

DOWN

replace agg. $\rightarrow v' = \text{softmax}(Q \times k^T) \cdot V$
 \rightarrow replace MLP by linear w/ $\# \text{ cells} = \mathcal{O}(1)$

Algorithm 1: GNN maps state s into scalar $V(s)$

Input: State s , set of atoms true in s , set of objects

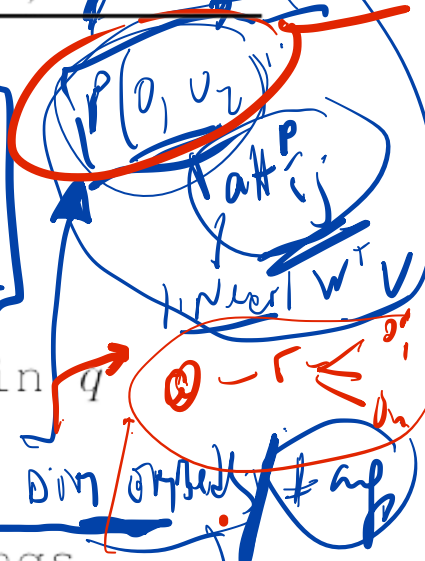
Output: $V(s)$

```

1  $f_0(o) \sim \mathbf{0}^{k/2} \mathcal{N}(0, 1)^{k/2}$  for each object  $o \in \mathcal{O}$ 
2 for  $i \in \{0, \dots, L-1\}$  do
3   for each atom  $q := p(o_1, \dots, o_m)$  true in  $s$  do
4     // Msgs  $q \rightarrow o$  for each  $o = o_j$  in  $q$ 
5      $m_{q,o} := [\text{MLP}_p(f_i(o_1), \dots, f_i(o_m))]_j$ 
6     for each  $o$  in  $s$  do
7       // Aggregate, update embeddings
8        $f_{i+1}(o) := \text{MLP}_U(f_i(o), \text{agg}(\{m_{q,o} | o \in q\}))$ 
9   // Final Readout
10   $V := \text{MLP}_2(\sum_{o \in s} \text{MLP}_1(f_i(o)))$ 
  
```

why not V narrow?

