Artificial Neural Networks for the Estimation of Pedestrian Interaction Forces

Simone Göttlich and Stephan Knapp

Abstract We present a data fitting approach for the social force model by Helbing and Molnár using artificial neural networks. The latter are used as a universal approximation for the unknown interaction forces between pedestrians. We train the artificial neural network simultaneously with other parameters arising in the model by utilizing a tailored cost function and stochastic gradient techniques. We test our approach using real data sets for the unidirectional and bidirectional flow in corridors and point out the advantages and drawbacks of the proposed approach.

1 Introduction

The modeling of crowd dynamics provides a useful tool for the evacuation or capacity planning problem. A good overview of existing literature on pedestrian flow models can be found in [2, 3, 8, 13], where mainly two classes of modeling approaches, i.e. microscopic and macroscopic models, are distinguished. Microscopic pedestrian models typically rely on Newton-type dynamics, e.g. in [8, 19, 26] or cellular automata, e.g. in [6, 23, 29], while macroscopic models can be either derived via limiting processes [8, 10, 15, 24] or phenomenologically, see e.g. [18, 21, 26, 27]. Starting from a microscopic level, model extensions include, for example, vision cones [11], shortest-path information [12], and stochastic velocities [10, 31, 32].

In fact, all kinds of models are typically based on information about the pedestrians' behavior such as their maximal acceleration, comfort velocity, interaction with other pedestrians, and obstacles. If these parameters are well-known, the models can be used to predict reliably the movements of pedestrians. The latter is an important issue for analyzing capacities of buildings and the detection of safe escape routes.

S. Göttlich (✉) · S. Knapp
Department of Mathematics, University of Mannheim, Mannheim, Germany
e-mail: goettlich@uni-mannheim.de; stknapp@mail.uni-mannheim.de

L. Gibelli (ed.), *Crowd Dynamics, Volume 2*, Modeling and Simulation in Science, Engineering and Technology, https://doi.org/10.1007/978-3-030-50450-2_2

Unluckily, it is hard to obtain good approximations of the modeling parameters and the parameters should be estimated using real data from studies about pedestrian dynamics.[1] Real data should be used carefully since behavior of humans may additionally depend on further influences, i.e. cultural aspects or panic. A good overview of issues concerning parameter estimation can be found in [1, 22, 30].

In this contribution, we aim to estimate the pedestrian interaction forces for the unidirectional and bidirectional flow in corridors for a microscopic pedestrian model. In contrast to [7, 14], where a Bayesian probabilistic method has been applied for the estimation, we focus on artificial neural networks [4]. In a very recent result by Tordeux [33], artificial neural networks have been used for the estimation of pedestrian speed in corridors and bottleneck situations. In our case, the artificial neural network is a building block in a physically motivated model, the so-called social force model, and is used as a function approximation. The application is non-classical in the sense that we do not intend to match a given input with the outcome of the artificial neural network. More precisely, the compared output depends on unknown parameters which we aim to find during the training of the model. Due to this structure, we have to deal with a non-classical cost function and the parameter identification gets more involved. Since parameter estimation for pedestrian flow models is an emerging research area, in particular in combination with real data, it opens new challenging questions from a theoretical and computational viewpoint. We try to address those while presenting our numerical results.

2 Parameter Estimation

This section is devoted to the application of artificial neural networks for parameter estimation in the pedestrian flow model by Helbing and Molnár. To do so, we first introduce the modeling details and explain the mathematical framework of the artificial neural network. Since our intention is to estimate the interaction between pedestrians, the crucial point will be the computation of weights for the artificial neural network.

Inspired by the social force model [19], our investigations are based on Newton-type microscopic equations which describe the movement and acceleration of every pedestrian due to obstacles, destinations, and interaction forces. Let us consider $i = 1, \dots, N, N \in \mathbb{N}$, pedestrians having positions $X_i(t) \in \mathbb{R}^2$ and velocities $V_i(t) \in \mathbb{R}^2$ at time $t \geq 0$. This means that we look from top on to pedestrians, which are represented by their center of mass $X_i(t)$ and their direction of movement $V_i(t)$. In general, the model can be written as

$$\begin{aligned} \frac{d}{dt} X_i(t) &= V_i(t), \\ \frac{d}{dt} V_i(t) &= \frac{1}{\tau}(D_i(X_i(t))v_c - V_i(t)) + F(X_i(t), V_i(t), \mathbf{X}_{-i}(t), \mathbf{V}_{-i}(t)) \\ &\quad + F^w(X_i(t), V_i(t)), \end{aligned} \tag{1}$$

[1] http://ped.fz-juelich.de/database/.

where $\tau > 0$ denotes a relaxation time describing how fast pedestrians achieve their comfort velocity, D_i is the unit vector towards the destination, and $v_c > 0$ is the comfort velocity. The forces F and F^w describe the acceleration caused by interaction with other pedestrians, where $z_{-i} = (z_1, \dots, z_{i-1}, z_{i+1}, \dots, z_N)$, and the acceleration caused by walls or obstacles, respectively.

The component $D_i(X_i(t))v_c$ describes the desired velocity vector given that there are no pedestrians and walls around pedestrian i. Hence, $\frac{1}{\tau}(D_i(X_i(t))v_c - V_i(t))$ is an acceleration along the direction of the desired velocity vector of pedestrian i. Assuming that $D_i(X_i(t)) = D_i$ is independent of the position then, in the absence of other pedestrians and walls, we get

$$\frac{d}{dt}V_i(t) = \frac{1}{\tau}(D_i v_c - V_i(t)) \tag{2}$$

with solution $V_i(t) = D_i v_c + e^{-\frac{t}{\tau}}(V_i(0) - D_i v_c)$. That means, $\frac{1}{\tau}$ is the decay rate of the initial deviation away from the desired velocity vector $v_c D_i$. Regarding the interaction acceleration function F, it seems unpromising estimating this high dimensional function, i.e. for large N, by using data. Therefore, we assume the following structure of F:

$$F(X_i(t), V_i(t), \mathbf{X}_{-i}(t), \mathbf{V}_{-i}(t)) = \sum_{k=1,k\neq i}^{N} G(X_i(t) - X_k(t), V_i(t) - V_k(t)). \tag{3}$$

In fact, this means that we assume identical reactions of every single pedestrian given the other pedestrians. Common choices for G in the literature are potentials [15, 28] equipped with a vision cone [11]. In [28], the repulsive forces are given by a parameterized (Morse-type) potential and the parameters are estimated using data to solve a minimization problem with quadratic costs. Reasonable assumptions are made but the shape of the function G has to be proposed. However, we will assume that Eq. (3) holds and our goal is to recover $G\colon \mathbb{R}^4 \to \mathbb{R}^2$ using data for an artificial neural network as an approximation tool.

The acceleration at boundaries given by the function F^w is crucial since obstacles and walls can be assumed as solid objects and these objects represent a reflective boundary. This fact can be only incorporated into F^w by considering unbounded accelerations, which makes the system (1) very hard to solve. One way out of this has been introduced in [15], where the boundary is incorporated by manipulating the realized velocity $\frac{d}{dt}X_i(t) = \mathcal{V}(X_i(t), V_i(t))$. Then, $V_i(t)$ can be interpreted as the desired, but maybe not realized, velocity in the presence of obstacles. For this reason, we neglect the wall forces in the rest of this manuscript and comment on boundary treatment individually in examples.

Summarizing, we need a general tool to fit the function G appropriately. If we assume a potential again, e.g. a Morse potential, and try to find the parameters, we assume too much and could be wrong. Therefore, we choose an artificial neural network as function approximation for the function G. For completeness, we introduce artificial neural networks in the following and state relevant results for our purpose.

2.1 *Artificial Neural Network as Universal Function Approximation*

Unlike existing contributions for the parameter estimation in pedestrian models [7, 14, 28], we focus on artificial neural networks for the parameter estimation. Motivated by recent works for neural networks applied to ordinary differential equations [5, 16], we will now embed our parameter estimation problem into this context.

