Simple GNN Regularisation for 3D Molecular Property Prediction & Beyond

Authors

Published as a conference paper at ICLR 2022

SIMPLE GNN REGULARISATION FOR 3D MOLECULAR PROPERTY PREDICTION & BEYOND

Jonathan Godwin, Michael Schaarschmidt, Alexander Gaunt, Alvaro Sanchez-Gonzales, Yulia Rubanova, Petar Veličković, James Kirkpatrick & Peter Battaglia
DeepMind, London
{jonathangodwin}@deepmind.com

Overview

- **Main goal**: Address the over-smoothing problem
- **Technical route**: Using regularisers to penalise node latent similarity and encourage meaningful node representations
- **Proposed method: Noisy Nodes**
  - Corrupt the input graph with noise
  - Add a noise correcting node-level loss
  - 1. Encourage latent node diversity
    2. Denoising objective encourages to learn meaningful graph representations

Background

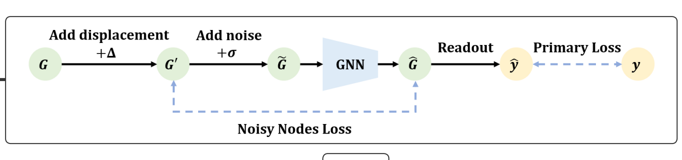
- **What is over-smoothing problem?**: When the node latent vectors of a GNN become very similar after successive layers of message passing
- **What cause the over-smoothing problem?**: The averaging effect of the summation forces the nodes to become almost identical.
- **What will over-smoothing cause?**
  - No relational information contained in the nodes
  - No higher-order latent graph representations can be learned
- **Previous methods**
  - Residual connections
  - Multi-head attention

Notation

- $G = (V, E, g)$ — Input graph
- $V = \{v_1, \dots, v_{|V|}\}$, where $v_i \in \mathbb{R}^{d_v}$ — Node
- $E = \{e_1, \dots, e_{|E|}\}$ — Directed edge
  - $e_k = (s_k, r_k, e_k)$ — sender, recevier
    - $s_k, r_k \in \{1, \dots, |V|\}$
    - $e_k \in \mathbb{R}^{d_e}$
- $g \in \mathbb{R}^{d_g}$ — Graph-level property

Methodology

- Noisy Nodes mechanics during training



- Corrupts nodes by noise
  - $\tilde{G} = (\tilde{V}, \tilde{E}, \tilde{g})$ — Only the node-level feature will be corrupted by noise
    - σ_i — Noise
    - $\tilde{v}_i = v_i + \sigma_i$
    - $\tilde{E} = E,\ \tilde{g} = g$

- Noisy Nodes Training Step

Algorithm 1: Noisy Nodes Training Step

$G = (V, E, g)$ // Input graph
$\tilde{G} = G$ // Initialize noisy graph
λ // Noisy Nodes Weight
if *not_provided*(V') **then**
  $V' \leftarrow V$
end
if *predict_differences* **then**
  $\Delta = \{v_i' - v_i | i \in 1, \dots, |V|\}$
end
for each $i \in 1, \dots, |V|$ **do**
  σ_i = sample_node_noise(shape_of(v_i));
  $\tilde{v}_i = v_i + \sigma_i$;
  if *predict_differences* **then**
    $\tilde{\Delta}_i = \Delta_i - \sigma_i$;
  end
endfor
$\tilde{E}$ = recompute_edges($\tilde{V}$);
$\hat{G}'$ = GNN($\tilde{G}$);
if *predict_differences* **then**
  $V' = \tilde{\Delta}_i$;
end
Loss = λ NoisyNodesLoss($\hat{G}'$, V') + PrimaryLoss($\hat{G}'$, V'));
Loss.minimise()

