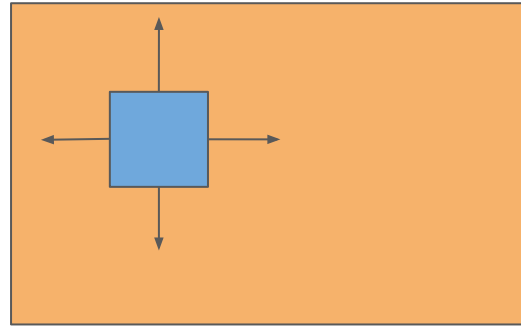
X

Arrays \Rightarrow Dense NN



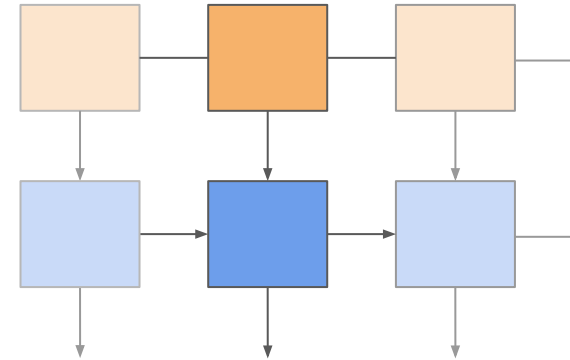
Components are independent.

2D images \Rightarrow Convolutional NN



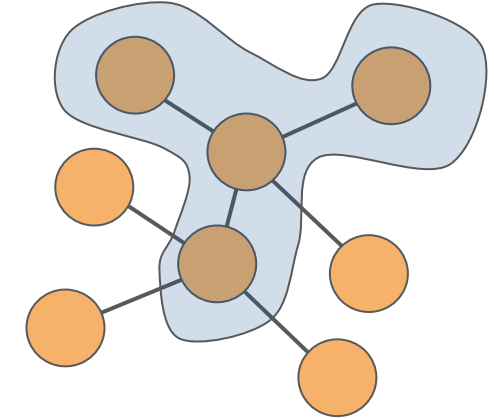
The same features can be found anywhere in an image. Locality.

Text \Rightarrow Recurrent NN



Sequential data. Next input/output depends on input/output that has come before.

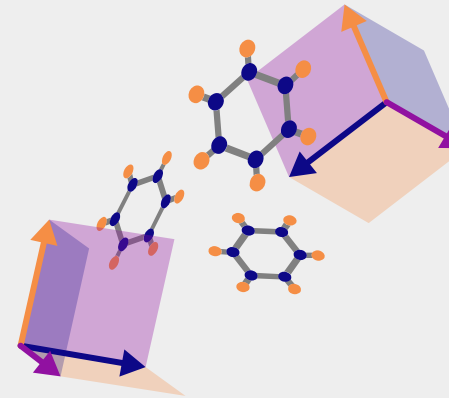
Graph \Rightarrow Graph (Conv.) NN



Topological data. Nodes have features and network passes messages between nodes connected via edges.

3D physical data
 \Rightarrow Euclidean NN

Data in 3D Euclidean space.
Freedom to choose coordinate system.

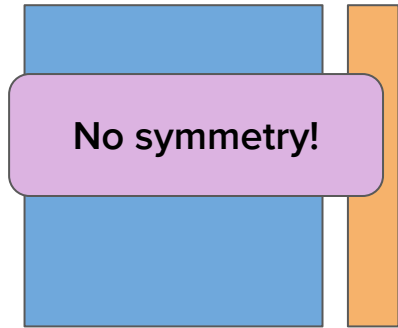


W

X

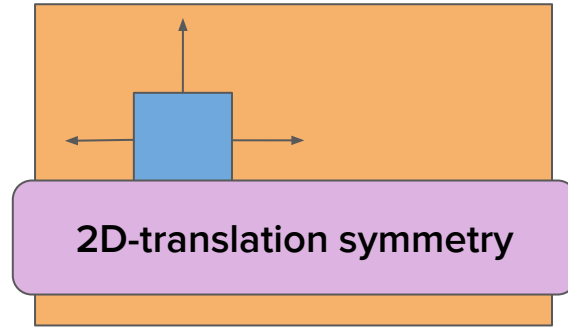
Neural networks are specially designed for different data types. Assumptions about the data type are built into how the network operates.

