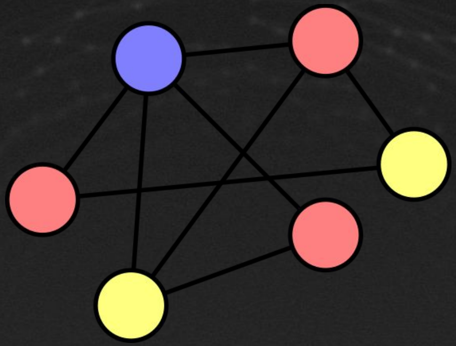


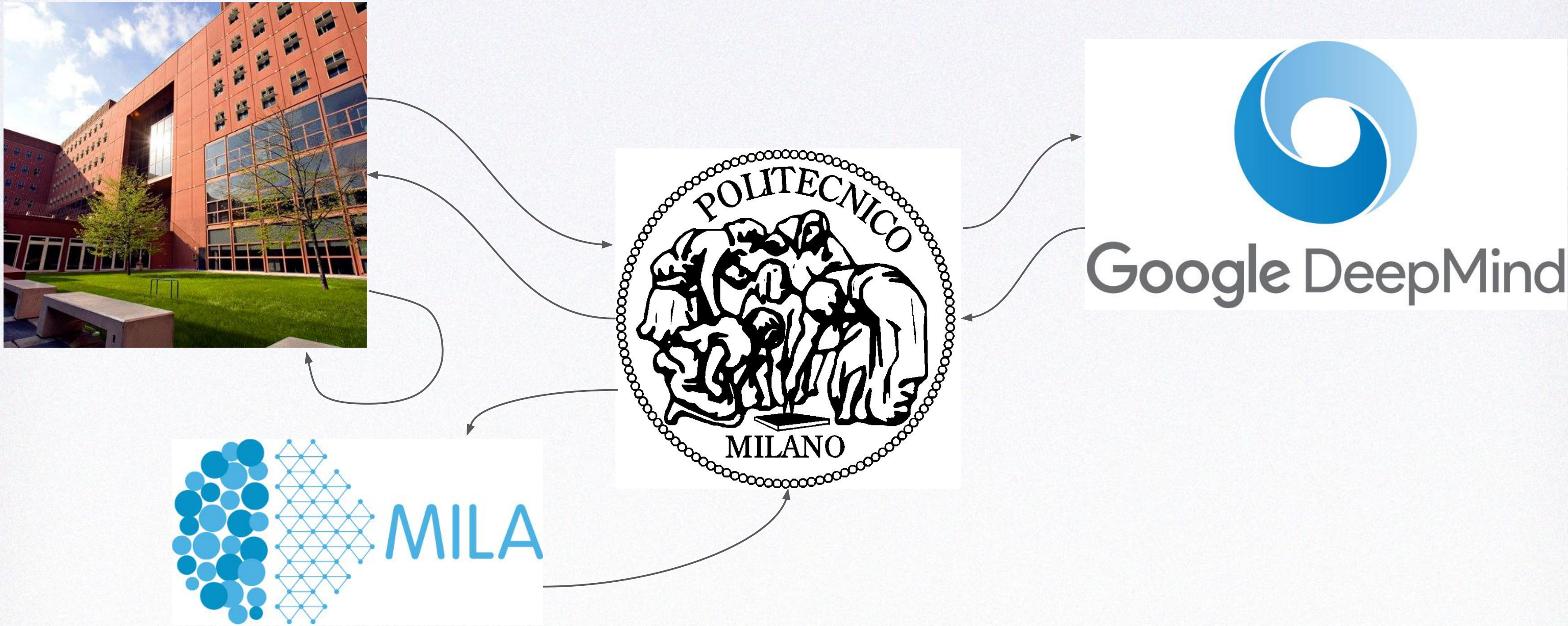
Graph neural networks

February 2018

Visin Francesco

DeepMind

Who I am

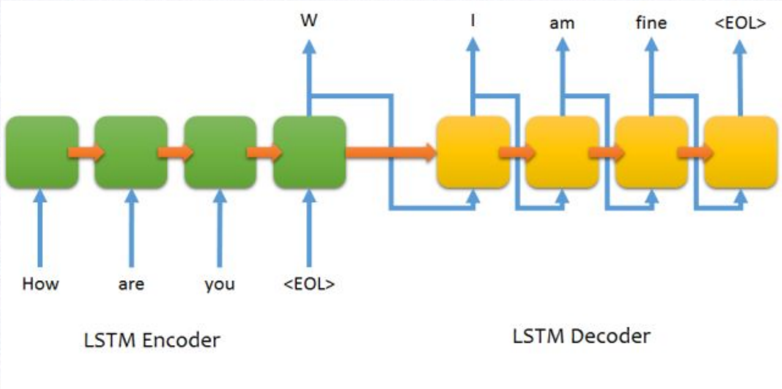


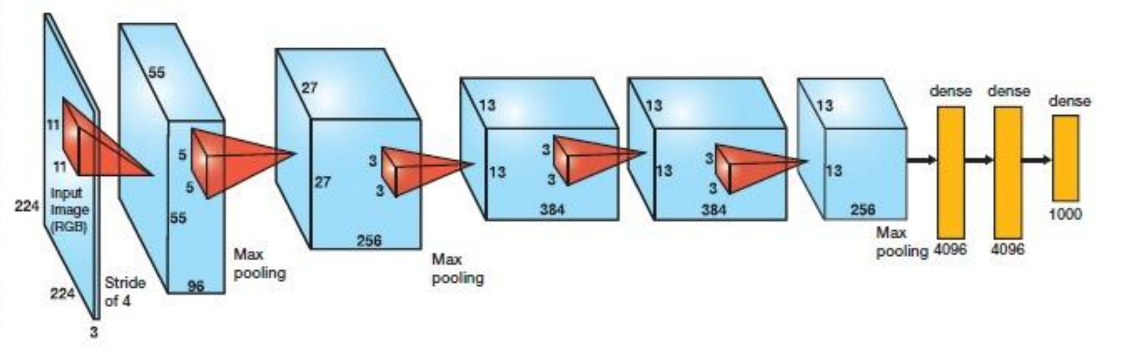
Outline

- Motivation and examples
- Graph nets
  - (Semi)-formal definition
  - Interaction network
  - Relation network
  - Gated graph sequence neural network
  - Attention is all you need
- Implementation example
- Conclusions

Motivation

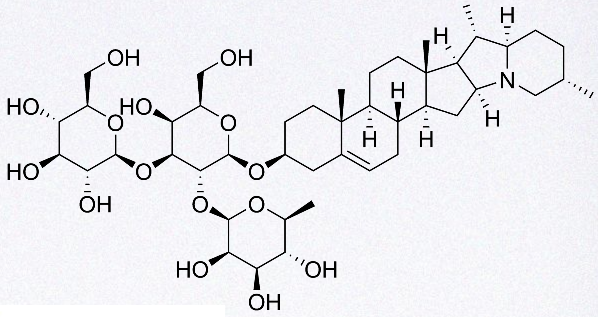
- The modern machine learning toolkit is well-suited to data that is:
  - **Fixed-length (MLPs)**
  - **Sequential (RNNs)**
  - **Spatial (CNNs)**





Motivation

- The modern machine learning toolkit is well-suited to data that is:
  - **Fixed-length (MLPs)**
  - **Sequential (RNNs)**
  - **Spatial (ConvNets)**
- But there is not much to support **graph structured** data.



