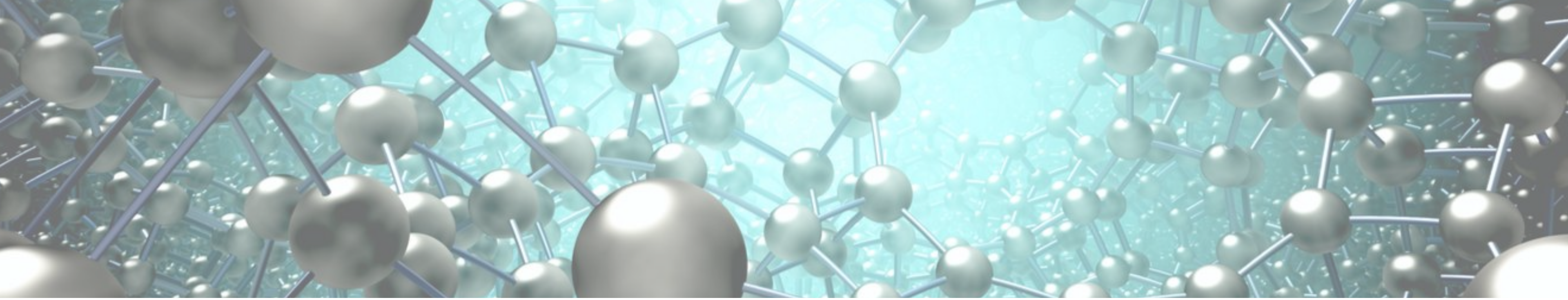


Neural Message Passing for Quantum Chemistry

ICML 2017 - Sydney

Gilmer, Schoenholz, Riley, Vinyals, Dahl

Motivation

- Applications of quantum chemistry:

  ➡ Drug design and discovery

  ➡ Toxicity analysis

  ➡ Material science

  ➡ Understanding chemical bonding

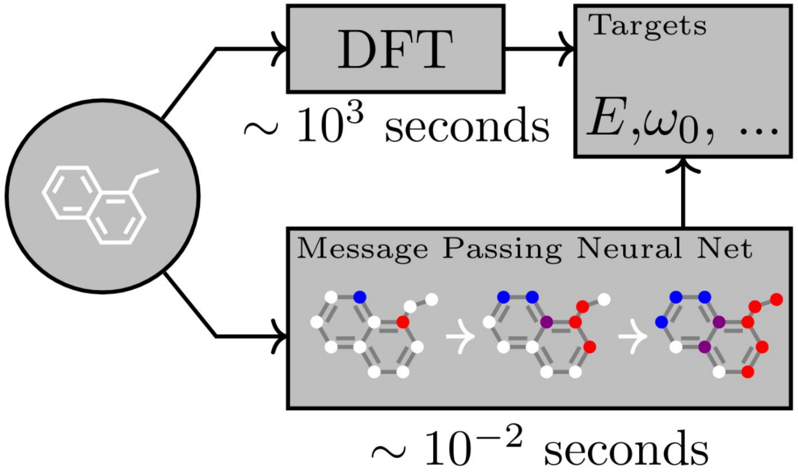
  ➡ Nanotechnology and biotechnology

  ➡ ...

- Fundamental limitation: huge information space to explore.
- Fundamental advantage: huge data sets available.

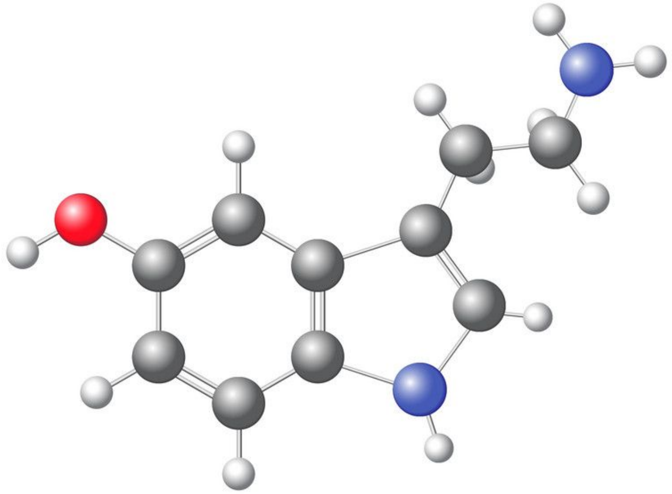
Background: density functional theory (DFT)

- **Density functional theory** (DFT) is one of the most important computational method used to predict the electronic structure and properties of atoms, molecules, and solids. It is used to study the electronic structure of many-body quantum systems.
- Still too **slow**: ~1-10h for a 20 atom molecule.
- This work approximates DFT results but its inference time is up to 300,000x faster!