Arrays \Rightarrow Dense NN



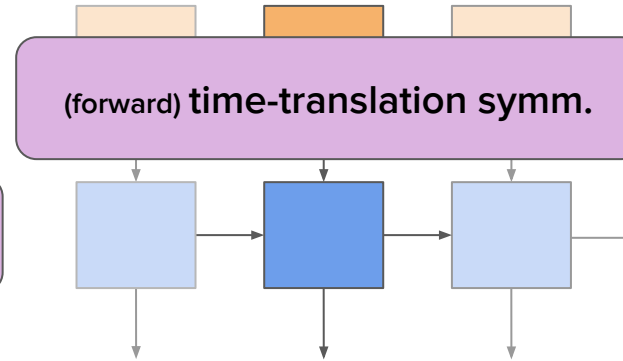
Components are independent.

2D images \Rightarrow Convolutional NN



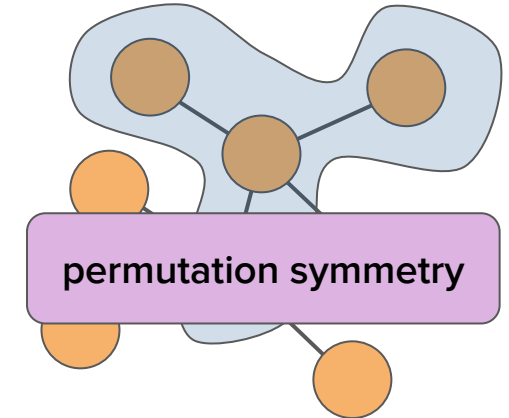
The same features can be found anywhere in an image. Locality.

Text \Rightarrow Recurrent NN



Sequential data. Next input/output depends on input/output that has come before.

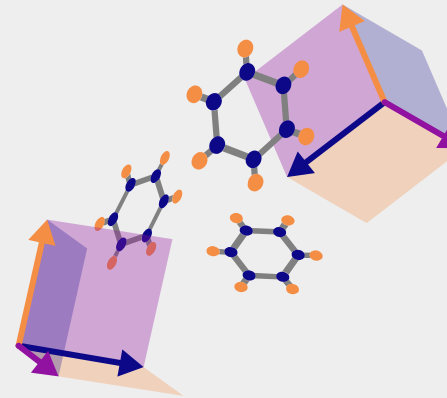
Graph \Rightarrow Graph (Conv.) NN



Topological data. Nodes have features and network passes messages between nodes connected via edges.

3D physical data \Rightarrow Euclidean NN

3D Euclidean symmetry E(3): 3D rotations, translations, and inversion

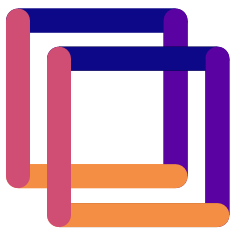


Making Models Symmetry-Aware

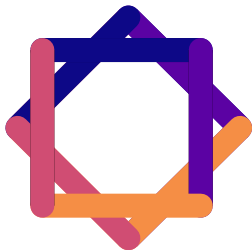
- Approach 1: Data Augmentation
 - Brute Force
- Approach 2: Invariant Models
 - Most current models
- Approach 3: Equivariant Models
 - Goal of today's talk