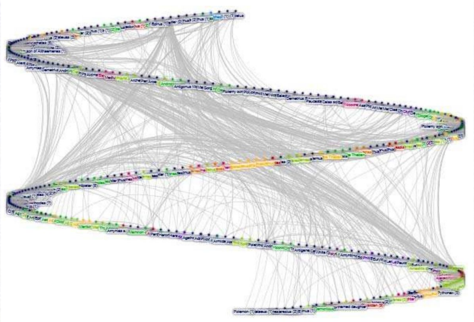
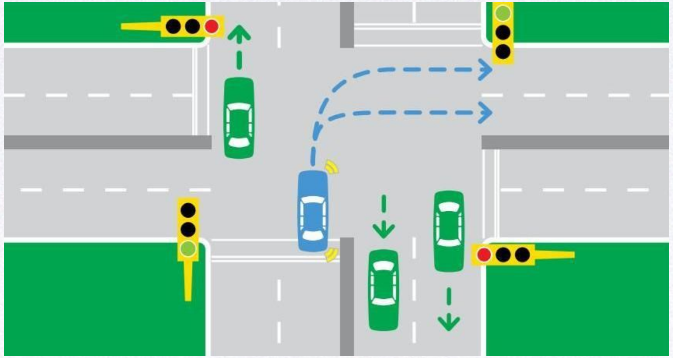


Image Credit - Diane Harris Cline, Vossman

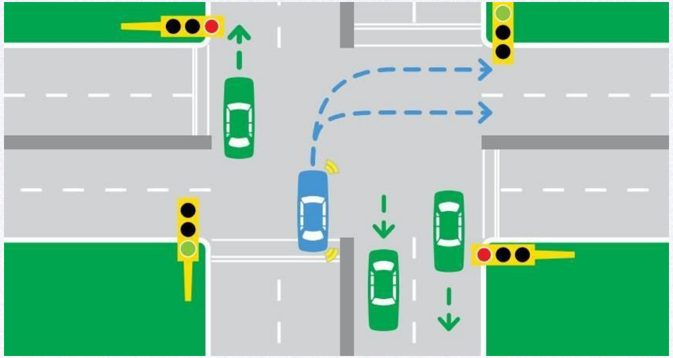
Motivation

- The world is complex, yet very structured



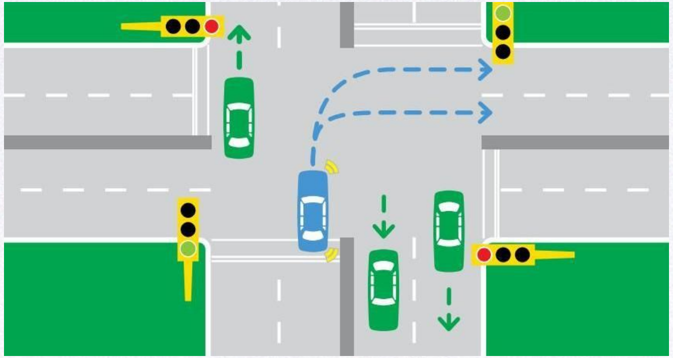
Motivation

- The world is complex, yet very structured
- Many things and problems are fundamentally relational

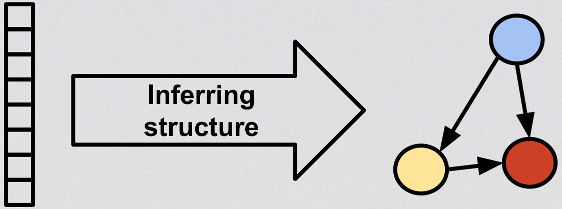


Motivation

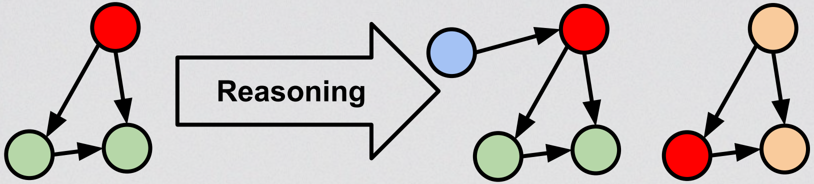
- The world is complex, yet very structured
- Many things and problems are fundamentally relational
- Many problems can be naturally represented as graphs.



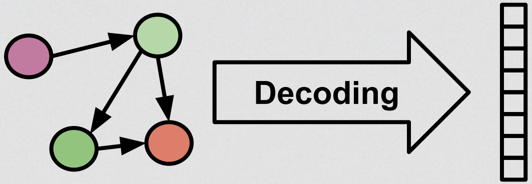
Tasks on graphs categorization



Inferring structure from unstructured data



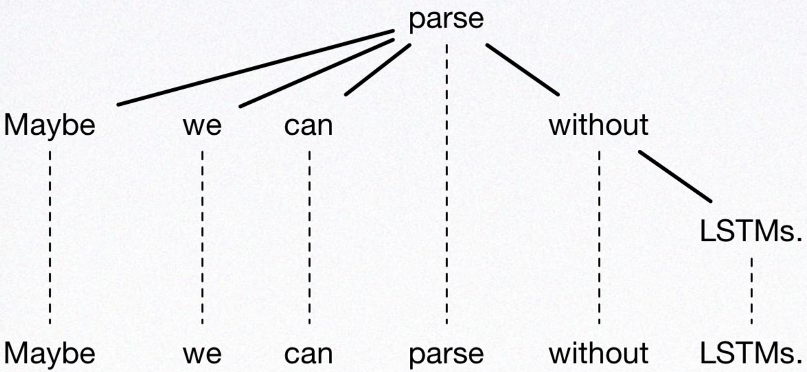
Reasoning about structure



Decoding structure

Categorization examples

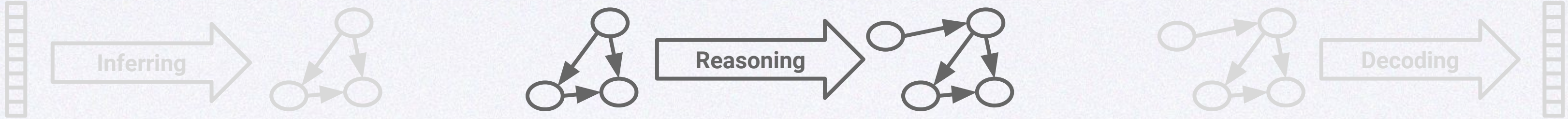
Parsing language



Visual scene recognition

Image Credit - Leibe et Al.

Categorization examples



Physical state prediction

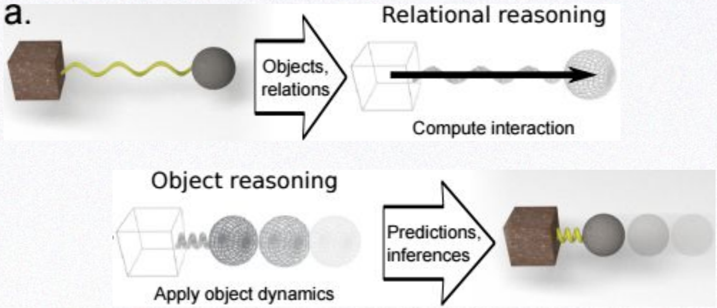


Image Credit - Battaglia et al.

Molecule structure generation

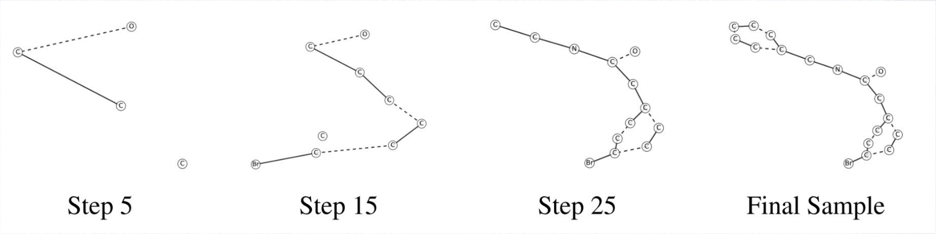
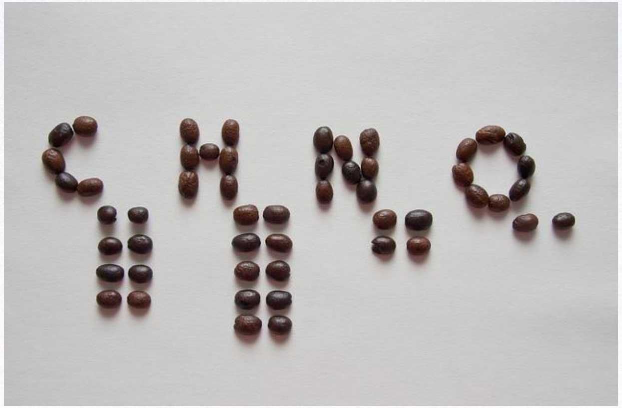


Image Credit - Li et al.

