Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Node-level Prediction Tasks with Agent-based Graph Neural Networks

Bachelor's Thesis

Jannek Ulm

janulm@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Karolis Martinkus
Prof. Dr. Roger Wattenhofer

January 13, 2023

Acknowledgements

First, I want to thank my supervisor Karolis Martinkus for his incredible support, help, and inspiring ideas during the last few months. Further, I want to thank Prof. Dr. Roger Wattenhofer, the Distributed Computing Lab (ETH DISCO), and the D-ITET Computing facility for making this thesis possible and providing the necessary computational power and technical assistance. Last but not least, I want to thank my girlfriend, my friends, and my family for their emotional support throughout this thesis.

Abstract

Martinkus et al. [1] recently published a novel graph neural network model called AgentNet. This model was tailored for graph-level prediction tasks. We modified AgentNet for node-level prediction tasks such as node classification and regression. Different readout and propagation strategies were investigated, and results were discussed and compared to a baseline graph neural network model. We showed that the modified AgentNet works well and can even outperform the baseline model on a really hard PageRank dataset.

Contents

Chapter 1

Introduction

1.1 Introduction

Most of the world and the data surrounding us in our daily life can be modeled as graphs. A graph consists of nodes and edges connecting these nodes and can represent various things such as molecules, social networks, or even the internet. Gaining insight into this data and the real world is a crucial task impacting our daily lives. One of the tasks that comes with graphs naturally is predicting node-level properties. Two possible tasks are classifying a node or predicting a numerical value (regression). For node classification, each node in the graph is assigned a category from a predefined set of classes [2]. The task is to be able to learn to classify the nodes of the graph correctly. Node Property Regression is computing a numerical value assigned to each node in the graph. Many model architectures, such as the traditional message-passing graph neural networks, work well for these task.

1.2 AgentNet

For graphs, not only node-level predictions are essential tasks but also predictions on the graph level, i.e., predicting or classifying the graph as a whole. Just recently, Martinkus et al. [1] presented a novel graph neural network architecture called AgentNet. AgentNet uses a group of trained agents that all walk the graph in parallel and collectively decide on the output together. Martinkus et al. [1] were able to achieve good results and classify graphs in sub-linear time (sub-linear in the number of nodes in the graph).

1.3 PageRank

PageRank, a widely-used algorithm developed by Page et al. [3], measures the importance of webpages on the internet by utilizing the Random Surfer Model.

This model simulates the behavior of a "random" user browsing the web, following links between pages (or "nodes" in the web graph, transitioning over edges). The movement of information through the connections in a graph is known as "diffusion." Just like in the case of the Random Surfer Model, AgentNet "diffuses" through the graph, spreading information to nodes as it goes. Because PageRank is a random walk model [3], and AgentNet can perform random walks, AgentNet should, in theory, be able to simulate PageRank. Given this natural relationship between AgentNet and PageRank, we will investigate whether AgentNet is indeed capable of simulating the PageRank algorithm.

1.4 Our Contributions

We adapted AgentNet to solve node-level classification and regression tasks. Then we tested the adapted AgentNet on real-world graph regression and classification tasks. The results achieved by AgentNet were compared to those achieved by a baseline message-passing graph neural network. Furthermore, we investigated how well different strategies for AgentNet work and if AgentNet can simulate PageRank.

Chapter 2

Related Work

AgentNet is a novel approach for agent-based graph prediction problems. For node-level predictions, however, there are no other publications available that are really similar to AgentNet. We already noted the relation between AgentNet and random walks in the introduction.

Gasteiger et al. [4] developed a model called PPNP, which stands for personalized propagation of neural predictions. The model's training time was on par or faster, and the number of parameters was on par or lower than comparable models [4]. Their model "utilizes a propagation scheme derived from personalized PageRank [4]." Their model also adds a chance of teleporting back to the root node, which provides a balance between preserving locality and leveraging the information from a large neighborhood [4]. This functionality is closely related to resetting the agent's position to its starting node, a feature we also introduced to AgentNet.

In a follow-up paper, Gasteiger et al. [5] "remove the restriction of using only the direct neighbors by introducing a powerful, yet spatially localized graph convolution: Graph diffusion convolution (GDC). GDC leverages generalized graph diffusion, examples of which are the heat kernel and personalized PageRank". Their work shows that replacing "plain" message passing with GDC leads to significant performance improvements across a wide range of models and a variety of datasets [5]. PRGo, a model "which utilizes an efficient approximation of information diffusion in GNNs" [6] resulted in significant speed gains while maintaining "state-of-the-art prediction performance. They also demonstrate that PPRGo outperforms baselines" [6]. Generally, there are many improvements over [4], and the achieved state-of-the-art performance in node-classification [5, 6].

Another related model to AgentNet is DeepWalk, introduced in 2014 by Perozzi et al. [7]. DeepWalk works by first sampling random walks from a starting node. These walks are then used as input to a Skip-Gram model to learn node representation that can lateron be used for tasks such as node classification [7]. AgentNet is closely related to this idea of sampling random walks from a starting node, especially if agent reset is used, which is a feature we added to AgentNet.

CHAPTER 3

Models

3.1 AgentNet

AgentNet was initially designed to work on graph-level predictions [1]. The model uses a group of agents (each assigned a unique identifying id) that intelligently or randomly walk a fixed amount of steps on the graph. Note that we only show how the model works without edge data since the datasets we considered did not include any.

Table 3.1: Notation

v_i^t	node embedding of node with id i at time-step t
a_i^t	agent embedding of agent with id i at time-step t
$A(v_i^t)$	set of agents that visit node i at time-step t
$N(v_i)$	set of nodes reachable from v_i by an incident edge
$V(a_i^t)$	node that the agent i is visiting a time-step t
o_i	output for node i

Consider this high-level overview of how the model operates:

1. **Initialization:** The agents get placed according to the selected initialization strategy on the nodes. More detail for this and the other strategies are below in the parameters section

2. **For each step** of the agents, the model performs the following operations (for the final iteration, the last step (Agent Transition) is omitted since the agent's next position will not be used):

   (a) **Node Update:** Each active node (i.e., all nodes currently visited by an Agent.) gets updated using a skip connection. Here f_v is a 2-Layer

MLP that takes the current node embedding and all currently visiting agents embeddings as input.

$$v_i^t = v_i^{t-1} + f_v\left(v_i^{t-1}, \sum_{a_j^{t-1} \in A(v_i)} a_j^{t-1}\right) \quad \text{if } |A(v_i)| > 0 \text{ else } v_i^{t-1}.$$

(b) **Neighborhood Aggregation:** Each active node gets updated using a skip connection. Here f_n is a 2-Layer MLP that takes the current node embedding and all neighboring node embeddings as input. Please note that this update is performed separately and after the Node Update step.

$$v_i^t = v_i^t + f_n\left(v_i^t, \sum_{v_j^t \in N(v_i)} v_j^t\right) \quad \text{if } |A(v_i)| > 0 \text{ else } v_i^t$$

(c) **Agent Update:** Update all agents embedding using the current node's embedding and the aggregated neighborhood information. Here the model also uses a skip connection, f_a being a 2-Layer MLP taking the old's agent state and its currently visiting node embedding as an input.

$$a_i^t = a_i^{t-1} + f_a\left(a_i^{t-1}, v_{V(a_i)}^t\right)$$

(d) **Agent Transition:** The agents either travel to an adjacent node or stay on the current one. The method for choosing the next node uses probabilities the transition strategy assigns for each possible next node. Since this depends on the transition strategy, we cover this in more detail below. Assume the transition strategy returns the values $z_{a_i \to v_j}$ for all $v_j^t \in N^t(a_i)$. The model samples the next node from this distribution using GumbelSoftmax. GumbelSoftmax outputs a stochastic one-hot sample of the input distribution while remaining gradients for backpropagation [8, 9].

$$V(a_i) \leftarrow \text{GumbelSoftmax}\left(\left\{z_{a_i \to v_j} \text{ for } v_j^t \in N^t(a_i)\right\}\right)$$

3. **Readout:** Output prediction for either classification or regression. The model either uses node or agent embeddings to output the final prediction. The behavior is controlled by the selected readout strategy and depends on the node and agent embeddings.

$$o_i = \text{readout-strategy}(\text{agent\_embedding}, \text{node\_embedding})$$

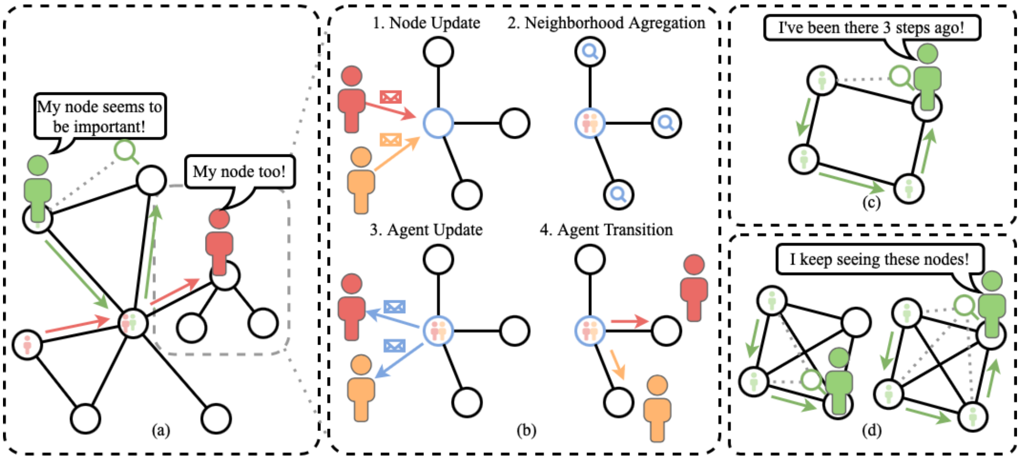

Figure 3.1: "AgentNet architecture. We have many neural agents walking the graph (a). Each agent at every step records information on the node, investigates its neighborhood, and makes a probabilistic transition to another neighbor (b). If the agent has walked a cycle (c) or a clique (d) it can notice." (Source: [1])

3.1.1 Parameters

For this thesis, we modified AgentNet by making a more apparent distinction between what strategies the agents use. Further, we added a specialized initialization strategy, reset-transition functionality, and all the different readout strategies introduced below. The model's high-level behavior depends on the selected initialization strategy, transition strategy, and readout strategy. Of course, many more parameters change the model's performance. Here we will focus on the most important ones. A complete list is given in the Appendix B.2.