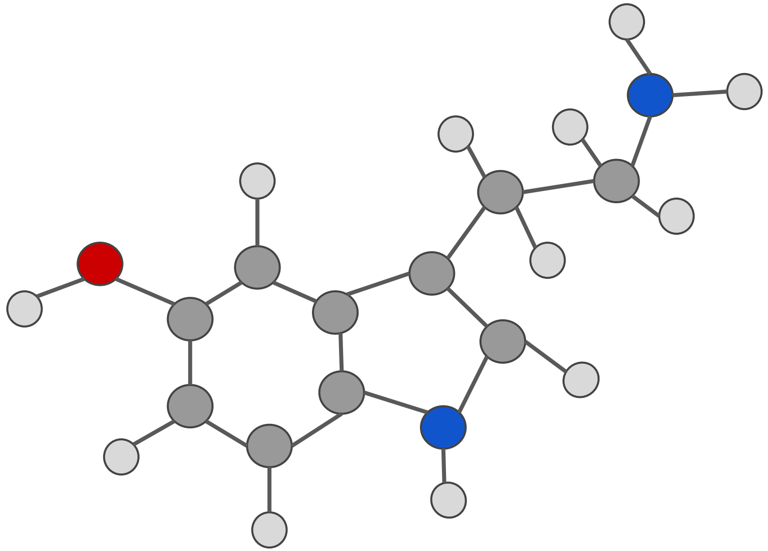
Background: molecules as graphs

molecule



serotonin

graph



atoms → nodes
bonds + distance → edges

(graph is actually fully connected
because of spatial distance information)

Background: previous work on ML on graphs

- Two main branches of work on using machine learning on graphs:

  ➡ **feature engineering**: Coulomb matrix, bag of bonds...

  ➡ **neural networks**: relational networks, spectral networks, graph convolutional neural networks... These are invariant to graph isomorphism.

- This work aims to unify the different models of neural networks for graphs into one framework: **message passing neural networks** (MPNNs)
- This work **improves** on previous MPNNs algorithms by adding novel variations to this family of NNs.

Model: message passing neural network (MPNN)

- Consider an undirected graph G with node features x_v and edge features e_{vw} . This graph has an associated adjacency matrix *A*.
- We aim to obtain some graph-level target.
- Computation is done in two phases.
  (1) Message passing phase.
  (2) Readout phase.
- The model requires learning three differentiable functions: message function M_t , vertex update functions U_t, and readout function *R*.

(1) Message passing phase

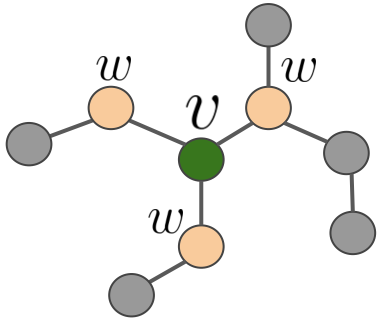
- Hidden states at each vertex are updated over *T* steps, indexed by *t*.
- This phase requires a:

  ➡ Message passing function M_t .

  ➡ Update function U_t .
- At each time-step *t* → *t+1* the hidden state on vector v is updated as follows:

(a) $$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

(b) $$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$



(2) Readout phase

- The readout phase computes a feature vector *y* for the whole graph using some readout function *R*, such that:

$$\hat{y} = R(\{h_v^T \mid v \in G\})$$

- The function *R* must be invariant to permutation of the node states.

Model: previous MPNN work

- We can build different MPNNs by changing how we define the learnable functions M_t , U_t and *R*.
- Previous main work include:
  - ➡ Convolutional Networks for Learning Molecular Fingerprints, Duvenaud et al. (2015)
  - ➡ Gated Graph Neural Networks (GG-NN), Li et al. (2016)
  - ➡ Interaction Networks, Battaglia et al. (2016)
  - ➡ Molecular Graph Convolutions, Kearnes et al. (2016)
  - ➡ Deep Tensor Neural Networks, Schütt et al. (2017)

Model: novelties of this work

🟩 : novel

- Develop novelties around the GG-NN (Li et al.).
- Functions:

(a) $M_t = A(e_{vw})h_w$ allows for vector valued edge features!

