Characterizing the Relationship between Environment Layout and Crowd Movement using Machine Learning

Weining Liu*
Rutgers University
wl420@scarletmail.rutgers.edu

Kaidong Hu*
Rutgers University
hukaidong@gmail.com

Sejong Yoon
The College of New Jersey
yoons@tcnj.edu

Vladimir Pavlovic
Rutgers University
vladimir@cs.rutgers.edu

Petros Faloutsos
York University
pfal@cse.yorku.ca

Mubbasir Kapadia
Rutgers University
mk1353@cs.rutgers.edu

ABSTRACT

Crowd simulations facilitate the study of how an environment layout impacts the movement and behavior of its inhabitants. However, simulations are computationally expensive, which make them infeasible when used as part of interactive systems (e.g., Computer-Assisted Design software). Machine learning models, such as neural networks (NN), can learn observed behaviors from examples, and can potentially offer a rational prediction of a crowd's behavior efficiently. To this end, we propose a method to predict the aggregate characteristics of crowd dynamics using regression neural networks (NN). We parametrize the environment, the crowd distribution and the steering method to serve as inputs to the NN models, while a number of common performance measures serve as the output. Our preliminary experiments show that our approach can help users evaluate a large number of environments efficiently.

CCS CONCEPTS

• **Computing methodologies** → Neural networks; Modeling and simulation;

KEYWORDS

Crowd Simulation, Neural Networks, Computer Aided Design

ACM Reference format:
Weining Liu, Kaidong Hu, Sejong Yoon, Vladimir Pavlovic, Petros Faloutsos, and Mubbasir Kapadia. 2017. Characterizing the Relationship between Environment Layout and Crowd Movement using Machine Learning. In *Proceedings of MIG 2017, UPC, Barcelona, Spain, November 2017,* 6 pages.
https://doi.org/10.475/123_4

1 INTRODUCTION

Crowd simulation is used in a variety of applications such as disaster and security simulations, architectural design, urban planning, and first responder training. A key requirement is the need to predict and analyze the behavior of human crowds in previously unseen situations (e.g., a change in environment layout, increased crowd densities, or a stampede with increased levels of aggression from the crowd members). However, it is often computationally expensive, especially for applications that require the execution of a large number of simulations with different parameters (e.g, Computer-Aided Design software). The motivation for this study is to explore the potential of machine learning to predict the relationship between the layout of an environment and the aggregate dynamics of the crowd, without the need to run expensive simulations.

To meet these requirements, we propose to model the aggregate dynamics of a virtual crowd and their relationship to the environment, by training a neural-network on simulated crowd movement data, generated using state-of-the-art crowd simulation techniques (e.g., Social Force [Helbing and Molnar 1995], and ORCA [van den Berg et al. 2008]). We explore different NN architectures to systematically study their ability to fit the training data, while generalizing to new situations. Our preliminary experiments demonstrate the potential of neural networks to capture crowd characteristics, in comparison to baseline linear regression models. While not the focus of this paper, our approach is general and can be used with real human crowd observations.

2 RELATED WORK

There is a wealth of research in simulating crowds with numerous approaches that rely on expert rules or examples for decision-making. We only provide a short review below and refer the readers to comprehensive surveys [Kapadia et al. 2015] for more details.

Rule-based systems date back to the work on flocking behaviors using particle systems [Reynolds 1987, 1999]. These particle approaches are further refined in the social force models [Helbing and Molnar 1995; Pelechano et al. 2007]. In some approaches, steering is regarded as an integral part of the agents' abilities. Geometric algorithms are used [Guy et al. 2009; van den Berg et al. 2008] to determine the velocity of next time to avoid a collision with another agent. Agents have also used affordance fields [Kapadia et al. 2009] to try to find safe passage to a goal. A cognitive system was used in the work [Pelechano et al. 2007] which included utility functions for desires, an attentional system to limit perception of the environment, and a motor system to carry out actions. The system can also switched between other steering algorithms [Singh et al. 2011] to best suit an agent's needs. When the environment is complex, we have to employ path planning [Huang et al. 2014; Kallmann and Kapadia 2016] to promise all agents can reach the target without

*Both authors contributed equally to the paper

MIG 2017, November 2017, UPC, Barcelona, Spain

ACM ISBN 123-4567-24-567/17/06…$15.00
https://doi.org/10.475/123_4

being trapped. Some approach can solve path planning in dynamic environments [Kallmann and Kapadia 2014; Kapadia et al. 2013a; Ninomiya et al. 2015]. Parallelized approaches [Garcia et al. 2014; Kapadia et al. 2013b] are also be use to accelerate the searching.

