Report for Final Project from group I

Tasks addressed: 5
Authors: Kejia Gao (03779844)
Jingyi Zhang (03785924)
Maximilian Mayr (03738789)
Yizhi Liu (03779947)
Felipe Antonio Diaz Laverde (03766289)
Last compiled: 2024–07–15

The work on tasks was divided in the following way:

Kejia Gao (03779844)	Task 1	22%
	Task 2	22%
	Task 3	22%
	Task 4	22%
	Task 5	22%
Jingyi Zhang (03785924)	Task 1	19.5%
	Task 2	19.5%
	Task 3	19.5%
	Task 4	19.5%
	Task 5	19.5%
Maximilian Mayr (03738789)	Task 1	19.5%
	Task 2	19.5%
	Task 3	19.5%
	Task 4	19.5%
	Task 5	19.5%
Yizhi Liu (03779947)	Task 1	19.5%
	Task 2	19.5%
	Task 3	19.5%
	Task 4	19.5%
	Task 5	19.5%
Felipe Antonio Diaz Laverde (03766289)	Task 1	19.5%
	Task 2	19.5%
	Task 3	19.5%
	Task 4	19.5%
	Task 5	19.5%

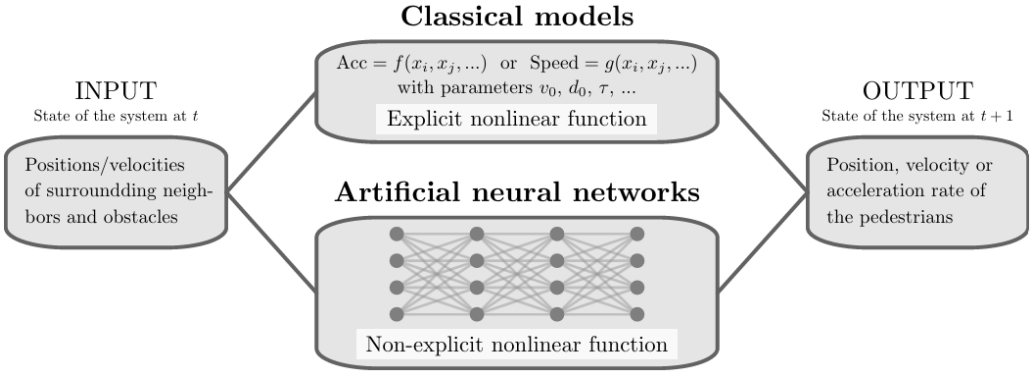

Figure 1: Conceptual usage of the models. Source: [2]

1 TASK 1

Report on task TASK 1, Problem description

In the prediction of crowd dynamics, microscopic models are commonly used. These microscopic models depend on very few parameters linked to physical properties. The fundamental diagram (FD), which describes the relation between the distance to obstacles and neighbors, and the speed, can be used to model the pedestrians' speeds. Although microscopic models contain few parameters and are simple, they can model realistic pedestrian flows fairly well. However, in complex structures like stations the model's predictions are still lacking as real pedestrians tend to adapt their speed and behavior based on the spatial structure. [2]

1.1 Background of the research

Tordeux et al. tackled this problem by using artificial neural networks (ANN) [2]. They compared the classical approach using a FD-model (a model based on a fundamental diagram) and the usage of an ANN on two different data sets. Their results show that using ANNs improves the prediction accuracy of the pedestrians' speeds compared to those computed by the FD-model. Figure 1 portrays the usage of the models.

In the following, their work is described in more detail. Section 1.1.1 gives a more detailed insight in the data sets used in the experiments, section 1.1.2 describes the two methods to determine speeds Tordeux et al. compared, and section 1.1.3 shows their results more precisely.

1.1.1 Data sets

Tordeux et al. used data from experiments conducted in laboratory conditions during the HERMES project.[1] The used data resulted from two experiments. In the first experiment, participants were supposed to walk in a circle, and in the second experiment they had to pass a bottleneck. During both experiments, all pedestrians were walking in the same direction. Figure 2 display the setup of the first experiment and Figure 3 the setup of the second. The data of these two experiments will be referred to as ring and bottleneck data.

The data consists of the positions of each participant over a course of several time steps, with the sampling frequency of $16Hz$. From this data it can be observed that the speeds tend to be higher in the bottleneck scenario than in the ring scenario for any given mean distance to the closest neighbors. This tendency is shown in Figure 4. [1, 2]

1.1.2 Methods

The fundamental diagram used in the FD-model is the Weidmann model. It models the speed by a non-linear function depending on the mean space to the neighbors. The velocity is computed according to Equation 1

$$v = v_0(1 - e^{\frac{l-\bar{s}_K}{v_0 T}}) \tag{1}$$

[1]The data used by Tordeux et al. can be found at `https://zenodo.org/records/1054017`.

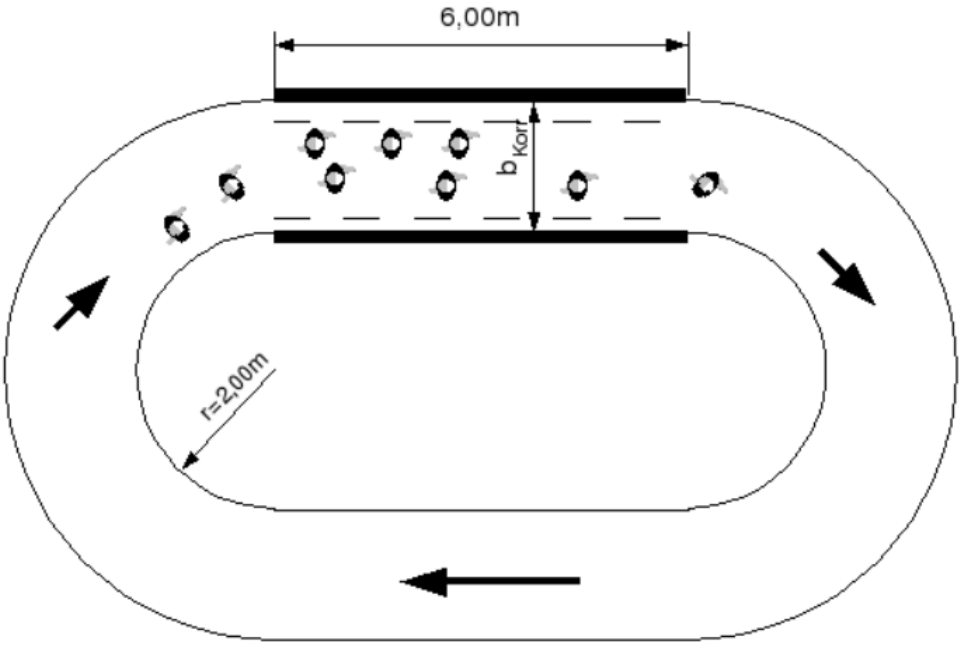

Figure 2: The setup of the ring experiment. Source: [1]

