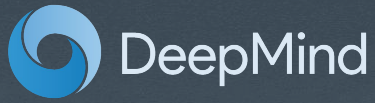
#TMLSS 2018

Graph Networks: relational inductive biases for deep learning

Razvan Pascanu
razp@google.com



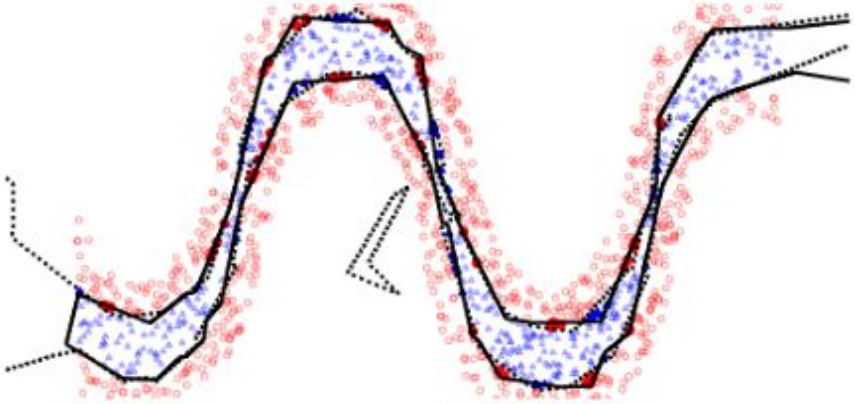
About slides

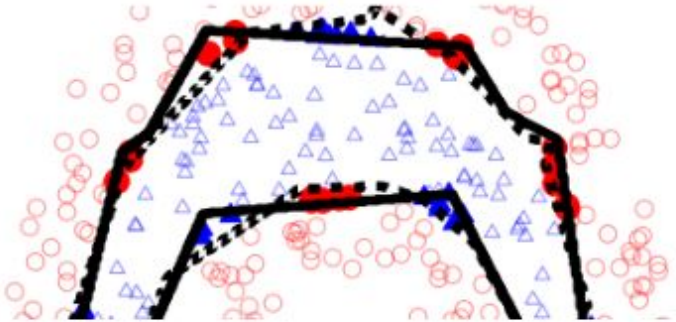
- About slides:
  - Interrupt me whenever you have questions -- unless you think is not worth the interruption :)
  - Last lecture of the school, I know you are tired, so let me know if I'm jumping through slides too fast
  - Be aware of inherent bias (as with anything)

Representational power of NN

— Shallow (20 units)

– - Deep (2x10 units)

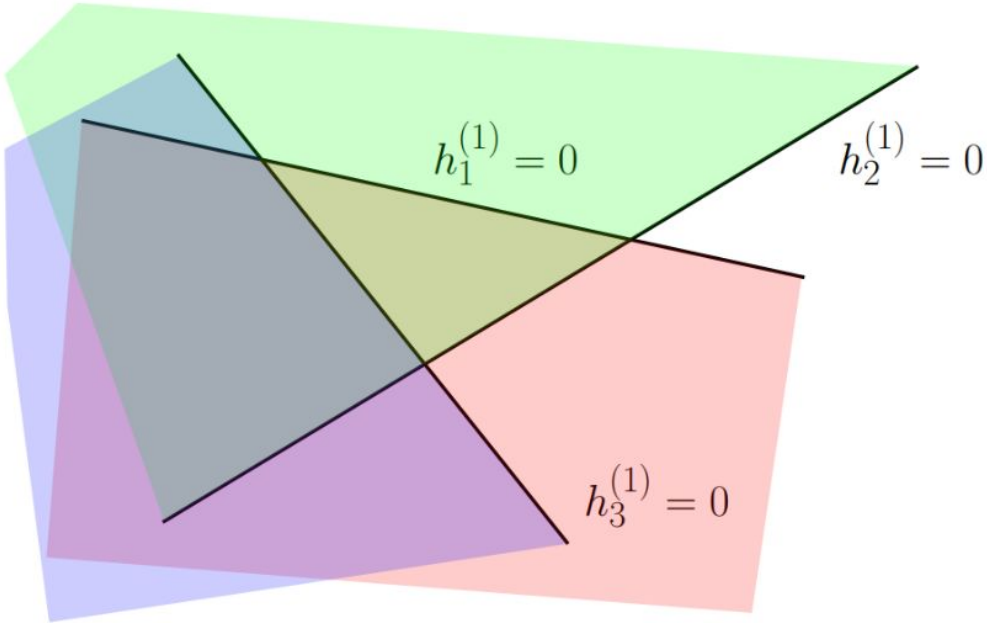




https://arxiv.org/abs/1402.1869
See also my phd thesis

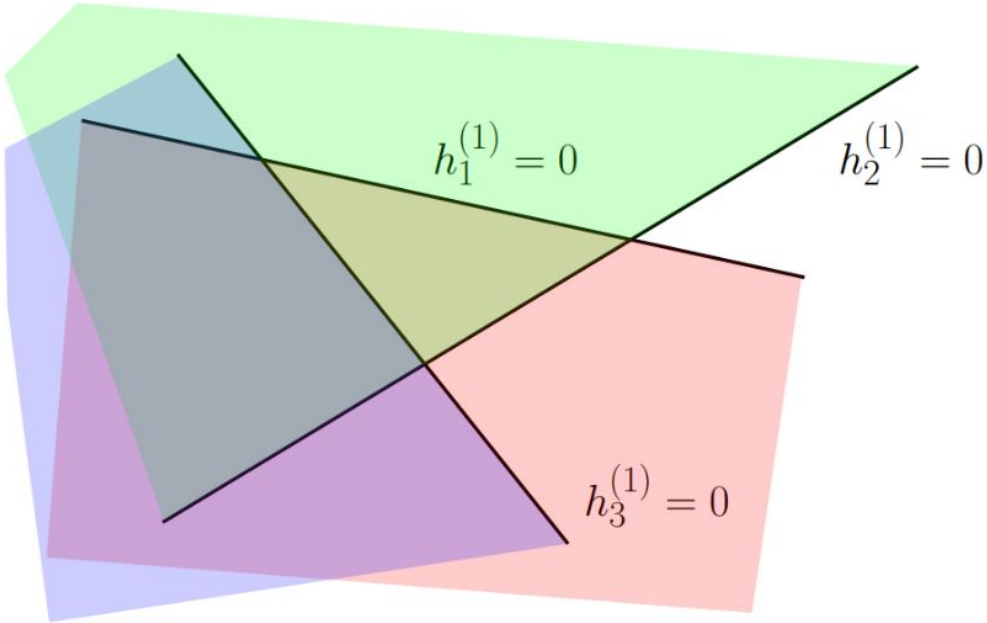
Single layer MLPs

$$\mathbb{I}_{x>b}(x) = \begin{cases} 1 & \text{, iff } x > b \\ 0 & \text{, otherwise} \end{cases}$$



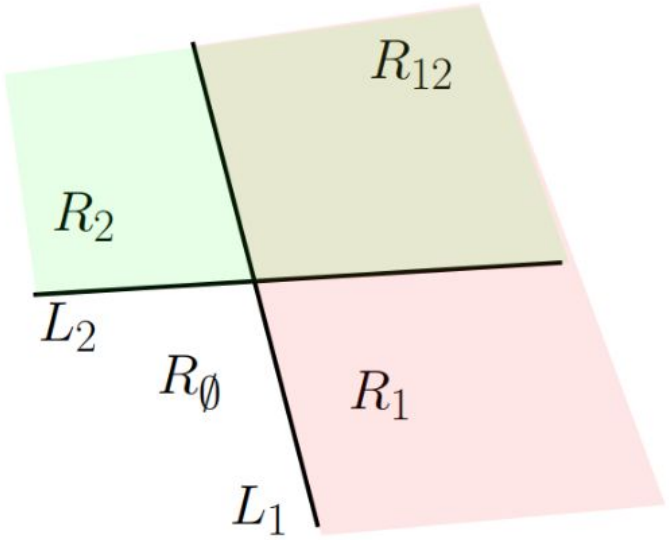
Single layer MLPs

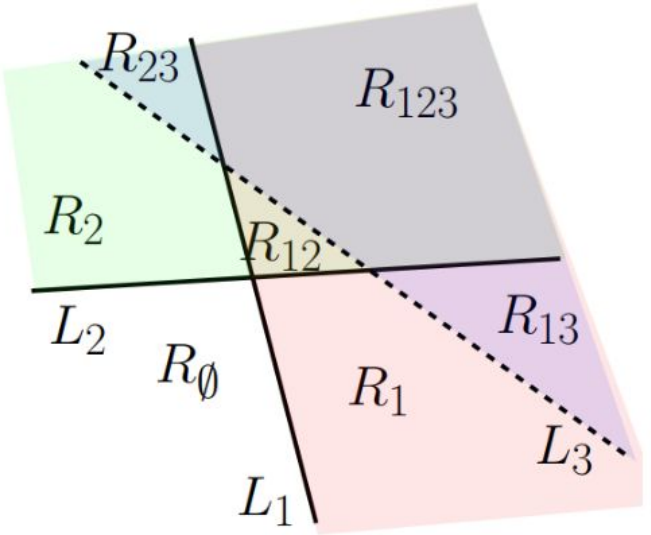
$$r(\mathcal{A}_m) = \binom{m}{2} + m + 1.$$



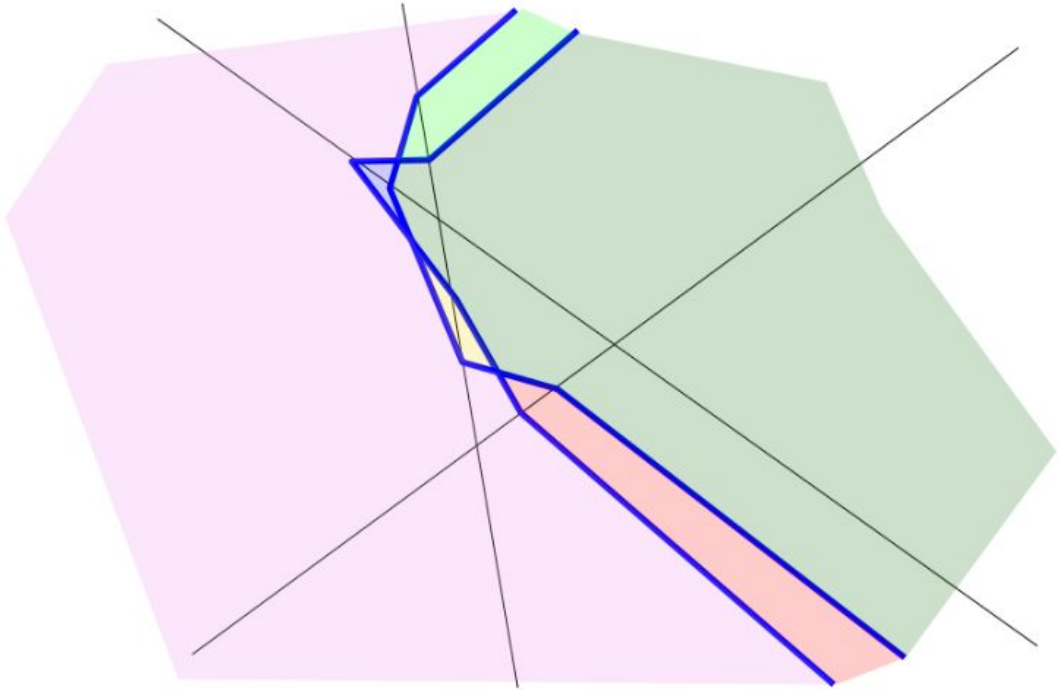
The proof

$$r(\mathcal{A}_m) = \binom{m}{2} + m + 1.$$

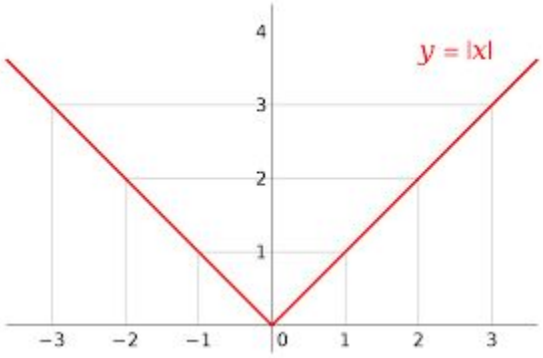




But deep models can do better..



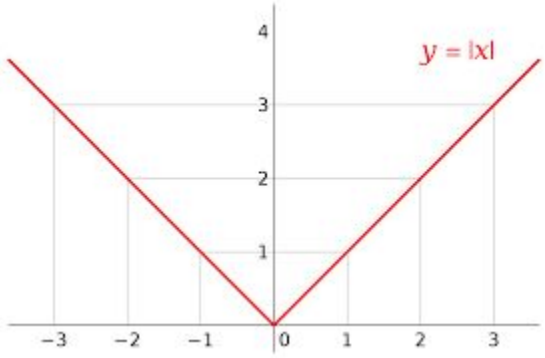
How do deep models do this?



Absolute value

How do deep models do this?

$$f_1(\mathbf{x}) = \max\{0, [+1, 0]\,\mathbf{x}\},$$
$$f_2(\mathbf{x}) = \max\{0, [-1, 0]\,\mathbf{x}\},$$
$$f_3(\mathbf{x}) = \max\{0, [0, +1]\,\mathbf{x}\},$$
$$f_4(\mathbf{x}) = \max\{0, [0, -1]\,\mathbf{x}\},$$

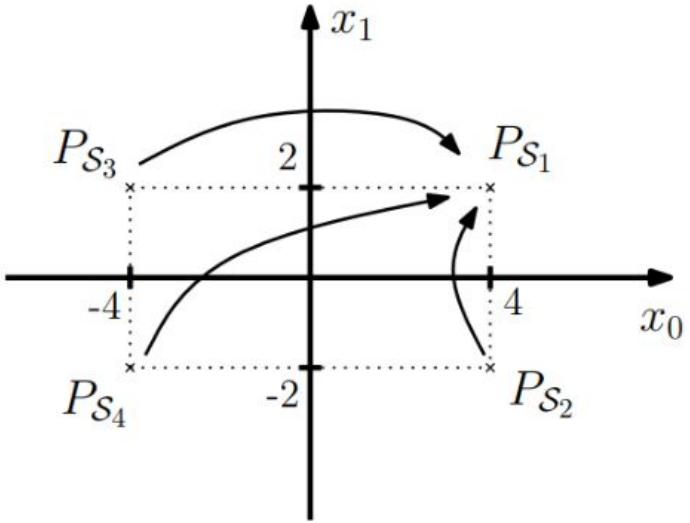


Absolute value

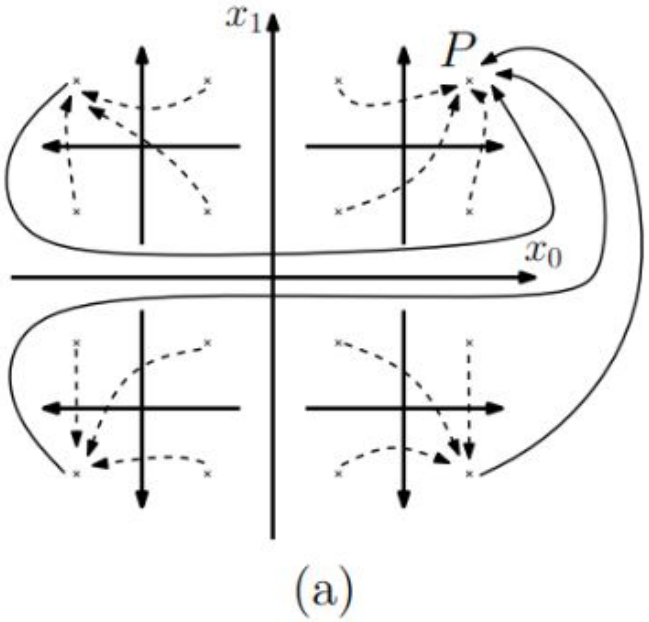
$$\begin{bmatrix} g_1(\mathbf{x}) \\ g_2(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ f_3(\mathbf{x}) \\ f_4(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \mathrm{abs}(x_1) \\ \mathrm{abs}(x_2) \end{bmatrix}.$$

How do deep models do this?

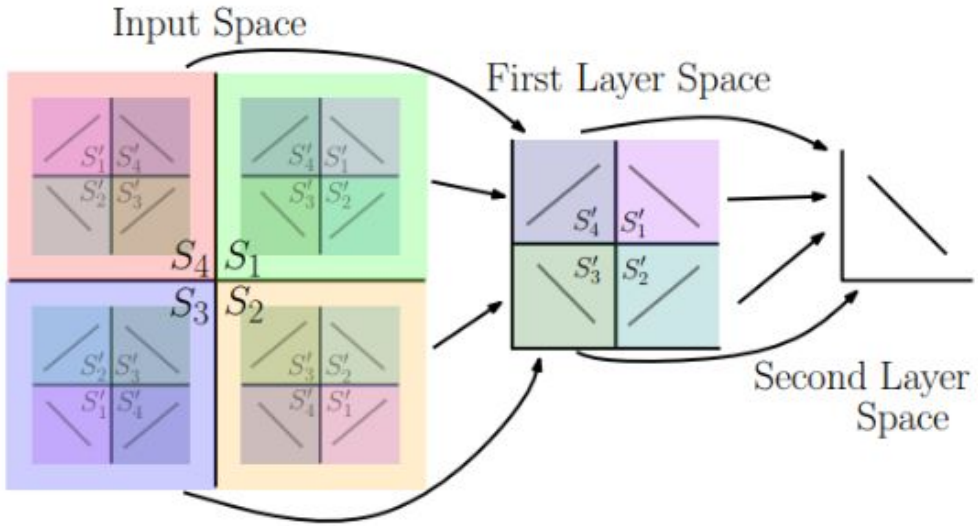
$$\mathcal{S}_1 = \{(x_1, x_2) \mid x_1 \geq 0, x_2 \geq 0\}$$
$$\mathcal{S}_2 = \{(x_1, x_2) \mid x_1 \geq 0, x_2 < 0\}$$
$$\mathcal{S}_3 = \{(x_1, x_2) \mid x_1 < 0, x_2 \geq 0\}$$
$$\mathcal{S}_4 = \{(x_1, x_2) \mid x_1 < 0, x_2 < 0\}.$$



How do deep models do this?

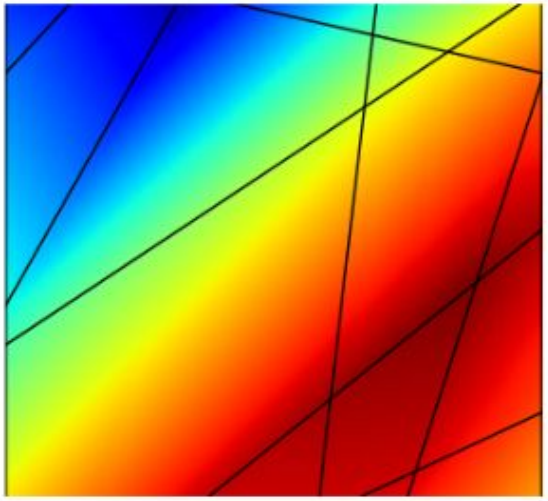


(a)

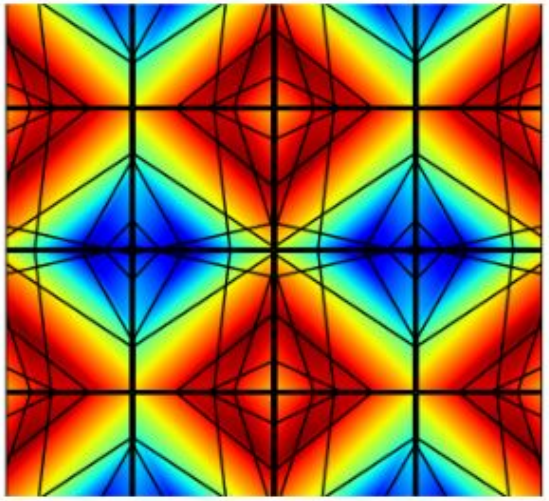


(b)

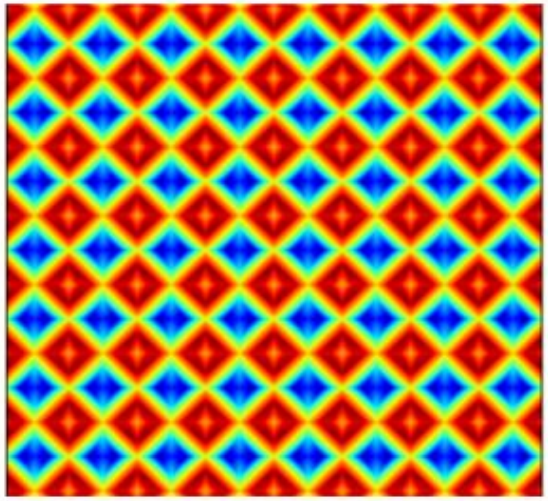
How do deep models do this?



