Geometric Deep Learning

18 May 2022

John Tan Chong Min

Most slides extracted from the book:

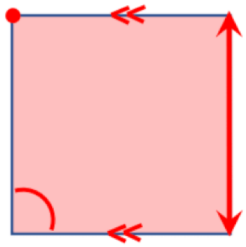
Geometric Deep Learning: Grids, Grounds, Graphs, Geodesics and Gauges by

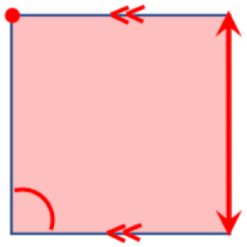
Michael M. Bronstein, Joan Bruna, Taco Cohen, Petar Veličković

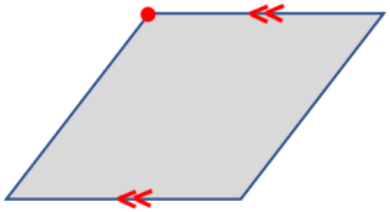
Background

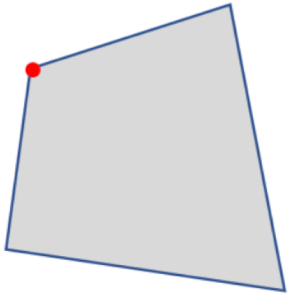
- Much of Deep Learning is fueled by Neural Network design
- Neural Network design uses a particular optimal structure suited to the data
- Structure has many forms of invariances, symmetries, which can be exploited and understood in new and novel ways to form new structures

Erlangen Program to unify Geometry





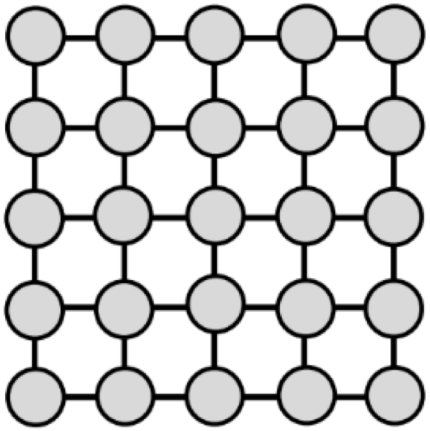


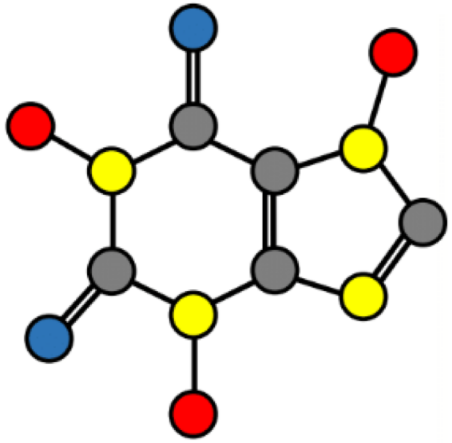


	Euclidean	Affine	Projective
angle	+	—	—
distance	+	—	—
area	+	—	—
parallelism	+	+	—
intersection	+	+	+

Can we do the same to classify invariance/symmetries in Deep Learning?

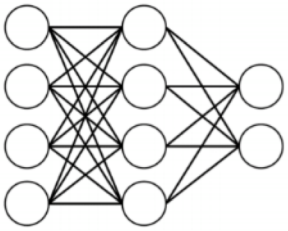
5Gs of Geometric Deep Learning



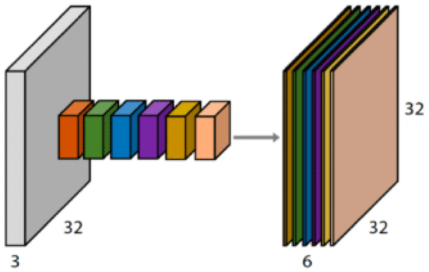


Geometry:	Grids	Groups	Graphs	Geodesics & Gauges
Used for:	Images, Sequences	Homogeneous Spaces	Graphs and Sets	Manifold, Meshes, Geometric Graphs

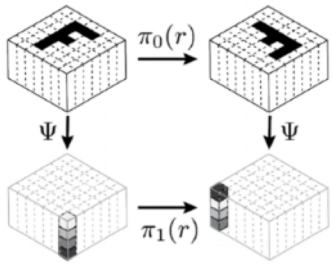
Invariances in Deep Learning Architectures



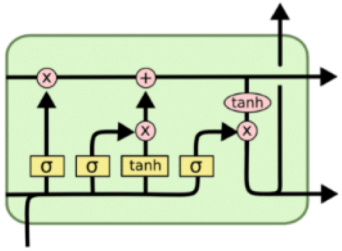
Perceptrons
Function regularity



CNNs
Translation



Group-CNNs
Translation+Rotation,
Global groups



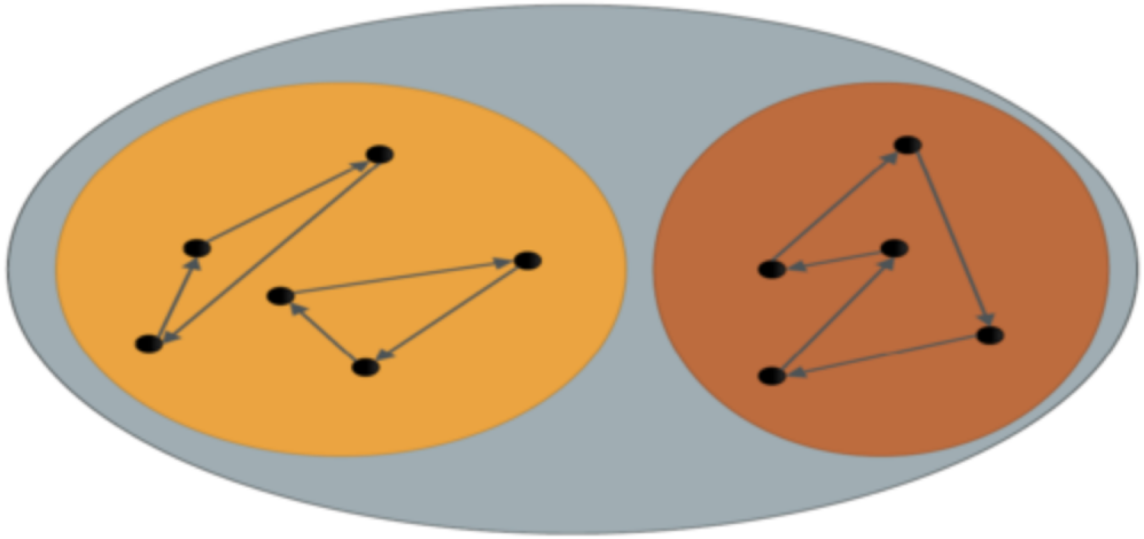
LSTMs
Time warping

DeepSets / Transformers
Permutation

GNNs
Permutation

Intrinsic CNNs
Isometry / Gauge choice

Why are knowing symmetries important?



- If we knew all the symmetries in each class, we can learn the class with just **ONE** sample.
- Easy to generalize from a single input (one-shot)

Symmetries

Let Ω denote a geometric domain

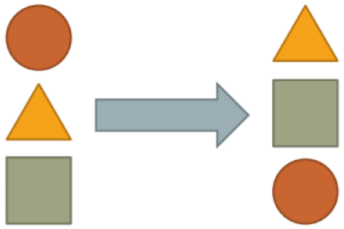
A transformation $\mathfrak{g} : \Omega \to \Omega$ is a symmetry if it preserves the structure of Ω

Examples:

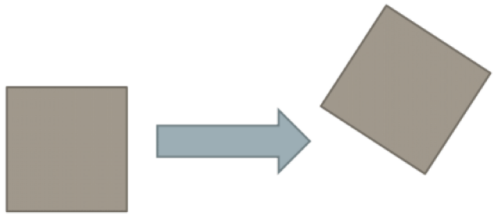
- Permutation of the elements of a *set* preserves set membership
- Euclidean isometries (rotation, translation, reflection) preserve distances and angles in Euclidean space $\Omega = \mathbb{R}^d$
- A general diffeomorphism preserves the smooth structure of a manifold Ω

Symmetry Group:

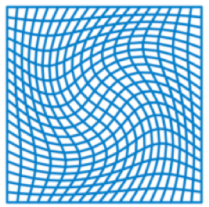
$\mathfrak{g} \in \mathfrak{G}$



Permutation



Euclidean isometry



Diffeomorphism

Symmetry Group

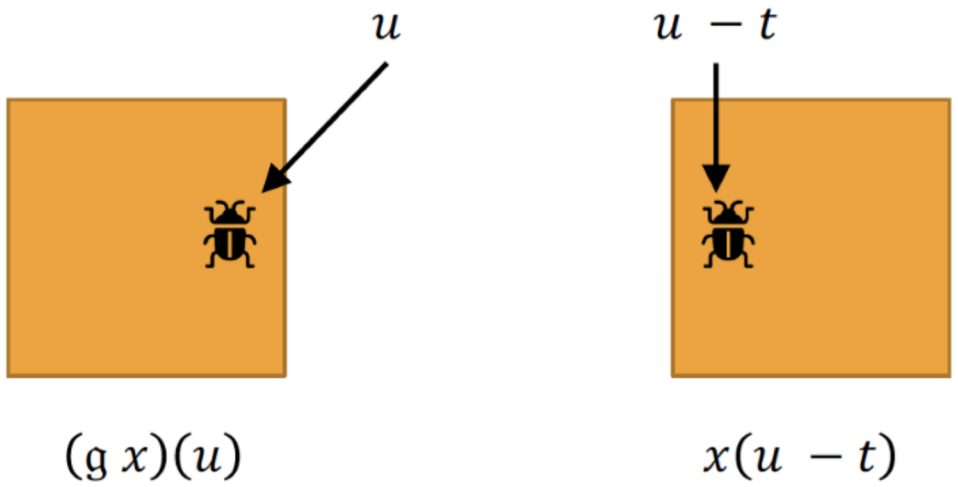
A *group* is a set $\mathfrak{G}$ with a binary operation denoted $\mathfrak{gh}$ satisfying the following properties:

- *Associativity:* $(\mathfrak{gh})\mathfrak{k} = \mathfrak{g}(\mathfrak{hk})$ for all $\mathfrak{g}, \mathfrak{h}, \mathfrak{k} \in \mathfrak{G}$
- *Identity*: there exists a unique $\mathfrak{e} \in \mathfrak{G}$ satisfying

$$\mathfrak{ge} = \mathfrak{eg} = \mathfrak{g}$$

- *Inverse:* for each $\mathfrak{g} \in \mathfrak{G}$ there is a unique inverse $\mathfrak{g}^{-1} \in \mathfrak{G}$, such that $\mathfrak{gg}^{-1} = \mathfrak{g}^{-1}\mathfrak{g} = \mathfrak{e}$
- *Closure*: for every $\mathfrak{g}, \mathfrak{h} \in \mathfrak{G}$, we have $\mathfrak{gh} \in \mathfrak{G}$

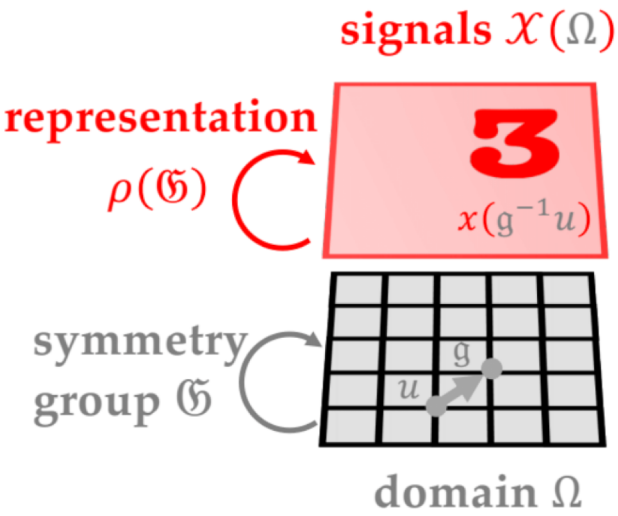
Group Action

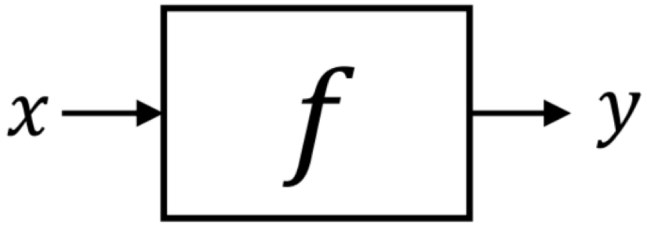


$$(\mathfrak{g}.x)(u) = x\left(\mathfrak{g}^{-1}u\right)$$

- x: input space
- u: domain space (where all possible inputs reside)
- g: translation t units to the right
- Value of the translated image $\mathfrak{g}.x$ at point u will be the same value in the original image at point u - t, *i.e.*, $\mathfrak{g}.x(u) = x(u - t)$.

Group Action



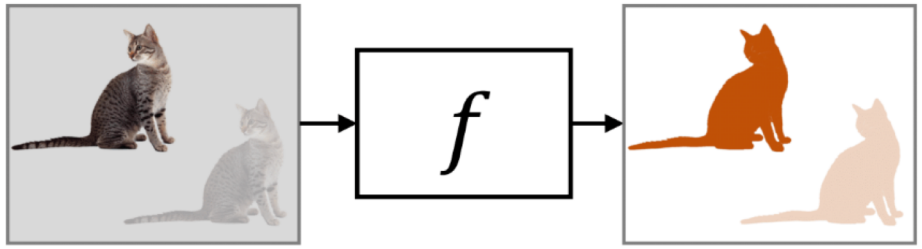


$$\rho(\mathfrak{g})x(u) = x(\mathfrak{g}^{-1}u)$$

The points on the original image after being transformed = original image with the inverse transformation applied to the points

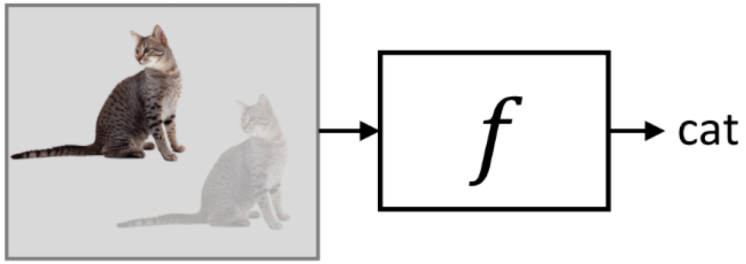
Equivariance vs Invariance

$\mathfrak{G}$-equivariance $f(\rho(\mathfrak{g})x) = \rho(\mathfrak{g})f(x)$



Segmentation

$\mathfrak{G}$-invariance $f(\rho(\mathfrak{g})x) = f(x)$



Classification

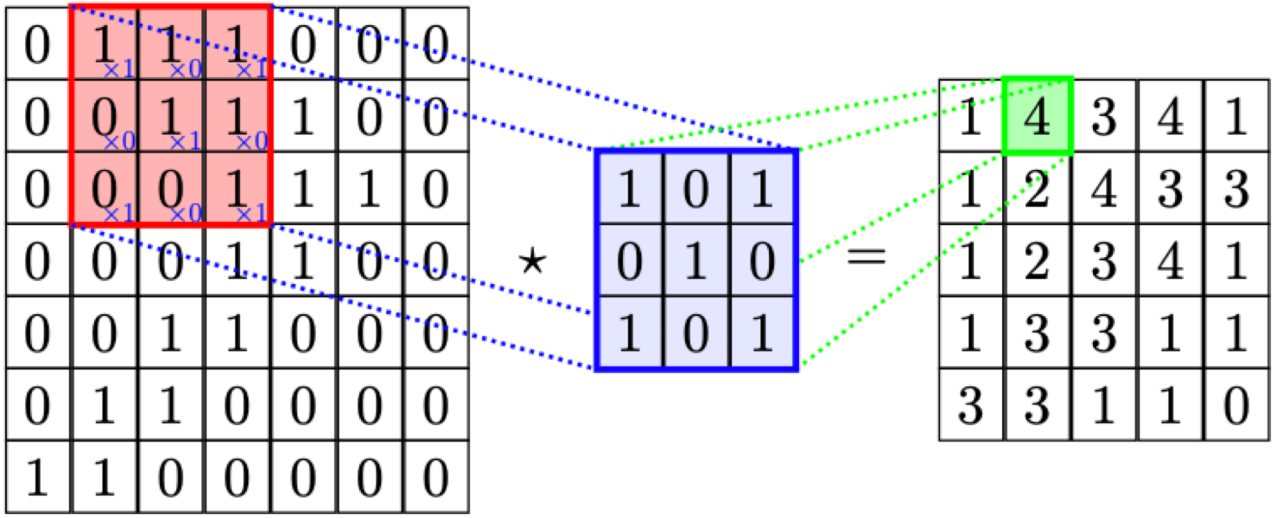
Caveat: Invariance/Equivariance should not be applied at all levels of network