Categorization examples

Drug toxicity prediction



Finding the treasure

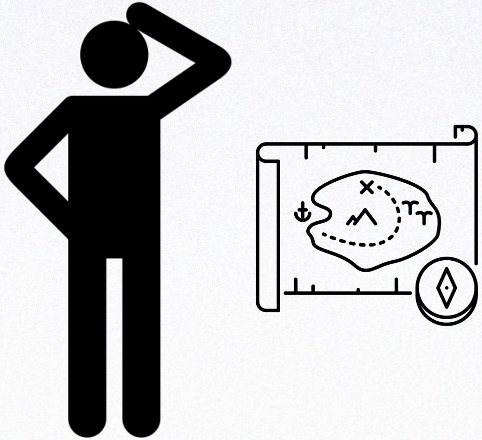


Image Credit - Greg Rodgers

Image Credit - Andrew Doane and lastspark, from Noun Project

Graphs & Computer Science/Machine Learning

- Graph algorithms
  - Dijkstra
  - Bellman-Ford

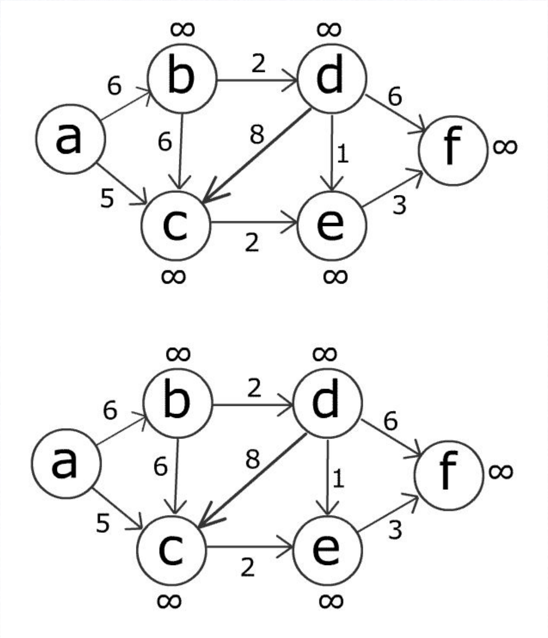


Image Credit - Artyom Kalinin

Graphs & Computer Science/Machine Learning

- Graph algorithms
  - Dijkstra
  - Bellman-Ford
- Hand-crafted features, graph kernels, etc ..
  - Measure similarity between graphs

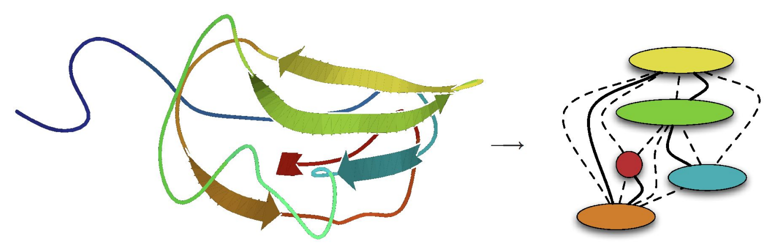


Image Credit - Vishwanathan et al.

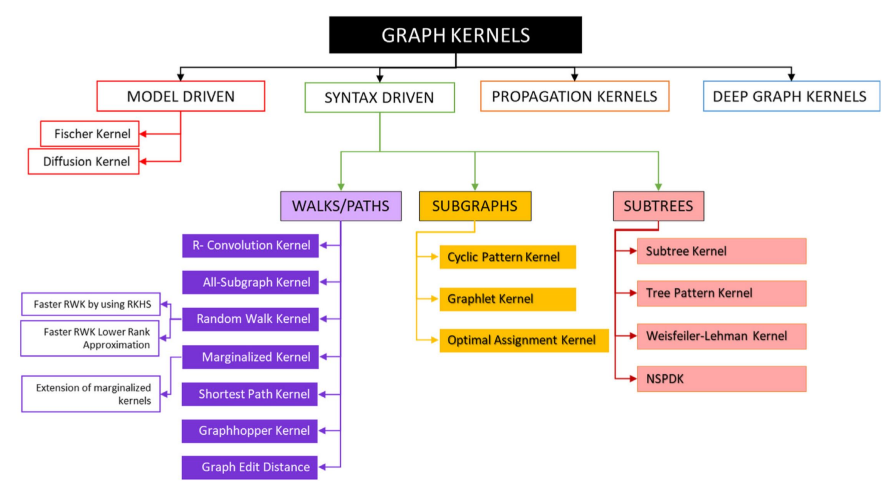


Image Credit -Ghosh et al.

Graphs & Computer Science/Machine Learning

- Graph algorithms
  - Dijkstra
  - Bellman-Ford
- Hand-crafted features, graph kernels, etc ..
  - Measure similarity between graphs
- Graphical models
  - Restricted Boltzmann Machines
  - Deep Belief Networks
  - Deep Boltzmann Machines

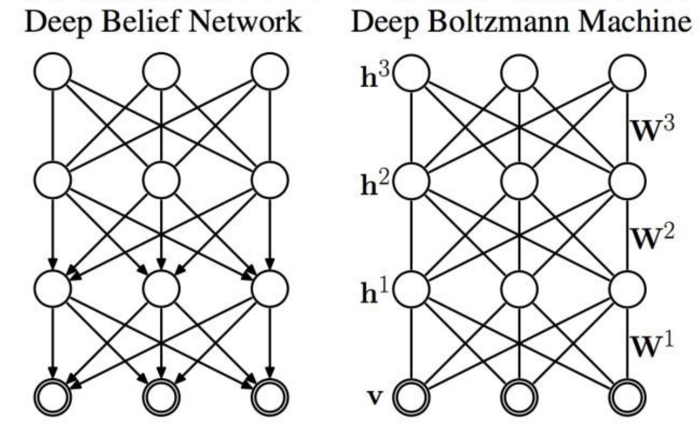


Image Credit - Geoffrey Hinton

Graphs & Computer Science/Machine Learning

- Graph algorithms
  - Dijkstra
  - Bellman-Ford
- Hand-crafted features, graph kernels, etc ..
- Graphical models
  - Restricted Boltzmann Machines
  - Deep Belief Networks
  - Deep Boltzmann Machines
- **Graph neural nets!**

Image Credit - Back to the future!

Right! ....but what is a Graph net??

Graph nets

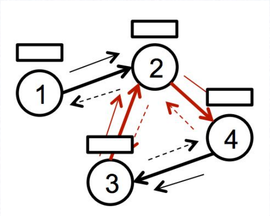
Literature (partial)

- **Growing interest:**
  - Graph Neural Networks (Scarselli et al 2007; 2008)
  - Pointer Networks (Vinyals et al 2015)
  - Graph Convolutional Networks (Bruna et al 2013; Duvenaud et al 2015; Henaff et al 2015; Kipf & Welling 2016; Schlichtkrull et al 2017; Defferrard et al 2017)
  - Gated Graph Neural Networks (Li et al 2015)
  - Interaction Networks (Battaglia et al 2016; Watters et al 2017;Raposo et al 2017;)
  - Message Passing Networks (Gilmer et al. 2017)
  - Deep Generative Models of Graphs (Li et al. 2018)

Graph nets

High level

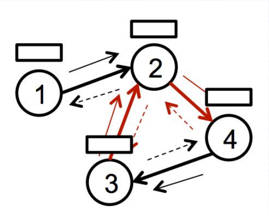
- Graph nets (GNs) are a class of models that:
  - Use graphs as
    - inputs and/or
    - outputs and/or
    - latent representation

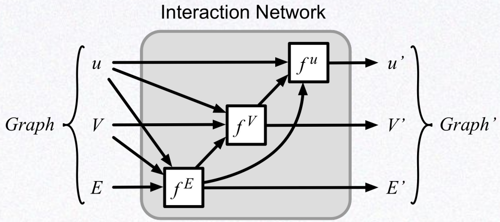


Graph nets

High level

- Graph nets (GNs) are a class of models that:
  - Use graphs as
    - inputs and/or
    - outputs and/or
    - latent representation
  - Manipulate graph-structured representations