Data-driven techniques use local-space samples which are generated from observations to create steering policies. In [Lerner et al. 2007] video samples were compiled into a database based on which the agents steer. The work of [Lee et al. 2007] focused more on recreating group dynamics than individual steering. The work of [Torrens et al. 2011] used a more constrained state space of discretized slices around an agent.

There has been prior works using machine learning algorithms to understand or learn crowd motion [Scovanner and Tappen 2009], including those using data-driven techniques, e.g. [Bera et al. 2016a,b; Metoyer and Hodgins 2003]. However, most of these approaches are focused on human movement without much consideration on the relationship between the environment and the crowd motion. Our motivation is different, we aim to build a computation model to predict the aggregated characteristics of crowd dynamics in relation to changes in the environment.

3 OVERVIEW

We start with an environment that consists of mutable and immutable elements. The parameters of the mutable elements and their limits contribute to the parametrization domain. We also configure a simulated crowd and parametrize its density and the initial conditions, as well as properties of the agents. The union of all parameters constitute the parametrization domain.

We then run a large number of simulations corresponding to a dense sampling of the parametrization domain. For each simulation we compute common metrics, like evacuation time and collisions, that characterize the behavior of the crowd. As a result, we end up with a large training set which provides a discrete mapping between the parameter domain and crowd behavior metrics.

We use this training set to train a regression neural network with fully connected layers. The trained model allows to predict the behavior metrics for previously unseen environment and crowd configurations at a fraction of the computational cost of a full crowd simulation. We experiment with two different network structures, and compare their performance against standard linear regression.

4 PROBLEM FORMULATION

We construct our problem as a mapping between environment & crowd configurations and performance metrics computed from simulating the crowds' movement within the environment using standard steering algorithms[Singh et al. 2009b]. The remainder of this section presents the details of our formulation.

4.1 Environment Configuration

In our context, environments consist of mutable and immutable elements. Typically the outer walls are fixed, while a number of internal elements are allowed to undergo rigid body transformations. The 2D position (x, y) of an element and its orientation with respect to the x-axis, ϕ, form the element's configuration parameters, $\mathbf{o} = [x, y, \phi]$. We group all such parameters into the *environment configuration vector*, $\mathbf{e} = [\mathbf{o}_1, .., \mathbf{o}_n]$, n is the number of parameters in a layout. Environment Configuration is referred to as 'Env' in the tables in later chapters.

4.2 Crowd Configuration

For each simulation run, new agents are evenly distributed (with random velocities) in predefined areas within an environment. The numbers of all agents in those areas are recorded separately per area and grouped into a vector, $\mathbf{d} = [d_1, .., d_m]$, m is the number of predefined areas. We refer to Crowd Configuration as 'Crowd' in the tables in later chapters.

4.3 Crowd Simulator

In this paper, we use two Crowd Simulation Algorithms, Social Forces (SF) [Helbing and Molnar 1995] and ORCA, [Snape et al. 2012; van den Berg et al. 2008].

Each algorithm exposes a number of parameters that govern the behavior of the agents. The Social Forces algorithm uses 12 parameters including the mass of agent, the personal space threshold, observation radius, max speed, and factors of forces (repulsive force, proximity force and frictional force) between two agents and between one agent and one obstacle. The ORCA algorithm uses 4 parameters which can describe observation radius, safe time to avoid collision between agents, safe time to avoid collision between agent and obstacles, and max speed. We group all such parameters into $\mathbf{a} = [a_1, ..., a_k]$, k is the number of parameters of simulator's parameters. We sample these parameters using the uniform random for each simulation run, and record their values as part of the input vector for the machine learning aspect of our work. Crowd Simulator is referred to as 'AI' in the tables in later chapters.

4.4 Configuration Space

The combined configurations described above constitute the domain or input space of our machine learning approach, $\mathcal{D} = [\mathbf{E}, \mathbf{D}, \mathbf{A}]$. For building A, the environment configuration contains 31 parameters, the crowd configuration contains 10 parameters. The building B uses 22 and 27 parameters to describe the environment configuration and the crowd configuration, respectively. Social Force AI has 12 factors while ORCA has 4. Table 1 outlines the cardinality of the different configuration spaces we used for training our NN models.

