



HAL
open science

Comparing machine learning techniques for predicting glassy dynamics

Rinske Alkemade, Emanuele Boattini, Laura Filion, Frank Smallenburg

► **To cite this version:**

Rinske Alkemade, Emanuele Boattini, Laura Filion, Frank Smallenburg. Comparing machine learning techniques for predicting glassy dynamics. *The Journal of Chemical Physics*, 2022, 156 (20), pp.204503. <10.1063/5.0088581>. <hal-03870000>

HAL Id: hal-03870000

<https://hal.science/hal-03870000v1>

Submitted on 24 Nov 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



HAL Authorization

Comparing machine learning techniques for predicting glassy dynamics

Rinske M. Alkemade,¹ Emanuele Boattini,¹ Laura Filion,¹ and Frank Smallenburg²

¹*Soft Condensed Matter, Debye Institute of Nanomaterials Science, Utrecht University, Utrecht, Netherlands*

²*Université Paris-Saclay, CNRS, Laboratoire de Physique des Solides, 91405 Orsay, France*

In the quest to understand how structure and dynamics are connected in glasses, a number of machine learning based methods have been developed that predict dynamics in supercooled liquids. These methods include both increasingly complex machine learning techniques, and increasingly sophisticated descriptors used to describe the environment around particles. In many cases, both the chosen machine learning technique and choice of structural descriptors are varied simultaneously, making it hard to quantitatively compare the performance of different machine learning approaches. Here, we use three different machine learning algorithms – linear regression, neural networks, and GNNs – to predict the dynamic propensity of a glassy binary hard-sphere mixture using as structural input a recursive set of order parameters recently introduced by Boattini *et al.* [Phys. Rev. Lett. **127**, 088007 (2021)]. As we show, when these advanced descriptors are used, all three methods predict the dynamics with nearly equal accuracy. However, the linear regression is orders of magnitude faster to train making it by far the method of choice.

The relationship between local structure and dynamics in glassy systems has been a heavily debated question in condensed matter for several decades^{1–3}. Over the last few years, one avenue for exploring this relationship has been an effort to predict dynamical behavior based on local structural features using various machine learning (ML) methods. Pioneered by Cubuk *et al.*⁴ using support vector machines (SVMs) to predict rearrangement probabilities in glassy mixtures, this area of research has now embraced a wide variety of ML techniques, including e.g. linear regression, convolutional neural networks (CNNs), graph neural networks (GNNs), autoencoders, and community inference, see e.g. Refs. 5–10. This raises the question of what ML technique one should choose when predicting the dynamics of a glassy system.

This question is far from straightforward, since in addition to choosing a machine learning technique, one also has to make a choice with respect to the encoding of the local structure in terms of data that can be interpreted by an ML algorithm. For most ML approaches, the structure around a particle is encoded into a set of structural order parameters that capture e.g. the local density and symmetry of the distribution of neighbors around that particle. However, some sophisticated ML approaches can work from much more restricted data. For example, in Ref. 5, it was shown that graph neural networks are capable of predicting the dynamic propensity of a glassy Lennard-Jones mixture based solely on encoding the structure into a graph of nearest neighbors and the pair distances between them. Such data would not be sufficient for e.g. a simple linear regression approach, but using a GNN it was enough to drastically outperform both SVMs and CNNs trained with more sophisticated input. More recently, however, it was shown that even a simple linear regression approach can rival the predictive power of GNNs if supplied with sufficiently intelligent input data⁸. In particular, Boattini *et al.* proposed a method to iteratively construct generations of structural order parameters that successively take into account locally averaged order in expanding shells. These descriptors, in combination with linear regression performed essentially as well as a GNN which was fed just the particle coordinates.

This raises the intriguing question of whether more sophisticated ML techniques supplied with more intelligently cho-

sen structural parameters can result in even better predictions. Here, we use three different ML algorithms – linear regression, neural networks, and GNNs – to predict the dynamic propensity of a glassy binary hard-sphere mixture, based on the hierarchical set of order parameters from Ref. 8, and compare and contrast the results. As we show, out of the three methods, linear regression provides the best compromise between accuracy and efficiency when combined with these advanced structural descriptors.

I. MODEL AND DESCRIPTORS

A. Model

The glassy system that we use here to compare the three different ML methods is a binary hard-sphere mixture at packing fraction $\eta = 0.58$. It consists of hard spheres of two sizes, with a size ratio of $\sigma_S/\sigma_L = 0.85$, where $\sigma_{L(S)}$ is the diameter of a large (small) particle. The composition $x_L = N_L/(N_L + N_S) = 0.3$, where $N_{L(S)}$ is the number of large (small) spheres. Note that this is the same glassy mixture as was studied in Refs. 6, 8, and 11.

We simulate the evolution of our system using event-driven molecular dynamics (EDMD)¹². The simulations are performed in the microcanonical ensemble (constant number of large and small particles N_L and N_S , volume V , and kinetic energy U). The time unit of our simulation is defined as $\tau = \sqrt{m\sigma_L^2/k_B T}$ where k_B is Boltzmann's constant and m is the particle mass. Note that we set the masses of all particles to be equal. All simulated systems contained 2000 particles in total (600 large, 1400 small).