- Part-whole relations may require parts to respect certain orientation
- The whole can be rotation-invariant, but not the parts

Convolutional Neural Networks (CNNs)

Translational Invariance / Equivariance

CNN – Translation Equivariance



$\star$ $=$

$\mathbf{x}$ $\qquad$ $\mathbf{C}(\boldsymbol{\theta})$ $\qquad$ $\mathbf{x} \star \boldsymbol{\theta}$

$$\overbrace{\boldsymbol{\theta}_{11} + \boldsymbol{\theta}_{13} + \boldsymbol{\theta}_{22} + \boldsymbol{\theta}_{31} + \boldsymbol{\theta}_{33}}$$

CNN Filter - Convolution

Any convolution with a compactly supported filter of size $H^f \times W^f$ can be written as a linear combination of generators $\boldsymbol{\theta}_{1,1}, \ldots, \boldsymbol{\theta}_{H^f,W^f}$, given for example by the unit peaks $\boldsymbol{\theta}_{vw}(u_1, u_2) = \delta(u_1 - v, u_2 - w)$. Any local linear equivariant map is thus expressible as

$$\mathbf{F}(\mathbf{x}) = \sum_{v=1}^{H^f} \sum_{w=1}^{W^f} \alpha_{vw} \mathbf{C}(\boldsymbol{\theta}_{vw}) \mathbf{x} \,, \tag{26}$$

$C(\theta)$: filter
α: normalizing factor
x: input

which, in coordinates, corresponds to the familiar 2D convolution (see Figure 14 for an overview):

$$\mathbf{F}(\mathbf{x})_{uv} = \sum_{a=1}^{H^f} \sum_{b=1}^{W^f} \alpha_{ab} x_{u+a,v+b} \,. \tag{27}$$

CNN – Overall Equation

In summary, a 'vanilla' CNN layer can be expressed as the composition of the basic objects already introduced in our Geometric Deep Learning blueprint:

$$\mathbf{h} = \mathbf{P}(\sigma(\mathbf{F}(\mathbf{x}))) \,, \tag{29}$$

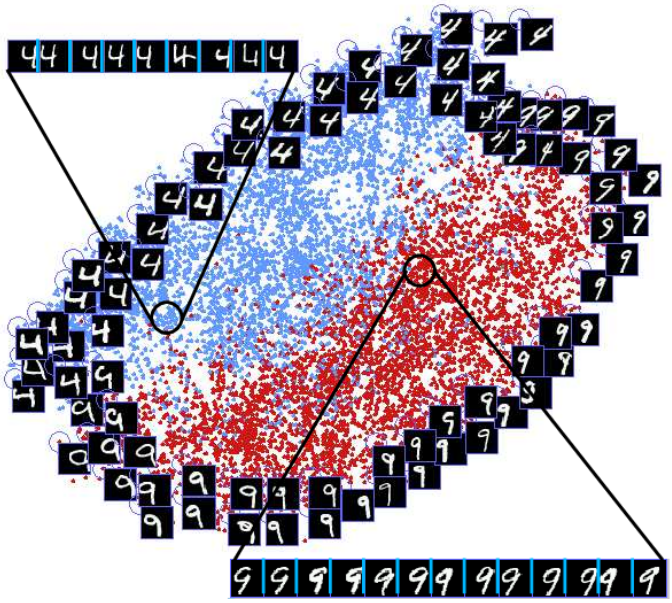
i.e. an equivariant linear layer $\mathbf{F}$, a coarsening operation $\mathbf{P}$, and a non-linearity σ. It is also possible to perform translation invariant *global* pooling operations within CNNs. Intuitively, this involves each pixel—which, after several convolutions, summarises a *patch* centered around it—*proposing* the final representation of the image, with the ultimate choice being guided by a form of aggregation of these proposals. A popular choice here is the average function, as its outputs will retain similar magnitudes irrespective of the image size (Springenberg et al., 2014).

Examples
P: Pooling
(translational invariance)

F: Shared filter weights
(translational equivariance)

The role of Data Augmentation

- CNNs has translation equivariance explicitly built into network via filters, translational invariance via pooling
- Vanilla CNN does not have other invariances like color changes, rotation, scaling built in
- Using Data Augmentation (with potentially Contrastive Loss) can help with building these invariances
  - Helps map the data into the correct manifold space where similar images are clustered together
  - Sub-optimal in terms of sample complexity



Dimensionality Reduction by Learning an Invariant Mapping, LeCun. (2016).

Modelling other invariances into CNNs

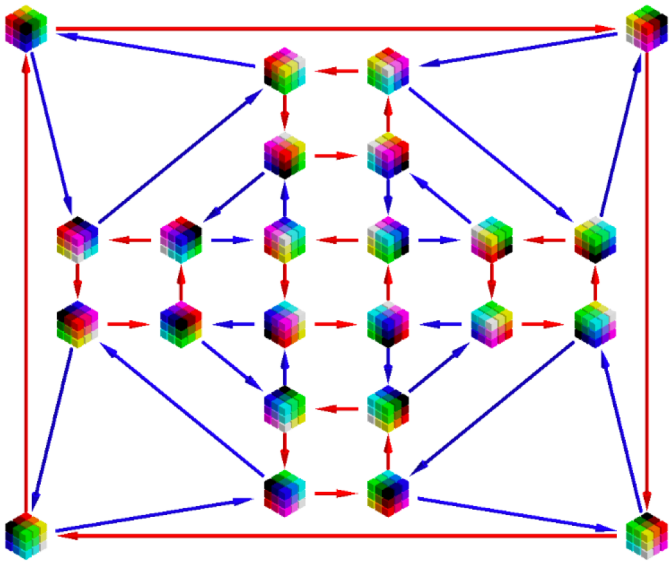


Figure 16: A 3×3 filter, rotated by all 24 elements of the discrete rotation group O_h, generated by 90-degree rotations about the vertical axis (red arrows), and 120-degree rotations about a diagonal axis (blue arrows).

- Apply equivariant symmetry-preserving group operations to the filters

$$(x \star \theta)(\mathfrak{g}) = \sum_{u \in \Omega} x_u \rho(\mathfrak{g}) \theta_u,$$