Initialization Strategy

The AgentNet model [1] contained a random initialization strategy, and we added the one_to_one initialization strategy.

- **random:** All the agents are placed uniformly at random nodes. Note that multiple agents on one starting node are possible.
- **one_to_one** For each node, precisely one agent is starting on it. We added this strategy since we wanted to follow the thought of one agent being responsible for predicting exactly one node's output.

Transition Strategy

The transition strategy is one of the listed below. Please note that all nodes of the graph the model operates on always have self-loops. The self-loops ensure that at least one incident edge will always exist that an agent can use to transition to the next node. Therefore an agent may stay for multiple steps at the same node.

- **random:** Each agent decides uniformly at random which incident outgoing edge it will take. U(0,1) is the distribution that draws a number uniformly at random from $[0, 1]$.

$$z_{a_i \to v_j} = U(0,1) \text{ for } \quad v_j^t \in N^t(a_i)$$

- **random_reset:** This strategy is equivalent to the "random" strategy, but after a fixed number of steps (reset_neighbourhood_size), the agent's position gets reset to its initial starting position. This reset forces an agent only to see other nodes that are at most reset_neighbourhood_size-steps away from the agent's initial starting position.

- **attention:** In this strategy, the model computes standard self-attention [10] for all possible next nodes. Martinkus et al. [1] state that the query vector $Q(a_i^t)$ is a linear projection of the agent embedding, and the key vector $K(v_i^t, v_j^t)$ is a linear projection of the start and end node of the used edge for the transition. Where h is the size of the hidden dimension, Vaswani et al. [10] state that division by $sqrt(h)$ scales down the dot product and thus keeps the gradient bigger. This strategy's linear projections for the query and key are trainable. Therefore the model can learn to which nodes to transit next.

$$z_{a_i \to v_j} = \frac{Q\left(a_i^t\right)^T K\left(v_i^t, v_j^t\right)}{\sqrt{h}} \text{ for } \quad v_j^t \in N^t(a_i)$$

- **attention_reset:** This strategy is equivalent to the "attention" strategy, but after a fixed number of steps (reset_neighbourhood_size), the agent's position gets reset to its initial starting position. For this step, GumbelSoftMax isn't used, and no gradients are attached to the trainable parameters such as key and query in model training.

- **bias_attention:** This strategy is an extension of the attention strategy. First, the model keeps track of which agents visited which node in the past. The implementation of this tracking is really memory intensive and puts a constraint on the number of agents the model will work with, with current GPU memory availability. Let $x(a_i, v_j) \in [0, 1]$ denote the tracking, which indicates if or how recently an agent visited a specific node. If an agent i is placed on a new node j, then $x(a_i, v_j) = 1$; initially, this value is 0. After each step, there is a decay applied to this value ($x(a_i^{t+1}, v_j^{t+1}) = x(a_i^t, v_j^t) * 0.9$). To compute the desired bias-attention coefficient, the model combines the dot product attention from above with a bias for each node. This bias is a weighted sum of trainable parameters and indicator variables.

Table 3.2: Notation for bias_attention

$g_{previous}$	bias for the agents last node
$g_{current}$	bias for the agents current node
$g_{explored}, g_{unexplored}$	bias for explored and unexplored nodes
$1_{v_j} = V^{t-1}(a_i)$	indicator variable for previous node
$1_{v_j} = V(a_i)$	indicator variable for current node

$$
\begin{aligned}
f_{attention}\left(a_i^t, v_j^t\right) &= \frac{Q\left(a_i^t\right)^T K\left(v_i^t, v_j^t\right)}{\sqrt{h}} \\
f_{bias}\left(a_i^t, v_j^t\right) &= g_p\left(a_i^t\right) \cdot 1_{v_j = V^{t-1}(a_i)} + g_c\left(a_i^t\right) \cdot 1_{v_j = V(a_i)} \\
&\quad + g_e\left(a_i^t\right) \cdot x\left(a_i, v_j\right) + g_u\left(a_i^t\right) \cdot \left(1 - x\left(a_i, v_j\right)\right) \\
z_{a_i \to v_j} &= f_{bias}\left(a_i^t, v_j^t\right) + f_{attention}\left(a_i^t, v_j^t\right) \quad \text{for} \quad v_j^t \in N^t\left(a_i\right)
\end{aligned}
$$

- **bias_attention_reset:** This strategy is equivalent to the "bias-attention" strategy, but after a fixed number of steps (reset_neighbourhood_size), the agent's position gets reset to its initial starting position. For this step, GumbelSoftMax isn't used, and there are no gradients attached to the trainable parameters such as query, key, and bias parameters in training.

AgentNet [1] initially already contained random, attention, and bias-attention transition strategies. We added the reset functionality for each of these strategies.

Readout Strategy

The selected readout strategy can be one of the following choices and determines how and when the prediction for each node is made. Since the model initially only contained readout strategies for graph-level tasks, all of the readout strategies are new.

- **node_embedding:** This strategy is the most obvious one. Here the output for each node is computed using f_{out}, a 2-Layer MLP that uses the same node's final embedding as the input.

$$o_i = f_{out}(v_i)$$

- **agent_start:** Using this strategy, the model remembers the starting position of each agent. Note that this readout strategy requires the one_to_one initialization strategy. This is to ensure that for every node, an agent exists starting on it. Now the model uses the final agent embedding of the agent that started on the node and the 2-Layer MLP f_{out} to compute the output.

$$o_i = f_{out}(a_j^{\text{num\_steps}}) \text{ with } V(a_j^0) = i$$

- **last_agent_visited:** The above strategies always use the final (after the last step) node or agent embedding. In this strategy, we update an intermediate output vector for each step after the agent_update is complete. The current agent embedding and a 2-Layer MLP f_{out} are used to compute the output for the node the agent is visiting.

$$\text{in each step t: } o_i = f_{out}(a_j^t) \text{ with } V(a_j^t) = i$$

If the prediction task is node classification and not node regression, we added a LogSoftMax-layer to normalize the output for each node.

$$o_i = \text{LogSoftMax}(o_i)$$

Please note that not all combinations of strategies are valid. E.g., selecting to use readout=agent_start requires initialization=one_to_one to ensure that for each node, there exists an agent starting on this node.

Other Parameters

The behavior and performance of AgentNet not only depends on the listed strategies above but also on other arguments. Here we list the most important ones. The complete list can be found in the Appendix B.2.

Table 3.3: AgentNet Model Arguments

num_agents	number of agents that travel the graph
num_steps	number of steps that each agent takes
reset_neighbourhood_size	(only used when transition reset is used) number of steps -1 on which the agents position gets reset
classification	flag that indicates if LogSoftMax is used on the final output
hidden_units	size of embedding for each agent, node, and width of MLP middle-layer (factor 0.5)

3.2 Baseline-GCN Model

To compare the results, we will achieve with AgentNet, we need a baseline model that can be used for both node classification and node regression. For this purpose, we implemented a configurable GCN Model we will call Baseline-GCN. The model uses PyTorch's GCNConv-Layers, a convolutional/message-passing layer introduced by Kipf et al. [11]. By stacking up r layers, the model gets messages containing information about all the nodes in its r-hop neighborhood. After each layer (except the last one), we used a normalization layer. If the task is classification, we apply Softmax on the output to get class probabilities.

3.2.1 Parameters

Here we list the three most important arguments to the model. An exhaustive list can be found in the Appendix B.1.

Table 3.4: Baseline-GCN Model Arguments

num_layers	number of layers to control depth of the model and observable neighborhood
hidden_units	depth of the convolution layers
classification	flag that indicates if output gets passed trough a LogSoftMax-layer

Chapter 4

Datasets

4.1 Node Classification

We decided to use three different datasets to test both models. The datasets of choice are Cora [12], PubMed [12], and OGB-Arxiv [13], which all are citation networks in which each node is a scientific paper. Two directed edges represent a citation between two papers (i.e., for each citation, the datasets contain two edges, one from the citing paper to the cited paper and vice versa). The table below shows the number of nodes, edges, classes, and features. It also contains the training set proportion, which is the number of nodes in the training mask divided by the total number of nodes and the average number of outgoing edges per node. The average number of outgoing edges is also the average number of neighboring nodes observed by an agent in neighborhood aggregation or by a GCN message passing layer.

Table 4.1: Dataset statistics

Dataset	**Nodes**	**Edges**	**Classes**	**Features**	**Training prop.**	**Avg. Out**
Cora	2708	10,556	7	1,433	0.052	3.90
PubMed	19,717	88,648	3	500	0.003	4.49
OGB-Arxiv	169,343	1,116,243	40	128	0.537	6.59

4.2 PageRank

We decided to focus on PageRank for the node regression task because AgentNet should be able to simulate the Random Surfer Model. Therefore we need different datasets. We reused the inherited graph structure from the datasets previously used (Cora, PubMed, OGB-Arxiv) and recomputed the missing parts. Missing is the correct PageRank weight and the feature vector for each node. We assigned each node the initial vector v = [1] as the feature vector. The PageRank weight, introduced by Page et al. [3], was computed using NetworkX PageRank power iteration implementation with a damping factor of 0.85 [14].

