A. Appendix

A.1. Node Classification

A.1.1 ogbn-products

MLP: perturbation step size $\alpha=2e-02$, only labeled nodes are used in the training phase.

GraphSAGE: labeled perturbation step size $\alpha_l=8e-03$, $\alpha_u/\alpha_l=2$, and neighbor sampling is used for scalable training.

GAT: labeled perturbation step size $\alpha_l=5e-03$, $\alpha_u/\alpha_l=2$, and neighbor sampling is used for scalable training.

DeeperGCN: labeled perturbation step size $\alpha_l=5e-03$, $\alpha_u/\alpha_l=2$, and the model is trained on NVIDIA Tesla V100 (32GB GPU memory).

A.1.2 ogbn-proteins

GCN: labeled perturbation step size $\alpha_l=1e-03$, $\alpha_u/\alpha_l=1$, and the model is trained in the full-batch manner.

GraphSAGE: labeled perturbation step size $\alpha_l=1e-03$, $\alpha_u/\alpha_l=1$, and the model is trained in the full-batch manner.

DeeperGCN: labeled perturbation step size $\alpha_l=8e-03$, $\alpha_u/\alpha_l=1$, and the model is trained on NVIDIA Tesla V100 (32GB GPU memory).

A.1.3 ogbn-arxiv

MLP: perturbation step size $\alpha=2e-03$, only labeled nodes are used in the training phase.

GCN: labeled perturbation step size $\alpha_l=1e-03$, $\alpha_u/\alpha_l=1$, and the model is trained in the full-batch manner.

GraphSAGE: labeled perturbation step size $\alpha_l=1e-03$, $\alpha_u/\alpha_l=1$, and the model is trained in the full-batch manner.

GAT: labeled perturbation step size $\alpha_l=1e-03$, $\alpha_u/\alpha_l=2$, and the model is trained in the full-batch manner.

DeeperGCN: labeled perturbation step size $\alpha_l=8e-03$, $\alpha_u/\alpha_l=1$, and the model is trained on NVIDIA Tesla V100 (32GB GPU memory).

A.1.4 ogbn-mag

R-GCN: labeled perturbation step size $\alpha_l=1e-04$, $\alpha_u/\alpha_l=1$, and the model is trained with neighbor sampling for scalability.

A.2. Link Prediction

A.2.1 ogbl-ddi

GCN perturbation step size $\alpha=3e-03$, and GraphSAGE perturbation step size $\alpha_l=3e-03$. Models are both trained in the full-batch manner. During each gradient ascent loop negative edges are resampled for computing negative losses.

A.2.2 ogbl-collab

GCN perturbation step size $\alpha=3e-03$, and GraphSAGE perturbation step size $\alpha_l=3e-03$. Models are both trained in the full-batch manner. During each gradient ascent loop negative edges are resampled for computing negative losses.

A.3. Graph Classification

A.3.1 ogbg-molhiv

GCN: perturbation step size $\alpha=1e-02$, when virtual node is added we use a smaller $\alpha=1e-03$.

GIN: perturbation step size $\alpha=5e-03$, when virtual node is added we use a smaller $\alpha=1e-03$.

DeeperGCN: perturbation step size $\alpha=1e-02$, and the model is trained on NVIDIA Tesla V100 (32GB GPU memory).

A.3.2 ogbg-molpca

GCN: perturbation step size $\alpha=8e-03$ for both the vanilla model and the one augmented by virtual node.

GIN: perturbation step size $\alpha=8e-03$ for both the vanilla model and the one augmented by virtual node.

DeeperGCN: perturbation step size $\alpha=8e-03$ with virtual node added, and the model is trained on NVIDIA Tesla V100 (32GB GPU memory).

Figure 5. An abstract PyTorch Implementation of our method.

We summarize implementation details and selected hyperparameters in this section. Note that for ALL of our method, we fix the ascent step number $M$ to 3 for simplicity. We leave more thorough step number search for future research. Experiments are done on hardware with Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz, and 128GB of RAM. If without mentioning, we use a single GeForce RTX 2080 Ti (11GB GPU memory). To highlight, for fair comparisons, we do not modify model architectures nor optimizing algorithms.

def flag(gnn, X, y, optimizer, criterion, M, alpha):
    gnn.train()
    optimizer.zero_grad()
    pert = torch.FloatTensor(X.shape) * u
    pert.data = torch.randn(pert_data)
    out = gnn(pert)
    loss = criterion(out, y) / M
    for _ in range(M-1):
        loss.backward()
        pert.data = pert.detach() + alpha * torch.sign(pert.grad.detach())
        pert.data = pert.detach() / M
        out = gnn(pert)
        loss = criterion(out, y) / M
    optimizer.step()
A.3.3 ogbg-ppa

GCN: perturbation step size $\alpha=2e^{-03}$, when virtual node is added we use a larger $\alpha=5e^{-03}$.

