Distributed Learning with Graph Neural Networks

Alejandro Ribeiro, Zhan Gao and Alejandro Parada Mayorga

November 12, 2020

Graph neural networks (GNNs) explore the irregular structure of graph signals, and exhibit superior performance in various applications of recommendation systems, wireless networks and control. A key property GNNs inherit from graph filters is the distributed implementation. This property makes them suitable candidates for distributed learning over large-scale networks, where global information is not available at individual agents. Each agent must decide its own actions from local observations and communication with immediate neighbors. In this lab assignment, we focus on the distributed learning with graph neural networks.

1 Distributed Control Systems

We consider a team of N agents tasked with controlling a dynamical process that is evolving in the space where the team is deployed. The system is distributed because the agents are located in different positions; see Figure 1. We characterize this dynamical process by the collection of state values $\mathbf{x}_i(t) \in \mathbb{R}^p$ as well as the collection of control actions $\mathbf{u}_i(t) \in \mathbb{R}^q$ that agents take. In this notation, i represents an agent index and t represents continuous time. We further group local states and local actions into matrices $\mathbf{X}(t) = [\mathbf{x}_1(t), \ldots, \mathbf{x}_N(t)]^T \in \mathbb{R}^{N\times p}$ and $\mathbf{U}(t) = [\mathbf{u}_1(t), \ldots, \mathbf{u}_N(t)]^T \in \mathbb{R}^{N\times q}$ that represent the global state of the system and the global control action at time t. Observe that *row* n of $\mathbf{X}(t)$ is the state associated with agent n and that *row* n of $\mathbf{U}(t)$ is the control action taken by agent n.

With these definitions we write the evolution of the dynamical process through a differential equation of the form,

$$\dot{\mathbf{X}}(t) = f\Big(\mathbf{X}(t), \mathbf{U}(t)\Big). \tag{1}$$

In order to design a controller for this dynamical system we operate in discrete time by introducing a sampling time T_s and a discrete time index n. We define $\mathbf{X}_n = \mathbf{X}(nT_s)$ as the discrete time state and $\mathbf{U}_n$ as the control action held from time nT_s until time $(n+1)T_s$. Solving the differential equation between times nT_s and $(n+1)T_s$, we end up with the discrete dynamical system

$$\mathbf{X}_{n+1} \;=\; \mathbf{X}_n \;+\; \int_{nT_s}^{(n+1)T_s} f\Big(\mathbf{X}(t), \mathbf{U}_n\Big)\; dt, \qquad \text{with } \mathbf{X}(nT_s) = \mathbf{X}_n. \tag{2}$$

At each point in (discrete) time, we consider a cost function $c(\mathbf{X}_n, \mathbf{U}_n)$. The objective of the control system is to choose actions $\mathbf{U}_n$ that reduce the accumulated cost $\sum_{n=0}^{\infty} c(\mathbf{X}_n, \mathbf{U}_n)$. When the collection of state observations $\mathbf{X}_n = [\mathbf{x}_{1n}^T; \ldots; \mathbf{x}_{Nn}^T]$ is available at a central location it is possible for us to consider centralized policies Π that choose control actions $\mathbf{U}_n = \Pi(\mathbf{X}_n)$ that depend on global information. In such case the optimal policy is the one that minimizes the expected long term cost,

$$\Pi_{\mathrm{C}}^{*} = \underset{\Pi}{\operatorname{argmin}}\, \mathbb{E}\left[\sum_{n=0}^{\infty} c\Big(\mathbf{X}_n, \Pi(\mathbf{X}_n)\Big)\right]. \tag{3}$$

If the dynamics in $f(\mathbf{X}(t), \mathbf{U}(t))$ and the costs $c\big(\mathbf{X}_n, \mathbf{U}_n\big)$ are known, as we assume here, there are several techniques to find the optimal policy Π^*. Our interest in this exercise is in decentralized controllers that operate without access to global information. In those situations, implementing (9) is not possible because of delays in percolating information through the distributed system as we explain in the following section.

1.1 Decentralized Control

In distributed systems we face the challenge of collecting information. Implementing the policy in (9) requires knowledge of the global system state $\mathbf{x}(n)$. This can't be done if we consider systems where the number of agents N is large. When N is large we leverage the locality of the problem by relying on communication between nearby agents.