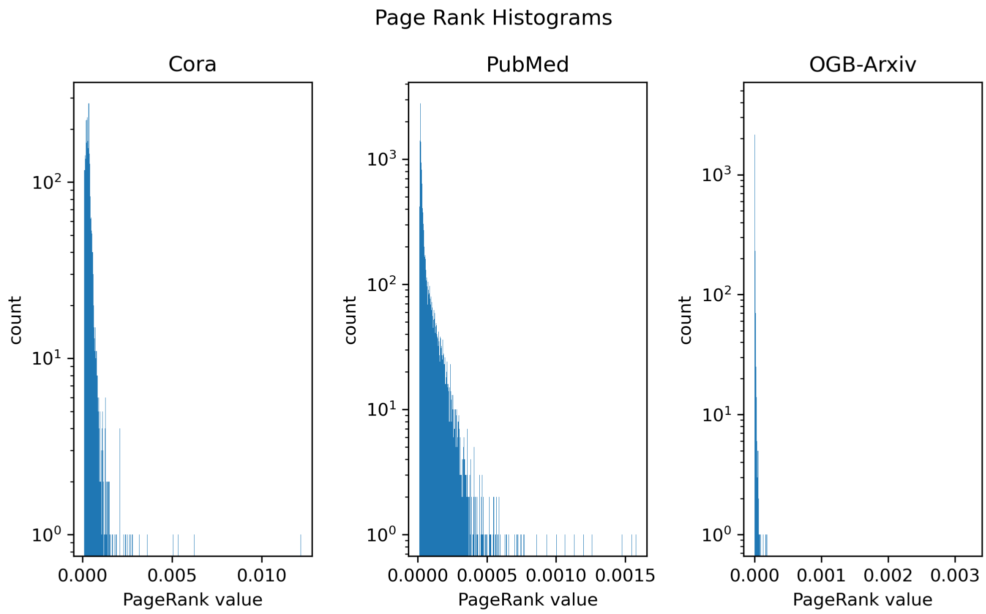

Figure 4.1: Histogram of PageRank values for each dataset.

We looked at the histograms to understand how the PageRank distribution differs between the datasets. Please note that most of OGB-Arxiv's nodes are distributed much tighter than in the other two datasets.

4.2.1 Random Surfer Model

The PageRank weights can also be modeled and computed using the Random Surfer Model [3]. This notion corresponds to a group of "randomly" surfing agents that walk on the webpages and randomly decide on which link to click next. The damping factor is used to model a switch of search interest or the start of a new search. For each next step, the random surfer either takes any outgoing link randomly with probability of the damping factor and with the probability of (1 - damping factor), the surfer goes to a new randomly selected node in the graph. Each time a surfer goes to a node, he increments a node's counter, starting at 0. These counters and their total sum can be used to compute the PageRank values. The notion of a randomly surfing agent is close to how AgentNet can operate using the random transition_strategy. How well this works is covered below in the chapter for node regression.

Chapter 5

Node Classification

In this chapter, we will see how well each model does on the selected datasets and how well AgentNet's different strategies work.

5.1 Experiment Setup

Both models got trained using the AdamW optimizer with multiple possible values for the learning rate and a negative log-likelihood loss function. AgentNet was trained for a maximum of 4000 epochs, with early stopping enabled after 1500 epochs with a window size of 400. Baseline-GCN was trained for at most 2000 epochs, with early stopping enabled after 1000 epochs with a window size of 200. For every epoch, we logged the validation and test classification accuracy (i.e., how many percent of all nodes got classified with the correct category) and returned the test accuracy of the epoch with the highest validation accuracy.

5.2 Baseline-GCN

The model was trained as described above for 53 total combinations of parameters. A more detailed list can be found in the Appendix C.1. The possible number of layers were = $[1, 2, 3]$. We only report the test accuracy for the best combination of arguments besides the number of layers here. The best-achieved accuracy is highlighted.

Table 5.1: Baseline-GCN test accuracy node classification

Num. layers	**Cora**	**PubMed**	**OGB-Arxiv**
1	0.748	0.733	0.644
2	**0.772**	**0.767**	0.717
3	0.739	0.750	**0.720**

5.3 AgentNet

The model was trained as described above with a selection of possible argument combinations. This is due to the high number of parameters and the large search space of parameter combinations. We used the one_to_one initialization strategy for all three datasets and set the number of agents equal to the number of nodes. Due to PubMed and OGB being much larger graphs (in terms of the number of nodes), we could not run all transition strategies on every dataset. The visited tracking, used in bias_attention and bias_attention_reset, memory consumption depends on the (number of agents · number of nodes). Due to this, using bias_attention on PubMed and OGB was not feasible with the available GPU Memory. We also excluded the random transition strategy and focused on the remaining transition strategies where the model "actively" decides on which node to proceed. All possible readout strategies were considered.

Here we visualize the resulting accuracies in parallel lines plots on a selection of all considered arguments. The best argument combination is highlighted in green. A more detailed list of all computation results can be found in the Appendix C.2. Please note that if the lines collide, the one with the highest achieved accuracy gets drawn on top. This is to get a better insight into the maximum achievable accuracy per selected arguments.

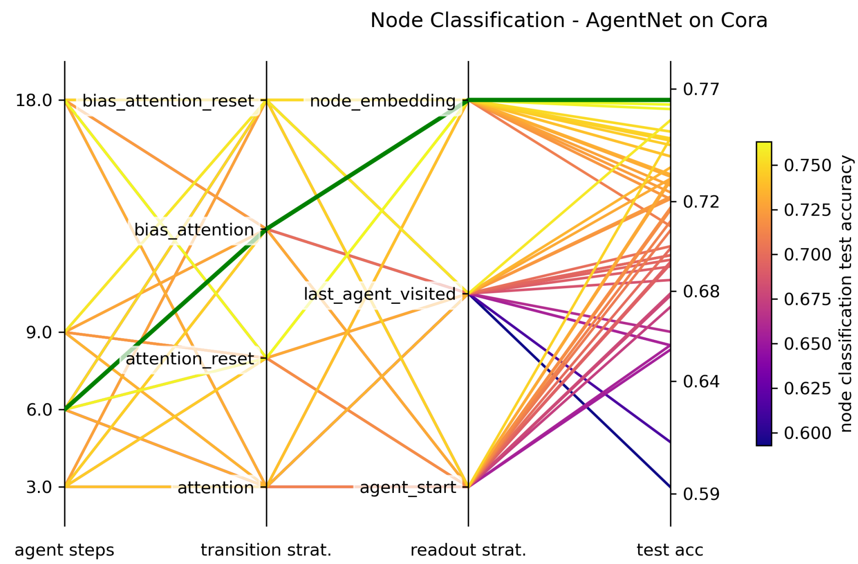

Figure 5.1: Node classification test accuracy of AgentNet on Cora

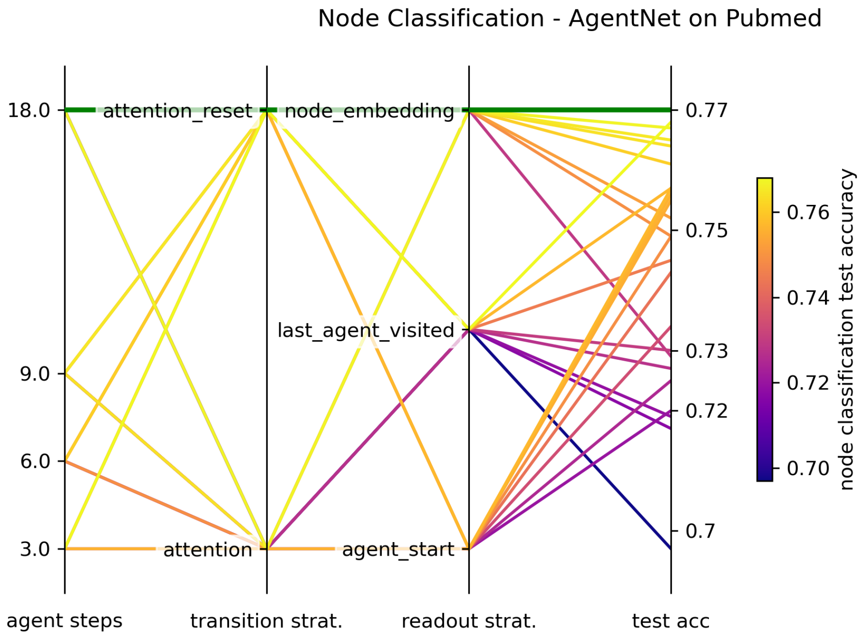

Figure 5.2: Node classification test accuracy of AgentNet on PubMed

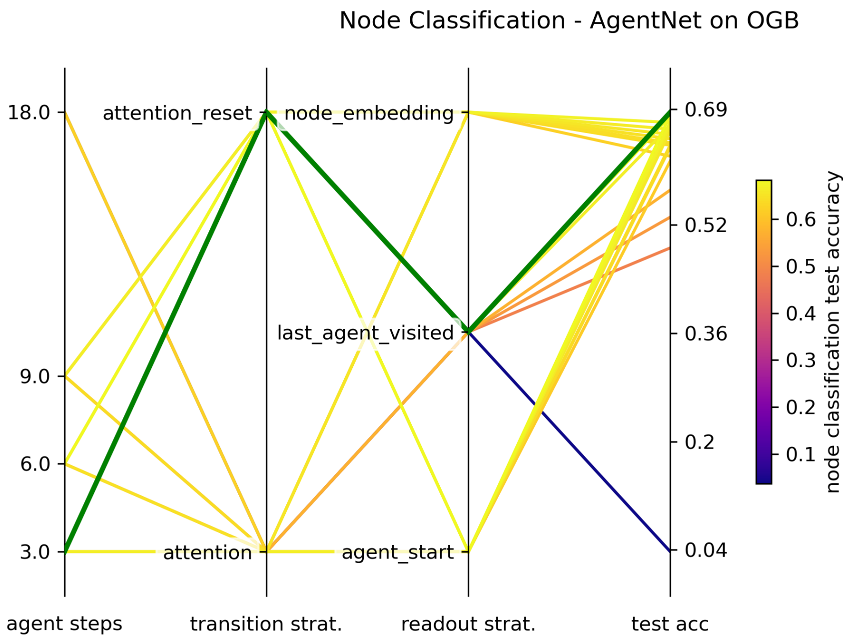

Figure 5.3: Node classification test accuracy of AgentNet on OGB-Arxiv

The parallel coordinate plots gave insight into the spread of outcomes achieved by selecting different strategies. Here we highlight across all datasets and across all other arguments using node_embedding as the readout strategy gives a consistent and good result. Using last_agent_visited as the readout strategy has a vast spread ranging from 4% to 69% achieved accuracy for the OGB dataset (Figure 5.3). Using agent_start for readout consistently has a much tighter spread in achieved accuracy and performs par on par or even better than last_agent_visited. This means that node_embedding and agent_start are less dependent on the choice of the other arguments and result in a more robust achieved result, whereas last_agent_visited heavily depends on a good choice of the other arguments. We only consider the best-achieved accuracy, achieved by selecting each transition and readout strategy.