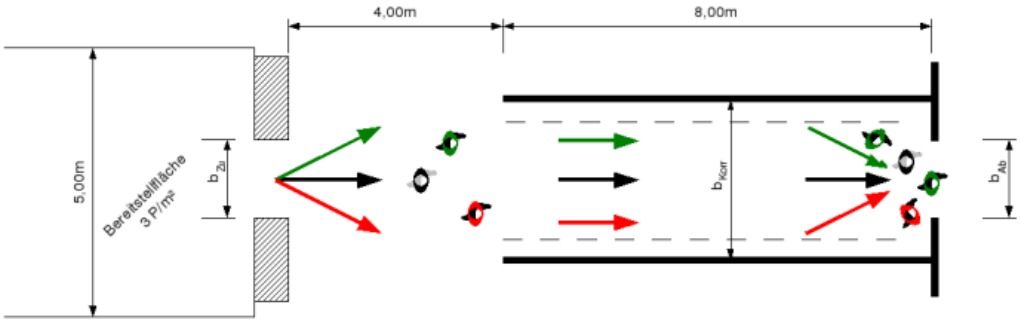

Figure 3: The setup of the bottleneck experiment. Source: [1]

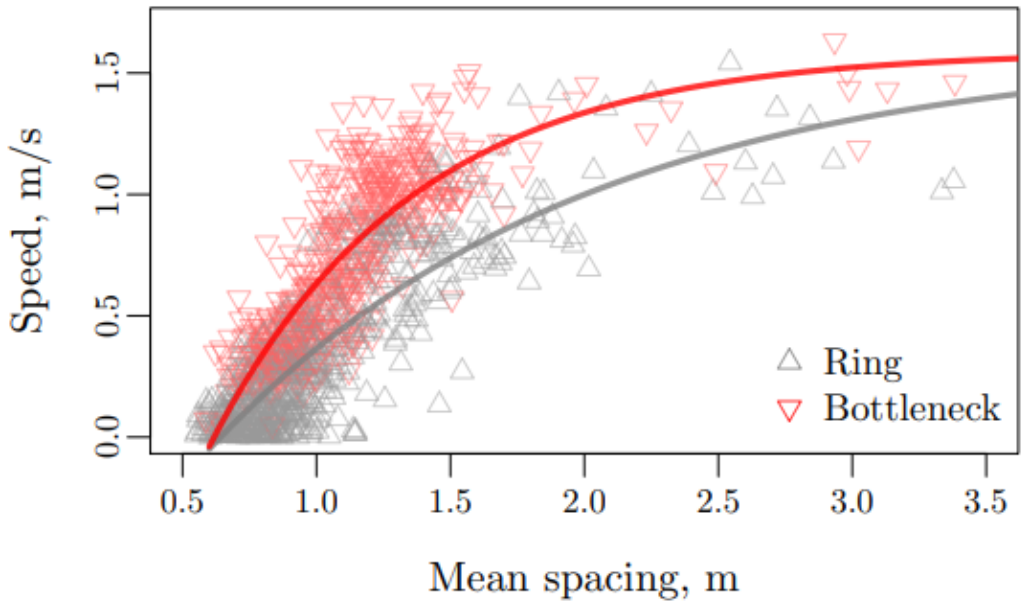

Figure 4: Speeds in the ring and bottleneck scenario. Source: [2]

where $\bar{s}_K = \frac{1}{K}\sum_{i=1}^{K}\sqrt{(x-x_i)^2+(y-y_i)^2}$ is the average distance between a pedestrian's position (x, y) and the positions (x_i, y_i) of its $K = 10$ closest neighbors, and the time gap T, pedestrian size l and desired speed v_0 are model parameters.

This FD-model was compared to a feed-forward ANN with hidden layers $H = (l, n)$ (l hidden layers with n neurons in each). The speed computation can be formalized by Equation 2.

$$v = NN(H, \bar{s}_K, (x_1 - x, y_1 - y), ..., (x_K - x, y_K - y)) \tag{2}$$

The network's inputs are the average distances and the relative positions of all K neighbors considered for the mean spacing. This leads to a total of $2K + 1$ input values. The number of weights depends on H.

The model was trained on 50 sub-samples of the data using cross-validation and bootstrapping. Moreover, a 50%/50% split of the datasets into training and test sets is performed. Training sets originating from different datasets (8 ring datasets and 4 bottleneck datasets) are concatenated without test sets and test sets are concatenated excluding training sets, so that data leakage in the evaluation will not happen. The models' goal is to minimize the mean squared error between the predicted speeds and its actual counterparts. Tordeux et al. have discovered that a minimum testing error (before overfitting increases) is achieved for $H = (1, 3)$ (or simply $H = 3$), when using a single hidden layer with 3 neurons. A comparison of different values for H can be seen in Figure 5. [2]

1.1.3 Results

Tordeux et al. compared the FD-model with the ANN with $H = 3$. Experiments were conducted on different combinations of the bottleneck data (denoted as B), the ring data (R), and combined data (R+B). The results show that the network outperforms the FD-model in all cases. It is slightly (about 5%) better when trained and tested on the ring data, and significantly (approximately 15%) better when trained and tested on the bottleneck data or when trained on either of the two data sets and tested on the unobserved data. In the case when the network was trained with the combined data, it even improved over the classical model by 20% for all three possible test data sets. These results are shown in Figure 6.

