Equivariant Graph Neural Networks

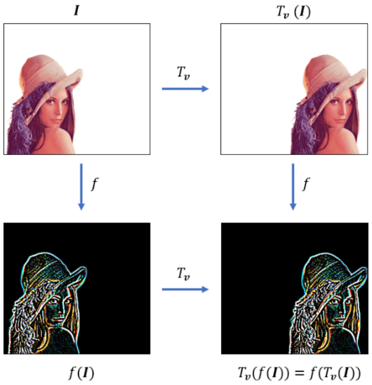
Kfir Eliyahu Ben Eliav Jonathan Kouchly

January 1, 2025

Outline

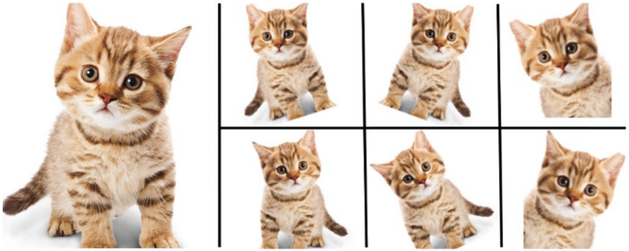
Motivation

- Our neural networks can operate on data of many types.
- We often work with images, text, audio, graphs and more.
- These data types have different structures and qualities, and we would like to build architectures that best suit them.



Motivation

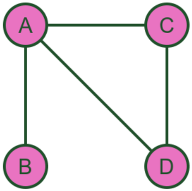
- A cat is a cat no matter how you look at it.



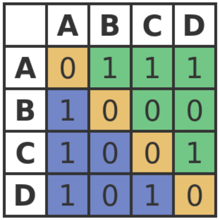
- It is acceptable to assume that being invariant to the rotation of the cat is a good property for a classification network.

Motivation

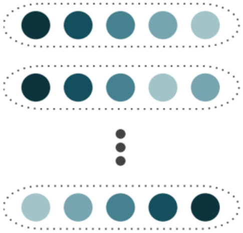
- Our focus today is on sets and graph data.



Simple graph



	A	B	C	D
A	0	1	1	1
B	1	0	0	0
C	1	0	0	1
D	1	0	1	0



Construction of an Equivariant Neural Network

When contructing an equivariant neural network, two things should always be considered:

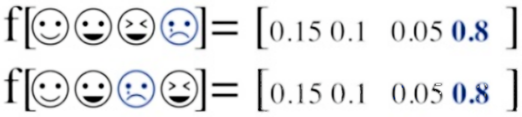
- The symmetries of the data:
  - What inherent structure should our model be oblivious to?
- The space of functions learnable by the network:
  - Are we fully utilizing the space of functions that are equivariant to the symmetries of the data?

The Permutation Group S_n

- The permutation group S_n is the group of all permutations of n elements.
- It has $n!$ elements, representing the $n!$ ways to order n elements.
- Given a set $X = \{x_1, x_2, \dots, x_n\}$, a permutation $\pi \in S_n$ is a bijection $\pi : X \to X$
- e.g. $x = (x_1, x_2, x_3)$, and $\pi = (1, 2, 3) \in S_3$ is the permutation that maps $1 \to 2$, $2 \to 3$ and $3 \to 1$.
- We denote the **action** of π on x as $\pi x = (x_3, x_1, x_2)$.

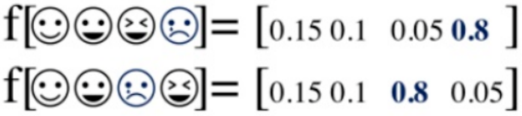
Permutation Invariance

- Let $H \leq S_n$ be a subgroup of the symmetric group.
- $f : \mathbb{R}^n \to \mathbb{R}$ is *H-invariant* if $f(x) = f(\pi x)$ for all $\pi \in H$.



Permutation Equivariance

- Let $H \leq S_n$ be a subgroup of the symmetric group.
- $f : \mathbb{R}^n \to \mathbb{R}^n$ is *H-equivariant* if $\pi f(x) = f(\pi x)$ for all $\pi \in H$.