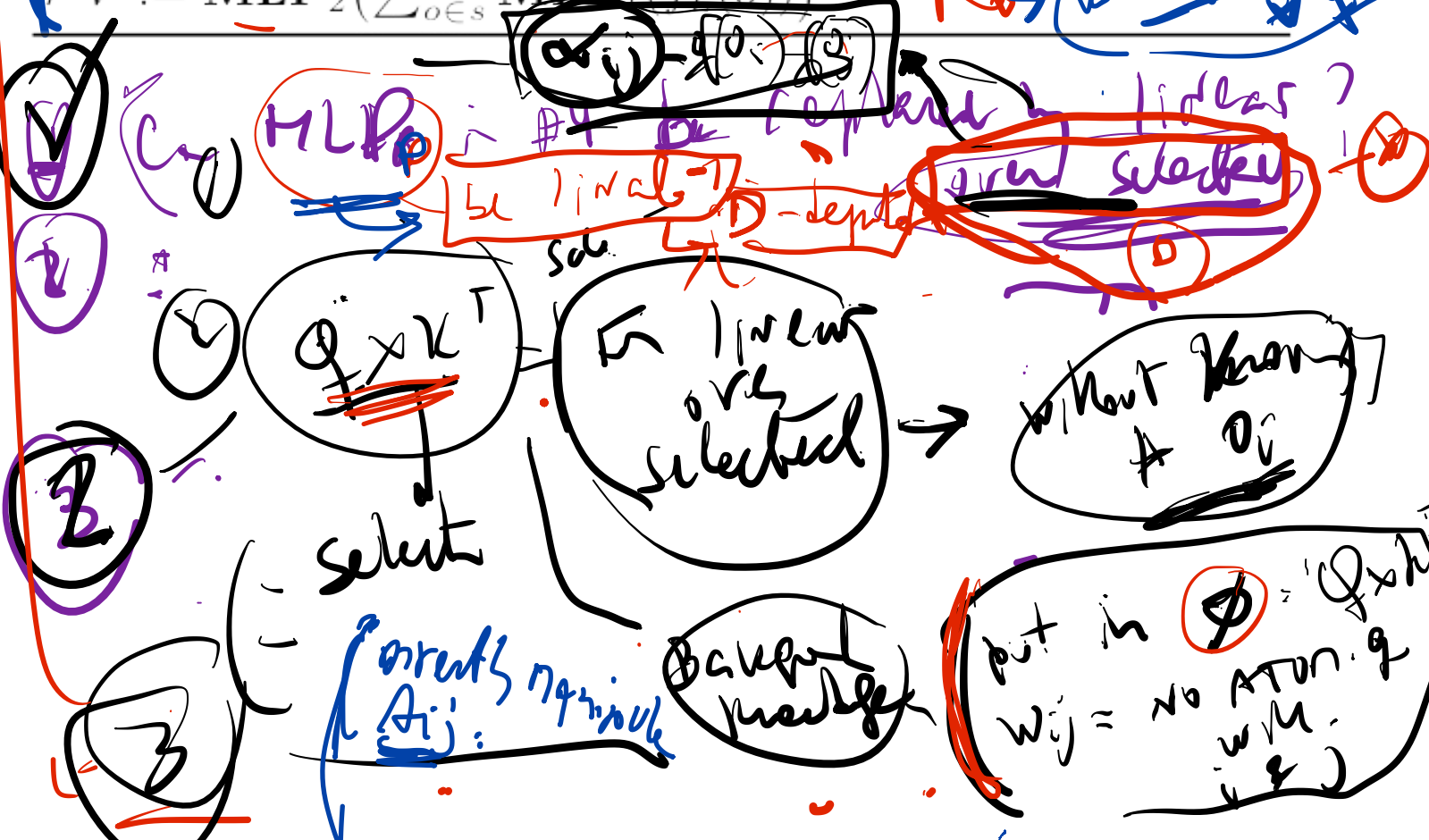
$V' = W \cdot z$
 selective
 all



key

Needs to know dim objects \neq $\#$ obj

all aggregated $\rightarrow q$



Relational Morris:

GNNs are often realized as follows (Morris et al., 2019). In each layer, $t > 0$, we compute vertex features

$$h_v^{(t)} := \sigma \left(h_v^{(t-1)} W_0^{(t)} + \sum_{w \in N(v)} h_w^{(t-1)} W_1^{(t)} \right) \in \mathbb{R}^e, \quad (1)$$

for v in $V(G)$, where $W_0^{(t)}$ and $W_1^{(t)}$ are parameter matrices from $\mathbb{R}^{d \times e}$ and σ denotes an entry-wise non-linear function, e.g., a sigmoid or a ReLU function.⁴ Following Gilmer et al. (2017) and Scarselli et al. (2009), in each layer, $t > 0$, we can generalize the above by computing a vertex feature

$$h_v^{(t)} := \text{UPD}^{(t)} \left(h_v^{(t-1)}, \text{AGG}^{(t)} \left(\{ \{ h_w^{(t-1)} \mid w \in N(v) \} \} \right) \right),$$

where $\text{UPD}^{(t)}$ and $\text{AGG}^{(t)}$ may be differentiable parameterized functions, e.g., neural networks.⁵ In the case of graph-level tasks, e.g., graph classification, one uses

$$h_G := \text{READOUT} \left(\{ \{ h_v^{(T)} \mid v \in V(G) \} \} \right),$$

to compute a single vectorial representation based on learned vertex features after iteration T . Again, READOUT may be a differentiable parameterized function. To adapt the parameters of the above three functions, they are optimized end-to-end, usually through a variant of stochastic gradient descent, e.g., (Kingma and Ba, 2015), together with the parameters of a neural network used for classification or regression.