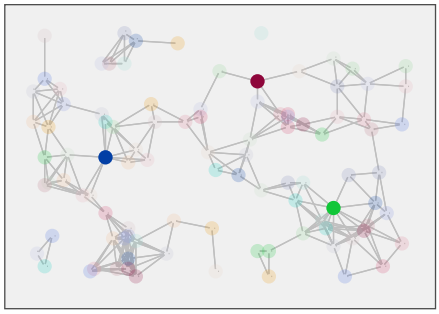

Figure 1.

We therefore consider a communication network described with a set of edges $(i,j) \in \mathcal{E}_n$. The presence of the edge (i,j) in the set $\mathcal{E}_n$ indicates that j may send data to i at time n. This set may vary over time. When this happens we say that j is a neighbor of i and define the neighborhood of i at time n as the collection of all its neighbors,

$$\mathcal{N}_{in} = \{j : (i,j) \in \mathcal{E}_n\}. \tag{4}$$

We can also define multi-hop neighborhoods of a node. To do so begin by convening that the 0-hop neighborhood of i is $\mathcal{N}_{in}^0 = \{i\}$; namely, the node itself. Further rename the neighborhood of n as the 1-hop neighborhood and denote $\mathcal{N}_{in}^1 = \mathcal{N}_{in}$. We can now define the k-hop neighborhood of i as the set of nodes that can reach node i in exactly k hops. Their formal definition can be given by the recursion

$$\mathcal{N}_{in}^k = \left\{ j' \in \mathcal{N}_{j(n-1)}^{k-1}, \text{ such that } j \in \mathcal{N}_{in} \right\}. \tag{5}$$

That is, the set of k-hop neighbors of i is the set of nodes that were $(k-1)$-hop neighbors at time $n-1$ of the nodes that are neighbors of i at time n.

This neighborhood definition in (5) characterizes the information that is available to node i at time n; see Figure 2. This information includes the local state $\mathbf{x}_{in}$ that can be directly observed by node i at time n. It also includes the value of the state $\mathbf{x}_{j(n-1)}$ of all the nodes that are 1-hop neighbors of i at time n. But these state observations make it to node i with a delay of 1 time step. Node i can observe the state $\mathbf{x}_{j(n-1)}$, not the state $\mathbf{x}_{jn}$. Communication is not instantaneous. It is also possible for node i to observe the states $\mathbf{x}_{j(n-2)}$ of its 2-hop neighbors. This is because the

information can be relayed from 1-hop neighbors. But the information makes it to node i with two units of delay. It is the state $\mathbf{x}_{j(n-2)}$, not the state $\mathbf{x}_{jn}$ that is available at i for its 2-hop neighbors. We can repeat this argument for general k-hop neighborhoods to define the information history $\mathcal{H}_{in}$ of node i at time n as the collection of state observations out to a maximal history depth K

$$\mathcal{H}_{in} = \bigcup_{k=0}^{K-1} \Big\{ \mathbf{x}_{j(n-k)}, \text{ such that } j \in \mathcal{N}_{in}^{k} \Big\}. \tag{6}$$

The challenge in decentralized control is that it is impossible for agent i to implement the controller in (9) with the information available in (6). We can extend the history depth K to the diameter of the network and make sure we observe states for all nodes. But the information received by node i is delayed. For a k hop neighbor node i has access to the state $\mathbf{x}_{j(n-k)}$ corresponding to discrete time index $n-k$. Implementing (9) needs access to the state $\mathbf{x}_{jn}$ at the current time index n.

The information history in (6) indicates that the control action of node i must be of the form,

$$\mathbf{u}_{in} = \pi_{in}(\mathcal{H}_{in}). \tag{7}$$

Using the local control actions in (7), which depend on local information histories, results in the distributed control policies

$$\Pi_{\mathrm{D}n} = \Big[\pi_1(\mathcal{H}_{1n}), \ldots, \pi_N(\mathcal{H}_{Nn}) \Big]^T. \tag{8}$$

And therefore leads to the replacement of the optimal control problem in (9) by the optimal control problem

$$\Pi_{\mathrm{D}n}^{*} = \underset{\Pi_{\mathrm{D}n}}{\operatorname{argmin}} \, \mathbb{E}\left[\sum_{n=0}^{\infty} c\Big(\mathbf{X}_n, \Pi_{\mathrm{D}n} \Big) \right], \tag{9}$$

in which the optimization is over control polciies that respect the information structure of the distributed system. The problem in (9) is famously difficult to solve except in some particularly simple scenarios

