Report for exercise 6 from group K

Tasks addressed:	5
Authors:	SONJA KRAFFT (03681252)
	LUDWIG-FERDINAND STUMPP (03736583)
	TIMUR WOHLLEBER (03742932)
Last compiled:	2022–02–10
Source code:	https://gitlab.lrz.de/00000000014A9334/mlcs-ex6-nn-speed-prediction

The work on tasks was divided in the following way:

SONJA KRAFFT (03681252)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
	Task 5	33%
LUDWIG-FERDINAND STUMPP (03736583)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
	Task 5	33%
TIMUR WOHLLEBER (03742932)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
	Task 5	33%

1 Introduction

Setup

Due to the great selection of publicly available libraries, Python is chosen as the programming language for this exercise. The following is a listing of required software and Python packages in order to run and maintain the code.

As a container for our datasets we use the Python library `pandas`[1] and its class `pandas.DataFrame`. For plotting, we choose the standard `matplotlib.pyplot`[2] and `NumPy`[3] is used for the core mathematical operations. `Tensorflow`[4] with its high-level `Keras`[5] is utilized for the custom artificial neural network implementation and `scikit-learn`[6] is used for the random forest regressor and some standard data management functions.

We choose `JupyterLab`[7] for prototyping and analysis, `black`[8] for style formatting and `mypy`[9] for static typing. These make up our development dependencies, only needed to run and develop the code, not necessarily dependencies of the internal codebase. The used version number can be found in the `requirements-dev.txt` file inside the root of the repository.

python	3.9.5
vscode	1.63.2

Table 1: Python version and code editor.

matplotlib	3.4.3
pandas	1.3.5
scikit_learn	1.0.2
numpy	1.21.3
tqdm	4.62.3
tensorflow	2.7.0

Table 2: List of required Python packages to run the code of the repository. Find the corresponding `requirements.txt` file inside the root of the repository.

[1] https://pandas.pydata.org/
[2] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html
[3] https://numpy.org/
[4] https://www.tensorflow.org/
[5] https://keras.io/
[6] https://scikit-learn.org/stable/
[7] https://jupyter.org/
[8] https://black.readthedocs.io/en/stable/
[9] http://mypy-lang.org/

2 Individual Tasks

Report on task 1/5: Setup Introduction

Notebook: `None`
Code: `None`

Motivation

The prediction of pedestrian speed is a complex undertaking, as a variety of parameters can have an influence on the resulting speed, depending on the scenario. For example, the speed is a function of age, gender, motivation, the distance to be covered and environmental influences such as the weather, the distance to other pedestrians and objects, but also the geometric conditions such as the presence of slopes in the form of ramps or stairs. This small extract of parameters already shows how complex the prediction can become. In practice, a number of different models have been developed for this purpose. In recent years, classic microscopic models have dominated, which are usually determined for a specific scenario and a subset of the previously mentioned influencing parameters. As in robot motion planning, pure data-based approaches are increasingly becoming the focus of scientific research. In the following section, these two approaches will be compared.

Classical Approach: Microscopic Model

Microscopic models have their origin in traffic engineering and usually relate an output variable to a small number of physical input variables. According to the authors these approaches can generally be divided into three categories: decision-based, velocity-based and force-based models. The hyperparameters belonging to the respective model are usually ascertained by conducting empirical studies and is thus associated with considerable effort. In practice, for example, this could be walking speed as a function of people density.

The experience to date with simulations based on microscopic motion models indicate realistic pedestrian flows and are also able to describe self-organization phenomena such as lane formation. Nevertheless for the respective user of the simulation, this not only requires domain knowledge of the respective model and scenario, but is often also associated with a labour-intensive adjustment of the model parameters. In essence, this requires iterative processing of the worksteps illustrated in sketch in Figure 1.

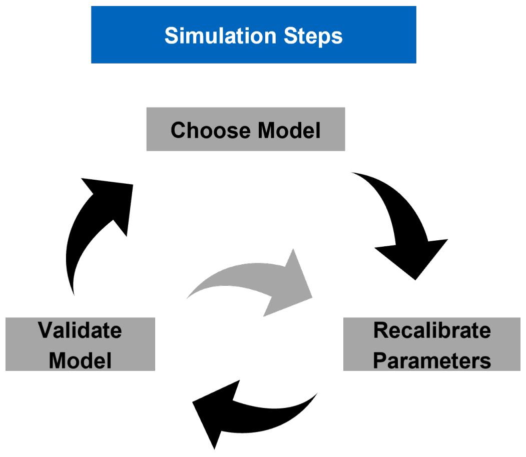


Figure 1: Sketch of the work steps to adjust a model for a certain simulation purpose. The grey boxes illustrate the corresponding work step and the arrows the workflow. The light shadowed arrow indicates that their might be multiple parameter adjustments if the validation on unseen data is not successful.

Datadriven Approach: Artificial Neural Network

The authors explain the motivation of their work by identifying two main weaknesses of the classical approach:

- Problem 1: complex model adjustment to the respective simulation purpose
- Problem 2: limited informative value in complex scenarios

Microscopic models often reach their limits in complex scenarios, as they are often based on only a few parameters that are designed to capture a certain scenario or behaviour. Further, the authors demonstrate the relevance of the topic by showing how often we find ourselves in such complex scenarios in everyday life. Specifically, they illustrate this with a railway station or a football stadium. According to their statement it no longer possible to capture the transition between the variety of elements such as a corridor, T-junction and bottlenecks with a model based on a single fundamental diagram.

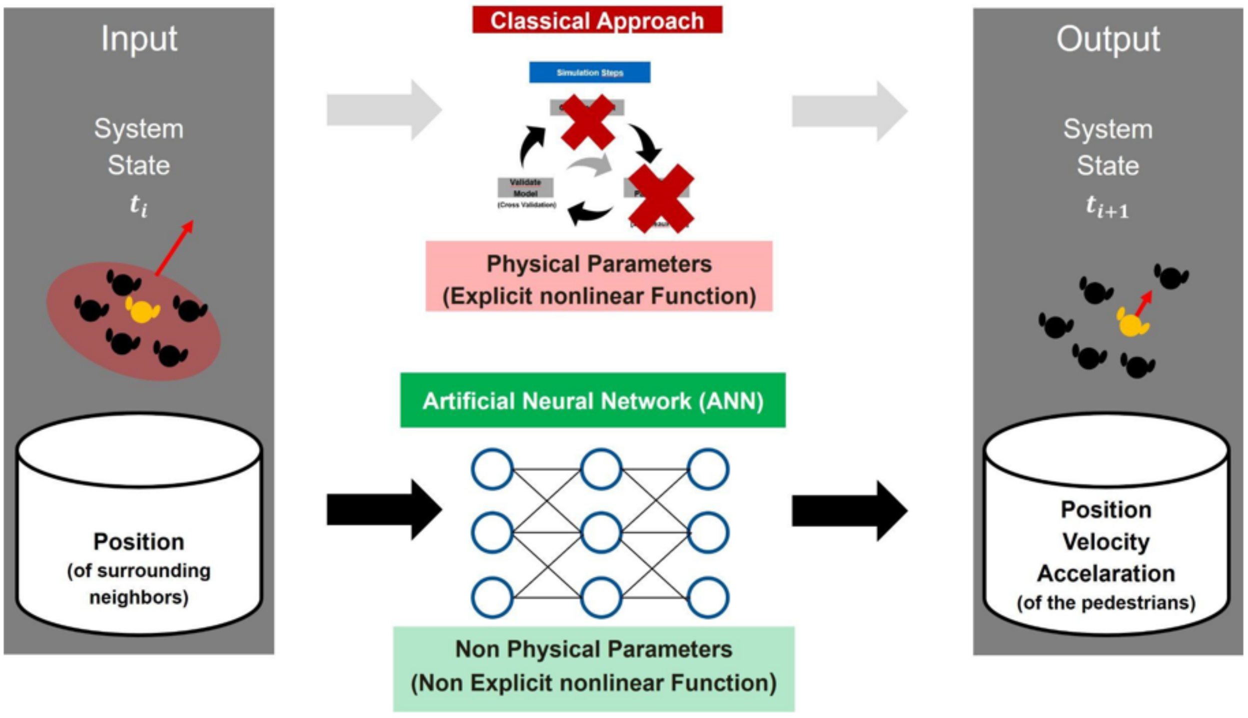

Input
System State t_i
Position (of surrounding neighbors)
Classical Approach
Simulation Steps
Validate Model (Cross Validation)
Physical Parameters (Explicit nonlinear Function)
Artificial Neural Network (ANN)
Non Physical Parameters (Non Explicit nonlinear Function)
Output
System State t_{i+1}
Position
Velocity
Accelaration
(of the pedestrians)

Figure 2: Shows a sketch of the workflow that the authors suggest to determine the pedestrian speed with an Artificial Neuronal Network (ANN) instead of using the classical approach. On the left hand side the input data of the system at time stamp t_i is shown. The data contains the positions and velocities of the surrounding neighbors of the pedestrian highlighted in yellow. On the right hand side the output is shown which contains the position, velocity and acceleration rate of the pedestrians. In between a simple sketch of the ANN is shown.

