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)

$f_0(o) \sim \mathbf{0}^{k/2} \mathcal{N}(0,1)^{k/2}$ for each object $o \in s$
for $i \in \{0, \ldots, L-1\}$ **do**
  for *each atom* $q := p(o_1, \ldots, o_m)$ *true in* s **do**
    // Msgs $q \to o$ for each $o = o_j$ in q
    $m_{q,o} := [\mathbf{MLP}_p(f_i(o_1), \ldots, f_i(o_m))]_j$;
  for *each* o *in* s **do**
    // Aggregate, update embeddings
    $f_{i+1}(o) := \mathbf{MLP}_U\left(f_i(o), agg(\{\!\{m_{q,o} | o \in q\}\!\})\right)$;
// Final Readout
$V := \mathbf{MLP}_2\left(\sum_{o \in s} \mathbf{MLP}_1(f_L(o))\right)$

Relational Morris:

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

$$\boldsymbol{h}_v^{(t)} := \sigma\Big(\boldsymbol{h}_v^{(t-1)}\boldsymbol{W}_0^{(t)} + \sum_{w \in N(v)} \boldsymbol{h}_w^{(t-1)}\boldsymbol{W}_1^{(t)}\Big) \in \mathbb{R}^e, \tag{1}$$

for v in $V(G)$, where $\boldsymbol{W}_0^{(t)}$ and $\boldsymbol{W}_1^{(t)}$ 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

$$\boldsymbol{h}_v^{(t)} := \mathsf{UPD}^{(t)}\Big(\boldsymbol{h}_v^{(t-1)}, \mathsf{AGG}^{(t)}\big(\{\!\!\{\boldsymbol{h}_w^{(t-1)} \mid w \in N(v)\}\!\!\}\big)\Big),$$

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

$$\boldsymbol{h}_G := \mathsf{READOUT}\big(\{\!\!\{\boldsymbol{h}_v^{(T)} \mid v \in V(G)\}\!\!\}\big),$$

to compute a single vectorial representation based on learned vertex features after iteration T. Again, $\mathsf{READOUT}$ 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

$$\boldsymbol{h}_{v,\mathsf{R\text{-}GCN}}^{(t)} := \sigma\Big(\boldsymbol{h}_{v,\mathsf{R\text{-}GCN}}^{(t-1)}\boldsymbol{W}_0^{(t)} + \sum_{i \in [r]}\sum_{w \in N_i(v)} \boldsymbol{h}_{w,\mathsf{R\text{-}GCN}}^{(t-1)}\boldsymbol{W}_i^{(t)}\Big) \in \mathbb{R}^e, \tag{2}$$

$$Q = HW_Q, \quad K = HW_K, \quad V = HW_V,$$
$$A = \frac{QK^\top}{\sqrt{d_K}}, \quad \text{Attn}(H) = \text{softmax}(A)\,V,$$

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)

$f_0(o) \sim \mathbf{0}^{k/2}\mathcal{N}(0,1)^{k/2}$ for each object $o \in s$;
for $i \in \{0, \ldots, L-1\}$ **do**
  for *each atom* $q := p(o_1, \ldots, o_m)$ *true in* s **do**
    // Msgs $q \to o$ for each $o = o_j$ in q
    $m_{q,o} := [\mathbf{MLP}_p(f_i(o_1), \ldots, f_i(o_m))]_j$;
  for *each* o *in* s **do**
    // Aggregate, update embeddings
    $f_{i+1}(o) := \mathbf{MLP}_U\left(f_i(o), agg(\{\!\{m_{q,o} | o \in q\}\!\})\right)$;
// Final Readout
$V := \mathbf{MLP}_2\left(\sum_{o \in s} \mathbf{MLP}_1(f_L(o))\right)$

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

), where $W_0^{(t)}$ and $W_1^{(t)}$ are parameter matrices from $\mathbb{R}^{d \times e}$ and σ denote
nction, e.g., a sigmoid or a ReLU function.[4] Following Gilmer et al. (201
, in each layer, $t > 0$, we can generalize the above by computing a verte