The challenge arises from the need to define policies that depend on information histories. This creates an exponential complexity that is typical of systems that lack the Markov memoryless property. The complexity of finding the decentralized policy $\Pi_{\mathrm{D}n}^{*}$ of (9) vis-à-vis the relative simplicity of finding the optimal centralized policy Π_{C}^{*} of (9) motivates the

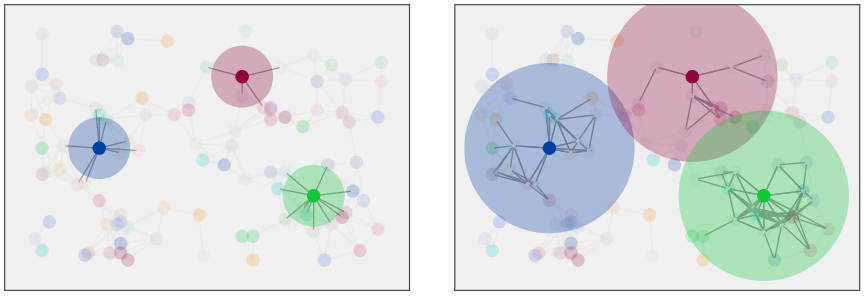

Figure 2.

introduction of a method that learns to mimic the global controller. We describes this in the following section.

1.2 Imitation of Centralized Policies

Introduce a parameterized policy $\pi(\mathcal{H}_{in}, \mathbf{A})$ that maps local information histories $\mathcal{H}_{in}$ to local actions $\mathbf{u}_{in} = \pi(\mathcal{H}_{in}, \mathbf{A})$. We consider global policies in which all the agents execute policies $\pi(\mathcal{H}_{in}, \mathbf{A})$ with the same parameter $\mathbf{A}$.

$$\Pi(\mathbf{A}) = \Big[\pi(\mathcal{H}_{1n}, \mathbf{A}), \ldots, \pi(\mathcal{H}_{in}, \mathbf{A})\Big]^{T}. \tag{10}$$

We point out that in (10) it is not immediately apparent how we can reuse the same parameter $\mathbf{A}$ at all nodes. Neighborhood cardinalities can be different for instance and the orderings of neighbors are arbitrary. We will resolve this issue with the use of graph filters and graph neural networks in Section 3 but we set aside this concern until then.

The policy in (10) is, by deign, one that respects the information structure of the decentralized policies defined in (8). To mimic the centralized policy in (9) we consider a loss function $\mathcal{L}(\Pi(\mathbf{A}), \Pi_{\mathrm{C}}^{*})$ to measure the difference between the optimal centralized policy Π_{C}^{*} and a system where all agents (locally) execute the (local) policy $\pi(\mathcal{H}_{in}, \mathbf{A})$. Our goal is to find the tensor of parameter $\mathbf{A}$ that solves the optimization problem

$$\mathbf{A}^{*} = \underset{\mathbf{A}}{\operatorname{argmin}}\, \mathbb{E}^{\Pi_{\mathrm{C}}^{*}}\Big[\mathcal{L}\Big(\pi\big(\mathcal{H}_{in}, \mathbf{H}\big), \pi^{*}(\mathbf{x}_{n})\Big)\Big], \tag{11}$$

where we use the notation $\mathbb{E}^{\Pi_{\mathrm{C}}^{*}}$ to emphasize that the distribution of ob-

served states $\mathbf{x}_n$ over which we compare the parametrized policy with the optimal centralized policy $\Pi(\mathbf{A})$ and Π_{C}^* is that of a system that follows the optimal policy Π_{C}^*.

The formulation in (11) is one in which we want to learn a local policy $\pi(\mathcal{H}_{in}, \mathbf{A})$ that mimics the rows of the centralized policy Π^* to the extent that this is possible with the information that is available to each individual node. The success of this effort depends on the appropriate choice of the parameterization that determines the family of policies $\pi(\mathcal{H}_{in}, \mathbf{H})$ that can be represented by different choices of parameters, $\mathbf{H}$. In this exercise we explore the use of graph filters and graph neural network (Section 3). We demonstrate their applications to the problem of network consensus (Section 2) and to the problem of flocking with collision avoidance (Section 4).

2 Network Consensus

To ground the discussion in Section 1 we consider a consensus problem. The goal of a consensus problem is to compute an average over a network. It is not a difficult problem. In fact, one would venture that it is elementary. Yet, the solution in a decentralized setting becomes complicated. It therefore serves as a good illustration of the challenges of decentralized control.