Instead the authors suggest to use an Artifial Neural Network (ANN) to find a non-explicits nonlinear function that maps the inpute data of the current time stamp t_i to the next time stamp t_{i+1}. The workflow to predict the pedestrian speed is outlined in Figure 2.So the main idea is to avoid the tedious task of finding physically significant parameters for classical models and instead to find a larger number of parameters without physical significance for the ANN with through established optimisation procedures. For this purpose, different architectures are tried out, which are examined in more detail in chapter three. the fundamental diagram according to weidmann serves as a reference for the performance.

Report on task 2/5: Data Pre-Processing

Notebook: `task_2.ipynb`
Code: `helpers.data`, `helpers.plots`

Motivation

To train data-based models of any kind, one needs high-quality data. Unfortunately, certain tasks very often require data that is not readily available in nature. For this purpose, studies are conducted under very strict conditions to collect the necessary data. In most cases, however, the raw data are not sufficient to serve directly as input for model training. Thorough data pre-processing and labeling is a very important and time-consuming prerequisite, as the required features and targets have to be created first. This chapter serves as an overview about the used data, how it has been recorded and which pre-processing steps are necessary

General introduction about the datasets

In their paper *Prediction of Pedestrian Speed with Artificial Neural Networks* [2], Tordeux et. al use two pedestrian datasets created under laboratory conditions by the Civil Safety Research Unit of Forschungszentrum Jülich[10]. They are part of a larger collection of pedestrian experiments that are publicly available on the web[11]. The first involves a closed loop of pedestrians, while the second shows behavior in a bottleneck scenario. The corridor and bottleneck datasets are unidirectional, meaning that all pedestrians are walking in the same direction and both can be found on the open science platform Zenodo[12], where we also obtained the data.

The bottleneck dataset

We specified our analysis in this work only on the bottleneck dataset. In their paper, Tordeux et. al [2] also analyse the corridor dataset and combine set of varying sizes during their model training. In the bottleneck dataset, pedestrians first enter a corridor of width $b_{Korr} = 1.8m$ walking towards a bottleneck of varying length $b_{Ab} = \{0.7m, 0.95m, 1.20m, 1.80m\}$. The number of participants is declared as constant with a total of $N = 150$. Figure 3 shows the setup of the experiment as conducted by the laboratory study[13].

The data of the four different experiments of varying bottleneck width can be found in four different text files containing all data for the specific experiment. Here is an overview about the columns and their respective meaning:

- `ID`: Unique pedestrian identifier, constant during one experiment.
- `FRAME`: Frame number and therefore indicator of time. The time delta between two frames is constant with a value of $1/16s$.
- `X`: X-position of the pedestrian in $[cm]$.
- `Y`: Y-position of the pedestrian in $[cm]$.
- `Z`: Z-position of the pedestrian in $[cm]$.

While the total data as a concatenation of all four different experiment settings has a total shape of $(254371, 6)$, we decided to only use the experiment with the setting $b_{Ab} = 0.7m$ with a total of 75336 rows. Note that one row represents one pedestrian at a time frame during the experiment.

[10] `https://fz-juelich.de/ias/ias-7/EN/Home/home_node.html`
[11] `https://ped.fz-juelich.de/database/doku.php`
[12] `https://zenodo.org/record/1054017`
[13] `https://ped.fz-juelich.de/experiments/2009.05.12_Duesseldorf_Messe_Hermes/docu/VersuchsdokumentationHERMES.pdf`

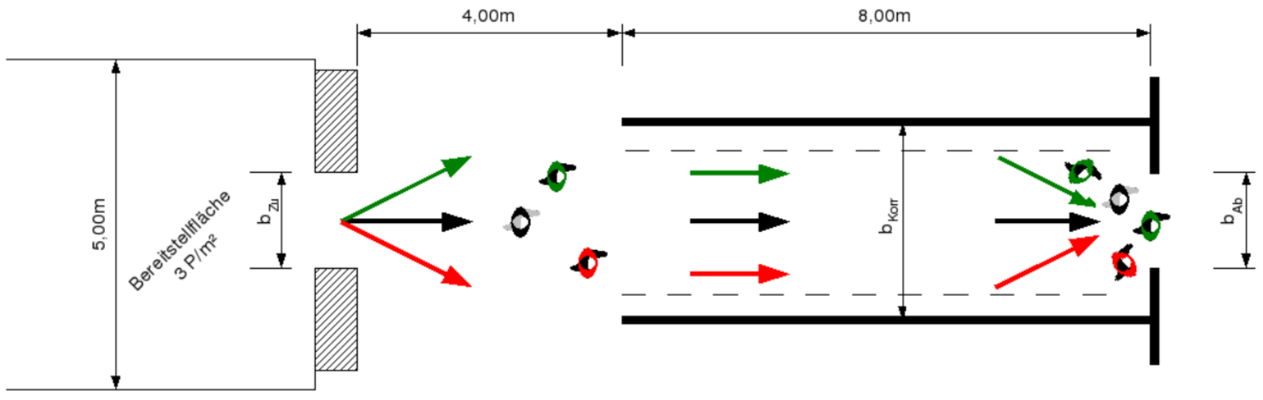


Figure 3: Setup of the bottleneck experiment from Civil Safety Research Unit of Forschungszentrum Jülich. For the available data in our paper, the value $b_{Korr} = 1.8m$, $b_{Ab} = \{0.7m, 0.95m, 1.20m, 1.80m\}$ and a total of N = 150 pedestrians are chosen.

Required transformations on the data

Tordeux et. al [2] are proposing to estimate the scalar value of absolute velocity v by looking at distance-based features of the K-closest pedestrians and training a Neural Network in a supervised fashion:

$$v = \mathrm{NN}(H, \bar{s}_K, (x_i - x, y_i - y, 1 \leq i \leq K)) \tag{1}$$

Here, $\bar{s}_K = \sum_{i=1}^{K} \sqrt{(x - x_i)^2 + (y - y_i)^2}$ is the average distance between the current pedestrian for which we want to predict the velocity and its $K = 10$ - closest neighbors. This makes up for a total of $2K + 1$ features.

Since the raw data does not include any distance-relative features, nor the target we want to predict, one first needs to compute these values and add them as respective columns to the data.

Computing the scalar velocities v

Since both the positions as well as the constant time-deltas Δt are known, one can use central differences to compute the target velocities:

$$v_n = \frac{\sqrt{(x_{n+1} - x_{n-1})^2 + (y_{n+1} - y_{n-1})^2}}{2\Delta t} \tag{2}$$

where all the data is from one unique pedestrian only. For the first and the last measurement, we are using forward and backward differences to calculate the velocity.

Computing the relative distances to K-closest neighbors

The $2K + 1$ features for our model include the average distance $\bar{s}$ to the K closest neighbors as well as values $(\Delta x_i, \Delta y_i)$ for each of the K closest neighbors. In order for the model to generalize, the delta value pairs are sorted by the total distance to the i-th closest neighbor in a ascending order. This results in $(\Delta x_1, \Delta y_1)$ being the spatial distances to the closest neighbor, and $(\Delta x_{10}, \Delta y_{10})$ of the furthest one.

In order to be able to calculate these features, one needs to:

1. for each pedestrian id and time frame:
2. filter all other rows of the same time frame.
3. compute relative distances to all of these rows.

4. sort the neighbors according to their relative distances in an ascending order and only keep the first $K = 10$ which correspond to the closest ones.
5. add all the distance values for these $K = 10$ neighbors as additional columns to the row of the pedestrian we looked at.

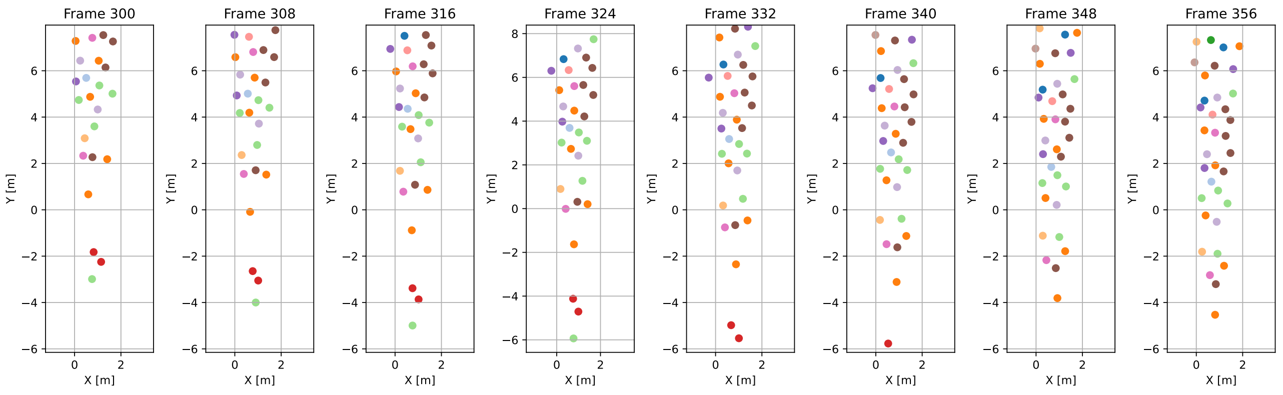
Due to the complexity of $O(N^2)$, the process of calculating the relative distance-based features was very time intensive.