GIN: perturbation step size $\alpha=8e^{-03}$, when virtual node is added we use a smaller $\alpha=5e^{-03}$.

DeeperGCN: perturbation step size $\alpha=8e^{-03}$, and the model is trained on NVIDIA Tesla V100 (32GB GPU memory).

A.3.4 ogbg-code

GCN: perturbation step size $\alpha=8e^{-03}$ for both the vanilla model and the one augmented by virtual node.

GIN: perturbation step size $\alpha=8e^{-03}$ for both the vanilla model and the one augmented by virtual node.

DeeperGCN: perturbation step size $\alpha=8e^{-03}$ with virtual node added, and the model is trained on NVIDIA Tesla V100 (32GB GPU memory).

B. Loss Landscape Visualization

Figure 6 shows the loss landscape of GIN model. We can see that our method further regularizes the loss landscape.
Figure 6. Loss landscape visualization. The test is conducted on one random validation graph from ogbg-molhiv. Two models are GIN trained with FLAG and a vanilla GIN. (a) and (b) projects loss onto a random direction and the other adversarial direction, while (c) and (d) use two random directions.

<table>
<thead>
<tr>
<th>Name</th>
<th>#Nodes</th>
<th>#Edges</th>
<th>#Tasks</th>
<th>Train/Val/Test</th>
<th>Task Type</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>ogbn-products</td>
<td>2,449,029</td>
<td>61,859,140</td>
<td>1</td>
<td>8/2/90</td>
<td>Multi-classification</td>
<td>Accuracy</td>
</tr>
<tr>
<td>ogbn-proteins</td>
<td>132,534</td>
<td>39,561,252</td>
<td>112</td>
<td>65/16/19</td>
<td>Binary classification</td>
<td>ROC-AUC</td>
</tr>
<tr>
<td>ogbn-arxiv</td>
<td>169,343</td>
<td>1,166,243</td>
<td>1</td>
<td>54/18/28</td>
<td>Multi-classification</td>
<td>Accuracy</td>
</tr>
<tr>
<td>ogbn-mag</td>
<td>1,939,743</td>
<td>21,111,007</td>
<td>1</td>
<td>85/9/6</td>
<td>Multi-classification</td>
<td>Accuracy</td>
</tr>
<tr>
<td>Cora</td>
<td>2485</td>
<td>5069</td>
<td>1</td>
<td>no official split*</td>
<td>Multi-classification</td>
<td>Accuracy</td>
</tr>
</tbody>
</table>

Table 9. Node classification datasets statistics. * denotes we follow the split of [21].

<table>
<thead>
<tr>
<th>Name</th>
<th>#Nodes</th>
<th>#Edges</th>
<th>Train/Val/Test</th>
<th>Task Type</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>ogbl-ddi</td>
<td>4,267</td>
<td>1,334,889</td>
<td>80/10/10</td>
<td>Link prediction</td>
<td>Hits@20</td>
</tr>
<tr>
<td>ogbl-collab</td>
<td>235,868</td>
<td>1,285,465</td>
<td>92/4/4</td>
<td>Link prediction</td>
<td>Hits@50</td>
</tr>
</tbody>
</table>

Table 10. Link prediction datasets statistics.

<table>
<thead>
<tr>
<th>Name</th>
<th>#Graphs</th>
<th>Avg #Nodes</th>
<th>Avg #Edges</th>
<th>#Tasks</th>
<th>Train/Val/Test</th>
<th>Task Type</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>ogbg-molhiv</td>
<td>41,127</td>
<td>25.5</td>
<td>27.5</td>
<td>1</td>
<td>80/10/10</td>
<td>Binary classification</td>
<td>ROC-AUC</td>
</tr>
<tr>
<td>ogbg-molpcba</td>
<td>437,929</td>
<td>26.0</td>
<td>28.1</td>
<td>128</td>
<td>80/10/10</td>
<td>Binary classification</td>
<td>AP</td>
</tr>
<tr>
<td>ogbg-ppa</td>
<td>158,100</td>
<td>243.4</td>
<td>2,266.1</td>
<td>1</td>
<td>49/29/22</td>
<td>Multi-classification</td>
<td>Accuracy</td>
</tr>
<tr>
<td>ogbg-code</td>
<td>452,741</td>
<td>125.2</td>
<td>124.2</td>
<td>1</td>
<td>90/5/5</td>
<td>Sub-token prediction</td>
<td>F1 score</td>
</tr>
</tbody>
</table>

Table 11. Graph classification datasets statistics.