- So output another graph w/ updated edge & node info!

↓

again, stackable!

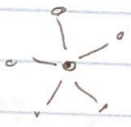
graph → MP layer1 → MP layer2 → ...

before final layer, must involve perm inv't aggregation
e.g. sum or max pool.
e.g. $\sum_{\text{nodes}}$

→ MLP → out

- example: dynamic graph CNN (1801.07829)

kNN edges → new edges based on kNN of learned features



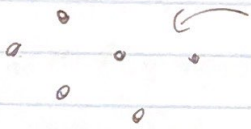
$$\begin{cases} e'_{ab} = \phi^{edge}(\widetilde{a,b}^{kNN}) \\ \phi a' = \square_{\text{max, sum, ...}} e'_{ac} \end{cases} \rightarrow \text{new edges}$$

- Particle Net based on DGCNN won top tagging competition ca 2019! (10% better R30 than ResNext)
- Graph nets — very flexible & powerful framework!
- Downside: compute scales w/ N_{nodes}^2 for DGCNN — kNN is expensive ($N_{nodes} = N_{particles}$ @ LHC ...)

Another application of GNNs

good ref: Batzner et al Nature Comm. 13 (2022)

Regression for "inter-atomic potential" in materials science



← atoms positions $\vec{r}_i$, atomic # Z_i

→ wat total pot'l energy ϵ_{pot} & forces acty on atoms $\vec{F}_i$

good approx (?)

$$\epsilon_{pot} = \sum_i \epsilon_{i,\text{atomic}}$$ sum of individual local atomic energies

$$\vec{F}_i = -\nabla_i \epsilon_{pot}$$

can be calculated using Density Functional Theory but very computationally expensive.

A lot of data to play with: Materials Project website

A growing literature on regressing DFT w/ NNs → GNNs especially well-suited!

system already in the form of a graph!

- node attributes r_i, Z_i
- edge attributes r_{ij}

(+ rotational & translational symmetry)

MP-NN → final graph output gives $\epsilon_{i,\text{atomic}}$ at each node → ϵ_{pot} from final sum over nodes

Fully differentiable! → forces

One more architecture: Transformers (Vaswani et al 2017)

- Used in state of the art NLP like ChatGPT
- in simplest form is also permutation equivariant but well suited to sequences
- like an MLP w/ input-dependent weights $x \to W(x) \cdot x$

  so much more powerful in principle!

- Key idea: self-attention mechanism
- Key idea: learning embeddings

transformers also ↑ a type of GNN!

learns relations btw constituents of a data instance

NLP example

"The animal didn't cross the street because it was tired"

attention → it is associated w/ "the animal" and not "the street"

Previous approaches to NLP were more sequence oriented
— trouble learning relations btw distant words
"long term correlations"

Transformers, being inherently orderless, work much better!

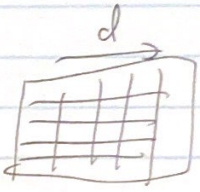
Self Attention Bahdanau (2014)

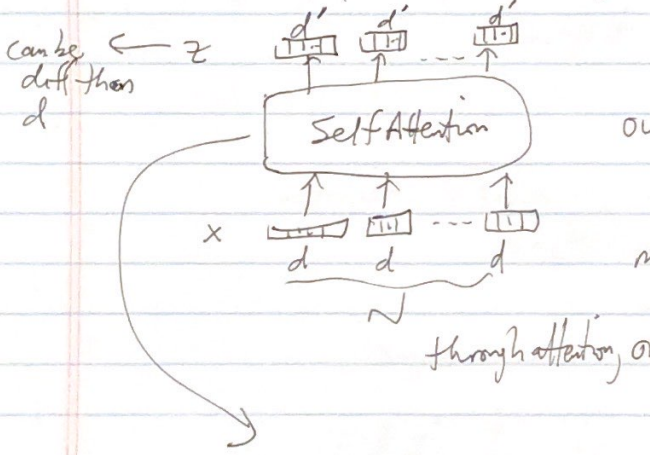
First step → embed words (constituents) into fixed size vector d

↓

could be a learned MLP

So each event / sentence / etc w/ N constituents → (N, d)





output are the transformed embeddings

↓

note: it is an all-all process!

through attention, output embeddy encodes what other constituents were relevant

Learn 3 matrices W^Q, W^K, W^V "query, key, value"

$d \times d'$

↳ same for every constituent of every event!

$$\begin{cases} Q = X W^Q & (N, d') \\ K = X W^K & (N, d') \\ V = X W^V & (N, d') \end{cases}$$

comes from database literature (k_i, v_i) and q

QK^T is $N \times N$

↓

softmax(QK^T) across each row → $N \times N$

$$\begin{pmatrix} p_1^{(1)} & p_2^{(1)} & \cdots & p_N^{(1)} \\ \vdots & & & \\ p_1^{(N)} & \cdots & & p_N^{(N)} \end{pmatrix}$$

each row sums to 1

$softmax(QK^T) \cdot V$

↳ weighted sum of value rows

$$\begin{pmatrix} \vec{p}^{(1)} \\ \vdots \\ \vec{p}^{(N)} \end{pmatrix} \begin{pmatrix} \vec{v}_1 \\ \vdots \\ \vec{v}_N \end{pmatrix}$$

$$\downarrow$$

$$\begin{pmatrix} \sum p_i^{(1)} \vec{v}_i \\ \vdots \\ \sum p_i^{(N)} \vec{v}_i \end{pmatrix}$$ output of self-attention block.

- Intuition: softmax output is where attention happens
  weights irrelevant pairs → 0
  relevant pairs → O(1).

Example: the "it" word: element 8 in sentence w/ N=10

Might get: $\vec{p}^{(8)} = (0.5, 0.5, 0, \cdots 0)$

But "it" also relates to "tired" — what about that?

→ "Multiheaded attention" analogous to multiple filters in CNN

Learn multiple $W^{Q,K,V}_{a=1,\cdots N_h}$

each gives transformed embedding (N, d') Z_a

concatenate $(N, d' \cdot N_h)$ $(Z_1 Z_2 \cdots Z_{N_h}) \downarrow N \equiv Z$

Learn final matrix W^o $(d' \cdot N_h, d'')$

$$\boxed{Z_{output} = Z W^o}$$