Exploratory analysis on the transformed data

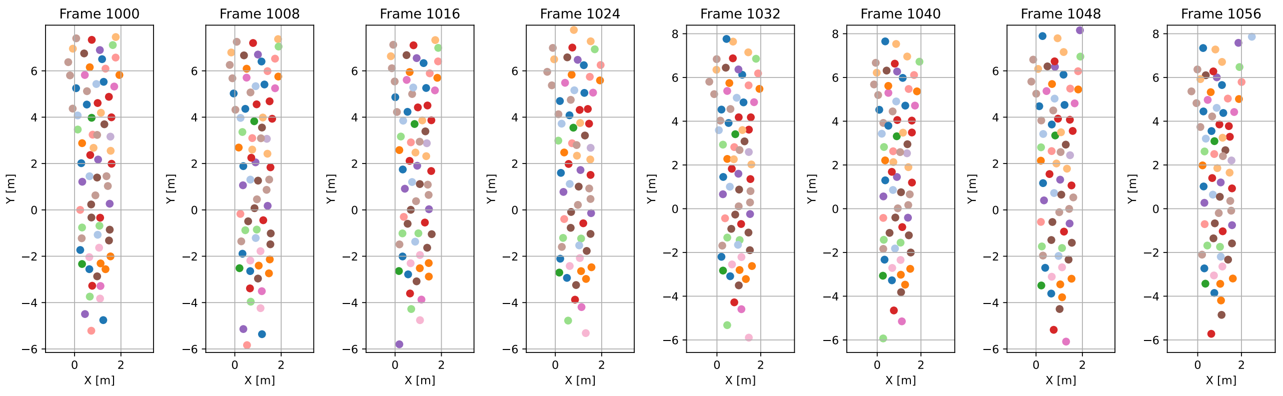
After transforming the data, we conclude a short exploration in order to learn more about the nature of the underlying dataset what we want to train our Neural Network on.

Movement of the pedestrians over consecutive frames

Figure 4 shows the evolution of the pedestrian positions over time for two different time periods during the experiment. One can see the influence of the bottleneck in slowing down the pedestrians.



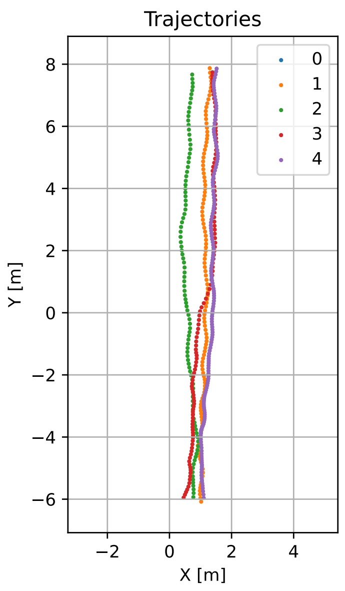
(a) Frame 300 - 356. Pedestrians are walking freely to the bottleneck.



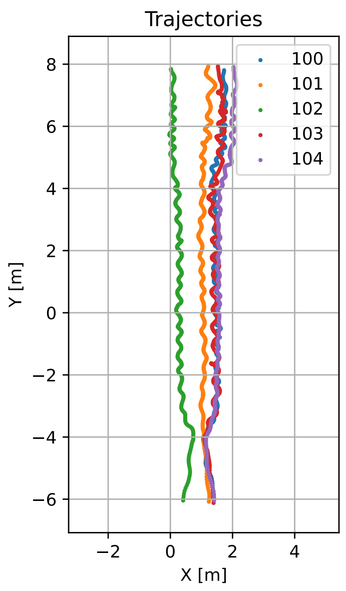
(b) Frame 1000 - 1056. Congestion around the bottleneck slows pedestrians down.

Figure 4: Movement of the pedestrians through the bottleneck over a period of 3.5 s each. Bottleneck with a width of $b_{Ab} = 0.7\ m$ is located at approx. $y = -4\ m$. Subfigure (a) shows the movement during the beginning phases of the experiment while (b) is set towards the end. Comparing both, one can see the effect of the congestion before the bottleneck slowing down the pedestrians.

The effect of the bottleneck on the movement of the pedestrians in the experiment can also be found by visualizing and comparing the trajectories of earlier and later pedestrians passing the bottleneck (see figure 5).



(a) Trajectories of IDs 0 - 4. Pedestrians are walking freely to the bottleneck at relatively high speed.



(b) Trajectories of IDs 100 - 104, slow zig-zag movement due to congestion around bottleneck

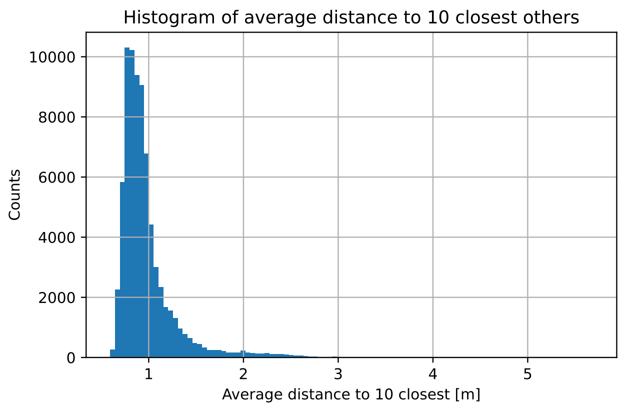
Figure 5: Trajectories of pedestrians through the bottleneck. Bottleneck with a width of $b_{Ab} = 0.7m$ is located at approx. $y = -4m$. Subfigure (a) shows the movement during the beginning phases of the experiment while (b) is set towards the end. Slowing down effect of bottleneck is visible as zig-zag and thicker trajectory line.

Analysis of features and target

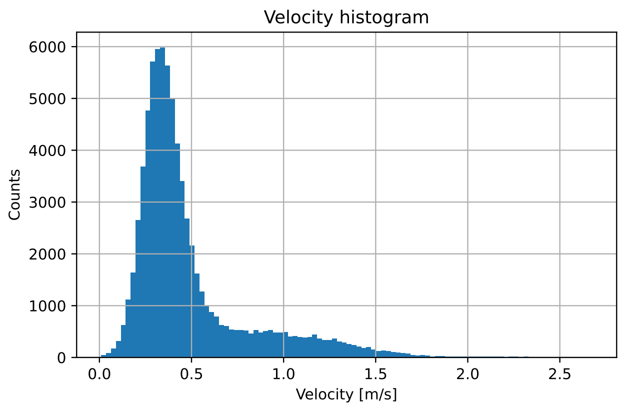
We want to analyse the input and output of our model, namely the relative spatial features, as well as the target velocity that is to be predicted by our model. Intuitively, one would assume that the closer the surrounding pedestrians, the slower the velocity. Note that the cause-effect relation does not necessarily need to go in one direction only since slow movements often lead to pushing and shoving, triggered by impatience due to the slow speed. Still, there seems to be a correlation between the two as shown in figure 6c which can potentially be learnt by our model.

Taking a closer look at the histogram of the target velocities v in figure 6b one can identify two parts of the distribution which add up to the total distribution. One gaussian like shape around $v = 0.3\ m/s$ representing the movement during the congestion and another one around $v = 1.0\ m/s$ which might indicate the typical walking speed.

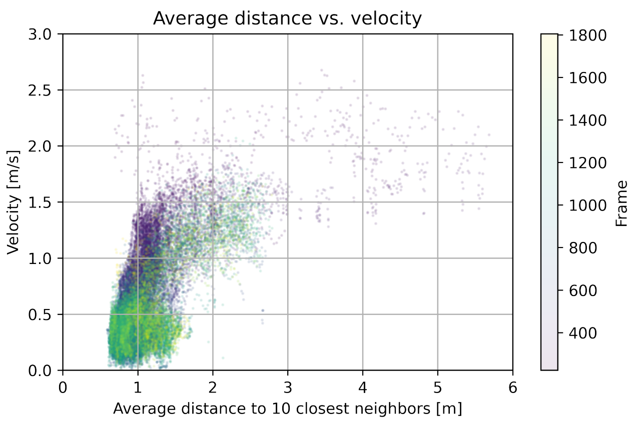
These two areas can also be seen in the scatterplot (figure 6c). Interestingly, if we set the average distance to the 10 nearest neighbors to a value of $\bar{s} = 1.3$ m, we can see that there is still a wide range of possible velocity values v. Here, the velocity values show a high correlation with the frame number within the experiment, with the congestion around the bottleneck in the middle of the experiment starting to build up slowly. This may indicate that it is not sufficient to consider only the relative spatial distances to the neighbors. We humans also consider how these distances to our surroundings change over time. If this change is constant, we could still be walking fast even though the distance to the person in front of us is not too great. This notion of situational awareness through speed is not considered in the Thordeux et. al. paper, but could be of interest in subsequent studies. For the remainder of our project, however, we restrict ourselves to relative distance-based features, as shown in equation 1.



(a) Histogram of average distance $\bar{s}$ to $K = 10$ closest pedestrians with a mean of $\mu_{\bar{s}} = 0.98\ m$.



(b) Histogram of average distance $\bar{s}$ to $K = 10$ closest pedestrians with a mean of $\mu_v = 0.48\ m/s$.



(c) Relation between the average distance $\bar{s}$ to the K-closest pedestrians and the target velocity v, colored by frame number.

Figure 6: Inspection of the features and targets. Subfigure (a) shows the histogram of average distance $\bar{s}$, subfigure (b) the histogram of the target velocity v, and (c) the relation between the two. All plots are based on the bottleneck experiment with a width of 70 cm.