Furthermore, it was shown that the ANN correctly identifies that there are higher speeds in the bottleneck scenario, although the difference between the velocities of the two scenarios is smaller than in the data. Figure 7 shows these results. [2]

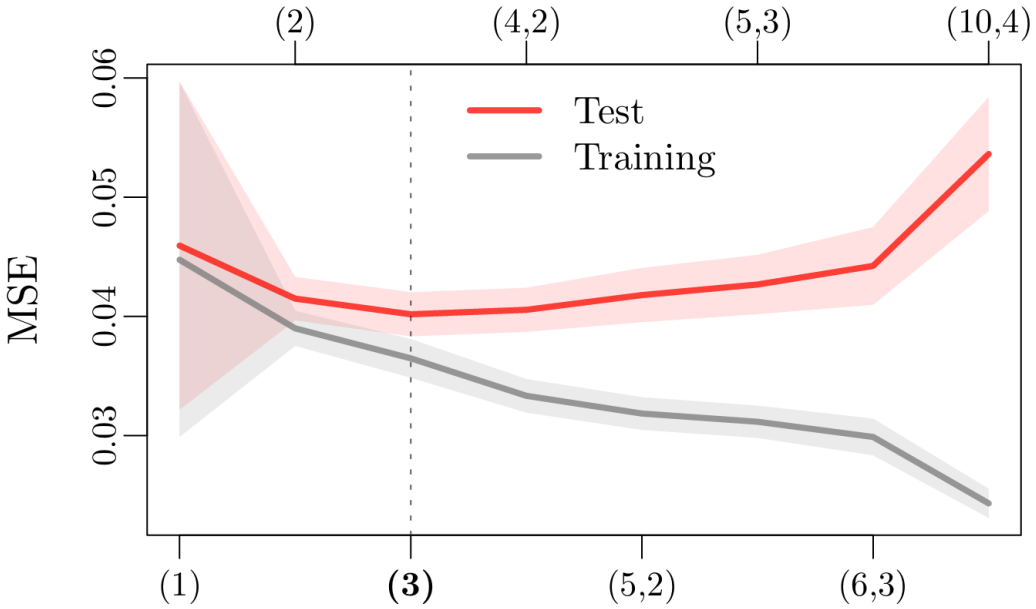

Figure 5: Training and testing errors for different values of H. Source: [2]

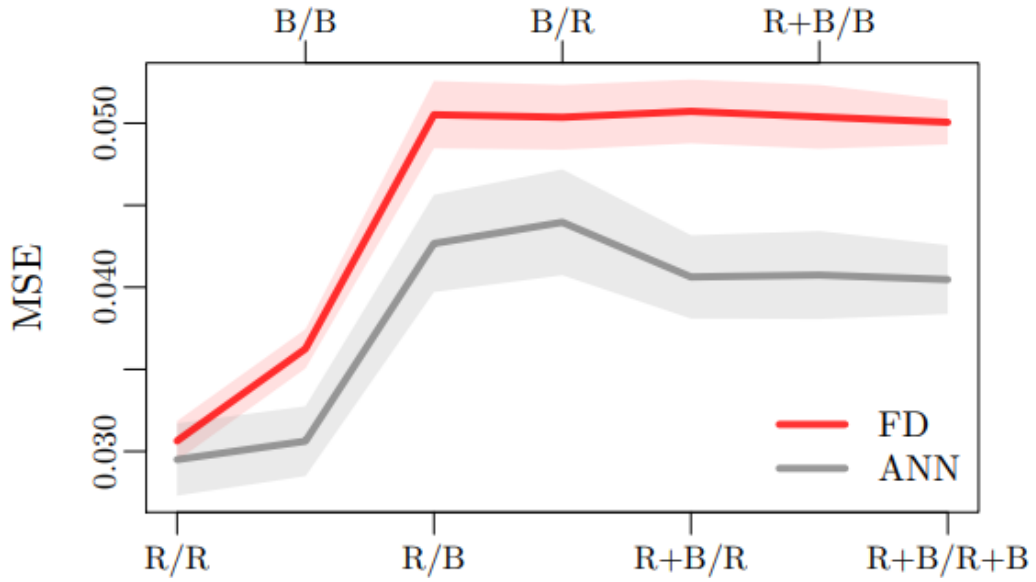

Figure 6: Testing errors of the FD-model and ANN. x/y denotes that the training happened with data x while the testing happened with data y. Source: [2]

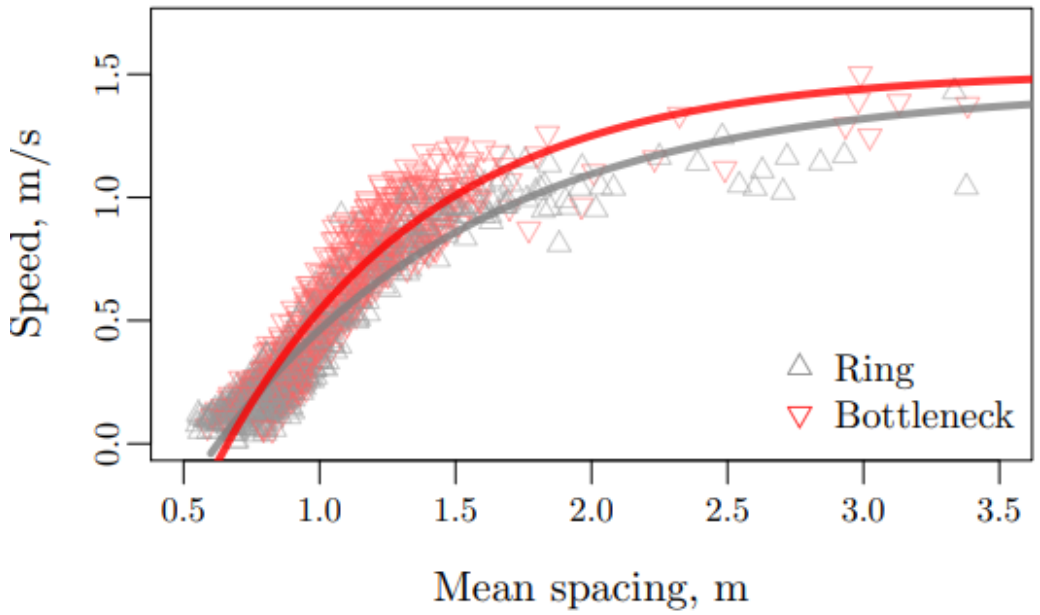

Figure 7: Speeds in the ring and bottleneck scenario predicted by the ANN (trained and tested on the combined data). Source: [2]

1.2 Aim of this final project

This final project is a literature review of the paper by Tordeux et al. [2]. We tried to reproduce their results by implementing similar models and conducting similar experiments. In the following sections, first the data preprocessing is described, followed by the implementations of both models, and finally, our results are analyzed.