Permutation of a Set

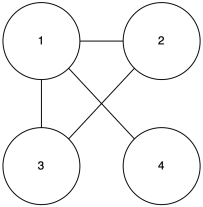
- Assume our set is $X = \{x_1, x_2, \dots, x_n\}$.
- We can represent X as a matrix $X \in \mathbb{R}^{n \times d}$.
- Any permutation $g \in S_n$ can be represented as a permutation matrix $\boldsymbol{P} \in \mathbb{R}^{n \times n}$,
- The action of g on X is then $\boldsymbol{P}X$.
- An invariant neural network is a function $f : \mathbb{R}^{n \times d} \to \mathbb{R}^{d'}$ such that $f(X) = f(\boldsymbol{P}X)$.
- An equivariant neural network is a function $f : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d'}$ such that $\boldsymbol{P}f(X) = f(\boldsymbol{P}X)$.

Permutation of a Graph

- Our data is now a graph adjacency matric $A \in \mathbb{R}^{n \times n}$.
- A permutation matrix $\boldsymbol{P} \in \mathbb{R}^{n \times n}$ acts on the adjacency matrix A.
- The action of $\boldsymbol{P}$ on A is:

$$\boldsymbol{P}^{\boldsymbol{T}} A \boldsymbol{P}$$

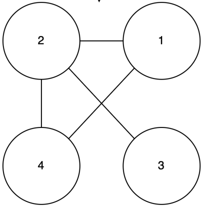
Permutation of a Graph



$$A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

$$\pi = (1,2)(3,4) \qquad P_\pi = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$A' = P_\pi^T A P_\pi$$



$$A' = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

Equivariant Network Construction

Theorem

Let L be a linear equivariant layer, and let f be a neural network constructed be stacking L and non-linearities σ. Then f is permutation equivariant.

Proof.

Let x be a set of n elements, and let $g \in S_n$ be a permutation.

$$f(gx) = L(\sigma(L(\sigma(\ldots L(gx)\ldots)))) = L(\sigma(L(\ldots g\sigma(L(x))\ldots))) = \ldots$$

$$gL(\sigma(L(\sigma(\ldots L(x)\ldots)))) = gf(x)$$

$\square$

Invariant Network Construction

Theorem

Let f be an equvariant neural network and let ϕ be a permutation invariant function. Then $h = \phi(f(x))$ is a permutation invariant neural network.

Proof.

Let x be a set of n elements, and let $g \in S_n$ be a permutation.

$$h(gx) = \phi(f(gx)) = \phi(gf(x)) = \phi(f(x)) = h(x)$$

$\square$

Deep Sets

- A seminal work in the field of equivariant neural networks.
- Recall the two properties we mentioned earlier (symmetries of the data and the space of functions learnable by the network).
- *DeepSets* is an architecture that is equivariant to set permutations and is maximally expressive in the space of permutation equivariant functions.
- We are going to see the construction and prove it satisfies equivariance and expressiveness.

DeepSets Layer

- We saw a general structure of an invariant and equivariant network.
- To fill in the details, we need to define the equivariant layer L and the invariant function ϕ.

Definition

Consider a set $x = \{x_1, x_2, \dots, x_n\}$, where $x_i \in \mathbb{R}$. Define its representation $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{R}^n$.
A *DeepSets* layer is defined as

$$L(x) = \lambda \mathrm{I}\mathbf{x} + \mathbf{x1}$$

.

DeepSets Layer

Definition

Consider a set $x = \{x_1, x_2, \dots, x_n\}$, where $x_i \in \mathbb{R}$.
A *DeepSets* layer is defined as

$$L(x) = \lambda \mathrm{I}\mathbf{x} + \mathbf{x1}$$

.

Theorem

A DeepSets layer is permutation equivariant.

Proof.

Let x be a set of n elements, and let $g \in S_n$ be a permutation.

$$L(gx) = \lambda \mathrm{I}(gx) + (gx)\mathbf{1} = g(\lambda \mathrm{I}x) + x\mathbf{1} = gL(x)$$

$\square$

DeepSets Layer

Theorem

Any linear equivariant layer is of the shape $L(x) = \lambda \mathrm{I}\mathbf{x} + \mathbf{x}\mathbf{1}$.

Proof (through example).

Let $\mathbf{x} = \{x_1, x_2, x_3\}$ be a set, $W \in \mathbb{R}^{3\times 3}$, and $L(x) = W\mathbf{x}$ such that L is equivariant, $\mathrm{W} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$.

Consider permutation $g = (2,3) \in S_3$. Note that

$$g\left(W\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}\right)_1 = (Wx)_1 = (Wgx)_1 = \left(W\begin{pmatrix} x_1 \\ x_3 \\ x_2 \end{pmatrix}\right)_1$$

DeepSets Layer

Proof (through example) - continued.