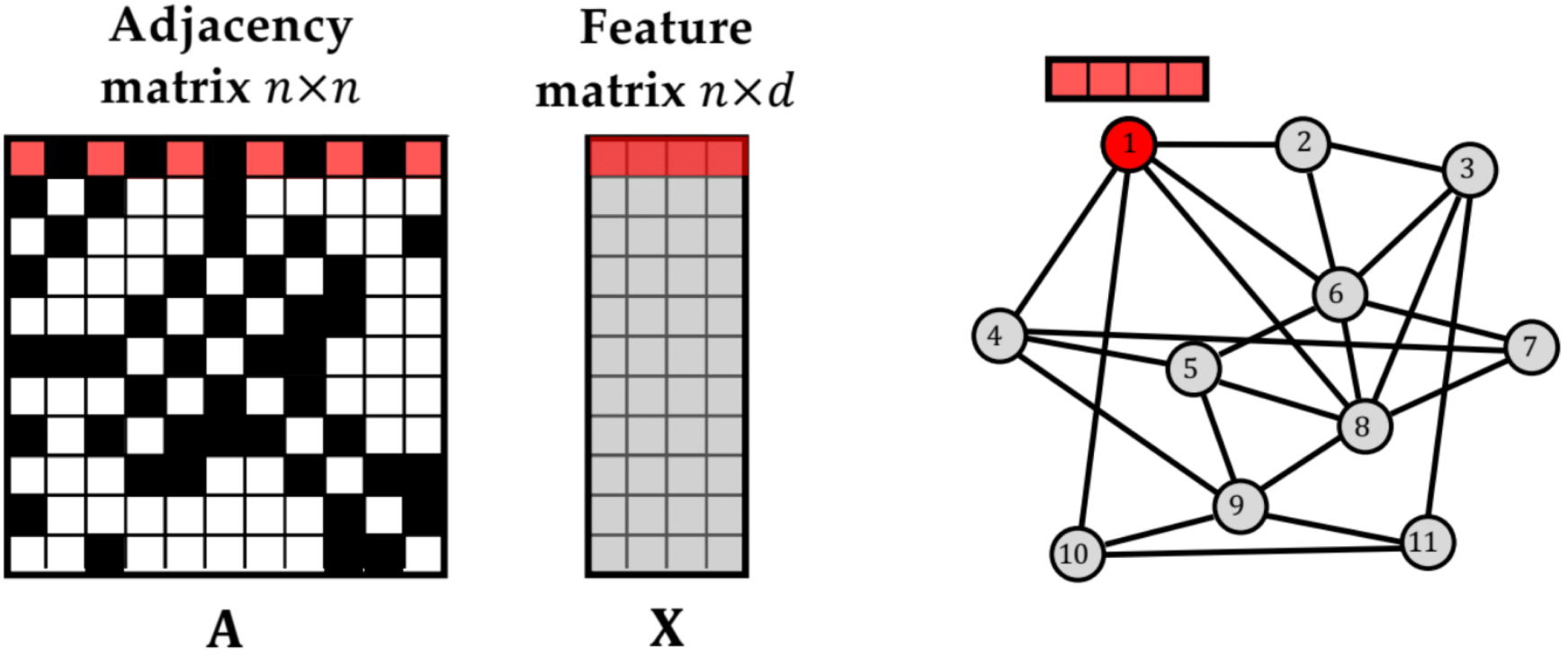
Ω: Neighbourhood of x to perform convolution on

g: Symmetry-preserving transformation

Graph Neural Networks (GNNs)

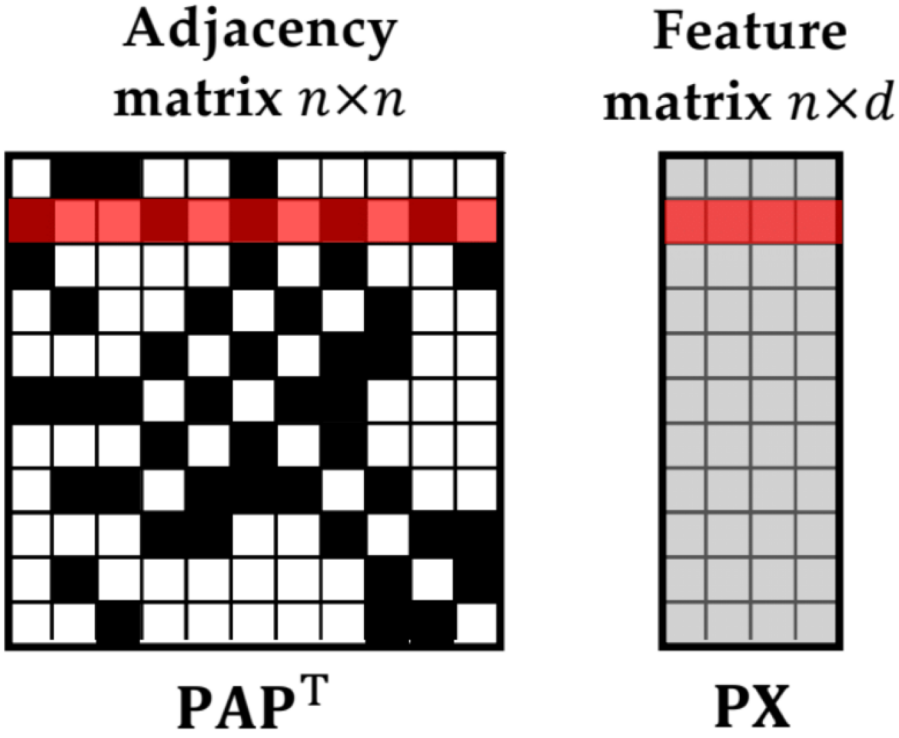
Permutation Invariance

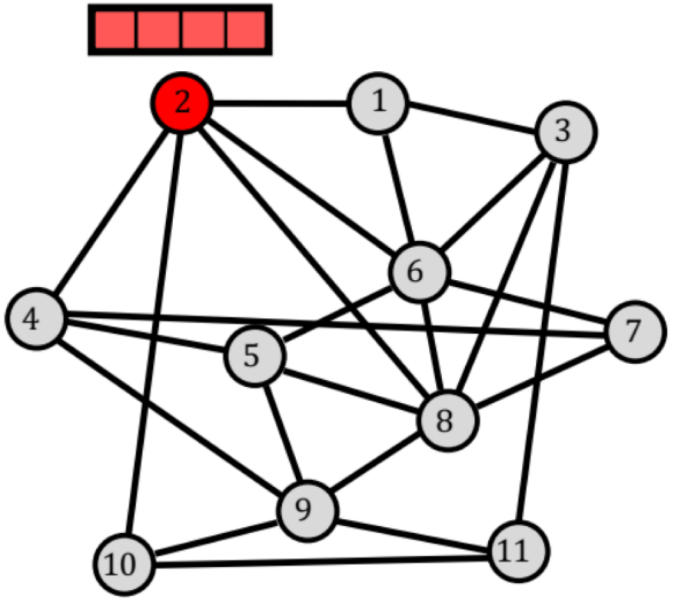
Graph Illustrated



Note: Numbering of nodes is arbitrary.

Graphs can be permuted arbitrarily





P: Permutation Matrix

Graph Definitions

As per our discussion in Section 4.1, we consider a graph to be specified with an adjacency matrix $\mathbf{A}$ and node features $\mathbf{X}$. We will study GNN architectures that are *permutation equivariant* functions $\mathbf{F}(\mathbf{X}, \mathbf{A})$ constructed by applying shared *permutation invariant* functions $\phi(\mathbf{x}_u, \mathbf{X}_{\mathcal{N}_u})$ over local neighbourhoods. Under various guises, this local function ϕ can be referred to as "diffusion", "propagation", or "message passing", and the overall computation of such $\mathbf{F}$ as a "GNN layer".

That is, we consider *graphs* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$

We can represent these edges in an *adjacency matrix*, $\mathbf{A}$, such that:

$$a_{ij} = \begin{cases} 1, & (i,j) \in \mathcal{E} \\ 0, & (i,j) \notin \mathcal{E} \end{cases}$$

Graph Definitions

Graphs give us a broader context: a node's *neighbourhood*
For a node i, its (1-hop) neighbourhood, $\mathcal{N}_i$, is commonly defined as:

$$\mathcal{N}_i = \{ j : (i,j) \in \mathcal{E} \ \vee (j,i) \in \mathcal{E}) \}$$

Accordingly, we can extract neighbourhood features, $\mathbf{X}_{\mathcal{N}_i}$, like so:

$$\mathbf{X}_{\mathcal{N}_i} = \{\{ \mathbf{x}_j : j \in \mathcal{N}_i \}\}$$

and define a *local* function, $\phi(\mathbf{x}_i, \mathbf{X}_{\mathcal{N}_i})$, operating over them.