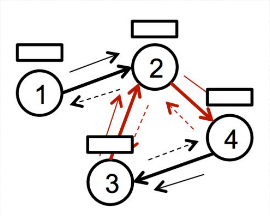


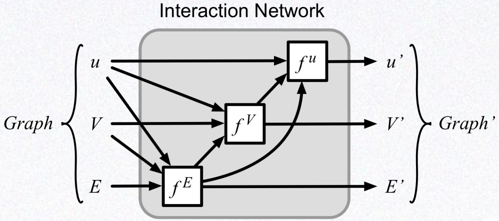


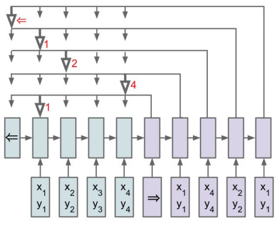
Graph nets

High level

- Graph nets (GNs) are a class of models that:
  - Use graphs as
    - inputs and/or
    - outputs and/or
    - latent representation
  - Manipulate graph-structured representations
  - Share model components across entities and relations



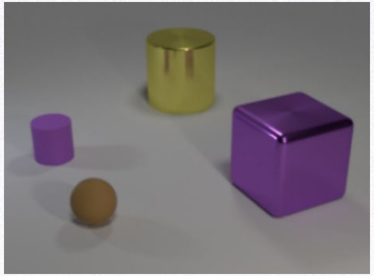




Graph nets

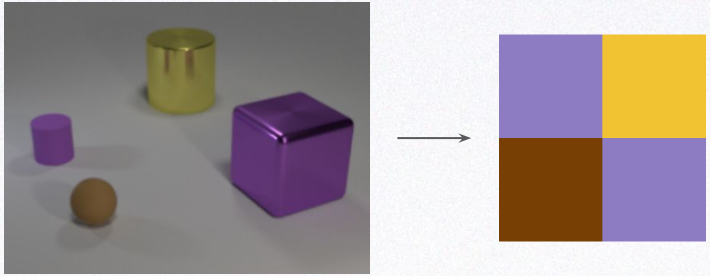
Core idea

Task: Is there a golden object in the scene?



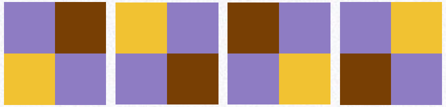
Graph nets

Core idea



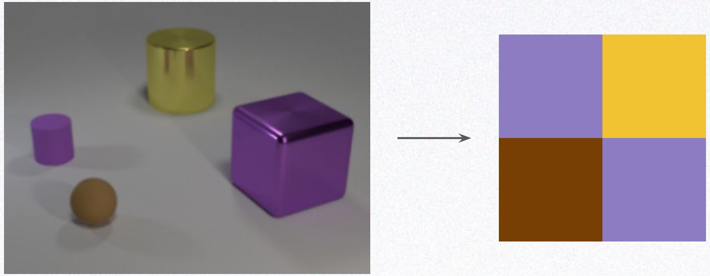
Task: Is there a golden object in the scene?

MLP: Inefficient learning; needs to see object of interest in all possible positions



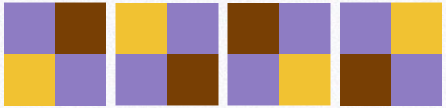
Graph nets

Core idea



Task: Is there a golden object in the scene?

MLP: Inefficient learning; needs to see object of interest in all possible positions

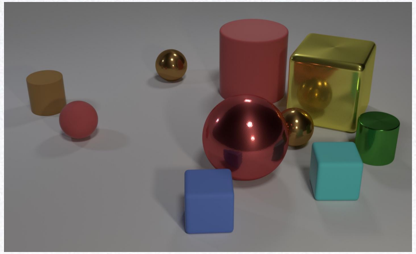


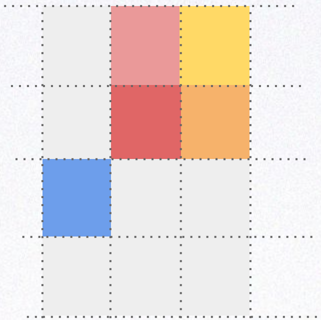
ConvNet: efficient; applies same function at each position (e.g. is the golden object at position x?) and then pools over the outcomes

Graph nets

Core idea

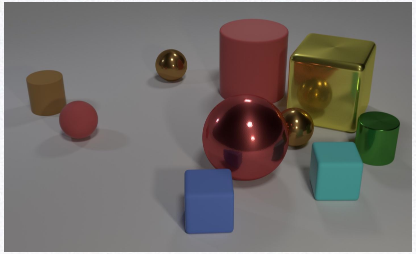
Task: Are there two red objects close to each other?

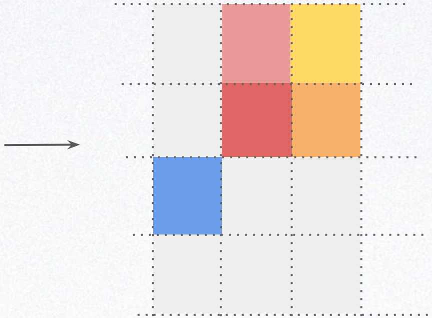




Graph nets

Core idea

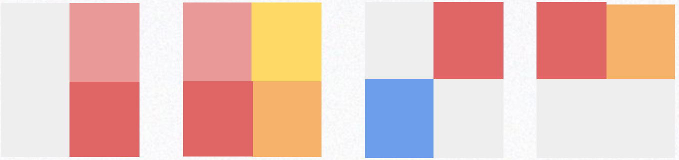




Task: Are there two red objects close to each other?

ConvNet: Use a kernel large enough (e.g. 2x2 for this) and check if one has 2 red object in its receptive field

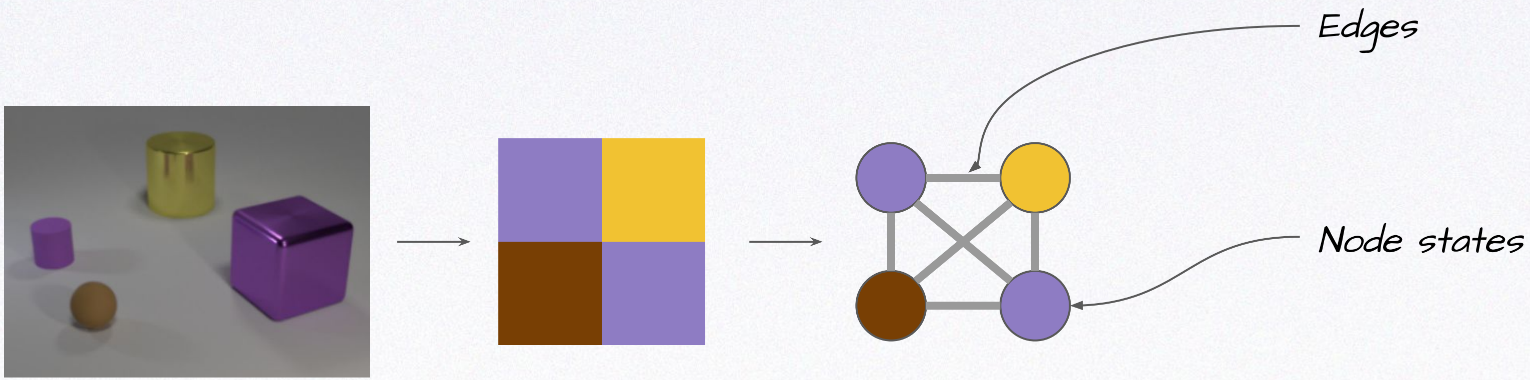
Observed patches by the 2x2 kernel are:



Note: within the patch, convnet needs to experience all permutations

Graph nets

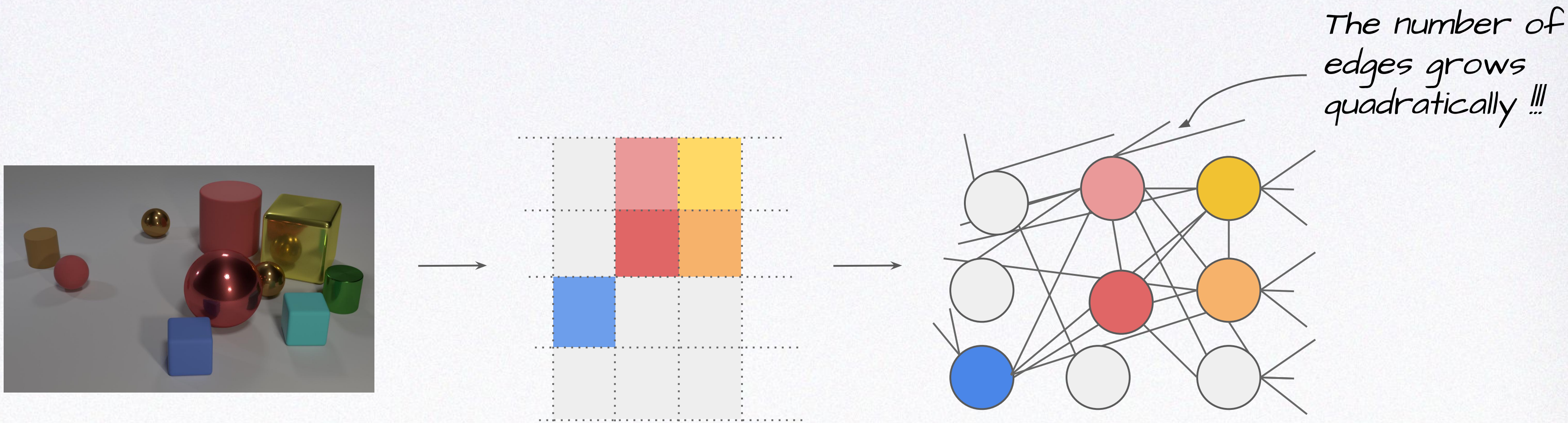
Core idea



Graph Nets: Think of your observation as a graph

Graph nets

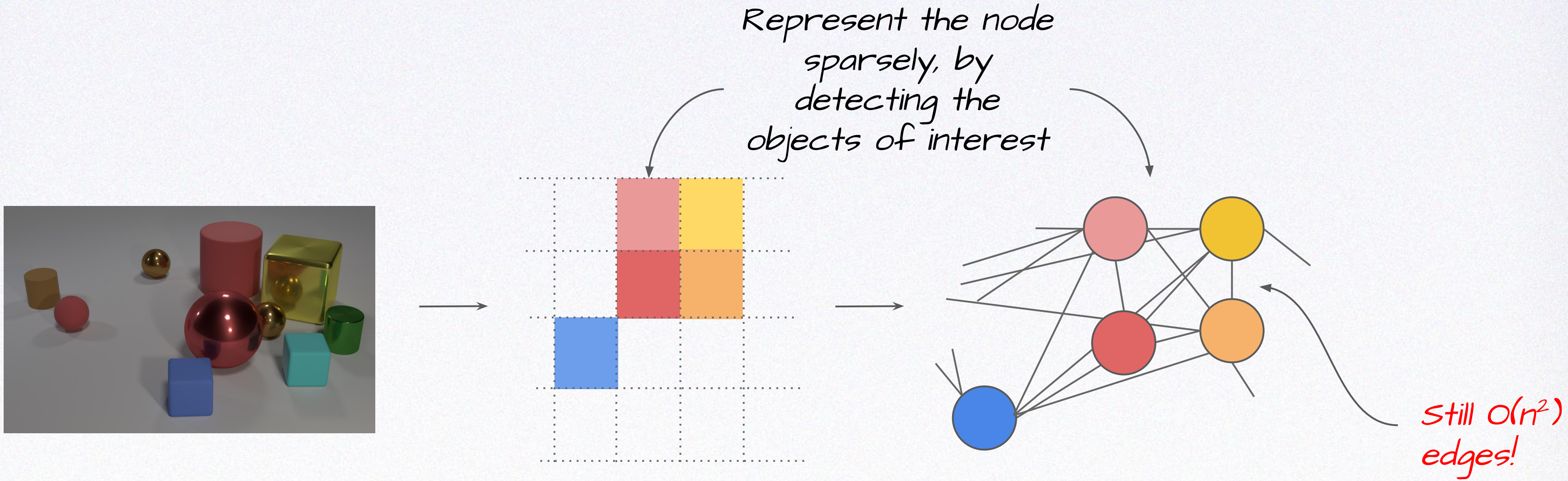
Core idea



Graph Nets: Think of your observation as a graph

Graph nets

Core idea

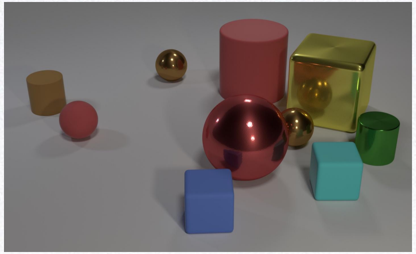


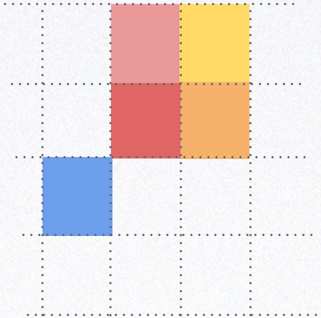
Graph Nets: Think of your observation as a graph

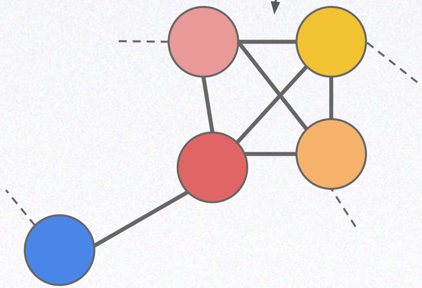
Graph nets

Core idea

Use a sparse representation for the edges, if you know which objects are allowed to interact



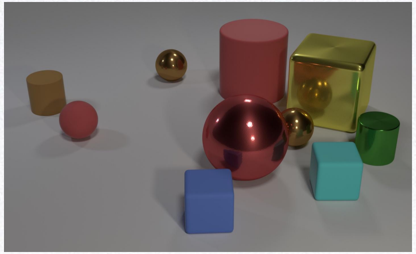


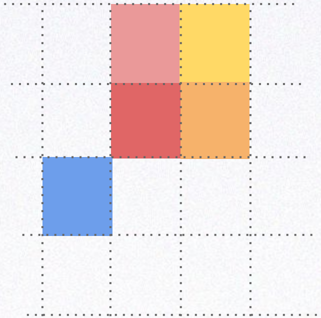


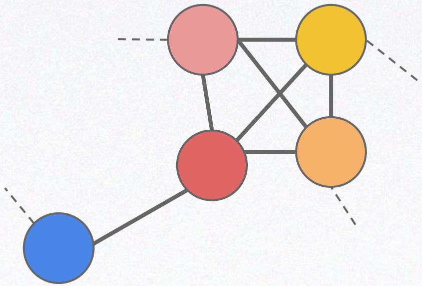
Graph Nets: Think of your observation as a graph

Graph nets

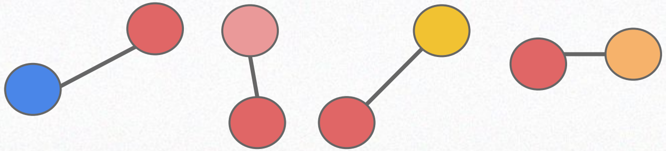
Core idea





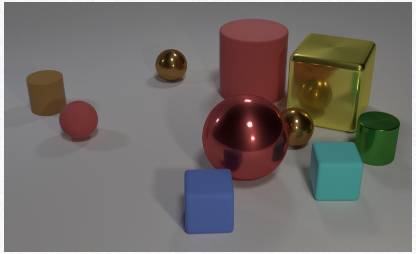


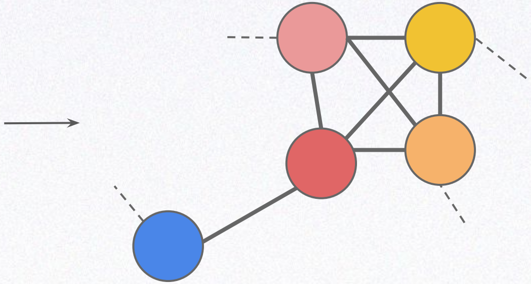
Reason in terms of pairs !



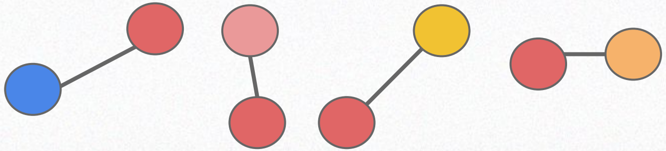
Graph nets

Core idea





Reason in terms of pairs !