Euclidean Transformations

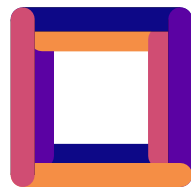
3D Translations



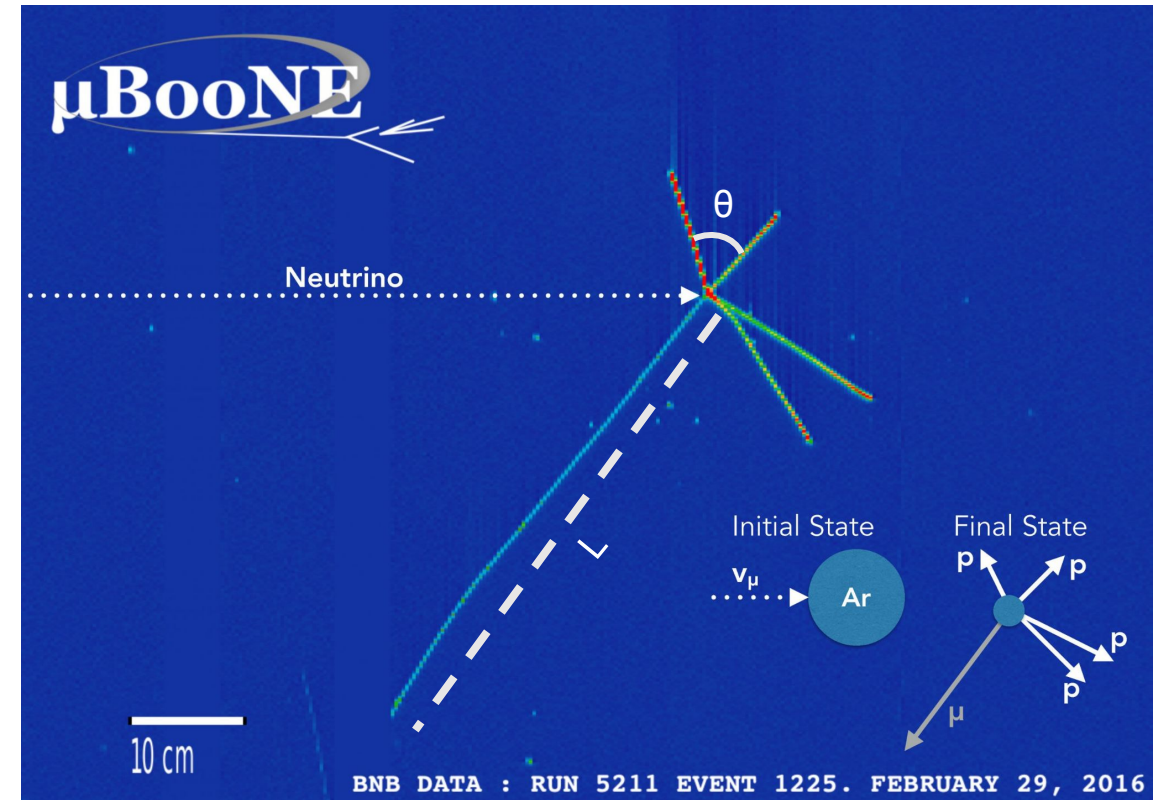
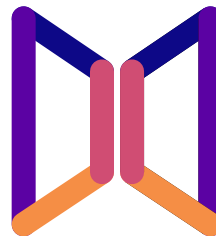
3D Rotations



