Report for Project from group A

Tasks addressed: 5

Authors: Hirmay Sandesara (03807348)
Yikun Hao (03768321)
Yusuf Alptigin Gün (03796825)
Subodh Pokhrel (03796731)
Atahan Yakici (17065009)

Last compiled: 2025–07–29

The work on tasks was divided in the following way:

Hirmay Sandesara (03807348)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Yikun Hao (03768321)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Yusuf Alptigin Gün (03796825)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Subodh Pokhrel (03796731)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Atahan Yakici (17065009)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%

1 Task 1: Literature Background and Problem Formulation

Modeling pedestrian dynamics accurately in heterogeneous environments is a complex problem in the field of crowd simulation. Classical solutions such as velocity and decision-based models typically rely on nonlinear functions with a small number of interpretable parameters like desired speed v_0, time gap T, and pedestrian size ℓ; aiming to describe local interactions in a simple way.

However, empirical observations reveal that pedestrian behavior is strongly influenced by the type of infrastructure, such as corridors or bottlenecks, leading to geometry-dependent effects in speed and flow. For instance, the pedestrian speed for a given mean spacing differs significantly between ring (corridor) and bottleneck configurations (see Fig. 1). This makes it difficult for classical models to accurately predict movement in mixed or dynamic environments.

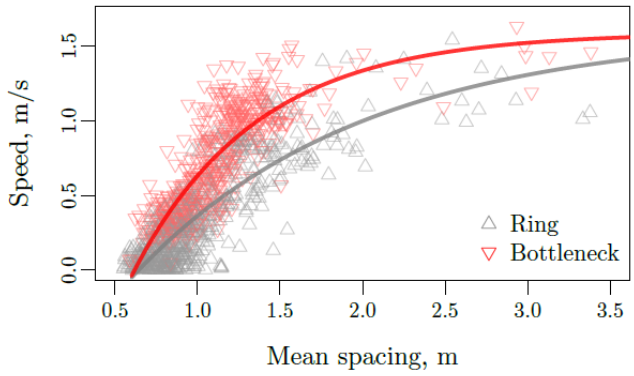

Figure 1: Observed speed as a function of the mean spacing with the 10 closest neighbours in ring and bottleneck experiments, from [2].

As an alternative, data-driven approaches such as Artificial Neural Networks (ANNs) have been proposed to learn complex nonlinear relationships directly from data. Unlike classical models, ANNs don't rely on physical assumptions and can utilize a variety of input features, such as the relative positions of a pedestrian's nearest neighbors (see Fig. 2).

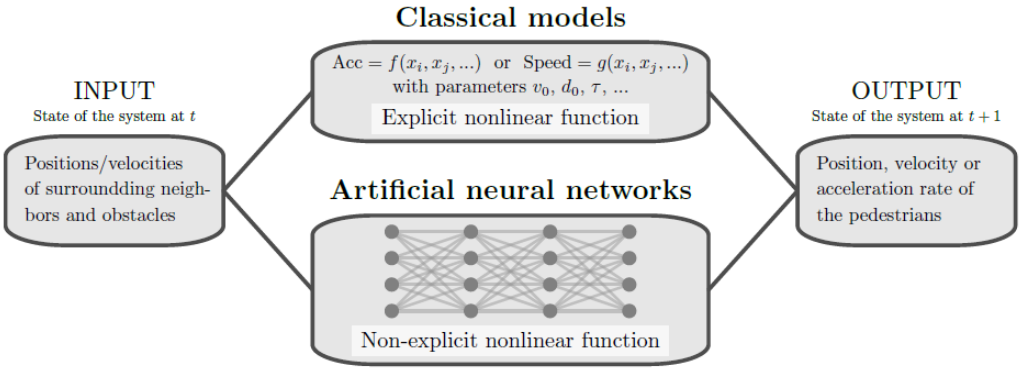

Figure 2: Comparison between classical models and ANN approaches, from [2].

In this project, we reproduce the methodology proposed by Tordeux et al. [2], comparing the predictive performance of the Weidmann fundamental diagram (FD) model and a feed-forward neural network. Both models are trained and tested on pedestrian trajectory datasets obtained from controlled experiments in two geometries: a closed ring corridor and a bottleneck. The evaluation metric used is the Mean Squared Error (MSE) between observed and predicted speeds.