To generate the initial configurations, we use a separate EDMD simulation in which the particles grow over time until the desired packing fraction $\eta = 0.58$ is reached. After this, the system is equilibrated for at least $10^5 \tau$. Note that from previous work⁶, we know that the relaxation time of this system is on the order of $\tau_\alpha = 10^4 \tau$.

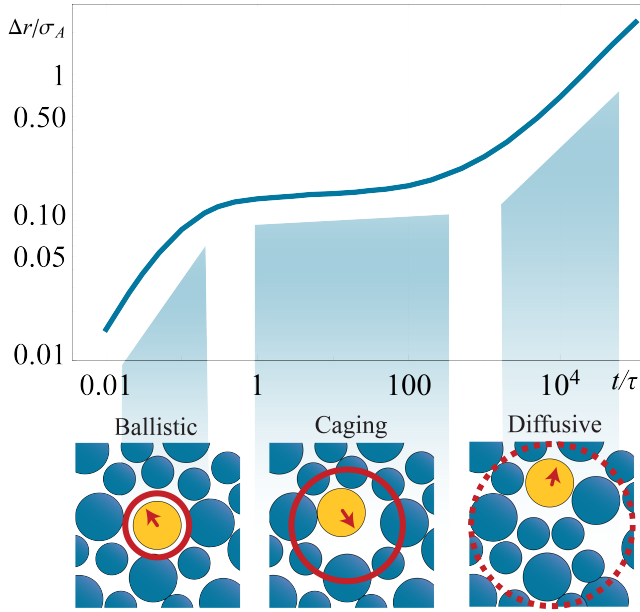


FIG. 1. Globally averaged dynamical propensity $\langle \Delta r_i(t) \rangle$ as a function of time. The insets at the bottom illustrate three different dynamical regimes: the ballistic regime, in which particles are not yet interacting with their neighbors, the caging regime, in which particles are trapped by surrounding particles, and the diffusive regime, where particles have escaped their cages.

B. Dynamic propensity

To characterize the dynamical heterogeneity in our glassy system, we use the dynamic propensity^{13,14}. This quantity is closely related to the mean squared displacement, and captures how far individual particles on average move over time. To measure the dynamic propensity, the evolution of a glassy system is measured multiple times, each time starting from the same initial configuration, while assigning each particle a random velocity drawn from a Maxwell-Boltzmann distribution at the desired temperature. This ensemble is called the isoconfigurational ensemble¹³. To obtain the propensity $\Delta r_i(t)$ of particle i , we average the absolute distance it traveled over the time interval t over all trajectories

$$\Delta r_i(t) = \langle |\mathbf{r}_i(t) - \mathbf{r}_i(0)| \rangle_{\text{iso}}, \quad (1)$$

where $\langle \cdot \rangle_{\text{iso}}$ indicates the average taken over all trajectories in the isoconfigurational ensemble.

To measure the propensity, we average over simulations starting from 100 different initial snapshots, and for each initial snapshot we average over 50 trajectories with different initial velocities. The dynamic propensity is measured at a logarithmically spaced set of time intervals t , between $t/\tau = 10^{-2}$ and $t/\tau = 10^5$.

To illustrate the behavior of the dynamic propensity, we plot in Fig. 1 the globally averaged dynamic propensity as a function of time.

C. Structural descriptors

To describe the local environments of particles, we use the structural order parameters used in Ref. 8, which consist of a combination of radial densities and angular functions measured in different shells around a particle.

For the radial functions, we use essentially the same descriptors as used in Refs. 5, 8, and 9. These descriptors measure a weighted particle density inside a spherical shell with a thickness of approximately 2δ at distance r with respect to a reference particle i . The functions are defined as

$$G_i^{(0)}(r, \delta, s) = \sum_{j \neq i, s_j = s} e^{-\frac{(r_{ij}-r)^2}{2\delta^2}}. \quad (2)$$

Here i is the reference particle, s is the particle type and r_{ij} is the distance between particles i and j . The summation is carried out over all particles with particle type $s_j = s$, which means that the radial density that is measured is type-specific.

The angular descriptors that we use are based on bond order parameters^{15,16}. These bond order parameters expand the local environment in terms of spherical harmonics. To obtain the angular descriptors for a particle i , we first calculate the complex coefficients

$$q_i^{(0)}(l, m, r, \delta) = \frac{1}{Z} \sum_{j \neq i} e^{-\frac{(r_{ij}-r)^2}{2\delta^2}} Y_l^m(\mathbf{r}_{ij}). \quad (3)$$

Here $Y_l^m(\mathbf{r}_{ij})$ is the spherical harmonic of order l , with m an integer that runs from $-l$ to l , and Z is a normalization factor given by

$$Z = \sum_{j \neq i} e^{-\frac{(r_{ij}-r)^2}{2\delta^2}}. \quad (4)$$

Note that although the summation runs over all particles, again the exponent makes sure that mainly particles within a spherical shell at distance r and thickness 2δ will contribute to $q_i^{(0)}(l, m, r, \delta)$. Finally we sum over m to obtain the rotationally invariant angular descriptors

$$q_i^{(0)}(l, r, \delta) = \sqrt{\frac{4\pi}{2l+1} \sum_{m=-l}^{m=l} |q_i^{(0)}(l, m, r, \delta)|^2}. \quad (5)$$

Due to the symmetries of the spherical harmonics, $q^{(0)}(l, r, \delta)$ for a certain l is expected to detect l -fold symmetry in the environment at the chosen distance r .