4.5 Crowd Analytics

The output space consists of four commonly used metrics that measure different aspects of a simulator's performance:

(1) Collisions (c). The average number of collisions over agents that happened among agents and between agents and obstacles.
(2) Traveling distance (l). The average distance over agents covered by the agents.
(3) Traveling time (t). The average time it took the agents to reach their destinations.
(4) PLE (p). A measure proposed by[J. et al. 2010; Whittle 2014], which measures the effort exerted by the agents to reach their destinations. We compute it on the average.

All four metrics are generated by the SteerBench [Kapadia et al. 2011a; Singh et al. 2009a] module in SteerSuite [Singh et al. 2009b].

Table 1: The number of samples in datasets for each configuration, as well as the number of parameters (dimension). Each cell shows (size/dimension). For Env, Crowd, AI categories, only corresponding class, 'Environment', 'Crowd', or 'Agent' is treated as parameters, and we assign other class with default values, make them constant to the whole dataset. Some cells are marked as '-', denoting we do not have the experiments that need to feed such data. For 'All' class, all were treated as parameters and varied in the dataset.

Map - AI	Env	Crowd	AI	All
A - SF	27501/31	13792/10	32164/12	18748/53
B - ORCA	4372/22	45867/27	6089/4	24281/53
A - ORCA	-	-	-	24987/45
B - SF	-	-	-	24658/61

5 PROPOSED MACHINE LEARNING FRAMEWORK

In this section we describe in detail our machine learning approach that results in a model that can efficiently predict the output vector (metrics) given an arbitrary input configuration, $\mathcal{D}$.

5.1 Training Data Generation

We statistically sample the parameter values of the configurations $\mathcal{D} = [\mathbf{E}, \mathbf{D}, \mathbf{A}]$ to generate a large corpus of training data. Given a specific configuration, described in Section 4, we use SteerSuite [Singh et al. 2009b] to simulate the crowd behavior. For each simulation we compute the output metrics (Section 4), which together with the configuration parameters form an element of the training set.

The size of datasets can be seen in Table 1. We use 80% of the dataset to train, and the rest 20% for testing. The size of the dataset depends on the configuration and not the metrics.

5.2 Neural Network Architecture

We use a regression neural network to model the desired mapping. The input of a NN model includes parameters of one or more Configuration and a noise value, further described below. Regardless of what aspects are to be taken into account for prediction (environment layout, crowd configuration, and/or agent configuration), the input layer takes the input data in the same form as:

$$\mathbf{x} = [\mathbf{e}, \mathbf{d}, \mathbf{a}, z] = [o_1, o_2, ..., o_n, d_1, d_2, ..., d_m, a_1, a_2, ..., a_k, z]^\top,$$

where z represents the noise. The noise value represents the randomly set initial status of agents including position and velocity. Though all agents' initial status can affect the final result, we can use one single noise scalar to represent the joint affect. The output of NN models are a set of measures characterizing the aggregate dynamics of the crowd behavior, such as evacuation time, average distance, or energy expenditure.

Loss Function. We define $\hat{y}$ as a prediction of a matrix C/D/P/T), and y as the corresponding ground truth value. We use the root mean square error (RMSE) between $(\hat{\mathbf{y}})$ and $(\mathbf{y})$ as our loss function for training. The loss function $L(\hat{\mathbf{y}}, \mathbf{y})$ is computed as follows:

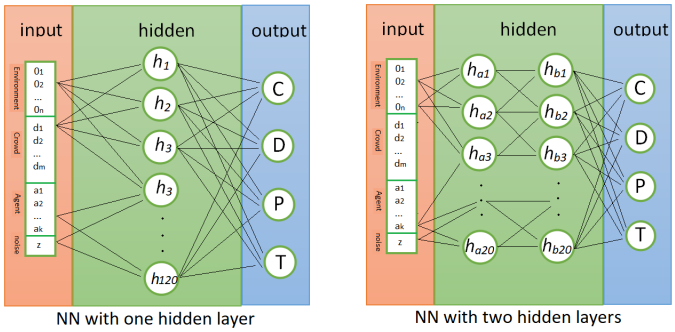

Figure 1: This figure shows the NN structures used in our experiments. The left image shows the NN with only one hidden layer containing many Neurons (120). The right image shows the NN with two smaller hidden layers (20 Neurons each).

$$L(\hat{\mathbf{y}}, \mathbf{y}) = \sqrt{\frac{1}{N}\sum(\hat{y} - y)^2} \tag{1}$$

Training Method. We use Adam [Kingma and Ba 2014] for training our NN models. Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is well suited for problems that are large in terms of data and/or parameters, and works well for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning. We also employ the dropout technique to reduce overfitting. We used 50,000 as the maximum number of iterations for training the NN models. In our experiments, we observe that the loss functions plateau at about 20,000 iterations.