Permutation Matrix

Within linear algebra, each permutation defines a $|\mathcal{V}| \times |\mathcal{V}|$ **matrix**

- Such matrices are called **permutation matrices** (*group action* $\rho(\mathfrak{g})$*!*)
- They have exactly one 1 in every row and column, zeroes elsewhere
- Their effect when left-multiplied is to permute rows of $\mathbf{X}$, like so:

$$\mathbf{P}_{(2,4,1,3)}\mathbf{X} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} - & \mathbf{x}_1 & - \\ - & \mathbf{x}_2 & - \\ - & \mathbf{x}_3 & - \\ - & \mathbf{x}_4 & - \end{bmatrix} = \begin{bmatrix} - & \mathbf{x}_2 & - \\ - & \mathbf{x}_4 & - \\ - & \mathbf{x}_1 & - \\ - & \mathbf{x}_3 & - \end{bmatrix}$$

Permutation Invariance vs Equivariance

We need to appropriately permute both **rows** and **columns** of $\mathbf{A}$
When applying a permutation matrix $\mathbf{P}$, this amounts to $\mathbf{PAP}^{\mathrm{T}}$

We arrive at updated definitions of suitable functions over graphs:

Invariance: $$f\left(\mathbf{PX}, \mathbf{PAP}^{\mathrm{T}}\right) = f(\mathbf{X}, \mathbf{A})$$

Equivariance: $$\mathbf{F}\left(\mathbf{PX}, \mathbf{PAP}^{\mathrm{T}}\right) = \mathbf{PF}(\mathbf{X}, \mathbf{A})$$

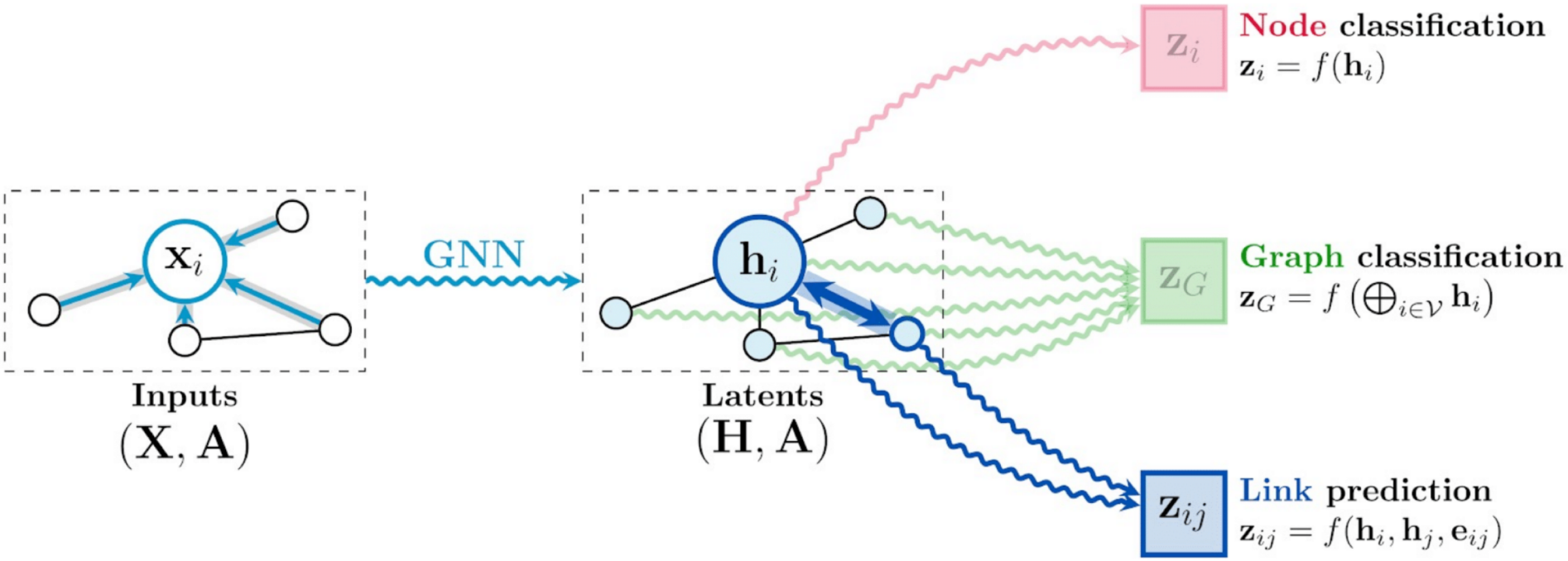
Generic GNN Structure

Now we can construct permutation equivariant functions, $\mathbf{F}(\mathbf{X}, \mathbf{A})$, by appropriately applying the local function, ϕ, over *all* neighbourhoods:

$$\mathbf{F}(\mathbf{X}, \mathbf{A}) = \begin{bmatrix} - & \phi(\mathbf{x}_1, \mathbf{X}_{\mathcal{N}_1}) & - \\ - & \phi(\mathbf{x}_2, \mathbf{X}_{\mathcal{N}_2}) & - \\ & \vdots & \\ - & \phi(\mathbf{x}_n, \mathbf{X}_{\mathcal{N}_n}) & - \end{bmatrix}$$

To ensure equivariance, it is sufficient if ϕ does not depend on the **order** of the nodes in $\mathbf{X}_{\mathcal{N}_i}$ (i.e. if it is *permutation invariant*).

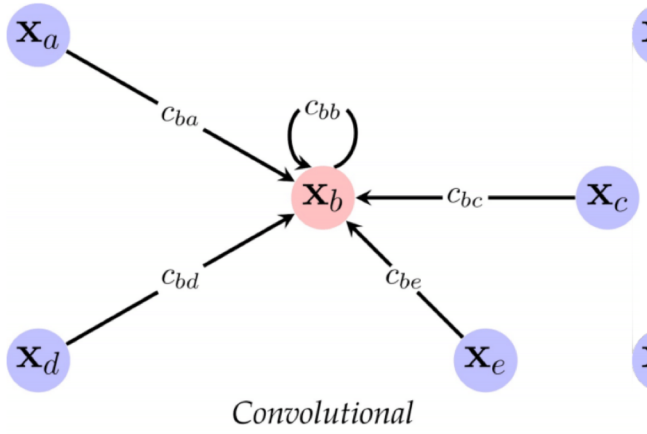
GNN Overall Architecture



Overall architecture of GNN

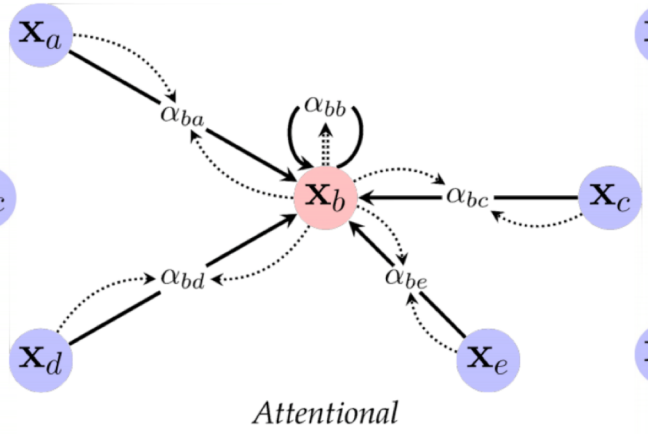
In all three flavours, permutation invariance is ensured by *aggregating* features from $\mathbf{X}_{\mathcal{N}_u}$ (potentially transformed, by means of some function ψ) with some permutation-invariant function $\bigoplus$, and then *updating* the features of node u, by means of some function ϕ. Typically, ψ and ϕ are learnable, whereas $\bigoplus$ is realised as a nonparametric operation such as sum, mean, or maximum, though it can also be constructed e.g. using recurrent neural networks (Murphy et al., 2018).