The previous equation can be written as:

$$ax_1 + bx_2 + cx_3 = ax_1 + bx_3 + cx_2$$

This should hold for any choice of $\mathbf{x} \Rightarrow b = c,\ d = f, g = h$.
For the permutation $\sigma = (1, 3, 2)$:

$$W(\sigma x) = W \begin{pmatrix} x_2 \\ x_3 \\ x_1 \end{pmatrix} = \begin{pmatrix} ax_2 + bx_3 + bx_1 \\ dx_2 + ex_3 + dx_1 \\ fx_2 + fx_3 + ix_1 \end{pmatrix}$$

$$\sigma W \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = (1,3,2) \begin{pmatrix} ax_1 + bx_2 + bx_3 \\ dx_1 + ex_2 + dx_3 \\ fx_1 + fx_2 + ix_3 \end{pmatrix} = \begin{pmatrix} dx_1 + ex_2 + dx_3 \\ fx_1 + fx_2 + ix_3 \\ ax_1 + bx_2 + bx_3 \end{pmatrix} \overset{!}{=}$$

$$= W \begin{pmatrix} x_2 \\ x_3 \\ x_1 \end{pmatrix}$$

DeepSets Layer

Proof (through example) - continued.

We can use the same logic to show that $a = e = i$, $b = c = d = f = g = h$.
We get that all values in the diagonal must be equal and all values in the off-diagonal must be equal, and W must have the form:

$$W = \begin{pmatrix} a & b & b \\ b & a & b \\ b & b & a \end{pmatrix}$$

. □

DeepSets Network

- We have an initial layer L, which we proved is equivariant.
- A deep sets invariant network is now constructed as:

$$f(x) = \phi(L\sigma(L\sigma(\dots L\sigma(L(x))\dots))) \quad \text{where} \quad \phi(x) = \sum_{i=1}^{n} x_i$$

- It is easy to see that ϕ is permutation invariant, and thus f is permutation invariant.
- For a classification network, take some classification module ρ (e.g. an MLP), and define the final network as:

$$h(x) = \rho(f(x))$$

DeepSets Network

- Notice that the network is only defined for sets with elements in $\mathbb{R}$.

- We can extend this to a set $X = \{x_1, x_2, \ldots x_n\} \subset \mathbb{R}^d$ with representation $\mathbf{X} \in \mathbb{R}^{n \times d}$ by defining L as:

$$L(x) = \mathbf{X}W_1 + \mathbf{1}\mathbf{1}^T\mathbf{X}W_2$$

- This keeps the general structure of the layer: a linear transformation of the distinct elements of the set summed with the mean of the set.

DeepSets Expressivity

- We have shown that the *DeepSets* network is permutation invariant.
- We now want to show that it is maximally expressive in the space of permutation invariant functions.
- We show outline for the proof, showing *DeepSets* can separate any two sets that are not equal.

Theorem

Let $x, y \in \mathbb{R}^{n\times d} \quad s.t \quad x \neq gy \quad \forall g \in S_n$. Then there exists a DeepSets network that separates x and y, namely $F(x;\theta) \neq F(y;\theta)$.

DeepSets Expressivity

Proof outline:

1. Consider distinct rows of x, y.
2. Construct input output pairs using standard basis elements (one hot encodings of the elements in the set).
3. In each layer $L(x) = \mathbf{X}W_1 + \mathbf{1}\mathbf{1}^T\mathbf{X}W_2$, set $W_2 = 0$, collapsing our network to a standard MLP.
4. According to a result of the universal approximation theorem, we can learn a one to one mapping between the input and output pairs.
5. Output for each set the sum of the output pairs.
6. For each set, we computed the hitogram of its elements, which is a unique representation of the set.

Invariant and Equivariant Graph Networks

- We saw a simple, maximally expressive equivariant network for sets.

- We now want to extend this to graphs.

Invariant and Equivariant Graph Networks

- Recall the action of the permutation matrix $\boldsymbol{P}$ on the adjacency matrix A is:
$$\boldsymbol{P}^{\boldsymbol{T}} A \boldsymbol{P}$$
- Let's use this to define a linear invariant layer $L \in \mathrm{Hom}(\mathbb{R}^{n \times n}, \mathbb{R})$. We want to satisfy the following:
$$L(\boldsymbol{P}^{\boldsymbol{T}} A \boldsymbol{P}) = L(A)$$
- It's hard from this to understand the conditions on L.