2.2 *Setting Up an Artificial Neural Network*

We consider feed forward artificial neural networks here, which means that the input given into the input layer is fed forward through the network only, i.e. there exists no connections backwards. More detailed, let L be the number of layers including the input and hidden layer (Fig. 1).

The value of the so-called neurons $a_k^{(l)}$ in layer l and neuron k is computed as follows:

Input Layer

$$a_1^{(1)} = 1, \quad a_k^{(1)} = x_{k-1} \tag{4}$$

for $k \in \{2, \dots, n^{(1)} + 1\}$, where $x \in \mathbb{R}^{n^{(1)}}$ is the input (feature) and $n^{(1)}$ is the number of neurons without the bias unit $a_1^{(1)}$.

Hidden Layers

$$a_1^{(l)} = 1, \quad a_k^{(l)} = g^{(l)} \left(\sum_{\tilde{k}=1}^{n^{(l-1)}+1} \theta_{\tilde{k},k}^{(l-1)} a_{\tilde{k}}^{(l-1)} \right) \tag{5}$$

for $l \in \{2, \dots, L-1\}$ and $k \in \{2, \dots, n^{(l)} + 1\}$.

Output Layer

$$a_k^{(L)} = g^{(L)} \left(\sum_{\tilde{k}=1}^{n^{(L-1)}+1} \theta_{\tilde{k},k}^{(L-1)} a_{\tilde{k}}^{(L-1)} \right) \tag{6}$$

for $k \in \{1, \dots, n^{(L)}\}$. One has to recognize that the output layer does not contain a bias unit, i.e. where a fixed value is set to 1. The entry $\theta_{i,j}^{(l)}$ of the matrices $\theta^{(l)} \in \mathbb{R}^{n^{(l-1)} \times n^{(l)}}$ describes the weight from neuron $a_i^{(l-1)}$ to the value of the neuron $a_j^{(l)}$.

Example 1 We consider the case of a single hidden layer in this example, i.e. $L = 3$. Further, we assume an output layer size of $n^{(L)} = 1$ here. Then, we can write

$$\begin{aligned} a^{(3)} &= g^{(3)} \left(\sum_{\tilde{k}=1}^{n^{(2)}+1} \theta_{\tilde{k},1}^{(2)} a_{\tilde{k}}^{(2)} \right) \\ &= g^{(3)} \left(\sum_{\tilde{k}=1}^{n^{(2)}+1} \theta_{\tilde{k},1}^{(2)} g^{(1)} \left(\sum_{s=1}^{n^{(1)}+1} \theta_{s\tilde{k}}^{(1)} a_s^{(1)} \right) \right) \\ &= g^{(3)} \left(\theta_{1,1}^{(2)} + \sum_{\tilde{k}=2}^{n^{(2)}+1} \theta_{\tilde{k},1}^{(2)} g^{(1)} \left(\theta_{1,\tilde{k}}^{(1)} + \sum_{s=2}^{n^{(1)}+1} \theta_{s\tilde{k}}^{(1)} x_{s-1} \right) \right). \end{aligned}$$

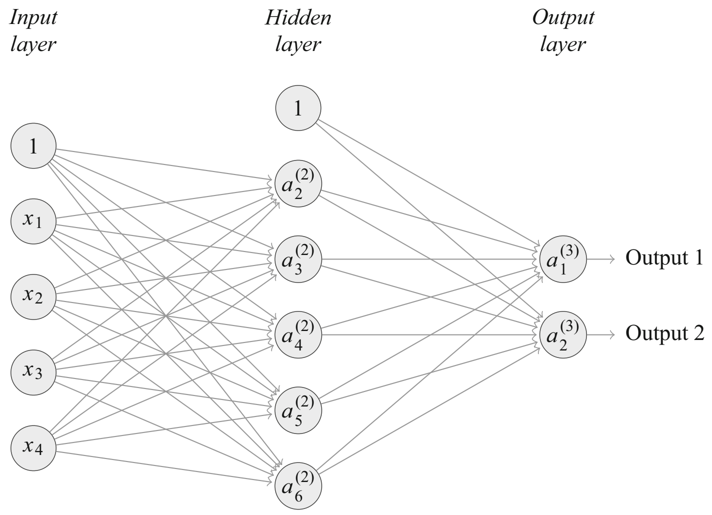


Fig. 1 Graphical representation of a feed forward artificial neural network with one hidden layer

Hence, the output a^3 is a nested sum of weighted evaluations of functions of the input x. A quite common choice is $g^{(L)}(z) = z$ as the identity. This representation allows for an increasing size of the hidden layer $n^{(2)}$ that we can profit from a special class of approximation functions. Indeed, this is the result of the so-called universal approximation theorems:

Theorem 1 ([9, 20])

1. *Let f be a continuous function on $[0,1]^{n^{(1)}}$ and $\epsilon > 0$ as well as the activation function $g^{(1)}$ is assumed to be a continuous sigmoid function, i.e. $\lim_{t\to\infty} g^{(1)}(t) = 1$ and $\lim_{t\to-\infty} g^{(1)}(t) = 0$. Then, there exists an artificial neural network in the form of*

$$a^{(3)}(x) = \sum_{\tilde{k}=2}^{n^{(2)}+1} \theta^2_{\tilde{k},1} g^{(1)} \left(\theta^{(1)}_{1,\tilde{k}} + \sum_{s=2}^{n^{(1)}+1} \theta^1_{s\tilde{k}} x_{s-1} \right)$$

such that

$$\|a^{(3)} - f\|_\infty < \epsilon.$$

2. *Let $f \in L^p(\mu)$, where μ is a finite measure on $\mathbb{R}^{n^{(1)}}$ and $\epsilon > 0$ as well as the activation function $g^{(1)}$ is assumed to be unbounded and nonconstant. Then, there exists an artificial neural network in the form of*

$$a^{(3)}(x) = \sum_{\tilde{k}=2}^{n^{(2)}+1} \theta^2_{\tilde{k},1} g^{(1)} \left(\theta^{(1)}_{1,\tilde{k}} + \sum_{s=2}^{n^{(1)}+1} \theta^1_{s\tilde{k}} x_{s-1} \right)$$

such that

$$\left(\int_{\mathbb{R}^{n^{(1)}}} |a^{(3)}(x) - f(x)|^p \mu(dx) \right)^{\frac{1}{p}} < \epsilon.$$

Theorem 1 implies that we can find an artificial neural network with one single hidden layer such that a function is approximated appropriately in this way.

In order to obtain the weights for approximating functions we use a minimization problem. To do so, let $h_\theta(x) = (a_k^{(L)})$ be the output of the network for a given input x. Then, we define the cost function as

$$J(\theta) = \frac{1}{m} \left(\sum_{i=1}^{m} C(h_\theta(x^{(i)}), y^{(i)}) + \lambda \mathcal{R}(\theta) \right), \tag{7}$$

where $C\colon \mathbb{R}^{n^{(L)}} \times \mathbb{R}^{n^{(L)}} \to \mathbb{R}$ describes the cost of the difference between the feature value $x^{(i)}$ and the measured output $y^{(i)}$, $\lambda \geq 0$, and $\mathcal{R}\colon \mathbb{R}^{(n^{(1)}+1)\times(n^{(2)}+1)} \times \cdots \times \mathbb{R}^{n^{(L-1)+1} \times n^{(L)}} \to \mathbb{R}$ is a regularization of the parameters θ.

Typical choices are quadratic costs

$$C(z, y) = \frac{1}{2}\|z - y\|_2^2$$

and

$$\mathcal{R}(\theta) = \frac{1}{2}\sum_{l=1}^{L-1}\sum_{i=2}^{n^{(l)}+1}\sum_{j=1}^{n^{(l+1)}+1}\left(\theta_{i,j}^{(l)}\right)^2, \tag{8}$$

where the bias units $\theta_{1,j}^{(l)}$ are not considered in the regularization $\mathcal{R}(\theta)$.