Report on task 3/5: Implementation of the ANN

Notebook: `task_3.ipynb`
Code: `helpers.plots, helpers.models.utils`

Motivation

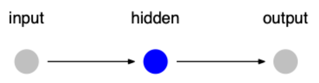
Since one of the main results of the paper [2] is that *Artificial Neural Networks* work better than classical approaches for the given datasets, these neural networks are implemented in order to reproduce the results. Tordeux et al. [2] claim that simple neural networks with a single layer and only a few number of neurons achieve the best results for the ring and the bottleneck dataset. Their approach reaches a *Mean Squared Error* of 0.03 for the simplest dataset and the best configuration. This task tries to reproduce the results by evaluating the given model configurations and settings while trying to beat the performances given by the paper. Experiments are carried out on the bottleneck dataset because the paper shows that it is the more complex one.

The neural networks are implemented with *Python*, *Tensorflow* and its framework *Keras* as it provides all necessary elements that are described in the paper.

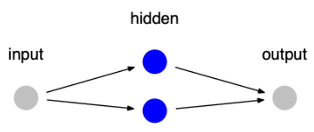
Model Definition

The paper by Tordeux et al. [2] suggests the use of a *feed forward neural network* where the information goes from some input nodes to the output nodes through some hidden layers. Furthermore, the network is supposed to minimise the *Mean Squared Error*. Apart from that there are no suggestions or restrictions given. That is why, standard methods and values for parameters are chosen in order to define and train the resulting models.

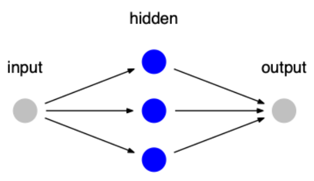
Because of the fact that different model configurations are to be evaluated, the definition of the model depends on the given configurations and on the number of nearest neighbors given in the dataset produced in Task 2. In the case of 10 nearest neighbors this results in 21 input nodes corresponding to the relative positions of the 10 nearest neighbors in addition to the average distance to these pedestrians (see equation 1). The hidden layers are defined according to the model configurations illustrated in Fig. 7. The number of parameters in a configurations corresponds to the number of hidden layers and each parameters states the number of hidden nodes in the respective layer. Exemplary, the configuration (10,4) has two hidden layers with 10 nodes in the first layer and 4 nodes in the second layer. It is obvious that all suggested model architectures are very simple having only one or two hidden layers and at most 12 hidden nodes per layer. In general, models with only a few of hidden nodes only can extract a few number of features but models with a high number of hidden nodes have the risk of overfitting the training data and, thus, do not generalise well for unseen data. Similar results are observed in the given paper even though the most complex neural network only has two hidden layers with 12 and 5 hidden nodes.



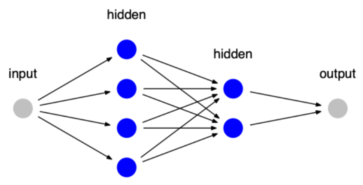
(a) Configuration (1)



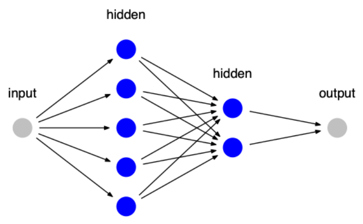
(b) Configuration (2)



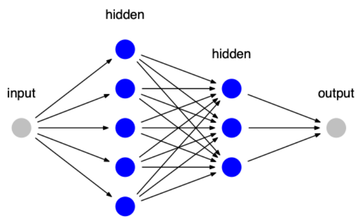
(c) Configuration (3)



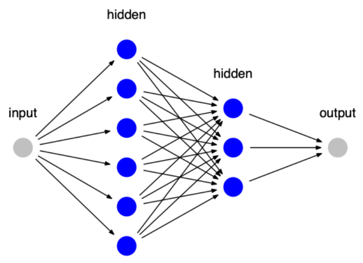
(d) Configuration (4,2)



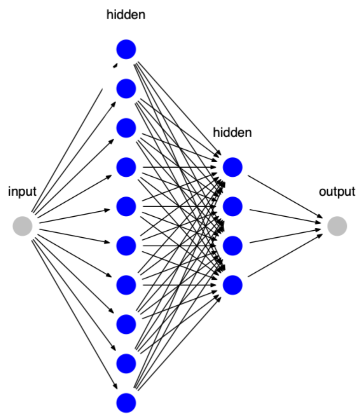
(e) Configuration (5,2)



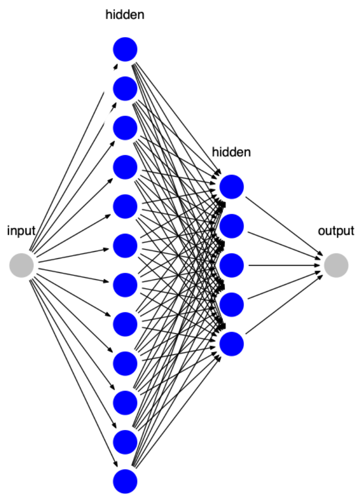
(f) Configuration (5,3)



(g) Configuration (6,3)



(h) Configuration (10,4)



(i) Configuration (12,5)

Figure 7: Model configurations: a single input node represents the 21 input nodes, the width and the depth of the neural network is variable with different numbers of hidden layers and hidden nodes, and a single output node that predicts the velocity of the pedestrian

The models are implemented by providing a single method that returns a defined and compiled model. Each model is initialized as `keras.Sequential` that allows stacking of multiple layers. The first layer is a `Input` layer that receives input of the shape $(2K + 1,)$ where K corresponds to the number of nearest neighbors for which the relative positions are calculated.

The hidden layers are added according to the given configurations. A model with the configuration (10,4) has two `Dense` layers with the given number of 10 and 4 hidden nodes as *units* input. These `Dense` layers have *sigmoid* as activation function. *Sigmoid* adds non-linearity to the network such that it is not restricted to the learning of linear functions. Furthermore, this activation function is easy to differentiate and is widely used. That is why, it is likely that it is also used in the paper even though they do not specify any activation functions. Additionally, because only neural networks with a small number of hidden layers are evaluated, the main disadvantage of *sigmoid* - the vanishing gradients problem - will not affect the networks' backpropagation.

While the width and depth, specified by the number of hidden layers and number of neurons, is variable, all networks have a single `Dense` layer as output layer. The activation function of this last layer is the identity function because it is a regression task that predicts the velocity of the pedestrian. By using *keras*, layers are automatically connected by the framework that already has implementations for the forward and backward passes.

The defined models are then compiled with the *Mean Squared Error* loss function as specified in the paper. *Adam* is chosen as optimizer because of its common usage for a variety of machine learning tasks where it achieves great out-of-the-box performance. All other parameters are set to the default values because the goal of this task is not to tune a single model but to find the best model out of the given configurations. That is why, we are not interested in hyperparameter tuning for parameters like the learning rate but fix the parameters in order to compare the performance of different models.

Training of all Model Configurations

In order to find the best model, all different model architectures are evaluated. Tordeux et al. [2] achieve the best results for a single hidden layer with three hidden nodes (Fig. 7c). Similar results are expected for our implementation.

According to the paper, different configurations are tested with a unusual combination of bootstrapping and cross validation without providing sufficient explanation how these methods are combined. Furthermore, it is not specified how the data is prepared for training. We have decided to use bootstrapping to sample data that is used to train 50 models for each configuration and, then, use cross validation for the training of the models. While bootstrapping is a good method if there is a huge number of data, it is not that useful in our case because we have already limited ourselves to a single dataset. However, we implement the same method because the implementation is reusable for a bigger dataset such that the bootstrapping process might be more useful.

The data is split into 80% for the learning process and 20% for the final testing of the performance of the best model. Like the paper specified for its bootstrapping process we sample 50 sets of 2000 samples each from the 80% of the learn dataset. Each set is then again split into two equally sized sets for training and validation of the models. The preparation of the data is visualised in Fig. 8.

For each configuration and set a separate model is fitted, resulting in a total number of 450 models to be trained. Because no batch size is mentioned in the source material, *stochastic gradient descent* is chosen with a batch size of 1 such that the gradient is updated after each training sample. This parameter has been chosen because it leads to the best results after having compared it to full-batch and mini-batch gradient descent. The maximum number of epochs is chosen to be 100 as it is sufficient for most model fittings. For each model, the final *Mean Squared Error* for the training and validation set is saved such that different model configurations can be compared.

Because it has already been apparent in the experiments of Tordeux et al. [2] that overfitting states a huge problem for more complex models, *early stopping* is implemented in this task in order to prevent that undesired behaviour. The patience is set to 10 such that most models terminate training after 50-80 epochs.