Moreover, several ANN architectures were tested to evaluate how the complexity of the network influences performance. These architectures ranged from very small networks with a single hidden layer

(e.g., (1), (3)) to deeper (though still quite small) networks with multiple layers and increasing numbers of neurons (e.g., (4, 2), (5, 3), (10, 4)). The training and testing performance for each configuration is illustrated in Fig. 3, where the MSE is plotted for different architectures.

According to the paper, while increasing the number of hidden layers and neurons reduces the training error, this doesn't necessarily translate to better generalization. In fact, for more complex architectures, the testing error begins to increase, showing overfitting. The optimal trade-off between model complexity and generalization ability is achieved at the architecture with a single hidden layer and 3 neurons. This configuration yields the lowest testing error, suggesting it captures the essential nonlinear dynamics in the data without overfitting.

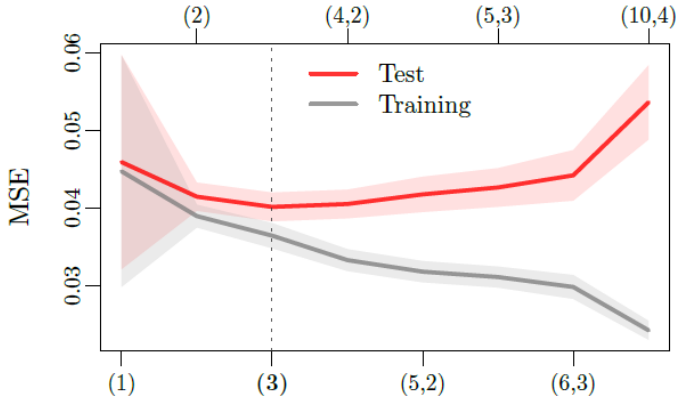

Figure 3: Training and testing error for different ANN architectures, from [2].

The final results highlight the inherent limitations of classical FD-based models in adapting to heterogeneous environments. While these models perform reasonably well in isolated, uniform scenarios, they lack the flexibility to generalize across varying geometries. These shortcomings are because of their constrained analytical structure and reliance on a small number of parameters, which makes them less capable of capturing context-dependent pedestrian behaviors.

Contrary to this, the data-driven nature of ANNs allows them to learn complex, nonlinear interactions among pedestrians from observed data, without requiring model assumptions. The ANNs demonstrate improved accuracy in predicting pedestrian speeds, showing better adaptability to unseen geometries. These findings support the adoption of ANN-based approaches as a more robust alternative for modeling pedestrian dynamics in complex real-world scenarios such as train stations, stadiums, or evacuation settings, where spatial structures and crowd densities can vary significantly.

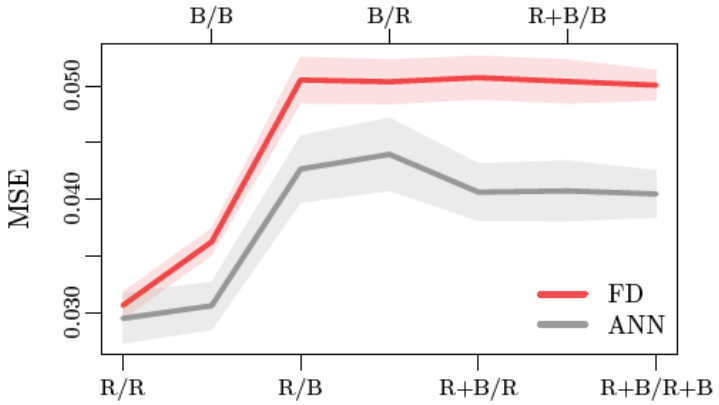

Figure 4: Fitted fundamental diagrams for Ring and Bottleneck geometries, from [2].

2 Task 2: Implementation of the Models

The code used in the project is split into two files, called `utils.py` and `main.ipynb`. The `utils.py` consists of the supporting functions `process_dataset`, `weidmann`, and `neural_network`; used to create the necessary elements required in the paper. With these supporting functions, the `main.ipynb` file sequentially processes the data, and uses the Weidmann model plus the neural network functions for validation and testing. The objective of this task was to create the necessary utility functions, which could then be used for rerunning the experiments defined in the paper with some additions of our own.