(b) $U_t = GRU(h_v^t, m_v^{t+1})$ same as in GG-NN

(c) R = *set2set* model defined by Vinyals et al (2015)

- Adding **virtual edges** to link nodes that are not initially connected. Can even create a latent **master node** (connected to every input node).
- More efficient propagation protocol: break-up node embeddings.

Dataset and Task

- The dataset used in this problem is the **QM9** [1] which contains 130,000 organic molecules with at most 30 atoms of the elements H, C, O, N, F.
- There are 13 tasks that we will complete to compare performance with DFT.

  Tasks include:

  ➡ Find the energy of the highest occupied molecular orbital.

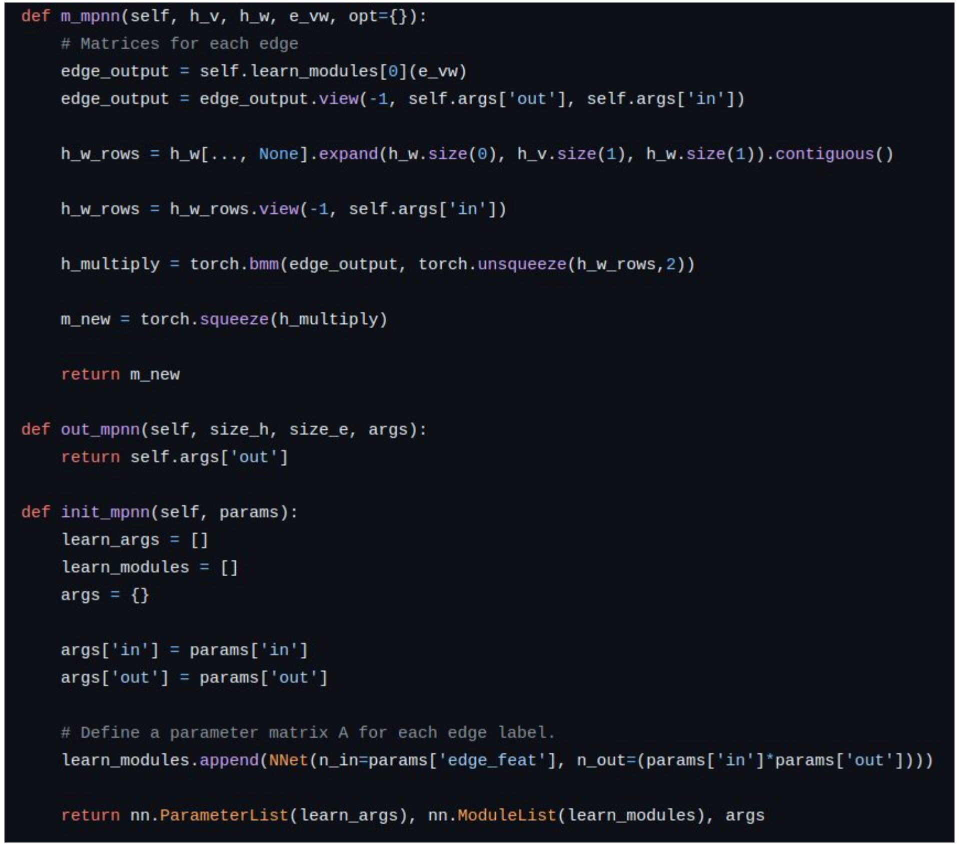
  ➡ Heat capacity of the molecule

  ➡ ...

[1] R. Ramakrishnan, P. O. Dral, M. Rupp, O. A. von Lilienfeld, *Quantum Chemistry Structures and Properties of 134 kilo Molecules*, Scientific Data 1, 140022, 2014.