3D Inversion



Mirrors

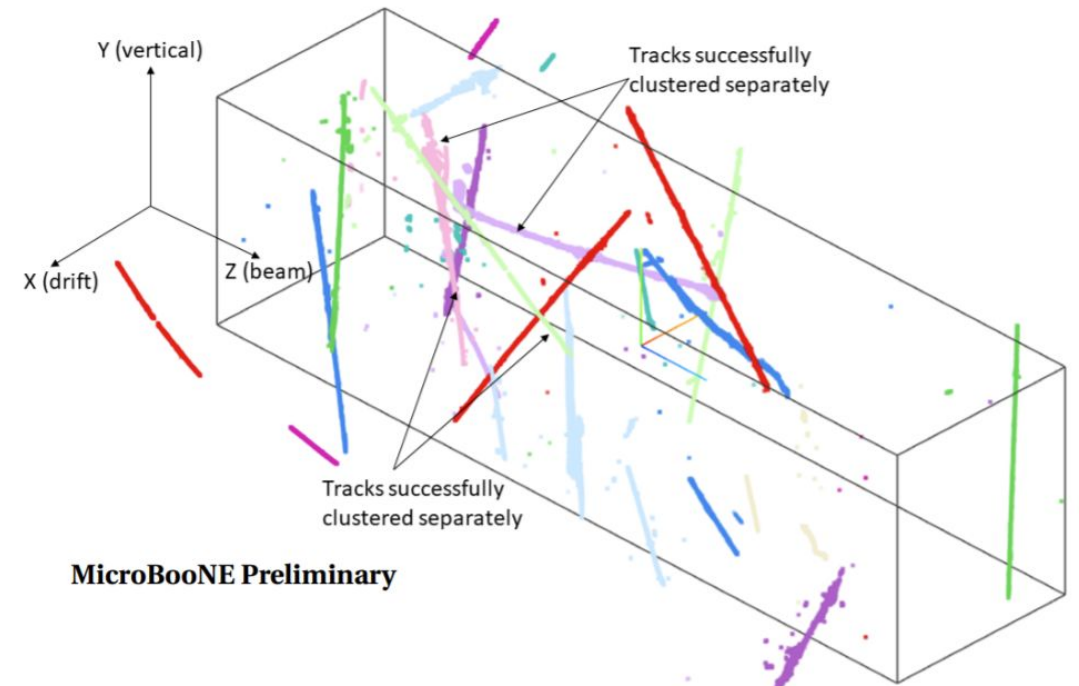
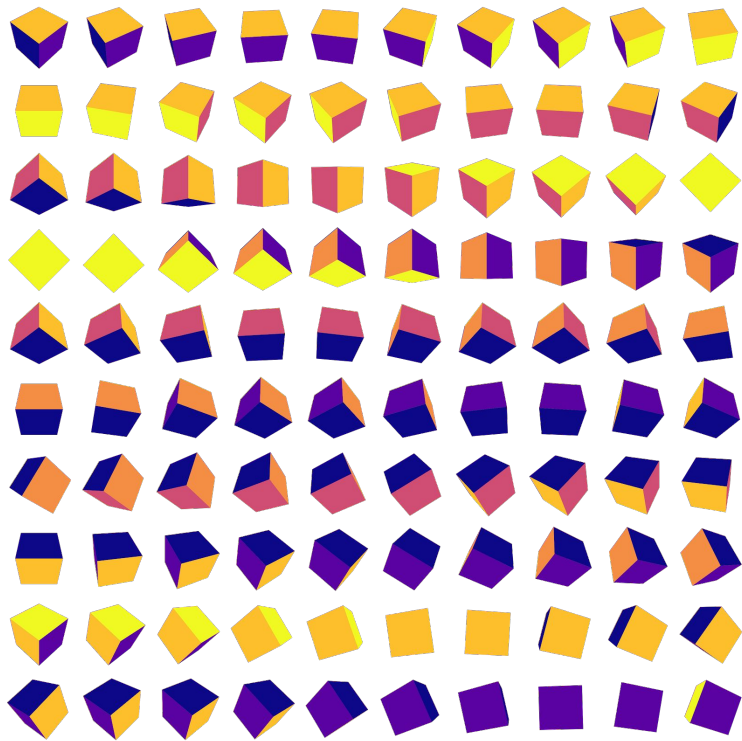


[MicroBooNE Article](#)