Table 5.2: AgentNet test accuracy node classification - Transition-Strategy

Transition-Strategy	**Cora**	**PubMed**	**OGB-Arxiv**
attention	0.740	0.767	0.667
attention_reset	0.763	**0.770**	**0.686**
bias_attention	**0.765**	-	-
bias_attention_reset	0.756	-	-

Transition-Strategy: (Table 5.2) For Cora and OGB, attention_reset has about a 2% accuracy advantage over attention. For PubMed, this advantage is negligible. Due to memory limitations, bias_attention was not available for PubMed and OGB. On Cora, bias_attention got the best results, but what was on par with attention_reset. Hence adapting attention and introducing the extra reset step improved the model's architecture and performance.

Table 5.3: AgentNet test accuracy node classification - Readout-Strategy

Readout-Strategy	**Cora**	**PubMed**	**OGB-Arxiv**
node_embedding	**0.765**	**0.770**	0.670
last_agent_visited	0.756	0.768	**0.686**
agent_start	0.748	0.757	0.682

Readout-Strategy: (Table 5.3) We note that for all datasets the "right" choice of the readout-strategy gains about 1.5% better accuracy. For Cora and PubMed, node_embedding works best, and last_agent_visited performs worst. Surprisingly for OGB, it is the other way around, and last_agent_visited is the best and node_embedding the least good one. Due to that, the results for each strategy are just slightly different. That gives some insight into how AgentNet must learn to classify nodes. Both for agent_start and last_agent_visited, AgentNet accumulates the "important" information for classification in the agent's embeddings. For node_embedding this information is accumulated in the nodes embeddings. These results show that AgentNet can successfully accumulate the necessary information for node classification in both agents and nodes.

5.4 Comparision of Baseline-GCN and AgentNet

Table 5.4: AgentNet vs Baseline-GCN - Node Classification

Model	**Cora**	**PubMed**	**OGB-Arxiv**
AgentNet	0.765	**0.770**	0.686
Baseline-GCN	**0.772**	0.767	**0.720**

For Cora and PubMed, AgentNet and the Baseline-GCN's performance are almost identical. On OGB-Arxiv, the Baseline-GCN model had an advantage of 3.4% greater accuracy. The results of this experiment show that AgentNet can achieve great results on graph-classification tasks [1] but also of working reasonably well for the node-classification task and can keep up with our Baseline-GCN model.

Chapter 6

Node Regression - PageRank

In this chapter, we will see how well each model does on the modified PageRank datasets and how well AgentNet's different strategies work.

6.1 Experiment Setup

Both models were trained using the AdamW optimizer with multiple possible values for the learning rate and a MSE-loss function (Mean Squared Error). AgentNet was trained for a maximum of 4000 epochs, with early stopping enabled after 1000 epochs with a window size of 200. Baseline-GCN was trained for at most 5000 epochs, with early stopping enabled after 1000 epochs with a window size of 200.

The MSE-loss works well for training this task but has no intuitive interpretation. We use Spearman's rank correlation coefficient to have a comparable metric, such as accuracy, for the classification task. The Spearman coefficient measures the ranking of two variables and can be computed as follows [15]. The function get_rank(x) takes the input vector x of size n of unique numbers and returns the vector containing the indices of all the numbers (from 1 to n) if x were to get sorted.

$$\mathit{spear}(x,y) = 1 - \frac{6\sum d_i^2}{n\left(n^2-1\right)} \text{ , where } d_i = \text{get\_rank(x) - get\_rank(y)}$$

The Spearman rank correlation coefficient ranges from -1 to +1 and intuitively expresses how the two input vectors, x and y rank's, are correlated.

- spear(x,y) = +1 $\iff$ x and y are perfectly associated
- spear(x,y) = 0 $\iff$ x and y are not associated
- spear(x,y) = -1 $\iff$ x and y are perfectly negatively associated

For training, we stick to using the MSE-loss. However, for the evaluation of results, we use Spearman's coefficient to get a sense of how well the computed PageRank weights are associated with the ground truth values. For every epoch, we logged the validation and test Spearman coefficient and returned the test Spearman coefficient of the epoch with the highest validation Spearman coefficient.

6.2 Baseline-GCN

The model was trained as described above for 71 total combinations of parameters. A more detailed list can be found in the Appendix D.1. The possible number of layers was $= [1, 2, 3, 4]$. Here we only report the Spearman coefficient for the best combination of arguments besides the number of layers. The best-achieved coefficient is highlighted.

Table 6.1: Baseline-GCN Spearman rank correlation coefficient - PageRank

Num. layers	Cora	PubMed	OGB-Arxiv
1	0.741	0.491	0.285
2	0.963	0.957	**0.325**
3	0.983	**0.967**	0.319
4	**0.986**	0.950	0.320

For Cora and PubMed, the model works well. The results of OGB-Arxiv, on the other hand, are poor. A Spearman coefficient of 0.3 is considered a low correlation. As we saw in the histograms (Figure 4.1), the distribution of OGB's PageRank values for the nodes is much tighter and hence harder to separate. Furthermore, the number of average neighbors per node is higher than in the other two datasets (Table 4.1). Potentially that is why the model is having more trouble with this dataset.

6.3 AgentNet

The model was trained as described in the experimental setup section and only with a selection of all possible argument combinations. The high number of parameters and choices results in a vast search space.

For all three datasets, we used the random initialization strategy and used the following choices for the number of agents (n = the number of nodes in the dataset): [0.25n,0.5n,n,1.5n,2n]. We tested the following choices for transition strategy: [random, attention, attention_reset]. Readout strategies got limited to [node_embedding,last_agent_visited]. We cut the choices of transition and readout strategy since we drastically increased the possible number of agents and still had to keep a feasible computation time. A more detailed list of other computation results can be found in the Appendix D.2. The results below were visualized in a parallel coordinate plot displaying the most relevant parameter choices. Please note that if the lines collide, the one with the highest achieved Spearman coefficient gets drawn on top. This is to get a better insight into the maximum achievable coefficient per selected argument. The green line marks the best overall achieved Spearman coefficient.