Invariant and Equivariant Graph Networks

- Recall the action of the permutation matrix $\boldsymbol{P}$ on the adjacency matrix A is:

$$\boldsymbol{P}^{\boldsymbol{T}} A \boldsymbol{P}$$

- Let's use this to define a linear invariant layer $L \in \mathrm{Hom}(\mathbb{R}^{n \times n}, \mathbb{R})$. We want to satisfy the following:

$$L(\boldsymbol{P}^{\boldsymbol{T}} A \boldsymbol{P}) = L(A)$$

- It's hard from this to understand the conditions on L.
- We define an equivalent condition by column stacking the adjacency matrix to get a vector, and then looking at L's matrix representation: $\mathbf{L} \in \mathbb{R}^{1 \times n^2}$.

$$\mathbf{L}\mathrm{vec}(\boldsymbol{P}^{T} A \boldsymbol{P}) = \mathbf{L}\mathrm{vec}(A)$$

Invariant and Equivariant Graph Networks

- We now introduce a crucial property of the Kronecker product:

$$\mathrm{vec}(XAY) = Y^T \otimes X\mathrm{vec}(A)$$

- Using this, the previous condition can be written as:

$$\mathbf{L}(P^T \otimes P^T)\mathrm{vec}(A) = \mathbf{L}\mathrm{vec}(A)$$

- For this condition to hold for all A, we need $\mathbf{L}$ to satisfy:

$$\mathbf{L}(P^T \otimes P^T) = \mathbf{L}$$

- transposing the equation, and with severe abuse of notation ($\mathbf{L}^T = \mathrm{vec}(\mathbf{L})$), we finally get:

$$(P \otimes P)\mathrm{vec}(\mathbf{L}) = \mathrm{vec}(\mathbf{L})$$

Invariant and Equivariant Graph Networks

- After some work and moderate logic jumps, we got a condition for a permutation invariant linear layer $\mathbf{L}$.
- Developing the condition for an equivariant layer $\mathbf{L} \in \mathbb{R}^{n^2 \times n^2}$ is very similar:

$$\mathbf{L}\text{vec}(\boldsymbol{P}^T A \boldsymbol{P}) = \boldsymbol{P}^T \mathbf{L}\text{vec}(A)\boldsymbol{P}$$

- Using the Kronecker product property, we get:

$$\mathbf{L}(P^T \otimes P^T)\text{vec}(A) = (P^T \otimes P^T)\mathbf{L}\text{vec}(A)$$

- This should hold for every A, so we get:

$$\mathbf{L}(P^T \otimes P^T) = (P^T \otimes P^T)\mathbf{L}$$

Invariant and Equivariant Graph Networks

- $(P^T \otimes P^T)$ is an $n^2 \times n^2$ permutation matrix, and its inverse is $(P \otimes P)$.
$$(P \otimes P)L(P^T \otimes P^T) = \mathbf{L}$$
- Once again using the Kronecker product property, we get:

$$\text{vec}(\mathbf{L}) = \text{vec}\left((P \otimes P)\mathbf{L}(P^T \otimes P^T)\right) = (P{\otimes}P){\otimes}(P{\otimes}P)\text{vec}(\mathbf{L})$$

$$= (P \otimes P \otimes P \otimes P)\text{vec}(\mathbf{L})$$

Invariant and Equivariant Graph Networks

- What if our adjacency matrix is actually a tensor $\mathbf{A} \in \mathbb{R}^{n^k}$?
- Instead of looking at relationships between pairs of nodes, we now look at relationships between k-tuples of nodes.
- We want to extend the previous condition $\mathbf{L}\text{vec}(\boldsymbol{P} \circ \mathbf{A}) = \mathbf{L}\text{vec}(\mathbf{A})$ to hold for any A.
- Note that the action of $\boldsymbol{P}$ on $\mathbf{A}$ is not necessarily $\boldsymbol{P}^T\mathbf{A}\boldsymbol{P}$ as this may not be defined over k-order tensors.
- We can extend the previous result to
  1. For invariant layers:

$$P^{\otimes k}\text{vec}(\mathbf{L}) = \text{vec}(\mathbf{L})$$

  2. For equivariant layers:

$$P^{\otimes 2k}\text{vec}\mathbf{L} = \text{vec}(\mathbf{L})$$

Invariant and Equivariant Graph Networks

- We developed conditions for a permutation invariant and equivariant linear layer L. Call these the Fixed Point Equations.
- Unfortunately, compared to the simple structure of the *DeepSets* layer, these equations are less interpretable. Let's try and make some sense of them.
- The condition we got is one that determines the values of the entries of L. We want to understand what entries of L should be equal.
- For an invariant layer $L \in \mathbb{R}^{1 \times n^k}$, $L_{i_1,i_2,\dots,i_k}$ is the weight which multiplies entry $A_{i_1,i_2,\dots,i_k}$.