Approach 1: Data Augmentation

~500 Fold increase in training

training without rotational symmetry

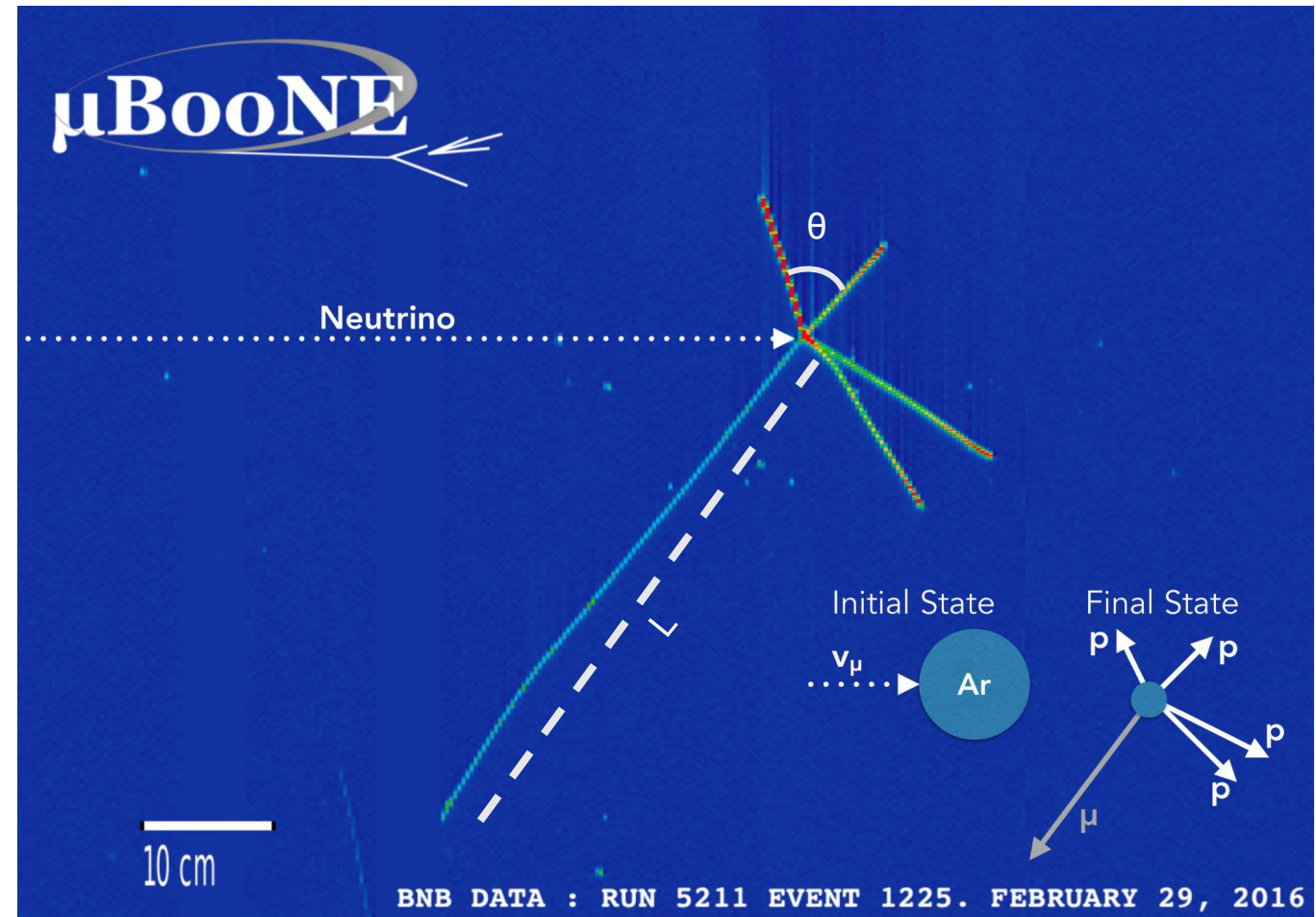


training with symmetry

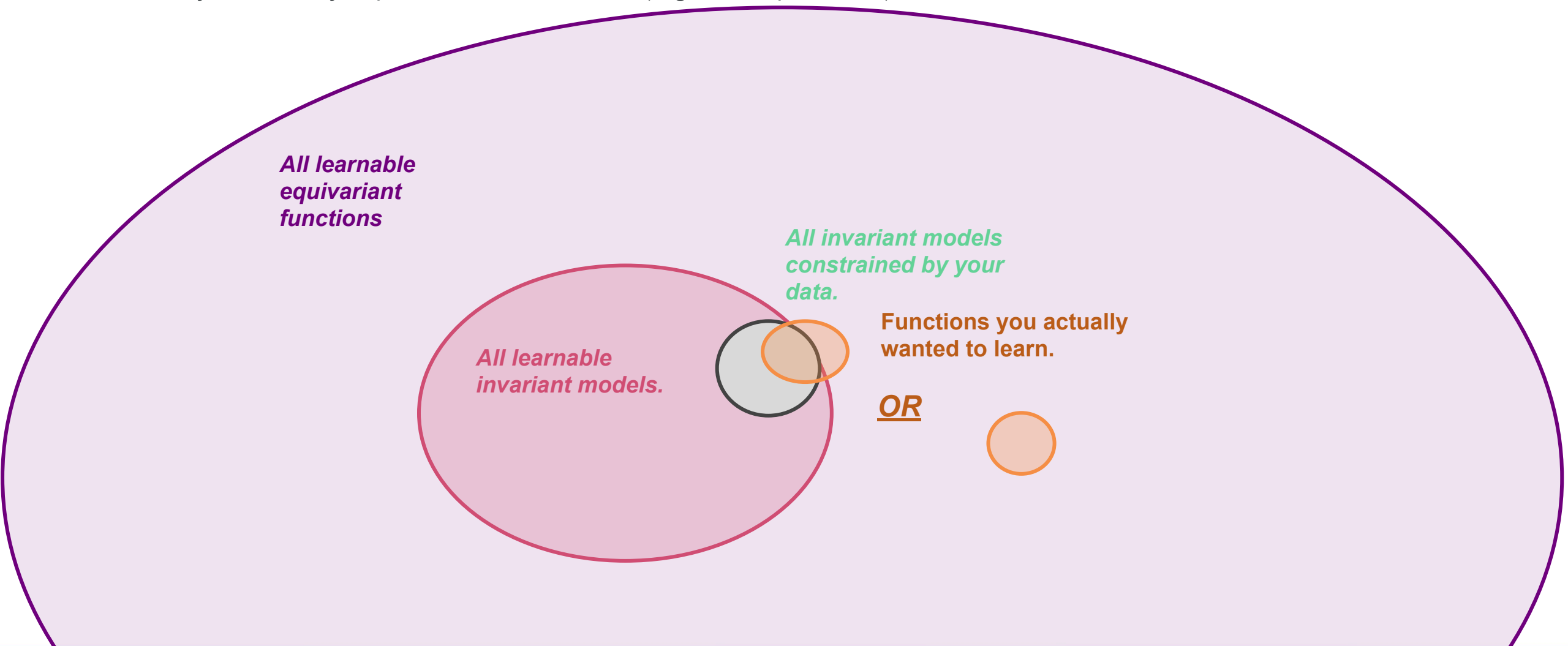


Approach 2: Invariant Models