(a)



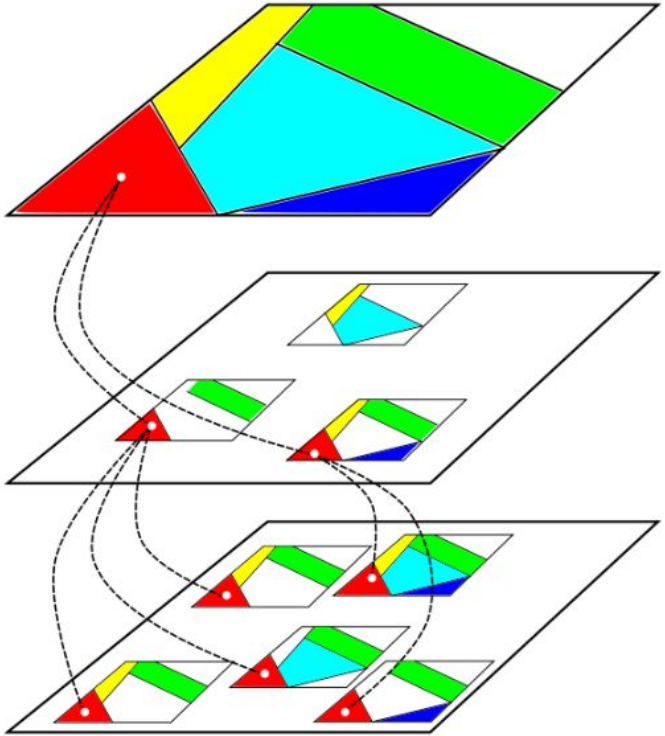
(b)



(c)

How do deep models do this?

- Is this clear?
- Does it make sense?



How do deep models do this?

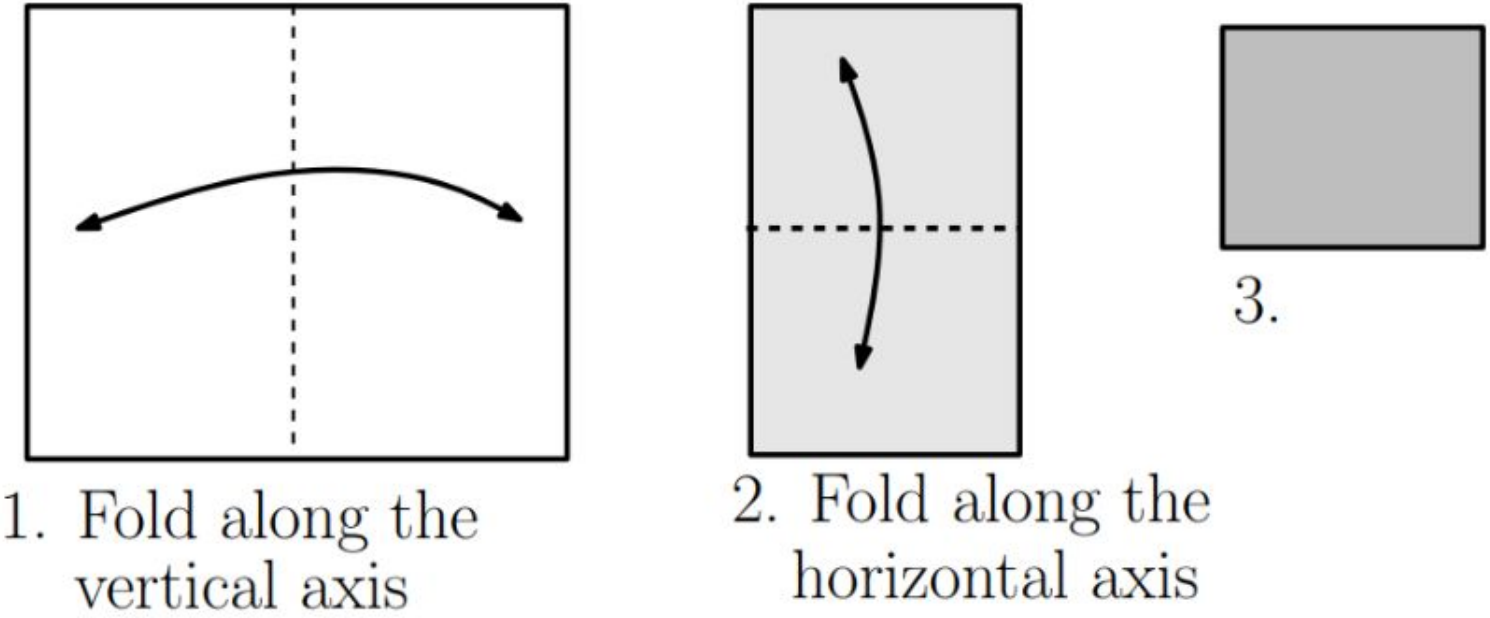


Figure 3.14: Space folding of 2–D Euclidean space along the two axes.

How do deep models do this?

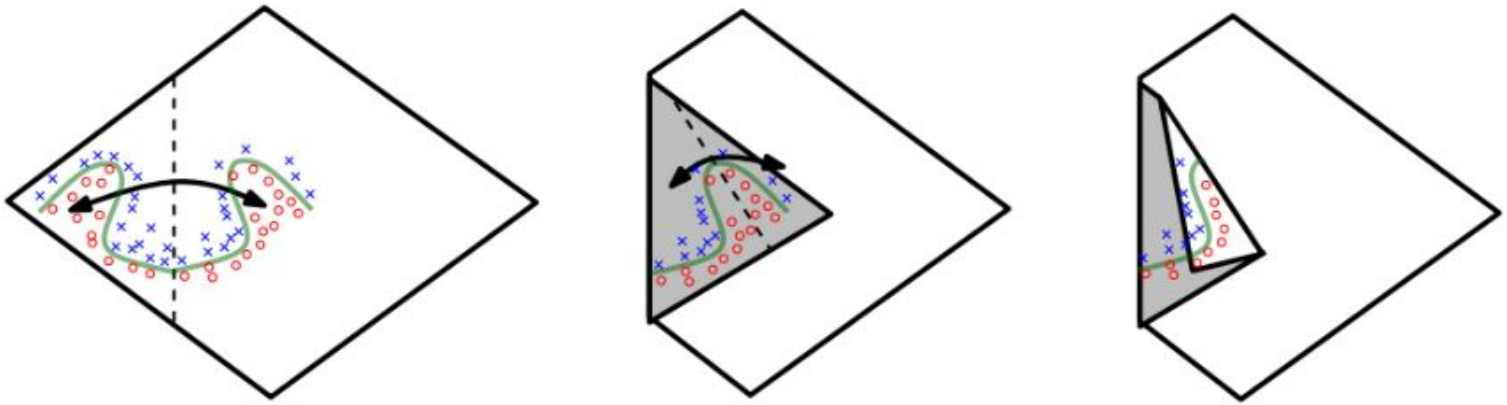
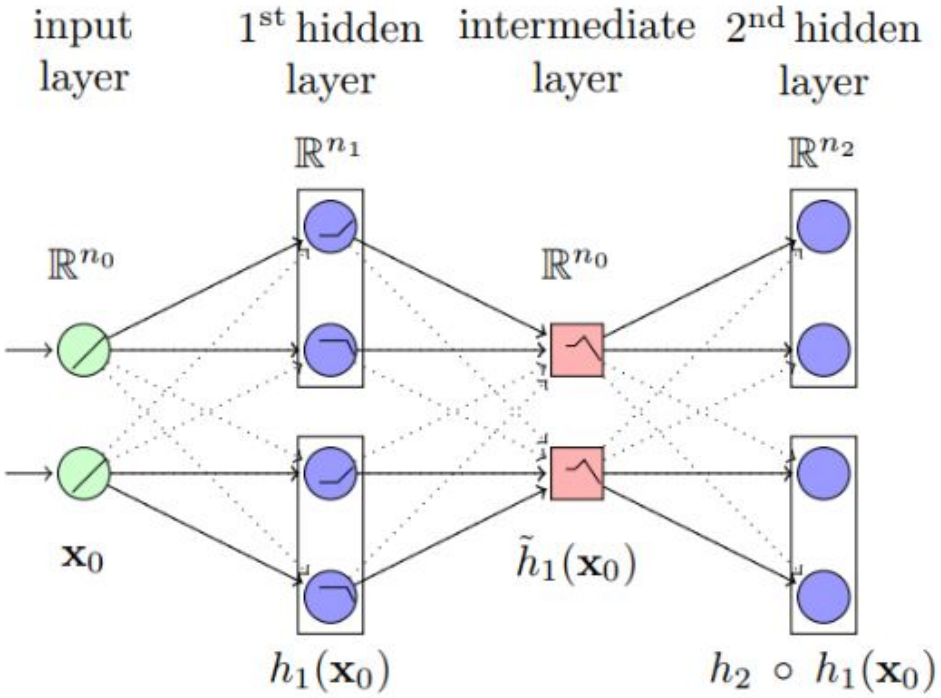
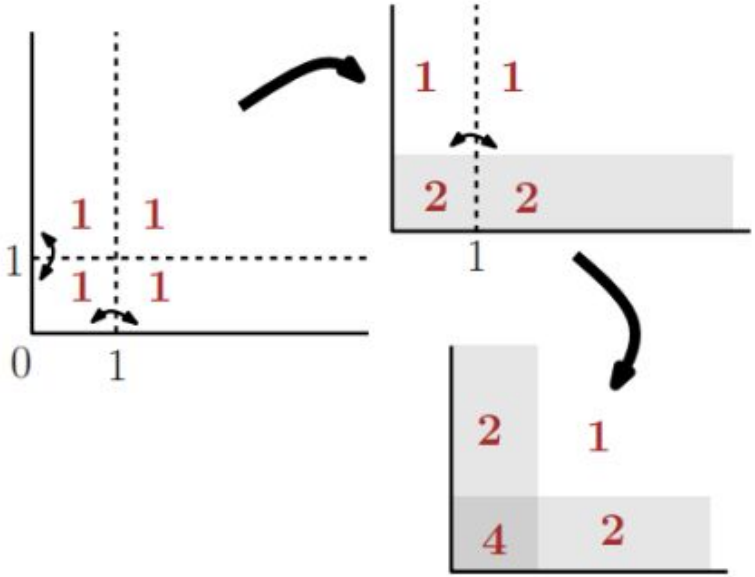


Figure 3.15: Space folding of 2–D space in a non-trivial way. Note how the folding can potentially identify symmetries in the boundary that it needs to learn.

How do deep models do this?





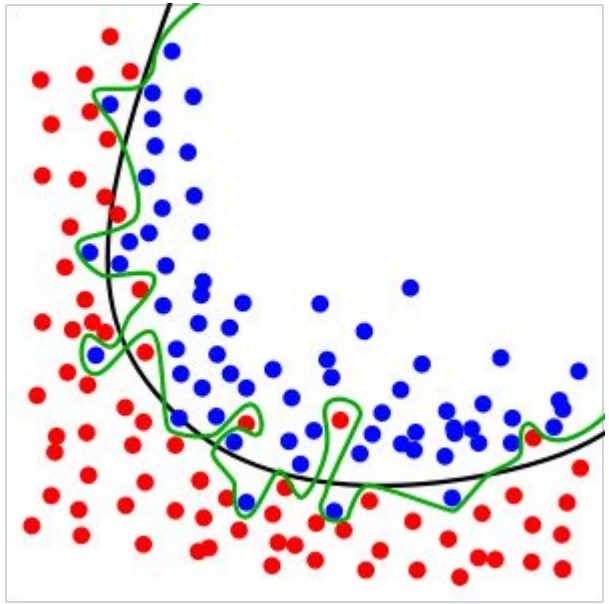
How do deep models do this?

- For a deep model: $\Omega\left(\left(n/n_0\right)^{n_0(L-1)} \frac{n^{n_0-2}}{L}\right)$.
- For a shallow model: $O\left(L^{n_0-1} n^{n_0-1}\right)$.

But are we happy with this? Do we want these kind of numbers!?

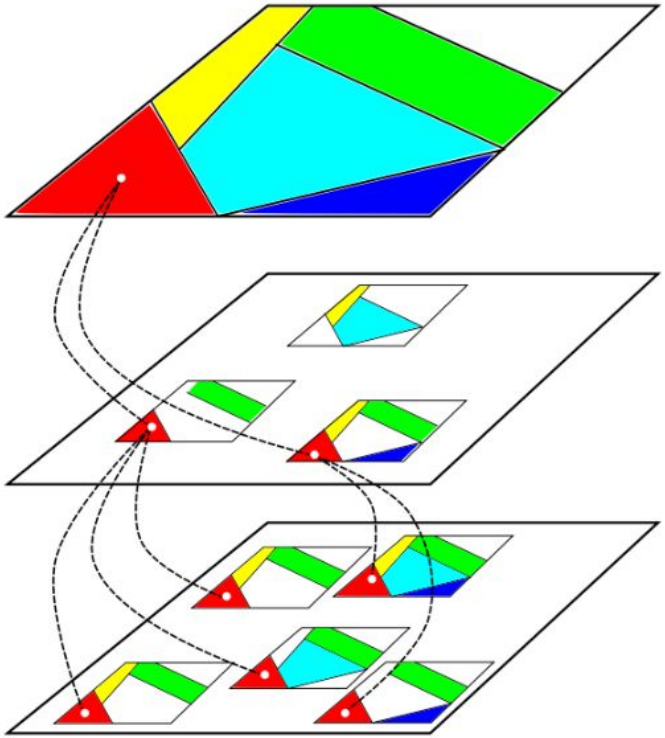
What value is this for n=100; n_0=25; L=3 ?

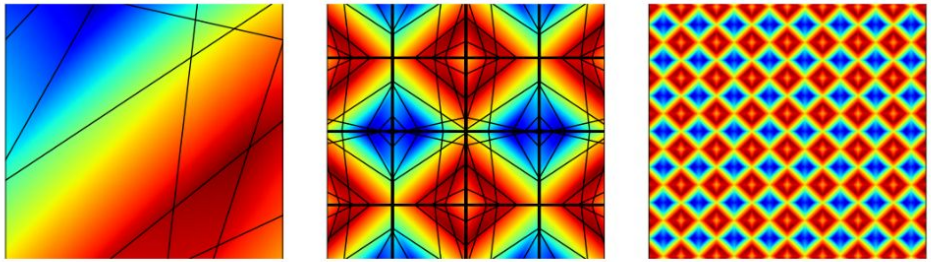
Overfitting!?



- Why not?

Symmetries in response





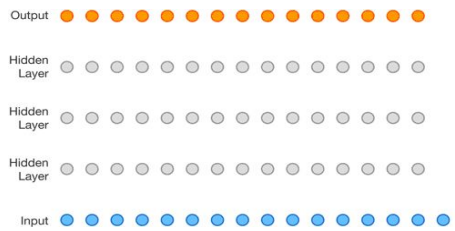
There are:

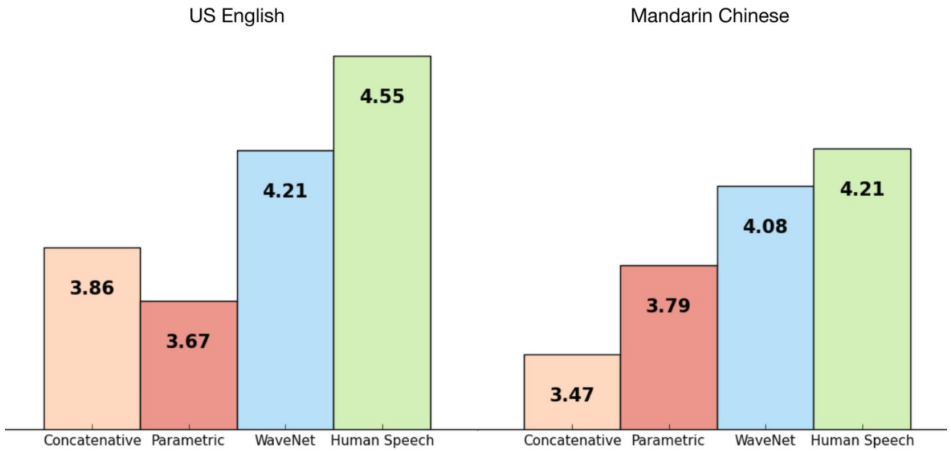
- Symmetries
- Structure
- prior

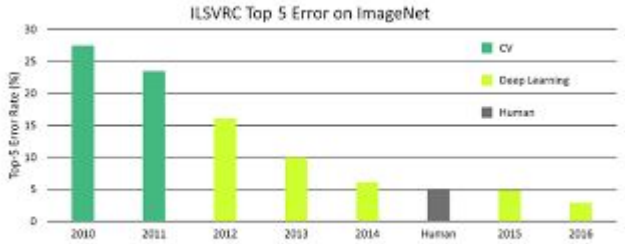
Strawman argument: Deep Learning mantra

- End-to-end learning
- No hand-engineering, no priors, no knowledge of the task

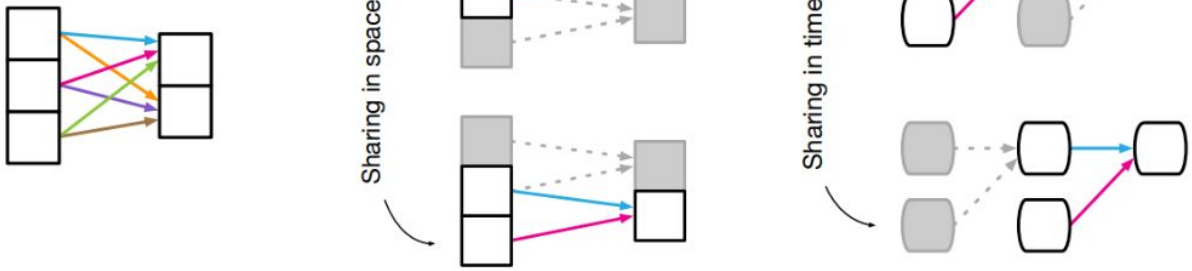
E.g.: pixel-cnn, wavenet







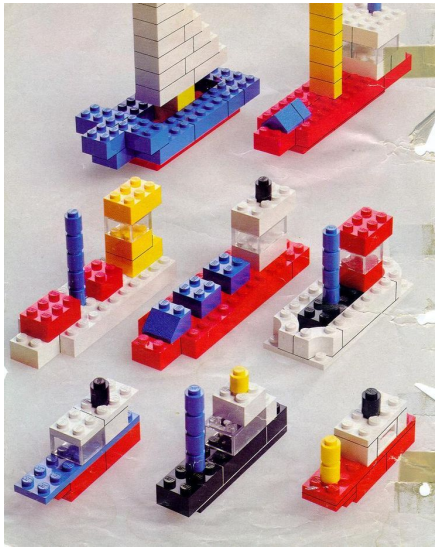
Structural bias in DL

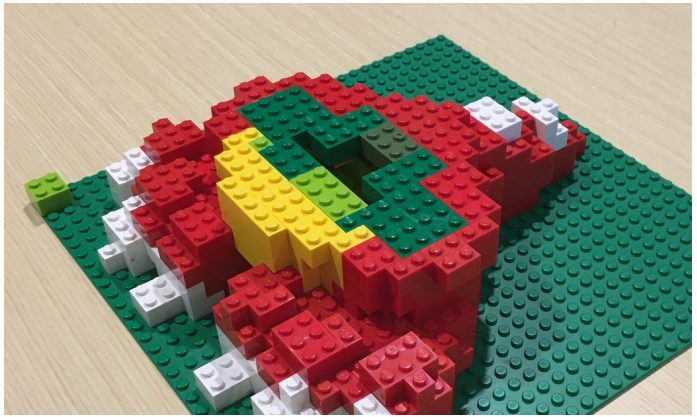


(a) Fully connected (b) Convolutional (c) Recurrent

Focusing: Combinatorical generalization

- “Infinite use of finite means” (Humboldt, 1836; Chomsky 1965)
- World is compositional (or we understand it in compositional terms)