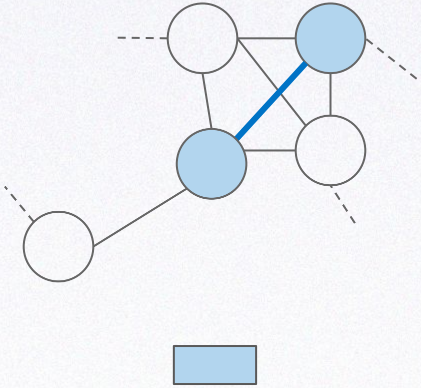


- Same function re-used for **every edge** to compute its effect
- Same function re-used for **every node** to compute its new state given sum of all incoming effects and its state
- Invariant to the position of objects !
- Invariant to the distance between objects in topological space

Graph nets

States update

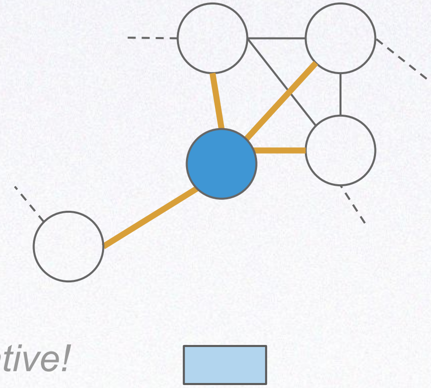
- **Edge effects:**
  - **edge** type
  - <*related* **node** states>
  - **global** state



Graph nets

States update

- **Edge effects:**
  - edge type
  - <*related* node states>
  - global state
- **Node update:**
  - **node** state
  - ***summed*** effects on incoming edges
  - **global** state

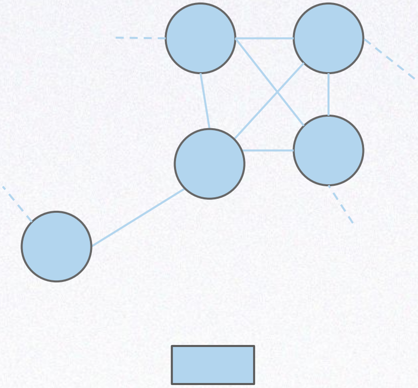


Sum is commutative and associative!

Graph nets

Global state update

- **Edge effects:**
  - edge type
  - <*related* node states>
  - global state
- **Node update:**
  - node state
  - summed effects on incoming edges
  - global state
- **Global state update:**
  - <**node** states>
  - **global** state



DeepMind

Graph nets

Components

- $\mathbf{G} = \langle O, R \rangle$ — **Graph**
- $\mathbf{O} = \{o_1, o_2, .., o_m\}$ — Collection of **objects**
  - **o_i** $= \{o_i^1, o_i^2, ..., o_i^n\}$ — Object *i* with *n* **features**
- $\mathbf{R} = \{r_1, r_2, .., r_k\}$ — Collection of **relations** between objects
  - $\mathbf{r_k} = \langle o_i, o_j, a_k \rangle$ — Relation *k* between o_i *and* o_j with **attribute** a_k
- $\mathbf{E} = \{e_k\}$ — Collection of **effects** of relations
- $\mathbf{X} = \{x_l\}$ — Collection of **external effects**

Graph nets

Components

$$\text{out} = \phi_A(g(\phi_O(\alpha(X, \phi_R(m(G))))))$$

Graph

Marshalling function

Relation model:
predict the effects

External effects

Aggregation function

Object model

Aggregation function

Abstraction model

Interaction networks

$$\mathbf{out} = \phi_A(g(\phi_O(\alpha(X, \phi_R(m(G))))))$$

MLP

Fixed! Matrix prod

MLP columnwise

Fixed! Matrix prod

MLP columnwise

Fixed! Matrix prod + concat

Graph

Marshalling function

Relation model: predict the effects

External effects

Objects

Aggregation function

Object model

Aggregation function

Abstraction model

Interaction networks

$$\textbf{out} = \phi_A(g(\phi_O(\alpha(X, \phi_R(m(G))))))$$

- Reason about interactions between objects
- Physical reasoning
  - n-body domain: galaxy of objects that interact
  - bouncing balls: balls in a box bounce on walls and can collide
  - string comprised of masses connected by springs
- Learnable physics engine simulation!
- Works/transfers to variable number of objects!
- Graph to graph

Interaction networks

$$\mathbf{out} = \phi_A(g(\phi_O(\alpha(X, \phi_R(m(G))))))$$

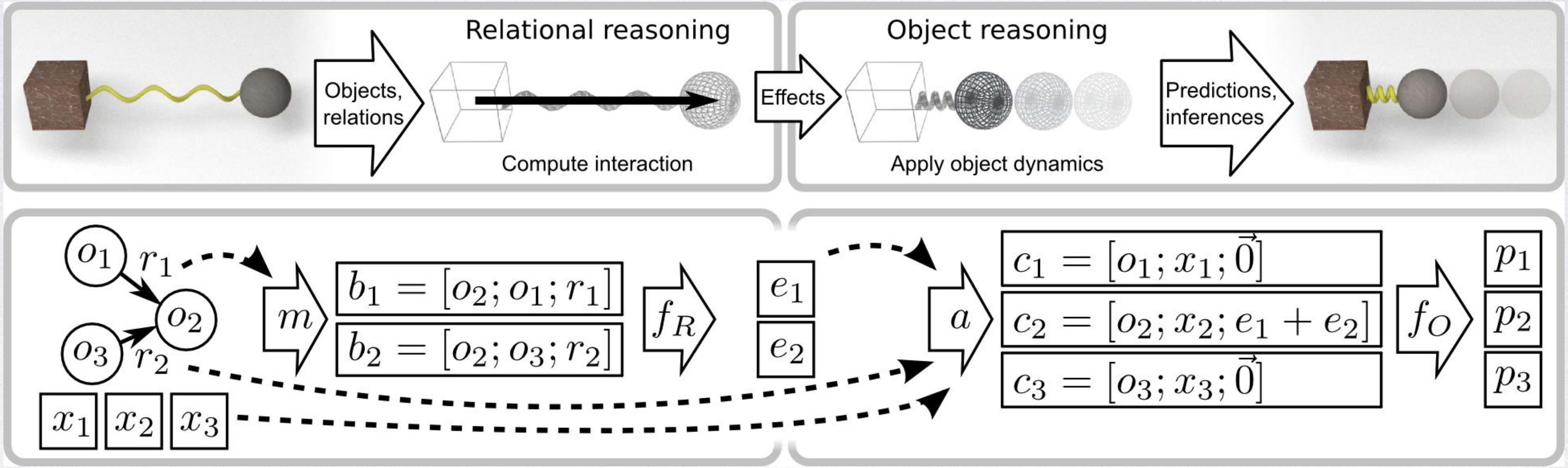
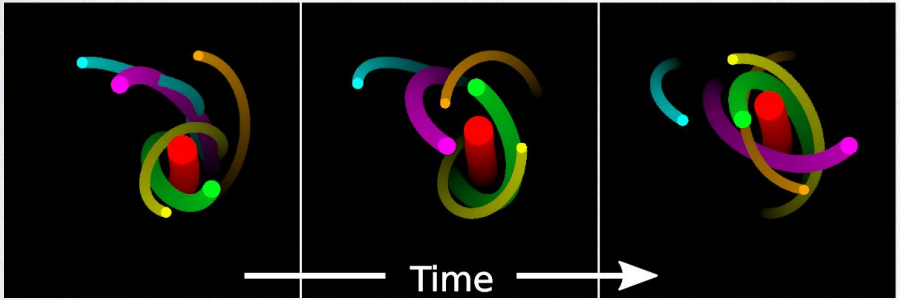


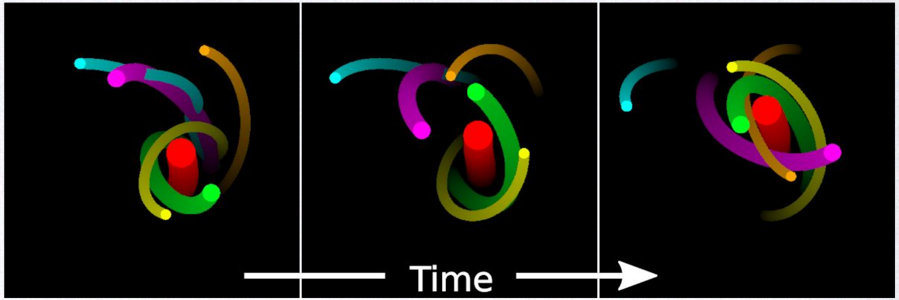
Image Credit - Battaglia et al.