2.1 Processing the Dataset

To train models from the data, features need to be extracted into the necessary forms given in the paper. This is done with the `process_dataset` function. The original dataset contains pedestrian trajectory data across sequential frames, with each row representing a pedestrian at a given time step. This data is divided into many `.txt` files, with each of them containing a different number of entries but the same structure of the data. The following are the columns in each `.txt` file:

Column	Description
`id`	Unique pedestrian identifier
`frame`	Frame number (frame rate is 1/16s)
`x`	x-coordinate of the pedestrian position in 3D
`y`	y-coordinate of the pedestrian position in 3D
`z`	z-coordinate of the pedestrian position in 3D

Table 1: Structure of the unprocessed dataset

The goal in data processing is to structure the data into the two datasets given in the paper:

- A dataset for the Weidmann model: $(\text{mean spacing}, \text{speed})$
- A dataset for the neural network: $([\text{mean spacing}, \Delta x_1, \Delta y_1, \ldots, \Delta x_K, \Delta y_K], \text{speed})$

where $\Delta x_i, \Delta y_i$ is the relative distance of the pedestrian to its 10 closest neighbors in the x and y axises.

To extract the necessary features, `process_dataset` processes pedestrian trajectory data frame by frame. To do this, the data is first grouped by frames. By doing so, it's able to eliminate frames containing fewer than $K = 10$ pedestrians. Since the approach in the paper is done for $K = 10$, each frame needs at least 11 pedestrians. It then starts looping through each frame and does the following:

- For each pair of consecutive frames t and $t + 1$, the function retrieves all positions and finds the pedestrians that are present in both frames.
- Extracts the positions of the pedestrians present at both time steps. It'll use this information to calculate the speed of the pedestrians.
- Constructs a KD-Tree from the positions at time t for efficient nearest-neighbor lookup. By querying the KD-Tree, the IDs and positions of the $K + 1$ nearest neighbors are retrieved (including the agent itself for now).
- For each pedestrian, computes the speed v_i from the Euclidean distance between their positions at t and $t + 1$, scaled by the frame rate $f = 16$:

$$v_i = \left\| \mathbf{x}_i^{t+1} - \mathbf{x}_i^t \right\| \cdot f \tag{1}$$

- Identifies the K nearest neighbors (excluding self) and calculates their relative positions $(\Delta x_i, \Delta y_i)$ with respect to the initial pedestrian.
- Computes the mean spacing $\bar{s}_K$, which is the average distance to the $K = 10$ nearest neighbors:

$$\bar{s}_K = \frac{1}{K} \sum_{i=1}^{K} \sqrt{(x - x_i)^2 + (y - y_i)^2} \tag{2}$$

- Finally, it creates the two datasets mentioned before.
  - For the Weidmann model: It appends the tuple $(\bar{s}_K, v_i)$.
  - For the neural network: It appends input vector $[\bar{s}_K, \Delta x_1, \Delta y_1, \ldots, \Delta x_K, \Delta y_K]$ with target v_i.

The constructed datasets are then returned for use in validation and testing. Also, there are a couple of important final details to note about data processing, which are listed below.

- In our approach, each text file is processed by the `process_dataset` function one by one and combined afterwards to create the final "bottleneck" and "ring" datasets. The way of handling the dataset was abstract and not properly documented in the paper; thus, we decided to process the data like this.
- The units given in the dataset are in centimeters (cm). The conversion from cm to meters (m) is handled in the `main.ipynb` file by dividing each x, y, and z column data by 100.

2.2 The Weidmann Model

The Weidmann model describes pedestrian behavior as a function of the average local density. The function's analytical form is given as:

$$v = FD(\bar{s}_K, v_0, T, \ell) = v_0 \left(1 - \exp\left(\frac{\ell - \bar{s}_K}{v_0 T}\right)\right) \tag{3}$$

where:

- $\bar{s}_K$: mean spacing to K nearest neighbors
- v_0: desired walking speed
- T: time gap between pedestrians
- ℓ: size of a pedestrian

Since the Weidmann model is a mathematical function with parameters, it's easy to create as code. By creating the `weidmann` function with the necessary parameters and returning the formula, we're able to use the `curve_fit` function of `scipy` library to fit the Weidmann model to the Weidmann dataset. Thus, the model forms the baseline for comparing the learned behavior from the neural network.