Graph Neural Networks for Multi-relational Graphs. In the following, we describe GNN layers for multi-relational graphs, namely R-GCN (Schlichtkrull et al., 2018) and CompGCN (Vashishth et al., 2020). Initial features are computed in the same way as in the previous subsection.

R-GCN. Let G be a labeled multi-relational graph. In essence, R-GCN generalizes Equation 1 by using an additional sum iterating over the different relations. That is, we compute a vertex feature

$$h_{v, \text{R-GCN}}^{(t)} := \sigma \left(h_{v, \text{R-GCN}}^{(t-1)} W_0^{(t)} + \sum_{i \in \{r\}} \sum_{w \in N_i(v)} h_{w, \text{R-GCN}}^{(t-1)} W_i^{(t)} \right) \in \mathbb{R}^e, \quad (2)$$

$i \in \{r\} \mid w \in N_i(v)$

binary
relats

(1): MLP
in
sigmoid

- linear
- No concat map
- binary relats

(2): Matrices W_i^R one per R, separate linear

How $R(i,j)$ output of $R(j,i)$ undirected? ?

$$Q = HW_Q, \quad K = HW_K, \quad V = HW_V,$$

$$A = \frac{QK^\top}{\sqrt{d_v}}, \quad \text{Attn}(H) = \text{softmax}(A)V,$$

$A_{ij} = w_{ij}^R$
 $= \begin{cases} 0 & \text{if } i=j \\ \dots & \text{if } i \neq j \end{cases}$
 $\times W$ not used
 Binary R!

Algorithm 1: GNN maps state s into scalar $V(s)$

Input: State s : set of atoms true in s , set of objects
Output: $V(s)$

- 1 $f_0(o) \sim \mathbf{0}^{k/2} \mathcal{N}(0, 1)^{k/2}$ for each object $o \in s$;
- 2 **for** $i \in \{0, \dots, L-1\}$ **do**
- 3 **for each atom** $q := p(o_1, \dots, o_m)$ **true in** s **do**
- 4 // Msgs $q \rightarrow o$ for each $o = o_j$ in q
- 5 $m_{q,o} := [\text{MLP}_p(f_i(o_1), \dots, f_i(o_m))]_j$;
- 6 **for each** o **in** s **do**
- 7 // Aggregate; update embeddings
- 8 $f_{i+1}(o) := \text{MLP}_U(f_i(o), \text{agg}(\{m_{q,o} \mid o \in q\}))$;
- 9 // Final Readout
- 10 $V := \text{MLP}_2(\sum_{o \in s} \text{MLP}_1(f_L(o)))$

Why not use unary pred here

Fn diff reads: CONCAT?

Linear MLP
 make part of $f(o)$
 @ node (p o i l)
 Subseq Recp

Barcelo/Morales

q same $f(i)$

$$h_v^{(t)} := \sigma(h_v^{(t-1)}W_0^{(t)} + \sum_{w \in N(v)} h_w^{(t-1)}W_1^{(t)}) \in \mathbb{R}^c,$$

), where $W_0^{(t)}$ and $W_1^{(t)}$ are parameter matrices from $\mathbb{R}^{d \times c}$ and σ denote activation, e.g., a sigmoid or a ReLU function.⁴ Following Gilmer et al. (2017), in each layer, $t > 0$, we can generalize the above by computing a vertex

$$h_v^{(t)} := \text{UPD}^{(t)}(h_v^{(t-1)}, \text{AGG}^{(t)}(\{h_w^{(t-1)} \mid w \in N(v)\})),$$

6 W nodes

(1) Each o_i selects some o_j through local TA
 (2) selection based on $f(o_i)$ NOT o_j

R-GCN. Let G be a labeled multi-relational graph. In essence, R-GCN generalizes Equation 1 using an additional sum iterating over the different relations. That is, we compute a vertex feature

$$h_{v,R\text{-GCN}}^{(t)} := \sigma(h_{v,R\text{-GCN}}^{(t-1)}W_0^{(t)} + \sum_{i \in [r]} \sum_{w \in N_i(v)} h_{w,R\text{-GCN}}^{(t-1)}W_i^{(t)}) \in \mathbb{R}^c,$$

for v in $V(G)$, where $W_0^{(t)}$ and $W_i^{(t)}$ for i in $[r]$ are parameter matrices from $\mathbb{R}^{d \times c}$, and σ denote entry-wise non-linear function. We note here that the original R-GCN layer defined in Schlichtkrull et al. (2018) uses a mean operation instead of a sum in the most inner sum of Equation 2. We investigate the empirical advantages of these two variations in Section 5.