In order to obtain an approximation for a given function $f\colon \mathbb{R}^{n^{(1)}} \to \mathbb{R}^{n^{(L)}}$, we use a training set $x^{(i)} \in \mathbb{R}^{n^{(1)}}$, $i \in \{1, \dots, m\}$ with $m \in \mathbb{N}$ samples and define the output as $y^{(i)} = f(x^{(i)})$. Then, we solve

$$\min_{\theta} J(\theta)$$

to obtain an optimal value θ^* such that the artificial neural network approximates the function f well in the sense of the costs J. It is clear that the cost function J might have several local minima leading to difficulties in finding the global solution.

We briefly comment on the choice of activation functions $g^{(l)}$. Although there exist various choices, it is quite common to choose $g^{(L)}(z) = z$ as the identity and the other $g^{(l)}$ as the sigmoid function $g^{(l)}(z) = \frac{1}{1+e^{-z}}$, as the rectified linear unit (ReLU) $g^{(l)}(z) = \max(z, 0)$ or as the smoothed version of the latter, i.e. the softplus or SmoothReLU function $g^{(l)}(z) = \ln(1 + e^z)$. The identity $g^{(L)}(z) = z$ in the last layer allows the space $\mathbb{R}$ as the image of the artificial neural network because the output is then given as linear combination of the previous layer. Sigmoid activation functions $g^{(l)}(z) = \frac{1}{1+e^{-z}}$ can be basically used to mimic decisions (true or false) due to their shape. The rectified linear unit $g^{(l)}(z) = \max(z, 0)$ is bio-inspired, see e.g. [17] and has been used successfully in artificial neural networks for a faster training of the network. Due to the lack of differentiability at $z = 0$ smoothed versions like the SmoothReLU mentioned above are common alternatives.

2.2.1 Gradient Descent and Related Algorithms

In order to minimize the cost function (7), we use a descent gradient approach. Let us denote by $\alpha \in \mathbb{R}^K$ the K parameters to fit and we assume a general cost function J in the form of

$$J(\alpha) = \frac{1}{m}\left(\sum_{i=1}^{m} Q_i(\alpha)\right), \tag{9}$$

where $Q_i(\alpha)$ denotes the cost of sample i associated with the parameters α. The negative gradient $-\nabla J(\alpha)$ indicates the steepest descent of J at the point α and minimizing J can be therefore achieved by "walking" into the latter direction of the steepest descent. Let $\eta > 0$ be a parameter scaling of the step size and $\alpha_0 \in \mathbb{R}^k$ be given, then

$$\alpha_{k+1} = \alpha_k - \eta \nabla J(\alpha_k)$$

provides an iteration α_k, which might end up in a local minimum of the cost function J. Here, we face several problems:

- the computation time of $\nabla J(\alpha_k)$ is too high, which is caused by a large number m of samples or a costly computation of ∇Q,
- the iteration α_k might not converge,
- the algorithm ends up in a local minimum.

The first item can be tackled by considering the so-called stochastic gradient or mini batch gradient descent schemes. The idea is as follows: Starting with a randomly chosen subset I_k of $\{1, \dots, m\}$ containing $\tilde{m} \leq m$ elements, we adapt the iteration of the gradient descent in the following way:

$$\begin{aligned}\alpha_{k+1} &= \alpha_k - \eta \frac{1}{\tilde{m}}\left(\sum_{i \in I_k} \nabla Q_i(\alpha_k)\right)\\ &=: \alpha_k - \eta \nabla J_k^{\tilde{m}}(\alpha_k).\end{aligned}$$

If we choose $\tilde{m} = m$, then we obtain the so-called full batch gradient descent, which is the classical gradient descent algorithm.

A very crucial parameter is the step size, or so-called learning rate η. If η is chosen too large, the iteration might diverge and, in contrast, if η is too small, it takes a large amount of iterations to reach some local minimum. Therefore, efficient heuristics have been developed to choose and update the learning rate. A brief overview can be found in e.g. [35] as well as in.[2] A first step is the use of ADAGRAD, which assumes an individual learning rate for every parameter, i.e.

[2]https://ruder.io/optimizing-gradient-descent/index.html#visualizationofalgorithms.

$$\alpha_{k+1,i} = \alpha_{k,i} - \frac{\eta}{\sqrt{\sum_{l=0}^{k} (\nabla J_l^{\tilde{m}}(\alpha_l)_i)^2}} \nabla J_k^{\tilde{m}}(\alpha_k)_i$$

for $i = 1, \ldots, m$. This means that the learning rates decay fast in the presence of high gradients and cannot increase anymore. Thus, as soon as a direction has a very small learning rate, the algorithm cannot improve this direction appropriately. To overcome this issue, the algorithm ADAGRAD has been developed in [35] by considering a finite accumulate over window. Let $E[g^2]_k$ satisfy $E[g^2]_{k+1} = \rho E[g^2]_k + (1-\rho)(\nabla J_k^{\tilde{m}}(\alpha_k))^2$ for $E[g^2]_0 = 0$ and $\rho \in (0, 1)$. Then $E[g^2]_k$ adapts to the squared gradient information iteratively with rate ρ. The latter resolves the problem of always decaying learning rates but does not explain how to choose η in a proper way. We define $\Delta\alpha_k = \alpha_{k+1} - \alpha_k$ and the ADADELTA algorithm reads then as follows

$$\begin{aligned} E[g^2]_k &= \rho E[g^2]_{k-1} + (1-\rho)(\nabla J_{k-1}^{\tilde{m}}(\alpha_{k-1}))^2, \\ \Delta\alpha_k &= -\frac{\sqrt{E[\Delta\alpha^2]_{k-1} + \epsilon}}{\sqrt{E[g^2]_k} + \epsilon} \nabla J_k^{\tilde{m}}(\alpha_k), \\ E[\Delta\alpha^2]_k &= \rho E[\Delta\alpha^2]_{k-1} + (1-\rho)\Delta\alpha_k^2, \end{aligned}$$

where $\epsilon > 0$ is a chosen parameter to avoid singular values by division by zero and initially $E[\Delta\alpha^2]_0 = 0$.

Since the presented algorithm only uses local information, it might converge to a local minimum in the presence of several minima, which is often the case considering artificial neural networks. In [25], an additive noise has been added to the gradient, which is damped to zero as the number of iterations increases. This allows for a probability to escape from local minima. Because this is a relevant and non-difficult issue, there is recent research considering multiplicative noise or more advanced noises to improve the probability of escaping a local minimum, see, for example, [34, 36]. In our formulas, we replace the gradient $\nabla J_k^{\tilde{m}}(\alpha_k)$ by $\nabla J_k^{\tilde{m}}(\alpha_k) + N_k$, where $N_k \sim \mathcal{N}(0, \Sigma_k)$ is a multivariate normal distributed random vector with zero mean and covariance matrix Σ_k. According to [25], we choose $(\Sigma_k)_{ii} = \frac{\eta_1}{(1+k)^{\eta_2}}$ for some constants $\eta_1, \eta_2 > 0$ and $(\Sigma)_{ij} = 0$ whenever $i \neq j$.

We briefly comment about the calculation of the gradient of the cost function. An artificial neural network is simply speaking a chain of functions and to obtain the gradient with respect to the parameters θ, the chain rule needs to be applied. This can be done by backpropagation through the network, i.e. starting with the output layer the derivatives are computed according to the chain rule backwards to the first hidden layer. For more information about backpropagation, or computation of the gradient, we recommend [4].