Evaluation of Different Network Models. We tried 8 different NNs including vanilla linear regression and single/double layer NN to illustrate potential utility of our approach. Here, we present comparative results between linear regression and one selected NN model. Possible network structures are given in Figure 1.

For the NN with one hidden layer, given the input $\mathbf{x}$, the output of the hidden layer, $\mathbf{h}$ is computed as follows:

$$\mathbf{h} = Relu(\mathbf{W}^{(0)\intercal} \cdot \mathbf{x} + \mathbf{b}^{(0)}), \tag{2}$$

where $\mathbf{W}^{(0)}$ is the weights of first layer, and $\mathbf{b}^{(0)}$ is the bias of fist layer. The final output, $\hat{\mathbf{y}}$, is

$$\hat{\mathbf{y}} = \mathbf{W}^{(1)\intercal} \cdot \mathbf{h} + \mathbf{b}^{(1)}. \tag{3}$$

where $\mathbf{W}^{(1)}$ is the weights of output layer, and $\mathbf{b}^{(1)}$ is the bias of output layer.

For the NN with two hidden layers, the output is similarly calculated as follows:

$$\mathbf{h}^{(0)} = Relu(\mathbf{W}^{(0)\intercal} \cdot \mathbf{x} + \mathbf{b}^{(0)}), \tag{4}$$

$$\mathbf{h}^{(1)} = Relu(\mathbf{W}^{(1)\intercal} \cdot \mathbf{h}^{(0)} + \mathbf{b}^{(1)}), \tag{5}$$

$$\hat{\mathbf{y}} = \mathbf{W}^{(2)\intercal} \cdot \mathbf{h}^{(1)} + \mathbf{b}^{(2)}. \tag{6}$$

where $\mathbf{W}^{(0)}$ is the weights of first layer, and $\mathbf{b}^{(0)}$ is the bias of first layer, $\mathbf{W}^{(1)}$ is the weights of second layer, and $\mathbf{b}^{(1)}$ is the bias of second layer, $\mathbf{W}^{(2)}$ is the weights of output layer, and $\mathbf{b}^{(2)}$ is the bias of output layer The above description assumes using the default rectified linear activation function for training. However other activation functions may also be used. Our experiments below evaluate the impact of the activation functions, namely ReLU and leaky ReLU, on the prediction accuracy of the trained networks.

5.3 Evaluation

We use three different measures for evaluating the fidelity of the NN models. First, we use the standard RMS error for quantifying prediction error. In addition, we compute the following measure over the test dataset which normalizes the proportion of the RMSE error with the mean value of the ground truth data:

$$A(\hat{\mathbf{y}}, \mathbf{y}) = 1.0 - \frac{\sqrt{\frac{1}{N}\sum(\hat{y} - y)^2}}{\mu}, \tag{7}$$

where μ is the mean of the ground truth data. In the ideal case, A = 1.0. A higher score means a better performance. Note that the value of A may be negative.

We also use the R^2 score [Magee 1990] to measure the quality of the prediction. This score is the proportion of the variance in the dependent variable that is predictable from the independent variable.

$$R^2(\hat{\mathbf{y}}, \mathbf{y}) = 1.0 - \frac{\frac{1}{N}\sum(\hat{y} - y)^2}{\sigma^2}, \tag{8}$$

where σ is the standard deviation. In the ideal case, the R^2 score = 1.0. A higher R^2 score means a better prediction performance. Its value may also be negative.

6 EXPERIMENTS

This section describes our preliminary experiments to evaluate the prediction accuracy of our proposed model.

6.1 Preliminaries

We used two building blue-prints in our experiments. Obstacles inside the buildings are parameterized to permit rigid transformations. Each building is divided into a number of areas, in which the number of agents are sampled. We also sample the factors of AI algorithms that determine the agents' behavior. For the purpose of our experiments, we use two representative steering algorithms, Social Forces [Helbing and Molnar 1995] and ORCA [Snape et al. 2012; van den Berg et al. 2008].

The layout can be seen from Figure 2. The outline of the building is fixed, while the inside obstacles can move and rotate. From the default (a, c) layout, we use blue bexes to demonstrate obstacles that can only move vertically or horizontally, green ones can rotate, and reds can both translate and rotate. Building A is divided into 10 areas based on the region of rooms and halls, and Building B is evenly divided into 27 areas. We use gray blocks to denote the region of each area (can be seen from the default layouts), where crowds of different densities can be generated.