CompGCN. Let G be a labeled multi-relational graph. A CompGCN layer generalizes Equation 1 by encoding relational information as edge features. That is, we compute a vertex feature

$$h_{v,\text{CompGCN}}^{(t)} := \sigma(h_{v,\text{CompGCN}}^{(t-1)}W_0^{(t)} + \sum_{i \in [r]} \sum_{w \in N_i(v)} \phi(h_{w,\text{CompGCN}}^{(t-1)}, z_i^{(t)})W_i^{(t)}) \in \mathbb{R}^c,$$

$W_K W_V$

(3) Same W_i $\neq o_i o_j$

via this one
 arbitrary R?

OWN - replace agg - msgs $\rightarrow v' = \text{softmax} \left(\frac{Q \times K^T}{\sqrt{d_k}} \right) \cdot V$
 replace MLP_p by linear w/ # slots $(\frac{q}{p})$

Algorithm 1: GNN maps state s into scalar $V(s)$

Input: State s : set of atoms true in s , set of objects

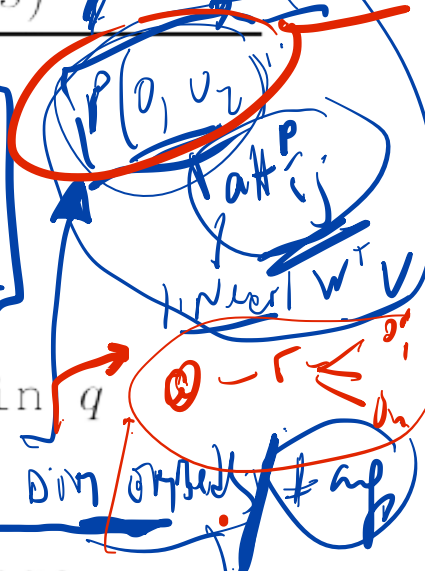
Output: $V(s)$

```

1  $f_0(o) \sim \mathbf{0}^{k/2} \mathcal{N}(0, 1)^{k/2}$  for each object  $o \in \text{objects}$ 
2 for  $i \in \{0, \dots, L-1\}$  do
3   for each atom  $q := p(o_1, \dots, o_m)$  true in  $s$  do
4     // Msgs  $q \rightarrow o$  for each  $o = o_j$  in  $q$ 
5      $m_{q,o} := [\text{MLP}_p(f_i(o_1), \dots, f_i(o_m))]_j$ 
6     for each  $o$  in  $s$  do
7       // Aggregate, update embeddings
8        $f_{i+1}(o) := \text{MLP}_U(f_i(o), \text{agg}(\{m_{q,o} | o \in q\}))$ 
9   // Final Readout
10   $V := \text{MLP}_2(\sum_{o \in s} \text{MLP}_1(f_L(o)))$ 
  
```

very low

$V' = W \cdot [z_{ij}]$
 selective NOT on i all

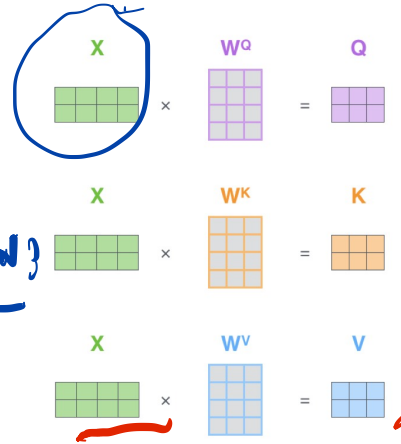


Needs to know dim objects \neq agg

all aggregated $\downarrow q$

if $MLP_p \rightarrow$ self affect $V \rightarrow V'$
 $V' \rightarrow MLP_q \rightarrow V''$ (approx)

\rightarrow w/ GNN work with $MLP_p \Rightarrow$ linear?
 which matrix $= k, q, w \Rightarrow$ will yield GNN?