Three flavours of GNNs



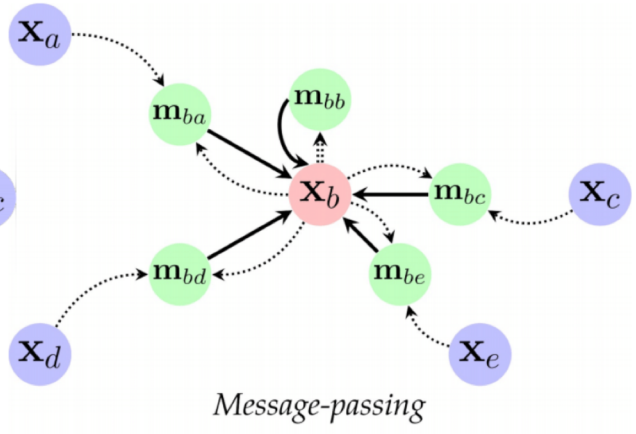
Convolutional

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij}\psi(\mathbf{x}_j)\right)$$



Attentional

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} a(\mathbf{x}_i, \mathbf{x}_j)\psi(\mathbf{x}_j)\right)$$



Message-passing

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} \psi(\mathbf{x}_i, \mathbf{x}_j)\right)$$

Deep Sets/ Transformers

As a form of Convolutional/Attentional GNN

Empty Edge Set (aka Deep Sets)

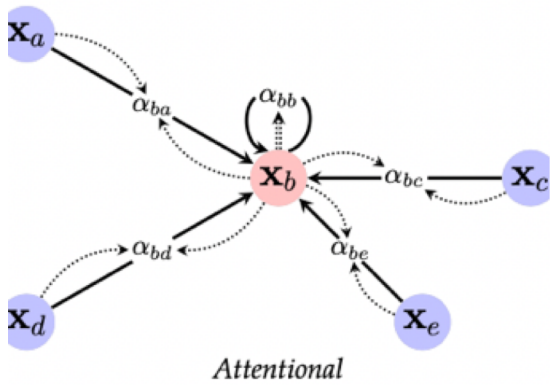
- Unordered sets do not pre-impose any additional structure or geometry
- Most natural way to process them is to treat each set element independently
- Similar way to process each set element motivates expression in the "Convolutional" flavor of GNNs

$$\mathbf{h}_u = \psi(\mathbf{x}_u),$$

where ψ is a learnable transformation. It may be observed that this is a special case of a convolutional GNN with $\mathcal{N}_u = \{u\}$—or, equivalently, $\mathbf{A} = \mathbf{I}$.

Complete Edge Set (aka Transformers)

- Elements in the set have some form of relational structure
- Links between each element dependent on the element itself
- Motivates expression using the “Attentional” flavor of GNNs
- Transformers can be viewed as inferring soft adjacency
  - Letting GNN choose its own edges



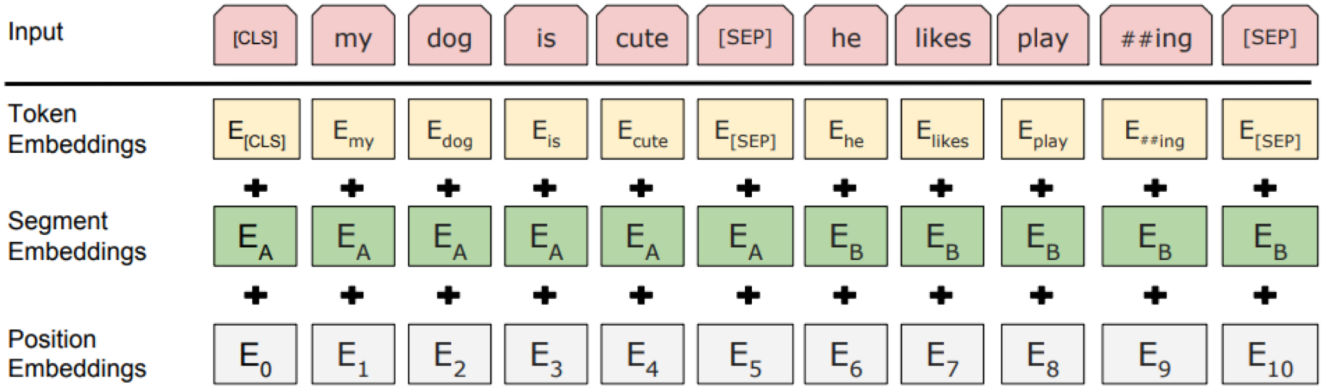
$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{V}} a(\mathbf{x}_u, \mathbf{x}_v)\psi(\mathbf{x}_v)\right)$$

Self-attention, typically constrained in the range [0, 1] using softmax

Where is the sequence info in Transformers?

- If Transformers are purely attentional GNNs over complete graph, there would be no information about sequence position

- This was explicitly given as an input to the model via positional embeddings

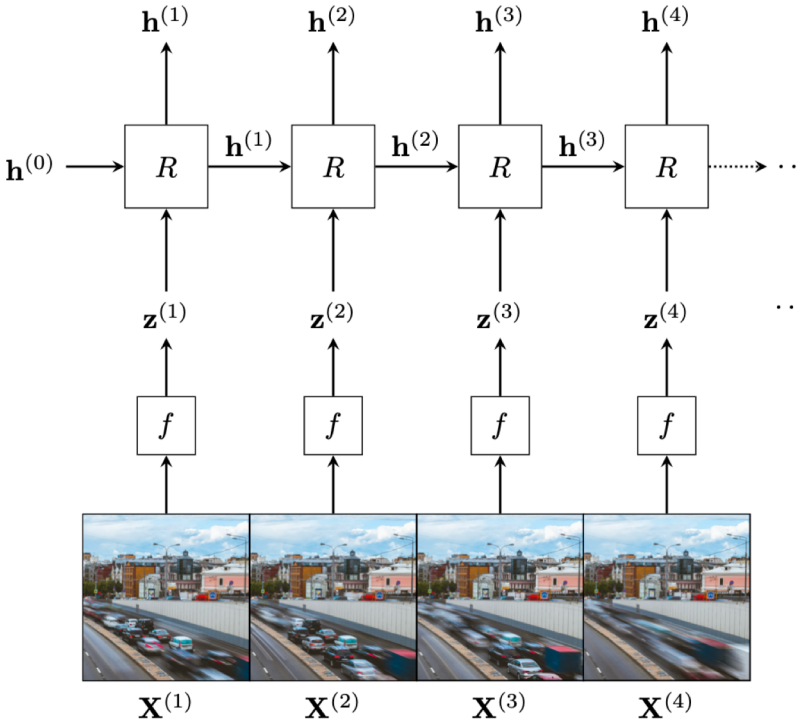


BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Devlin et al. (2018).

Recurrent Neural Networks (RNN)

Time Warping

Recurrent Neural Networks

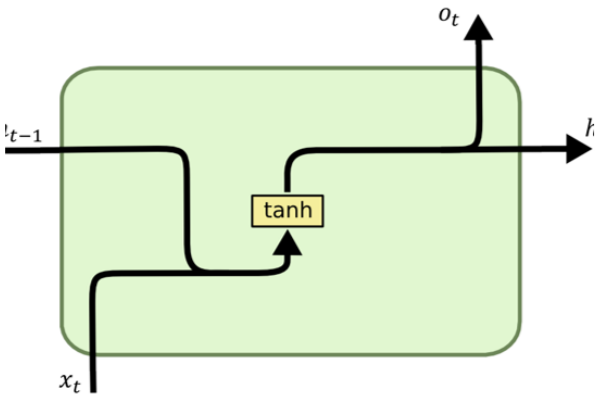


$$\mathbf{h}^{(t)} = R(\mathbf{z}^{(t)}, \mathbf{h}^{(t-1)})$$

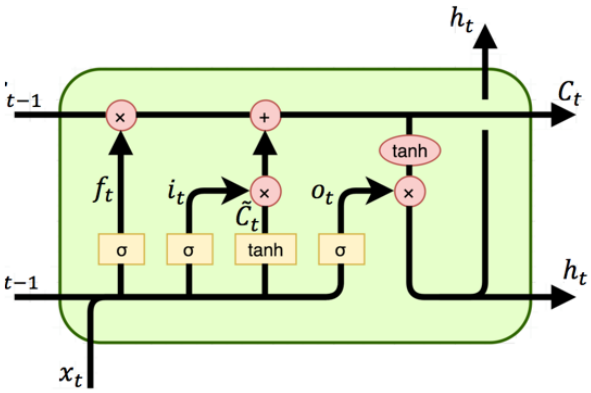
Output dependent only on current input, and previous hidden state

Types of RNN architecture

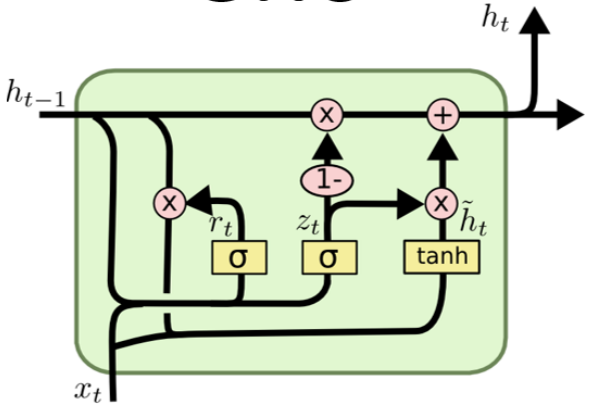
RNN



LSTM



GRU



Has the “Vanishing / Exploding Gradient Problem”

Taken from: http://dprogrammer.org/rnn-lstm-gru

Time Equivariance?