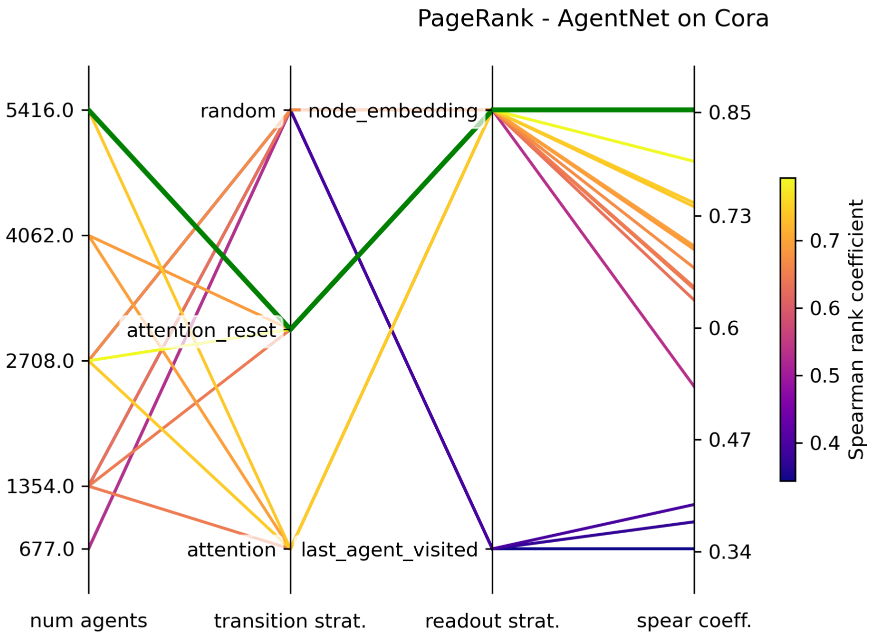

Figure 6.1: PageRank Spearman coefficient of AgentNet on Cora

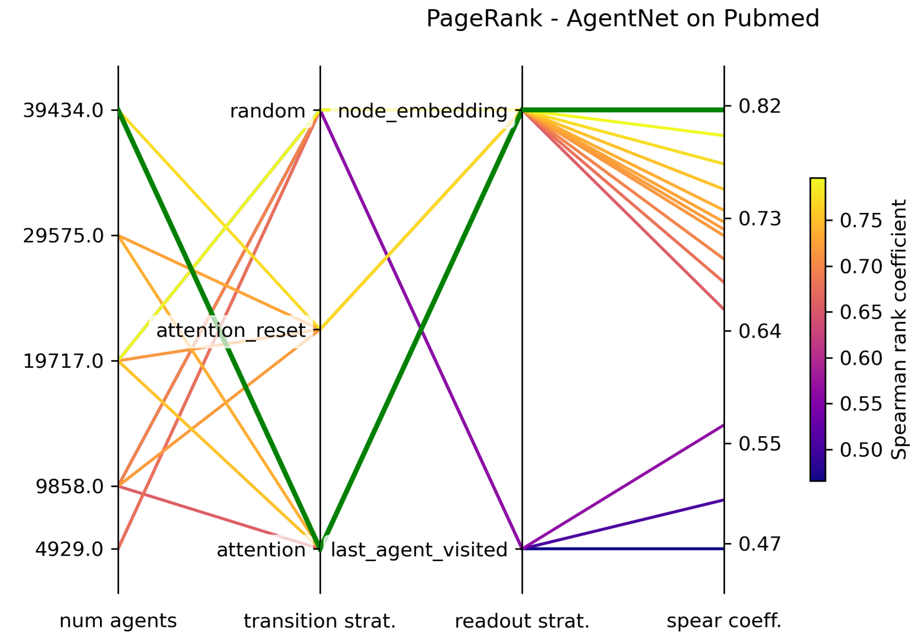

Figure 6.2: PageRank Spearman coefficient of AgentNet on PubMed

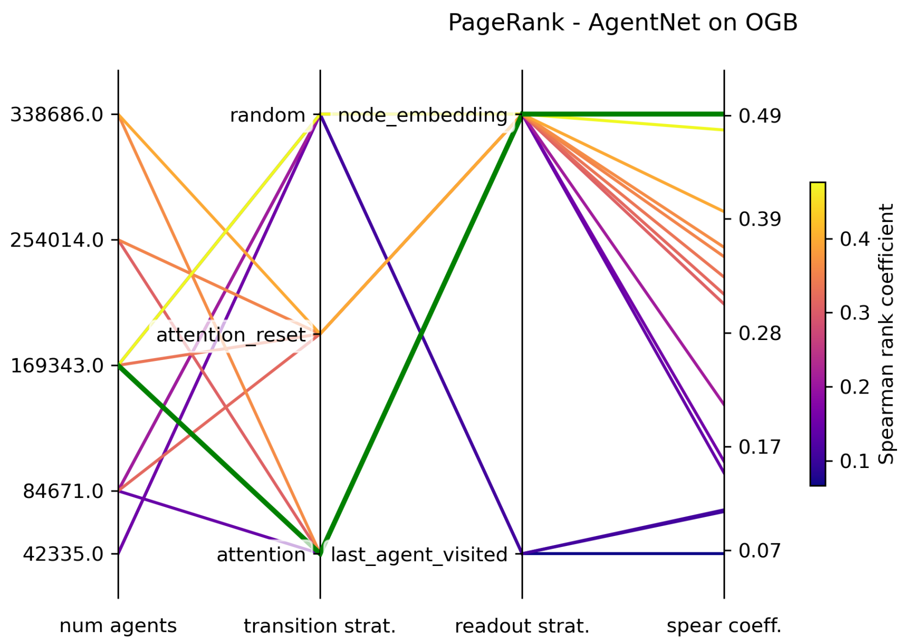

Figure 6.3: PageRank Spearman coefficient of AgentNet on OGB-Arxiv

The parallel coordinate plots gave insight into the spread of outcomes achieved by selecting different strategies. Across all datasets, using the last_agent_visited readout strategy gave poor results. Reading out the nodes embedding gave much better results, although the spread is high. This big spread means that the node_embedding strategy only works well with the right choice of other parameters. Now let us investigate the impact of each choice of the following parameters: number of agents, transition, and readout strategy.

Table 6.2: PageRank Spearman coefficient of AgentNet - Number of Agents (n = number of nodes)

Number of Agents	Cora	PubMed	OGB-Arxiv
0.25n	0.532	0.679	0.156
0.5n	0.647	0.721	0.317
1n	0.793	0.796	**0.491**
1.5n	0.695	0.736	0.353
2n	**0.853**	**0.817**	0.397

Number of Agents: For Cora and PubMed, one can see (Table 6.2) that with an increasing number of agents, the achieved Spearman coefficient increases (almost linearly). For OGB, on the other hand, the choice of "the number of agents" = "the number of nodes" gives the best result. This parameter had a significant effect on the achieved coefficient for all datasets.

Table 6.3: PageRank Spearman coefficient of AgentNet - Transition-Strategy

Transition-Strategy	Cora	PubMed	OGB-Arxiv
random	0.669	0.796	0.476
attention	0.745	**0.817**	**0.491**
attention_reset	**0.853**	0.773	0.397

Transition-Strategy: (Table 6.3) For PubMed and OGB attention gives about a 2% increase over the random transition-strategy and even more over attention_reset. For Cora, on the other hand, attention_reset has a 10% advantage over the attention strategy.

Table 6.4: PageRank Spearman coefficient of AgentNet - Readout-Strategy

Readout-Strategy	Cora	PubMed	OGB-Arxiv
node_embedding	**0.853**	**0.817**	**0.491**
last_agent_visited	0.395	0.565	0.109

Readout-Strategy: On all datasets, the readout-strategy node_embedding worked better by consistently providing a minimum 44% increase in Spearman rank coefficient over the last_agent_visited strategy. This is a different result than the tests in node classification achieved (Table 5.3), where node_embedding and last_agent_visited had almost identical results. This difference shows a fundamental difference in how AgentNet learns to classify nodes vs. how AgentNet computes PageRank coefficients. We can also conclude that for node classification, AgentNet was able to store the "important" information for classification in the agents and the nodes-embeddings (due to that, both strategies work well). For computing PageRank weights, on the other hand, AgentNet seems to be better capable of storing the "important" information for PageRank estimation in the nodes embedding instead of the agent's embeddings. One can wonder if AgentNet possibly similarly computes the PageRank weights as the Random Surfer Model by storing something similar as a counter on each node embedding.

6.4 Comparison of Baseline-GCN and AgentNet

Table 6.5: AgentNet vs Baseline-GCN - PageRank Spearman coefficient

Model	Cora	PubMed	OGB-Arxiv
AgentNet	0.853	0.817	**0.491**
Baseline-GCN	**0.986**	**0.967**	0.325

For Cora and PubMed, the Baseline-GCN performed great and even had a pretty good (13%) advantage over AgentNet's poorer performance. For OGB, both models struggled to achieve even a mediocre Spearman rank coefficient, but AgentNet outperforms Baseline-GCN by a significant margin. The results of this experiment show that AgentNet is not only capable of achieving good results on graph-classification tasks [1]. While AgentNet is somewhat worse on the "easier" datasets, maybe due to the stochasticity and worse exploration, it was able to better deal with the "hard" case of OGB-Arxiv than the baseline model was.

Chapter 7

Conclusion

7.1 Conclusion

The focus of this work was to modify graph-level AgentNet to work for node level prediction task. This has been successfully implemented building on the initial work of Martinkus et al. [1]. On this basis we asked ourselves the following questions:

- How well do different readout strategies work?
- Is AgentNet able to simulate the PageRank algorithm?

First, we successfully showed that all three readout strategies work reasonably well for node classification and are on par with our baseline model.

The second central insight was that AgentNet indeed can simulate the PageRank algorithm. We discovered that reading out the node embeddings works well while trying to read out using the accumulated information of the agents only works poorly.

7.2 Future Work

A problem we faced during testing was the implementation of tracking required for bias_attention. As a first step, it could be made more memory efficient to make the model feasible for many agents for larger datasets. Since agent based node-level prediction is a novel approach and the results were promising, we are convinced that further research and investigation will be rewarding. A start could be adapting AgentNet even more. For example, one could study transition and readout strategies that are not implemented yet. Furthermore, one could also investigate the dependence of the number of agents for node classification to potentially reduce prediction cost while maintaining accuracy. Also even higher number of agents could be tested for computing PageRank.

Bibliography

[1] K. Martinkus, P. A. Papp, B. Schesch, and R. Wattenhofer, "Agent-based graph neural networks," 2022. [Online]. Available: https://arxiv.org/abs/2206.11010

[2] L. Wu, P. Cui, J. Pei, and L. Zhao, *Graph Neural Networks: Foundations, Frontiers, and Applications.* Singapore: Springer Singapore, 2022.

[3] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Technical Report 1999-66, November 1999, previous number = SIDL-WP-1999-0120. [Online]. Available: http://ilpubs.stanford.edu:8090/422/

[4] J. Gasteiger, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," 2018. [Online]. Available: https://arxiv.org/abs/1810.05997

[5] J. Gasteiger, S. Weißenberger, and S. Günnemann, "Diffusion improves graph learning," 2019. [Online]. Available: https://arxiv.org/abs/1911.05485

[6] A. Bojchevski, J. Klicpera, B. Perozzi, A. Kapoor, M. Blais, B. Rózemberczki, M. Lukasik, and S. Günnemann, "Scaling graph neural networks with approximate pagerank," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 2464–2473. [Online]. Available: https://doi.org/10.1145/3394486.3403296

[7] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, aug 2014. [Online]. Available: https://doi.org/10.1145%2F2623330.2623732

[8] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," 2016. [Online]. Available: https://arxiv.org/abs/1611.01144

[9] W. Joo, D. Kim, S. Shin, and I.-C. Moon, "Generalized gumbel-softmax gradient estimator for various discrete random variables," 2020. [Online]. Available: https://arxiv.org/abs/2003.01847

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[11] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016. [Online]. Available: https://arxiv.org/abs/1609.02907

[12] P. G. Team. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/_modules/torch_geometric/datasets/planetoid.html

[13] C. O. Team. [Online]. Available: https://ogb.stanford.edu/docs/nodeprop/

[14] N. Team. [Online]. Available: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html

[15] Wikipedia contributors, "Spearman's rank correlation coefficient — Wikipedia, the free encyclopedia," 2022, [Online; accessed 11-January-2023]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Spearman%27s_rank_correlation_coefficient&oldid=1125564582

[16] Jannek Ulm, "Agentnet bachelor thesis - jannek ulm - implementation," 2023, [Online; accessed 11-January-2023]. [Online]. Available: https://gitlab.ethz.ch/disco-students/hs22/janulm_agent_nodes

Appendix A

Implementation

The modified implementation of AgentNet, Baseline-GCN, and all the code we used to run the experiments on ITET's computing facility can be found in the Gitlab repository [16]. Furthermore, the code used to generate the parallel lines plot and the code for creating the tables is also in the appendix.