2 TASK 2

Report on task TASK 2, Data preprocessing

This section explains the methods and strategies for converting raw datasets into structured datasets suitable for analysis and model building. Two datasets are used in this training, named Corridor data and Bottleneck data, which store pedestrian trajectories in two different scenarios. Each dataset has five columns, storing the pedestrian's ID, frame number, and 3D coordinates.

2.1 Calculate speed

The process begins with logging the start of dataset creation. The code then processes each raw data file by reading it into a DataFrame containing columns ['ID', 'FRAME', 'X', 'Y'], which represent an object's identifier, the frame number, and its X and Y coordinates. The Z coordinate data is deleted here because by observing the coordinate changes of the same pedestrian, it can be found that there is no displacement in the Z direction, so it does not contribute to the calculation speed. Subsequently, the code calculates the speed for each point by determining the Euclidean distance moved between consecutive frames and converting this distance to meters per second. This speed calculation is performed individually for each object (denoted by 'ID') and is added as a new column in the DataFrame.

```
for pedestrian in data_groupedbyID.values():
    coordinate_diff = pedestrian[['X', 'Y']].apply(lambda row: row.diff(), axis=0)
    distance_calculation = np.sqrt(coordinate_diff['X']**2 + coordinate_diff['Y']**2)
    distance_calculation.iloc[0] = distance_calculation.iloc[1]  # Let the first and second frame have the same s
    pedestrian["speed"] = distance_calculation * 0.16  # The unit is m/s
```

Figure 8: speed calculating

First we process the data for each pedestrian and group them by their ID. For each pedestrian, it calculates the difference in X and Y coordinates between consecutive frames using the diff() function. The Euclidean distance between consecutive points is then calculated using the square root of the sum of the squared differences in the X and Y coordinates. The first and second frames are assigned the same speed value for consistency. Finally, the calculated speed is converted to meters per second by multiplying it by 0.16 and added to the pedestrians' DataFrame as a new column labeled "speed".

2.2 Finding 10 nearest neighbors

The next step is to get the specific 10 nearest neighbors around each point. First, we classify pedestrians by frame to ensure that all points within the same frame are processed collectively. Next, for each frame, the code identifies the K nearest neighbors of each point using Euclidean distance, calculates the coordinate differences between each point and its neighbors, and flattens these differences into a single array for each point. It also calculates the mean spacing ($\bar{s}_K$) of each point, which is the average distance to its K nearest neighbors.

```
for frame in df_groupbyF.values():

    if frame.shape[0] <= K:
        continue

    points = frame[['X', 'Y']].values
    neighbors = NearestNeighbors(n_neighbors=K+1, metric='euclidean').fit(points)
    k_distances, k_indices = neighbors.kneighbors(points)
    nearest_diffs = np.array([points[k_indices[i]][1:] - points[i] for i in range(len(points))])
    flattened_diffs = nearest_diffs.reshape(nearest_diffs.shape[0], -1)
    sk = k_distances.sum(axis=1) / K  # mean spacing
    features = np.concatenate((sk.reshape(-1, 1), flattened_diffs), axis=1)
    frame_samples.append(np.concatenate((features, frame[['speed']].values), axis=1))
```

Figure 9: Find nearest neighbors

We use the "generate_data" function to process the DataFrame "*df*_with_speed" by grouping it by the "FRAME" column. It iterates over each frame, recording the frame number and the number of points. For each frame, it skips frames with less than K points, and for the remaining frames, it extracts the X and Y coordinates of the points. Using the Nearest Neighbors algorithm, it finds the K+1 nearest neighbors for each point and calculates the coordinate difference.

2.3 Training data generate

To summarize the above results, we use the calculated speed as the target, mean spacing and the distance between each pedestrian and 10 neighbors (expressed by Δx and Δy), generating a total of 21 columns of data. We use these 21 columns of data as features to generate data for training.

3 TASK 3

Report on task TASK 3, Implementation of Weidmann model

3.1 Datasets

The datasets used in this study contain detailed records of how pedestrians move, gathered under strict experimental conditions that mimic real places like hallways and tight spots where people often crowd together. This data includes the exact time and place of each participant during the experiments. With this information, we can accurately study how fast pedestrians move in different areas. This makes it a great resource for checking if the Weidmann model can correctly predict how pedestrians flow and move in these settings. And two different datasets will be used, namely R (ring or corridor) and B (bottleneck) dataset. The datasets have one feature (mean spacing $\bar{s}_K$) and one target (velocity v). The dataset R and B are shuffled and split respectively, with the ratio of test set being 50%. As a result, 4 datasets are generated, namely `train_R, train_B, test_R, test_B`. A model is fitted with `train_R` and tested with `test_R`. The same for `train_B` and `test_B`.

3.2 Fit the models and get Weidmann parameters

To implement the Weidmann model, it was very important to correctly figure out the settings or the right parameters that describe how pedestrians behave in different spaces. The Weidmann model suggests that there's a complex link between crowd density, pedestrian size and their initial speed. This idea is shown in a basic diagram in the model. The model assumes that as places get more crowded, people walk more slowly until the space is so crowded that they can hardly move.

To fit the model to our data, we employed non-linear least squares regression, aiming to find the parameter values that best reconcile predicted speeds with observed data. The fitting process involved adjusting parameters such as the desired speed (v_0), the pedestrian size(l), and the adaptation time (T). These parameters are critical as they determine the model's sensitivity to changes in pedestrian density and its ability to predict speed variations accordingly.

3.3 Predictions on test sets and fitted curves

The Weidmann model's predictions were plotted against actual speed measurements from the bottleneck and ring scenarios, as depicted in Figure 10. The green and orange curves represent respectively the fitted models for the ring and bottleneck scenarios. The fitted curves reveal that while the model generally captures the trend of speed reduction with increased mean spacing, it underestimates speeds at lower densities, especially in the bottleneck scenario. It is evident that the bottleneck scenario exhibits higher variability in pedestrian speeds at similar densities. This suggests that the Weidmann model may require adjustments to better accommodate scenarios with varying pedestrian densities. The original measurements (scatters) reveal that the data points for bottleneck with higher mean spacing ($\bar{s}_K \geq 4$) are less sufficient, which causes worse performance with lower densities.