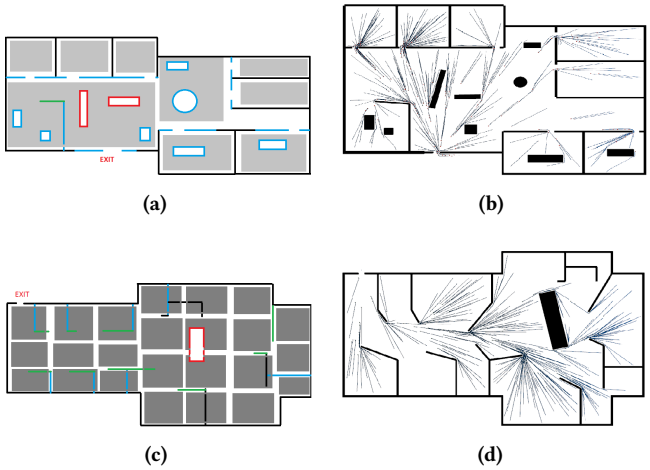

Figure 2: Building configuration and corresponding simulation cases. Left tow: Layouts corresponding to building A and building B. right two: Selected simulating cases upon each map. Building layouts with mutable elements shown with colored lines; the predefined areas where agents are stochastically distributed are indicated in gray blocks. Thin lines in simulation cases are agent traces when running the simulator.

6.2 Implementation Details

We used SteerSuite [Singh et al. 2009b] to generate the training data using SF [Helbing and Molnar 1995; Pelechano et al. 2007] and ORCA [Snape et al. 2012] which were implemented as modules in SteerSuite. We used the sampling technique described in [Kapadia et al. 2011b] for training data generation. The metrics were computed using SteerBench [Kapadia et al. 2011a; Singh et al. 2009a]. We trained the Neural Networks using TensorFlow.

6.3 Experiment Results

Here, three different workout sets have been selected to demonstrate our training results. In Tables 2 and 3, we attempted to train models that only sensible to one type of configurations, based on either Env, crowd or AI class. SF steering algorithm or OCRA is used as the steering algorithm, separately. Table 4 then records result based on models that change all three type of configurations. In each table, vanilla linear regression and 120 nodes single layer neural network are selected for the comparison. As the result, we can see that both neural network models provided competitive performance in the context of different system complexities. In general, most of tests are able to provide reasonable result in average travel distance, PLE energy and traveling time predictions. Most of them show less than 5 percent error rate to the amplification, lead to a reliable result we can use in further application. However, the relative weak result evaluated in R score shows the prediction module cannot tune itself to give a precise prediction right now. This tell us we still have space to improve our model, and asked us for further investigations.

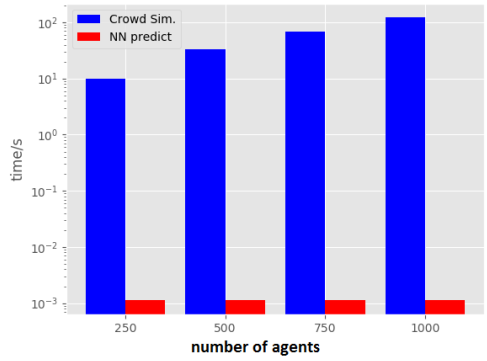

Figure 3: Comparison of Performance Times to run crowd simulations in order to compute metrics, or to query the neural networks. NN query times are a fraction of the time taken to run simulations and are independent of number of agents to simulate.

6.4 Computational Performance

We verify our hypothesis on the efficiency of using neural networks for predicting crowd measures. In this experiment, we compute the crowd measures by running crowd simulations and by directly querying the NN models. We repeat this experiment for increasing number of agents. Crowd simulations takes several seconds (even minutes) to compute and the time increases quadratically with number of agents. At the same time, predicting values using neural networks takes constant time in only milliseconds. Figure 3 illustrates the experiment results. For the consideration of time cost in data-generating and training, we believe that by taking advantage from the generalizablility behind neural networks, we could reduce working expensiveness on such as optimization problem using this module. Further experiments on this region are still on process.

7 CONCLUSION

Our present experiments investigated using neural network predicting performance matrix from environment and crowd configurations, with expectation to replace the traditional simulator. The result show its possibility, and an absolute performance advancement proves it potential values on processing compute dense works, such as the building inner placement optimization. Consider its potential, in the future, further neural network development and applying them on a practical situation are necessary to the researchers.