Boattini *et al.*⁸ showed that the propensity prediction of a particle improves significantly, when the prediction is based not only on the structural parameters associated with the particle itself, but also on averaged structural information of neighbouring particles. Inspired by the architecture of graph neural networks, this was done by recursively constructing higher-order averaged structural parameters, which are defined as

$$x_i^{(n)} = \frac{\sum_{j: r_{ij} < r_c} x_j^{(n-1)} e^{-r_{ij}/r_c}}{\sum_{j: r_{ij} < r_c} e^{-r_{ij}/r_c}}, \quad (6)$$

where x_i can be any of the radial or angular order parameters of particle i . Additionally, $x_i^{(n)}$ represents the n^{th} generation of parameter x_i , and the sum runs over all neighboring particles within a cutoff distance r_c , including i itself. The cutoff value r_c is chosen to be $r_c/\sigma_L = 2.1$, which approximately corresponds to the second minimum of the radial distribution function⁸. However, as already shown in Ref. 8, the exact value does not have a significant influence on our results.

In total, we consider 354 0th-generation structural descriptors: 162 radial descriptors and 192 angular descriptors. For the radial descriptors we use 46 equally spaced spherical shells in the interval $r/\sigma_L = [0.86, 2.0]$, 20 equally spaced spherical shells in the interval $r/\sigma_L = (2.0, 3.0]$ and 15 equally spaced spherical shells in the interval $r/\sigma_L = (3.0, 4.5]$. For the angular descriptors we consider $l = 1$ to 12 in 16 equally spaced spherical shells in the interval $[1, 2.5]$. The full environment of each particle is then described with up to 3 generations of these 354 parameters each, leading to a total of 1062 parameters.

Before using the structural parameters as an input for the machine learning algorithms, they are standardized by evaluating

$$\mathbf{x}_i^{\text{st}} = \frac{\mathbf{x}_i - \bar{\mathbf{x}}}{\sigma_{\mathbf{x}}}, \quad (7)$$

where \mathbf{x}_i is the vector containing all parameters associated with particle i , \mathbf{x}_i^{st} is the standardized parameter vector, and where $\bar{\mathbf{x}}$ and $\sigma_{\mathbf{x}}$ are respectively the mean and standard deviation of the parameter vector considering all particles of the same species as i . The standardization ensures that all descriptors have zero mean and unit variance, which can be helpful when using regularization in machine learning techniques. This will be discussed in more detail below.

II. MACHINE LEARNING METHODS

In this paper, we compare three different machine learning approaches for predicting the dynamic propensity based on the structural parameters introduced above. In particular, we compare linear regression (LR), neural networks (NN), and graph neural networks (GNN). Unless otherwise specified, we train separate models for large and small particles, and separate models for each time interval at which we are trying to predict the dynamic propensity.

Note that each of these approaches has a number of hyperparameters that tune the model fitted by the method to the supplied training data. For example, this can be the number of layers inside the neural network, parameters controlling regularization techniques that reduce overfitting, or the learning rate of the optimization algorithm for NNs and GNNs.

A. Linear regression

Linear regression is the simplest of the three methods, and simply finds the best linear combination of all structural de-

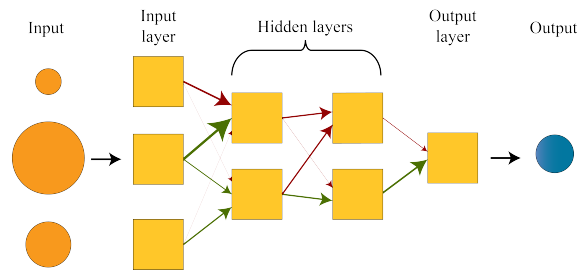


FIG. 2. A Neural network consisting of an input layer, hidden layers and an output layer. The orange and blue circles represent input and output parameters respectively. The red and green arrows represent the weights that connect each layer in the neural network.

scriptors to predict the dynamic propensity. Here, we make use of L2-regularization (also known as Ridge regression) to reduce overfitting¹⁷. This approach penalizes large weights in the linear fit. Note that this is the reason we standardized our structural parameters: since the different parameters have the same mean and variance, the effect of the regularization on each parameter is the same. For linear regression the only hyperparameter that can be tuned is α , which sets the strength of the large-weight penalty in Ridge regression.

B. Neural network

Neural networks are loosely based on the biological neural networks that make up our brains. A neural network consists of multiple layers of connected nodes, see Fig. 2, which mimic the neurons and synapses in the brain. The first and last layer are respectively called the input and output layer, which in our case take the structural parameters of each particle as an input, and give the predicted propensity for a certain time as the output. All the layers that lie between the input and output layer are called hidden layers. In a fully connected feed-forward neural network, as we use here, each node in a specific layer is connected to all the nodes in the following layer.