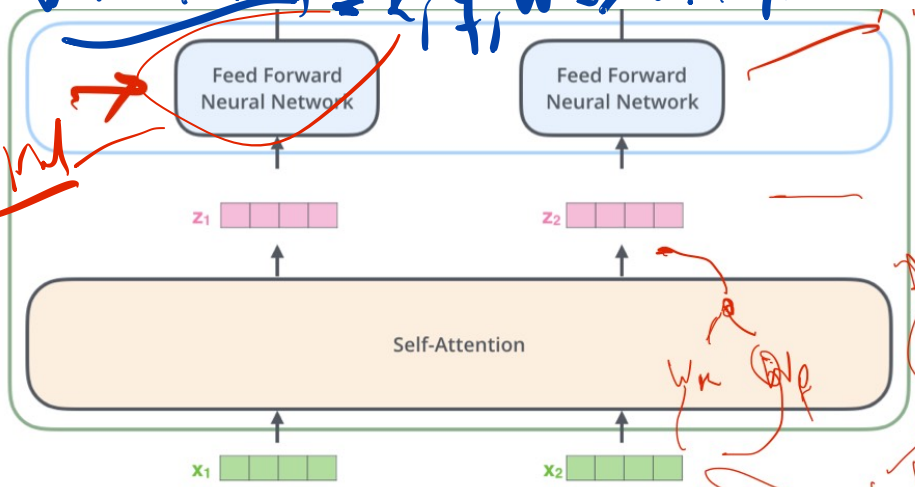
matrix corresponds to a word in the input sentence. We again see the difference in size of the embedding boxes in the figure, and the $q/k/v$ vectors (64, or 3 boxes in the figure)

re dealing with matrices, we can condense steps two through six in one formula to i -attention layer.

$$\text{softmax} \left(\frac{Q \times K^T}{\sqrt{d_k}} \right) \cdot V = Z$$

The self-attention calculation in matrix form

ENCODER #1



1-2nd

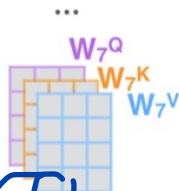
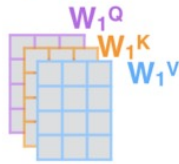
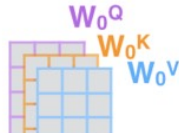
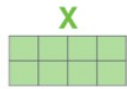
$w_k \otimes p$

multithead

That's pretty much all there is to multi-headed self-attention. It's quite a handful of matrices, I realize. Let me try to put them all in one visual so we can look at them in one place

- 1) This is our input sentence*
- 2) We embed each word*
- 3) Split into 8 heads. We multiply X or R with weight matrices
- 4) Calculate attention using the resulting Q/K/V matrices
- 5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

Thinking Machines



* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



From select objects o_i of o_1, o_2, \dots, o_n to 'select' \rightarrow Embeddings \rightarrow $R(i,j)$ \rightarrow Binary \rightarrow Multiple heads \rightarrow Concatenation

Multiple (8) heads

Extract

CONCAT $Z_0 - Z_7$ \rightarrow Linear W^O

$$\hat{h}_i^{l+1} = O_h^l \parallel \left(\sum_{k=1}^H \left(\sum_{j \in S} w_{ij}^{k,l} V^{k,l} h_j^l \right) \right)$$

where, $w_{ij}^{k,l} = \text{softmax}_j \left(\frac{Q^{k,l} h_i^l \cdot K^{k,l} h_j^l}{\sqrt{d_k}} \right)$

if don't know R \rightarrow use Q, K

Trans - \rightarrow GNN

Concat $A^R V$ \rightarrow pass MLP

Matrices V, K

represent in variety