After the training of 450 models, the mean and the variance are calculated for each model configuration. The results are visualized in Fig. 9. Surprisingly, the results differ from the ones by the given paper. Instead of the configuration (3), best results are achieved by the most complex model architectures (10,4) and (12,5) where

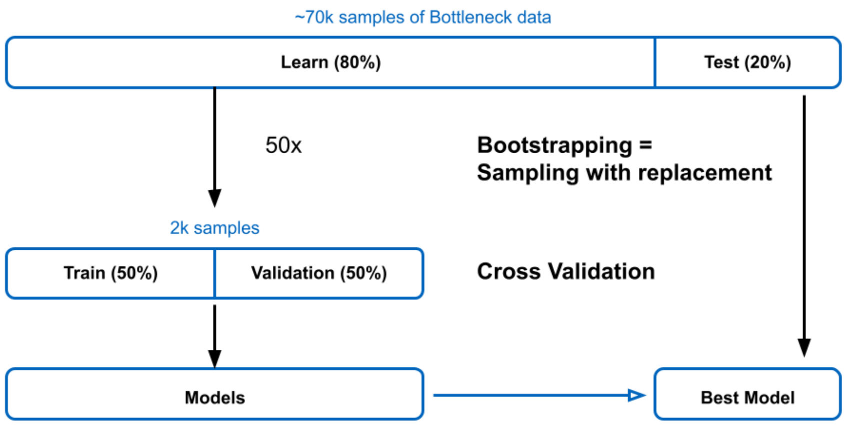


Figure 8: Preparation of the data consisting of 70.000 samples for the training, validation and final testing

the latter is marginally better. Reasons for these deviations could be the use of *early stopping*, other techniques that are not defined in the original paper or the usage of a different dataset. However, the paper claims that it also achieves the best results for a single layer with three nodes for any combination of datasets.

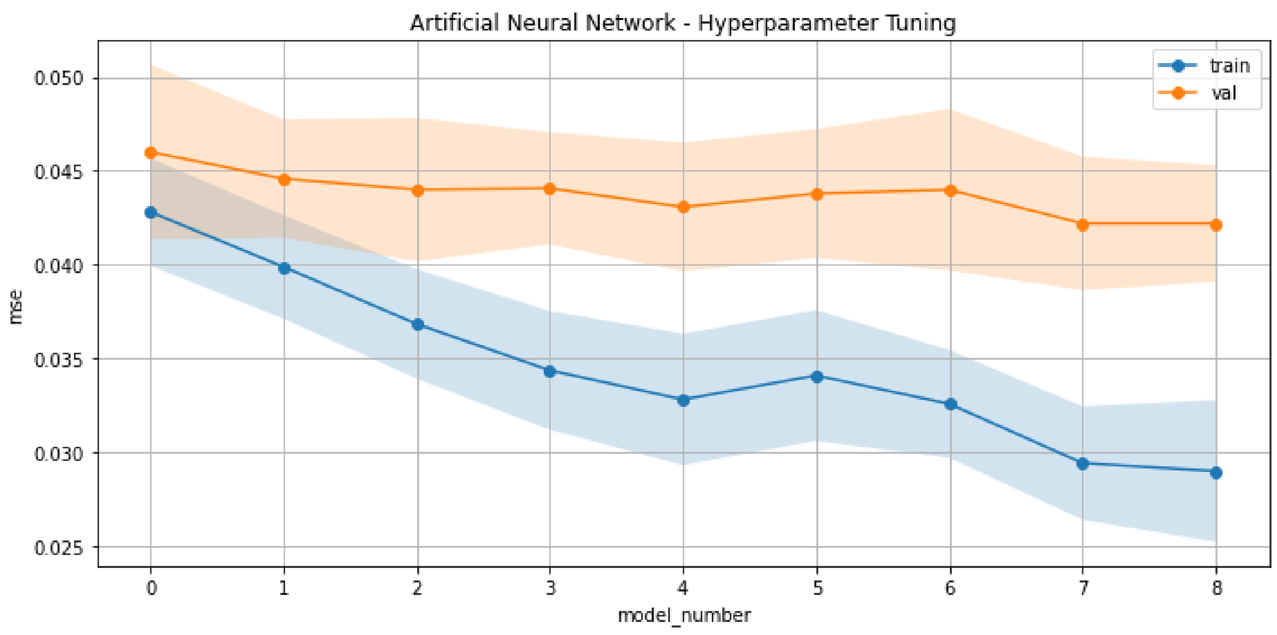


Figure 9: Mean (line) and variance (filled area) of the *Mean Squared Errors* of all models trained on the 50 bootstrapped sets for the training (blue) and validation (orange) of the model configurations in the order (1), (2), (3), (4,2), (5,2), (5,3), (6,3), (10,4), and (12,5): the minimal mean of errors is achieved by more complex configurations and is best for (12,5)

Final Training of the Best Model

After having found the best model configuration by using bootstrapping, it needs to be trained on the whole dataset for learning. The batch size of 1 and a standard validation split of 20% is chosen. The loss curve for the training and validation is illustrated in Fig. 10. After 50 epochs, this best model configuration results in a *Mean Squared Error* of 0.023 for the train set and 0.025 on the test set. This is a lot better than the results of the original paper that only achieved an error of around 0.03.

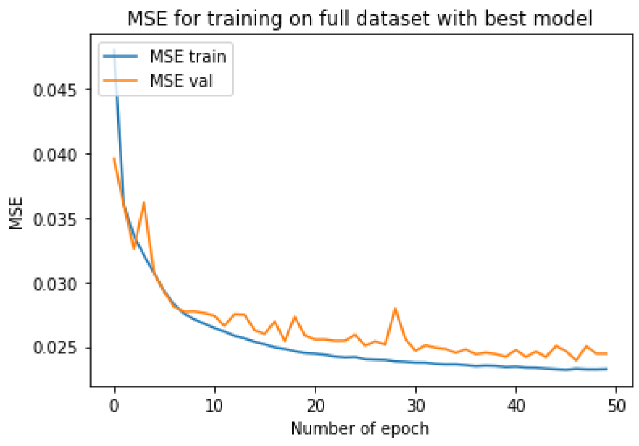


Figure 10: Loss curve of the *Mean Squared Error* of the training and validation on the whole bottleneck dataset with the best configuration (12,5)

Results

Even though the results for the bootstrapping of this implementation and the one done by Tordeux et al. differ, if a single model is fitted on the whole dataset with the configuration (3), similar values are achieved for the *Mean Squared Error* for train and test set. This indicates that the implementation is decent and not too far from the implementation of the original paper. However, the best configuration is no longer (3) but the most complex one for the given dataset. It needs to be evaluated if even more complex neural networks result in better predictions for the velocity but this is out of scope for this task.

Report on task 4/5: Counter-proposal: Decision Tree method

Notebook: `task_4.ipynb`
Code: `helpers.plots`, `helpers.models.utils`

Motivation

In recent decades, supervised learning methods have divided into non-neural and neural approaches. Typically, neural networks are considered standard methods for audio, image, and text tasks, while classical approaches such as decision tree-based algorithms are often used for tabular data, especially in typical industrial environments where criteria such as maintenance and explainability are of great importance. In a recent paper titled *Tabular Data: Deep Learning is Not All You Need* [1], Shwartz-Ziv and Armon show that ensemble-tree methods such as XGBoost[14] perform better than neural networks in predicting tabular data, while a combination of the two yields the best results. Table 3 compares decision tree based methods with neural networks with respect to the task domain of tabular data tasks.

Since Tordeux et al. only studied the performance of a neural network in velocity prediction, we will also train a random forest regressor to compare the final results and see which approach seems to work best.

Decision Tree Methods	**Neural Networks**
Pro Decision-Tree	
Require little data to get reasonably good results, and are less prone to overfitting.	Lots of data needed to properly train the network without overfitting due to generally high model capacity.
Well understood method and very easy to explain.	Black box that is difficult to explain.
Standard parameter settings achieve results close to the theoretical optimum.	Correct hyperparameter settings require a lot of trial & error as well as expertise.
No preprocessing required except for handling missing values, can work with different feature ranges since decisions are based on one feature at a time.	Must normalize all features to a similar range because of the fully connected layers and the vanishing and exploding gradient problems.
Relatively fast training results even without GPU.	Need access to GPU to train efficiently and in reasonable time.
Pro Neural Network	
Require hand-crafted expressive features and therefore require domain experts.	The first layers automatically learn an appropriate feature representation.
Approximation of feature input ranges in partially linear domains. No explicit representation of nonlinear relationships.	Nonlinearities allow to approximate reasonably well any continuous function in a sufficiently wide single-layer neural network (Universal Approximation Theorem).

Table 3: Comparison between decision tree methods and neural networks for tabular data
.