Invariant and Equivariant Graph Networks

- The Fixed Point Equation for an invariant layer is:

$$P^{\otimes k}\text{vec}(L) = \text{vec}(L)$$

- The k-th permutation matrix permutes the k-th dimension of the input.

$$P^{\otimes k}\text{vec}(L)_{i_1,\dots i_k} = \text{vec}(L)_{\sigma(i)_1,\dots \sigma(i)_k}$$

- This partitions the entries of L into equivalence classes, where all entries in the same class are equal.
- For $k = 2$ we get 2 equivalence classes:

$$L_{i,j} \quad \text{where} \quad i \neq j \quad \text{and} \quad L_{i,i}$$

Invariant and Equivariant Graph Networks

- For an equivariant layer $L \in \mathbb{R}^{1 \times n^k}$, $L_{i_1,i_2,\dots,i_k,j_1,j_2,\dots,j_k}$ is the weight which multiplies entry $A_{i_1,i_2,\dots,i_k}$ and sends it to entry $j_1, j_2, \dots, j_k$ of the output.
- The Fixed Point Equation for an equivariant layer is:

$$P^{\otimes 2k}\text{vec}(L) = \text{vec}(L)$$

- The k-th permutation matrix permutes the k-th dimension of the input.

$$P^{\otimes 2k}\text{vec}(L)_{i_1,i_2,\dots,i_k,j_1,j_2,\dots,j_k} = \text{vec}(L)_{\sigma(i)_1,\dots\sigma(i)_k,\sigma(j)_1,\dots\sigma(j)_k}$$

- Once again, this partitions the entries of L into equivalence classes.
- For $k = 2$ we get 15 equivalence classes:

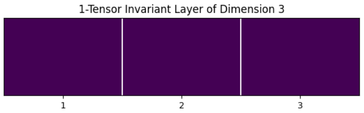
$$\{\{1\}\{2\}\{3\}\{4\}\}, \{\{1,2\}\{3\}\{4\}\}, \{\{1\}\{2,3\}\{4\}\} \dots \{\{1,2,3,4\}\}$$

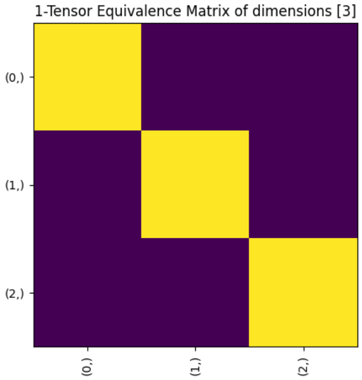
In general, the number of equivalence classes is $Bell(2k)$ for an (equivariant) invariant layer of order k

Invariant and Equivariant Graph Networks

- Did you notice somthing similar to the *DeepSets* layer?
- The construction of the *DeepSets* layer is identicle to the construction of an equivariant layer of order 1.
- We managed to generalize the equivariant layer to higher orders of node connectivity.
- We also managed to generalize the invariant layer to higher orders of node connectivity.

Invariant and Equivariant Graph Networks





Invariant and Equivariant Graph Networks

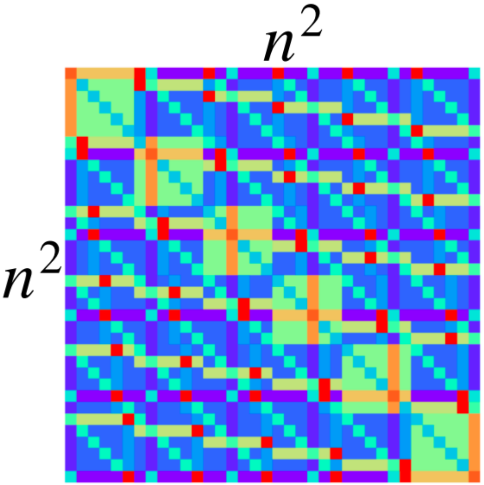


Figure taken from S. Ravanbakhsh (follow-up work)

Invariant and Equivariant Graph Networks

- Let's extend the construction of the equivariant layer even further, changing between the connectivity order, adding node features, and a bias term.

- We skip a few steps, but it makes sense that the bias term should satisfy the same fixed point equation as the linear term.

$$\mathbf{P}^{T\otimes k}\mathrm{vec}(\mathbf{C}) = \mathrm{vec}(\mathbf{C})$$

  The main idea is that every pair of k-hyperedges which permute under some permutation should have the same bias term.