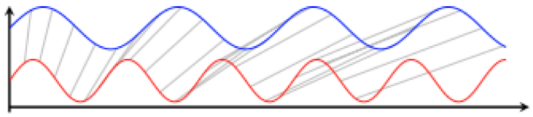
- Actually, **NOT time equivariant**, as the processing step at a particular time t is still dependent on the earlier time steps

- Shifting the order of the input can lead to changes in output

- Exception: Left-zero-padded sequence can be shifted left up to t' time steps without changing subsequent processing steps

$$\bar{\mathbf{z}}^{(t)} = \begin{cases} \mathbf{0} & t \leq t' \\ \mathbf{z}^{(t-t')} & t > t' \end{cases}$$

Time Warping (Math)



Such warping operations can be *simple*, such as time rescaling; e.g. $\tau(t) = 0.7t$

- RNN update function satisfies the differential equation:

$$\frac{\mathrm{d}h(t)}{\mathrm{d}t} = h(t+1) - h(t) = R(z(t+1), h(t)) - h(t)$$

- If time is warped via $t' = \tau(t)$, RNN should satisfy:

$$\frac{\mathrm{d}h(\tau(t))}{\mathrm{d}\tau(t)} = R(z(\tau(t+1)), h(\tau(t))) - h(\tau(t))$$

$$\frac{\mathrm{d}h(\tau(t))}{\mathrm{d}t} = \frac{\mathrm{d}h(\tau(t))}{\mathrm{d}\tau(t)}\frac{\mathrm{d}\tau(t)}{\mathrm{d}t} = \frac{\mathrm{d}\tau(t)}{\mathrm{d}t}R(z(\tau(t+1)), h(\tau(t))) - \frac{\mathrm{d}\tau(t)}{\mathrm{d}t}h(\tau(t))$$

- Using First Order Taylor Expansion, we get:

$$\begin{aligned}\mathbf{h}^{(t+1)} &= \mathbf{h}^{(t)} + \frac{\mathrm{d}\tau(t)}{\mathrm{d}t}R(\mathbf{z}^{(t+1)}, \mathbf{h}^{(t)}) - \frac{\mathrm{d}\tau(t)}{\mathrm{d}t}\mathbf{h}^{(t)} \\ &= \frac{\mathrm{d}\tau(t)}{\mathrm{d}t}R(\mathbf{z}^{(t+1)}, \mathbf{h}^{(t)}) + \left(1 - \frac{\mathrm{d}\tau(t)}{\mathrm{d}t}\right)\mathbf{h}^{(t)}\end{aligned}$$

- Expressed as a learnable function $\tau(.)$, we get:

$$\mathbf{h}^{(t+1)} = \Gamma(\mathbf{z}^{(t+1)}, \mathbf{h}^{(t)})R(\mathbf{z}^{(t+1)}, \mathbf{h}^{(t)}) + (1 - \Gamma(\mathbf{z}^{(t+1)}, \mathbf{h}^{(t)}))\mathbf{h}^{(t)}$$

Only possible with Gated RNNs

Time Warping (Non-math)

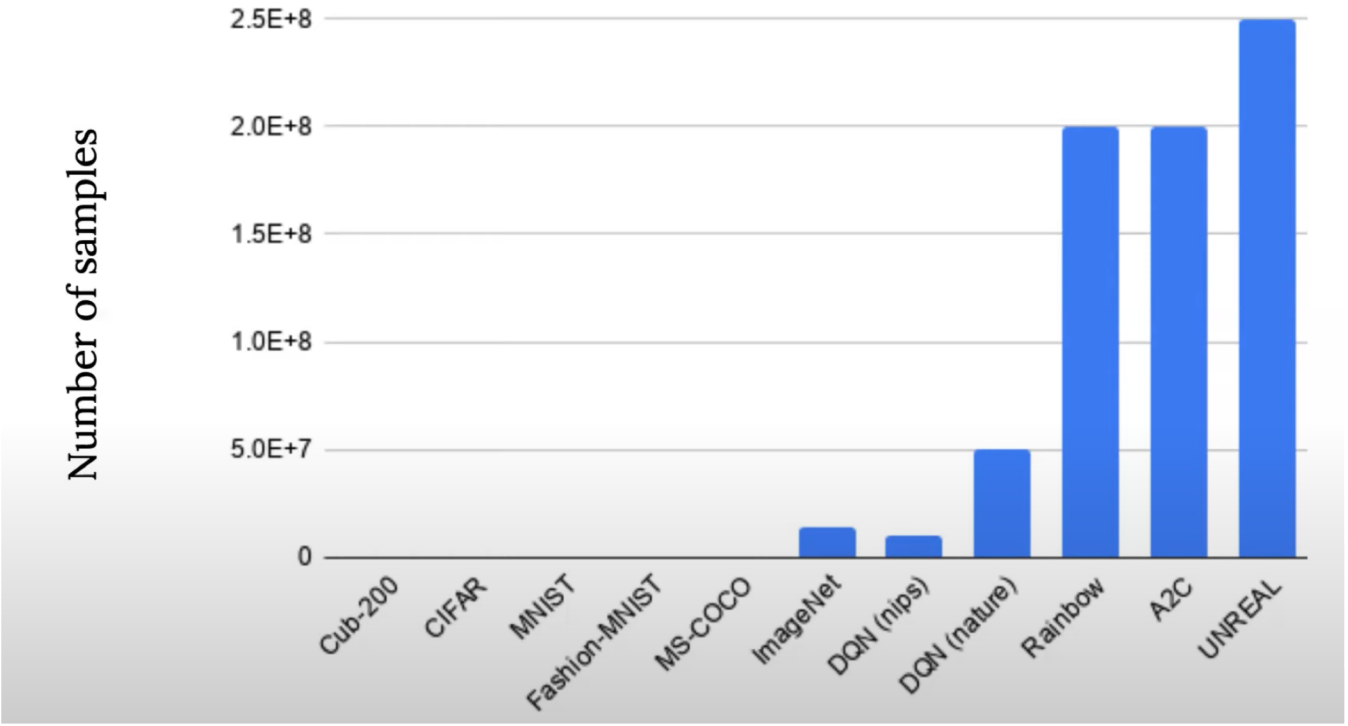
- RNN tends to lose information as there is a bottleneck of the hidden state vector encompassing information of all previous time steps
- If we have gates to allow for conditional flow of input through the model, then there will be a way to model long-term dependencies without losing information
- Hence, if the input sequence is put further apart, the RNN will also learn it and just ignore the expanded sequence
  - Original: The cat sat on a mat
  - Expanded: The <unk> cat <unk> sat <unk> on <unk> a <unk> mat
- BUT the warping must be trained together with the model for it to work. Same model can work for other warpings after training with it.