Appendix B

List of Model Arguments

B.1 Baseline-GCN

Table B.1: Baseline-GCN - List of Model Arguments

Argument	**Default value**
total num layers	-
training dropout	-
learning rate	-
weight decay	0.01
in channels	-
hidden channels	-
out channels	-
classification flag	-

B.2 AgentNet

Table B.2: AgentNet - List of Model Arguments

Argument	**Default value**
init strat	-
transition strat	-
readout strat	-
classification (flag)	-
hidden units	-
num agents	-
num steps	-
reset neighbourhood size	-
training dropout rate	-
reduce function	'sum'
use time embedding	true
weight decay	0.01
activation function	leaky-relu
leakyRELU_neg_slope	0.01
visited decay	0.9
use_mlp_input	True
post_ln	False
global_agent_node_update	False
global_agent_agent_update	False

Appendix C

Results - Node Classification

We decided to only list the best resulting runs when selecting unique combinations of parameters and omitting the other parameters such as learning rate or dropout.

For Baseline-GCN the best unique combinations of ['hidden units','num layers'] are listed. For AgentNet the best unique combinations of ['agent steps','neighborhood reset','transition strat.','readout strat.'] are listed.

C.1 Baseline-GCN

Cora

The model was trained with all possible combinations of these arguments:

- dims = [16,32,64]
- dropout rate in GCN Layer:= [0.0,0.3]
- learning rates:= [0.01,0.001,0.0001]
- num_layers:= [1,2,3]

Table C.1: Computation results of Baseline-GCN on Cora

	hidden units	dropout	lr	num layers	test acc
11	16.0	0.3	0.010	3.0	0.680
13	16.0	0.3	0.001	2.0	0.703
20	32.0	0.0	0.010	3.0	0.713
38	64.0	0.0	0.010	3.0	0.739
48	64.0	0.3	0.001	1.0	0.748
3	16.0	0.0	0.001	1.0	0.748
30	32.0	0.3	0.001	1.0	0.748
19	32.0	0.0	0.010	2.0	0.750
46	64.0	0.3	0.010	2.0	0.772

PubMed

The model was trained with all possible combinations of these arguments:

- dims = [16,32,64]
- dropout rate in GCN Layer:= [0.0,0.3]
- learning rates:= [0.01,0.001,0.0001]
- num_layers:= [1,2,3]

Table C.2: Computation results of Baseline-GCN on PubMed

	hidden units	dropout	lr	num layers	test acc
10	16.0	0.3	0.0100	2.0	0.718
29	32.0	0.3	0.0100	3.0	0.724
17	16.0	0.3	0.0001	3.0	0.729
21	32.0	0.0	0.0010	1.0	0.733
39	64.0	0.0	0.0010	1.0	0.733
0	16.0	0.0	0.0100	1.0	0.733
47	64.0	0.3	0.0100	3.0	0.750
31	32.0	0.3	0.0010	2.0	0.752
46	64.0	0.3	0.0100	2.0	0.767

OGB-Arxiv

The model was trained with all possible combinations of these arguments:

- dims = [256, 32,64,128]
- dropout rate in GCN Layer:= [0.0,0.3,0.5]
- learning rates:= [0.01,0.001,0.0001]
- num_layers:= [1,2,3]

Table C.3: Computation results of Baseline-GCN on OGB

	hidden units	dropout	lr	num layers	test acc
0	256.0	0.5	0.010	1.0	0.643931
90	128.0	0.3	0.010	1.0	0.643931
72	64.0	0.0	0.010	1.0	0.643931
36	32.0	0.3	0.010	1.0	0.643931
46	32.0	0.0	0.010	2.0	0.695904
47	32.0	0.0	0.010	3.0	0.704627
73	64.0	0.0	0.010	2.0	0.705471

Continued on next page

Table C.3: Computation results of Baseline-GCN on OGB

	hidden units	dropout	lr	num layers	test acc
82	128.0	0.5	0.010	2.0	0.712014
74	64.0	0.0	0.010	3.0	0.713392
13	256.0	0.3	0.001	2.0	0.717096
92	128.0	0.3	0.010	3.0	0.717898
5	256.0	0.5	0.001	3.0	0.720635

C.2 AgentNet

Cora

For this dataset the model was trained with all possible combinations of these arguments:

- dims = [16,32,64]
- training dropout rate = [0.0,0.3]
- learning rates = [0.01,0.001,0.0001]
- steps = [3,6,9,18]
- num agents = [n] (n = number of nodes in graph)
- learning rates = [0.01,0.001,0.0001]
- readout strategies = [node_embedding,last_agent_visited,agent_start]
- transition strategies = [attention, attention_reset, bias_attention, bias_attention_reset]
- initialization strategies = [one_to_one]
- neighborhood_reset_sizes = [1,2,3]

Table C.4: Computation results of AgentNet on Cora

	transition strat.	readout strat.	agent steps	n.hood reset	test acc
108	bias_attention	last_agent_visited	6.0	1.0	0.593
32	attention	last_agent_visited	3.0	1.0	0.613
674	attention_reset	agent_start	6.0	2.0	0.625
682	attention_reset	last_agent_visited	6.0	2.0	0.628
673	attention_reset	agent_start	6.0	1.0	0.636
681	attention_reset	last_agent_visited	6.0	1.0	0.638
1393	attention_reset	agent_start	18.0	1.0	0.638
678	bias_attention_reset	agent_start	6.0	2.0	0.645

Continued on next page

Table C.4: Computation results of AgentNet on Cora

	transition strat.	readout strat.	agent steps	n.hood reset	test acc
686	bias_attention_reset	last_agent_visited	6.0	2.0	0.646
1191	bias_attention_reset	last_agent_visited	3.0	3.0	0.650
1248	attention	agent_start	6.0	1.0	0.654
677	bias_attention_reset	agent_start	6.0	1.0	0.654
1263	bias_attention_reset	last_agent_visited	6.0	3.0	0.655
679	bias_attention_reset	agent_start	6.0	3.0	0.656
1256	attention	last_agent_visited	6.0	1.0	0.656
828	bias_attention	last_agent_visited	18.0	1.0	0.656
1181	bias_attention_reset	agent_start	3.0	1.0	0.657
685	bias_attention_reset	last_agent_visited	6.0	1.0	0.662
611	attention_reset	last_agent_visited	3.0	3.0	0.663
245	bias_attention_reset	agent_start	18.0	1.0	0.668
1401	attention_reset	last_agent_visited	18.0	1.0	0.672
1402	attention_reset	last_agent_visited	18.0	2.0	0.672
1251	attention_reset	agent_start	6.0	3.0	0.673
1185	attention_reset	last_agent_visited	3.0	1.0	0.676
1392	attention	agent_start	18.0	1.0	0.678
253	bias_attention_reset	last_agent_visited	18.0	1.0	0.678
676	bias_attention	agent_start	6.0	1.0	0.679
1190	bias_attention_reset	last_agent_visited	3.0	2.0	0.682
242	attention_reset	agent_start	18.0	2.0	0.685
824	attention	last_agent_visited	18.0	1.0	0.685
1178	attention_reset	agent_start	3.0	2.0	0.686
1329	attention_reset	last_agent_visited	9.0	1.0	0.690
1189	bias_attention_reset	last_agent_visited	3.0	1.0	0.691
1320	attention	agent_start	9.0	1.0	0.692
1395	attention_reset	agent_start	18.0	3.0	0.693
827	attention_reset	last_agent_visited	18.0	3.0	0.694
1625	attention_reset	node_embedding	9.0	1.0	0.695
1332	bias_attention	last_agent_visited	9.0	1.0	0.696
1186	attention_reset	last_agent_visited	3.0	2.0	0.696
1330	attention_reset	last_agent_visited	9.0	2.0	0.698
1183	bias_attention_reset	agent_start	3.0	3.0	0.698
1396	bias_attention	agent_start	18.0	1.0	0.699
1476	bias_attention	last_agent_visited	3.0	1.0	0.700
1321	attention_reset	agent_start	9.0	1.0	0.700
1398	bias_attention_reset	agent_start	18.0	2.0	0.701
1182	bias_attention_reset	agent_start	3.0	2.0	0.707
1624	attention	node_embedding	9.0	1.0	0.709
1322	attention_reset	agent_start	9.0	2.0	0.709
1179	attention_reset	agent_start	3.0	3.0	0.710
1176	attention	agent_start	3.0	1.0	0.711
1630	bias_attention_reset	node_embedding	9.0	2.0	0.711
1627	attention_reset	node_embedding	9.0	3.0	0.713
1482	attention_reset	node_embedding	3.0	2.0	0.714
1177	attention_reset	agent_start	3.0	1.0	0.716
1325	bias_attention_reset	agent_start	9.0	1.0	0.716
831	bias_attention_reset	last_agent_visited	18.0	3.0	0.716
1323	attention_reset	agent_start	9.0	3.0	0.717
1399	bias_attention_reset	agent_start	18.0	3.0	0.717
1631	bias_attention_reset	node_embedding	9.0	3.0	0.718
1558	bias_attention_reset	node_embedding	6.0	2.0	0.719
1626	attention_reset	node_embedding	9.0	2.0	0.721
1485	bias_attention_reset	node_embedding	3.0	1.0	0.721