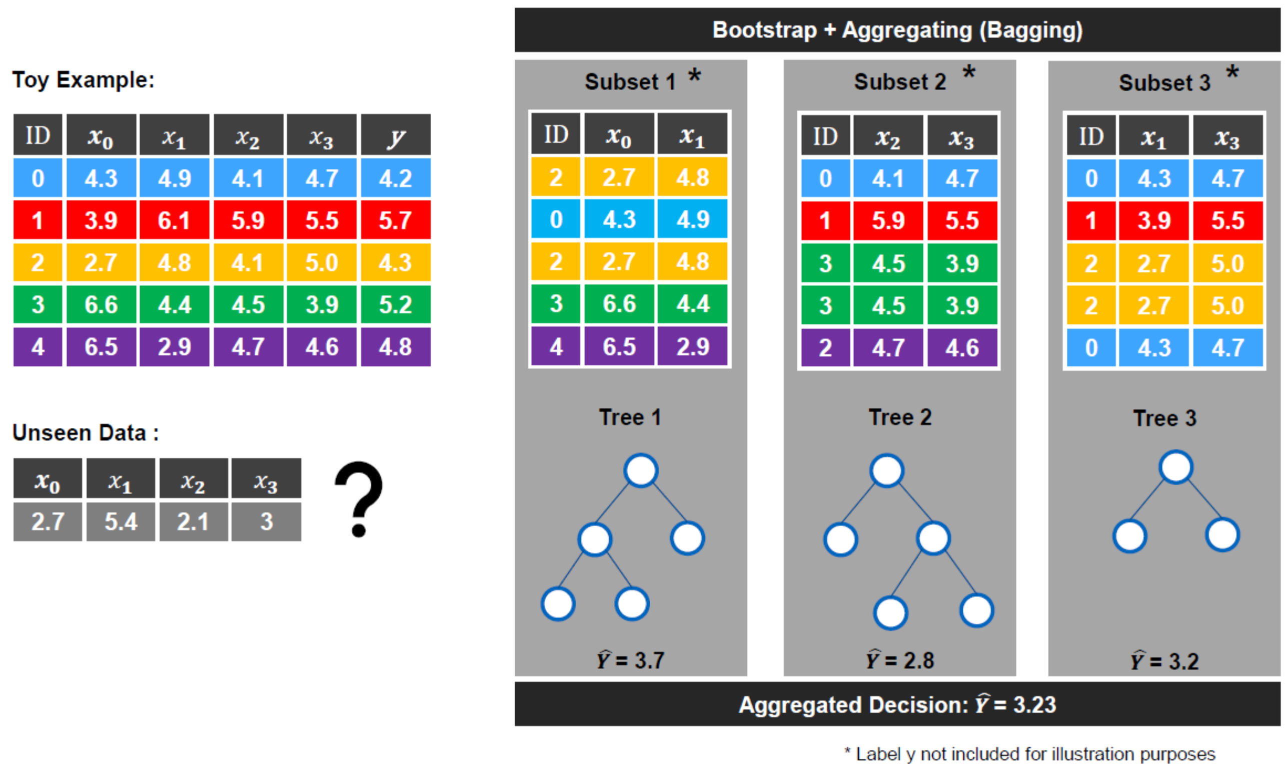
Theoretical introduction to Random Forest and ensemble methods

Random forest is an ensemble method of multiple decision trees that combines the predictions of individual weak learners into one final average vote. It uses bagging = (bootstrap aggregating) at both instance and feature level to create new datasets by sampling with replacement to then train a separate model on each subset. This

[14]`https://xgboost.readthedocs.io/en/stable/`

way, the total variance of the combined prediction is reduced and the joint model is less prone to overfitting and therefore achieves better results than a single decision tree. The additional bagging at feature level is needed since bagged decision trees can be highly correlated.

Figure 11 shows the process behind the random forest architecture. One can see how the bagging is performed, how multiple trees are learnt independently and how their individual votes are combined for the final prediction.


Bootstrap + Aggregating (Bagging)
Tree 1
Tree 2
Tree 3
$\hat{Y}$ = 3.7
$\hat{Y}$ = 2.8
$\hat{Y}$ = 3.2
Aggregated Decision: $\hat{Y}$ = 3.23
?

Toy Example:

ID	x_0	x_1	x_2	x_3	y
0	4.3	4.9	4.1	4.7	4.2
1	3.9	6.1	5.9	5.5	5.7
2	2.7	4.8	4.1	5.0	4.3
3	6.6	4.4	4.5	3.9	5.2
4	6.5	2.9	4.7	4.6	4.8

Unseen Data :

x_0	x_1	x_2	x_3
2.7	5.4	2.1	3

Subset 1 *

ID	x_0	x_1
2	2.7	4.8
0	4.3	4.9
2	2.7	4.8
3	6.6	4.4
4	6.5	2.9

Subset 2 *

ID	x_2	x_3
0	4.1	4.7
1	5.9	5.5
3	4.5	3.9
3	4.5	3.9
2	4.7	4.6

Subset 3 *

ID	x_1	x_3
0	4.3	4.7
1	3.9	5.5
2	2.7	5.0
2	2.7	5.0
0	4.3	4.7

* Label y not included for illustration purposes

Figure 11: Schematic drawing of the random forest architecture. Multiple individual decision trees are learnt on bootstrapped training data at instance and feature level. During inference time the final prediction is obtained through averaging the individual predictions of each tree therefore reducing the variance at a slight cost of greater bias. Note, that due to illustration purposes, the target column is not shown in the data subsets, although the individual training process still require it.

Random Forest applied to the bottleneck prediction task

In our work, we use the `RandomForestRegressor` implementation from scikit-learn in order to train the model and predict velocity values on the test set. As for the implementation of the NN model, we make use of the same $2K + 1$ features containing spatial distances to all K closest pedestrians in X- and Y- direction, as well as the average distance. Here again, the number of closest other pedestrians to consider is also set to $K = 10$:

$$v = \mathrm{RF}(\bar{s}_K, (x_i - x, y_i - y, 1 \leq i \leq K)) \tag{3}$$

Hyperparameter tuning

Although random forest models are very robust and the default set of hyperparameters often already achieves great results, we want to tune the minimum number of samples per leaf node at which the tree stops any further splitting.

As for our hyperparameter tuning of the Neural Network, we stick to the process of the paper [2], bootstrap 50 subsamples of 1000 datapoints each, use half of the datapoints for training and the other half for validation (see figure 8). This process is repeated for all tested hyperparameter configurations and one finally chooses the setting that shows the best validation loss. Apart from the tuned parameter `min_samples_leaf`, all other parameters are kept at their default values. Figure 3 shows the performance for each tested configuration. One can see that the training loss increases for growing numbers of minimum leaf samples while the validation loss shows an optimum at `min_samples_leaf` = 1. Still, the random forest model seems to be very robust since the validation errors are not different by a lot. Initially, one might think that a value of `min_samples_leaf` = 1 would lead to overfitting. However, the averaging effect of the ensemble of 100 trees seems to prevent for it. Due to the robustness of random forest and its great out-of-the-box performance, we decided not to use any other hyperparameter-tuning but to work with the default parameter values.

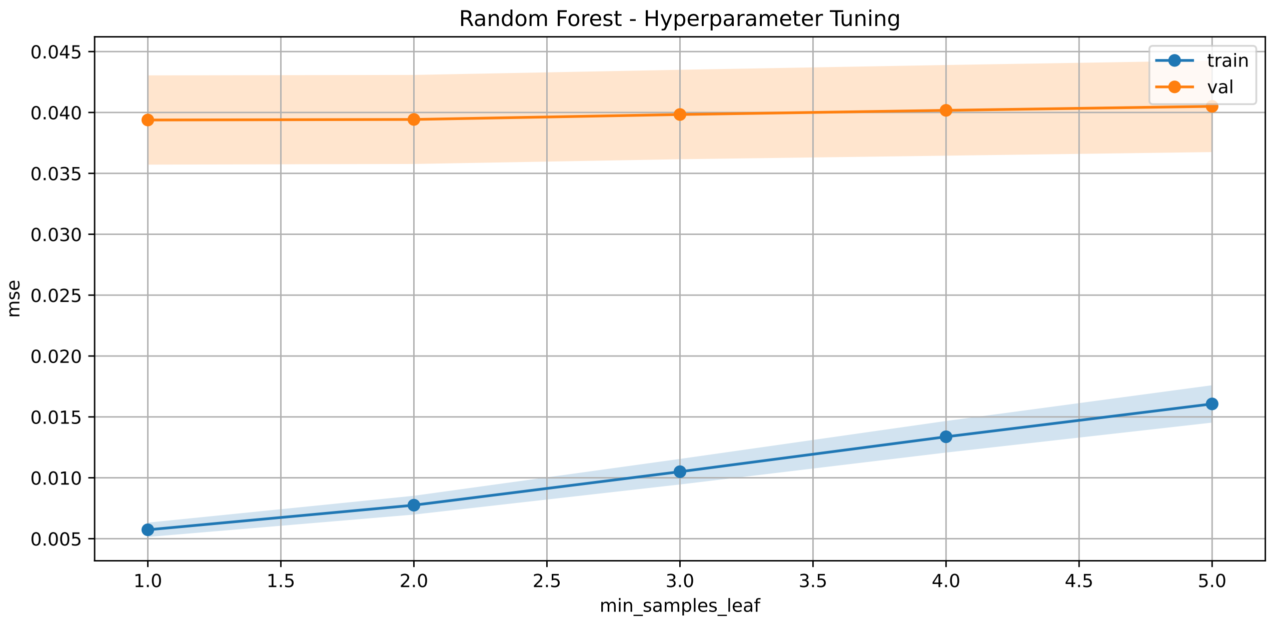


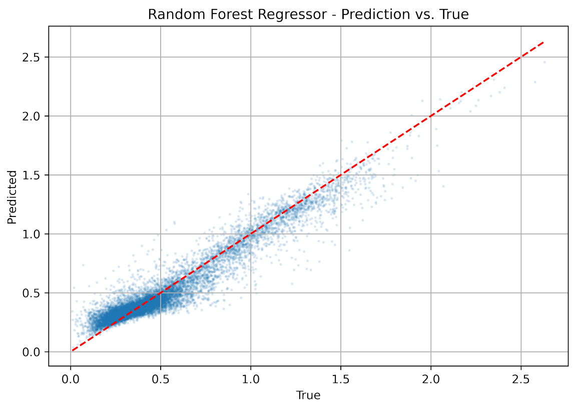
Figure 12: Training and validation errors for different number of minimum samples per leaf node. The line shows the mean over all 50 bootstrap sets while the bands describe the standard deviation. The validation error shows the optimal number of minimal leaf samples to be 1.