- GNS adaptations for molecular property prediction

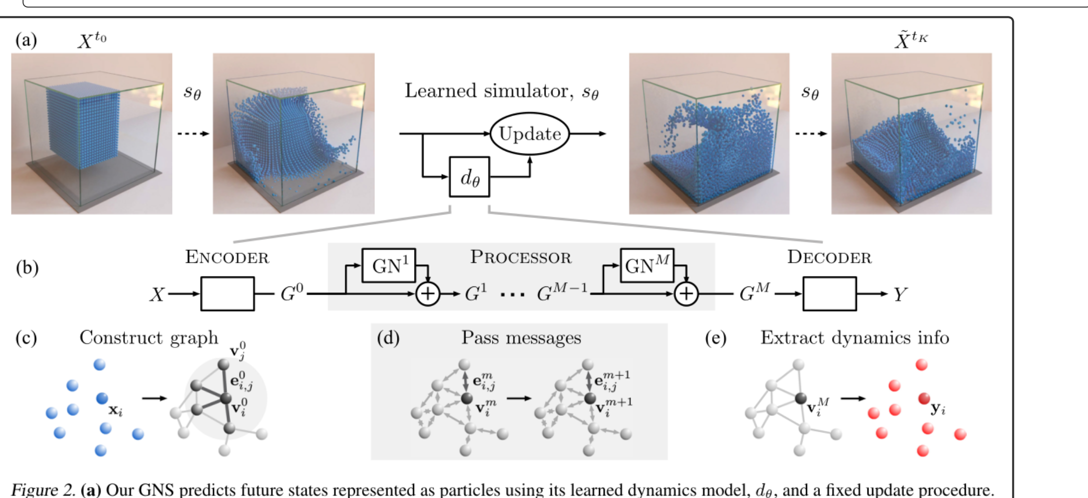


Figure 2. **(a)** Our GNS predicts future states represented as particles using its learned dynamics model, d_θ, and a fixed update procedure. **(b)** The d_θ uses an "encode-process-decode" scheme, which computes dynamics information, Y, from input state, X. **(c)** The ENCODER constructs latent graph, G^0, from the input state, X. **(d)** The PROCESSOR performs M rounds of learned message-passing over the latent graphs, $G^0, \dots, G^M$. **(e)** The DECODER extracts dynamics information, Y, from the final latent graph, G^M.

ICML'20-Learning to simulate complex physics with graph networks

- **Original Encoder**
  - $\mathcal{X} \rightarrow \mathcal{G}$ — **Objective:** Learn a graph from give X
  - $\mathbf{v}_i = \varepsilon^v(\mathbf{x}_i)$ — E^v is a **2-layers MLP** that is independently applied to each node
  - $\mathbf{e}_{i,j} = \varepsilon^e(\mathbf{r}_{i,j})$ — E^e is a **2-layers MLP** that is independently applied to each edge
    - $r_{i,j}$ can be displacement between the positions of node i and node j
- **Adapted Encoder**
  - $\mathcal{X} \rightarrow \mathcal{G}$ — **Objective:** Learn a graph from give X
  - $\mathbf{v}_i = \varepsilon^v(\mathbf{x}_i)$ — E^v is a **2-layers MLP** that is independently applied to each node
  - $e_k = \text{Concat}(\tilde{e}_{RBF,1}(|d|), ..., \tilde{e}_{RBF,c}(|d|), \frac{d}{|d|})$ — d is the edge vector displacements
    - $\tilde{e}_{RBF,c} = \sqrt{\frac{2}{R}} \frac{\sin(\frac{c\pi}{R} d)}{d}$
- **Processor**
  - $\mathbf{G} = (G^1, ..., G^M)$ — **Objective:** Generate a update latent graph
  - $G^{m+1} = \text{GN}^{m+1}(G^m)$ — GN is a **7 layers MPNN**
- **Original Decoder**
  - $\mathcal{G} \rightarrow \mathcal{Y}$ — **Objective:** mapping graph to the target
  - $\mathbf{y}_i = \delta^v(\mathbf{v}_i^M)$ — δ^v is a **2-layers MLP** that is independently applied to each node
- **Adapted Decoder**
  - $\mathcal{G} \rightarrow \mathcal{Y}$ — **Objective:** mapping graph to the target
  - $$y = W^{\text{Proc}} \sum_{i=1}^{|V|} \text{MLP}_{\text{Proc}}(a_i^{\text{Proc}}) + b^{\text{Proc}} + W^{\text{Enc}} \sum_{i=1}^{|V|} \text{MLP}_{\text{Enc}}(a_i^{\text{Enc}}) + b^{\text{Enc}}$$
    a_i^{Proc} denotes node latents from the **Processor**
    a_i^{Enc} denotes node latents from the **Encoder**

Experiments

- **Per layer node diversity**

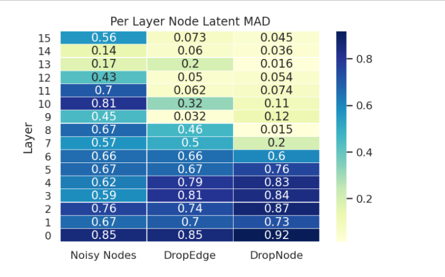


Figure 2: Per layer node latent diversity, measured by MAD on a 16 layer MPNN trained on OGBG-MOLPCBA. Noisy Nodes maintains a higher level of diversity throughout the network than competing methods.