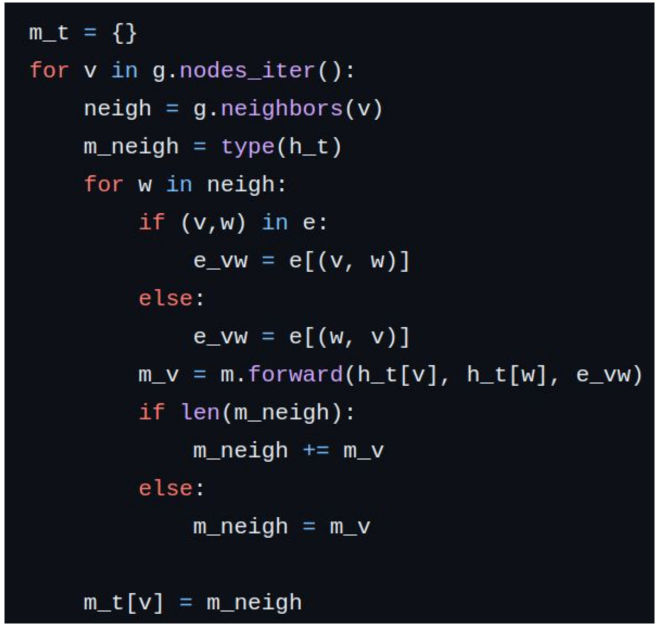
Code

Define message passing function:



```python
def m_mpnn(self, h_v, h_w, e_vw, opt={}):
    # Matrices for each edge
    edge_output = self.learn_modules[0](e_vw)
    edge_output = edge_output.view(-1, self.args['out'], self.args['in'])

    h_w_rows = h_w[..., None].expand(h_w.size(0), h_v.size(1), h_w.size(1)).contiguous()

    h_w_rows = h_w_rows.view(-1, self.args['in'])

    h_multiply = torch.bmm(edge_output, torch.unsqueeze(h_w_rows,2))

    m_new = torch.squeeze(h_multiply)

    return m_new

def out_mpnn(self, size_h, size_e, args):
    return self.args['out']

def init_mpnn(self, params):
    learn_args = []
    learn_modules = []
    args = {}

    args['in'] = params['in']
    args['out'] = params['out']

    # Define a parameter matrix A for each edge label.
    learn_modules.append(NNet(n_in=params['edge_feat'], n_out=(params['in']*params['out'])))

    return nn.ParameterList(learn_args), nn.ModuleList(learn_modules), args
```

Perform message passing:



```python
m_t = {}
for v in g.nodes_iter():
    neigh = g.neighbors(v)
    m_neigh = type(h_t)
    for w in neigh:
        if (v,w) in e:
            e_vw = e[(v, w)]
        else:
            e_vw = e[(w, v)]
        m_v = m.forward(h_t[v], h_t[w], e_vw)
        if len(m_neigh):
            m_neigh += m_v
        else:
            m_neigh = m_v

    m_t[v] = m_neigh
```

Code

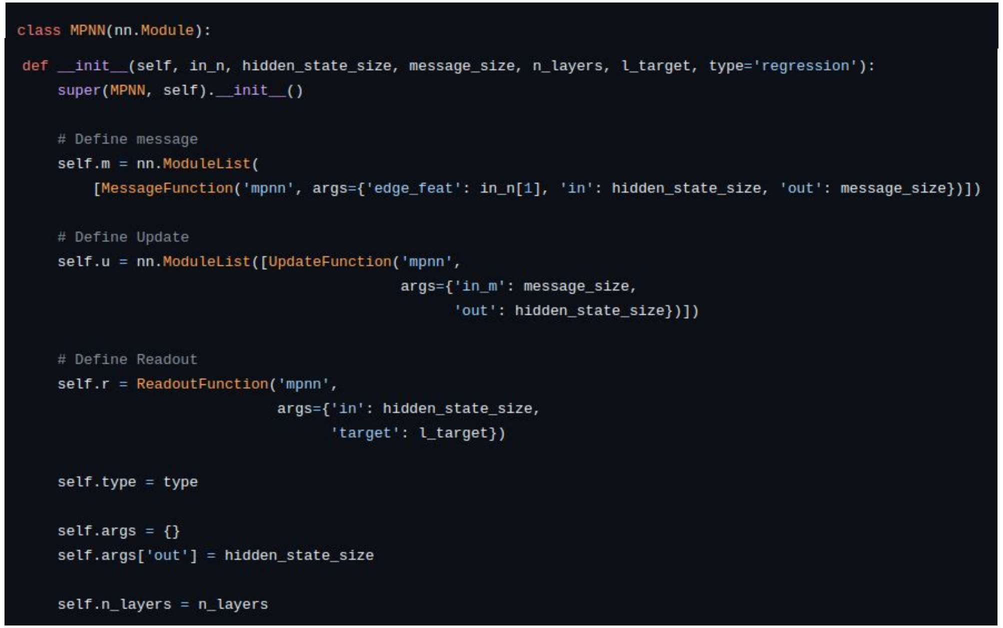
Define update function:

```
def u_ggnn(self, h_v, m_v, opt={}):
    h_v.contiguous()
    m_v.contiguous()
    h_new = self.learn_modules[0](torch.transpose(m_v, 0, 1), torch.unsqueeze(h_v, 0))[0]
    return torch.transpose(h_new, 0, 1)

def init_ggnn(self, params):
    learn_args = []
    learn_modules = []
    args = {}

    args['in_m'] = params['in_m']
    args['out'] = params['out']

    # GRU
    learn_modules.append(nn.GRU(params['in_m'], params['out']))

    return nn.ParameterList(learn_args), nn.ModuleList(learn_modules), args
```