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

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 featu

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

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

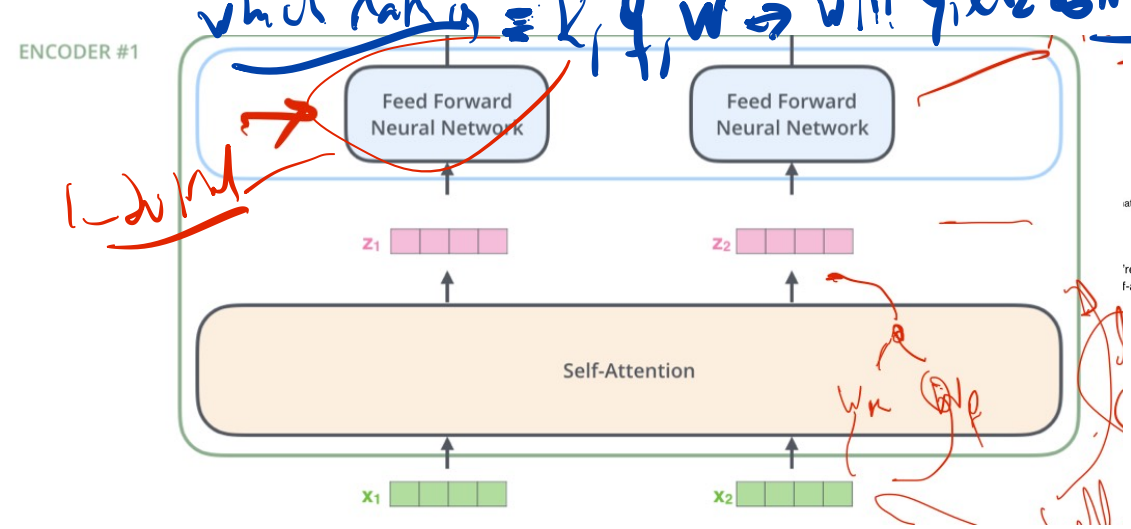
CompGCN. Let G be a labeled multi-relational graph. A CompGCN layer generalizes Equation
by encoding relational information as edge features. That is, we compute a vertex feature

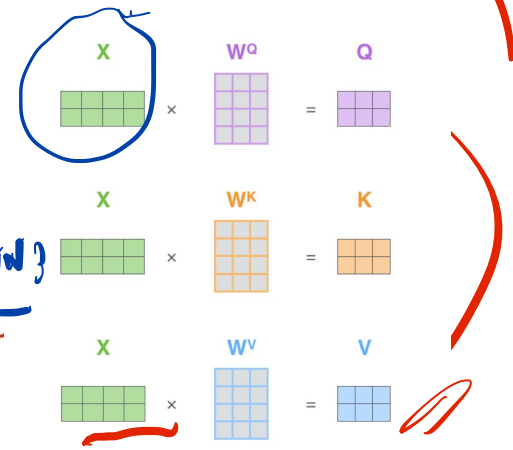
$$h_{v,\text{CompGCN}}^{(t)} := \sigma\left(h_{v,\text{CompGCN}}^{(t-1)} W_0^{(t)} + \sum_{i \in [r]} \sum_{w \in N_i(v)} \phi(h_{w,\text{CompGCN}}^{(t-1)}, z_i^{(t)}) W_1^{(t)}\right) \in \mathbb{R}^e,$$

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)

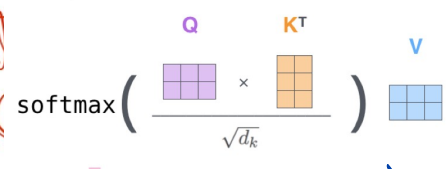
$f_0(o) \sim \mathbf{0}^{k/2}\mathcal{N}(0,1)^{k/2}$ for each object $o \in s$
for $i \in \{0, \ldots, L-1\}$ **do**
  for *each atom* $q := p(o_1, \ldots, o_m)$ *true in* s **do**
    // Msgs $q \to o$ for each $o = o_j$ in q
    $m_{q,o} := [\mathbf{MLP}_p(f_i(o_1), \ldots, f_i(o_m))]_j$;
  for *each* o *in* s **do**
    // Aggregate, update embeddings
    $f_{i+1}(o) := \mathbf{MLP}_U\left(f_i(o), agg(\{\!\{m_{q,o} | o \in q\}\!\})\right)$;
// Final Readout
$V := \mathbf{MLP}_2\left(\sum_{o \in s} \mathbf{MLP}_1(f_L(o))\right)$