Due to the connections between nodes, information can be passed through the neural network. Each connection between nodes is associated with a so called ‘weight’. The values of the nodes in the hidden layer and the output layer are found by multiplying the values of the nodes in the previous layer with the associated weights and adding a bias. The result is then passed to a non-linear function, in our case a Rectified Linear Unit (ReLU)¹⁷, to yield the value of the node. Hence, the value of a node a_m in layer l is calculated as

$$a_m^{(l)} = f \left(\sum_n w_{mn}^{(l)} a_n^{(l-1)} + b_m^{(l)} \right). \quad (8)$$

Here f is the ReLU function, $w_{mn}^{(l)}$ is the weight associated with the connection between nodes n and m , $a_n^{(l-1)}$ is the information of node n in layer $l-1$ and $b_m^{(l)}$ is the bias associated

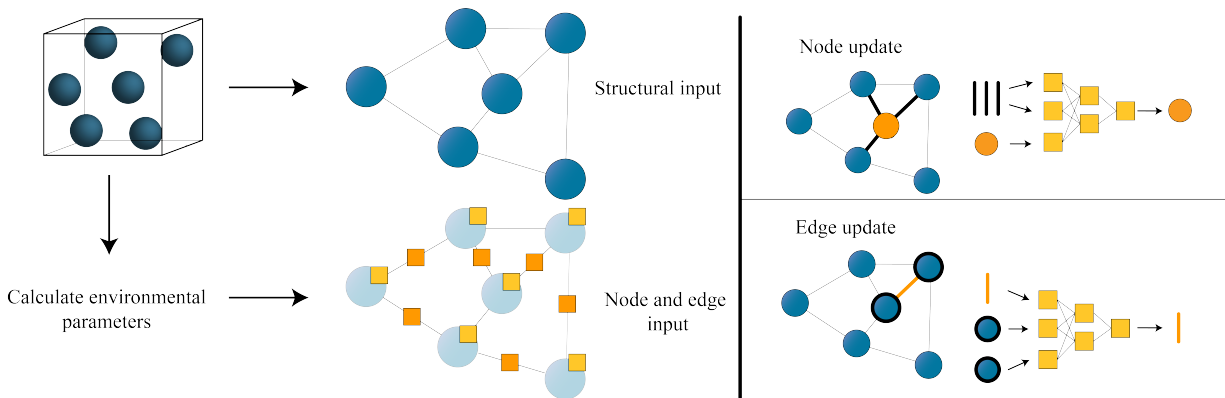


FIG. 3. (left) Mapping between the initial configuration data and the graph structure. From the configuration, we determine nearest neighbors in order to set up a graph structure, and then calculate structural parameters to fill in the data at each node and edge. (right) Node and edge update in a graph layer. To update the node data, the added and averaged parameters of the connected edges are given to the node neural network, together with the information of the node itself. For the edge update, the information of both neighbouring nodes, together with the edge information itself is given to the edge neural network. The output of the node/edge network is an updated value of the node/edge.

with node $a_m^{(l)}$. The summation over n goes over all nodes in layer $l - 1$.

We train the neural networks using the Python package Pytorch¹⁸, and in particular use an Adam optimizer¹⁹. This optimizer is an extension to the stochastic gradient descent procedure, and is used to find an efficient path to a locally optimal set of weights and biases via backpropagation¹⁷.

For neural networks there are many more hyperparameters that can be tuned, including the number of layers, and the number of nodes in each layer, as well as parameters associated with the learning process, such as the learning rate and the batch size. As the neural network turns out to be very sensitive to overfitting, we also run the neural network with ridge regression and drop-out²⁰.

C. Graph neural networks

GNNs^{21–23} are a relatively new class of machine learning techniques that combine neural networks with a graph-like data structure. As there are many variations of GNNs, our description is necessarily somewhat specific to the GNN we use in this paper.

In contrast to LR and NNs, which consider each particle as a separate training example, the GNN takes in an entire configuration at once. As a result, the GNN does not make a propensity prediction on a single-particle basis, but instead simultaneously makes a prediction for all particles (of a chosen species) in a configuration of the system.

In a GNN, input data is first mapped to a data structure which consists of a graph that holds numerical data at its nodes and edges (see Figure 3). In our case, each node corresponds to a particle in the configuration for which we are trying to predict propensities. When two particles are closer to each other than a certain distance $r_c = 2.1\sigma_L$, the corresponding nodes in the graph are connected by an edge.

As an input, every node holds the structural parameters of the corresponding particle as well as its species, and each edge

holds information about the distance between the two connected particles. Like a neural network, a GNN corresponds to a non-linear function of its input parameters, that gets calculated in multiple consecutive layers, called graph layers. Each layer itself is a non-linear function, which takes as an input a graph structure containing information on all nodes and edges, and outputs a new graph structure with the same graph topology but updated numerical data at the nodes and edges. For each node and edge, the update that takes place within this graph layer takes into account not only the information already at that node or edge, but also data from neighboring edges or nodes, as illustrated in Fig. 3. The internal functions that perform this update consist of fully connected feed-forward neural networks as described above. In addition to these graph layers, the full GNN incorporates an encoder layer, which maps the input node data to the data structure used in the graph layers, and a decoder, which makes a propensity prediction for each particle of a chosen species, based on the updated node data. Both of these layers are standard feed-forward neural networks. Note that analogous to the GNN in Ref. 5, our GNN additionally provides each graph layer with information about the graph data output by the encoder. In other words, a node update takes into account i) the node's current values, ii) the aggregated current values of all edges connected to the node, and iii) the node's values as they were just after the encoder layer.