To be more specific consider a reference velocity $\mathbf{r}_n = \mathbf{r}(nT_s) \in \mathbb{R}^2$. This reference velocity evolves according to a random acceleration model that we write in discrete time as

$$\mathbf{r}_{n+1} = \mathbf{r}_n + T_s \Delta \mathbf{r}_n. \tag{12}$$

In (12), the acceleration term $\Delta \mathbf{r}_{in}$ is white Gaussian with independent components and expected norm $\mathbb{E}(\|\Delta \mathbf{r}_n\|)$[1]. The initial reference velocity $\mathbf{r}_0$ is initialized with a normal distribution with zero mean independent components and expected norm $\mathbb{E}(\|\mathbf{r}_0\|)$. The values we will use for these and other parameters in numerical experiments are shown in Table 3.

[1] A Gaussian random variable X in two dimensions with independent components, each of which has variance σ^2, has a expected norm $\mathbb{E}(\|X\|) = \sigma\sqrt{\pi/2}$. To generate a Gaussian with expected norm $\mathbb{E}(\|X\|)$ we use variances $\sigma^2 = (2/\pi)\mathbb{E}^2(\|X\|)$ in each component.

We consider a team of n agents whose goal is to estimate this reference velocity. However, the agents do not observe the reference directly. Each of them observes a biased version of the reference velocity according to the model

$$\mathbf{r}_{in} = \mathbf{r}_n + \Delta\mathbf{r}_i. \tag{13}$$

The bias terms $\Delta\mathbf{r}_i$ of each agent are white Gaussian with independent components and expected norm $\mathbb{E}(\|\Delta\mathbf{r}_i\|)$. Notice that the bias term does not change over time. It is a systematic error in the velocity measurements of each agent.

The reference velocities in (13) are observations of agents. In addition to these observations, agents have velocity estimates $\mathbf{v}_{in}$ initialized as $\mathbf{v}_{in} = \mathbf{r}_0 + \Delta\mathbf{v}_i$, where the velocity estimate bias $\Delta\mathbf{v}_i$ is white Gaussian with independent components and expected norm $\mathbb{E}(\|\Delta\mathbf{v}_i\|)$. These velocities evolve according to the dynamical model

$$\mathbf{v}_{i(n+1)} = \mathbf{v}_{in} + T_s\mathbf{u}_{in}, \tag{14}$$

in which $\mathbf{u}_{in}$ is a control acceleration that is to be chosen by agent i. The goal of the agents is to make this estimate as close as possible to the reference velocity $\mathbf{r}_n$. This velocity, however, is not observed by the team. We therefore set to find a policy that controls for the cost function

$$\tilde{c}(\mathbf{V}_n, \mathbf{R}_n) = \frac{1}{2N}\sum_{i=1}^{N}\left\| \mathbf{v}_{in} - \frac{1}{N}\sum_{j=1}^{N}\mathbf{r}_{jn}\right\|^2. \tag{15}$$

According to (15), the goal is for each agent to find out velocities $\mathbf{v}_{in}$ that coincide with the average reference velocity $(1/N)\sum_{j=1}^{N}\mathbf{r}_{jn}$ observed by the team. If it helps to fix ideas, you can think of a flock that wants to move with the wind. They each observe the wind with some systematic error and collaborate to follow the average of their wind estimates.

To minimize the cost in (15) we just need to get the swarm to follow the average. Namely, to make $\mathbf{v}_{in} = (1/N\sum_{j=1}^{N}\mathbf{r}_{jn})$. To make the problem more interesting we ask for a balance between velocity reduction and energy. The latter depends on the acceleration $\mathbf{u}_{in}$. We therefore consider a relative weight λ and define the cost

$$c(\mathbf{V}_n, \mathbf{R}_n, \mathbf{U}_n) = \frac{1}{2N}\sum_{i=1}^{N}\left\| \mathbf{v}_{in} - \frac{1}{N}\sum_{j=1}^{N}\mathbf{r}_{jn}\right\|^2 + \frac{\lambda}{2N}\sum_{i=1}^{N}\left\| T_s\mathbf{u}_{in}\right\|^2. \tag{16}$$