Defining a “relational bias”

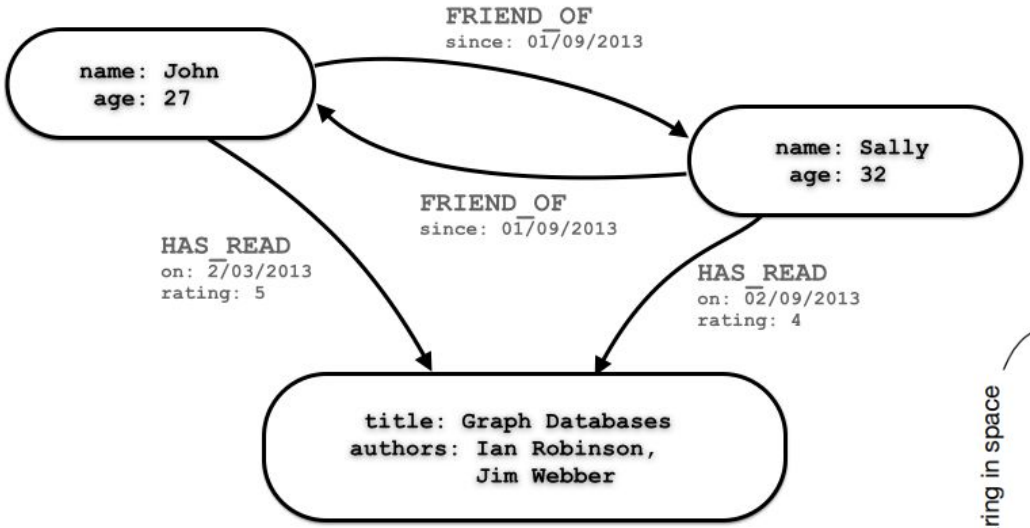
Box 1: Relational reasoning

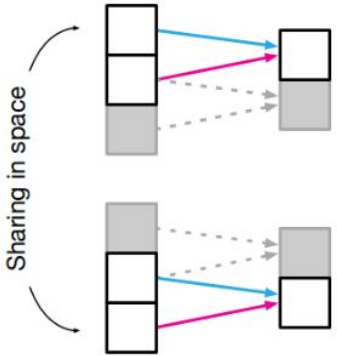
We define *structure* as the product of composing a set of known building blocks. “Structured representations” capture this composition (i.e., the arrangement of the elements) and “structured computations” operate over the elements and their composition as a whole. Relational reasoning, then, involves manipulating structured representations of *entities* and *relations*, using *rules* for how they can be composed. We use these terms to capture notions from cognitive science, theoretical computer science, and AI, as follows:

- An *entity* is an element with attributes, such as a physical object with a size and mass.
- A *relation* is a property between entities. Relations between two objects might include SAME SIZE AS, HEAVIER THAN, and DISTANCE FROM. Relations can have attributes as well. The relation MORE THAN X TIMES HEAVIER THAN takes an attribute, X, which determines the relative weight threshold for the relation to be TRUE vs. FALSE. Relations can also be sensitive to the global context. For a stone and a feather, the relation FALLS WITH GREATER ACCELERATION THAN depends on whether the context is IN AIR vs. IN A VACUUM. Here we focus on pairwise relations between entities.
- A *rule* is a function (like a non-binary logical predicate) that maps entities and relations to other entities and relations, such as a scale comparison like IS ENTITY X LARGE? and IS ENTITY X HEAVIER THAN ENTITY Y?. Here we consider rules which take one or two arguments (unary and binary), and return a unary property value.

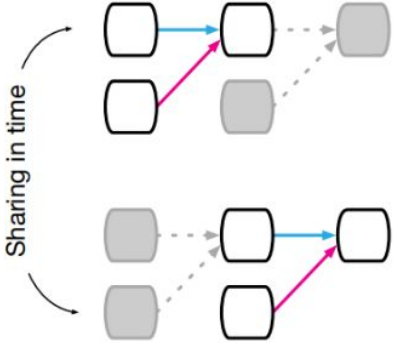
Relational inductive biases, deep learning, and graph nets. Battaglia et al.2018
https://arxiv.org/pdf/1806.01261.pdf

Relational reasoning and graphs





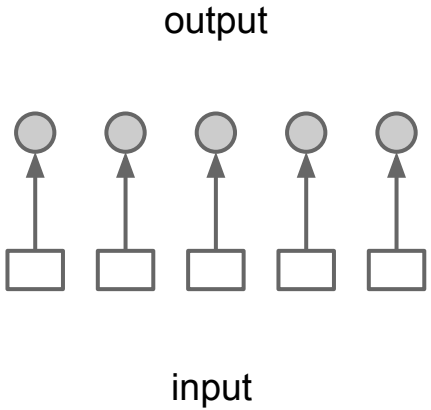
(b) Convolutional



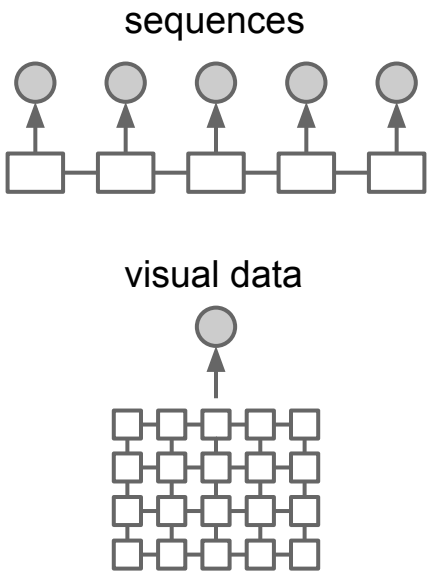
(c) Recurrent

What about deep learning?

Unstructured Data

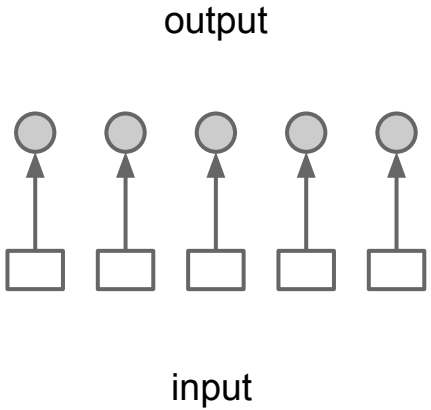


Data with Rigid Structure

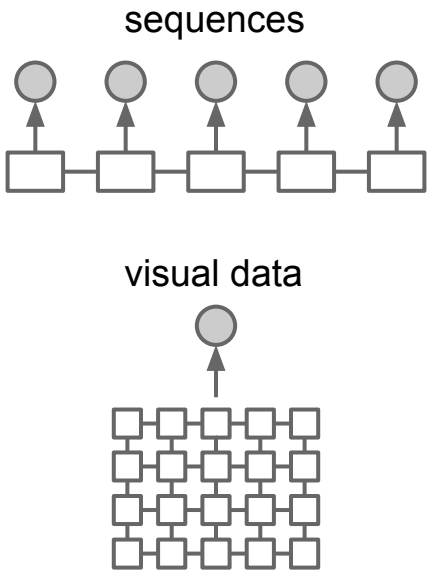


What about deep learning?

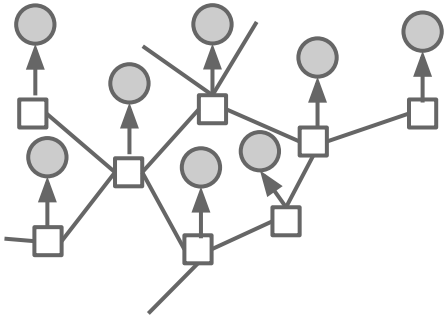
Unstructured Data



Data with Rigid Structure

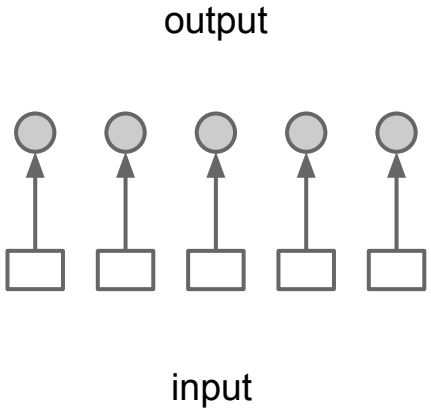


Graph Structured Data

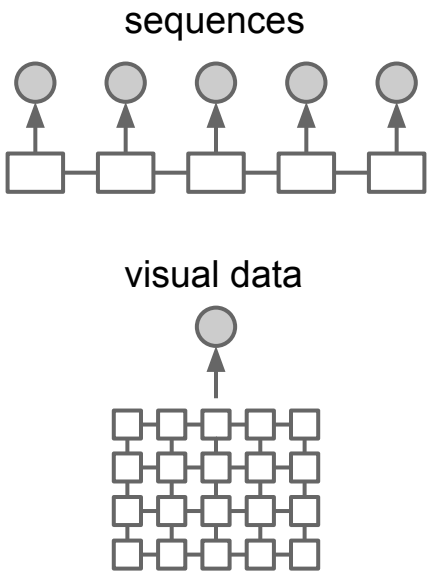


What about deep learning?

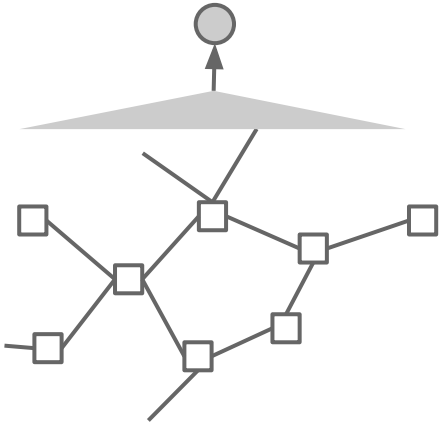
Unstructured Data



Data with Rigid Structure



Graph Structured Data



Graph Structured Data & DL

Graph Neural Networks (Gori et al., 2005; Scarselli et al., 2009) - Earliest graph neural net models

Graph (Spectral) Convolution (Bruna et al. 2013) - The first work that generalized convolutions to graphs in the spectral domain

Neural Graph Fingerprints (Duvenaud et al. 2015) - A special instantiation of graph nets for predicting chemical properties

Gated Graph (Sequence) Neural Networks (Li et al. 2015) - Node gating, various output modules and BPTT training

Interaction Networks (Battaglia et al. 2016) - Two-sided messages

Graph Conv Nets (Kipf et al. 2017) - First order approximation to spectral graph convolution

Message Passing Neural Nets (Gilmer et al. 2017) - A framework for unifying message-passing based neural networks

Graph Attention Networks (Velickovic et al. 2018) - Attention mechanism for aggregating messages

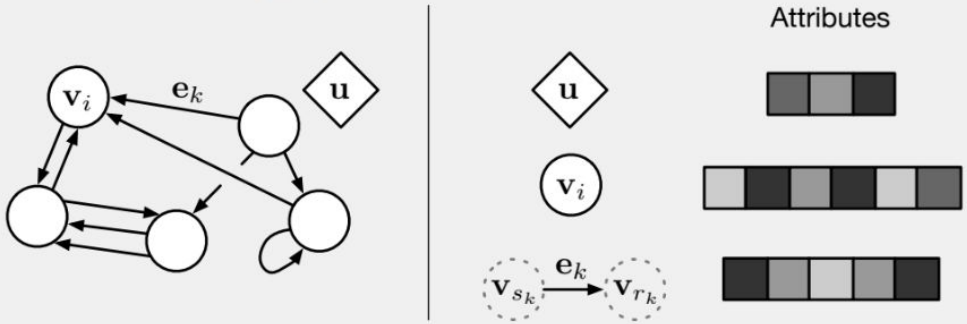
Deep Sets (Zaheer et al. 2017) / PointNet (Qi et al. 2017) - no messages on edges

Relation Net (Santoro et al. 2017) - no recurrence, fully connected graph, aggregate on edges

Self-Attention (in transformer / tensor2tensor) (Vaswani et al. 2017) - fully connected graph, attention mechanism for aggregating messages

Graph Networks

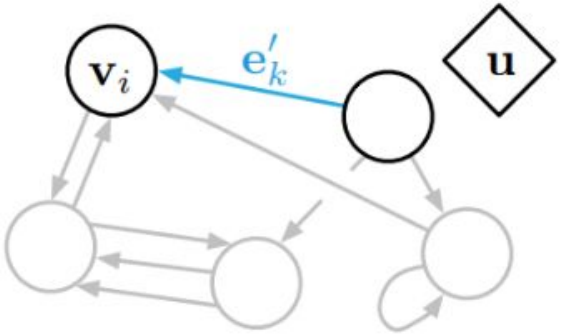
Box 3: Our definition of "graph"



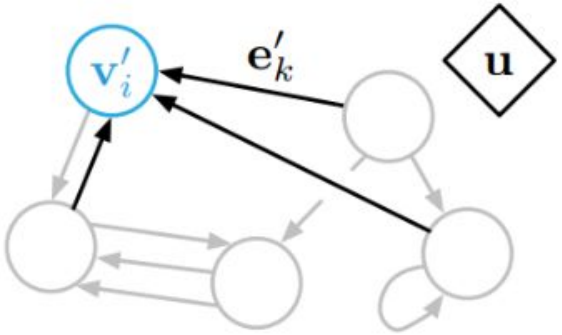
Here we use "graph" to mean a directed, attributed multi-graph with a global attribute. In our terminology, a node is denoted as $\mathbf{v}_i$, an edge as $\mathbf{e}_k$, and the global attributes as $\mathbf{u}$. We also use s_k and r_k to indicate the indices of the sender and receiver nodes (see below), respectively, for edge k. To be more precise, we define these terms as:

Directed : one-way edges, from a "sender" node to a "receiver" node.
Attribute : properties that can be encoded as a vector, set, or even another graph.
Attributed : edges and vertices have attributes associated with them.
Global attribute : a graph-level attribute.
Multi-graph : there can be more than one edge between vertices, including self-edges.

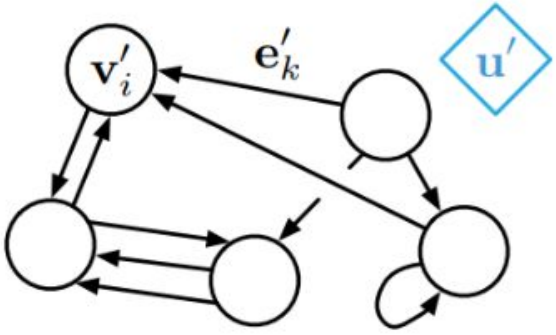
Graph Networks



(a) Edge update

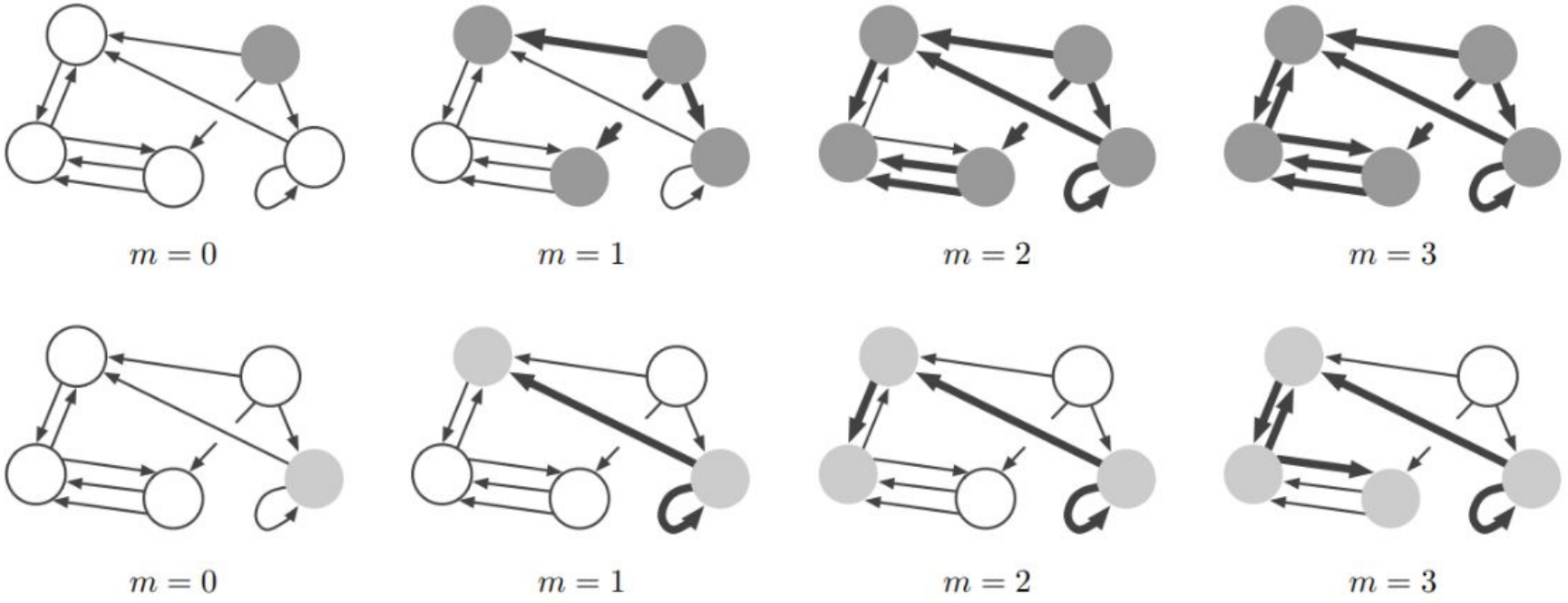


(b) Node update

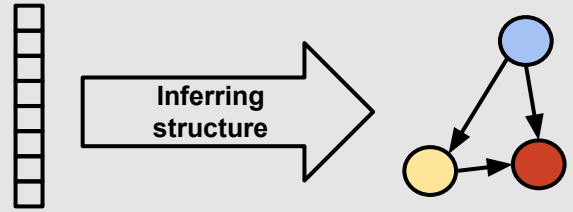


(c) Global update

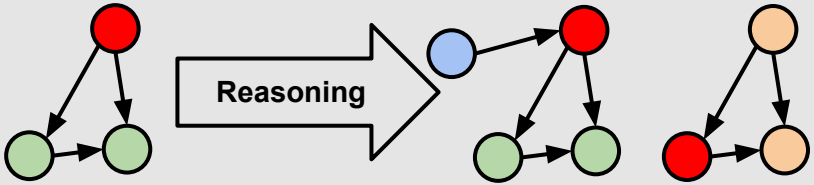
Graph Networks



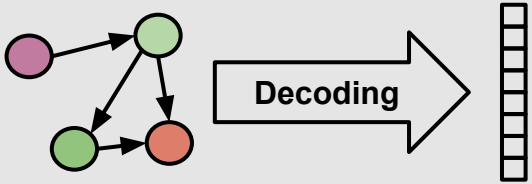
Tasks on graphs categorization



Inferring structure from unstructured data



Reasoning about structure



Decoding structure

Jumping through a few examples

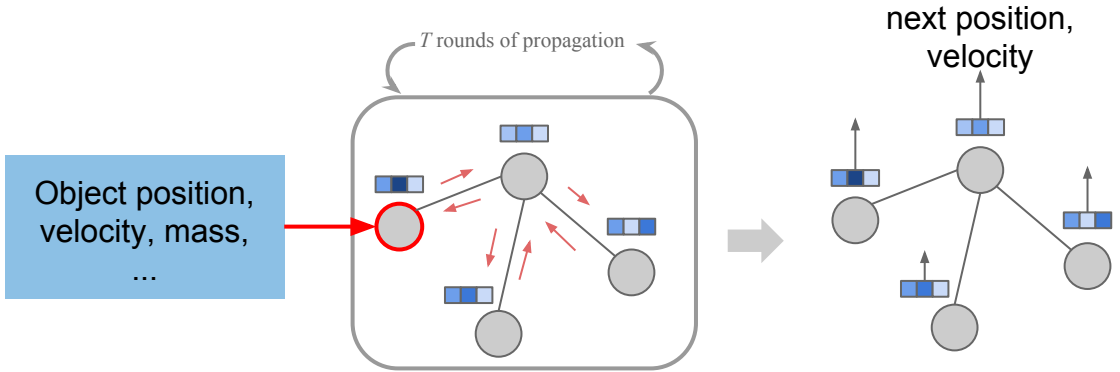
Categorization examples

Modeling Physical Systems

Data: objects and their movement in simulated environments

Task: predict object trajectories in the future

Graph Structure: objects and pairwise interactions between objects

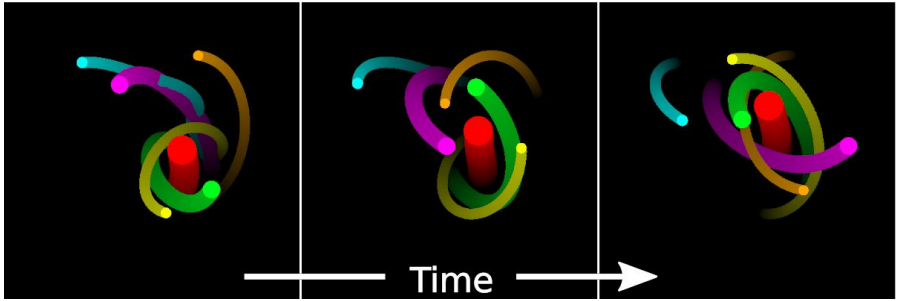


Interaction networks does very well on this.