The core benefit of the GNN is that the prediction for the propensity of a given particle can include information about the structure of multiple shells of neighboring particles. The distance over which information is included can be controlled by the number of included graph layers. As such, the GNN inherently includes the feature of recursively considering the average local structure of shells of neighbors.

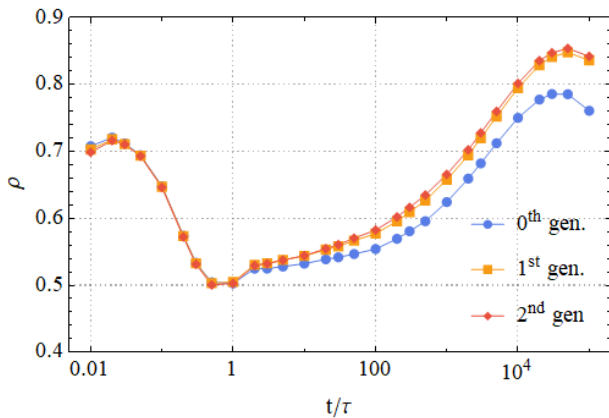


FIG. 4. Pearson’s correlation coefficient ρ between the dynamic propensity as predicted by linear regression and measured in simulations as a function of time, analyzed for three different generations of order parameters. Results are shown for the large particles.

III. RESULTS

In this section we first look at each method independently, and explore the influence of the different hyperparameter settings, and the inclusion of different generations of structural order parameters. In all cases, to compare the predictions to the measured propensity we use the Pearson correlation coefficient.

A. Linear regression

The only hyperparameter for linear regression is the regularization parameter α . As optimizing the fit for a single parameter is trivial, here we only present results that correspond to the best choice of α , optimized between 10^{-5} and 10^4 for the large particles. In Figure 4 we show the linear regression performance for different generations of order parameters. In all cases, when we refer to a generation, we include all lower generations as well. Note that these results are consistent with Ref. 8. We clearly see that the predictions from the zeroth generation of descriptors are significantly worse than the ones including higher-generation data, at least for longer times. In particular, we see that the information of the higher-order generations only starts to influence the performance when the system enters the caging regime. This is what we expected: before entering the caging regime not enough time has passed for particles to be influenced by particles from further away, meaning that higher-order generations will not add relevant information about the expected trajectories. Although adding the second generation of order parameters still improves the predictions for the propensity, the effect is small in comparison to the improvement of adding the first generation. Adding even higher generations (not shown here) does not significantly improve the performance beyond this point⁸.

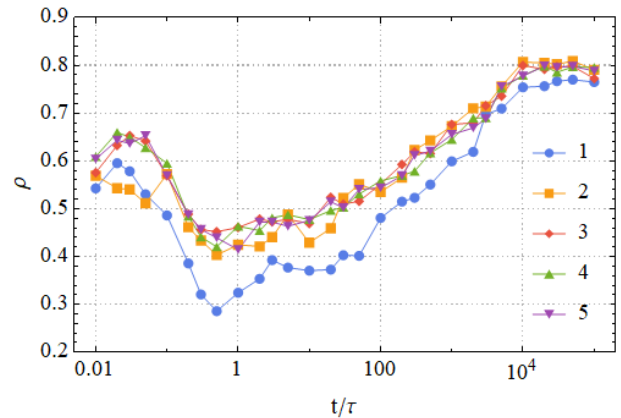


FIG. 5. Correlation between the dynamic propensity as predicted by neural networks and measured in simulations as a function of time, analyzed for different hyperparameters as specified in Table I. Results are shown for the large particles.

Number	Batch size	Learning rate	Hidden layers	Drop-out	α
1	50	10^{-4}	1 (16)	0	0
2	50	10^{-4}	3 (16,16,16)	0	0
3	50	10^{-4}	2 (16,16)	0	1.0
4	50	10^{-4}	3 (16,16)	0.25	0
5	50	10^{-4}	3 (16,16)	0.25	0.01

TABLE I. Hyperparameters for different neural networks used to predict the propensity. Each row corresponds to a line in Fig. 5. The *Hidden layers* entry contains both information about the number of hidden layers in the network, as well as the number of nodes in each hidden layer (shown in brackets). *Drop-out* shows the fraction of nodes in the second hidden layer that is set to zero during drop-out and α represents the value of the parameter associated with L_2 -regularization.

B. Neural networks

For the neural networks, in addition to the number of generations and regularization parameter, there are many hyperparameters to optimize. Since trying out all possible combination of settings would not be feasible, we limit ourselves to a small number of different combinations, shown in Table I. The network is trained on large and small particles separately in around 500 epochs. In order to limit overfitting, after each epoch we evaluate the performance of the NN on the test data set, and eventually use the network that had the highest correlation for this test data. In total for each time interval and for each generation of structural parameters, we trained five networks. Their performance is shown in Fig. 5. Here, all networks are trained using three generations of structural parameters.