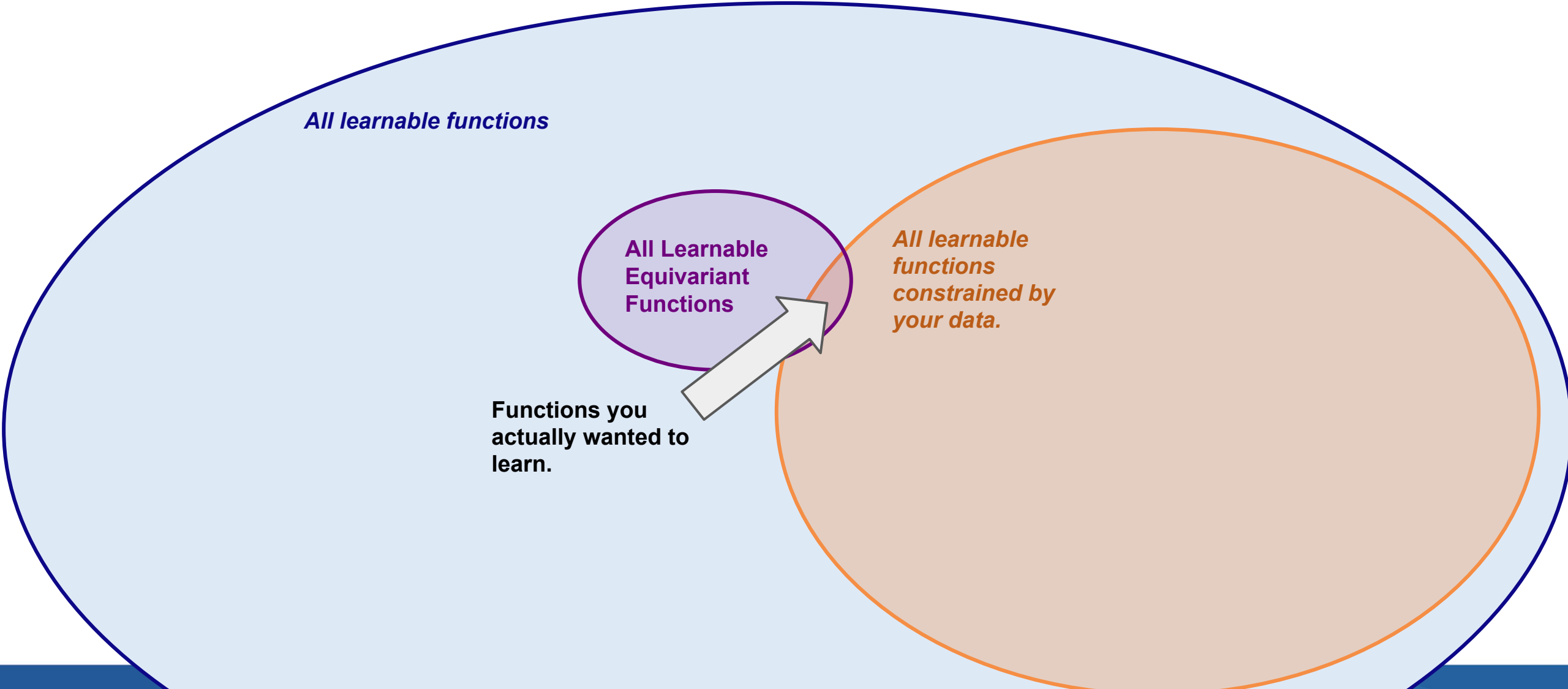
Invariant models pre-compute invariant features and throw away the coordinate system. Equivariant models keep the coordinate system AND if the coordinate system changes, the outputs change accordingly.