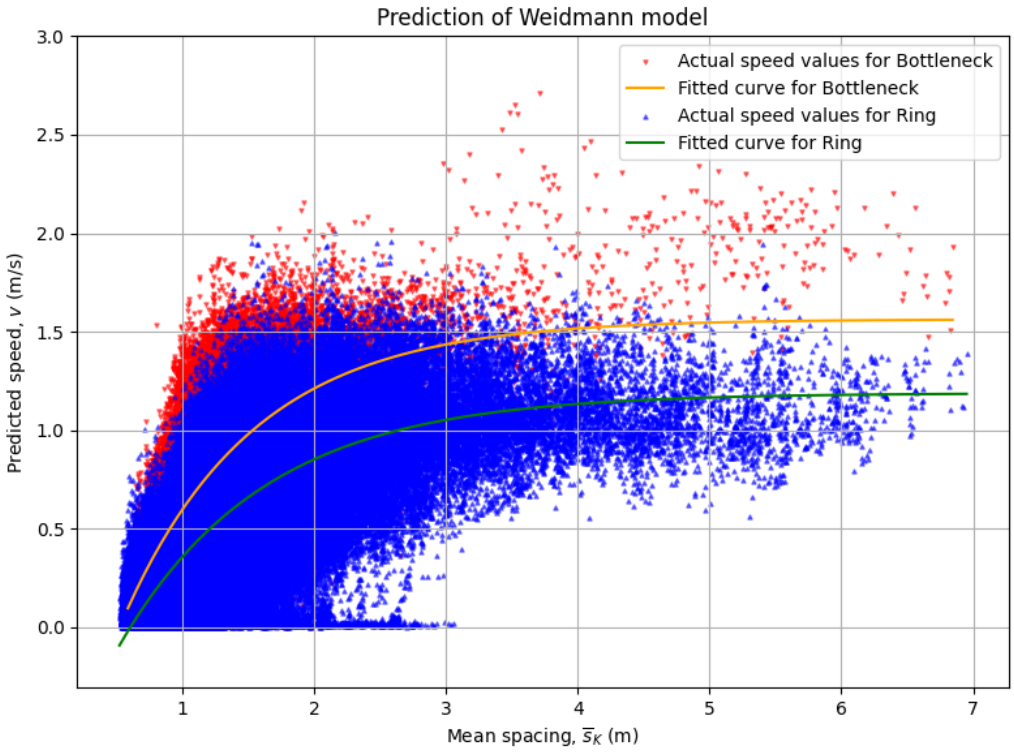

Figure 10: Prediction of Weidmann model on B/B and R/R

4 TASK 4

Report on task TASK 4, Implementation of neural network model

4.1 Training sets and test sets

For the implementation of the neural network model, the datasets were divided into training and test sets using the `train_test_split` function. The data was shuffled prior to splitting to guarantee randomness and maintain a balance in scenario variety between the training and testing sets, crucial for a fair evaluation of the model's ability to generalize across varied conditions. The division ensures that the model is trained on a diverse array of data while its performance is independently evaluated on unseen data. The datasets are shuffled and split equally, with each subset comprising 50% of the original data, ensuring a comprehensive representation of different pedestrian densities and scenarios typically encountered in urban settings. After we get `train_R, train_B, test_R, test_B`, `train_R` and `train_B` are concatenated and shuffled, which is named `train_RB`. `test_RB` is generated in the same way with `test_R` and `test_B`. Figure 11 displays the process of generating these datasets.

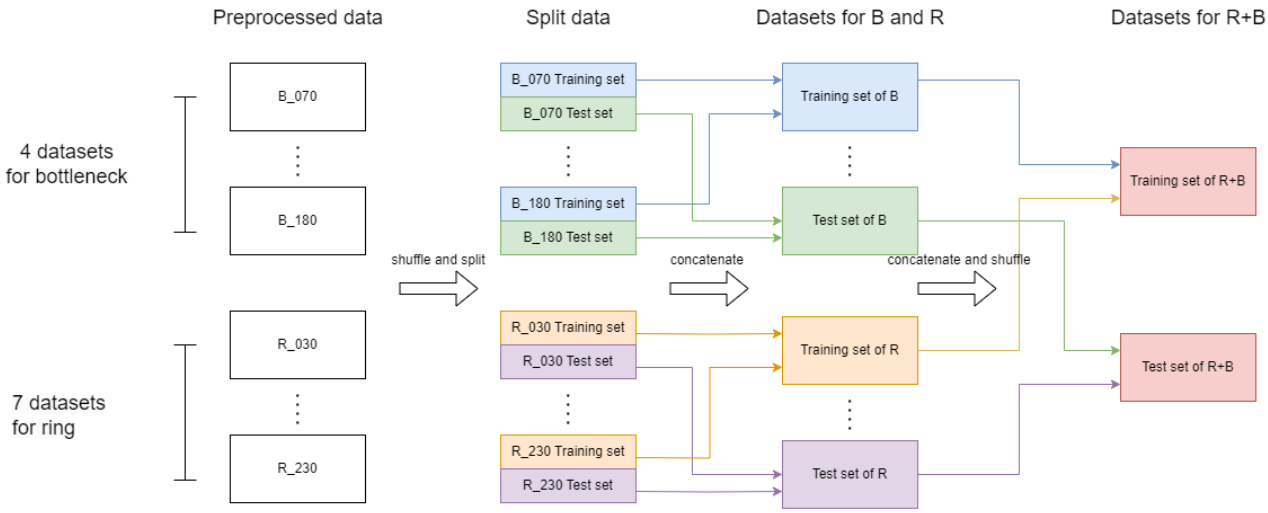

Figure 11: Generation of training and test sets for R, B and R+B

4.2 Hyperparameters

Choosing the right set of hyperparameters is crucial for the success of a neural network model. For this project, we explored several hyperparameters. The hyperparameters are demonstrated in Figure 12. Note that the $hidden\_sizes = [3]$ means the neural network has one hidden layer with 3 neurons, which is the optimal setting of the original thesis. Since the size of datasets in our implementation is much larger than that of the authors, this model size is too small to fit the data. In the following step the model structure will be further discussed.

4.3 Determine model structure

The model consists of multiple layers that help in capturing the non-linear relationships within the data. Each layer is fully connected and utilizes ReLU activation functions to introduce non-linearity, ensuring the model can learn more complex patterns such as the variations in speed due to different pedestrian densities and interactions. The $input\_size$ is 21 and the $output\_size$ is 1 because the datasets contain 21 features and 1 target.