Although the overall behavior of the neural network accuracy over time is similar to that found for linear regression, we see significant variation in performance between the different hyperparameter choices. In particular, we see that only using a single NN layer (blue line) or no regularization (blue and

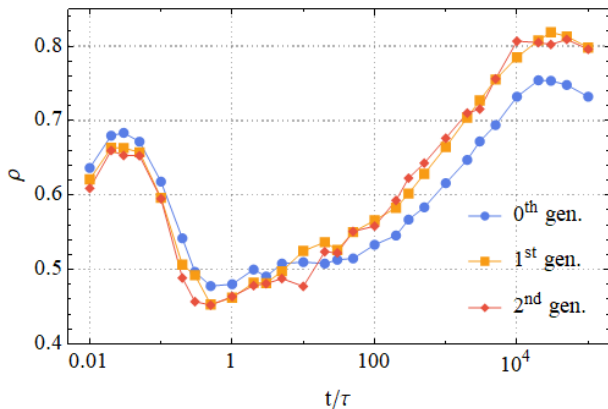


FIG. 6. Correlation between the dynamic propensity as predicted by neural networks and measured in simulations as a function of time, analyzed for three different generations of order parameters. Results are shown for the large particles.

yellow lines) lead to worse performance, especially at shorter times. However, once we include at least two NN layers and regularization, the results are relatively robust: the other three lines essentially coincide except for noise.

In order to examine the influence of different generations of order parameters on the performance, in Fig. 6 we show the performance of the NN for different generations, with the hyperparameters optimized for each time interval. Contrary to what we saw in the case of linear regression, providing the NN with more generations of order parameters does not always improve the performance, something that is especially clear for short times. This is likely the result of the higher number of weights and biases that the NN training needs to optimize when higher-order descriptors are included.

C. Graph neural networks

In Ref. 5, Bapst *et al.* demonstrated that hyperparameters did not play a large role in the accuracy of their GNNs for predicting dynamic propensity. Since the training of a GNN is considerably more expensive than a normal neural network, here we do not focus on optimizing the hyperparameters, but instead consider a few different variations of supplying data to the GNN. As a baseline, we consider a GNN with four graph layers, which considers the zero'th generation of structural parameters as the node inputs, and the absolute distance between particles as the edge inputs. The GNN is trained separately to predict the propensities of the large and small particles, but takes the information of all particles into account for both trainings. As a variation on this baseline, we also consider i) a GNN that predicts both the propensity of both the large and small species simultaneously, ii) a GNN that incorporates all three generations as node data, and iii) a GNN that uses the x , y , and z components of the vectors between neighboring particles as edge data. A complete summary of all relevant (hyper)parameters is shown in Table II.

In Fig. 7, we show the performance of the different vari-

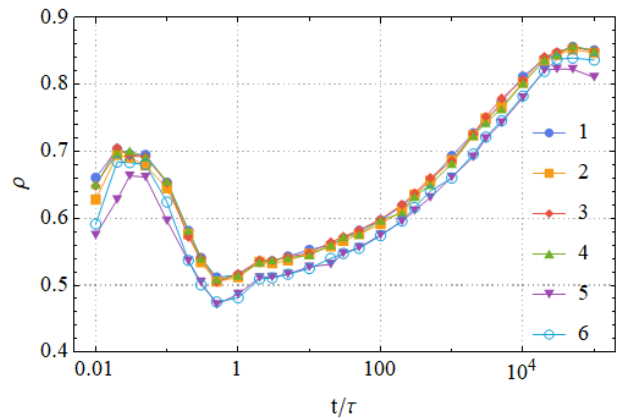


FIG. 7. Correlation between the dynamic propensity as predicted by graph neural networks and measured in simulations as a function of time, analyzed for different variations of the GNN model, which can be found in Table II.

ations of GNN. Clearly, none of the changes made to the GNN inputs have a significant effect on the overall performance. The most significant difference is associated with the networks that trained both small and large particles simultaneously – this adaptation performed worse for our system. In contrast to linear regression and neural networks, where taking into account additional generations clearly led to some improvement in the performance, evidently the inherent local averaging of the GNN is already sufficient to incorporate this type of structural information. Additionally, we observe that compared to the NN, the GNN is less sensitive to overfitting, resulting in relatively smooth lines. This is actually quite remarkable, since the number of weights and biases that need to be optimized in a GNN is significantly larger than in the case of a NN. Moreover, none of the training runs in Fig. 7 used any regularization methods. The fact that GNNs, compared to NNs, have less trouble converging and are less sensitive to overfitting, might be due to the fact that they are trained and evaluated on entire snapshots at once. Realistically, in a snapshot we know that there are strong correlations in mobility between neighboring particles. The fact that the GNN can take into account the mobility of neighbouring particles, will likely lead to smoother variation of the predicted propensity in space than the NN or LR, which results in fewer outliers.

D. Comparing the three methods

Finally, in Fig. 8, we compare the performance of all three methods. Again, we take for each time interval the best-performing result from either LR, NN, or GNN. The results of the three methods are remarkably similar, suggesting that all three approaches are capable of extracting essentially the same information from the input data. Overall, the NN approach performs the worst, likely due to an inability to find the globally optimal solution to its training problem. Intuitively, the fit from linear regression could be reproduced essentially exactly by the NN, with a sufficiently good optimiza-