Invariant and Equivariant Graph Networks

- We can also consider node features.
- The input to the layer now includes a set of n d-dimensional vectors.
- This is exactly the input to the *DeepSets* layer, we already know how to construct an equivariant layer for this input.
- The input to our layer is now the k-connectivity of each node, concatenated with it's feature:

$$A \in \mathbb{R}^{n^k \times d}$$

- The new general formulation, changing both the connectivity order and the feature dimensionality, is:

$$L : \mathbb{R}^{n^k \times d} \to \mathbb{R}^{n^{k'} \times d'}$$

Invariant and Equivariant Graph Networks

- We intoduce some new notation, please bare with us as we will use some Nifnufey Yadaim.

- We saw certain entries of the layer matrix L are equal. In *DeepSets* our connectivity is of order 1, so we got 2 equivalence classes. In the standard graph case, our connectivity is of order 2, so we got 15 equivalence classes.

- These equivalence classes are completely disjoint, meaning every entry in the linear layer belongs to exactly one equivalence class.

Invariant and Equivariant Graph Networks

- Let μ be some equivalence class of indices. We denote by $\mathbf{M}^{\mu}$ the binary tensor which is 1 at the indices in μ and 0 elsewhere.

- Denote by $\mathbf{B}^{\mu}, \mathbf{C}^{\mu}$ the linear layer and bias term of the layer, respectively.

- For an invariant layer $L : \mathbb{R}^{n^k \times d} \to \mathbb{R}^{n^1 \times d'}$, this defines a set of $dd'Bell(k) + d'$ binary tensors.

- For an equivariant layer $L : \mathbb{R}^{n^k \times d} \to \mathbb{R}^{n^{k'} \times d'}$, this defines a set of $dd'Bell(k + k') + d'Bell(k')$ binary tensors.

Invariant and Equivariant Graph Networks

The full charactarization of the space of invariant and equivariant layers is:

$$\text{invariant:} \quad \mathbf{B}_{\boldsymbol{a},i,i'}^{\lambda,j,j'} = \begin{cases} 1 & \boldsymbol{a}\in\lambda,\ \ i=j,\ \ i'=j' \\ 0 & \text{otherwise} \end{cases} \quad ; \quad \mathbf{C}_{i'}^{j'} = \begin{cases} 1 & i'=j' \\ 0 & \text{otherwise} \end{cases} \tag{9a}$$

$$\text{equivariant:} \quad \mathbf{B}_{\boldsymbol{a},i,\boldsymbol{b},i'}^{\mu,j,j'} = \begin{cases} 1 & (\boldsymbol{a},\boldsymbol{b})\in\mu,\ \ i=j,\ \ i'=j' \\ 0 & \text{otherwise} \end{cases} \quad ; \quad \mathbf{C}_{\boldsymbol{b},i'}^{\lambda,j'} = \begin{cases} 1 & \boldsymbol{b}\in\lambda,\ \ i'=j' \\ 0 & \text{otherwise} \end{cases} \tag{9b}$$

$$L(\mathbf{A})_{i'} = \sum_{\boldsymbol{a},i} \mathbf{T}_{\boldsymbol{a},i,i'} \mathbf{A}_{\boldsymbol{a},i} + \mathbf{Y}_{i'}; \quad \mathbf{T} = \sum_{\lambda,j,j'} w_{\lambda,j,j'} \mathbf{B}^{\lambda,j,j'}; \mathbf{Y} = \sum_{j'} b_{j'} \mathbf{C}^{j'} \tag{10a}$$

$$L(\mathbf{A})_{\boldsymbol{b},i'} = \sum_{\boldsymbol{a},i} \mathbf{T}_{\boldsymbol{a},i,\boldsymbol{b},i'} \mathbf{A}_{\boldsymbol{a},i} + \mathbf{Y}_{\boldsymbol{b},i'}; \quad \mathbf{T} = \sum_{\mu,j,j'} w_{\mu,j,j'} \mathbf{B}^{\mu,j,j'}; \mathbf{Y} = \sum_{\lambda,j'} b_{\lambda,j'} \mathbf{C}^{\lambda,j'} \tag{10b}$$

Invariant and Equivariant Graph Networks

- Turns out the set of these binary tensors $\mathbf{B}^{\mu}$ is an orthogonal basis for the space of equivariant linear layers $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^{k'}}$
- We won't show this, but we can prove the space of invariant/equivariant layers is of dimension $Bell(k')/Bell(k + k')$.
- The proof idea is showing the set of linear transformations is orthogonal and of the same dimension as the space of functions, making it a full basis.