ACKNOWLEDGMENTS

Yoon was supported in part by The College of New Jersey under SOSA 2017-2019 grant. The work in this project was partially supported by the NSERC Discovery and Create programs (Canada).

REFERENCES

Aniket Bera, Sujeong Kim, and Dinesh Manocha. 2016a. Interactive Crowd-Behavior Learning for Surveillance and Training. *IEEE Computer Graphics and Applications* 36, 6 (2016), 37–45. https://doi.org/10.1109/MCG.2016.113

Aniket Bera, Sujeong Kim, and Dinesh Manocha. 2016b. Online parameter learning for data-driven crowd simulation and content generation. *Computers & Graphics* 55 (2016), 68–79. https://doi.org/10.1016/j.cag.2015.10.009

FM Garcia, M Kapadia, and NI Badler. 2014. GPU-based dynamic search on adaptive resolution grids. *IEEE International Conference on Robotics and Automation, ICRA* (2014), 1631–1638.

S. J. Guy, J. Chhugani, C. Kim, N. Satish, M. Lin, D. Manocha, and P. Dubey. 2009. Clearpath: highly parallel collision avoidance for multi-agent simulation. *ACM SIGGRAPH/Eurographics SCA* (2009), 177–187.

D. Helbing and P. Molnar. 1995. Social force model for pedestrian dynamics. *Physical review* 51, 5 (1995), 4282.

T Huang, M Kapadia, NI Badler, and M Kallmann. 2014. Path planning for coherent and persistent groups. *IEEE International Conference on Robotics and Automation, ICRA* (2014), 1652–1659.

Guy S. J., Chhugani J., Curtis S., Dubey P., Lin M., and Manocha D. 2010. PLEdestrians: a least-effort approach to crowd simulation. *ACM SIGGRAPH/Eurographics SCA* (2010), 1–24.

M Kallmann and M Kapadia. 2014. Navigation meshes and real-time dynamic planning for virtual worlds. *Special Interest Group on Computer Graphics and Interactive Techniques Conference, SIGGRAPH* (2014), 3:1–3:81.

M. Kallmann and M. Kapadia. 2016. Geometric and discrete path planning for interactive virtual worlds. *ACM SIGGRAPH* (2016), 1–29.

M Kapadia, A Beacco, FM Garcia, V Reddy, N Pelechano, and NI Badler. 2013a. Multi-domain real-time planning in dynamic environments. *The ACM SIGGRAPH / Eurographics Symposium on Computer Animation, SCA* (2013), 115–124.

M Kapadia, FM Garcia, CD Boatright, and NI Badler. 2013b. Dynamic search on the GPU. *IEEE/RSJ International Conference on Intelligent Robots and Systems* (2013), 3332–3337.

M. Kapadia, N. Pelechano, J. Allbeck, and N. Badler. 2015. *Virtual crowds: steps toward behavioral realism.* Vol. 7. Synthesis Lectures on Visual Computing. 1–270 pages.

M. Kapadia, S. Singh, W. Hewlett, and P. Faloutsos. 2009. Egocentric affordance fields in pedestrian steering. *ACM SIGGRAPH I3D* (2009), 215–223.

Mubbasir Kapadia, Matthew Wang, Glenn Reinman, and Petros Faloutsos. 2011a. Improved Benchmarking for Steering Algorithms. *4th International Conference on Motion in Games* (2011).

Mubbasir Kapadia, Matthew Wang, Shawn Singh, Glenn Reinman, and Petros Faloutsos. 2011b. Scenario Space: Characterizing Coverage, Quality, and Failure of Steering Algorithms. *ACM SIGGRAPH Symposium on Computer Animation* (2011).

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). http://arxiv.org/abs/1412.6980

K. H. Lee, M. G. Choi, Q. Hong, and J. Lee. 2007. Group behavior from video: a data-driven approach to crowd simulation. *ACM SIGGRAPH/Eurographics SCA* 1 (2007), 109–118.

A. Lerner, Y. Chrysanthou, and D. Lischinski. 2007. Crowds by Example. *CGF* 26, 3 (2007), 655–664.

L. Magee. 1990. R2 measures based on Wald and likelihood ratio joint significance tests. *The American Statistician* (1990), 250.

R. A. Metoyer and J. K. Hodgins. 2003. Reactive pedestrian path following from examples. *CASA* 20 (2003), 149–156.

K Ninomiya, M Kapadia, A Shoulson, FM Garcia, and NI Badler. 2015. Planning approaches to constraint-aware navigation in dynamic environments. *Journal of Visualization and Computer Animation* (2015), 119–139.