Interaction networks

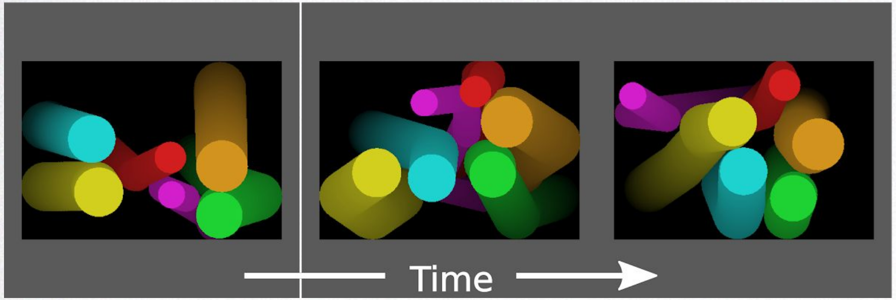
True



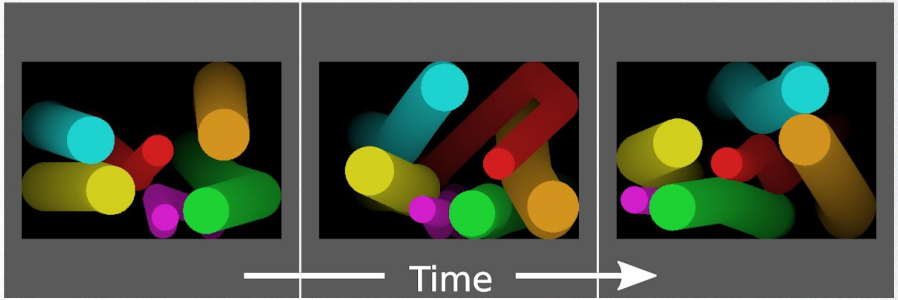
Model



True

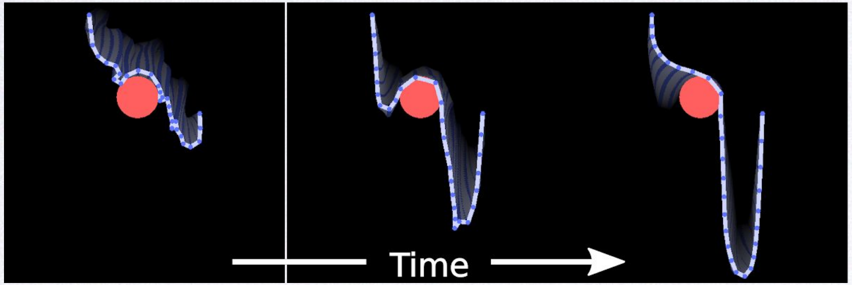


Model

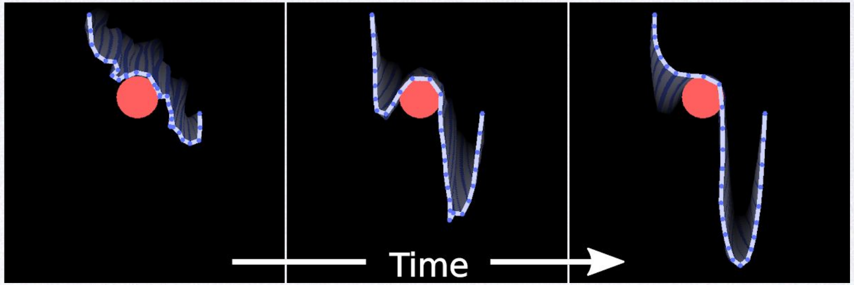


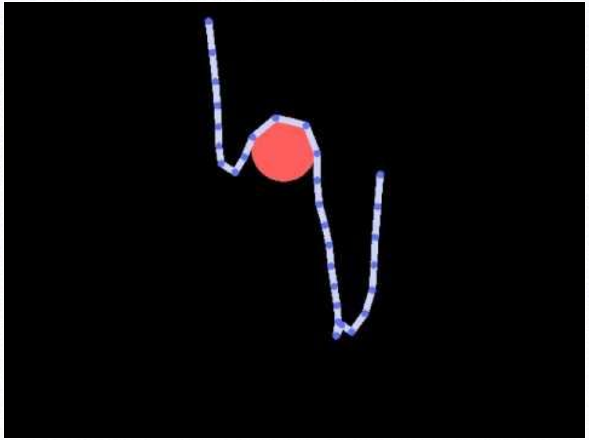
Interaction networks

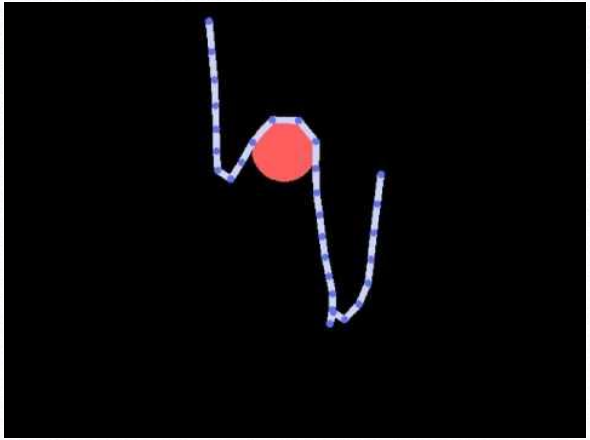
True



Model





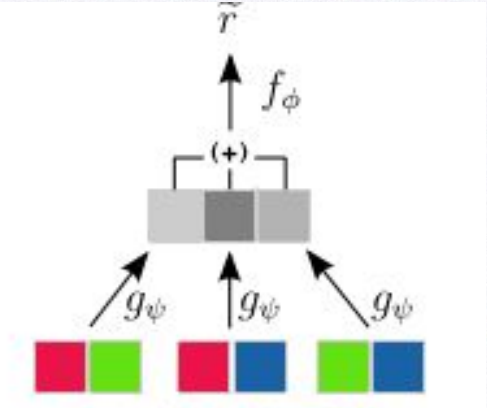


Relation networks

$$\text{out} = \phi_A(g(\phi_O(\alpha(X, \phi_R(m(G))))))$$

Implicit (fixed: tuples)