Invariant and Equivariant Graph Networks

- To recap what we saw so far:
- We saw the full construction linear layers which operate on graphs of a general degree of connectivity.
- Our construction supports changing between different levels of connectivity, adding node features and changing between the dimension of these features.
- The construction is also extreamly efficient. It is linear in the number of dimensions and does no depend on the number of nodes, only on the connectivity order.

Invariant and Equivariant Graph Networks

- We now need to prove the two properties of the network: equivariance and expressiveness.

- Wait a minute, do we really need to prove these properties?

Invariant and Equivariant Graph Networks

- Equivariance is a direct result of the construction of the layer. The layer is constructed to satisfy the Fixed Point Equation, which is the condition for equivariance.

Invariant and Equivariant Graph Networks

- Equivariance is a direct result of the construction of the layer. The layer is constructed to satisfy the Fixed Point Equation, which is the condition for equivariance.

- The layer is constructed to be a full basis for the space of *linear* equivariant functions. Hence the network can express all linear equivariant functions. *Is this true also for nonlinear equivariant functions?*

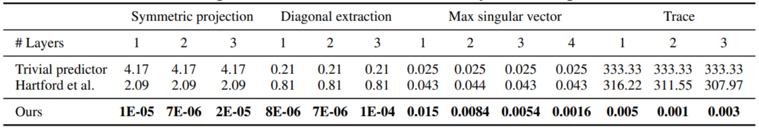
Invariant and Equivariant Graph Networks

- Equivariance is a direct result of the construction of the layer. The layer is constructed to satisfy the Fixed Point Equation, which is the condition for equivariance.
- The layer is constructed to be a full basis for the space of *linear* equivariant functions. Hence the network can express all linear equivariant functions. *Is this true also for nonlinear equivariant functions?*
- This allows for very efficient learning, and should also result in great generalization. The key here is a strong inductive bias.

Invariant and Equivariant Graph Networks

As researchers, we should present results that suit us, not those which make us look bad (these GPUs are not going to pay for themselves).

Table 1: Comparison to baseline methods on synthetic experiments.



	Symmetric projection			Diagonal extraction			Max singular vector				Trace		
# Layers	1	2	3	1	2	3	1	2	3	4	1	2	3
Trivial predictor	4.17	4.17	4.17	0.21	0.21	0.21	0.025	0.025	0.025	0.025	333.33	333.33	333.33
Hartford et al.	2.09	2.09	2.09	0.81	0.81	0.81	0.043	0.044	0.043	0.043	316.22	311.55	307.97
Ours	**1E-05**	**7E-06**	**2E-05**	**8E-06**	**7E-06**	**1E-04**	**0.015**	**0.0084**	**0.0054**	**0.0016**	**0.005**	**0.001**	**0.003**

Expressive Power of Equivariant Networks

- In previous lectures, we saw that a popular way to measure the expressivity of a GNN is by looking at the Weisfeiler-Lehman (WL) test.

Expressive Power of Equivariant Networks

- In previous lectures, we saw that a popular way to measure the expressivity of a GNN is by looking at the Weisfeiler-Lehman (WL) test.
- The k-WL tests are a family of graph isomorphism tests that can distinguish between increasingly many non-isomorphic graphs.
- We want to show that for some k, our k-order networks can distinguish between non-isomorphic graphs just as well as the k-WL test.

Expressive Power of Equivariant Networks

- In Maron et al. (2019), the authors show that the k-order equivariant network is able to distinguish between non-isomorphic graphs just as well as the k-WL test.
- The model proposed is much more efficient than k-GNNs which are also equivalent to the k-WL test.

Expressive Power of Equivariant Networks

- The proof that the k-order equivariant network is able to distinguish between non-isomorphic graphs just as well as the k-WL test is quite complex.

Expressive Power of Equivariant Networks

- The proof that the k-order equivariant network is able to distinguish between non-isomorphic graphs just as well as the k-WL test is quite complex.
- In general, we can "simulate" the $(k-1)$-FWL algorithm using a k-order equivariant network if we have a sufficient number of parameters.