N. Pelechano, J. M. Allbeck, and N. I. Badler. 2007. Controlling individual agents in high-density crowd simulation. *ACM SIGGRAPH/Eurographics SCA* 1 (2007), 108.

C. W. Reynolds. 1987. A distributed behavioral model. *ACM SIGGRAPH* 21, 4 (1987), 25–34.

C. W. Reynolds. 1999. Steering behaviors for autonomous characters. *GDC, Citeseer* (1999), 763–782.

P. Scovanner and M. F. Tappen. 2009. Learning pedestrian dynamics from the real world. In *2009 IEEE 12th International Conference on Computer Vision*. 381–388. https://doi.org/10.1109/ICCV.2009.5459224

Shawn Singh, Mubbasir Kapadia, Glenn Reinman, and Petros Faloutsos. 2009a. SteerBench: A Benchmark Suite for Evaluating Steering Behaviors. *Computer Animation and Virtual Worlds* (2009).

Shawn Singh, Mubbasir Kapadia, Glenn Reinman, and Petros Faloutsos. 2009b. SteerSuite: An Open Framework For Developing, Evaluating and Sharing Steering Algorithms. *International Conference on Motion in Games* (2009).

S. Singh, M. Kapadia, G. Reinman, and P. Faloutsos. 2011. Footstep navigation for dynamic crowds. *CAVW* 22, 2-3 (2011), 151–158.

J. Snape, S. J. Guy, D. Vembar, A. Lake, and M. C. Lin. 2012. Reciprocal collision avoidance and navigation for video games. *Game Developers Conf.* (2012).

P. Torrens, X. Li, and W. A. Griffin. 2011. Building Agent-Based Walking Models by Machine-Learning on Diverse Databases of Space-Time Trajectory Samples. *Transactions in GIS* 15 (2011), 67–94.

Jur van den Berg, Ming C. Lin, and Dinesh Manocha. 2008. Reciprocal velocity obstacles for real-time multi-agent navigation. *Proc. IEEE Int. Conf. Robotics and Automation* (2008), 1928–1935.

M. Whittle. 2014. *Gait analysis: an introduction.* Butterworth-Heinemann.

Table 2: Building A with Social Force AI. Each cell contains RMSE/A/R^2 scores (the best result are boldly marked). This table shows the result of vanilla linear regression (LR) as well as the result of NNs whose hidden layer contains 120 Neurons.

Categories	Avg. Collisions (number)	Avg. Traveling Distance (m)	Avg. PLE energy (J/kg)	Avg. Traveling Time (s)
Linear Regression				
Env	2.53/ 0.5157/ 0.1549	7.02/ 0.9312/ 0.0077	24.18/ 0.9309/ 0.0205	4.96/ 0.9367/ 0.0310
Crowd	2.70/ 0.6391/ 0.8080	3.12/ 0.9672/ 0.7798	11.46/ 0.9649/ 0.7737	2.23/ 0.9696/ 0.7920
Agent	8.43/ 0.2570/ 0.3776	**5.94**/ 0.9386/ 0.3254	28.18/ 0.9158/ 0.1298	**3.75**/ 0.9499/ 0.2467
Single Layer NN with 120 nodes				
Env	2.53/ 0.5150/ 0.1524	**4.72**/ 0.9538/ 0.5519	**12.86**/ 0.9632/ 0.7230	**4.11**/ 0.9475/ 0.3349
Crowd	**1.89**/ 0.7480/ 0.9064	**2.40**/ 0.9747/ 0.8694	**8.89**/ 0.9728/ 0.8640	**2.02**/ 0.9724/ 0.8281
Agent	**8.07**/ 0.2887/ 0.4296	7.01/ 0.9275/ 0.0610	**26.23**/ 0.9216/ 0.2461	4.68/ 0.9376/ -0.1714

Table 3: Building B with ORCA AI. Each cell contains RMSE/A/R^2 scores (the best results are boldly marked). This table shows the result of standard linear regression (LR) as well as the results of NNs whose hidden layer contains 120 Neurons.