2.3 *Parameter Estimation and Cost Function in Pedestrian Flow Model*

We go back now to the initial motivation that we want to recover the interaction function G between pedestrians from real data. In order to recover the interaction function G, we also need to estimate the remaining parameters τ, v_c simultaneously, which makes the setting different to classical artificial neural network applications. From the data point of view, we will have the measured positions $X_i^k \in \mathbb{R}^2$ at discrete times t_k, where $t_0 < t_1 < \cdots < t_m$. We first discretize the microscopic pedestrian model in the following way:

$$\begin{aligned} V_i(t_k) &= \frac{d}{dt} X_i(t_k) \approx \frac{X_i(t_{k+1}) - X_i(t_{k-1})}{t_{k+1} - t_{k-1}}, \\ S_i(t_k) &= \frac{d}{dt} V_i(t_k) \approx \frac{X_i(t_{k+1}) - 2X_i(t_k) + X_i(t_{k-1})}{\Delta t_k \Delta t_{k-1}}, \end{aligned}$$

where $\Delta t_k = t_{k+1} - t_k$. Therefore, we use $V_i^k = V_i(t_k)$ and $S_i^k = S_i(t_k)$ as approximations of the velocity and acceleration of pedestrian i at time t_k.

If we estimate all D_i, i.e. the destination of pedestrian i, the number of parameters increases significantly and cannot expect good estimations. Therefore, we perform a “pre-processing” and identify the destination by identifying pedestrians destination from trajectories. We will explain the used approximation more detailed while discussing the application examples.

The main difference to the classical learning and application of the artificial neural network is here that we do not have the classical input–output structure. More precisely, we can use the trajectories but have no explicit outputs given. Therefore, we introduce the following cost function, where we denote by $\alpha = (\tau, v_c, \theta)$ the collection of all parameters which we want to estimate from data. The cost function reads as follows:

$$\begin{aligned} \mathcal{J}(\alpha) = \frac{1}{m}\frac{1}{2}\sum_{k=1}^{m}\Big[\sum_{i=1}^{N^k}\Big|\tau S_i^k - (D_i^k v_c - V_i^k) - \tau \sum_{j=1, j\neq i}^{N^k} h_\theta(X_i^k - X_j^k, V_i^k - V_j^k)\Big|^2 \\ + \mathcal{R}(\theta)\Big]. \end{aligned}$$

First of all, we observe the classical structure of the cost function as in (7) that allows for using the toolbox of the stochastic gradient descent algorithms. The regularization $\mathcal{R}$ is the one introduced in (8), which regularizes the parameters θ that do not originate from a bias unit. The first part of the cost function simply results from the ODE system of the trajectories and shows that we want to satisfy the second equation (equation for the acceleration) as good as possible. Here, we multiply the equation for the acceleration by τ to avoid divisions by zero and more stable gradients with respect to this parameter.

2.3.1 Data and Preprocessing of Data

In this last part of the section we comment on the preprocessing of the data. We use the data from the data archive of experimental data from studies about pedestrian dynamics,[3] where next to the videos of the experiments one obtains trajectory data. More precisely, we focus on the corridor unidirectional and bidirectional flow data.[4] The main contents of the trajectory data are

- x and y coordinates at every frame number,
- personal ID identifying the corresponding individual as time evolves,
- the frame rate.

One significant difficulty is that the number of pedestrians and also the set of personal IDs change over time, which is clear since pedestrians enter and leave the corridor. Since we need approximations of the time derivative of the trajectory data, we need at least data of a pedestrian i in frame $k-1$, k, and $k+1$. We first cleaned them out and assigned the corresponding velocity and acceleration values.

In a next step, we identify the destination of each pedestrian by taking the normalized mean velocity direction. To avoid too many different destination directions, we round the latter values, since pedestrians in this scenario have to walk through the corridor and should not change their main direction.

3 Computational Results

First we introduce the setting and parameter used throughout this section. We randomly choose 60% from the data, i.e. from the frames, as the training set. The remaining 40% are used as test set. The initial parameters are $\alpha_0 = (\tau_0, v_{C,0}, \theta_0)$ with $\tau_0 = 0.1$, $v_{C,0}$ as the average norm of the velocities from the test set and θ_0 are uniformly distributed randomly chosen from $[-1, 1]$ with exception of the parameters originating from the last hidden bias unit, which are set to zero. As activation functions in the hidden layer we use smooth rectifier units (softplus) $g^{(2)}(x) = \ln(1 + e^x)$ with derivative $(g^{(2)})'(x) = \frac{1}{1+e^{-x}}$, which is the logistic function. The output layer activation function is given as the identity $g^{(3)}(x) = x$ in our case and we assume the parameter $\lambda = 10^{-2}$ for the regularization.

In the stochastic gradient descent algorithm we use the gradient noise with the parameters $\eta_1 = 0.5$ and $\eta_2 = 0.25$ as proposed in [25]. And according to [35], we use $\rho = 0.95$ and $\epsilon = 10^{-6}$ in the ADADELTA algorithm. The batch sizes will be varied in the examples as well as the size of the hidden layer of the artificial neural network.

[3]http://ped.fz-juelich.de/database/.

[4]https://doi.org/10.34735/ped.2013.5, https://doi.org/10.34735/ped.2013.6.

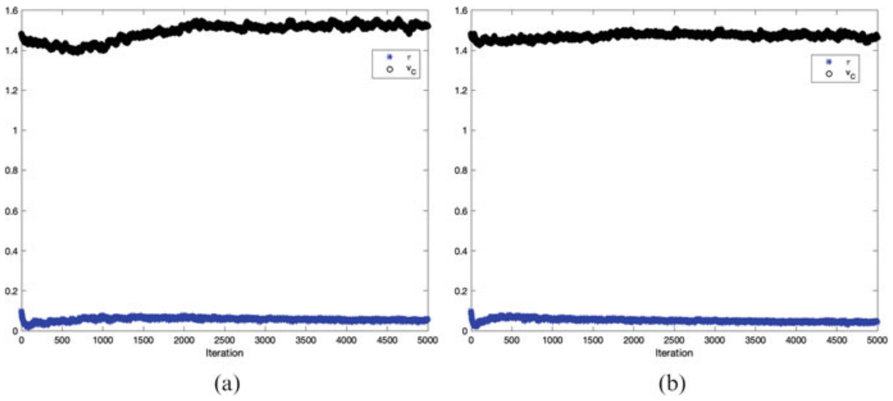


Fig. 2 Value of τ (blue star) and v_c (black circle) during iteration. (**a**) 4 neurons in hidden layer. (**b**) 40 neurons in hidden layer

3.1 Unidirectional Flow in Corridor

We use the data `trajUNICORR50001` from database[5] in the following. It contains the trajectory data for pedestrians walking from the right to the left in a corridor. As one can observe from the video, which can be found also there, we do not have a high amount of interaction between the pedestrians as well as we expect almost no acceleration influences from the data. For this reason, we choose $\tau_0 = 0.1$ here. We choose the configurations of the artificial neural network as $n^{(1)} = 4$, $n^{(3)} = 2$ and $n^{(2)} \in \{4, 40\}$ here.

In Fig. 2 the values of τ and v_c during the stochastic gradient are shown indicating that our initial guess of both parameters is close to the values during the iterations. Also, we observe no severe difference between the case of 4 hidden neurons (Fig. 2a) and 40 hidden neurons (Fig. 2b). Since the calculation of the full batch costs, i.e. the cost function evaluated on the complete training or test set, is very costly from the computational point of view, we only computed the cost function for the initial guess and the guess after 5000 iterations of the stochastic gradient algorithm. In Table 1 the relaxation time τ^* and comfort velocity v_c^* after 5000 stochastic gradient iterations are shown. They are reasonable since v_c^* are around 5–6 km/h. As mentioned before, we do not observe much acceleration in the data such that a small value of τ^* is reasonable as the comfort velocity is achieved very quickly. In the case of 4 neurons in the hidden layer, i.e. $5 \cdot 4 + 5 \cdot 2 = 30$ parameters for G, we have a decrease in the cost function on both, the training and test set. Surprisingly, in the case of 40 neurons in the hidden layer ($5 \cdot 40 + 41 \cdot 2 = 282$ parameters) the cost function increases. This might be due to not enough iterations, small batch size, or a wrong regularization parameter.