Reinforcement Learning

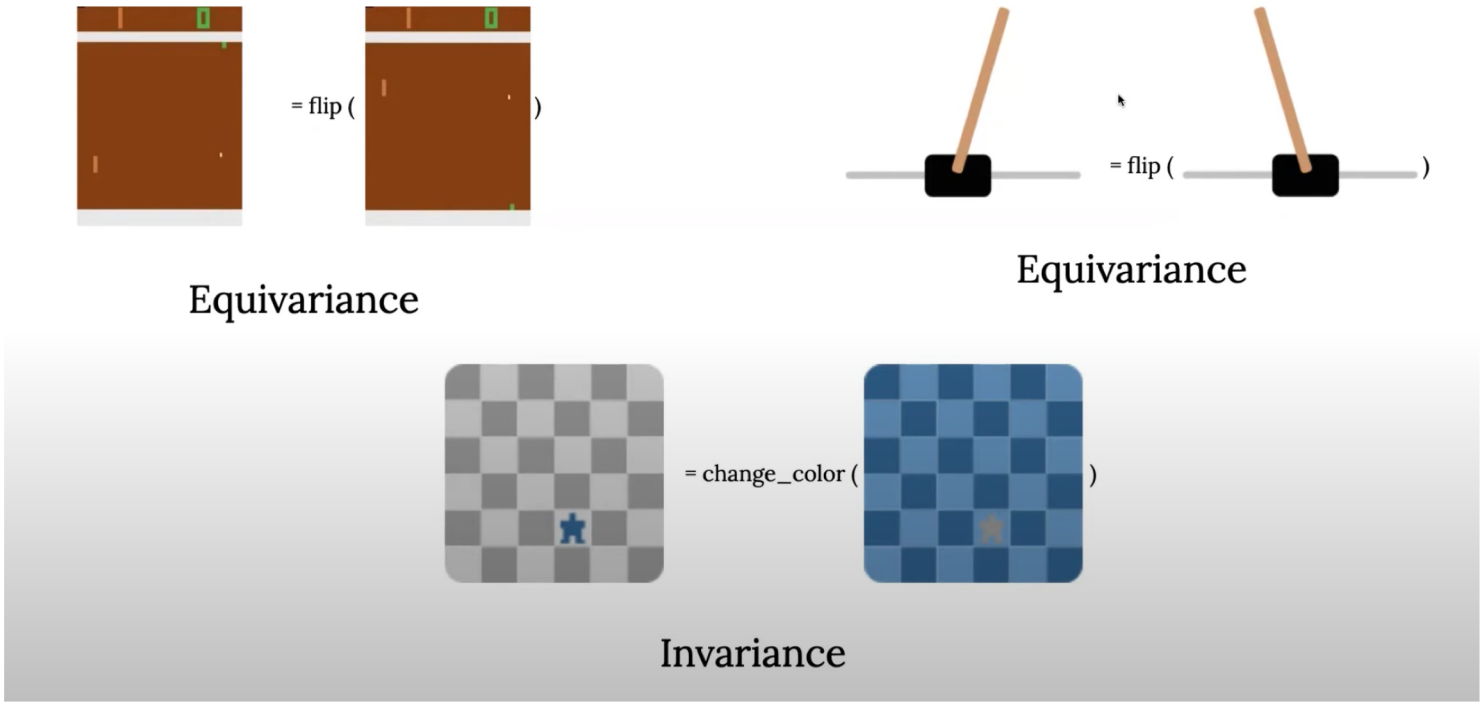
Learn more with less

(Content from AMMI Course Seminar 1 by Elise van der Pol)

RL is very data hungry

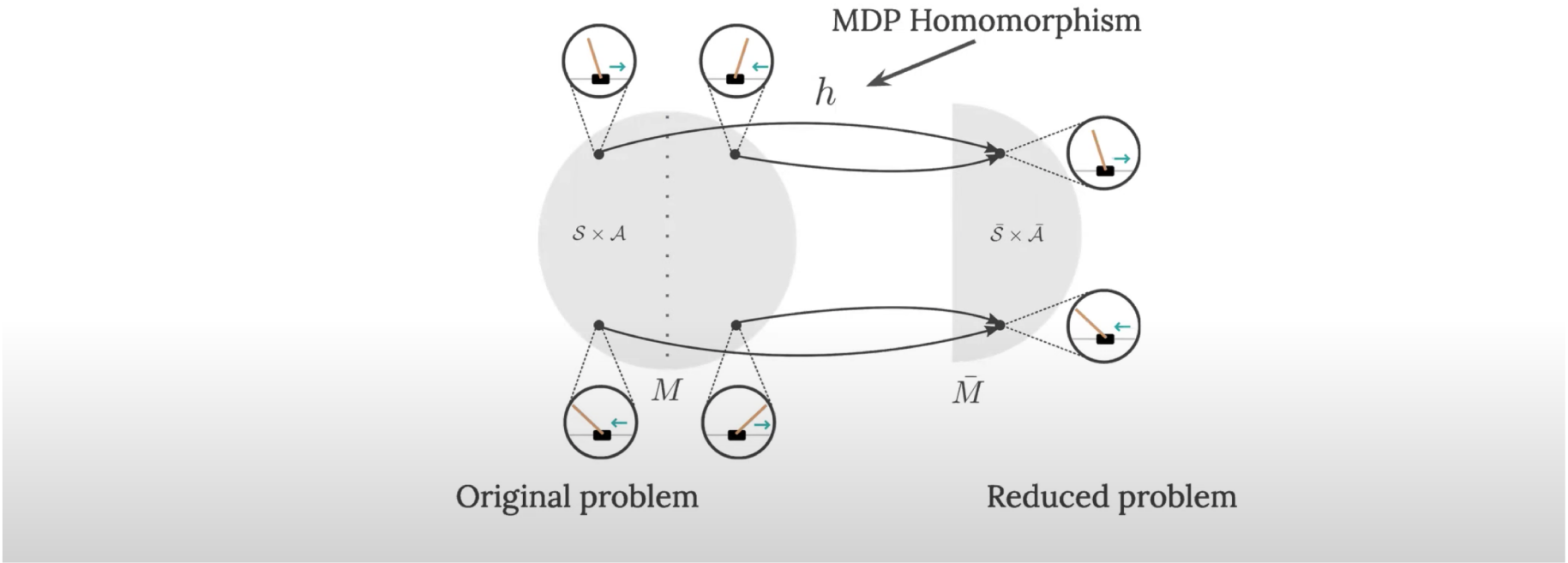


Exploiting equivariance/invariance in data



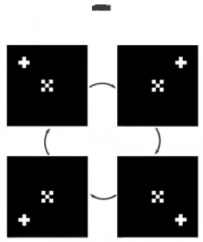
Markov Decision Process Homomorphisms

Map ground MDP → abstract MDP, preserve dynamics (Ravindran & Barto 2001)

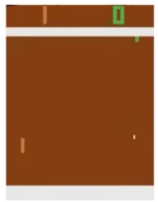


Fewer interactions with world to learn

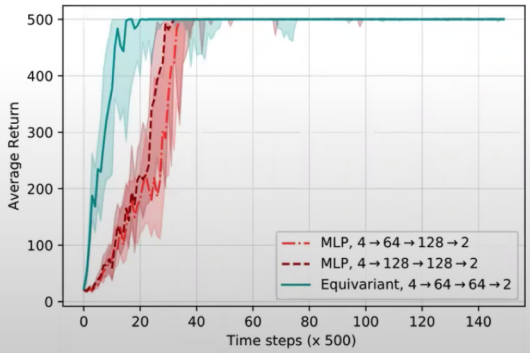
Cartpole
2 element symmetry group



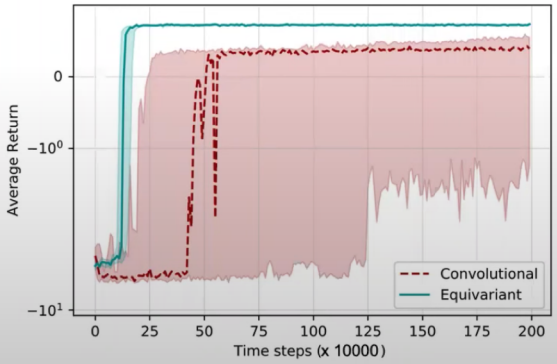
Grid World
4 element symmetry group



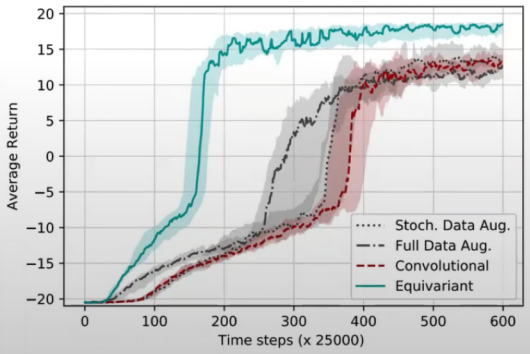
Pong
2 element symmetry group



PPO



A2C



PPO

Summary

Summary

- Structure explicitly encodes the invariances of processing the data
- Invariances can be implicitly (not sample efficient) encoded via perturbing the inputs through data augmentation
- Invariances can be implicitly encoded as a separate input feature to the model
- The more invariances encoded, the less general the model
  - CNNs can handle images well, but do not do well for tabular data in general
- Invariances/equivariances help to improve model convergence speed

Discussion