MAD: Mean Absolute Distance
DropEdge (ICLR'20): Randomly drop edge during training
DropNode (Expert Syst. Appl.'21) : Randomly drop node during training

- **3D molecular property prediction task**
  - **OC20**
    - **OC20-Initial Structure to Resulting Energy (IS2RE)**
      - **Setting**: Add I.I.D Gaussian noise with mean zero and σ = 0.3. **The Noisy Node target**: the relaxed structure.

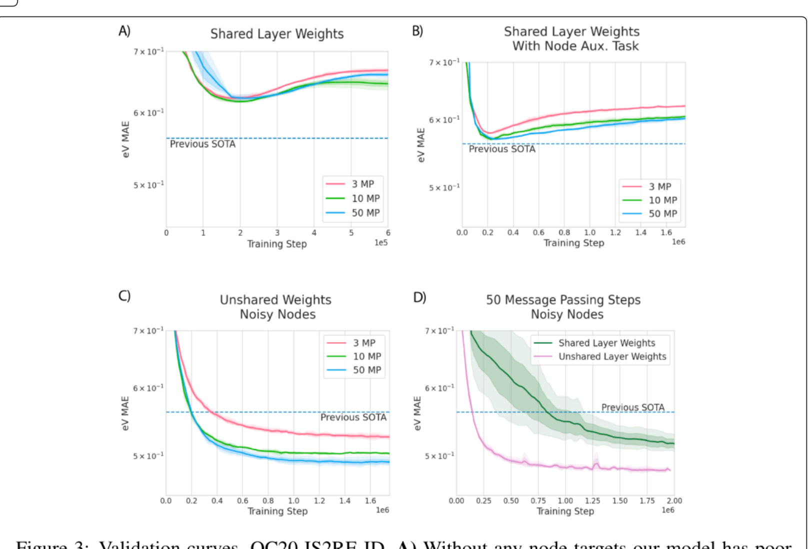


Figure 3: Validation curves, OC20 IS2RE ID. **A)** Without any node targets our model has poor performance and realises no benefit from depth. **B)** After adding a position node loss, performance improves as depth increases. **C)** As we add Noisy Nodes and parameters the model achieves SOTA, even with 3 layers, and stops overfitting. **D)** Adding Noisy Nodes allows a model with even fully shared weights to achieve SOTA.

Table 1: OC20 ISRE Validation, eV MAE, ↓.
"GNS-Shared" indicates shared weights. "GNS-10" indicates a group size of 10.

Model	Layers	OOD Both	OOD Adsorbate	OOD Catalyst	ID
GNS	50	0.59 ±0.01	0.65 ±0.01	0.55 ±0.00	0.54 ±0.00
GNS-Shared + Noisy Nodes	50	0.49 ±0.00	0.54 ±0.00	0.51 ±0.01	0.51 ±0.01
GNS + Noisy Nodes	50	0.48 ±0.00	0.53 ±0.00	0.49 ±0.01	0.48 ±0.00
GNS-10 + Noisy Nodes	100	**0.46**±0.00	**0.51** ±0.00	**0.48** ±0.00	**0.47** ±0.00

OOD: out of the distribution
ID: in domain

Table 2: Results OC20 IS2RE Test

eV MAE ↓	SchNet	DimeNet++	SpinConv	SphereNet	GNS + Noisy Nodes
OOD Both	0.704	0.661	0.674	0.638	**0.465 (-24.0%)**
OOD Adsorbate	0.734	0.725	0.723	0.703	**0.565 (-22.8%)**
OOD Catalyst	0.662	0.576	0.569	0.571	**0.437 (-17.2%)**
ID	0.639	0.562	0.558	0.563	**0.422 (-18.8%)**
Average Energy within Threshold (AEwT) ↑	SchNet	DimeNet++	SpinConv	SphereNet	GNS + Noisy Nodes
OOD Both	0.0221	0.0241	0.0233	0.0241	**0.047 (+95.8%)**
OOD Adsorbate	0.0233	0.0207	0.026	0.0229	**0.035 (+89.5%)**
OOD Catalyst	0.0294	0.0410	0.0382	0.0409	**0.080 (+95.1%)**
ID	0.0296	0.0425	0.0408	0.0447	**0.091 (+102.0%)**