Nr.	Batch size	Learning rate	GL	Node enc H.L.	Edge enc H.L.	Node H.L.	Edge H.L.	Node gen	Edge par	Tog or Sep
1	5	10^{-4}	4	2 (50, 50)	1 (5)	2 (30, 16) 2 (16, 16)	2 (16, 16) 2 (16, 16)	1	r	Sep
2	5	10^{-4}	4	2 (50, 50)	1 (5)	2 (30, 16) 2 (16, 16)	2 (16, 16) 2 (16, 16)	3	r	Sep
3	5	10^{-4}	4	2 (50, 50)	1 (5)	2 (30, 16) 2 (16, 16)	2 (16, 16) 2 (16, 16)	1	x, y, z	Sep
4	5	10^{-3}	4	2 (50, 50)	1 (5)	2 (30, 16) 2 (16, 16)	2 (16, 16) 2 (16, 16)	1	x, y, z	Sep
5	5	10^{-4}	4	2 (50, 50)	1 (5)	2 (30, 16) 2 (16, 16)	2 (16, 16) 2 (16, 16)	1	r	Tog
6	3	10^{-4}	3	2 (50, 50)	1 (5)	2 (30, 16) 2 (16, 16)	2 (16, 16) 2 (16, 16)	1	r	Tog

TABLE II. Hyperparameters for different graph neural networks used to predict the propensity. Each row corresponds to a line in Fig. 7. In the table the following abbreviations are used: *GL* is the number of graph layers, *Node enc H.L.* and *Edge enc H.L.* represent respectively the number of hidden layers in the node and edge encoder (for both, the number of input parameters is equal to the number of parameters associated with a node or an edge, while the number of output parameters is equal to 10). *Node H.L.* and *Edge H.L.* show the number of hidden layers for the respectively the node and edge hidden layers together with the number of nodes in each layers (the number of output nodes for each of these networks is equal to 10), *Node gen* represents up to how many generations of structural order parameters we provide the GNN with, *Edge par* indicates whether the edge input is given by the absolute distance between particles (r), or the vector distance (x, y, z). Finally *Tog or Sep* represents whether we train the network together (Tog) for small and large particles, or train two separate networks (Sep).

tion. Hence, at any point where the NN performs less well than the LR solution, there is at least some failure to fully optimize the network.

As we saw earlier, GNNs are less sensitive to their hyperparameters than NNs, and converge more easily. Moreover, during intermediate times in the caging regime and the beginning of the diffusive regime GNNs slightly outperform LR. This implies that the averaging that takes place in a GNN provides the network with slightly different information than the averaged parameters of the first and second generation. However, the improvement is extremely limited, and comes at the cost of a significantly more computationally costly training process. To illustrate this: for one choice of hyperparameters and the full set of time intervals, a typical training process takes approximately 3 minutes for LR, 24 hours for NNs, and 6.5 hours for GNNs. Note that these times are achieved by a laptop CPU for the LR, while the NN and GNN trainings made use of an Nvidia GeForce RTX 2080Ti GPU. We conclude that, given the discussion above, linear regression is the preferred method; it is fast, robust, and provides accurate predictions.

IV. CONCLUSION

In this paper, we compared three different ML methods for predicting the dynamic propensity of a glassy system at different times, namely linear regression, neural networks, and graph neural networks. We find surprisingly little difference in their performance over the full range of time intervals considered. The intuitive conclusion one can draw from the similar results of the three methods is that our prediction, at this point, is limited not by our fitting approach, but rather by the infor-

mation contained in the set of structural order parameters. On the bright side, this means that given this set of descriptors, a simple and efficient ML method like linear regression is sufficient for essentially optimal predictions. On the other hand, it suggests that more advanced ML techniques are not likely to provide a solution to the question of how to further improve the prediction of dynamics in these systems. The main question that remains is: what information are we currently missing in order to improve our ability to predict the dynamic propensity, in particular in the caging regime, where the correlation between prediction and reality is currently minimal? While answering this question will require further research, our results here suggest that linear regression is likely a sufficient method for evaluating the predictive capabilities of new sets of structural order parameters.

REFERENCES

- ¹L. Berthier and G. Biroli, “Theoretical perspective on the glass transition and amorphous materials,” *Reviews of Modern Physics* **83**, 587 (2011).
- ²C. Royall and S. R. Williams, “The role of local structure in dynamical arrest,” *Physics Reports* **560**, 1 (2015).
- ³H. Tanaka, H. Tong, R. Shi, and J. Russo, “Revealing key structural features hidden in liquids and glasses,” *Nature Reviews Physics* **1**, 333–348 (2019).
- ⁴E. D. Cubuk, S.S.Schoenholz, J. M. Rieser, B. D. Malone, J. Rottler, D. J. Durian, E. Kaxiras, and A. J. Liu, “Identifying structural flow defects in disordered solids using machine-learning methods,” *Physical Review Letters* **114**, 108001 (2015).
- ⁵V. Bapst, T. Keck, A. Grabska-Barwińska, C. Donner, E. D. Cubuk, S. J. Schoenholz, A. Obika, A. W. R. Nelson, T. Back, D. Hassabis, and P. Kohli, “Unveiling the predictive power of static structure in glassy systems,” *Nature Physics* **16**, 448 (2020).