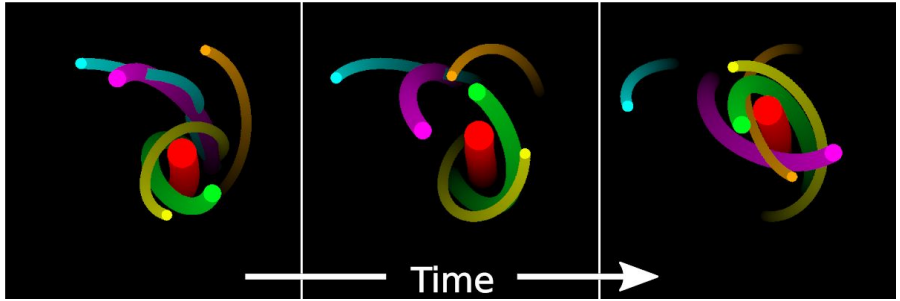
(Interaction Networks for Learning about Objects, Relations and Physics; Battaglia et al. 2016; https://arxiv.org/abs/1612.00222)

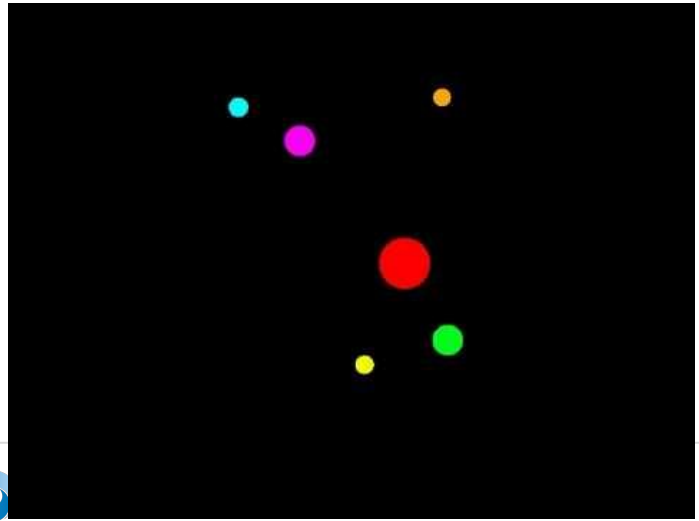
Physical domains: n-body (6 objects)

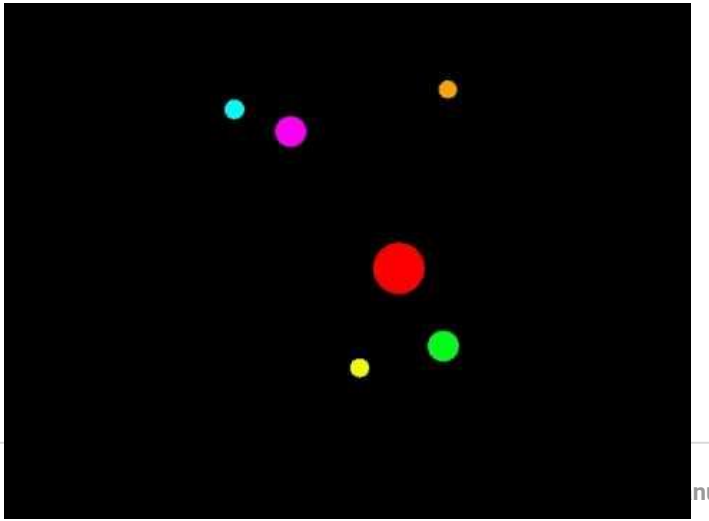
True



Model

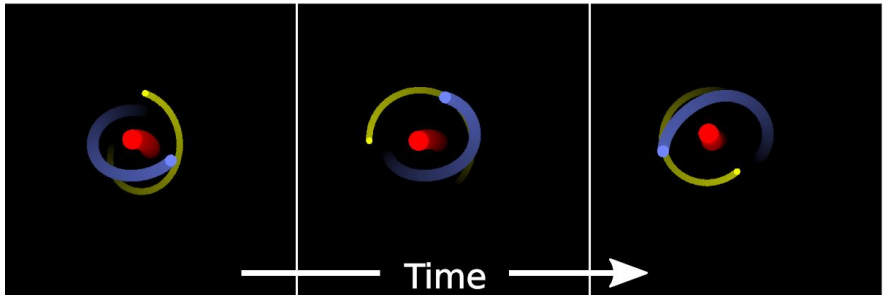




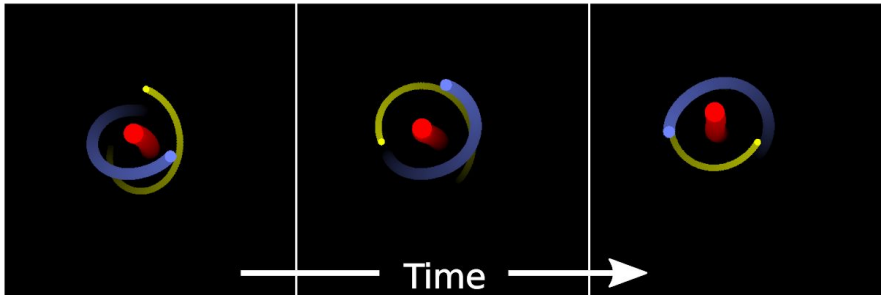


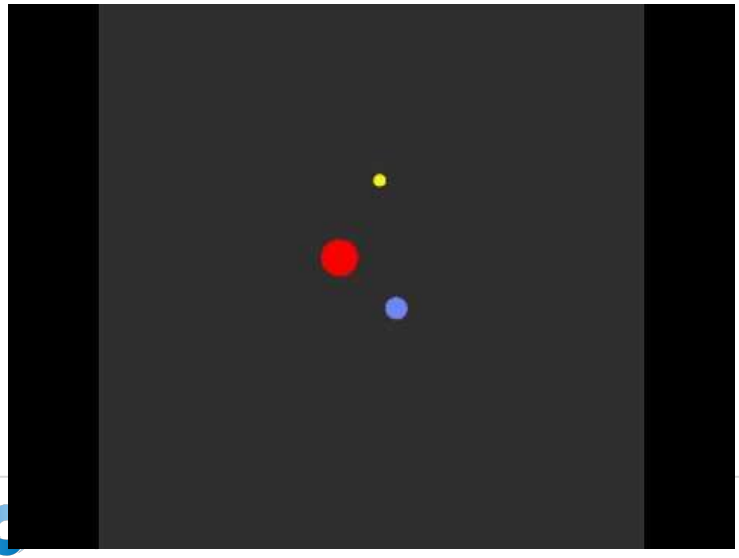
Physical domains: n-body (3 objects)

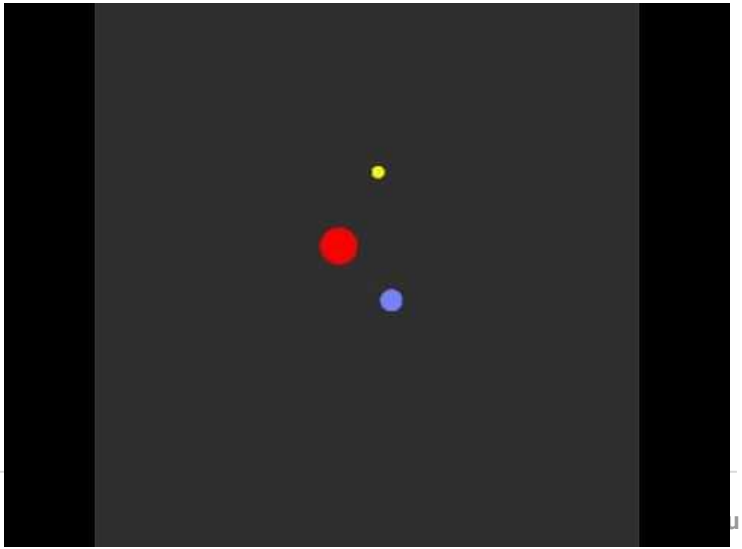
True



Model

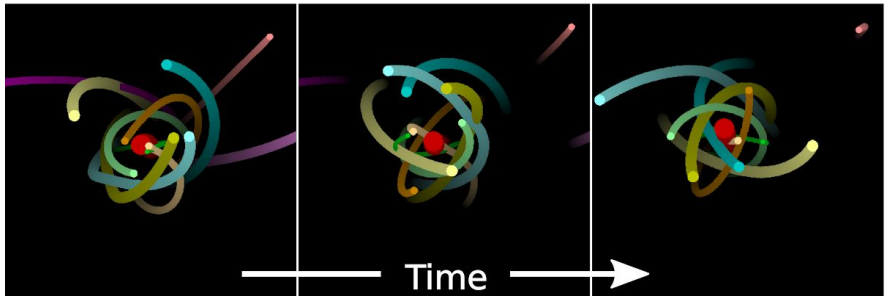




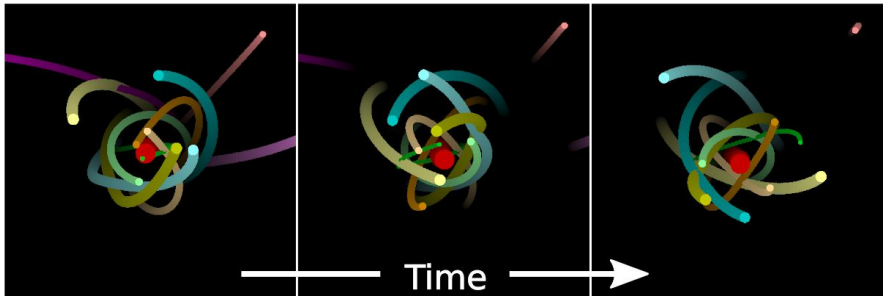


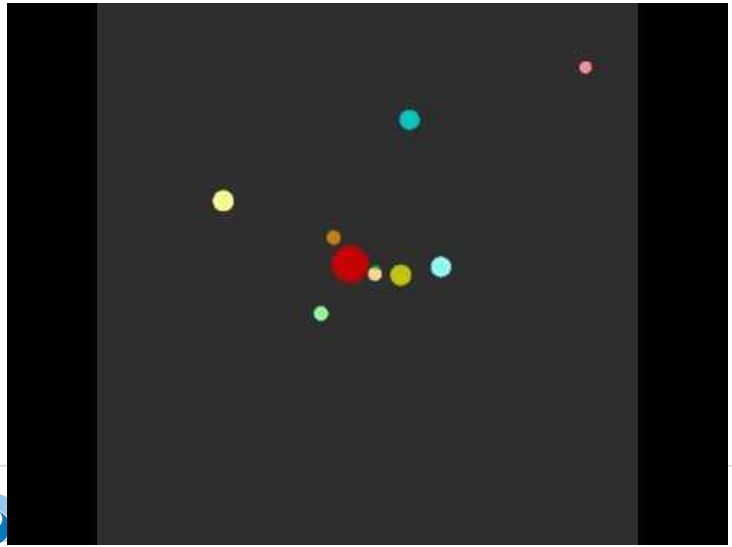
Physical domains: n-body (12 objects)

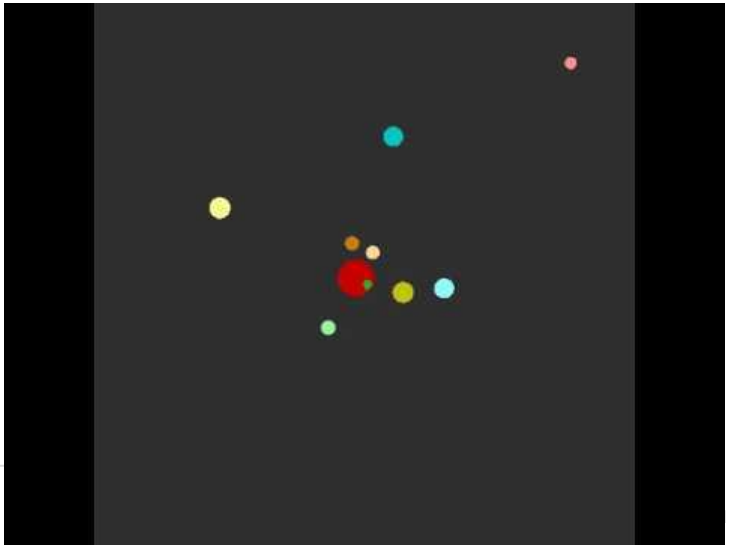
True



Model

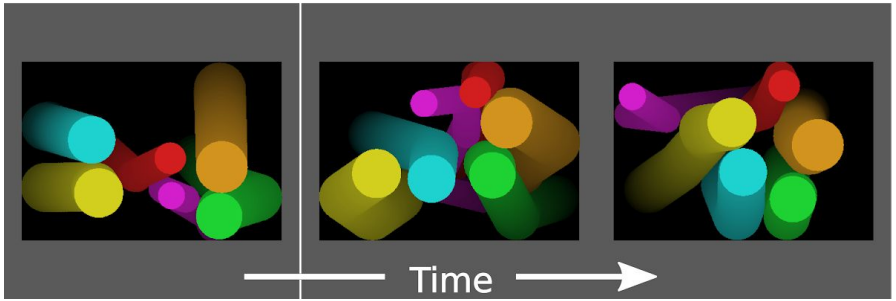




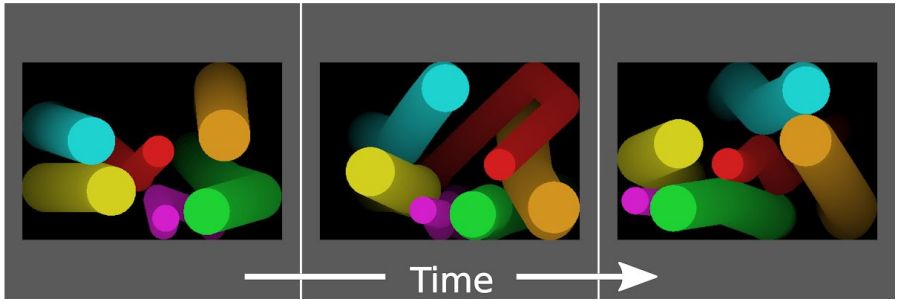


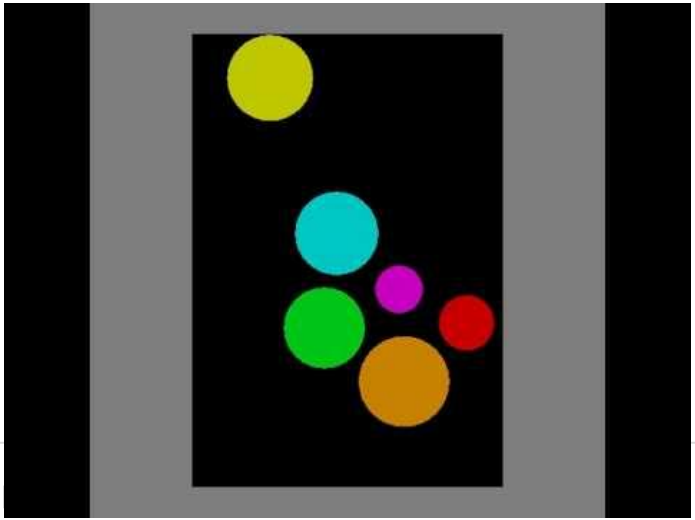
Physical domains: bouncing balls (6 balls)

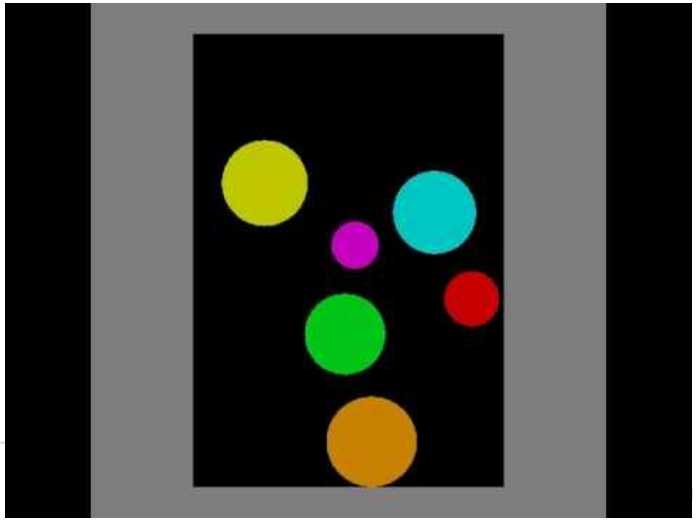
True



Model

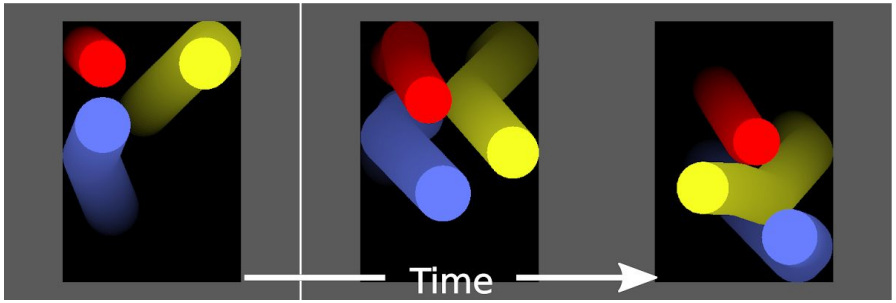




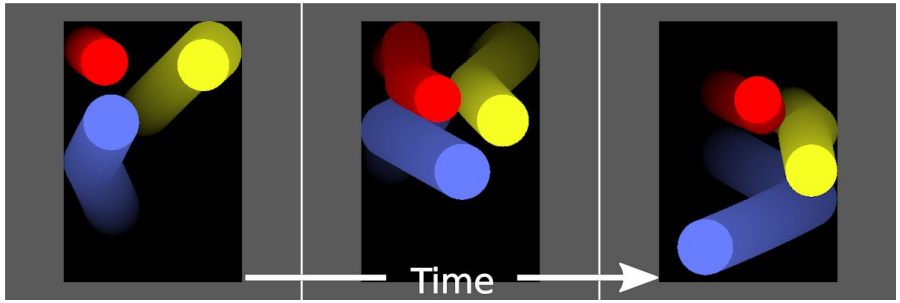


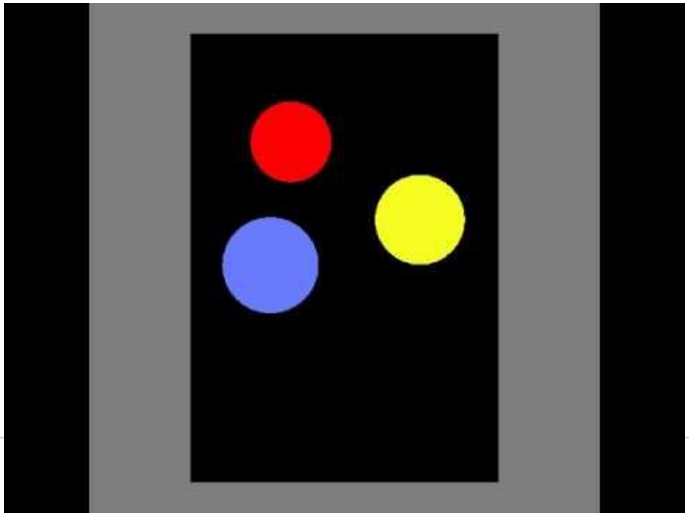
Physical domains: bouncing balls (3 balls)