Not modeled



(c) Object pair prior

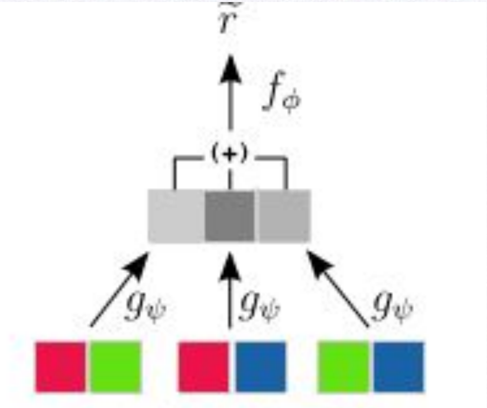
Relation networks

$$\text{out} = \phi_A(g(\phi_O(\alpha(X, \phi_R(m(G))))))$$

MLP columnwise

Any aggregation
function commutative
and associative

MLP



(c) Object pair prior

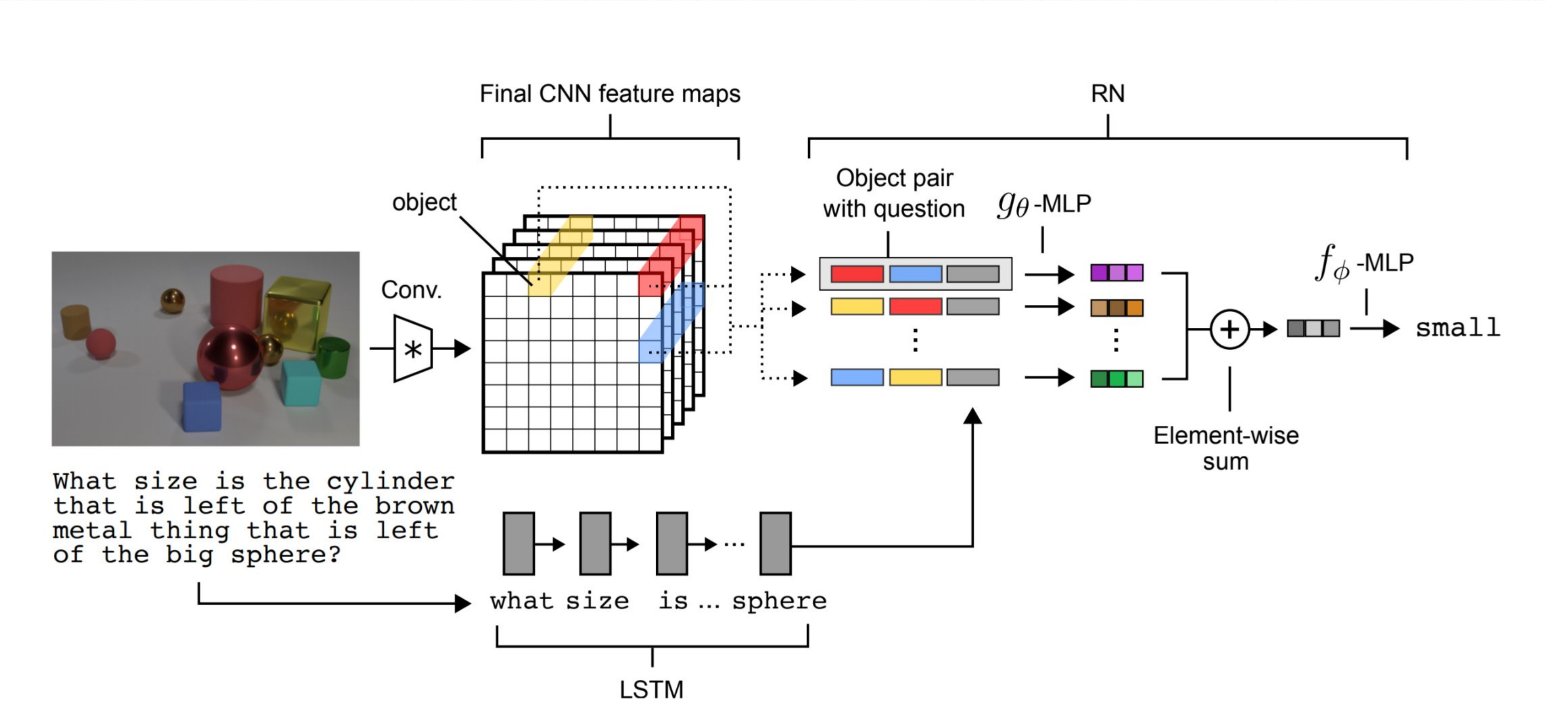
Relation networks

$$\textbf{out} = \phi_A(g(\phi_O(\alpha(X, \phi_R(m(G))))))$$

- Reason about objects and their relations
  - Classification of scenes from graph representation
  - Classification of scenes from raw input
  - Exploit the relations to perform one-shot relation learning on a new scene
- Graph to vectors

Relation networks on CLEVR

$$\textbf{out} = \phi_A(g(\phi_O(\alpha(X, \phi_R(m(G))))))$$



Gated graph sequence neural networks

$$\text{out} = \phi_A(g(\phi_O(\alpha(X, \phi_R(m(G))))))$$

Connectivity function

Linear

Sum over nodes +
soft self attention

GRU

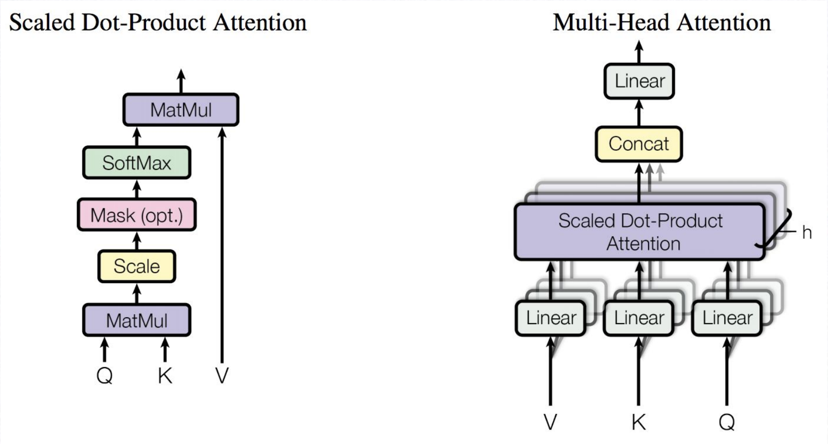
Gated graph sequence neural networks

$$\textbf{out} = \phi_A(g(\phi_O(\alpha(X, \phi_R(m(G))))))$$

- Reason about verification of computer programs
- Attempt to prove code properties, such as memory safety
- BABI task (language comprehension)
- Per-node predictions
- Gated (GRU style) updates to the nodes
- Internal propagation steps while generating outputs sequentially
- First follow up of Scarselli 2008

Attention is all you need

$$out = \phi_A(g(\phi_O(\alpha(X, \phi_R(m(G))))))$$



Different input tokens (nodes)

Masking (optional)

Linear (resulting in key, query & value)

Attention (including accumulation); head Concat

Attention is all you need

$$\textbf{out} = \phi_A(g(\phi_O(\alpha(X, \phi_R(m(G))))))$$

- Consider each time-step (input or hidden state) as a node
- Reason about the inner relations in its hidden state (over time)
  - Exploit self attention as a form of recursive memory
  - Multiple *attention heads* in parallel (eq. to multiple edges per same nodes in IN)
- Not explicitly introduced as a graph network approach
- Equivalent to Graph Convolutional Net, or Graph net with a fully connected graph but with attention on the edges
- Graph to vector

Cool! Where can I get one??

Graph Propagation Core

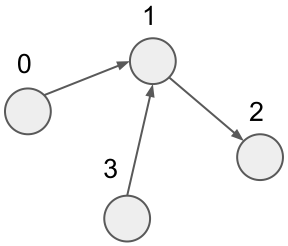
```python
# node states and edges
states = tf.placeholder(tf.float32)
edges = tf.placeholder(tf.int32)

# 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)

# 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))
```

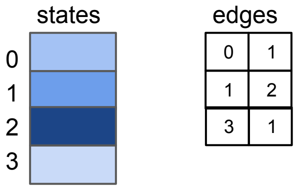


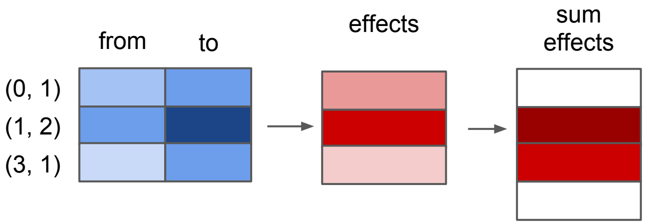
$$G = (V, E) \text{ state } \mathbf{x}_v \; \forall v \in V$$

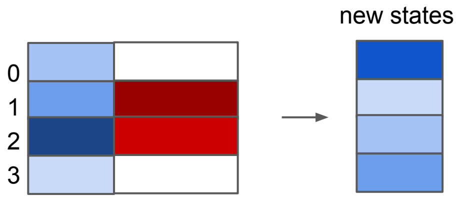
$$\forall (u, v) \in E, \; \mathbf{e}_{u \to v} = f_e(\mathbf{x}_u, \mathbf{x}_v)$$

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

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







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

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

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

```
# 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
```

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

Output Modules

```
# relational network style graph-level output

# concat_states <- paired states for all (i,j) pairs
effects = effect_net(concat_states)
graph_vec = tf.reduce_sum(effects, axis=0)
graph_output = graph_level_network(graph_vec)
...     # feed into your favorite loss
```

Relation network style graph-level output

$$\mathbf{o}_G = f_\phi\left(\sum_{i,j} g_\theta(\mathbf{x}_i, \mathbf{x}_j)\right)$$

Conclusions

- Graph nets are a powerful model to reason on graph related structures
- There are three main categories of tasks in this domain:
  - vector to graph
  - graph to graph
  - graph to vector
- Several variants of graph nets have been proposed in the literature
- They are easy to implement!
- Sky is the limit: surprise us!!!

THANK YOU

Credits

Razvan Pascanu, Victor Bapst