Final retraining and results on the test split

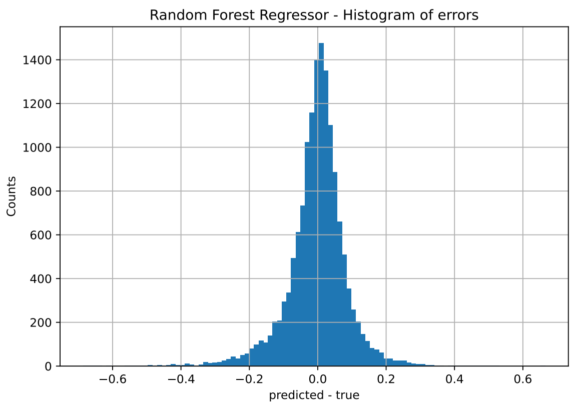
We retrain the final model configuration on the full learning data. Table 4 shows the final model configuration and the achieved scores during testing on the separate test split. Note, that the test split serves as a proxy for real unseen data to evaluate our models final ability to generalize and should not be used to tune any hyperparameters which is why we are only using it once at the very end.

`n_estimators`	`100`
`criterion`	`"squared_error"`
`max_depth`	`None`
`min_samples_split`	`2`
`min_samples_leaf`	`1`
`MSE_train`	`0.00109`
`R2_train`	`0.989`
`MSE_test`	`0.00729`
`R2_test`	`0.924`

Table 4: Final hyperparameter settings and achieved scores after final retraining on the full learning data and evaluating on the full test set. All parameters are set to their default values. For a complete list of parameters, see the scikit-learn implementation of the `RandomForestRegressor` predictor.



(a) Comparison of ground truth velocity values and the predictions of our final random forest model. Perfect prediction is shown as the diagonal line and serves as a reference.



(b) Histogram of residuals showing a Gaussian-like shape. One can see that most of the predictions are relatively good ($< 0.1m/s$) while there still are some outliers towards the tails of the histogram.

Figure 13: Final results of the random forest regressor on the test data after retraining on the full learning data.

Report on task 5/5: Discussion and Outlook

Notebook: `None`
Code: `None`

Motivation

The following chapter is intended to briefly discuss the challenges and deviations in data preparation and results that have arisen during the attempt to reproduce the results of the paper. In addition, this chapter is supposed to draw attention to points that lack of clarity where more detailed consultation with the authors is required. A brief outlook will then be given, indicating possible starting points for future work.

Paper: Insufficient Setup Description

Framework Selection: Already during the preliminary discussion on the implementation of the feed-forward network according to the specifications of the paper, it was noticeable that basic preprocessing and work steps were not treated in sufficient detail. Also, at no point in the paper it is explained why the authors chose to implement the simple feed-forward network in R using the `neuralnet` package. From the perspective of inexperienced programmers, the package in R appeared inferior to PyTorch and Tensorflow, making it difficult to reproduce and verify the results. The following section will therefore highlight the instructions and procedures that were particularly difficult to interpret.

Data Composition: The first challenge was to understand how the dataset is used to fit the ANN. For this we have identified the following points in the corresponding section of the paper (fitting the neural network) as a possible cause of obscurity. The authors speak only of "data" in this context, probably referring to the equally named section in their paper. In this data section, the authors present the two datasets available on the internet (Ring dataset and Bottleneck dataset). The authors claim to use a combination of the two datasets for the purpose of the calibration process. However, they leave it open how exactly the two datasets are combined here. The reader must once again assume that they mean an equally weighted composition, which in their nomenclature can presumably be expressed by R+B / R+B (R=Ring Dataset / B=Bottleneck Dataset) for train and test set. In addition, they state that the composition would not play a role, so that, if necessary, only the individual dataset could be used for this purpose. Also, unfortunately, they do not offer the possibility to verify this with data. Here, the authors simply assume that one have to believe them without providing any proof.

Cross Validation and Bootstrapping: Another questions is how the combination of bootstrapping and cross validation is carried out. In detail, it remains unresolved how many samples a bootstrapped sub-dataset consists of. Also the attached sources [17, 14] (see paper reference) can not provide more useful information. Therefore, this section could have been better used by focusing more on the differentiation of the terms sample and subsample. With the information available for the cross validation procedure, we have decided to implement pre-processing as described in task 3 (see 2).

Model and Training Implementation: The same problem of insufficient description of the setup is not only an issue in the data pre-processing and fitting of the network, but also in the general model structure and training description. For example, the authors do not go into detail about whether and which activation functions were used. Also for the training of the ANN no exact statements about the implementation are made. Among other things, it remains open which optimizer was used and on which data the gradient is estimated (no information about batch sizes given). Furthermore, it remains unanswered for how many epochs the different architectures were analyzed. As already described in task 3 (see 2), overfitting seems to be a real problem, so it is all the more surprising that the authors do not mention any techniques here. It is unclear why the authors do not consider even more extensive networks (increased number of layers and/or nodes).

Paper / Final Project: Implementation Deviations

Due to the lack of information on the majority of the setup mentioned in the previous section, we rely on our own empirical values during implementation. This is why, we use `Python` and the `Tensorflow Keras framework` instead of the `neuralnet` package from `R`. This also seems reasonable, since many scientists are probably more familiar with it and it would therefore be easier to reproduce our results. As already described in detail in chapter 3, own methods are implemented based on the `Tensorflow` framework for the training. The most important difference is that we evaluate the gradient using stochastic gradient descent. This approach seems advantageous to us because it reduces the probability of getting stuck in a bad local optimum. Also due to the lack of information on whether or which regularisation techniques are used, we could only make assumptions about how we could reproduce the authors' results as closely as possible. One of the most common techniques for this would be to use a norm-based penalty (L2 or L1-norm). However, since we could not sufficiently assess the impact of the respective norm on the application case, we have decided against using this technique. Specifically, it is possible that we could not assess how the sparsity-promoting L1 norm or the outlier-sensitive L2 norm would develop on this dataset. Since overfitting seems to be a big problem in the learning process, we use early stopping. Specifically, early stopping with a patience value of 10 is used here. Unfortunately, no statement is made about the duration of the training procedure. For reasons of time and resources, we have therefore set the training to a maximum of $n = 100$ epochs. This value comes from empirical experience in other projects and should therefore also be examined quantitatively in future work. Nevertheless, this limitation can be justified by the fact that early stopping ends the training process at around $n = 80$ epochs.

Paper / Final Project: Result Deviations

The first part of this section is intended to compare the ANN results to the paper and the second part should compare our ANN results to our random forest approach as described in section 2.

Figure 14 is taken from the paper (corresponds to figure four) and shows the course of the mean squared error over different compositions of the training and test dataset through a combination of the bottleneck and ring dataset. As expected, it can be seen that the best values are obtained for the individual datasets.

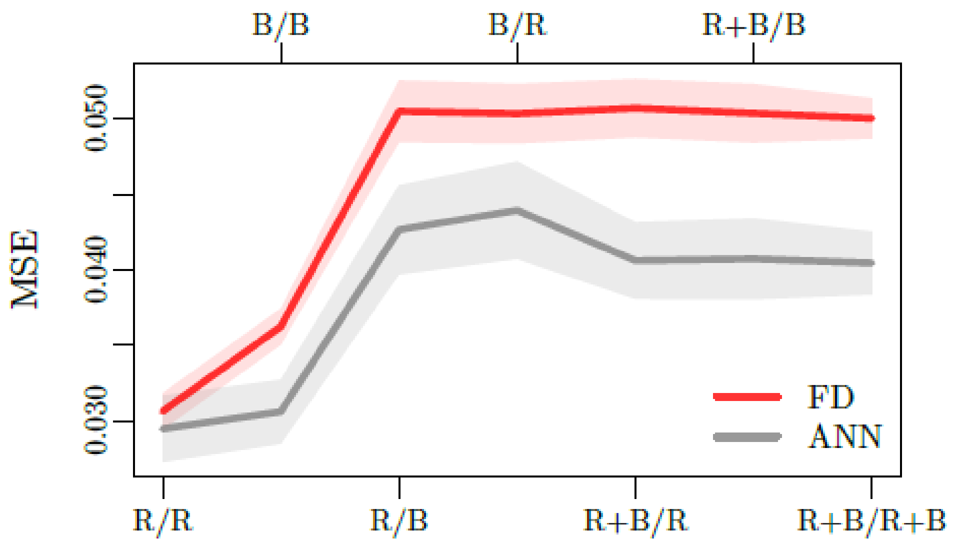


Figure 14: course of the mean squared error over different compositions of the training and test dataset through a combination of the bottleneck and ring dataset. This corresponds to Fig. 4 in the discussed paper [2]. The red line indicates the course of the Weidmann fundamental diagram and corresponds to the benchmark of the paper.

The authors arrive at an MSE of 0.029 for the ring dataset and an error of 0.031 for the bottleneck dataset. This comparison once again confirms the theory we put forward in section 2 that the bottleneck dataset is the more difficult of the two and thus legitimises our approach of concentrating on it initially.

Evaluation Data Composition: However, the authors not only evaluate different compositions of the dataset, but also nine different architectures of the feed forward network. The corresponding evaluation is visible in figure 15. Here the authors come to the conclusion that the architecture with one hidden layer and three nodes has the lowest error. It is striking that this error is about 0.042. Compared to the lowest value in 14 with 0.031, this value is about 35% higher. From this one can conclude that the evaluation of the architectures was probably carried out on a combination of the two datasets, since a plateau seems to result at about 0.042. Nevertheless, no exact information is provided, which is why we have decided to compare our result with the smallest error of 0.031 on the single bottleneck dataset.