Continued on next page

Table C.4: Computation results of AgentNet on Cora

	transition strat.	readout strat.	agent steps	n.hood reset	test acc
1331	attention_reset	last_agent_visited	9.0	3.0	0.721
1628	bias_attention	node_embedding	9.0	1.0	0.721
1406	bias_attention_reset	last_agent_visited	18.0	2.0	0.722
1333	bias_attention_reset	last_agent_visited	9.0	1.0	0.724
1700	bias_attention	node_embedding	18.0	1.0	0.724
1487	bias_attention_reset	node_embedding	3.0	3.0	0.726
1486	bias_attention_reset	node_embedding	3.0	2.0	0.727
1324	bias_attention	agent_start	9.0	1.0	0.729
1699	attention_reset	node_embedding	18.0	3.0	0.729
1259	attention_reset	last_agent_visited	6.0	3.0	0.730
1180	bias_attention	agent_start	3.0	1.0	0.730
1552	attention	node_embedding	6.0	1.0	0.731
1696	attention	node_embedding	18.0	1.0	0.732
1326	bias_attention_reset	agent_start	9.0	2.0	0.733
1698	attention_reset	node_embedding	18.0	2.0	0.734
1328	attention	last_agent_visited	9.0	1.0	0.735
1702	bias_attention_reset	node_embedding	18.0	2.0	0.736
1335	bias_attention_reset	last_agent_visited	9.0	3.0	0.737
1701	bias_attention_reset	node_embedding	18.0	1.0	0.739
1480	attention	node_embedding	3.0	1.0	0.740
1629	bias_attention_reset	node_embedding	9.0	1.0	0.740
1554	attention_reset	node_embedding	6.0	2.0	0.742
1483	attention_reset	node_embedding	3.0	3.0	0.745
1555	attention_reset	node_embedding	6.0	3.0	0.745
1481	attention_reset	node_embedding	3.0	1.0	0.745
1484	bias_attention	node_embedding	3.0	1.0	0.747
1327	bias_attention_reset	agent_start	9.0	3.0	0.748
1559	bias_attention_reset	node_embedding	6.0	3.0	0.748
1703	bias_attention_reset	node_embedding	18.0	3.0	0.748
1557	bias_attention_reset	node_embedding	6.0	1.0	0.751
1334	bias_attention_reset	last_agent_visited	9.0	2.0	0.756
1553	attention_reset	node_embedding	6.0	1.0	0.761
1697	attention_reset	node_embedding	18.0	1.0	0.763
1556	bias_attention	node_embedding	6.0	1.0	0.765

PubMed

For this dataset the model was trained with all possible combinations of these arguments:

- dims = [16,32,64]
- training dropout rate = [0.0,0.3]
- learning rates = [0.01,0.001,0.0001]
- steps = [3,6,9,18]
- num agents = [n] (n = number of nodes in graph)
- learning rates = [0.01,0.001,0.0001]

- readout strategies = [node_embedding,last_agent_visited,agent_start]
- transition strategies = [attention, attention_reset]
- initialization strategies = [one_to_one]
- neighborhood_reset_sizes = [1,2,3]

Table C.5: Computation results of AgentNet on PubMed

	transition strat.	readout strat.	agent steps	n.hood reset	test acc
595	attention_reset	last_agent_visited	3.0	3.0	0.677
699	attention_reset	last_agent_visited	18.0	1.0	0.688
412	attention	last_agent_visited	18.0	1.0	0.697
630	attention_reset	last_agent_visited	6.0	2.0	0.716
628	attention	last_agent_visited	6.0	1.0	0.717
697	attention_reset	agent_start	18.0	2.0	0.719
76	attention	last_agent_visited	9.0	1.0	0.719
396	attention	agent_start	18.0	1.0	0.720
626	attention_reset	agent_start	6.0	2.0	0.722
187	attention_reset	last_agent_visited	6.0	3.0	0.723
836	attention_reset	node_embedding	18.0	3.0	0.725
36	attention	agent_start	6.0	1.0	0.725
414	attention_reset	last_agent_visited	18.0	2.0	0.727
592	attention	last_agent_visited	3.0	1.0	0.727
589	attention_reset	agent_start	3.0	1.0	0.728
308	attention	node_embedding	3.0	1.0	0.729
701	attention_reset	last_agent_visited	18.0	3.0	0.730
372	attention	agent_start	9.0	1.0	0.734
377	attention_reset	last_agent_visited	9.0	1.0	0.735
625	attention_reset	agent_start	6.0	1.0	0.736
375	attention_reset	agent_start	9.0	3.0	0.737
373	attention_reset	agent_start	9.0	1.0	0.739
634	attention_reset	node_embedding	6.0	2.0	0.739
591	attention_reset	agent_start	3.0	3.0	0.740
590	attention_reset	agent_start	3.0	2.0	0.743
629	attention_reset	last_agent_visited	6.0	1.0	0.745
93	attention_reset	node_embedding	9.0	1.0	0.747
155	attention_reset	node_embedding	3.0	3.0	0.749
310	attention_reset	node_embedding	3.0	2.0	0.749
662	attention_reset	agent_start	9.0	2.0	0.749
761	attention	node_embedding	6.0	1.0	0.749
696	attention_reset	agent_start	18.0	1.0	0.749
799	attention_reset	node_embedding	9.0	2.0	0.750
379	attention_reset	last_agent_visited	9.0	3.0	0.750
191	attention_reset	node_embedding	6.0	3.0	0.751
726	attention_reset	node_embedding	3.0	1.0	0.752
627	attention_reset	agent_start	6.0	3.0	0.755
588	attention	agent_start	3.0	1.0	0.756
593	attention_reset	last_agent_visited	3.0	1.0	0.757
835	attention_reset	node_embedding	18.0	2.0	0.757
666	attention_reset	last_agent_visited	9.0	2.0	0.757
698	attention_reset	agent_start	18.0	3.0	0.757
774	attention_reset	node_embedding	6.0	1.0	0.761

Continued on next page

Table C.5: Computation results of AgentNet on PubMed

	transition strat.	readout strat.	agent steps	n.hood reset	test acc
809	attention	node_embedding	9.0	1.0	0.764
227	attention_reset	node_embedding	9.0	3.0	0.765
845	attention	node_embedding	18.0	1.0	0.767
294	attention_reset	last_agent_visited	3.0	2.0	0.768
846	attention_reset	node_embedding	18.0	1.0	0.770

OGB-Arxiv

For this dataset the model was trained with all possible combinations of these arguments:

- dims = [32,64,128]
- training dropout rate = [0.0,0.3]
- learning rates = [0.01,0.001,0.0001]
- steps = [3,6,9,18]
- num agents = [n] (n = number of nodes in graph)
- learning rates = [0.01,0.001,0.0001]
- readout strategies = [node_embedding,last_agent_visited,agent_start]
- transition strategies = [attention, attention_reset]
- initialization strategies = [one_to_one]
- neighborhood_reset_sizes = [1,2,3]

Table C.6: Computation results of AgentNet on OGB

	transition strat.	readout strat.	agent steps	n.hood reset	test acc
110	attention	last_agent_visited	18.0	1.0	0.037385
88	attention	last_agent_visited	9.0	1.0	0.485711
52	attention	last_agent_visited	6.0	1.0	0.531099
55	attention_reset	last_agent_visited	6.0	3.0	0.570520
77	attention_reset	last_agent_visited	9.0	1.0	0.570582
238	attention	last_agent_visited	3.0	1.0	0.571117
241	attention_reset	last_agent_visited	3.0	3.0	0.590663
203	attention_reset	last_agent_visited	9.0	3.0	0.594408
220	attention	agent_start	18.0	1.0	0.614077
109	attention	node_embedding	18.0	1.0	0.620970
239	attention_reset	last_agent_visited	3.0	1.0	0.621237

Continued on next page

Table C.6: Computation results of AgentNet on OGB

	transition strat.	readout strat.	agent steps	n.hood reset	test acc
95	attention_reset	node_embedding	9.0	3.0	0.632842
194	attention_reset	node_embedding	9.0	2.0	0.633233
184	attention	agent_start	9.0	1.0	0.634632
192	attention	node_embedding	9.0	1.0	0.637101
1	attention	agent_start	6.0	1.0	0.637759
186	attention_reset	agent_start	9.0	2.0	0.639035
193	attention_reset	node_embedding	9.0	1.0	0.640783
151	attention_reset	agent_start	6.0	3.0	0.643643
159	attention_reset	node_embedding	6.0	3.0	0.645598
4	attention	node_embedding	6.0	1.0	0.646688
150	attention_reset	agent_start	6.0	2.0	0.648581
185	attention_reset	agent_start	9.0	1.0	0.649013
245	attention_reset	node_embedding	3.0	3.0	0.652799
269	attention	node_embedding	3.0	1.0	0.653252
158	attention_reset	node_embedding	6.0	2.0	0.654651
187	attention_reset	agent_start	9.0	3.0	0.658972
157	attention_reset	node_embedding	6.0	1.0	0.660926
154	attention_reset	last_agent_visited	6.0	2.0	0.662325
282	attention_reset	node_embedding	3.0	1.0	0.665597
190	attention_reset	last_agent_visited	9.0	2.0	0.666626
261	attention	agent_start	3.0	1.0	0.667099
244	attention_reset	node_embedding	3.0	2.0	0.670761
2	attention_reset	agent_start	6.0	1.0	0.671605
264	attention_reset	agent_start	3.0	3.0	0.677057
263	attention_reset	agent_start	3.0	2.0	0.678353
153	attention_reset	last_agent_visited	6.0	1.0	0.680493
262	attention_reset	agent_start	3.0	1.0	0.682345
267	attention_reset	last_agent_visited	3.0	2.0	0.685986

Appendix D

Results - Node Regression - PageRank

We decided to only list the best resulting runs when selecting unique combinations of parameters and omitting the other parameters such as learning rate or dropout.

For Baseline-GCN the best unique combinations of ['hidden units','num layers'] are listed. For AgentNet the best unique combinations of ['number of agents','neighborhood reset','transition strat.','readout strat.'] are listed.