Expressive Power of Equivariant Networks

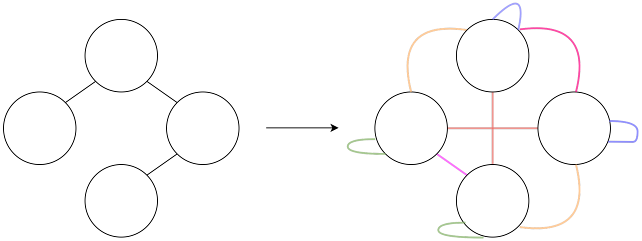
- The proof that the k-order equivariant network is able to distinguish between non-isomorphic graphs just as well as the k-WL test is quite complex.
- In general, we can "simulate" the $(k-1)$-FWL algorithm using a k-order equivariant network if we have a sufficient number of parameters.
- The information required to run the $(k-1)$-FWL algorithm is encoded in the input to the network.

Proof Sketch

- Given a graph $G = (V, E)$, we can represent it using a tensor $A \in \mathbb{R}^{n \times n}$ and a color matrix $C \in \mathbb{R}^{n \times e}$ (which is usually initialized uniformly).
- These two matrices can be combined into a single tensor $X \in \mathbb{R}^{n \times n \times e}$.

Proof Sketch

- Given a graph $G = (V, E)$, we can represent it using a tensor $A \in \mathbb{R}^{n \times n}$ and a color matrix $C \in \mathbb{R}^{n \times e}$ (which is usually initialized uniformly).
- These two matrices can be combined into a single tensor $X \in \mathbb{R}^{n \times n \times e}$.
- The initial coloring of the k-tuples in the graph encapsulates the *isomorphism type* of each k-tuple, as shown in the diagram for 2-FWL.



Proof Sketch

- The coloring method can be encoded in a tensor $B \in \mathbb{R}^{n^k \times k^2 \times (e+2)}$.
- The method is also equivariant, meaning changing the order of the nodes will change the coloring accordingly.
- Hence, this can be encoded using a single equivariant linear layer.

Proof Sketch

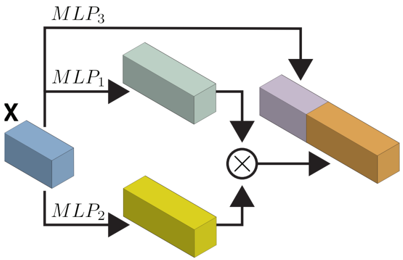
- Once we have the initial coloring encoded in the tensor B, we can simulate the $(k-1)$-FWL algorithm for every k-tuple (without proof).
- If we believe the statement above, we can encode the $(k-1)$-FWL algorithm in an equivariant graph network, despite it being not very efficient.

Proof Sketch

- Once we have the initial coloring encoded in the tensor B, we can simulate the $(k-1)$-FWL algorithm for every k-tuple (without proof).
- If we believe the statement above, we can encode the $(k-1)$-FWL algorithm in an equivariant graph network, despite it being not very efficient.
- The writers propose another, more simple GNN that can discriminate as well as 3-WL and doesn't require extremely high dimensional tensors.

A Simpler 3-WL Network

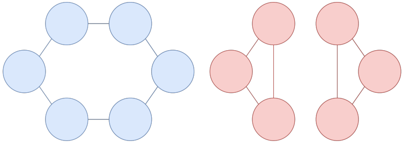
- Given an input tensor $X \in \mathbb{R}^{n \times n \times e}$, we can define $L(X) = \text{concat}(m_3(X), m_1(X) \cdot m_2(X))$.
- Here, m_1, m_2, m_3 are MLPs that are applied to each feature independently, $m_1, m_2 : \mathbb{R}^e \to \mathbb{R}^b$, $m_3 : \mathbb{R}^e \to \mathbb{R}^{b'}$. The matrix multiplication is also applied to each feature matrix independently.



- Matrix multiplication is equivariant to permutations, so the layer is equivariant.
- We stack several layers of this type followed by an invariant layer to receive an invariant network.

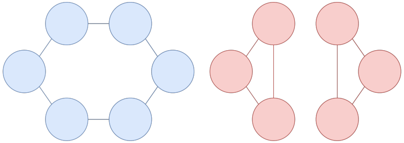
A Simpler 3-WL Network

- To see why this is equivalent to the 3-WL test, remember the following two graphs, and assume the only input is the adjacency matrix A ($e = 1$).



A Simpler 3-WL Network

- To see why this is equivalent to the 3-WL test, remember the following two graphs, and assume the only input is the adjacency matrix A ($e = 1$).



- The matrix multiplication in the model allows us to compute A^3.
- We can then apply the trace operation in the invariant linear layer to get $tr(A^3)$ which is the number of cycles of length 3 in the graph. This will distinguish between the two graphs.

Thank You!