if you only use invariant models (only use scalar multiplication),
you have to guarantee that your input features already
contain any necessary equivariant interactions (e.g. cross-products).



Equivariant Functions substantially shrink the space of learnable functions



All learnable functions

All Learnable Equivariant Functions

All learnable functions constrained by your data.

Functions you actually wanted to learn.

Taken with a grain of salt

- There is some problem with the equivariant model that is messing up the equivariance:
 - This could be the initialization, choice of hyperparameters, or even simply a bug
 - Another possibility is that the average pooling could be breaking the equivariance. May need to use smoothed gaussian downpooling.
- The mode struggles slightly more with tracks. Needs to be investigated
- The hyperparameter space is much bigger and as there are not many priors in the field of what works and what doesn't
- Although these models are harder to train, they generalize much better (training and validation gap is minimal).

Quantifying Error

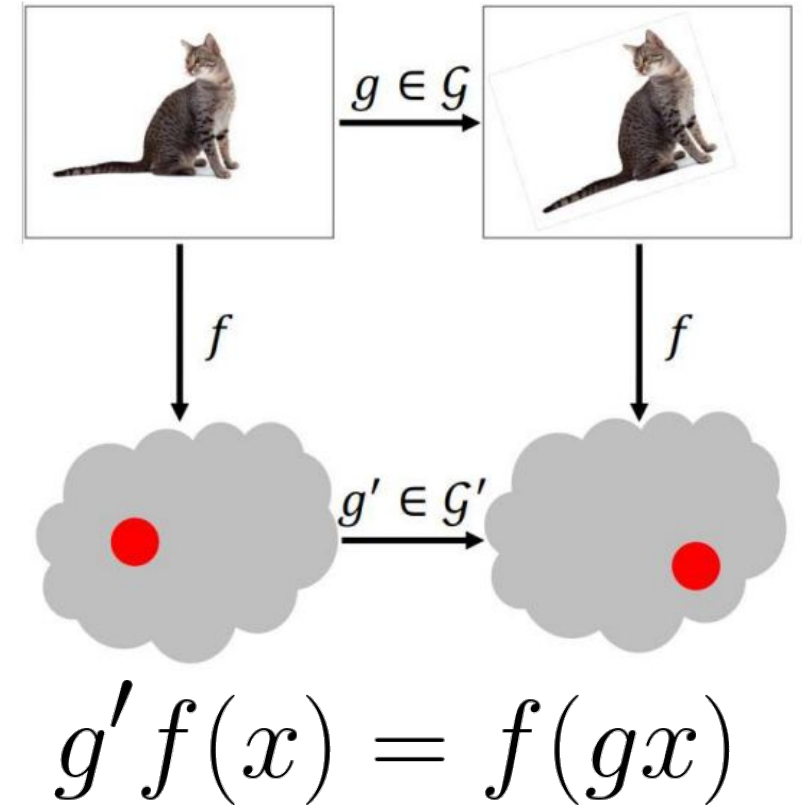
- The equivariance error is defined as

$$\Delta = g' f(x) - f(gx)$$

- This can be used for even the baseline non-equivariant model to check its equivariance
- I use a different parametrization