Categories	Avg. Collisions (number)	Avg. Traveling Distance (m)	Avg. PLE energy (J/kg)	Avg. Traveling Time (s)
Linear Regression				
Env	**1.92**/ 0.8647/ 0.2346	**1.83**/ 0.9866/ 0.8105	**6.86**/ 0.9852/ 0.7851	**1.72**/ 0.9840/ 0.7818
Crowd	**1.24**/ 0.9131/ 0.8507	**1.15**/ 0.9915/ 0.9732	**5.09**/ 0.9889/ 0.9567	**1.14**/ 0.9892/ 0.9631
Agent	**1.25**/ 0.9121/ 0.8457	**1.14**/ 0.9915/ 0.9735	**4.61**/ 0.9899/ 0.9642	**1.11**/ 0.9895/ 0.9647
Single Layer NN with 120 nodes				
Env	2.85/ 0.7990/ -0.6882	2.76/ 0.9798/ 0.5680	8.66/ 0.9814/ 0.6570	2.40/ 0.9777/ 0.5758
Crowd	1.25/ 0.9124/ 0.8481	1.82/ 0.9865/ 0.9332	6.03/ 0.9868/ 0.9391	1.57/ 0.9851/ 0.9299
Agent	1.70/ 0.8798/ 0.7118	1.98/ 0.9853/ 0.9207	5.16/ 0.9887/ 0.9551	1.94/ 0.9816/ 0.8929

Table 4: . (RMSE/A/R^2) of metrics conditioned on all 3 configurations. SF-A means Social Force AI running in building A. ORCA-B means ORCA AI running in building B. The numbers in the first column denote the architecture of NN models.

Categories	Avg. Collisions (number)	Avg. Traveling Distance (m)	Avg. PLE energy (J/kg)	Avg. Traveling Time (s)
Linear Regression				
SF-A	11.32/ 0.0803/ 0.4760	**9.68**/ 0.9092/ 0.3124	**37.80**/ 0.8975/ 0.2815	**6.50**/ 0.9205/ 0.3232
SF-B	**3.40**/ -0.3247/ 0.2719	**3.99**/ 0.9709/ 0.7920	**15.89**/ 0.9658/ 0.7423	**3.43**/ 0.9669/ 0.7356
ORCA-A	7.05/ 0.6935/ 0.3759	8.36/ 0.9180/ 0.2810	33.76/ 0.9069/ 0.2363	24.22/ 0.7176/ 0.1619
ORCA-B	3.06/ 0.7785/ 0.3542	**5.92**/ 0.9565/ 0.4881	**15.66**/ 0.9672/ 0.7223	27.83/ 0.7467/ -0.0159
Single Layer NN with 120 nodes				
SF-A	**10.54**/ 0.1434/ 0.5455	10.30/ 0.9034/ 0.2212	39.43/ 0.8930/ 0.2182	6.99/ 0.9145/ 0.2165
SF-B	3.75/ -0.4576/ 0.1186	6.67/ 0.9514/ 0.4182	19.78/ 0.9575/ 0.6008	5.27/ 0.9491/ 0.3750
ORCA-A	**6.39**/ 0.7224/ 0.4878	**6.42**/ 0.9371/ 0.5763	**23.58**/ 0.9350/ 0.6273	**22.92**/ 0.7328/ 0.2499
ORCA-B	**3.02**/ 0.7814/ 0.3714	5.95/ 0.9563/ 0.4831	16.21/ 0.9661/ 0.7027	**26.98**/ 0.7545/ 0.0454

Table 5: Optimized metric values using NNs / related metric values computed by simulating the optimized environment.

case	Avg. Collisions (num)	Avg. Distance (m)	Avg. PLE energy (J/kg)	Avg. Time (s)
120 SF A	-35.81/ 14.6	90.05/ 88.47	309.64/ 295.21	70.13/ 76.84
40-40 SF A	3.19/ 3.38	87.76/ 110.20	264.62/ 328.03	59.64/ 84.11
20-20 ORCA B	9.38/ 11.71	97.33/ 128.41	331.14/ 429.07	58.94/ 100.85
40-40 ORCA B	8.23/ 8.89	90.48/ 127.86	326.38/ 444.64	71.66/ 98.89

Table 6: Computation time to perform optimizations using neural network predictions / running simulations.

case	Collisions	Distance	PLE	Time
120 SF A	256s/ 4d14h	306s/ 22d16h	4.5s/ 19h	352s/ 21d2h
40-40 SF A	633s/ 4d17h	2.3s/ 9h	3.7s/ 13h	4.0s/ 11h
20-20 ORCA B	14s/ 2d22h	5.7s/ 28h	5.0s/ 22h	5.9s/ 25h
40-40 ORCA B	28s/ 6d5h	26s/ 5d22h	18s/ 3d16h	32s/ 6d17h