[5]https://doi.org/10.34735/ped.2013.6.

Table 1 τ^* and v_c^* after training and cost function evaluated on the test and training set for initial parameter set and after 5000 stochastic gradient iterations with a batch size of 1

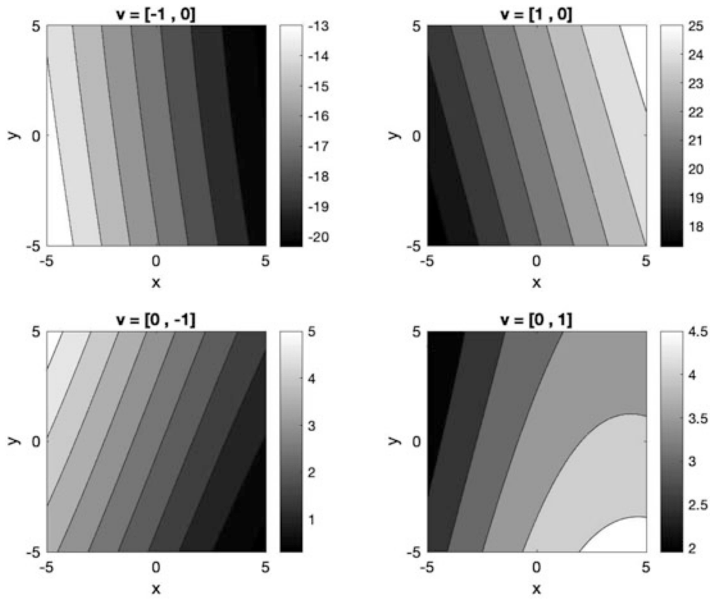
	τ^*	v_c^*	$J_{train}(\alpha_0)$	$J_{test}(\alpha_0)$	$J_{train}(\alpha^*)$	$J_{test}(\alpha^*)$
4 neurons	0.0564	1.5210	517.29	530.08	449.23	451.05
40 neurons	0.0409	1.4650	568.12	578.49	857.89	908.53

Figures 3 and 4 show the approximated function G, which is essentially the artificial neural network. Since the dimension of the input is of dimension 4 and the output's dimension is 2, we have drawn the mappings $(x, y) \mapsto G_1((x, y), v)$ and $(x, y) \mapsto G_2((x, y), v)$ for some fixed $v \in \mathbb{R}^2$ separately. We recover the intuition of G again: G_1 is the attraction or repulsion (depending on the sign) in the x direction whereas G_2 describes the same in the y direction. The inputs of G are the difference in the position, i.e. $X_i - X_k$, which corresponds to the values (x, y) and the difference of the velocities $V_i - V_k$ corresponding to v here. Let us assume pedestrian i is at x position $X_{i,1}(t) > X_{k,1}(t)$ but in y direction they are the same position, i.e. $X_{i,2}(t) = X_{k,2}(t)$, which implies $x > 0$ and $y = 0$. Pedestrians in this experiment walk from the right to the left and we imply by the latter assumption that pedestrian i walks behind pedestrian k. If $v_i = (-2, 0)$ and $v_k = (-1, 0)$, i.e. pedestrian i walks faster, then $v = (-1, 0)$. This situation is shown in Fig. 3 in the left first corners of the subfigures. We observe that the force decreases with x increasing, which is reasonable. One could worry about the values being always negative here but that is a problem of parameter identification since we only have single direction interactions in the data (all walk from right to the left). The plots corresponding to $v = (1, 0)$ show exactly the opposite, i.e. pedestrian i walks slower than pedestrian k. To understand the graphics with $v = (0, -1)$ and $v = (0, 1)$ we consider the following situation:

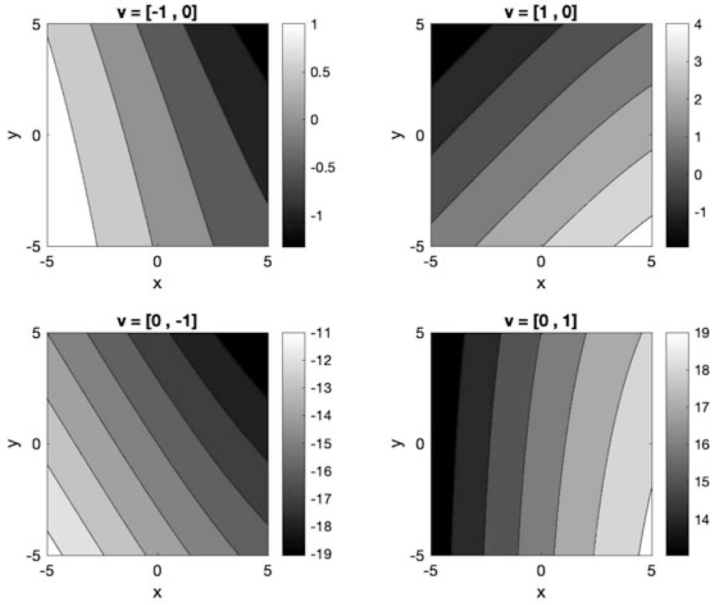
Let pedestrians i and k stay at the same x position but the y coordinate of i is assumed to be greater than the coordinate of k. If $v_i = (-1, -1)$ and $v_k = (-1, 1)$ that means they might run into each other, we obtain $v = (0, -2)$, or normalized $v = (0, -1)$. We see that for fixed $x = 0$ the value of G in x-direction (G_1) increases in y in Fig. 3a. In y-direction (b) it is the opposite as one would expect.

In Fig. 4 the behavior is different. One can observe that the function behaves more nonlinear and also covers an increasing and decreasing behavior as the distance $\|(x, y)\|$ changes. In order to obtain a better result in terms of the cost function, one has to use larger batch sizes as well as more iterations.

Since Figs. 3 and 4 indicate that 4 neurons are too less and 40 need a long time to converge in the stochastic gradient algorithm, we take 10 neurons and work with a batch size of 9. Figure 5 shows the result for G, which looks much better now although we reduced the number of iterations to 2500. Table 2 shows that we have a decreasing cost function again and the optimal values are close to the values in Table 1. A big difference can be seen in Fig. 5, where the shape of the function G fits better to the intuitions stated before. We also see that the x-direction has been learned better than the y-direction, which is clear from the data.



(a)



(b)

Fig. 3 Approximated function $G((x, y), (v_1, v_2))$. (**a**) x-direction with 4 neurons in hidden layer. (**b**) y-direction with 4 neurons in hidden layer