D.1 Baseline-GCN

For all datasets the model was trained with all possible combinations of these arguments:

- dims = [4,8,16]
- dropout rate in GCN Layer:= [0.0,0.3]
- learning rates:= [0.01,0.001,0.0001]
- num_layers:= [1,2,3,4]

Cora

Table D.1: Computation results of Baseline-GCN on Cora

	hidden units	dropout	lr	num layers	test loss	spear coeff.
15	4.0	0.3	0.0100	4.0	8.054900e-08	0.3454
59	16.0	0.0	0.0001	4.0	1.200000e-03	0.3880
44	8.0	0.3	0.0001	1.0	2.950000e-02	0.7409
56	16.0	0.0	0.0001	1.0	2.950000e-02	0.7409

Continued on next page

Table D.1: Computation results of Baseline-GCN on Cora

	hidden units	dropout	lr	num layers	test loss	spear coeff.
0	4.0	0.0	0.0100	1.0	1.768300e-08	0.7409
1	4.0	0.0	0.0100	2.0	8.214400e-08	0.8585
2	4.0	0.0	0.0100	3.0	6.343300e-07	0.9199
25	8.0	0.0	0.0100	2.0	3.110600e-07	0.9597
65	16.0	0.3	0.0010	2.0	2.926400e-05	0.9632
38	8.0	0.3	0.0100	3.0	4.206000e-08	0.9666
54	16.0	0.0	0.0010	3.0	1.972600e-05	0.9826
39	8.0	0.3	0.0100	4.0	6.895500e-07	0.9864

PubMed

Table D.2: Computation results of Baseline-GCN on PubMed

	hidden units	dropout	lr	num layers	test loss	spear coeff.
64	16.0	0.3	0.0010	1.0	4.911100e-07	0.4905
16	4.0	0.3	0.0010	1.0	4.911100e-07	0.4905
36	8.0	0.3	0.0100	1.0	6.244000e-10	0.4905
15	4.0	0.3	0.0100	4.0	2.194300e-07	0.5138
63	16.0	0.3	0.0100	4.0	3.401600e-06	0.6007
1	4.0	0.0	0.0100	2.0	1.164100e-06	0.8387
38	8.0	0.3	0.0100	3.0	1.524300e-06	0.8922
45	8.0	0.3	0.0001	2.0	6.000000e-04	0.9333
47	8.0	0.3	0.0001	4.0	1.100000e-03	0.9497
65	16.0	0.3	0.0010	2.0	4.164500e-05	0.9565
14	4.0	0.3	0.0100	3.0	6.914700e-07	0.9642
50	16.0	0.0	0.0100	3.0	8.628800e-06	0.9673

OGB-Arxiv

Table D.3: Computation results of Baseline-GCN on OGB

	hidden units	dropout	lr	num layers	test loss	spear coeff.
48	16.0	0.0	0.0100	1.0	2.691100e-11	0.2852
4	4.0	0.0	0.0010	1.0	2.353600e-08	0.2852
44	8.0	0.3	0.0001	1.0	8.240000e-02	0.2852
1	4.0	0.0	0.0100	2.0	1.234300e-06	0.2901
15	4.0	0.3	0.0100	4.0	1.550000e-07	0.2957
42	8.0	0.3	0.0010	3.0	1.138800e-06	0.3104
25	8.0	0.0	0.0100	2.0	1.294400e-08	0.3136
70	16.0	0.3	0.0001	3.0	5.000000e-04	0.3169
71	16.0	0.3	0.0001	4.0	3.000000e-04	0.3174
14	4.0	0.3	0.0100	3.0	9.915900e-09	0.3193
35	8.0	0.0	0.0001	4.0	3.800000e-03	0.3198

61	16.0	0.3	0.0100	2.0	3.157400e-07	0.3248

D.2 AgentNet

For all datasets the model was trained with all possible combinations of these arguments:

- dims = [8,16]
- training dropout rate = [0.3]
- learning rates = [0.01,0.001,0.0001]
- steps = [5,10,20]
- num agents = [0.25n,0.5n,n,1.5n,2n] (n = number of nodes in graph)
- learning rates = [0.01,0.001,0.0001]
- readout strategies = [node_embedding,last_agent_visited]
- transition strategies = [attention, attention_reset, random]
- initialization strategies = [random]
- neighborhood_reset_sizes = [2,3,4]

Cora

Table D.4: Computation results of AgentNet on Cora

	transition strat.	readout strat.	agent steps	num agents	n.hood reset	spear coeff.
6	random	last_agent_visited	5.0	677.0	3.0	0.3433
193	random	last_agent_visited	10.0	1354.0	3.0	0.3747
200	random	last_agent_visited	20.0	2708.0	3.0	0.3948
56	attention_reset	node_embedding	10.0	1354.0	3.0	0.5053
93	random	node_embedding	20.0	677.0	3.0	0.5315
204	attention_reset	node_embedding	10.0	1354.0	4.0	0.5687
107	attention_reset	node_embedding	20.0	5416.0	3.0	0.6125
197	attention_reset	node_embedding	10.0	2708.0	2.0	0.6201
382	random	node_embedding	5.0	1354.0	3.0	0.6319
196	attention_reset	node_embedding	10.0	1354.0	2.0	0.6447
96	attention	node_embedding	20.0	1354.0	2.0	0.6470
255	attention_reset	node_embedding	20.0	5416.0	4.0	0.6510
138	attention_reset	node_embedding	20.0	4062.0	3.0	0.6568
95	random	node_embedding	20.0	2708.0	3.0	0.6693
142	attention_reset	node_embedding	20.0	4062.0	4.0	0.6695

Continued on next page

Table D.4: Computation results of AgentNet on Cora

	transition strat.	readout strat.	agent steps	num agents	n.hood reset	spear coeff.
102	attention_reset	node_embedding	20.0	4062.0	2.0	0.6907
242	attention	node_embedding	20.0	4062.0	2.0	0.6947
109	attention_reset	node_embedding	20.0	2708.0	4.0	0.7176
195	attention	node_embedding	10.0	5416.0	2.0	0.7404
97	attention	node_embedding	20.0	2708.0	2.0	0.7448
105	attention_reset	node_embedding	20.0	2708.0	3.0	0.7931
247	attention_reset	node_embedding	20.0	5416.0	2.0	0.8528

PubMed

Table D.5: Computation results of AgentNet on PubMed

	transition strat.	readout strat.	agent steps	num agents	n.hood reset	spear coeff.
38	random	last_agent_visited	20.0	19717.0	3.0	0.4656
325	random	last_agent_visited	5.0	9858.0	3.0	0.5048
90	random	last_agent_visited	20.0	4929.0	3.0	0.5648
252	attention_reset	node_embedding	20.0	9858.0	4.0	0.6413
0	attention	node_embedding	5.0	9858.0	2.0	0.6569
270	attention_reset	node_embedding	20.0	29575.0	4.0	0.6633
250	attention_reset	node_embedding	20.0	29575.0	3.0	0.6779
315	random	node_embedding	20.0	4929.0	3.0	0.6786
105	attention_reset	node_embedding	20.0	19717.0	3.0	0.6828
364	random	node_embedding	20.0	9858.0	3.0	0.6972
8	attention_reset	node_embedding	5.0	9858.0	3.0	0.7050
109	attention_reset	node_embedding	20.0	19717.0	4.0	0.7059
5	attention_reset	node_embedding	5.0	19717.0	2.0	0.7159
4	attention_reset	node_embedding	5.0	9858.0	2.0	0.7210
198	attention_reset	node_embedding	10.0	29575.0	2.0	0.7269
159	attention_reset	node_embedding	5.0	39434.0	4.0	0.7299
242	attention	node_embedding	20.0	29575.0	2.0	0.7363
251	attention_reset	node_embedding	20.0	39434.0	3.0	0.7513
241	attention	node_embedding	20.0	19717.0	2.0	0.7531
199	attention_reset	node_embedding	10.0	39434.0	2.0	0.7734
419	random	node_embedding	20.0	19717.0	3.0	0.7961
243	attention	node_embedding	20.0	39434.0	2.0	0.8166

OGB-Arxiv

Table D.6: Computation results of AgentNet on OGB

	transition strat.	readout strat.	agent steps	num agents	n.hood reset	spear coeff.
42	random	last_agent_visited	20.0	42335.0	3.0	0.0666
362	random	last_agent_visited	20.0	169343.0	3.0	0.1073
295	random	last_agent_visited	10.0	84671.0	3.0	0.1090
0	attention	node_embedding	5.0	84671.0	2.0	0.1444
45	random	node_embedding	20.0	42335.0	3.0	0.1555

Continued on next page

Table D.6: Computation results of AgentNet on OGB

	transition strat.	readout strat.	agent steps	num agents	n.hood reset	spear coeff.
88	random	node_embedding	10.0	84671.0	3.0	0.2104
54	attention_reset	node_embedding	20.0	84671.0	2.0	0.2746
76	attention_reset	node_embedding	10.0	84671.0	4.0	0.2777
52	attention	node_embedding	20.0	254014.0	2.0	0.3076
56	attention_reset	node_embedding	20.0	84671.0	3.0	0.3171
106	attention_reset	node_embedding	20.0	254014.0	3.0	0.3188
71	attention_reset	node_embedding	5.0	169343.0	2.0	0.3214
48	attention_reset	node_embedding	5.0	254014.0	4.0	0.3285
15	attention_reset	node_embedding	10.0	169343.0	4.0	0.3318
59	attention_reset	node_embedding	5.0	169343.0	3.0	0.3337
65	attention_reset	node_embedding	5.0	338686.0	4.0	0.3499
102	attention_reset	node_embedding	20.0	254014.0	2.0	0.3534
107	attention_reset	node_embedding	20.0	338686.0	3.0	0.3593
99	attention	node_embedding	20.0	338686.0	2.0	0.3625
103	attention_reset	node_embedding	20.0	338686.0	2.0	0.3970
293	random	node_embedding	10.0	169343.0	3.0	0.4759
49	attention	node_embedding	10.0	169343.0	2.0	0.4912