Parameter	Symbol	Value
Initial reference velocity	$\mathbb{E}(\|\mathbf{r}_0\|)$	$1\,\mathrm{m/s}$
Reference velocity acceleration	$\mathbb{E}(\|\Delta\mathbf{r}_n)\|)$	$1\,\mathrm{m/s^2}$
Reference velocity agent bias	$\mathbb{E}(\|\Delta\mathbf{r}_i\|)$	$4\,\mathrm{m/s}$
Initial velocity estimate bias	$\mathbb{E}(\|\Delta\mathbf{v}_i\|)$	$1\,\mathrm{m/s}$
Maximum agent acceleration	$\mathbb{E}(\|\mathbf{u}_0\|)$	$3\,\mathrm{m/s^2}$
Sample time	T_s	$0.1\,\mathrm{s}$
Energy weight	λ	1
Initial agent density	ρ	$0.005\,\mathrm{agents/m^2}$
Network degree	d	2 agents
Collision avoidance weight	γ	$10\,\mathrm{m}$
Reference distance	d_0	$2\,\mathrm{m}$
Collision avoidance weight	μ	1

Figure 3. Parameter values.

In the general problem definition we introduced in Section 1 we consider the cost over a time horizon [cf. eqref(9)]. To keep the problem simple we consider here a greedy formulation where at time step n we want to minimize the cost $c(\mathbf{V}_{n+1}, \mathbf{R}_{n+1}, \mathbf{U}_{n+1})$ at the next time step. This result in a simple expression for the optimal control policy in which the acceleration of agent i is given by

$$\mathbf{u}_{in}^* = \frac{-1}{T_s(1+\lambda)} \left(\mathbf{v}_{in} - \frac{1}{N} \sum_{j=1}^{N} \mathbf{r}_{jn} \right). \tag{17}$$

The solution in (17) computes the difference between the velocity of agent i and the average reference velocity and accelerates in the opposite direction. The amount of acceleration is controlled by λ. The larger λ, the less we accelerate.

Question 2.1: System simulation. Create a class that simulates the system described by (12)-(14). The parameters in the table in Figure 3 are attributes of this class. The class also records the number of agents N as

an attribute.

The class has a method to simulate the generation of reference velocities as per (12) and (13). This method admits a time horizon T so that the system is simulated for T/T_s iterations.

The class has another method to simulate the evolution of velocities as per (14). The methods takes the system state $\mathbf{V}_n$ and the control action $\mathbf{U}_n$ as inputs and produces the next state $\mathbf{V}_{n+1}$ as an output.

Test the class with $N = 100$ and $T = 10\,\text{s}$, and the cost is averaged over trajectory time instances.

Question 2.2: Implementation of centralized Consensus. Implement the optimal controller in (17). Plot velocity trajectories $\mathbf{v}_{in}$ and compare them with the average reference velocity $(1/N \sum_{j=1}^{N} \mathbf{r}_{jn})$.

2.1 Decentralized Consensus

The consensus problem we have just described becomes more interesting when formulated in a distributed setting. To do that, assume that agents are connected through a communication network. As in Section 1.2 the network is described by a set of edges $(i, j) \in \mathcal{E}_n$ which induce the neighborhoods $\mathcal{N}_{in}$ defined in (4). With a graph restricting the exchange of information, the implementation of (17) becomes impossible.

An important point to repeat is that it becomes difficult to design appropriate controllers. Our goal is still the minimization of the cost in (16) but this is difficult to do with the partial information we are given. Faced with the impossibility of finding optimal controllers, we can resort to approximations. A natural idea is to forget about the fact that information is delayed and just evaluate (17) with the information we have. We therefore define the averages

$$\bar{\mathbf{r}}_{jn}^{k} = \frac{1}{\#\left(\mathcal{N}_{in}^{k}\right)} \sum_{j \in \mathcal{N}_{in}^{k}} \mathbf{r}_{j(n-k)}, \tag{18}$$

which represent the average velocity of the k-hop neighbors of node i corrected by appropriated delays. Using these averages we can now ap-

proximate the controller in (17) as

$$\mathbf{u}_{in}^{\dagger} = \frac{-1}{T_s(1+\lambda)} \left(\mathbf{v}_{in} - \frac{1}{K} \sum_{k=1}^{K} \bar{\mathbf{r}}_{jn}^{k} \right). \tag{19}$$

The controller defined by (18) and (19) is decentralized because it respects the information history structure we defined in (6). The values that are needed for implementing (19) are available at node i because they have been relayed through neighbors. That the controller can be implemented does not mean it will be any good. We are taking averages of out of date information.