Figure 13 reveals the test errors of ANN with different sizes of hidden layer w.r.t number of epochs. Initially, all models show a rapid decrease in error, indicating effective learning from the training data. Models with structures [22] and [26, 16] exhibited relatively higher test errors during training compared to other models, which showed approximately equal performance. Figure 14 demonstrates the lowest training and test errors w.r.t sizes of hidden layers. As the model size increases from [22] to [30, 25, 22], both training and test errors decrease. However, beyond [30, 25, 22], the training error begins to fluctuate, while the test error remains

```
hparams = {
    'input_size': 21,            # Number of input features
    'hidden_sizes': [3],         # List of numbers of neurons of each hidden layer
    'output_size': 1,            # Number of output neurons
    'num_epochs': 20,            # Number of training epochs
    'batch_size': 512,           # Batch size for training
    'learning_rate': 0.001,      # Learning rate for the optimizer
    'lr_decay_step_size': 10,    # Step size for learning rate decay
    'lr_decay_gamma': 0.1,       # Multiplicative factor of learning rate decay
    'random_state': 42           # Random seed for reproducibility
}
```

Figure 12: Hyperparameters in ANN

stable. According to these 2 figures, the models with structures [22] and [26, 16] are underfitted. Compared to the model [30, 25, 22], those with [40, 32, 25], [64, 32, 16] and [72, 36, 21] have more complex structures and more parameters but don't get improvement in performance, which indicates that they are potentially overfitted. Therefore, we choose the model of [30, 25, 22], to achieve a balance between model complexity and generalization capability.

4.4 Train and test on different datasets

The neural network is trained and tested on various datasets. The R dataset and the B dataset are also concatenated together to create a more sophisticated scenario. We also tried to train on one scenario and test on a different one. This showed us how the ANN reacts to the new environment. There are 7 combinations of training set/test set: namely 'R/R', 'B/B', 'R/B', 'B/R', 'R+B/R', 'R+B/B', 'R+B/R+B'.

4.5 Predictions on test sets, corresponding Weidmann parameters and fitted curves

The predictions of the ANN model R+B/R+B are depicted in the Figure 15. The input of the model is `test_R` and `test_B`. This graph highlights the model's capability to adapt its predictions based on the training data encompassing both bottleneck and ring scenarios. The model effectively differentiates between the two scenarios, predicting speeds higher for bottleneck conditions. We also fit the predicted speed curve for corresponding Weidmann model. The comparison with Figure 10 will be introduced in next section.

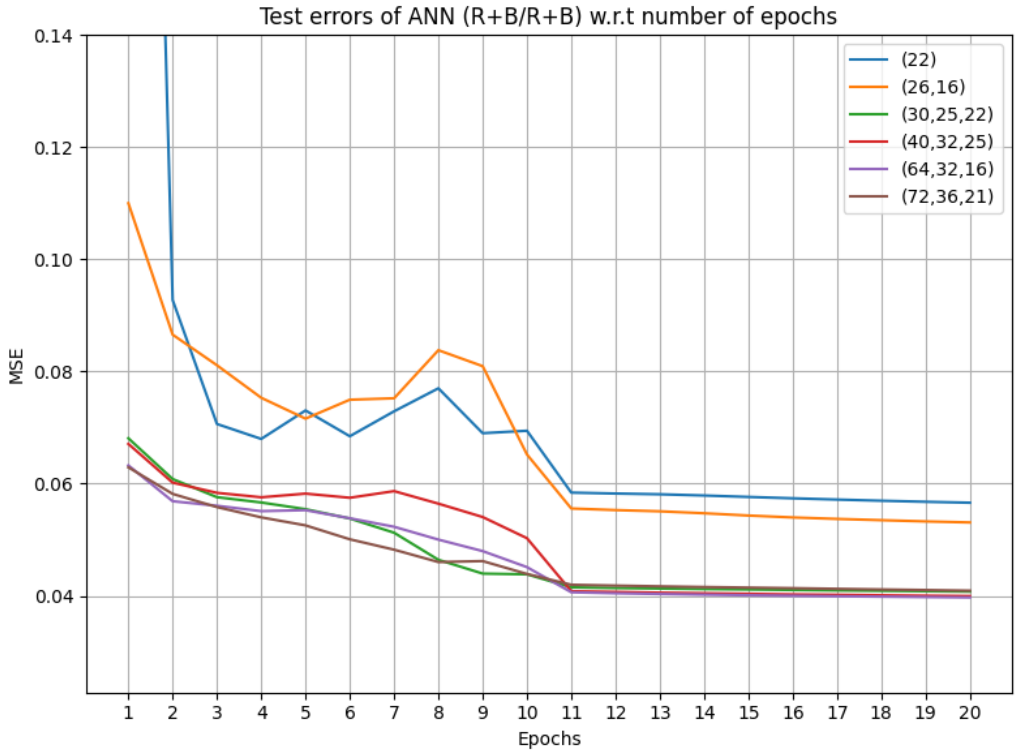

Figure 13: Test errors w.r.t number of epochs for models on R+B/R+B with different hidden sizes