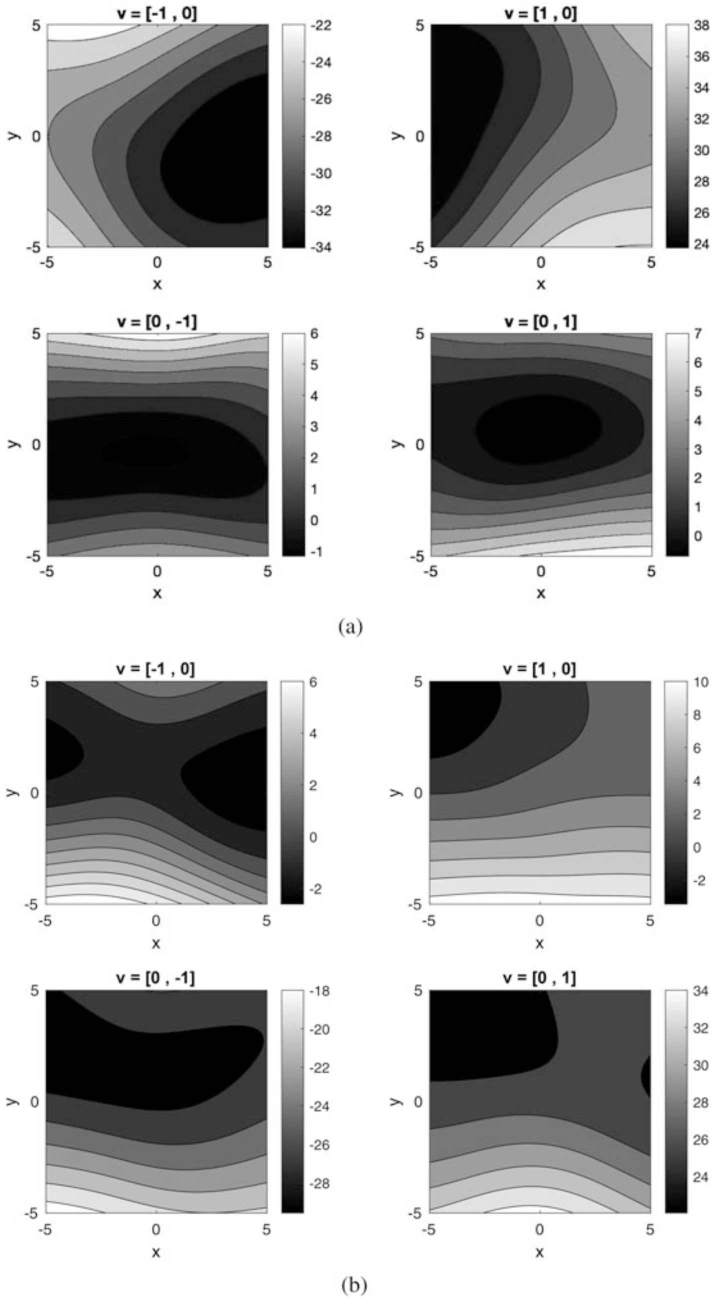


Fig. 4 Approximated function $G((x, y), (v_1, v_2))$. (**a**) x-direction with 40 neurons in hidden layer. (**b**) y-direction with 40 neurons in hidden layer

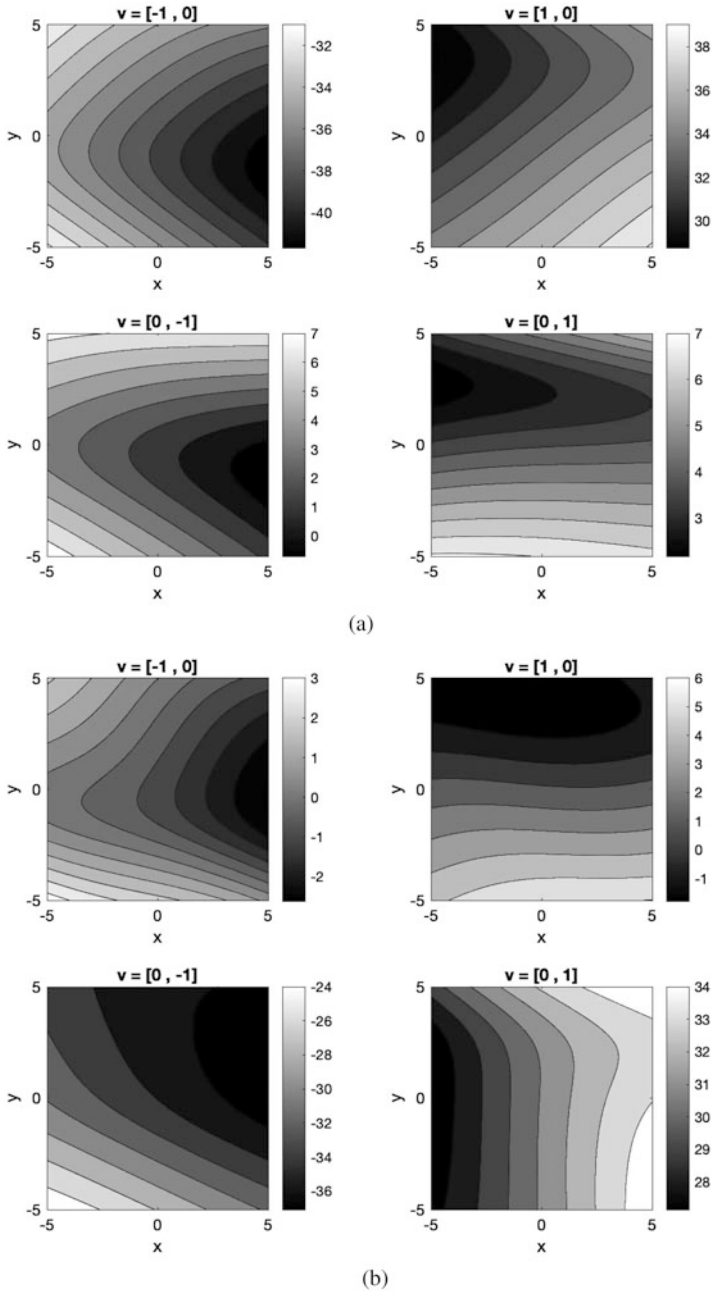


Fig. 5 Approximated function $G((x, y), (v_1, v_2))$. (**a**) x-direction with 10 neurons in hidden layer. (**b**) y-direction with 10 neurons in hidden layer

Table 2 τ^* and v_c^* after training and cost function evaluated on the test and training set for initial parameter set and after 2500 stochastic gradient iterations with a batch size of 9

	τ^*	v_c^*	$J_{train}(\alpha_0)$	$J_{test}(\alpha_0)$	$J_{train}(\alpha^*)$	$J_{test}(\alpha^*)$
10 neurons	0.0391	1.4683	556.61	567.52	493.79	502.09

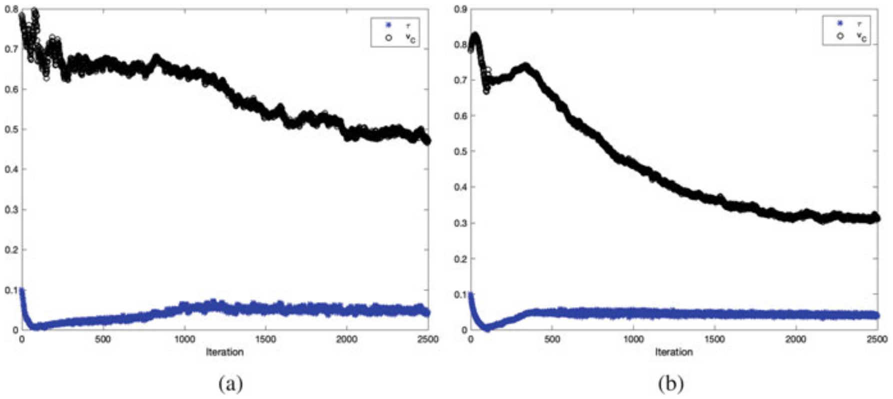


Fig. 6 Value of τ (blue star) and v_c (black circle) during iteration. (a) 4 neurons in hidden layer and batch size 1. (**b**) 10 neurons in hidden layer and batch size 9

3.2 Bidirectional Flow in Corridor

We also have a look at a second data set called `BICORR400A1`.[6] Here, two groups of pedestrian run against each other in a corridor, i.e. we have pedestrians walking to the right and to the left. As before, we use a batch size of one for the stochastic gradient algorithm and 4 neurons in the hidden layer. Due to computational limitations, we restrict on 2500 iterations here. Figure 6 shows the evolution of the parameters τ and v_c during the stochastic gradient, where left 4 neurons and a batch size of 1 have been used and on the right 10 neurons and a batch size of 9. The values for τ do not change significantly anymore and the values of v_c tend to being flat as well.

Table 3 contains the values for τ and v_c after the iterations. Since the number of pedestrians in the corridor is high, we have a lower average speed, which implies a lower v_c^*. The cost function value is lower for α^* than for the initial configuration α_0 and Fig. 7 indicates that the function G has been fitted in a better way. More detailed, we always have the increasing or decreasing behavior in x direction.