2.3 Neural Network

With the `neural_network` function, a fully connected feed-forward neural network training is implemented using TensorFlow/Keras. In addition, a helper function `build_model` is implemented to compile the actual network itself. Before the `neural_network` function goes through the training steps, it first calls the `build_model` function to compile the network with the given parameters. The architecture of the neural network is:

- Input layer of size $2K + 1$. With inputs:
  - The mean spacing to the K nearest neighbors, $\bar{s}_K$.
  - The K relative positions of those neighbors, $(x_i - x, y_i - y)$.
- Several hidden layers with ReLU activation.
- A single linear output neuron predicting speed.

The model is trained using Mean Squared Error (MSE) as the loss function and Adam optimizer. As discussed in the paper, bootstrapping over samples is added with the default cross-validation split of $n = 5$, as given in the paper. The network is fully customizable with many parameters. This allows us to do the tests given in the paper, like the 50/50 test, validation split, small networks with very few neurons in the hidden layers, while also giving us the freedom to do more. The paper tests various hidden layer configurations and finds that a single hidden layer with 3 nodes, $H = (3)$, is optimal. The parameters with example values of the `neural_network` function are given in the Table 2.

Hyperparameter	Value
X_train	50% of the dataset
Y_train	50% of the dataset
Hidden Layers	[3, 5]
Learning Rate	0.001
Batch Size	32
Epochs	100
K-folds	5

Table 2: Neural network training configuration.

2.3.1 Python Implementation

The `build\_model` function uses the Keras library to construct a flexible feed-forward neural network. The `layers` argument allows for specifying different architectures, including the optimal $H = (3)$ configuration found in the paper.