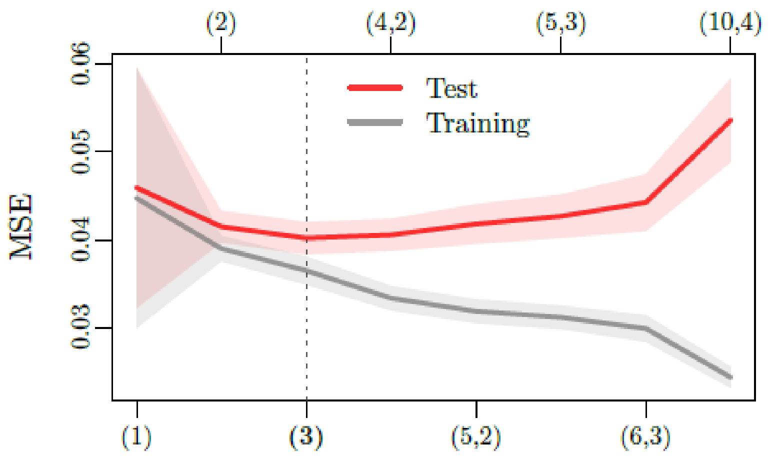


Figure 15: course of the mean squared error over different architectures. The evaluated dataset composition is not described in the paper. The red line indicates the evaluation on the test set. The grey line corresponds to the training set. The tuple with the numeric values correspond to the number of nodes and layers used in the ANN [2]

Evaluation Best ANN Architecture: Comparing the paper with our implementation shown in task 3 (see 2), we cannot reproduce their results regarding the best architecture. In our approach, we assume that "deeper" networks, so networks with more hidden layers and a larger number of nodes, lead to a lower error. For example, using the early stopping described in the previous section, the best architecture resulted in (12,5) instead of (3). By simply adding the early stopping method, our implementation of the (12,5) configuration was able to achieve a mean squared error of 0.025 on the test set. This is about 24% better than the authors' result on the single bottleneck dataset with an MSE of 0.031 on the test set.

Verification Datadriven Approach: Although the comparability of the results is almost impossible to legitimise scientifically due to the large number of missing setups and training parameters, this could be a first indication that the architectures selected by the authors were all clearly too simple. The low level of engagement with the results and also the simple architectures suggest that the authors only wanted to stimulate the scientific community to become aware of the potential of their approach and that the actual results were not very important to them. So it turns out that we can support and verify the basic thesis of the authors, that in comparison to the classical approach of the fundamental diagram according to Weidmann, really better results could be achieved. Our implementation even outperformed the Weidmann model by approximately 40% on the bottleneck dataset. In comparison to the improvements achieved by the authors (20%) this is a doubling of the enhancement.

As in the introduction to this section, we will now compare the results of our ANN approach with those of our random forest approach.

Comparison Random Forest: In figure 16 you can see the residual histograms for the random forest approach and the ANN approach. One can see that both courses are resembling a Gaussian distribution. From this one can conclude that a large number of the predictions are contained in a small error interval and that there are only a few outliers upwards and downwards. In practical terms, this means that for most predictions of pedestrian speed there is only a small deviation of less than 0.1 metres per second. It is noticeable in the direct comparison of the two approaches that the random forest approach (orange histogram) delivers better results. This can be seen from the more tapered course around the mean of zero. In practice, this means that most of the predictions of our ANN approaches contain an (however small) error. However, it appears that for the majority of errors, the interval results between $I \in [-0.1, 0.1]$, indicating that both approaches could be used in practice.

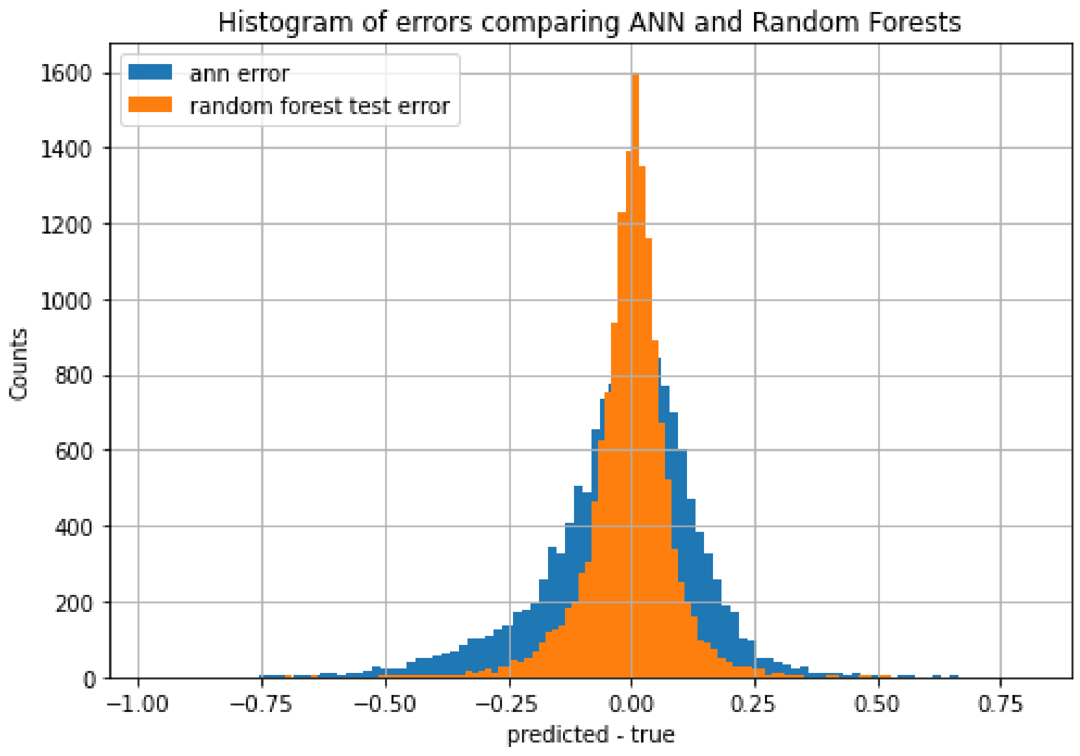


Figure 16: Two histograms of residuals. Both histograms resemble a Gaussian distribution but with different variances. The orange plot shows the corresponding histogram of the random forest approach and the blue histrogram visualises the distribution of the residuals for the ANN approach. One can see that both approaches achieve a small deviations in absolute values. However, based on the more peaked course of the orange histogram, one can see that the random forest approach yields slightly better then the broader ANN histogram.

If we also compare the absolute values of the mean squared error for the test set of the two approaches, it becomes clear that the random forest approach delivers better results than the ANN approach. Specifically, as described in more detail in section 2, an error of 0.00728 resulted here, while the ANN came to 0.025. The error is therefore greater by a factor of about 3.5 and can probably be attributed to the strength of the random forest approach, which often finds good features through bagging to find a representation of the data. Compared to the paper with an MSE of 0.031 on the Bottleneck test set, this is already an improvement by a factor of about 4.25. In addition, the great advantage of this approach is that there is no need for a time-consuming search for architectures and hyperparameters. Compared to the Weidmann benchmark on the Bottleneck dataset with MSE 0.035, the improvement is even a factor of 4.8 and thus shows the practicality.

Result Overview

The following subsection is intended to present a brief overview of our results. The evaluation was carried out for both training and testing on the individual bottleneck dataset. The results were rounded to the third decimal place for better presentation.

MSE \Approach	**Paper ANN**	**Our ANN**	**Random Forest**
Train Error	0.042	0.023	0.001
Test Error	0.031	0.025	0.007
Dataset Composition	B / B	B / B	B / B

Table 5: Compact overview of the mean squared error on the bottleneck dataset for both the training and the test set for the values of the paper, our reproduction of the ANN approach and the random forest approach.

Conclusion and Outlook

As already described in the motivation, the authors' fundamental idea is to reflect on how to substitute and facilitate the selection and the associated finding of the physical parameters of microscopic pedestrian models. We were also able to verify with our results that an ANN is a possible machine learning technique to predict the speed of pedestrians, especially for different geometries. We therefore agree with the authors' opinion that this field of research offers potential. In particular, we are convinced that the full potential of the ANN approach has not been exploited in this work, since, for example, only very simple architectures are used and not sufficient measures against overfitting were conducted. It would therefore be interesting to evaluate in future work whether the combination of more complex architectures with regularisation can offer a further boost in performance compared to the aggregated model based on fundamental diagrams. Nevertheless, our experiments with the random forest approach have also shown that there are alternatives to this which can already deliver competitive results with out-of-the-box parameter settings. If this result can also be verified on more complex datasets, we would have the advantage that the model fitting procedure is somewhat less complex and less time-consuming than the trial-and-error ANN approach. Our general recommendation for future work would therefore be to consider both techniques for pedestrian speed prediction and to investigate more into a thorough hyperparameter tuning process of ANNs.

References

[1] Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need.

[2] Antoine Tordeux, Mohcine Chraibi, Armin Seyfried, and Andreas Schadschneider. Prediction of pedestrian speed with artificial neural networks.