Considering the interaction function G for 10 neurons and using a larger batch size of 9, we see again a more nonlinear behavior, see Fig. 8. More detailed, Fig. 8a indicates that the interaction in x-direction, i.e. G_1 depends strongly on y compared to Fig. 7a. This is based on the data since the pedestrians form two walking blocks of

[6]https://doi.org/10.34735/ped.2013.6.

Table 3 τ^* and v_c^* after training and cost function evaluated on the test and training set for initial parameter set and after 2500 stochastic gradient iterations with a batch size of 9

	τ^*	v_c^*	$J_{train}(\alpha_0)$	$J_{test}(\alpha_0)$	$J_{train}(\alpha^*)$	$J_{test}(\alpha^*)$
4 neurons	0.0438	0.4715	500.68	513.71	449.23.71	451.05.09
10 neurons	0.0373	0.3112	–	–	–	–

groups: one walking to the left and one walking to the right and almost no mixture in the corridor. Figure 8 shows a quite similar behavior as it is the case for 4 neurons.

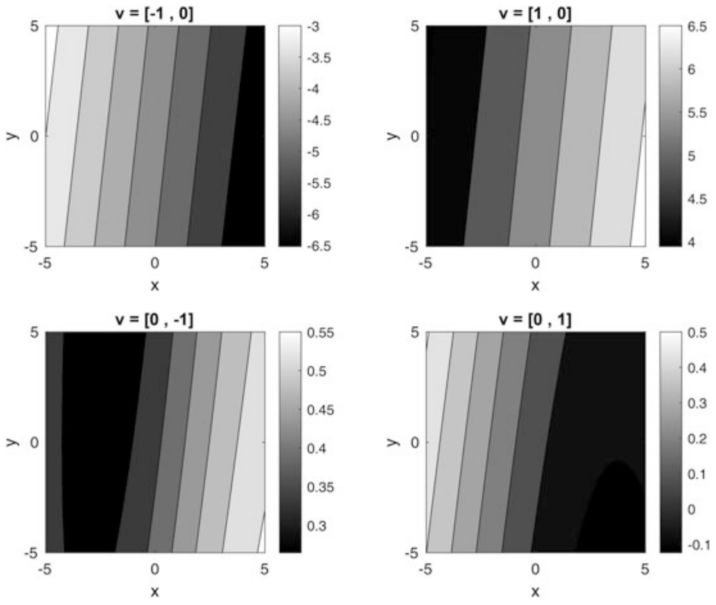
Summarizing, the choice of the number of neurons severely affects the results as one would expect. Also the type of data strongly influences the results. This is due to effects, which are or are not covered in the data.

4 Conclusion

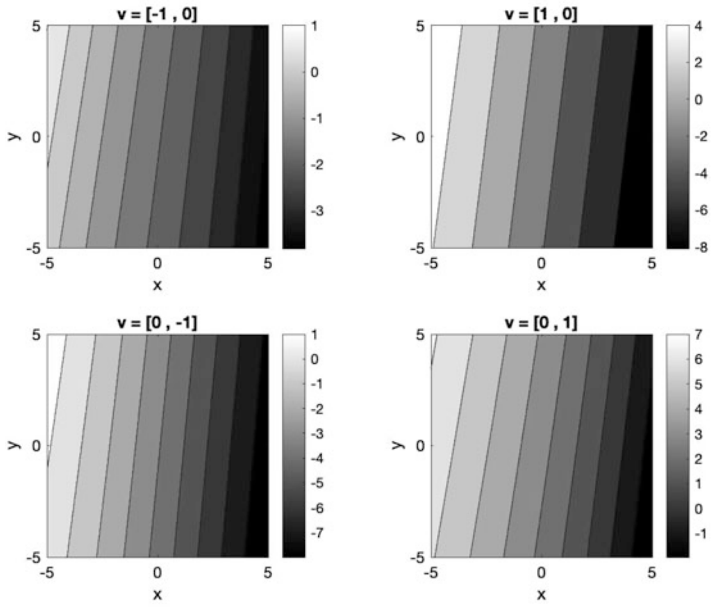
We have derived a data fit setting for the social force model, where the interaction is based on an artificial neural network. In addition, we have introduced the numerical treatment, including stochastic gradient descent heuristics and data preprocessing. The quality of the data fit severely depends on the number of artificial neurons and the number of stochastic gradient iterations as we have seen in results based on real data sets. Due to computational restrictions one cannot significantly increase the number of artificial neurons and the number of iterations simultaneously. From the results we also conclude that the interaction strongly depends on the velocity of both interacting pedestrians as well as their distance from each other.

In terms of computation time, the minimization of the cost function took about 2–3 h in the case of 4 neurons and a batch size of one, whereas the batch size of 9 and 10 neurons required 12 h (unidirectional) and 3 days (bidirectional) to reach the number of iterations on a standard desktop computer. The computation time also depends strongly on the number of pedestrians, which are present in a sample since the number of evaluations increases quadratically in the number of pedestrians. One possible way out of the computational restrictions might be the use of well-developed artificial neural network implementations and embed them into the presented model.

Summarizing, there is a recent trend to carry out data-driven approaches to estimate parameters in crowd models. So far, the Bayesian probabilistic approach [7, 14] and the artificial neural networks approach [33] are the most common tools in the literature. Both approaches are characterized by the computational evaluation of special optimization problems with differential equations as constraints and are therefore computationally costly. Furthermore, the choice of other problem-specific parameters and a reasonable initial guess might be crucial to numerically compute reasonable solutions. From a theoretical point of view, it seems that the Bayesian

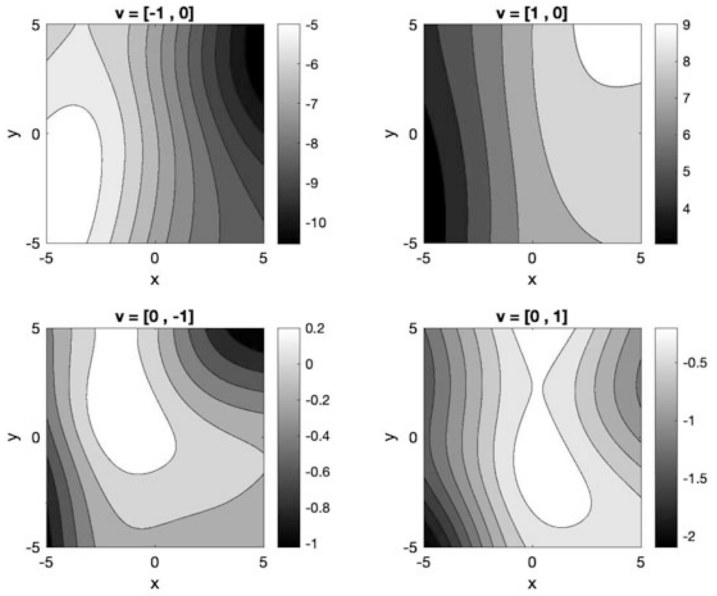


(a)

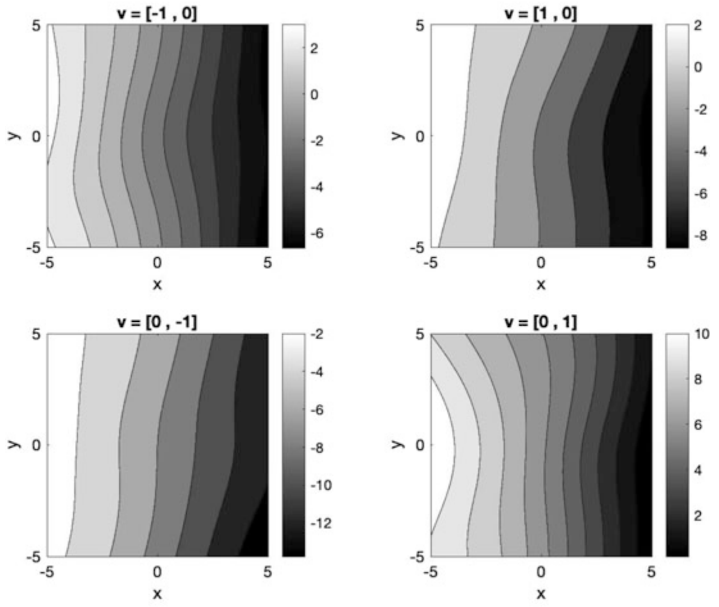


(b)

Fig. 7 Approximated function $G((x, y), (v_1, v_2))$. (**a**) x-direction with 4 neurons in hidden layer. (**b**) y-direction with 4 neurons in hidden layer



(a)



(b)

Fig. 8 Approximated function $G((x, y), (v_1, v_2))$. (**a**) x-direction with 10 neurons in hidden layer. (**b**) y-direction with 10 neurons in hidden layer

approach can rely on more results, while the theory on artificial neural network is rather in the early stages. However, a direct comparison of both approaches based on the same setting is still missing.