Listing 1: Function to build the ANN model using Keras.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
import keras.losses
```

2.3.2 Results Visualization

In addition to the three main functions used for the experiments, to better understand and compare the model performances, two additional helper functions were implemented: `train_with_bootstrap_validation` and `evaluate_nn`. These functions are important for validating the neural network under various training and testing configurations.

- `train_with_bootstrap_validation` trains a neural network multiple times using bootstrap resampling. In each iteration, it draws a random sample with replacement from the training set, trains the model, and evaluates both training and test error using Mean Squared Error (MSE). It returns the mean and standard deviation of MSEs for both the train and test datasets. All in all, this process gives insights into the generalization capability of the model.

- `evaluate_nn` builds on top of the bootstrap validation by running experiments between the corridor and bottleneck datasets. It tests the model in seven different training/testing configurations, including training and testing on the same dataset (e.g., Corridor → Corridor), as well as across datasets (e.g., Corridor → Bottleneck). It also includes a combined scenario where both datasets are merged. This function prints a summary table at the end, displaying average MSE scores and their standard deviations for all train/test configurations to make comparisons between how well the model generalizes when trained and tested on the same or different domains

3 Task 3: Tests on Simple Examples

After having implemented the appropriate functions for Weidmann and a neural network, we created a Jupyter notebook to use real-life data (see [1]), the same one used in [2] to validate our code. Initially, only one data file was processed, as described in Section 2, and the resulting plot of the dataset is seen in Figure 5.

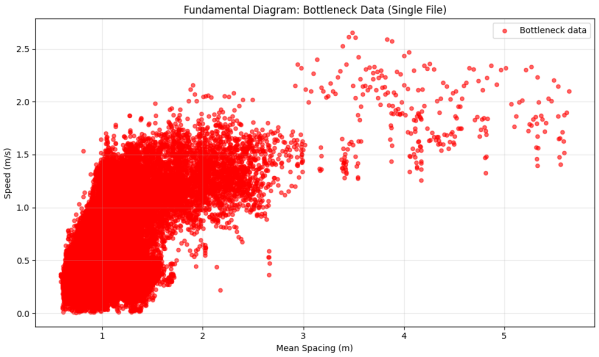

Figure 5: Fundamental Diagram for a Bottleneck Experiment With The Size 0.7m

Figure 5, also known as the Fundamental Diagram (FD), shows how the mean spacing between pedestrians affects their speed. As one expects, the speed is lower at higher density (low mean spacing) and vice versa. This behavior is clearly seen in the graph.

3.1 Weidmann Model

Doing a curve fit on this scatter plot gives the 3 different parameters that are then used for the Weidmann model [3] (see 3). Doing a curve fit (using an inbuilt function from `scipy`), gives the resulting image seen in Figure 6. This curve fit is done on a portion of the data, while the rest is used for testing.

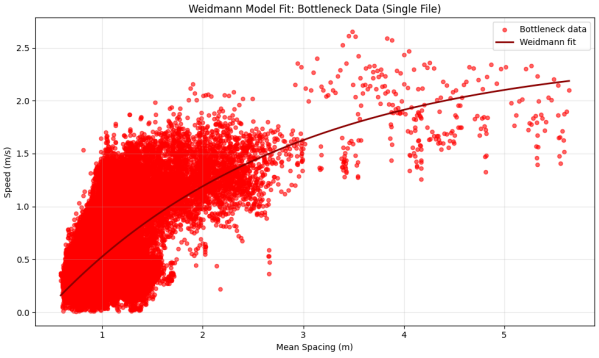

Figure 6: Figure 5 with a curve fit applied

The resulting parameters from the fit are:

$$v_0 = 2.4702 ms^{-1}, t = 0.8647 s, l = 0.4276 m$$