Question 2.3: Communication Network. We consider a scenario in which n transmitters are dropped in an area of dimensions w_x by w_y. The transmitters are dropped uniformly at random. An important parameter of the network is the agent density, defined as

$$\rho = (w_x w_y) \,/\, n. \tag{20}$$

We construct a network model in which agent i and agent j communicate with each other if and only if agent j is among the d agents closest to i *or* agent i is among the d agents closest to agent j. We refer to d as the network degree, even though d is not exactly the average degree of the connectivity graph. A node i is connected to its d closest agents and it it is also connected to all the agents j for which i is one of its d closest agents.

Write a class whose attributes are the the dimensions w_x and w_y along with the number of nodes n and the network degree d. These numbers are provided at initialization. Add attributes to represent the positions of agents and receivers. These are generated at initialization with transmitters and receivers placed at random as described above. This class is similar to the one you programmed for Lab 4.

Add an attribute to represent the unweighted adjacency matrix of a connectivity graph that follows the generating model we described above. Add a method that calculates the adjacency matrix when requested.

Test with the parameters in the table in Figure 3 along with $w_x = 200\,\text{m}$ and $w_y = 100\,\text{m}$.

Question 2.4: Decentralized Consensus Implement the decentralized

consensus controller in (19). Leverage the classes you constructed in Questions 2.1 and 2.3.

In your decentralized controller you can choose different values of K. Try several values. Plot velocity trajectories $\mathbf{v}_{in}$ and compare them with the average reference velocity $(1/N \sum_{j=1}^{N} \mathbf{r}_{jn})$. Compare with the optimal centralized controller in Question 2.2. It should be much worse irrespectively of your choice of K.

The fact that we are referring to $\mathbf{v}_{in}$ as the velocity of agent i at time n hints at moving agents. We are going to go there in Section 4, but for the time being we are neglecting the fact that agents are moving. This is important because as agents move the communication network changes. This is not an issue except that it increases the cost of your simulation due the the necessity to recompute the communication graph.

3 Learning Distributed Controllers

In Section 1.2 we explained the inherent complexity of decentralized control. In Section 2.1 we illustrated that these inherent complexities lead to important differences in practical performance. One can attribute the performance penalty to the fact that (19) forgets the fact that it is averaging information that can be out of date. This is true, but how can we correct for that? Say we choose to discount older information. This is a sensible idea but how do we choose discount factors? The problem is not that decentralized control becomes impossible to solve. But since finding the optimal controller *is* impossible, we are left to choose among many possible heuristics.

If we are settling for heuristics, there is no reason to overlook the use of learned heuristics. We use our models as a source of simulated data and fit a learned parametrization. This is the idea we described in Section 1.2. The goal of mimicking the optimal centralized policy means that, at least in principle, the learned heuristic can learn an optimal controller. This is something we give away in designed heuristics.

If we decide to settle for learned heuristics, our job as system designers is not complete. We should still choose the parametrization. Since we are considering a problem where a network plays a central role, the choice of

parametrization is clear: We should choose graph filters and graph neural networks (GNNs). In the case of distributed systems there is a second, more important reason for us to use graph filters and GNNs: They can be implemented in a distributed manner while respecting the information structure of distributed control. We explain this in the next section.

3.1 Distributed Implementation of Graph Filters and GNNs

To explain the use of graph filters and GNNs in distributed systems we start by recalling our definition of MIMO graph filters as polynomials on the graph shift operator $\mathbf{S}$.

$$\mathbf{Z} = \sum_{k=0}^{K} \mathbf{S}^k \mathbf{X} \mathbf{H}_k. \tag{21}$$

In the case of distributed systems the matrix graph signal $\mathbf{X}$ is evolving over time index n. We therefore introduce a time varying version of (21) in which the kth term of the polynomial multiplies the signal $\mathbf{X}_{n-k}$

$$\mathbf{Z}_n = \sum_{k=0}^{K} \mathbf{S}^k \mathbf{X}_{n-k} \mathbf{H}_k. \tag{22}$$