Acknowledgments This work was supported by DFG grants No. GO 1920/7-1, GO 1920/10-1 and the DAAD project "Stochastic dynamics for complex networks and systems" (Project-ID 57444394).

References

1. R.C. Aster, B. Borchers, C.H. Thurber, *Parameter Estimation and Inverse Problems*, 2nd edn. (Elsevier/Academic Press, Amsterdam, 2013)
2. N. Bellomo, C. Dogbe, On the modeling of traffic and crowds: a survey of models, speculations, and perspectives. SIAM Rev. **53**, 409–463 (2011)
3. N. Bellomo, C. Bianca, V. Coscia, On the modeling of crowd dynamics: an overview and research perspectives. SeMA J. **54**, 25–46 (2011)
4. C.M. Bishop, Pattern Recognition and Machine Learning. Information Science and Statistics (Springer, New York, 2006)
5. R.T.Q. Chen, Y. Rubanova, J. Bettencourt, D. Duvenaud, Neural ordinary differential equations (2018). arXiv:1806.07366
6. A. Chertock, A. Kurganov, A. Polizzi, I. Timofeyev, Pedestrian flow models with slowdown interactions. Math. Models Methods Appl. Sci. **24**, 249–275 (2014)
7. A. Corbetta, A. Muntean, K. Vafayi, Parameter estimation of social forces in pedestrian dynamics models via a probabilistic method. Math. Biosci. Eng. **12**, 337–356 (2015)
8. E. Cristiani, B. Piccoli, A. Tosin, *Multiscale Modeling of Pedestrian Dynamics*. Modeling, Simulation and Applications, vol. 12 (Springer, Cham, 2014)
9. G. Cybenko, Approximation by superpositions of a sigmoidal function. Math. Control Signals Syst. **2**, 303–314 (1989)
10. P. Degond, C. Appert-Rolland, M. Moussaïd, J. Pettré, G. Theraulaz, A hierarchy of heuristic-based models of crowd dynamics. J. Stat. Phys. **152**, 1033–1068 (2013)
11. P. Degond, C. Appert-Rolland, J. Pettré, G. Theraulaz, Vision-based macroscopic pedestrian models. Kinet. Relat. Models **6**, 809–839 (2013)
12. R. Etikyala, S. Göttlich, A. Klar, S. Tiwari, Particle methods for pedestrian flow models: from microscopic to nonlocal continuum models. Math. Models Methods Appl. Sci. **24**, 2503–2523 (2014)
13. L. Gibelli, N. Bellomo, (eds.), *Crowd Dynamics. Vol. 1. Theory, Models, and Safety Problems*. Modeling and Simulation in Science, Engineering and Technology (Birkhäuser/Springer, Cham, 2018)
14. S.N. Gomes, A.M. Stuart, M.-T. Wolfram, Parameter estimation for macroscopic pedestrian dynamics models from microscopic data. SIAM J. Appl. Math. **79**, 1475–1500 (2019)
15. S. Göttlich, S. Knapp, P. Schillen, A pedestrian flow model with stochastic velocities: microscopic and macroscopic approaches. Kinet. Relat. Models **11**, 1333–1358 (2018)
16. E. Haber, L. Ruthotto, Stable architectures for deep neural networks. Inverse Problems **34**, 014004 (2017)
17. R.H.R. Hahnloser, R. Sarpeshkar, M.A. Mahowald, R.J. Douglas, H.S. Seung, Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature **405**, 947–951 (2000)
18. D. Helbing, A fluid dynamic model for the movement of pedestrians. Complex Syst. **6**, 391–415 (1992)

19. D. Helbing, P. Molnár, Social force model for pedestrian dynamics. Phys. Rev. E **51**, 4282–4286 (1995)
20. K. Hornik, Approximation capabilities of multilayer feedforward networks. Neural Netw. **4**, 251–257 (1991)
21. R.L. Hughes, A continuum theory for the flow of pedestrians. Transp. Res. B **36**, 507–535 (2002)
22. J. Kaipio, E. Somersalo, *Statistical and Computational Inverse Problems*. Applied Mathematical Sciences, vol. 160 (Springer, New York, 2005)
23. A. Kirchner, A. Schadschneider, Cellular automaton simulations of pedestrian dynamics and evacuation processes, in *Traffic and Granular Flow'01*, ed. by M. Fukui, Y. Sugiyama, M. Schreckenberg, D.E. Wolf (Springer, Berlin, 2003), pp. 531–536
24. A. Klar, F. Schneider, O. Tse, Approximate models for stochastic dynamic systems with velocities on the sphere and associated Fokker-Planck equations. Kinet. Relat. Models **7**, 509–529 (2014)
25. A. Neelakantan, L. Vilnis, Q.V. Le, I. Sutskever, L. Kaiser, K. Kurach, J. Martens, Adding gradient noise improves learning for very deep networks (2015). http://arxiv.org/abs/1511.06807v1
26. B. Piccoli, A. Tosin, Pedestrian flows in bounded domains with obstacles. Contin. Mech. Thermodyn. **21**, 85–107 (2009)
27. B. Piccoli, A. Tosin, Time-evolving measures and macroscopic modeling of pedestrian flow. Arch. Ration. Mech. Anal. **199**, 707–738 (2011)
28. C. Rudloff, T. Matyus, S. Seer, D. Bauer, Can walking behavior be predicted? Analysis of calibration and fit of pedestrian models. Transp. Res. Rec. J. Transp. Res. Board **2264**, 101–109 (2011)
29. M. Schultz, *Stochastic Transition Model for Pedestrian Dynamics* (Springer, Cham, 2014), pp. 971–987
30. A. Tarantola, *Inverse Problem Theory and Methods for Model Parameter Estimation* (Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 2005)
31. A. Tordeux, A. Schadschneider, *A Stochastic Optimal Velocity Model for Pedestrian Flow* (Springer, Cham, 2016), pp. 528–538
32. A. Tordeux, A. Schadschneider, White and relaxed noises in optimal velocity models for pedestrian flow with stop-and-go waves. J. Phys. A **49**, 185101 (2016)
33. A. Tordeux, M. Chraibi, A. Seyfried, A. Schadschneider, Prediction of pedestrian speed with artificial neural networks, in *Traffic and Granular Flow'17*, ed. by S.H. Hamdar (Springer, Cham, 2019), pp. 327–335
34. J. Wu, W. Hu, H. Xiong, J. Huan, Z. Zhu, The multiplicative noise in stochastic gradient descent: data-dependent regularization, continuous and discrete approximation (2019). http://arxiv.org/abs/1906.07405v1.
35. M.D. Zeiler, ADADELTA: an adaptive learning rate method (2012). http://arxiv.org/abs/1212.5701v1
36. Z. Zhu, J. Wu, B. Yu, L. Wu, J. Ma, The anisotropic noise in stochastic gradient descent: its behavior of escaping from sharp minima and regularization effects. http://arxiv.org/abs/1803.00195v5