Using these parameters and the Weidmann equation, the speed value for each spacing (from the test set) is calculated. This resulting value is the prediction of the Weidmann model. This is a physics-based approach that uses pedestrian dynamics rather than learning patterns from data like a neural network. The speed is assumed based on the space around pedestrians using a mathematical relationship. Nonetheless, when comparing the real value to the predicted value, the mean squared error (MSE) is 0.049 $m^2 s^{-2}$ (the RMSE is 0.221 ms^{-1}). This means that the predicted value is, on average, off by 0.221 ms^{-1}. This error is significant for lower speed values, and is expected as Weidmann doesn't perform very well for lower speeds and is better suited for a free flow of pedestrians [3].

3.2 Neural Network

One of the outputs of data processing (see Section 2) is a dataset that can be used for training and testing a neural network. Using this data and the `neural_network` function (see Section 2), a feed-forward neural network is trained. The resulting MSE for train and test using parameters in Table 3 are 0.04122 m^2s^{-2} and 0.040886 m^2s^{-2}. This is better than the results obtained with the Weidmann model.

Hyperparameter	**Value**
X_train	50% of the dataset
Y_train	50% of the dataset
Hidden Layers	[3]
Learning Rate	0.01
Batch Size	64
Epochs	10
Bootstrap	20

Table 3: Neural network training configuration.

The neural network can learn crowd dynamics from the data. Unlike the Weidmann model, it isn't just limited to the mean spacing around pedestrians but also features of the data, such as neighbor speed, local densities, and more. The dataset is split into two halves for training the network and testing, and a bootstrapping approach with 20 independent training runs is done with randomly sampled data. This ensures robustness and avoids overfitting.

In our test case, both models were able to successfully predict pedestrian speed based on experimental data, with the neural network slightly outperforming the physics-based model. The data-driven approach for prediction definitely works and shows promise, mainly due to its ability to learn complex, nonlinear relationships in the data that are difficult to capture with traditional modeling. The adaptability of hyperparameters also means that the training and prediction can be altered to suit whatever is needed, such as accuracy and time consumption. Further discussions into this and discussions of results are done in Section 4.

4 Task 4: Comparison of the Results

This section compares the performance of the implemented Fundamental Diagram model and the Artificial Neural Network. For the purpose of comparative result analysis, tests under a variety of data combinations are conducted.

4.1 Combinations of Datasets

The combinations of datasets can be split into the following scenarios:

- **Homogeneous Validation**: The models are trained and tested on datasets from the same dataset (i.e., both trained and tested on Corridor or both trained and tested on Bottleneck). This combination is the baseline and indicates the basic model capacity.
- **Cross-Validation**: The models are trained on one geometry and tested on the other (i.e., trained on Corridor and tested on Bottleneck, vice versa). This combination indicates the model's generalization power on unseen data structures.