The reason for introducing this modification in (22) is that it is now possible to provide a distributed implementation that respects the information history structure we introduced in (6). To see how this is done we express (22) in terms of a modified diffusion sequence. This diffusion sequence is initialized as $\mathbf{Y}_{n0} = \mathbf{X}_n$ and its subsequent elements are computed recursively as

$$\mathbf{Y}_{nk} = \mathbf{S} \mathbf{Y}_{(n-1)(k-1)}. \tag{23}$$

Applying this definition recursively we have that $\mathbf{Y}_{nk} = \mathbf{S}^k \mathbf{Y}_{(n-k)0}$. Further recalling the initialization condition we have that $\mathbf{Y}_{(n-k)0} = \mathbf{X}_{n-k}$. Thus, $\mathbf{Y}_{nk} = \mathbf{S}^k \mathbf{X}_{n-k}$ and we have that the the filter in (22) can be equivalently written as a summation of elements of the diffusion sequence,

$$\mathbf{Z}_n = \sum_{k=0}^{K} \mathbf{Y}_{nk} \mathbf{H}_k \tag{24}$$

To show that we can implement (24) in a distributed manner, it suffices if we show that each of the $\mathbf{Y}_{nk}$ terms can be evaluated through local

computation exchanges. To see that this is true recall that the shift operator shares the sparsity pattern of the graph. Thus, when we look at the computation of the ith entry of $\mathbf{Y}_{nk}$ we can write

$$\left[\mathbf{Y}_{nk}\right]_i = \sum_{j \in \mathcal{N}_{ni}} \left[\mathbf{S}\right]_{ij} \times \left[\mathbf{Y}_{(n-1)(k-1)}\right]_j. \tag{25}$$

According to (25), node i can compute the ith entry of $\mathbf{Y}_{nk}$ if it receives information from neighboring agents j about the values of the jth entries of the $\mathbf{Y}_{(n-1)(k-1)}$. This is something that is available to node i because it can be communicated from its neighboring agents j. These are, by definition, the agents with which i can communicate.

If a graph filter can be implemented in a distributed manner, a GNN can be as well. This is because the nonlinearity is pointwise and can be implemented locally. Having seen that a GNN can respect the information structure of distributed systems, we can use them as learning parametrizations in decentralized control. This is the goal of the next question.

Question 3.1: Learning with a GNN Use a GNN to learn a distributed policy that mimics the optimal centralized controller in Question 2.2. Plot velocity trajectories $\mathbf{v}_{in}$ and compare them with the average reference velocity $(1/N \sum_{j=1}^{N} \mathbf{r}_{jn})$. Compare with the optimal centralized controller in Question 2.2 and with the decentralized controller in Question 2.4.

Given we have been working with GNNs for a while, you are free to play with the choice of shift operator and GNN architecture. An example of an architecture that works well is to use $\mathbf{R}_n$ and $\mathbf{V}_n$ as input features, have $L = 2$ layers with $F_1 = 64$ features and $F_2 = 2$ features. In Layer 1 you use filters of order $K_1 = 4$ and in Layer 2 filters of order $K_2 = 1$. The 2 features at the output of Layer 2 are mapped to the control inputs $\mathbf{u}_{in}$. The normalized adjacency is used as a shift operator.

As in Question 2.4 we disregard the movement of the agents.

4 Agent Mobility

We modify the problem in Section 2 to incorporate agent mobility. Denote as $\mathbf{p}_{in}$ the position of agent i at time n and augment the dynamical model

in (14) to incorporate the dynamics that control this variable,

$$\mathbf{p}_{i(n+1)} = \mathbf{p}_{in} + T_s \mathbf{v}_{in} + (T_s^2/2)\mathbf{u}_{in}. \tag{26}$$

When we incorporate agent mobility the first aspect of the problem that changes is that the network is no longer fixed. As agents move, their set of nearest neighbors change. This is consistent with our generic problem description in Section 1.2. To make our graph filters consistent with a time varying network we just need to modify the definition of the diffusion sequence to take time varying networks into consideration. We thus replace (22) by

$$\mathbf{Y}_{nk} = \mathbf{S}_n \mathbf{Y}_{(n-1)(k-1)}, \tag{27}$$

where $\mathbf{S}_n$ is the shift operator of the communication network at time n.

Question 4.1: Mobility. Modify the class in Question 2.1 to incorporate positions and mobility according to the dynamical model in (26).

Implement the controller you learnt in Question 3.1 in a system with mobility. This requires that you recompute the communication network at every step. Or every so often if you want a speedier simulation. This is a good moment to create a class that inherits from this class and the class that simulates the communication network in (26).

Question 4.2: It works so well. Despite the fact that you trained without mobility, the implementation with mobility in (4.1) has a small degradation in cost. This is something we knew was going to happen. Why?

Question 4.3: Training with Mobility. Retrain the GNN of Question 4.1 using the time varying networks that result from the incorporation of mobility.