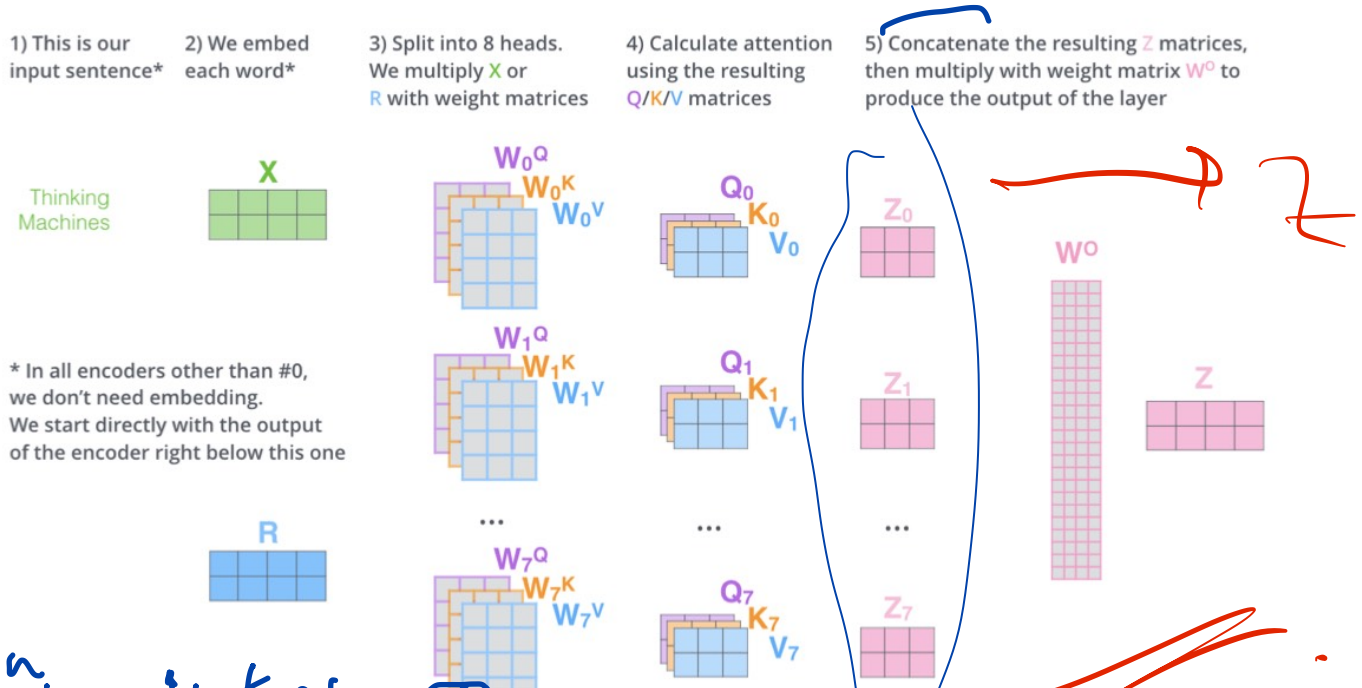
atrix corresponds to a word in the input sentence. We again see the difference in size of the embedc
boxes in the figure), and the q/k/v vectors (64, or 3 boxes in the figure)

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



The self-attention calculation in matrix form

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



$$\hat{h}_i^{\ell+1} = O_h^{\ell} \Big\Vert_{k=1}^{H} \Big(\sum_{j \in \mathcal{S}} w_{ij}^{k,\ell} V^{k,\ell} h_j^{\ell} \Big),$$

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