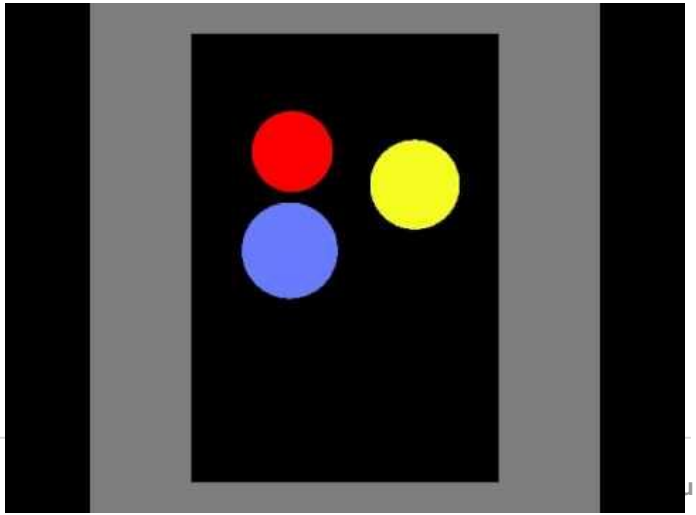
True



Model

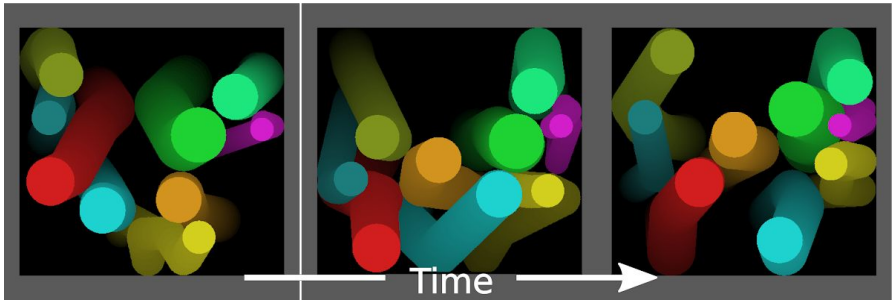




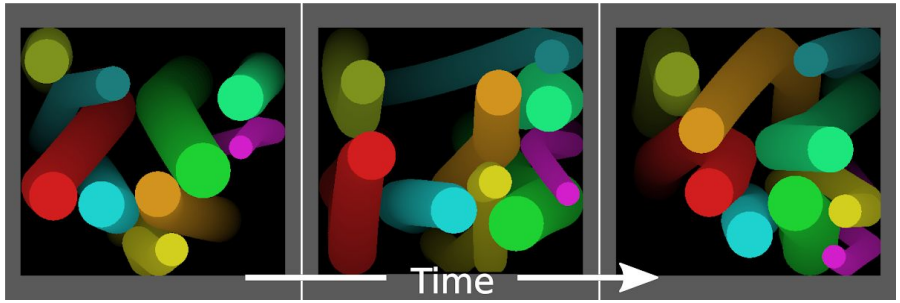


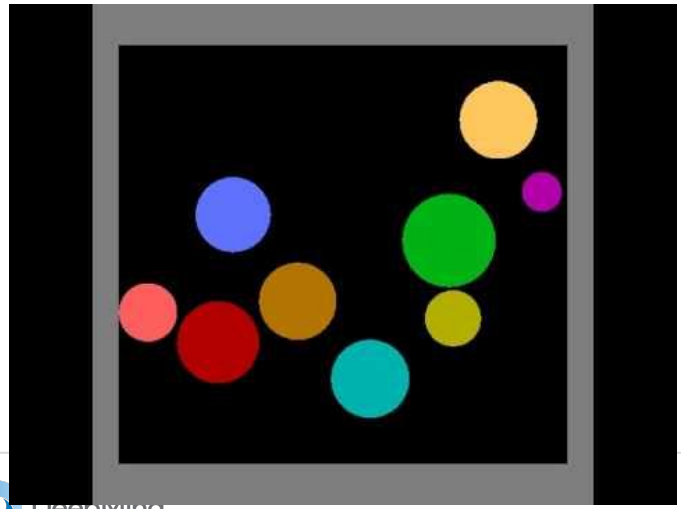
Physical domains: bouncing balls (9 balls)

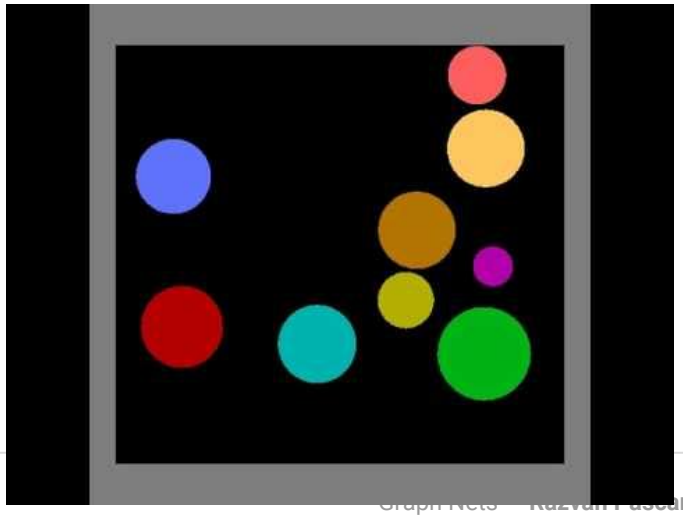
True



Model

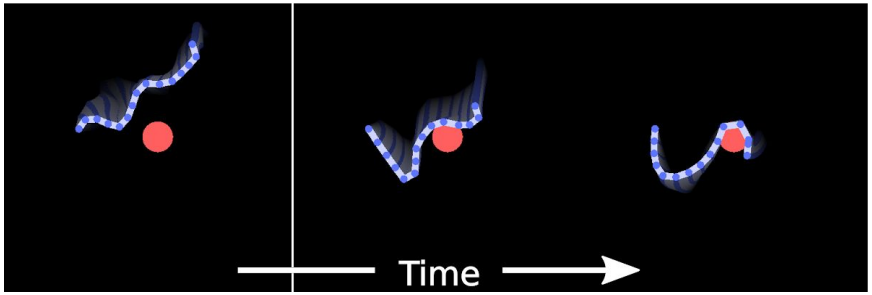




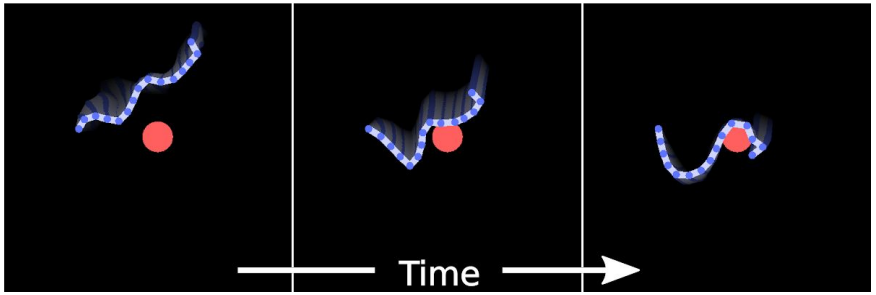


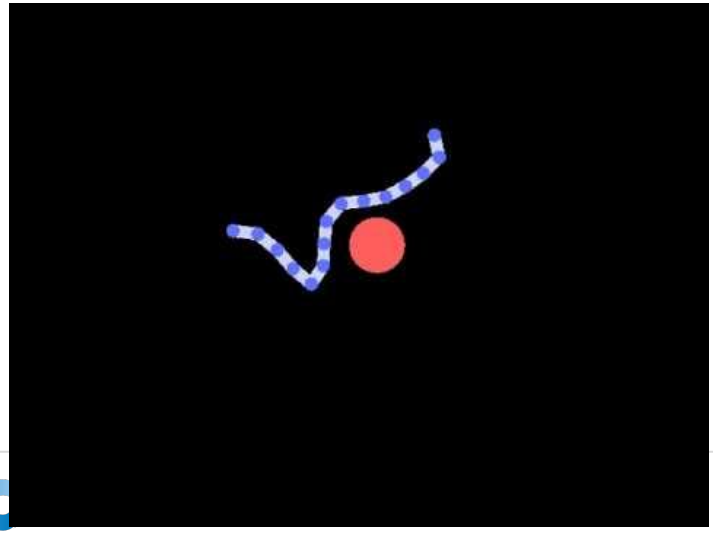
Physical domains: strings (15 masses, 1 pinned)

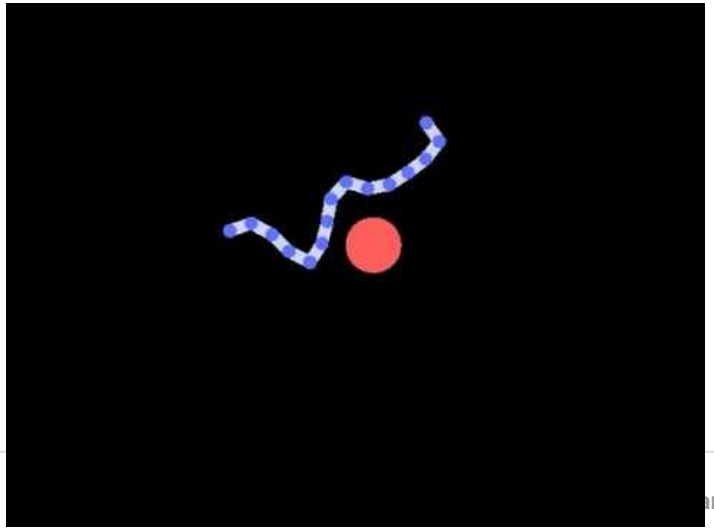
True



Model

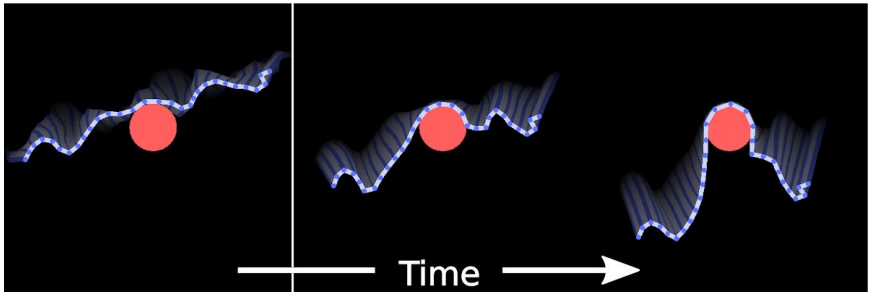




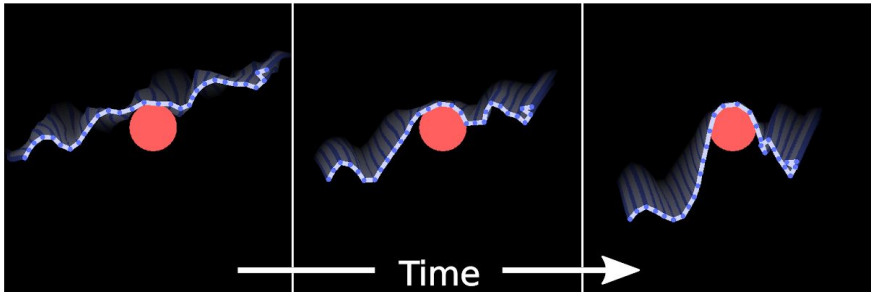


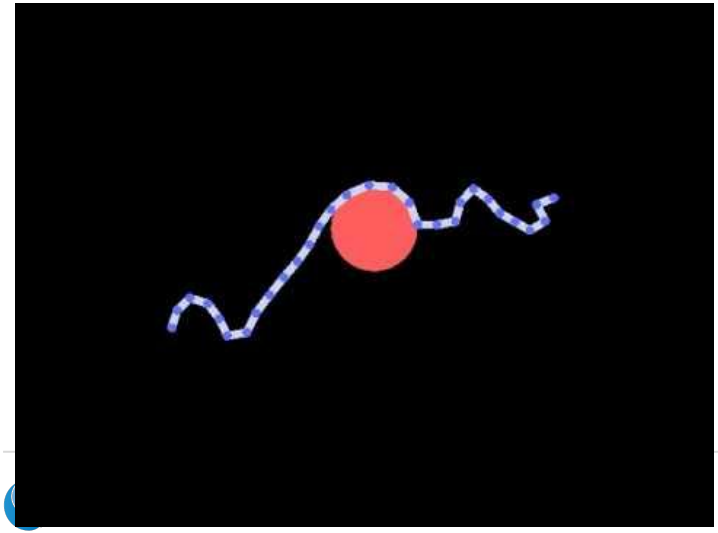
Physical domains: strings (30 masses, 0 pinned)

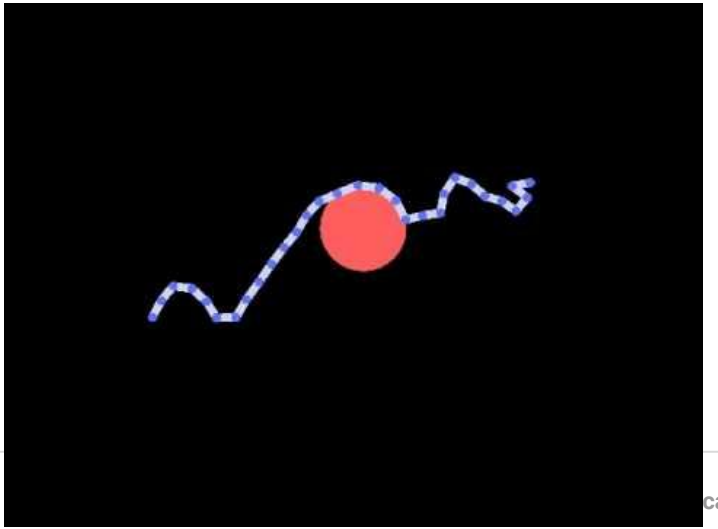
True



Model

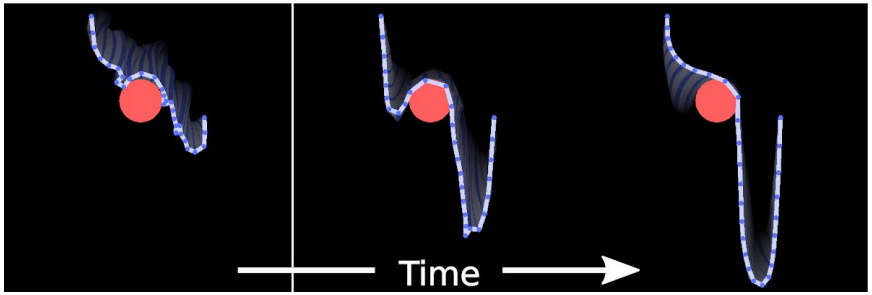




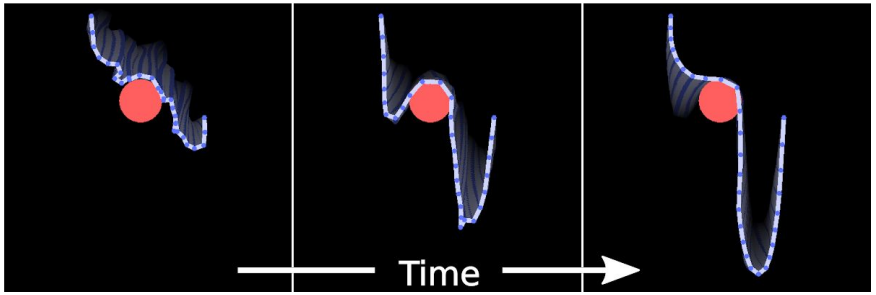


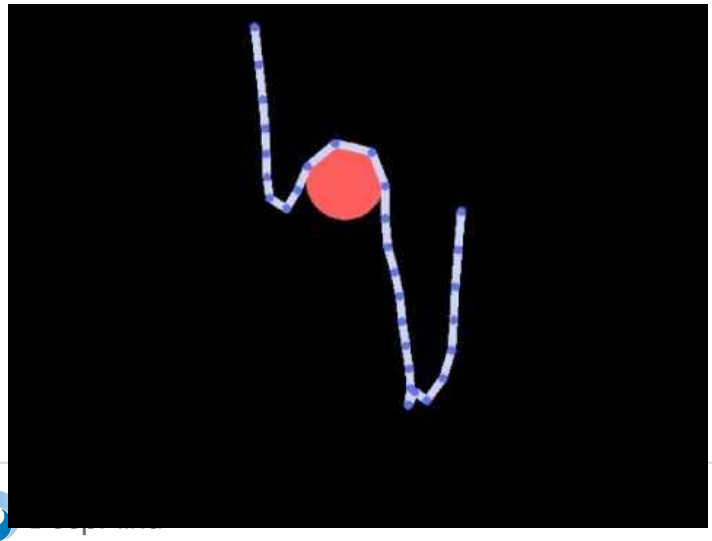
Physical domains: strings (30 masses, 2 pinned)

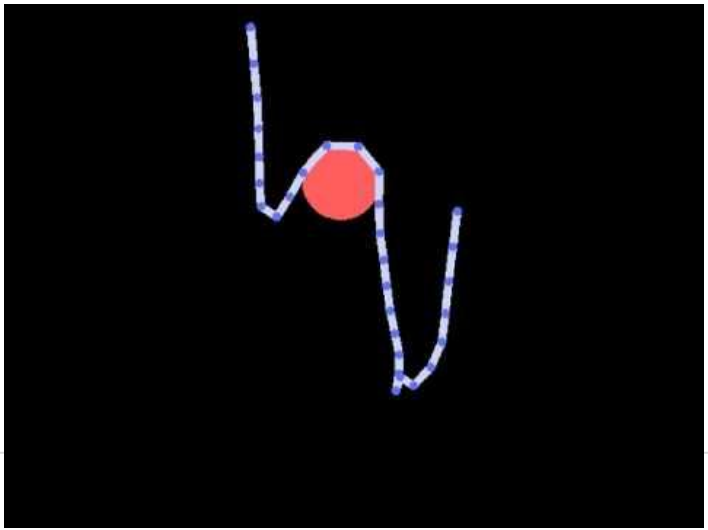
True



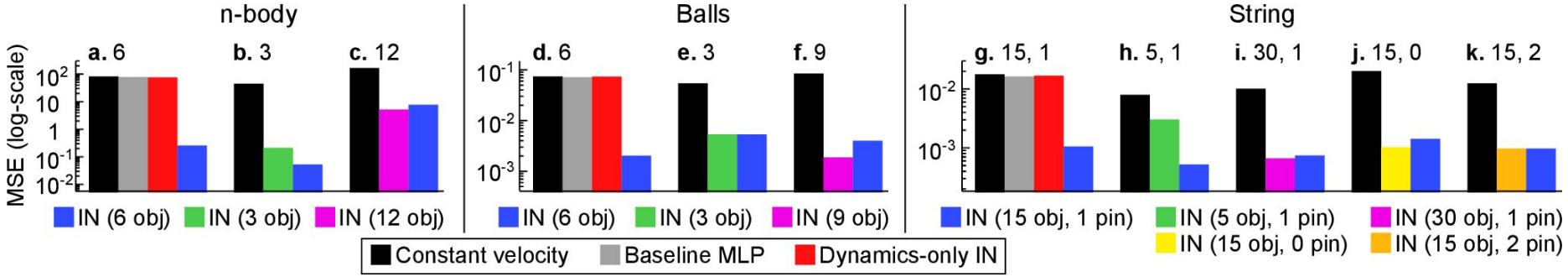
Model







Results

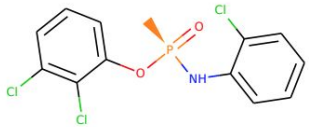

n-body
Balls
String
MSE (log-scale)
a. 6
b. 3
c. 12
d. 6
e. 3
f. 9
g. 15, 1
h. 5, 1
i. 30, 1
j. 15, 0
k. 15, 2
IN (6 obj)
IN (3 obj)
IN (12 obj)
IN (9 obj)
IN (15 obj, 1 pin)
IN (5 obj, 1 pin)
IN (30 obj, 1 pin)
IN (15 obj, 0 pin)
IN (15 obj, 2 pin)
Constant velocity
Baseline MLP
Dynamics-only IN

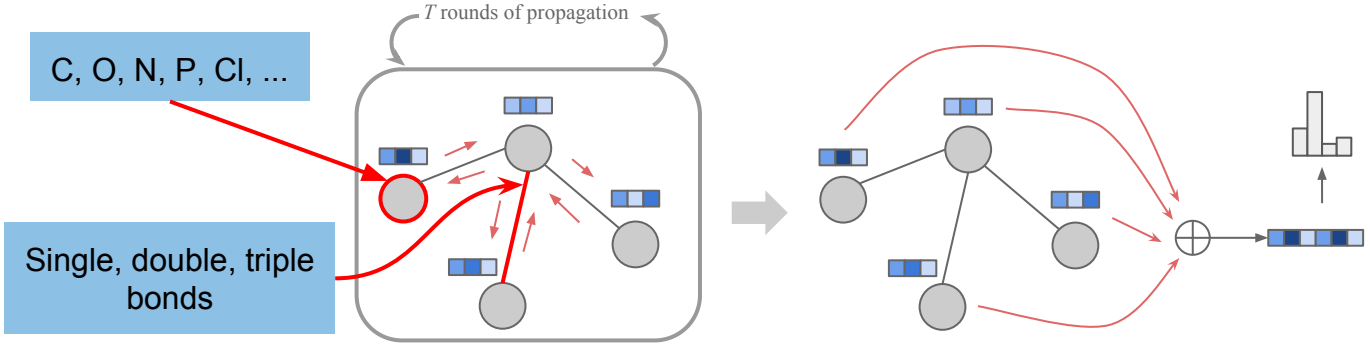
Chemical Property Prediction

Data: molecules represented as graphs

Task: predict quantum chemical properties (graph-level regression)

Graph Structure: molecule atoms and bonds





energy, heat capacity, polarizability, etc.