5 Collision and Spread Avoidance

The second aspect of the problem that changes when we incorporate mobility is that agents can get too close to each other, which would result in a collision. Agents can also get too far apart, which would result in their inability to communicate. To prevent both outcomes we utilize a potential function to encourage neighboring agents to stay within a target distance of each other. Formally, let $\mathbf{p}_{in}$ be the position of agent i at time n and

$\mathbf{p}_{jn}$ be the position of agent j. Agents j is a neighbor of agent i. We then define the collision avoidance potential

$$U(\mathbf{p}_{in}, \mathbf{p}_{jn}) = \begin{cases} \frac{d_0^2}{\|\mathbf{p}_{in}-\mathbf{p}_{jn}\|^2} - \log\left[\frac{\|\mathbf{p}_{in}-\mathbf{p}_{jn}\|^2}{d_0^2}\right], & \text{if } \|\mathbf{p}_{in}-\mathbf{p}_{jn}\| \leq \gamma \\ \frac{d_0^2}{\gamma^2} - \log\left[\frac{\gamma^2}{d_0^2}\right], & \text{otherwise} \end{cases} \tag{28}$$

where γ is the collision avoidance weight. This potential function has a minimum when the distance between agents i and j is $\|\mathbf{p}_{in} - \mathbf{p}_{jn}\| = d_0$ and is indifferent when $\|\mathbf{p}_{in} - \mathbf{p}_{jn}\| > \gamma$.

The potentials $U(\mathbf{p}_{in}, \mathbf{p}_{jn})$ in (28) are added to the cost function in (16) to define the loss function

$$\ell(\mathbf{P}_n, \mathbf{V}_n, \mathbf{R}_n, \mathbf{U}_n) = c(\mathbf{V}_n, \mathbf{R}_n, \mathbf{U}_n) + \frac{\mu}{2N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} U(\mathbf{p}_{in}, \mathbf{p}_{jn}). \tag{29}$$

The optimal centralized controller is the one that greedily minimizes the cost in (29) at each time step, which is given by

$$\mathbf{u}_{in}^* = \frac{-1}{T_s(1+\lambda)} \left(\mathbf{v}_{in} - \frac{1}{N} \sum_{j=1}^{N} \mathbf{r}_{jn} \right) - \frac{\mu}{2T_s} \sum_{j=1}^{N} \nabla_{\mathbf{p}_{in}} U\left(\mathbf{p}_{in}, \mathbf{p}_{jn}\right) \tag{30}$$

for $i = 1, \ldots, N$. Our goal in this section is to learn a GNN controller that mimics this optimal centralized controller.

Question 5.1: Relative mass. Before we set out to design this GNN, we define the following feature at node i,

$$\mathbf{q}_{in} = \sum_{j \in \mathcal{N}_{in}} \left(\mathbf{p}_{in} - \mathbf{p}_{jn}\right). \tag{31}$$

This is the mass of the neighbors of i measured relative to the position of node i. In this definition, it is not very important that $\mathbf{q}_{in}$ be a center of mass. But is is *very* important that the measure be *relative* so as to preserve permutation equivariance. Explain

Question 5.2: Learning with a GNN. Use a GNN to learn a distributed policy that mimics the optimal centralized controller that minimizes the cost in (29). The input features to the GNN are $\mathbf{V}_n$, $\mathbf{R}_n$, and $\mathbf{Q}_n$. Compare the cost with the cost of the optimal centralized controller.

The GNN architecture of Question 3.1 works well for this problem. But you should play around with different architectural choices.

6 Report

Do not take much time to prepare a lab report. We do not want you to report your code and we don't want you to report your work. Just give us answers to questions we ask. Specifically give us the following:

Question	**Report deliverable**
Question 2.1	Do not report.
Question 2.2	Plot with 10 representative trajectories. Cost averaged over 100 simulations.
Question 2.3	Do not report.
Question 2.4	Plot with 10 representative trajectories. Cost averaged over 100 simulations.
Question 3.1	GNN architecture parameters. Plot with 10 representative trajectories. Cost averaged over 100 simulations.
Question 4.1	Plot with 10 representative trajectories. Cost averaged over 100 simulations.
Question 4.2	One Paragraph.
Question 4.3	GNN architecture parameters. Plot with 10 representative trajectories. Cost averaged over 100 simulations.
Question 5.1	One Paragraph.
Question 5.2	GNN architecture parameters. Plot with 10 representative trajectories. Cost averaged over 100 simulations.

We will check that your answers are correct. If they are not, we will get back to you and ask you to correct them. As long as you submit responses, you get an A for the assignment. It counts for 16% of your final grade.