    - **Initial Structure to Resulting Structure (IS2RS)**

Table 3: OC20 IS2RS Validation, ADwT, ↑

Model	Layers	OOD Both	OOD Adsorbate	OOD Catalyst	ID
GNS	50	43.0%±0.0	38.0%±0.0	37.5% 0.0	40.0%±0.0
GNS + Noisy Nodes	50	50.1%±0.0	44.3%±0.0	44.1%±0.0	46.1% ±0.0
GNS-10 + Noisy Nodes	50	52.0%±0.0	46.2%±0.0	46.1% ±0.0	48.3% ±0.0
GNS-10 + Noisy Nodes + Pos only	100	**54.3**%±0.0	**48.3**%±0.0	**48.2**% ±0.0	**50.0**% ±0.0

Table 4: OC20 IS2RS Test, ADwT, ↑

Model	OOD Both	OOD Adsorbate	OOD Catalyst	ID
ForceNet	46.9%	37.7%	43.7%	44.9%
GNS + Noisy Nodes	**52.7%**	**43.9%**	**48.4%**	**50.9%**
Relative Improvement	**+12.4%**	**+16.4%**	**+10.7%**	**+13.3%**

  - **QM9**
    - **Setting:** Add I.I.D Gaussian noise with mean zero and σ = 0.02 to the input atom positions. **The Noisy Node target:** the original atom positions
    - **Results**

Table 6: QM9, Test MAE, Mean & Standard Deviation of 3 Seeds Reported.

Target	Unit	SchNet	E(n)GNN	DimeNet++	SphereNet	PaiNN	**GNS + Noisy Nodes**
μ	D	0.033	0.029	0.030	0.027	**0.012**	0.025 ±0.01
α	${a_0}^3$	0.235	0.071	**0.043**	0.047	0.045	0.052 ±0.00
ϵ_{HOMO}	meV	41	29.0	24.6	23.6	27.6	**20.4** ±0.2
ϵ_{LUMO}	meV	34	25.0	19.5	18.9	20.4	**18.6** ±0.4
$\Delta\epsilon$	meV	63	48.0	32.6	32.3	45.7	**28.6** ±0.1
$\langle R^2 \rangle$	${a_0}^2$	**0.07**	0.11	0.33	0.29	0.07	0.70 ±0.01
ZPVE	meV	1.7	1.55	1.21	**1.12**	1.28	1.16 ±0.01
U_0	meV	14.00	11.00	6.32	6.26	**5.85**	7.30 ±0.12
U	meV	19.00	12.00	6.28	7.33	**5.83**	7.57 ±0.03
H	meV	14.00	12.00	6.53	6.40	**5.98**	7.43±0.06
G	meV	14.00	12.00	7.56	8.0	**7.35**	8.30 ±0.14
c_v	$\frac{\text{cal}}{\text{mol K}}$	0.033	0.031	0.023	**0.022**	0.024	0.025 ±0.00
std. MAE	%	1.76	1.22	0.98	0.94	1.00	**0.88**
logMAE		-5.17	-5.43	-5.67	-5.68	**-5.85**	-5.60

- **Non-spatial task**
  - **OGBG-PCQM4M**
    - **Setting:** Randomly flip node and edge features at a rate of 5%. **The Noisy Node target:** the original node and edge features
    - **Results**

Table 7: OGBG-PCQM4M Results

Model	Number of Layers	Using Noisy Nodes	MAE
MPNN + Virtual Node	16	Yes	0.1249 ± 0.0003
MPNN + Virtual Node	50	No	0.1236 ± 0.0001
Graphormer (Ying et al., 2021)	-	-	0.1234
MPNN + Virtual Node	50	Yes	**0.1218 ± 0.0001**

  - **OGBG-MOLPCBA**
    - **Setting:** Randomly flip node and edge features at a rate of 5%. **The Noisy Node target:** the original node and edge features
    - **Results**

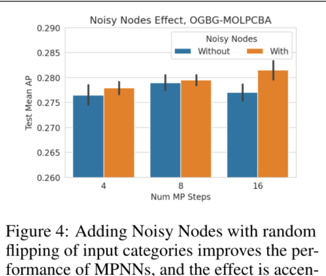


Figure 4: Adding Noisy Nodes with random flipping of input categories improves the performance of MPNNs, and the effect is accentuated with depth.

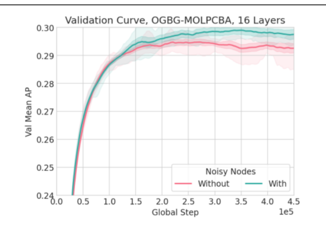


Figure 5: Validation curve comparing with and without noisy nodes. Using Noisy Nodes leads to a consistent improvement.