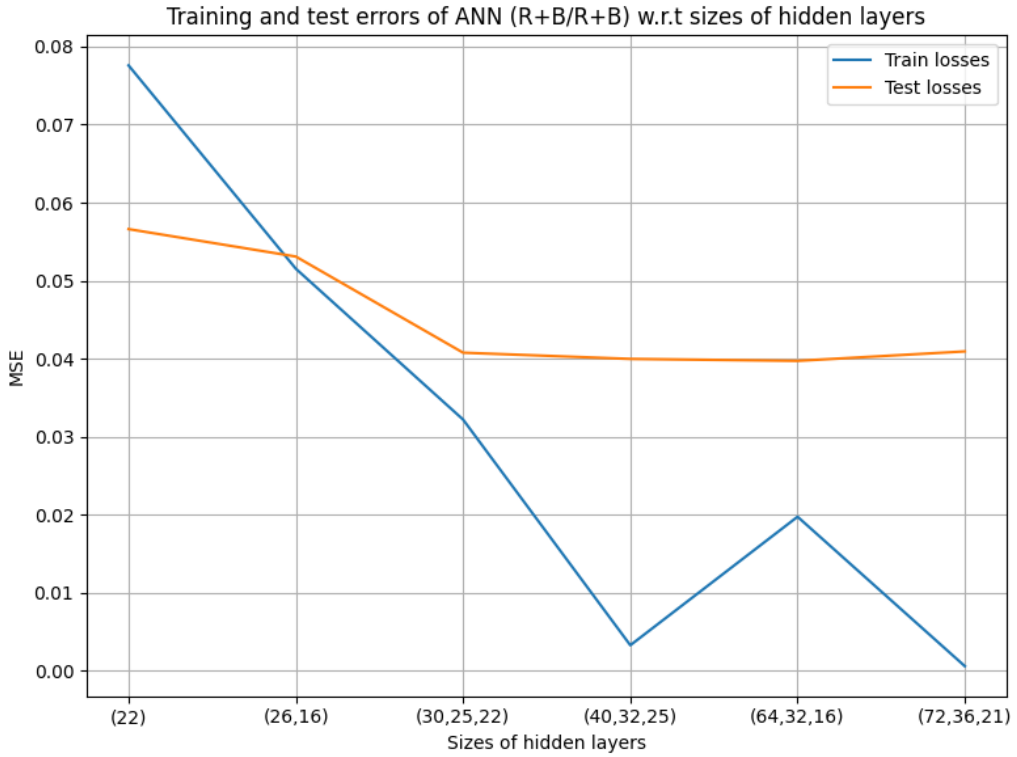

Figure 14: Training and test errors for models on R+B/R+B with different hidden sizes