This can help speed up the screening of possible new drug candidates.

(Neural Message Passing for Quantum Chemestry, Gilmer et al. 2017; https://arxiv.org/abs/1704.01212)

Chemical Property Prediction

Table 3. Models Trained Without Spatial Information

Model	Average Error Ratio
GG-NN	3.47
GG-NN + Virtual Edge	2.90
GG-NN + Master Node	2.62
GG-NN + set2set	**2.57**

Table 4. Towers vs Vanilla GG-NN (no explicit hydrogen)

Model	Average Error Ratio
GG-NN + joint training	1.92
towers8 + joint training	**1.75**
GG-NN + individual training	1.53
towers8 + individual training	**1.37**

Table 2. Comparison of Previous Approaches (left) with MPNN baselines (middle) and our methods (right)

Target	BAML	BOB	CM	ECFP4	HDAD	GC	GG-NN	DTNN	enn-s2s	enn-s2s-ens5
mu	4.34	4.23	4.49	4.82	3.34	0.70	1.22	-	**0.30**	0.20
alpha	3.01	2.98	4.33	34.54	1.75	2.27	1.55	-	**0.92**	0.68
HOMO	2.20	2.20	3.09	2.89	1.54	1.18	1.17	-	**0.99**	0.74
LUMO	2.76	2.74	4.26	3.10	1.96	1.10	1.08	-	**0.87**	0.65
gap	3.28	3.41	5.32	3.86	2.49	1.78	1.70	-	**1.60**	1.23
R2	3.25	0.80	2.83	90.68	1.35	4.73	3.99	-	**0.15**	0.14
ZPVE	3.31	3.40	4.80	241.58	1.91	9.75	2.52	-	**1.27**	1.10
U0	1.21	1.43	2.98	85.01	0.58	3.02	0.83	-	**0.45**	0.33
U	1.22	1.44	2.99	85.59	0.59	3.16	0.86	-	**0.45**	0.34
H	1.22	1.44	2.99	86.21	0.59	3.19	0.81	-	**0.39**	0.30
G	1.20	1.42	2.97	78.36	0.59	2.95	0.78	.84[2]	**0.44**	0.34
Cv	1.64	1.83	2.36	30.29	0.88	1.45	1.19	-	**0.80**	0.62
Omega	0.27	0.35	1.32	1.47	0.34	0.32	0.53	-	**0.19**	0.15
Average	2.17	2.08	3.37	53.97	1.35	2.59	1.36	-	**0.68**	0.52

Generating Molecules

Data: sets of drug-like molecules

Task: learn a model capable of generating similar but novel molecules

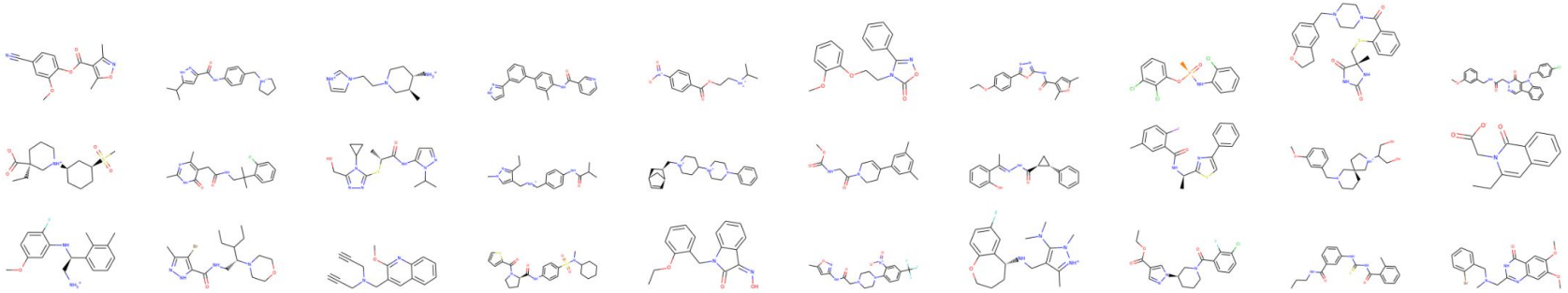
Graph Structure: molecule atoms and bonds

Our approach: sequentially generate one node and all the associated edges. Each step is a prediction problem on molecule graph.

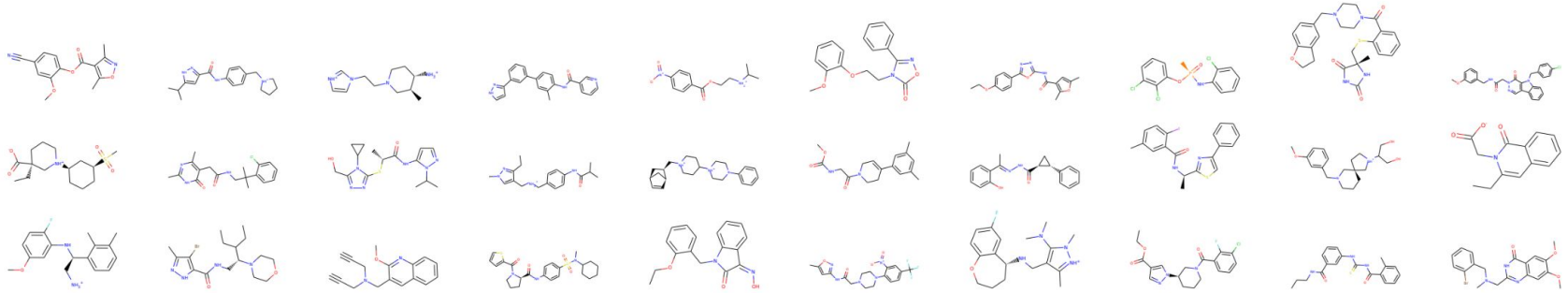
(Learning deep generative models of Graphs,

Li et al. 2018 https://arxiv.org/abs/1803.03324)

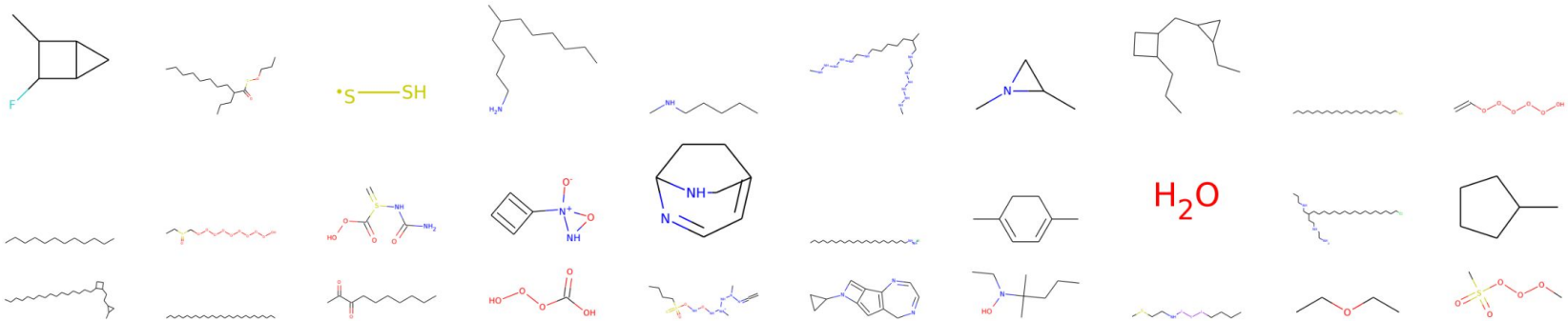
Dataset



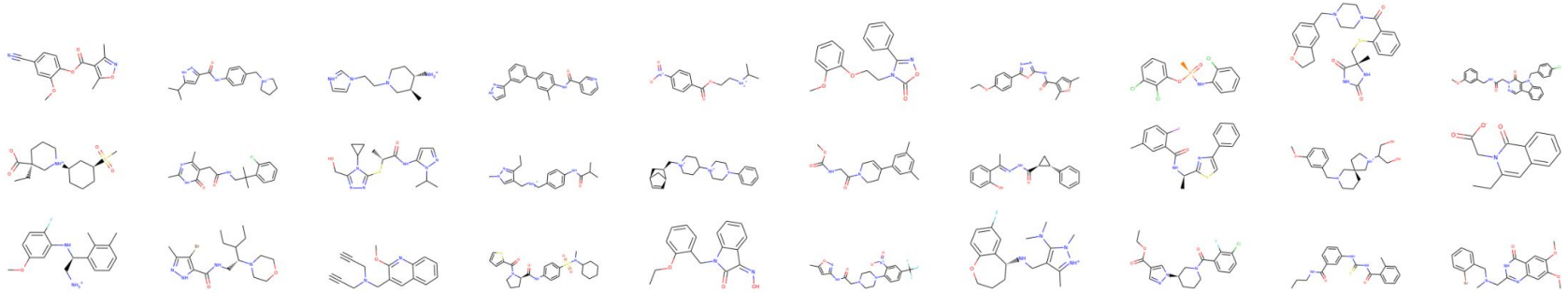
Dataset



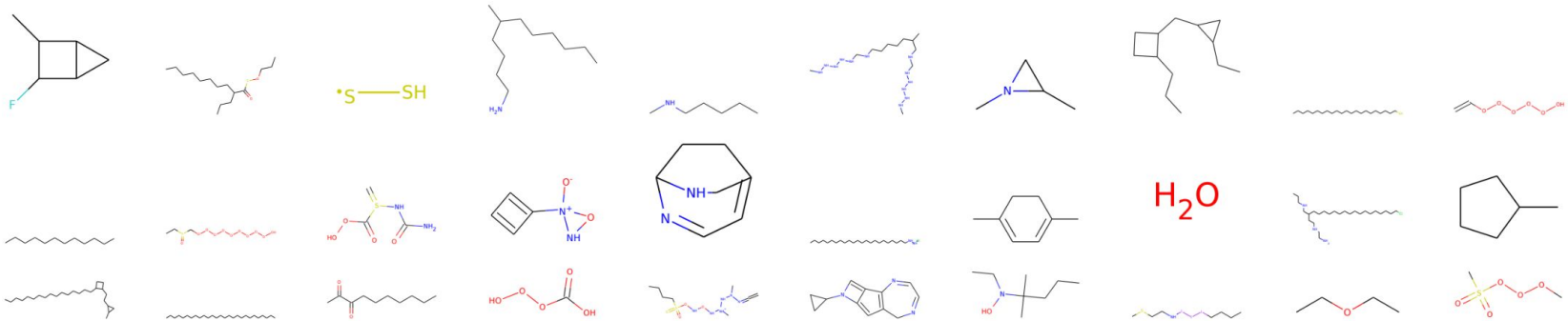
Grammar VAE (baseline)



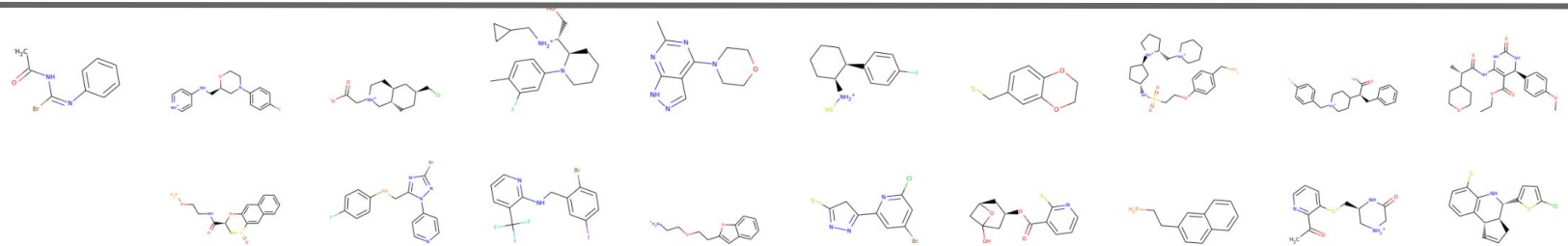
Dataset



Grammar VAE (baseline)

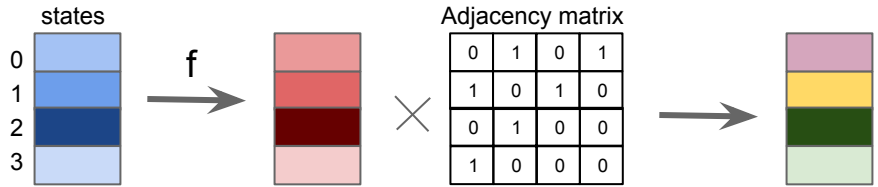


Our model



Graph Convolutions

- Geometric deep learning (not covered) -- Joan Bruna, Arthur Szlam
- Thomas Kipf/Max Welling https://arxiv.org/pdf/1609.02907.pdf



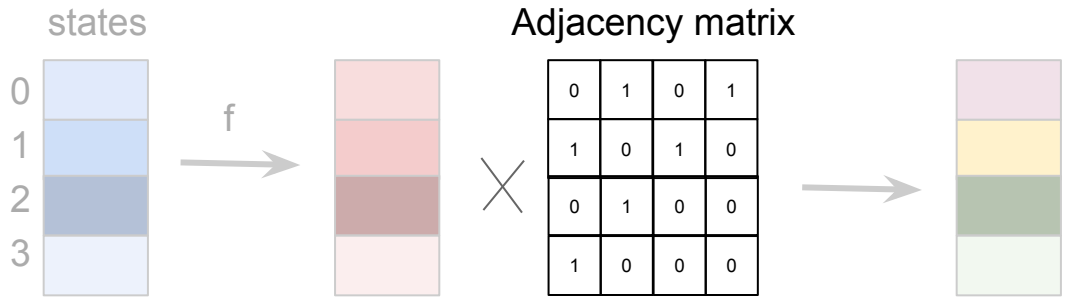
Graph Convolutions

Table 1: Dataset statistics, as reported in Yang et al. (2016).

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

Graph Convolutions

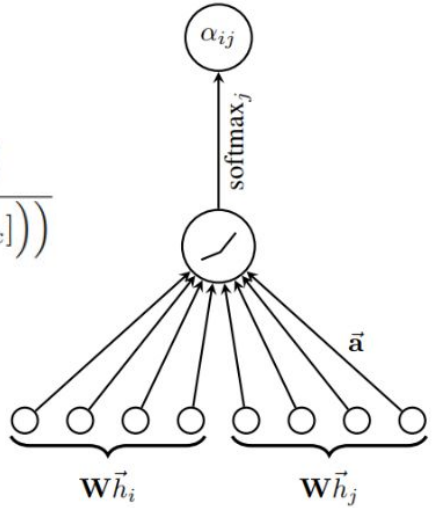


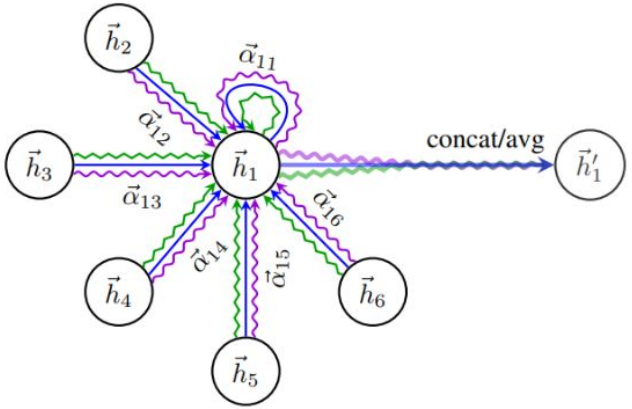
Graph Attention Networks (GAT)

https://arxiv.org/abs/1710.10903

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}\vec{h}_j\right)$$





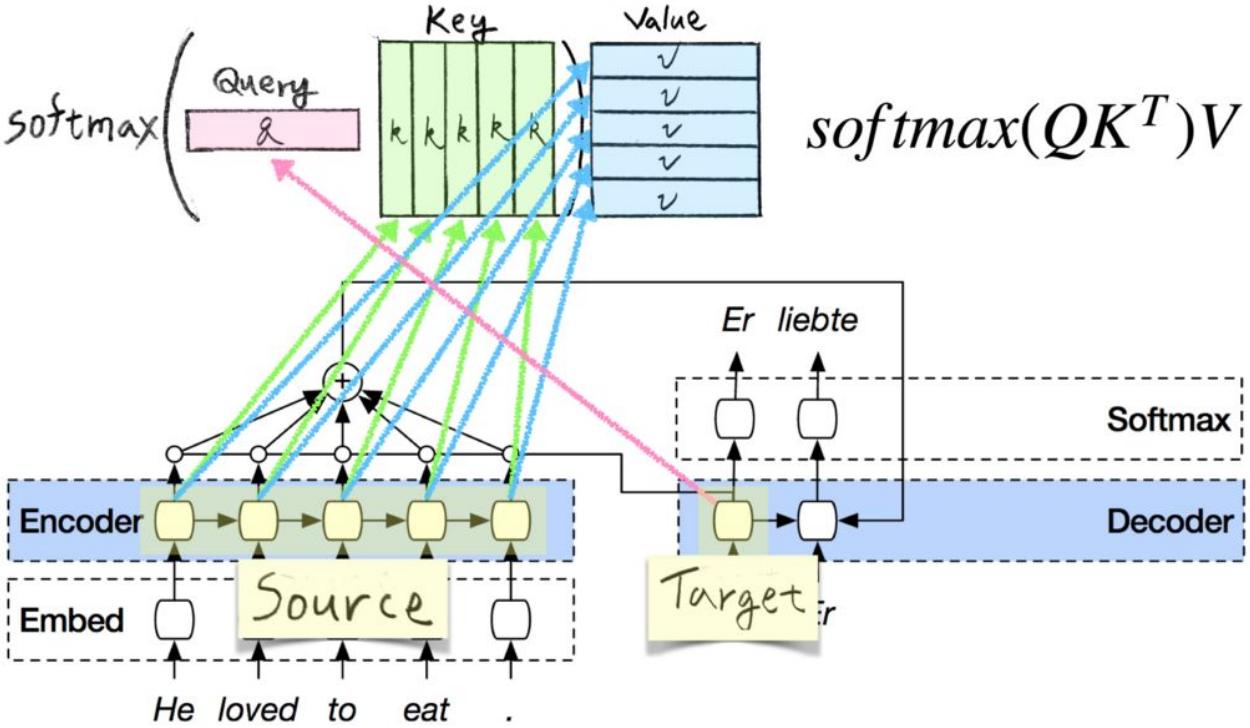
Graph Attention Networks (GAT)

https://arxiv.org/abs/1710.10903

	Transductive		
Method	**Cora**	**Citeseer**	**Pubmed**
MLP	55.1%	46.5%	71.4%
ManiReg (Belkin et al., 2006)	59.5%	60.1%	70.7%
SemiEmb (Weston et al., 2012)	59.0%	59.6%	71.7%
LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
ICA (Lu & Getoor, 2003)	75.1%	69.1%	73.9%
Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%
Chebyshev (Defferrard et al., 2016)	81.2%	69.8%	74.4%
GCN (Kipf & Welling, 2017)	81.5%	70.3%	**79.0%**
MoNet (Monti et al., 2016)	81.7 ± 0.5%	—	78.8 ± 0.3%
GCN-64*	81.4 ± 0.5%	70.9 ± 0.5%	**79.0** ± 0.3%
GAT (ours)	**83.0** ± 0.7%	**72.5** ± 0.7%	**79.0** ± 0.3%