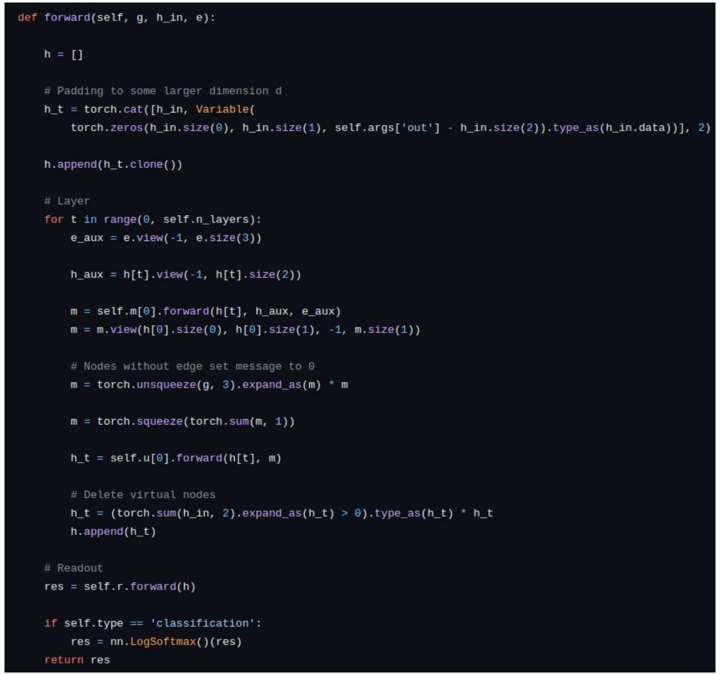
Code

Take readout function from *set2set* (Vinyals et al)

Define final MPNN:



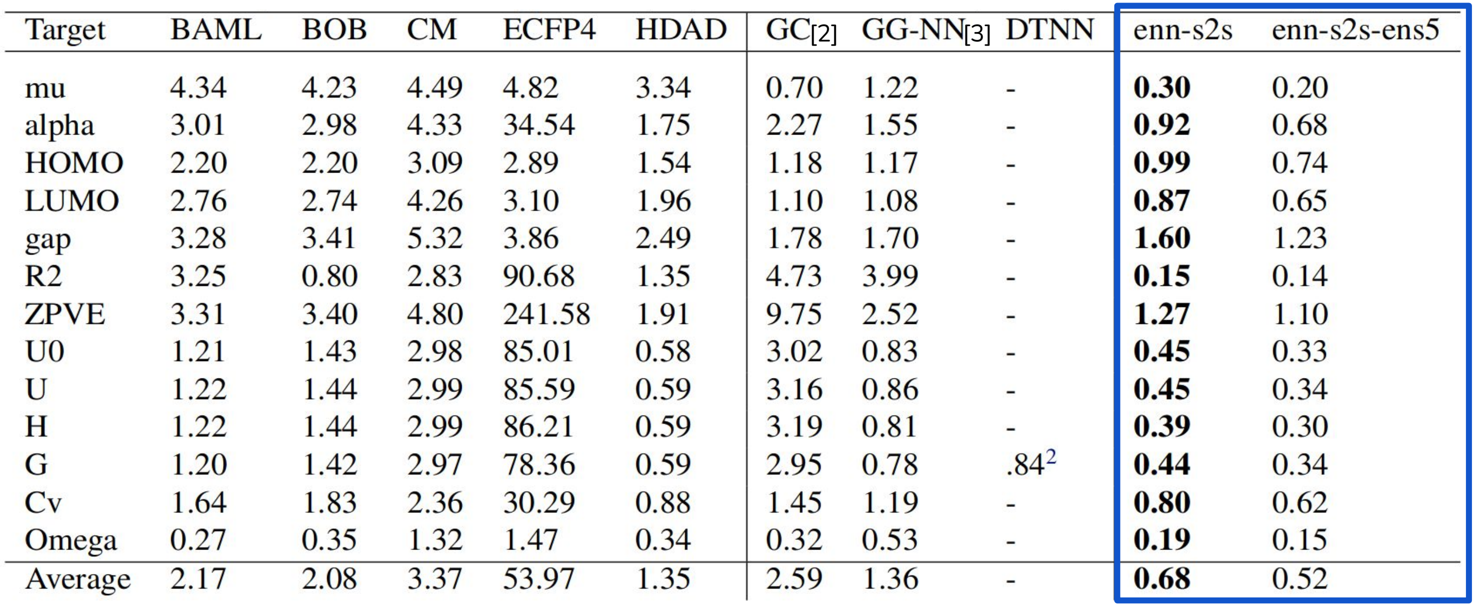
```
class MPNN(nn.Module):

 def __init__(self, in_n, hidden_state_size, message_size, n_layers, l_target, type='regression'):
     super(MPNN, self).__init__()

     # Define message
     self.m = nn.ModuleList(
         [MessageFunction('mpnn', args={'edge_feat': in_n[1], 'in': hidden_state_size, 'out': message_size})])

     # Define Update
     self.u = nn.ModuleList([UpdateFunction('mpnn',
                                            args={'in_m': message_size,
                                                  'out': hidden_state_size})])

     # Define Readout
     self.r = ReadoutFunction('mpnn',
                              args={'in': hidden_state_size,
                                    'target': l_target})

     self.type = type

     self.args = {}
     self.args['out'] = hidden_state_size

     self.n_layers = n_layers
```



```
def forward(self, g, h_in, e):

    h = []

    # Padding to some larger dimension d
    h_t = torch.cat([h_in, Variable(
        torch.zeros(h_in.size(0), h_in.size(1), self.args['out'] - h_in.size(2)).type_as(h_in.data))], 2)

    h.append(h_t.clone())

    # Layer
    for t in range(0, self.n_layers):
        e_aux = e.view(-1, e.size(3))

        h_aux = h[t].view(-1, h[t].size(2))

        m = self.m[0].forward(h[t], h_aux, e_aux)
        m = m.view(h[0].size(0), h[0].size(1), -1, m.size(1))

        # Nodes without edge set message to 0
        m = torch.unsqueeze(g, 3).expand_as(m) * m

        m = torch.squeeze(torch.sum(m, 1))

        h_t = self.u[0].forward(h[t], m)

        # Delete virtual nodes
        h_t = (torch.sum(h_in, 2).expand_as(h_t) > 0).type_as(h_t) * h_t
        h.append(h_t)

    # Readout
    res = self.r.forward(h)

    if self.type == 'classification':
        res = nn.LogSoftmax()(res)
    return res
```

Results

- This work obtains **state-of-the-art** predictions of DFT on all 13 tasks.
- Metric is proportional to mean-squared error (the smaller the better)



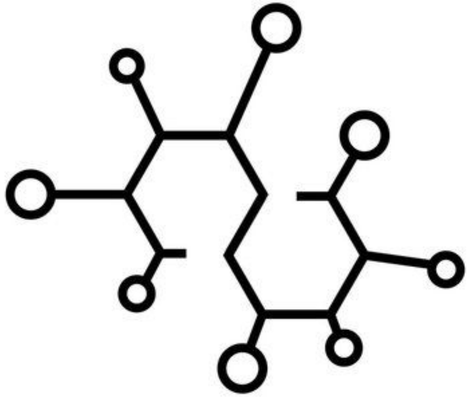
	feature engineering + off-the-shelf classifiers					previous MPNN models			this work	
Target	BAML	BOB	CM	ECFP4	HDAD	GC[2]	GG-NN[3]	DTNN	enn-s2s	enn-s2s-ens5
mu	4.34	4.23	4.49	4.82	3.34	0.70	1.22	-	**0.30**	0.20
alpha	3.01	2.98	4.33	34.54	1.75	2.27	1.55	-	**0.92**	0.68
HOMO	2.20	2.20	3.09	2.89	1.54	1.18	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

[2] Kearneset al. *Molecular Graph Convolutions: Moving Beyond Fingerprints*. Journal of Computer-Aided Molecular Design, 2016.
[3] Li et al. *Gated Graph Sequence Neural Networks*. ICLR, 2016.

Future work

- Train on larger molecular spaces.
- Add attention mechanism to the incoming message vectors.
- Material science uses infinite periodic graphs.

molecule



molecool