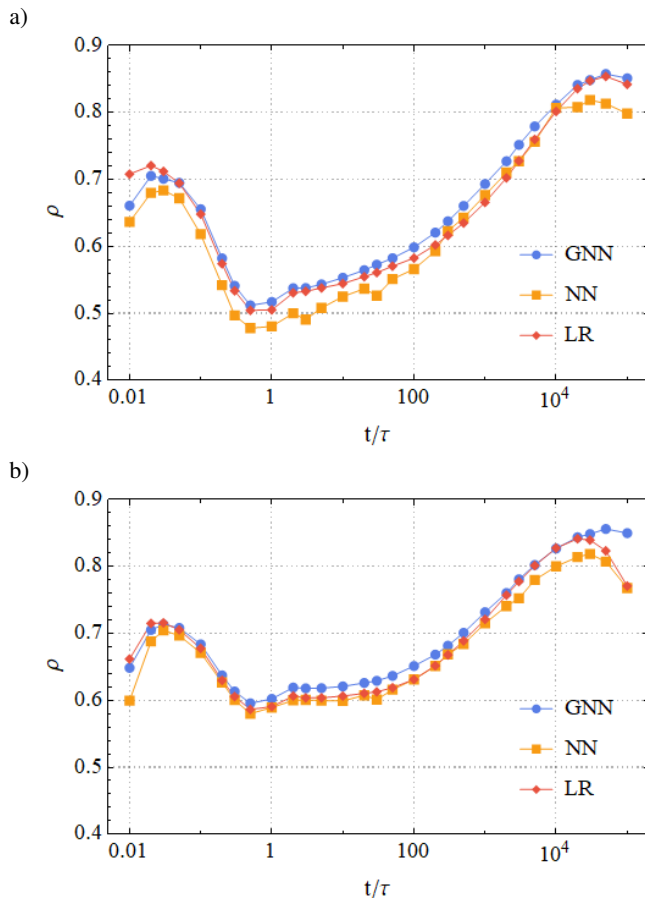


FIG. 8. Comparison of the accuracy of the dynamic propensity prediction over time as obtained with LR, NN and GNN. Results are shown for both the (a) large and (b) small particles. For each point in time we use for each method the set of hyperparameters that resulted in the best performance.

⁶E. Boattini, S. Marín-Aguilar, S. Mitra, G. Foffi, F. Smallenburg, and L. Filion, “Autonomously revealing hidden local structures in supercooled liquids,” *Nature Communications* **11**, 5479 (2020).

⁷J. Paret, R. L. Jack, and D. Coslovich, “Assessing the structural heterogeneity of supercooled liquids through community inference,” *The Journal of Chemical Physics* **152**, 144502 (2020).

⁸E. Boattini, F. Smallenburg, and L. Filion, “Averaging local structure to predict the dynamic propensity in supercooled liquids,” *Physical Review Letters* **127**, 088007 (2021).

⁹S. S. Schoenholz, E. D. Cubuk, D. M. Sussman, E. Kaxiras, and A. J. Liu, “A structural approach to relaxation in glassy liquids,” *Nature Physics* **12**,

469 (2016).

¹⁰D. Richard, M. Ozawa, S. Patinet, E. Stanifer, B. Shang, S. Ridout, B. Xu, G. Zhang, P. Morse, J.-L. Barrat, *et al.*, “Predicting plasticity in disordered solids from structural indicators,” *Physical Review Materials* **4**, 113609 (2020).

¹¹S. Marín-Aguilar, H. H. Wensink, G. Foffi, and F. Smallenburg, “Tetrahedrality dictates dynamics in hard sphere mixtures,” *Physical Review Letters* **124**, 208005 (2020).

¹²D. C. Rapaport, “The Event-Driven Approach to N-Body Simulation,” *Progress of Theoretical Physics Supplement* **178**, 5–14 (2009).

¹³A. Widmer-Cooper, P. Harrowell, and H. Fynewever, “How reproducible are dynamic heterogeneities in a supercooled liquid?” *Physical Review Letters* **93**, 135701 (2004).

¹⁴A. Widmer-Cooper and P. Harrowell, “On the study of collective dynamics in supercooled liquids through the statistics of the isoconfigurational ensemble,” *The Journal of Chemical Physics* **126**, 154503 (2007).

¹⁵W. Lechner and C. Dellago, “Accurate determination of crystal structures based on averaged local bond order parameters,” *The Journal of Chemical Physics* **129**, 114707 (2008).

¹⁶P. J. Steinhardt, D. R. Nelson, and M. Ronchetti, “Bond-orientational order in liquids and glasses,” *Physical Review B* **28**, 784 (1983).

¹⁷C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)* (Springer-Verlag, 2006).

¹⁸A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” *Advances in neural information processing systems* **32**, 8026 (2019).

¹⁹D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980* (2014).

²⁰N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research* **15**, 1929 (2014).

²¹P. W. Battaglia, J. B. C. Hamrick, V. B., A. S., V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. J. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, “Relational inductive biases, deep learning, and graph networks,” (2018).

²²P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, *et al.*, “Interaction networks for learning about objects, relations and physics,” *Advances in Neural Information Processing Systems* **29**, 4502 (2016).

²³F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” *IEEE Transactions on Neural Networks* **20**, 61 (2009).

ACKNOWLEDGEMENTS

The authors would like to thank Marjolein de Jager for many discussions. L.F. and E.B. acknowledge funding from The Netherlands Organisation for Scientific Research (NWO) (Grant No. 16DDS004), and L.F. acknowledges funding from NWO for a Vidi grant (Grant No. VI.VIDI.192.102).