Transformer (Attention is all you need)

https://arxiv.org/abs/1706.03762

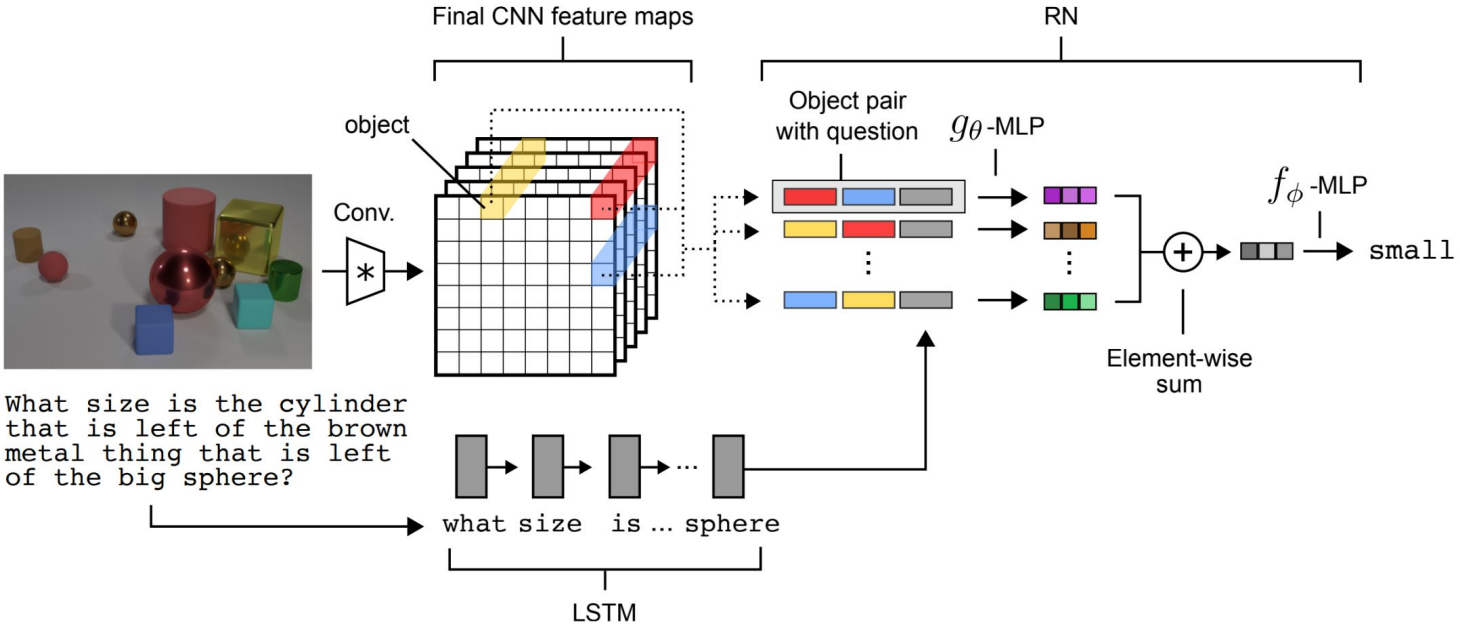


Credits:http://deeplearning.hatenablog.com/entry/transformer

Categorization examples

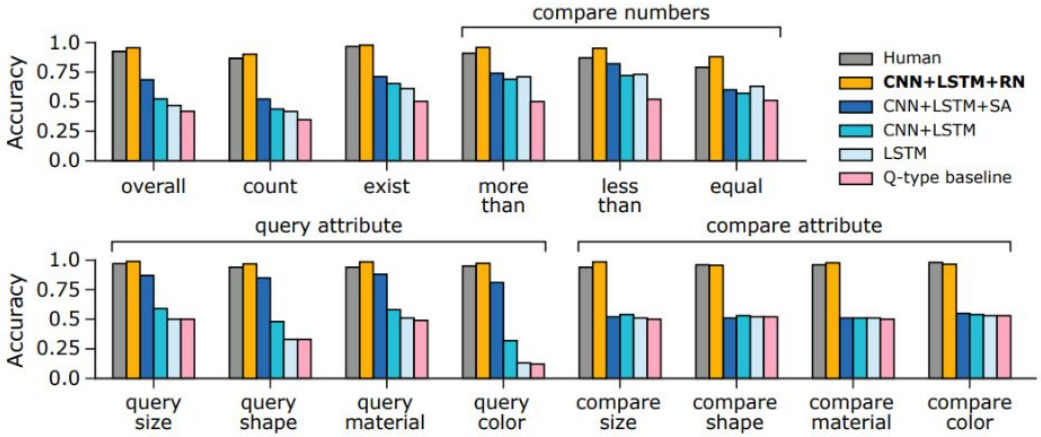
Image Credit - Leibe et Al.

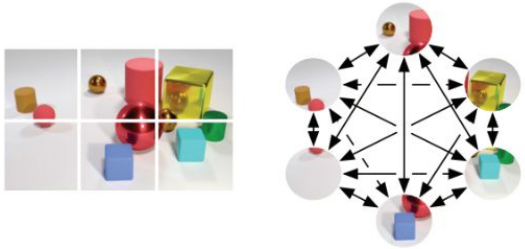
Relation Network



Santoro et al. 2017

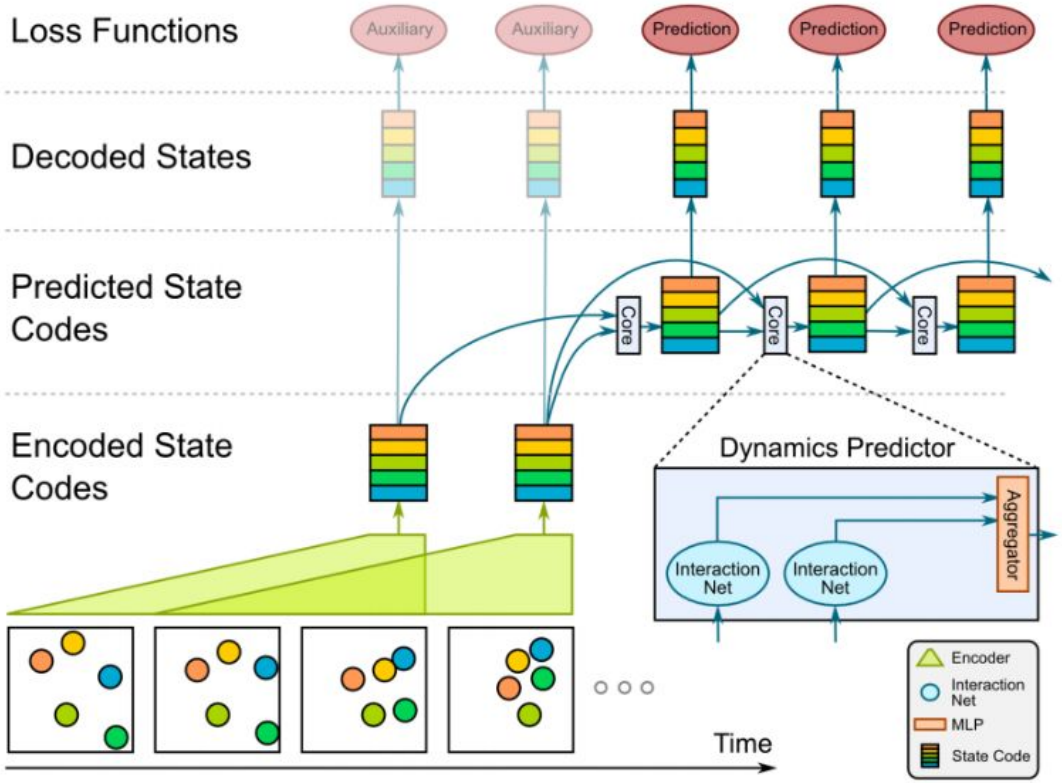
Relation Network





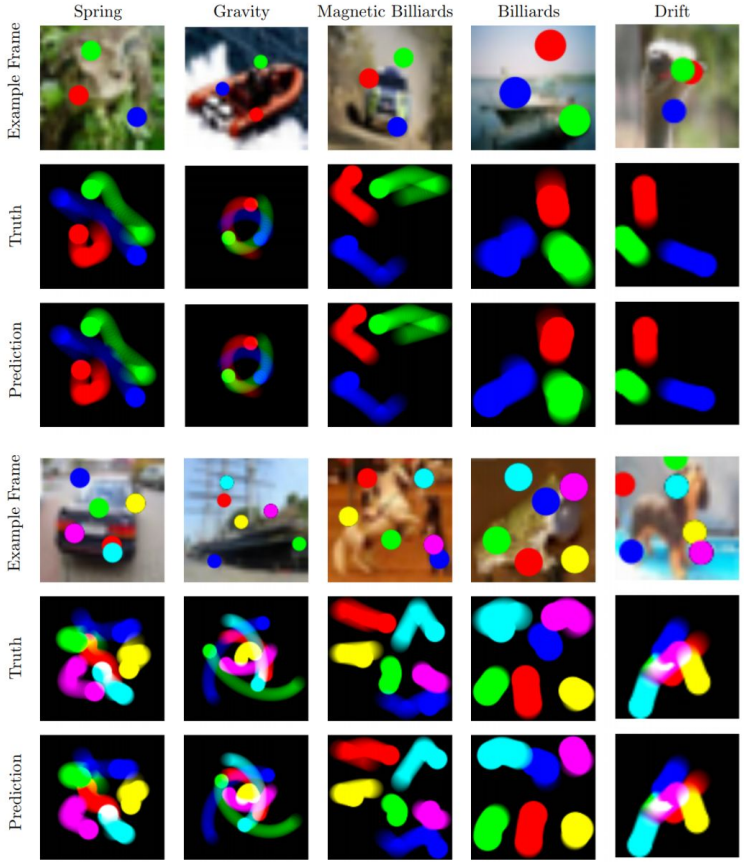
A simple neural network module for relational reasoning, Santoro et al. 2017; https://arxiv.org/pdf/1706.01427.pdf

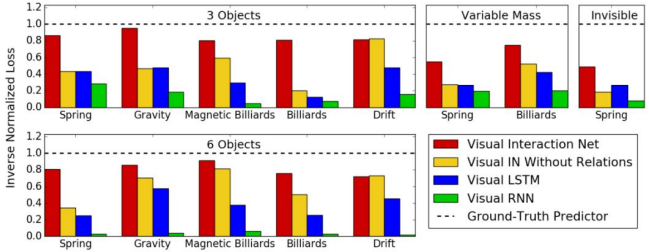
Visual Interaction Network



Visual Interaction Network, Watters et al. 2017
https://arxiv.org/abs/1706.01433

Visual Interaction Network

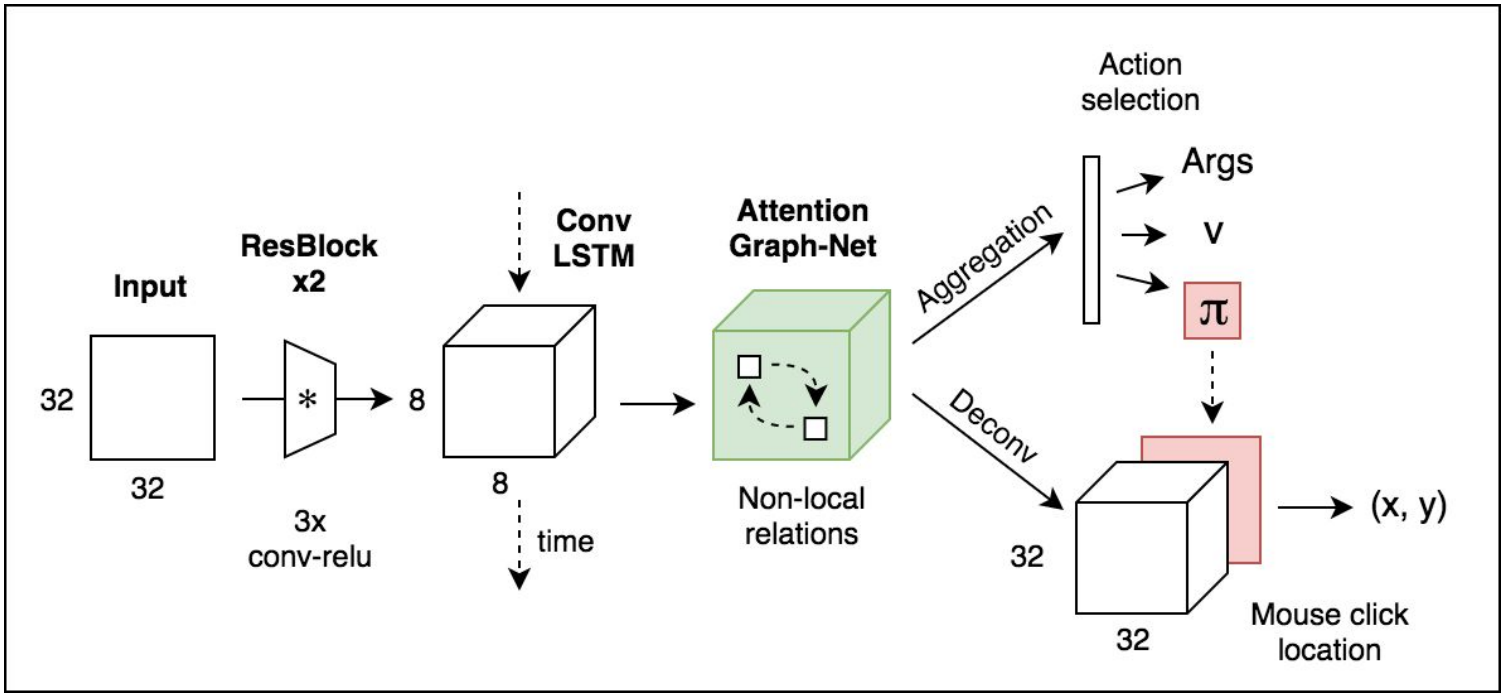




Relational reasoning in RL

Relational Deep Reinforcement Learning, Zambaldi et al. 2018; https://arxiv.org/pdf/1806.01830.pdf

Relational reasoning in RL



Relational Deep Reinforcement Learning, Zambaldi et al. 2018; https://arxiv.org/pdf/1806.01830.pdf

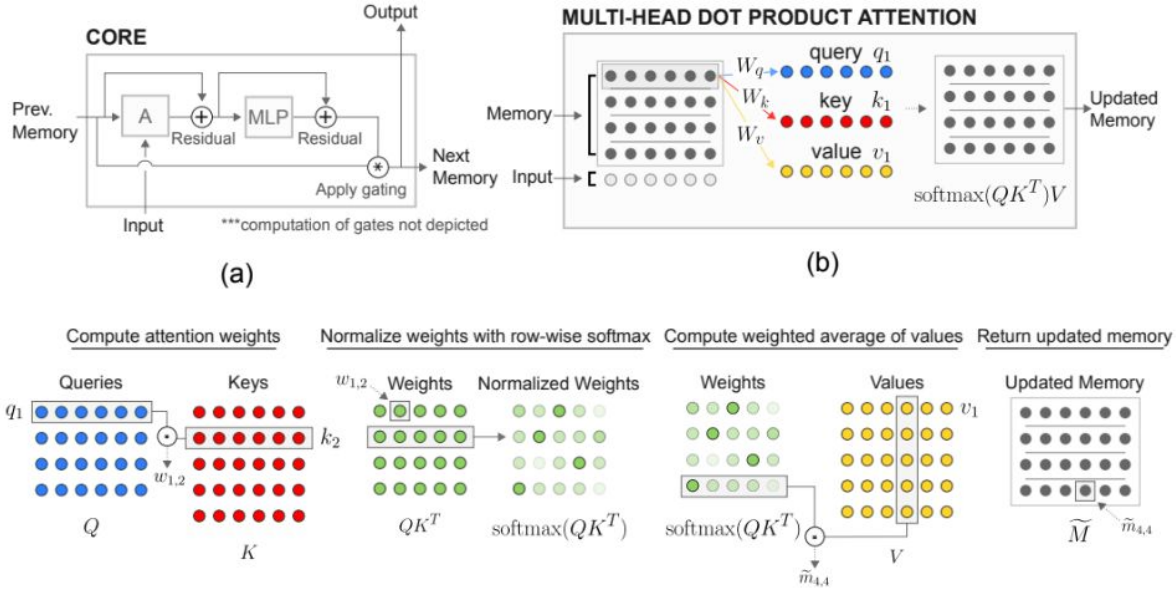
Relational reasoning in RL

Relational reasoning in RL

	Mini-game						
Agent	①	②	③	④	⑤	⑥	⑦
DeepMind Human Player [15]	26	133	46	41	729	6880	138
StarCraft Grandmaster [15]	28	177	61	215	727	7566	133
Random Policy [15]	1	17	4	1	23	12	< 1
FullyConv LSTM [15]	26	104	44	98	96	3351	6
PBT-A3C [31]	–	101	50	132	125	3345	0
Relational agent	**27**	**196** ↑	**62** ↑	**303** ↑	**736** ↑	4906	**123**
Control agent	**27**	187 ↑	61	295 ↑	602	**5055**	120

Relational Deep Reinforcement Learning, Zambaldi et al. 2018; https://arxiv.org/pdf/1806.01830.pdf

Relational recurrent neural networks



Relational recurrent neural networks, Santoro et al. 2018; https://arxiv.org/pdf/1806.01822.pdf

Relational recurrent neural networks

Table 1: Test per character Accuracy on Program Evaluation and Memorization tasks.