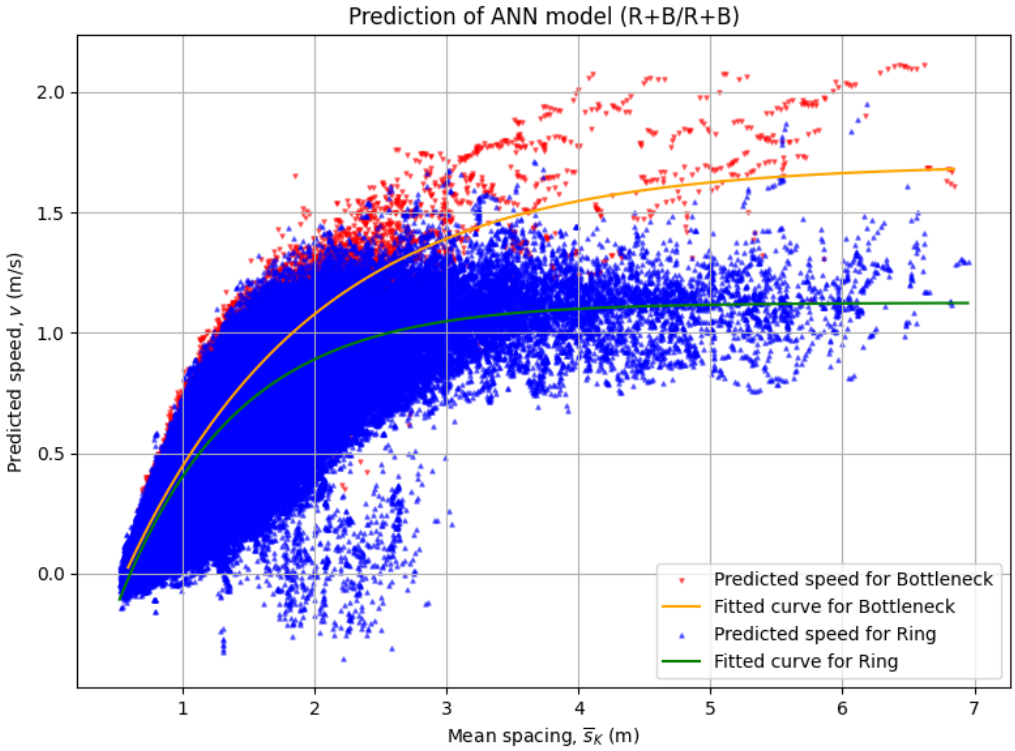

Figure 15: Prediction of ANN model on R+B/R+B

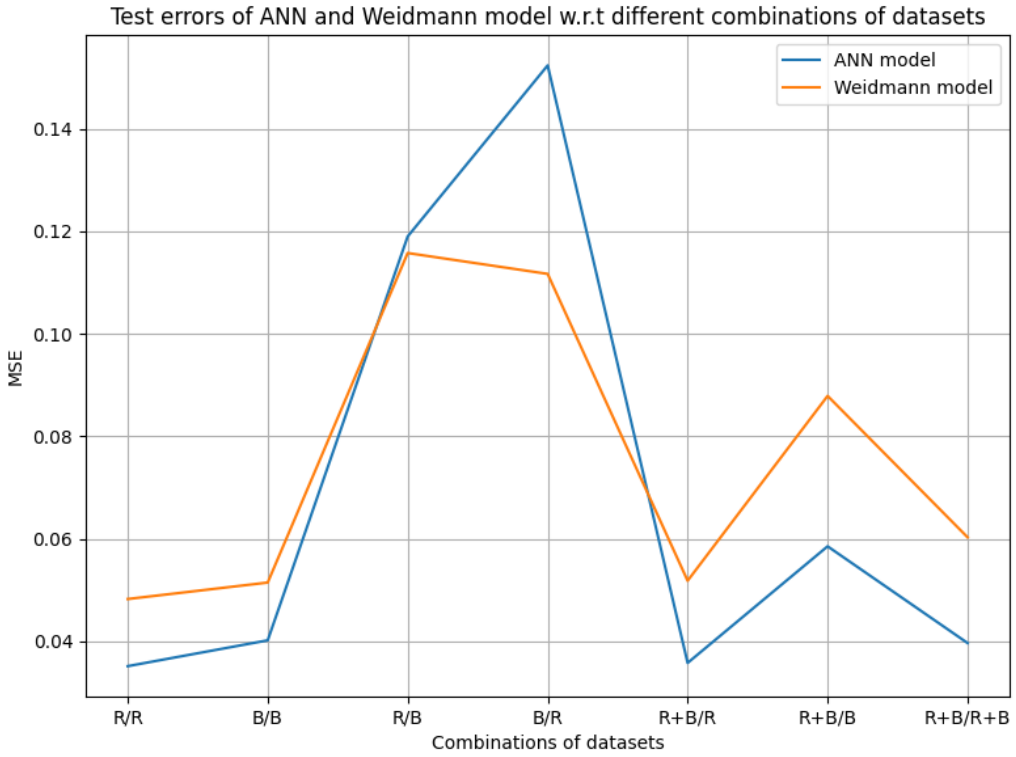

Figure 16: Test errors of ANN and Weidmann model w.r.t combinations of datasets

5 TASK 5

Report on task TASK 5, Visualization, comparison and discussion of results

5.1 Comparison of training and test losses (MSE) on different combinations of datasets between 2 models

Figure 16 demonstrates the lowest test errors of ANN and Weidmann model w.r.t combinations of datasets. When training set and test set are the same, i.e. R/R, B/B and R+B/R+B, the test errors are lower. When training set and test set are from different distributions, i.e. R/B and B/R, the test errors are extremely high. As for the rest, the models of R+B/R perform better than those of R+B/B, because dataset R have more data samples (817722) than dataset B (251338). In respect of model comparison, ANN performs better than Weidmann under most of circumstances except R/B and B/R, because neural network is more sensitive to the distributions of training and test sets. As for other combinations, ANN has higher fitting ability than Weidmann.

5.2 Comparison of predictions and Weidmann parameters between 2 models

In this subsection, predictions and Weidmann parameters of ANN model (Figure 15) and Weidmann model (Figure 10) will be compared. Note that the prediction of ANN model is a set of scatters, while that of Weidmann model is a fitted curve. The fitted curve of ANN model is based on fitting on predictions w.r.t mean spacing, which encodes the 3 Weidmann parameters for the corresponding ANN model.

l (m) stands for pedestrian size and is equal to $\bar{s}_K$ when $v = 0$, which means the crowd can't move at all when the mean spacing equals pedestrian size (an extreme case when the environment is very crowded, with people almost pressed against each other). In the figure, it represents the intercept of the curve with $\bar{s}_K$ axis. The fittings of l (m) are similar to each other for 2 models in 2 scenarios, which ranges from $0.5230m$ to $0.6045m$.

Table 1: Fitting of Weidmann model

Fitted parameters \ Experiment	R (Tordeux et al.)	R (ours)	B (Tordeux et al.)	B (ours)
$l\ (m)$	0.64	0.6045	0.61	0.5230
$T\ (s)$	0.86	0.9288	0.48	0.632
$v_0\ (m/s)$	1.60	1.1884	1.58	1.5633

Table 2: Fitting of ANN model on R+B/R+B

Fitted parameters \ Experiment	R (Tordeux et al.)	R (ours)	B (Tordeux et al.)	B (ours)
$l\ (m)$	0.63	0.6035	0.66	0.5650
$T\ (s)$	0.68	0.7919	0.50	0.8403
$v_0\ (m/s)$	1.44	1.1245	1.51	1.7008

$T\ (s)$ stands for time gap and reflects the steepness of the curve. The derivative of v w.r.t $\bar{s}_K$ is

$$\frac{\partial v}{\partial \bar{s}_K} = \frac{1}{T} e^{\frac{l-\bar{s}_K}{v_0 T}}$$

The steepness gets larger with smaller T. In the scenario of ring, the curve fitted by ANN model is steeper; in the scenario of bottleneck, the curve fitted by Weidmann model is steeper.

$v_0\ (m/s)$ represents desired speed and represents the converged speed when $\bar{s}_K$ tends to positive infinity, which means the pedestrians are so far away from each other that they will not influence each other's movement at all. The fittings of both models are similar with the scenario of ring, while the fitted $v_0\ (m/s)$ of ANN model (1.7008) is slightly higher than that of Weidmann model (1.5633).

The differences of fitting between Weidmann and ANN model are mainly due to the selection of input features. While the Weidmann model only takes $\bar{s}_K$ as the independent variable, the ANN model takes 21 features into consideration, which causes the difference of predictions.

5.3 Comparison with original thesis

The authors trained an extremely small dataset and the model size is also very small: one hidden layer with 3 neurons. Instead, our dataset includes all available data, which makes the model fitting more comprehensive. As Figure 4 and Figure 7 demonstrate, their datasets lack the samples with mean spacing over $3.5m$, which causes the models to be less representative in terms of larger mean spacing. As a result, it is incomplete to observe the convergence of the speed. In comparison, our models are able to predict the speeds with a wider range of mean spacing as input, which enables us to observe the convergence.

Moreover, a tiny dataset doesn't ensure the robustness of a model because the model learns insufficient features. If a few more data points are added to the training set, the model parameters may change unpredictably. On the contrary, we utilize all the available data generated from raw data and give the models sufficient data to learn, leading to stability when dealing with various inputs.

However, there is also drawback in our ANN model. Some predicted values of speed is negative, which is unreasonable in the problem definition. The weak interpretability of neural networks makes it difficult to avoid meaningless outputs.

The significant enlargement of dataset brings about some extreme and irregular cases that affects the model performance negatively. Compared to Figure 6 from original thesis, the test errors of our implementations are a little higher with following combinations: R/R, B/B, R/B, B/R, R+B/B. As for R+B/R and R+B/R+B, our ANN model outperforms the original one, while our Weidmann model underperforms the original one. For example, the test errors of FD and ANN on R+B/R+B in original thesis are approximately 0.050 and 0.041 respectively, while those in our implementation are 0.06028 and 0.03965.

Compared to the original thesis, our implementation of the Weidmann model in both scenarios shows smaller values for $l\ (m)$ and $v_0\ (m/s)$, and larger values for $T\ (s)$. The results of ANN model reveals smaller values for $l\ (m)$ and larger values for $T\ (s)$ in both scenarios. In the ring scenario, our model has a lower $v_0\ (m/s)$ than

that in original thesis, whereas in the bottleneck scenario, the opposite is observed. The cause of the difference may lie in the size of datasets. Thanks to more complete data, our models capture the features with much wider value ranges and this advantage is especially significant for the circumstances where mean spacing $\bar{s}_K$ is greater than $3.5m$.

References

[1] Christian Keip and Kevin Ries. Dokumentation von Versuchen zur Personenstromdynamik, Projekt "HERMES", 2009.

[2] Antoine Tordeux, Mohcine Chraibi, Armin Seyfried, and Andreas Schadschneider. Prediction of pedestrian speed with artificial neural networks, 2018.