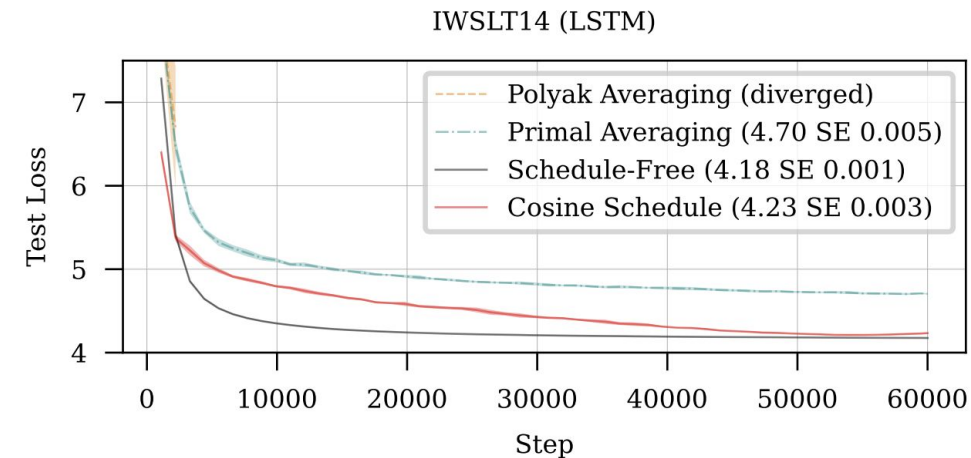
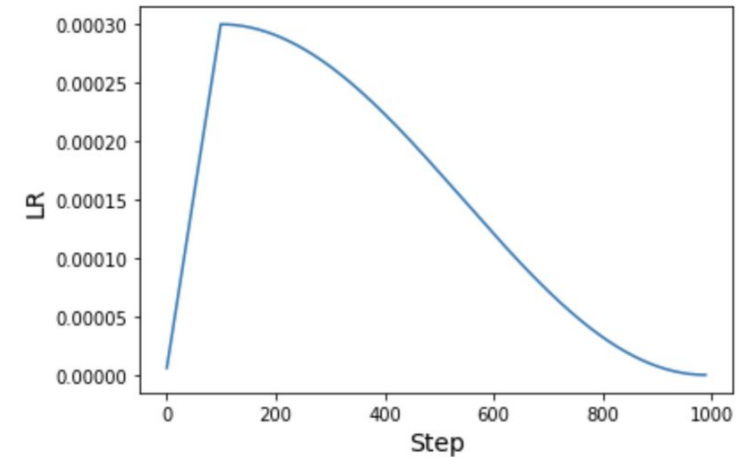
$$\Delta = g'(f gx) - f(x)$$

Equivariance



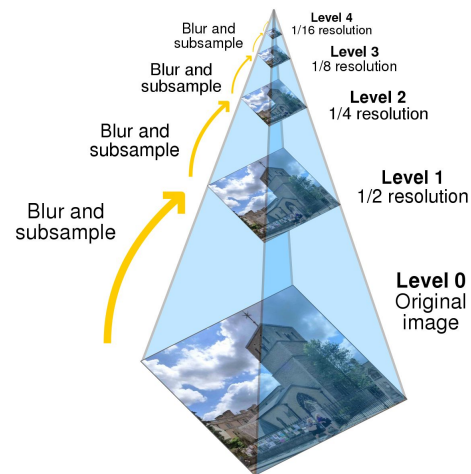
Training Dynamics

- ML Literature has shown that a well tuned cosine annealing LR schedule gives better performance than any alternative learning rate schedule. This adds an extra hyperparameter of max_steps
- A recent paper came out [[The Road Less Scheduled](#)] showing that by using polyak-primal averaging they can outperform a cosine schedule by both speed and performance (don't need to set max_steps).
- Works similar to Exponential Moving Average parameter training



Downsampling

- Downsampling breaks equivariance. CNNs are not fully shift equivariant because of downsampling!
- Can be solved by downsampling using gaussian pyramid
- Need special care for downsampling non-scalar irreps



Downsampling