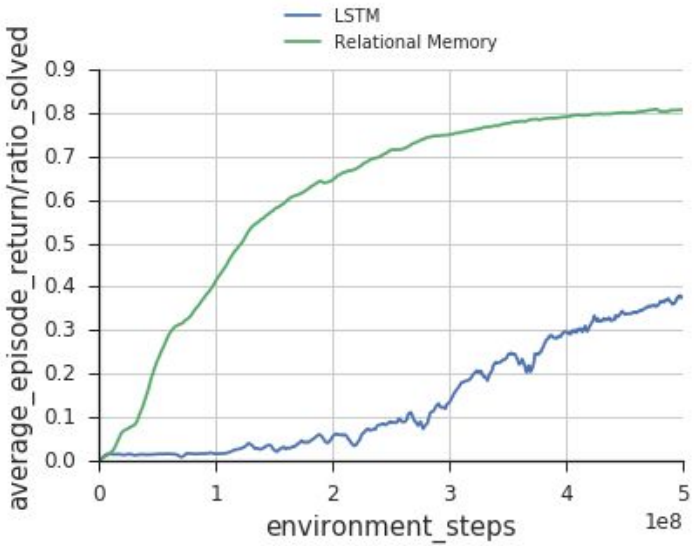
Model	Add	Control	Program	Copy	Reverse	Double
LSTM [3, 34]	99.8	97.4	66.1	99.8	99.7	99.7
EntNet [35]	98.4	98.0	73.4	91.8	**100.0**	62.3
DNC [5]	99.4	83.8	69.5	**100.0**	**100.0**	**100.0**
Relational Memory Core	**99.9**	**99.6**	**79.0**	**100.0**	**100.0**	99.8

Table 2: Validation and test perplexities on WikiText-103, Project Gutenberg, and GigaWord v5.

	WikiText-103		Gutenberg		GigaWord
	Valid.	Test	Valid	Test	Test
LSTM [37]	-	48.7	-	-	-
Temporal CNN [38]	-	45.2	-	-	-
Gated CNN [39]	-	37.2	-	-	-
LSTM [29]	34.1	34.3	41.8	45.5	43.7
Quasi-RNN [40]	32	33	-	-	-
Relational Memory Core	**30.8**	**31.6**	**39.2**	**42.0**	**38.3**

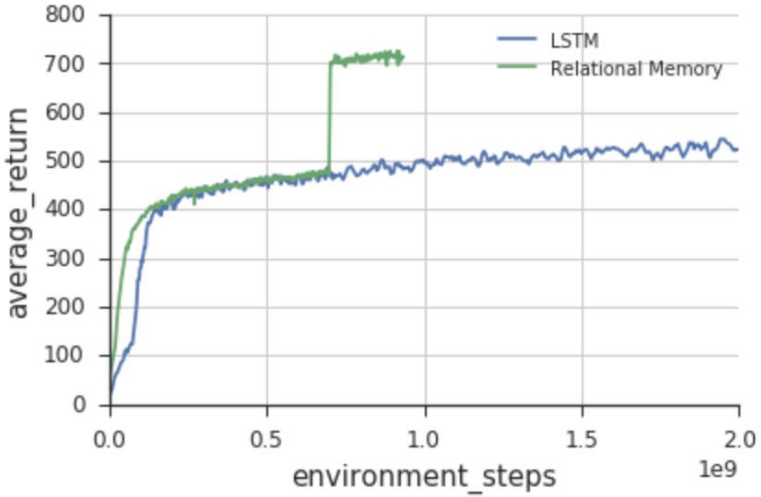
Relational recurrent neural networks, Santoro et al. 2018; https://arxiv.org/pdf/1806.01822.pdf

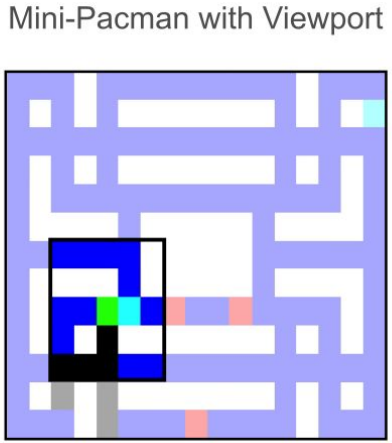
Boxworld



With viewport

Mini Pacman with viewport



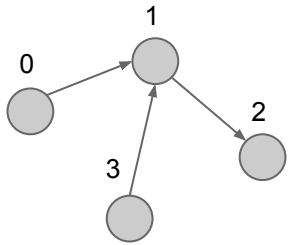


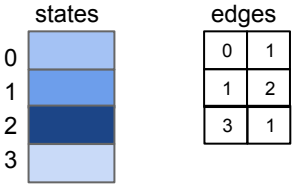
Implementing graph nets, in tensorflow

graph nets in tensorflow

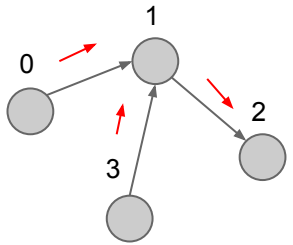
```
# node states and edges
states = tf.placeholder(tf.float32)
edges = tf.placeholder(tf.int32)

# use sonnet or your favorite toolkit to build
# feedforward networks
effect_net = snt.Sequential(...)
node_net = snt.Sequential(...)
```

$$G = (V, E) \text{ state } \mathbf{x}_v \ \forall v \in V$$



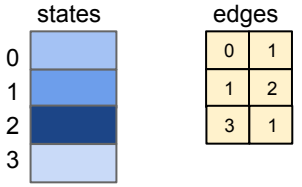


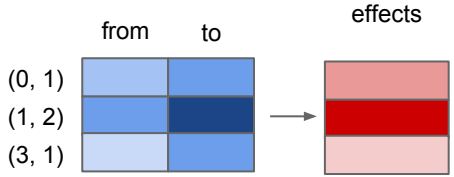
Graph Propagation Core




```
# node states and edges
states = tf.placeholder(tf.float32)
edges = tf.placeholder(tf.int32)

# use sonnet or your favorite toolkit to build
# feedforward networks
effect_net = snt.Sequential(...)
node_net = snt.Sequential(...)

# compute effects / messages along each edge, edge
# features can be fed into the model too if available
states_from = tf.gather(states, edges[:, 0])
states_to = tf.gather(states, edges[:, 1])
concat_states = tf.concat([states_from, states_to],
    axis=-1)
effects = effect_net(concat_states)
```

$$G = (V, E) \text{ state } \mathbf{x}_v \ \forall v \in V$$
$$\forall (u, v) \in E, \ \mathbf{e}_{u \to v} = f_e(\mathbf{x}_u, \mathbf{x}_v)$$





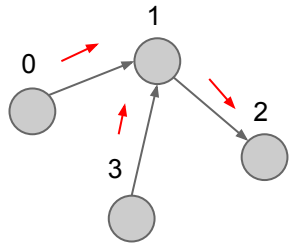
Graph Propagation Core

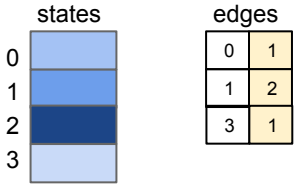
```
...
# compute effects / messages along each edge, edge
# features can be fed into the model too if available
states_from = tf.gather(states, edges[:, 0])
states_to = tf.gather(states, edges[:, 1])
concat_states = tf.concat([states_from, states_to],
    axis=-1)
effects = effect_net(concat_states)

# aggregate incoming effects by a sum, or average
aggregated_effects = tf.unsorted_segment_sum(effects,
    edges[:, 1], tf.shape(states)[0])
```

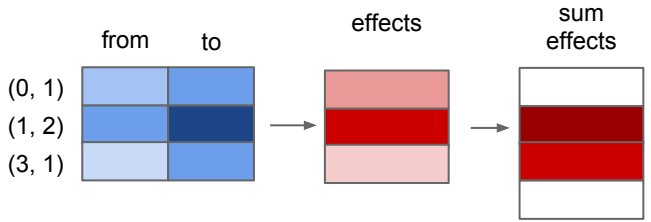
$$G = (V, E) \text{ state } \mathbf{x}_v \ \forall v \in V$$

$$\forall (u, v) \in E, \ \mathbf{e}_{u \to v} = f_e(\mathbf{x}_u, \mathbf{x}_v)$$

$$\forall v \in V, \ \mathbf{e}_v = \sum_{u:(u,v) \in E} \mathbf{e}_{u \to v}$$







Graph Propagation Core

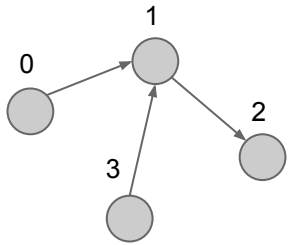
```
...
# compute effects / messages along each edge, edge
# features can be fed into the model too if available
states_from = tf.gather(states, edges[:, 0])
states_to = tf.gather(states, edges[:, 1])
concat_states = tf.concat([states_from, states_to],
    axis=-1)
effects = effect_net(concat_states)

# aggregate incoming effects by a sum, or average
aggregated_effects = tf.unsorted_segment_sum(effects,
    edges[:, 1], tf.shape(states)[0])

# update the node states
states = node_net(tf.concat([aggregated_effects,
    states], axis=-1))
```

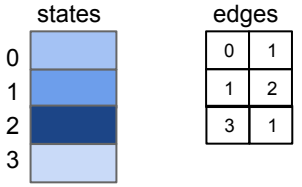
$G = (V, E)$ state $\mathbf{x}_v \ \forall v \in V$

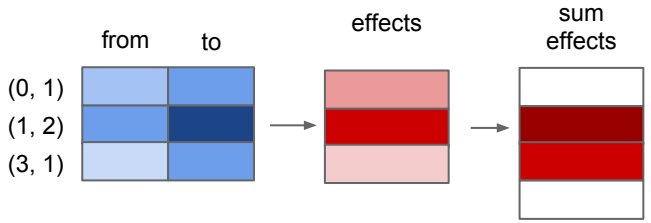
$\forall (u, v) \in E, \ \mathbf{e}_{u \to v} = f_e(\mathbf{x}_u, \mathbf{x}_v)$

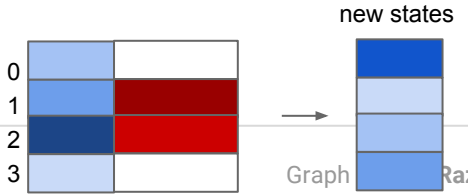
$\forall v \in V, \ \mathbf{e}_v = \sum_{u:(u,v) \in E} \mathbf{e}_{u \to v}$

$\forall v \in V, \ \mathbf{x}_v \leftarrow f_n(\mathbf{x}_v, \mathbf{e}_v)$









Output Modules

```
# graph-level output
graph_vec = tf.reduce_sum(states, axis=0)
graph_output = graph_level_network(graph_vec)
...     # feed into your favorite loss
```

Graph-level output $\mathbf{o}_G = g_G\left(\sum_{v \in V} \mathbf{x}_v\right)$

Output Modules

```
# graph-level output
graph_vec = tf.reduce_sum(states, axis=0)
graph_output = graph_level_network(graph_vec)
...     # feed into your favorite loss

# node-level output
node_outputs = node_level_net(states)
...     # feed into your favorite loss
```

Graph-level output $\mathbf{o}_G = g_G\left(\sum_{v \in V} \mathbf{x}_v\right)$

Node-level output $\mathbf{o}_v = g_n\left(\mathbf{x}_v\right)$

Output Modules

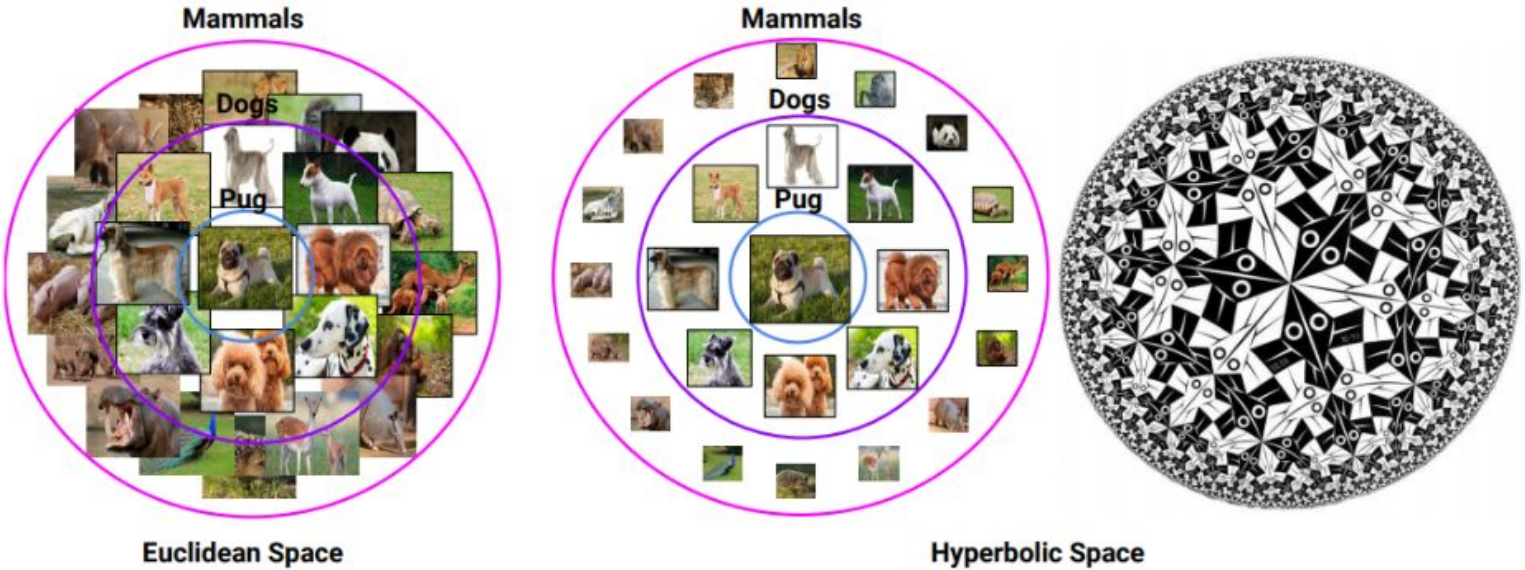
```
# graph-level output
graph_vec = tf.reduce_sum(states, axis=0)
graph_output = graph_level_network(graph_vec)
...     # feed into your favorite loss

# node-level output
node_outputs = node_level_net(states)
...     # feed into your favorite loss

# edge-level output
states_from = tf.gather(states, edges[:, 0])
states_to = tf.gather(states, edges[:, 1])
concat_states = tf.concat([states_from, states_to],
    axis=-1)
edge_outputs = edge_level_net(concat_states)
...     # feed into your favorite loss
```

Graph-level output $\mathbf{o}_G = g_G\left(\sum_{v \in V} \mathbf{x}_v\right)$

Node-level output $\mathbf{o}_v = g_n\left(\mathbf{x}_v\right)$

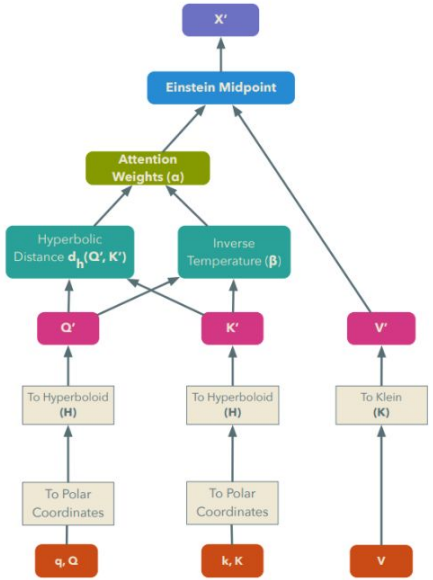
Edge-level output $\mathbf{o}_{u,v} = g_e\left(\mathbf{x}_u, \mathbf{x}_v\right)$

Hyperbolic Attention Networks



Hyperbolic Attention Networks, Gulchere et al. 2018; https://arxiv.org/abs/1805.09786

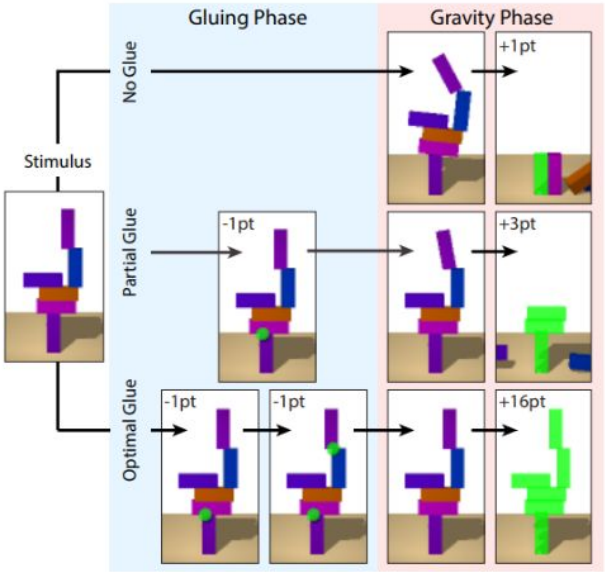
Hyperbolic Attention Networks



	WMT 2014 En-De BLEU Scores	
	Tiny	**Base**
Transformer (Vaswani et al. [41])	-	27.3
Transformer (Shaw et al. [36])	-	26.5
Transformer (Latest)	17.3	27.1
Hyperbolic Transformer (+Sigmoid)	17.3	27.4
Hyperbolic Transformer (+Softmax, +Polar)	17.5	27.0
Hyperbolic Transformer (+Sigmoid, +Polar)	**18.0**	**27.5**

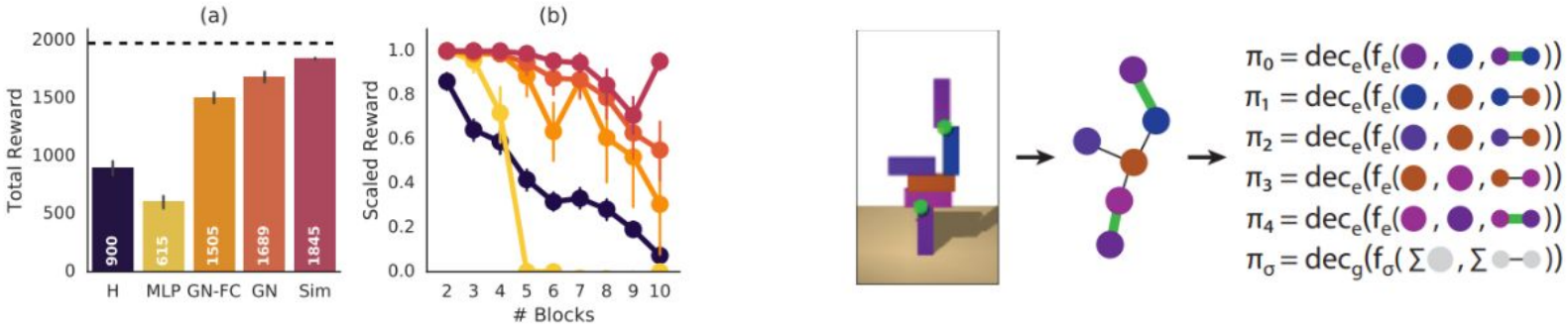
Hyperbolic Attention Networks, Gulchere et al. 2018; https://arxiv.org/abs/1805.09786

Relational inductive bias for physical construction in humans and machines



Relational inductive bias for physical construction in humans and machines, Hamrick et al. 2018; https://arxiv.org/pdf/1806.01203.pdf

Relational inductive bias for physical construction in humans and machines



Relational inductive bias for physical construction in humans and machines, Hamrick et al. 2018;
https://arxiv.org/pdf/1806.01203.pdf

Shortcomings of Graph Nets

- While they provide additional structure, and representational power they can't represent notions such as recursion, control flow, conditional computation etc.

Drawing a few conclusions

- Understanding how to add structure is important
- Neural networks are resilient to noise, as long as training and testing conditions are the same
- Structure inference is by far the hardest problem; though many tasks do not require the inference step
- Graph Nets are a family of models, and understanding and relating these architectures to each other is important

Credit where credit is due

Special thanks to:

- Peter Battaglia
- Jess Hamrick
- Yujia Li
- Adam Santoro
- David Raposo
- Vinicius Zambaldi
- Ryan Faulkner
- Victor Bapst
- Oriol Vinyals
- Nicolas Heess
- Andrea Tacchetti
- Francis Song
- Caglar Gulcehre
- Mateusz Malinowski
- Andrew Ballard
- Victoria Langston
- Chris Dyer
- Pushmeet Kohli
- Daan Wierstra
- Justin Gilmer
- George Dahl
- Many many more ...

Thank you