- **Mixed-Data Validation**: The models are trained on a combined dataset containing both Corridor and Bottleneck, then tested on individual as well as combined datasets. This scenario assesses the model's performance on a mixed dataset, reflecting more realistic and complex environments.

No.	Training Dataset	Testing Dataset
1	Corridor	Corridor
2	Bottleneck	Bottleneck
3	Corridor	Bottleneck
4	Bottleneck	Corridor
5	Combined (Corridor + Bottleneck)	Corridor
6	Combined (Corridor + Bottleneck)	Bottleneck
7	Combined (Corridor + Bottleneck)	Combined

Table 4: Training and Testing Combinations

4.2 The Weidmann Model Performance

After concatenating all data for each set, corridor, and bottleneck, we plot the fundamental diagram and look at the behavior of the crowd in these scenarios. Figure 7 shows the scatter plot and the curve fit, which gives the values needed for the calibration of the Weidmann model for the datasets seen in Table 5.

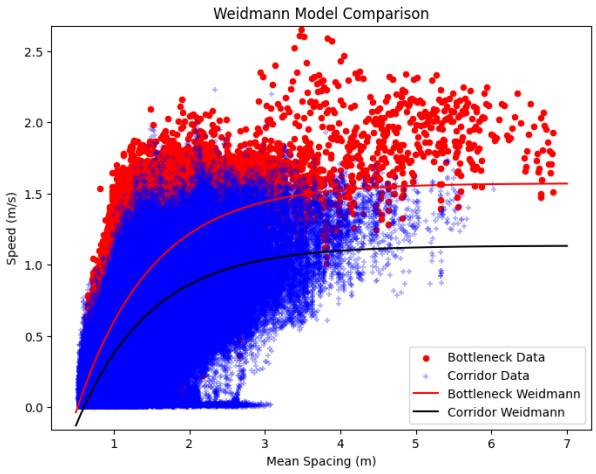

Figure 7: Fundamental diagram of all data points

Parameter	Bottleneck	Corridor
Desired speed (v_0)	1.573 ms^{-1}	1.13 ms^{-1}
Time gap (T)	0.633 s	0.865 s
Pedestrian size (l)	0.523 m	0.606 m

Table 5: Parameters for Weidmann model from curve fitting

Using these parameters, we can start training and testing the Weidmann model with different validation types.

Figure 8 shows the performance of the implemented Weidmann model across the different scenarios. We can observe from the figure that the Weidmann model achieves its best performance in homogeneous scenarios. The MSE for both Corridor → Corridor and Bottleneck → Bottleneck is at the lowest level observed in all tests, showing an average MSE of 0.0567. However, during cross-validation, the model's MSE increases significantly, peaking at 0.116 for the Bottleneck → Corridor test, and the average MSE sharply increases to 0.0904. This shows the model has poor generalization capability. At last, with mixed-data validation, the model yields an increasing performance in comparison to Cross-Validation, yielding an average MSE of 0.0675. This indicates that the model has learned more general features from both datasets.

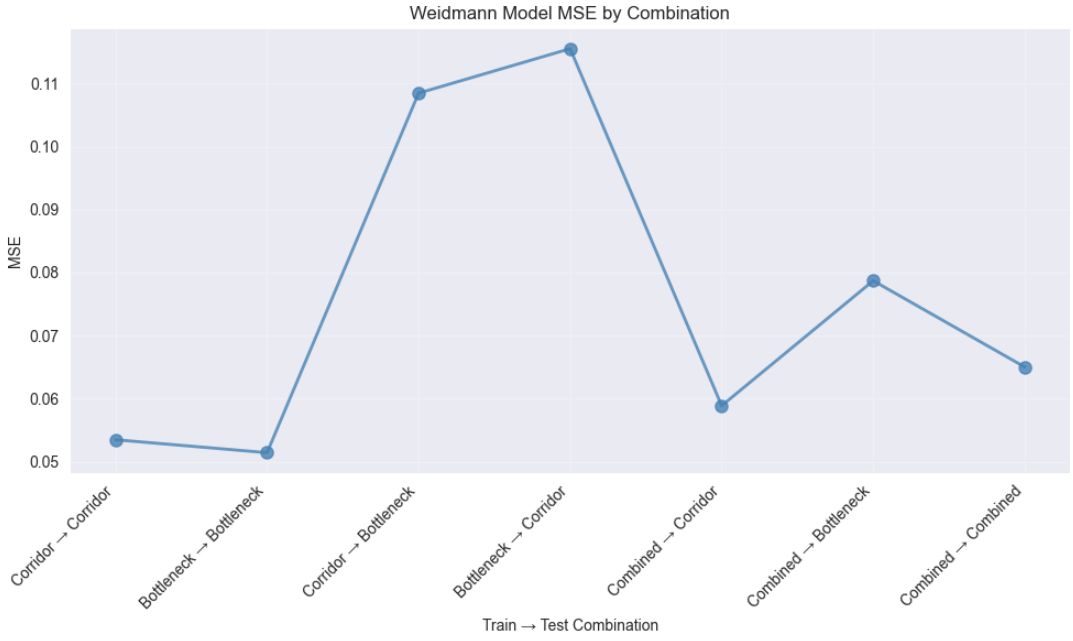

Figure 8: Weidmann Model MSE by Combination

4.3 The Neural Network Model Performance

We evaluate the implemented Artificial Neural Network (ANN) for the performance comparison between models. First, we explore how the artificial network structure influences the model's performance and identify the optimal architecture. A total of ten distinct network configurations, varying in the number and size of hidden layers, were implemented, namely `layer_configs`= [[1], [3], [4, 2], [5, 2], [5, 3], [6, 3], [10, 4], [8, 4, 2], [100,4], [100,10]]. This evaluation was performed with fast testing parameters to ensure the training and testing efficiency. For each architecture, the model was trained on a subset of 10,000 samples for 5 epochs, with the process repeated for 10 bootstrap runs. The learning rate is set to 0.01, and the batch size is 256. As a result, the training and testing process is relatively fast, generating a single data point on the graph took $\sim$ 10 seconds. The performance of each configuration across the seven validation scenarios is presented in Figure 9.

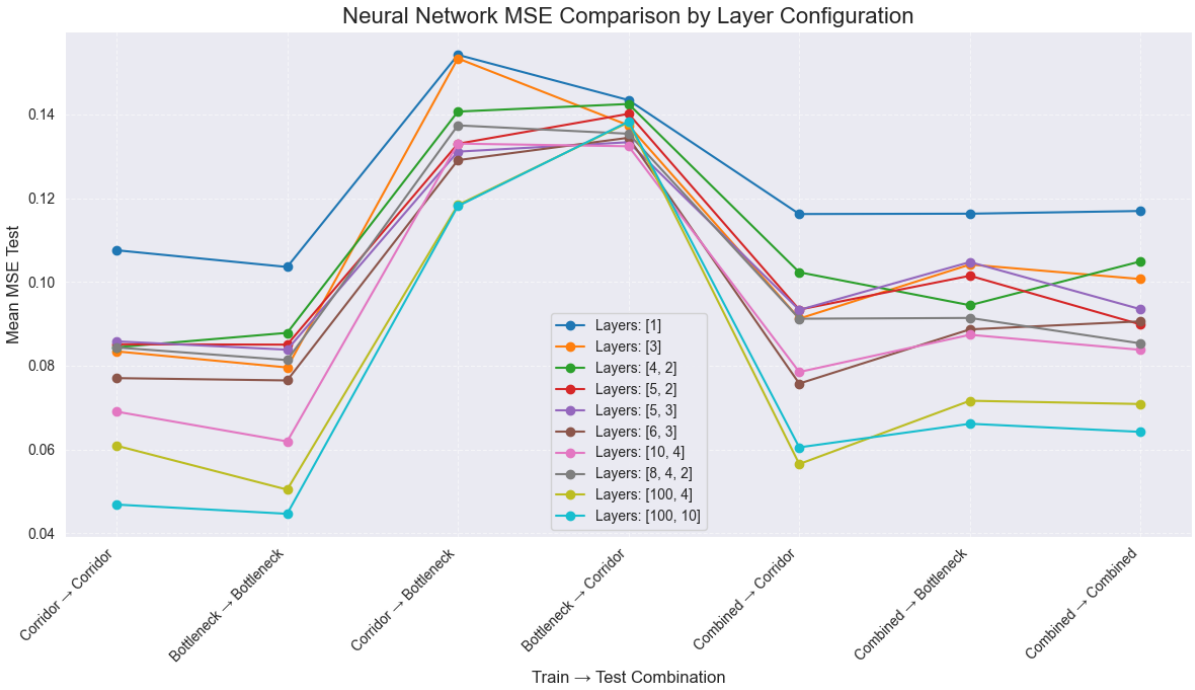

Figure 9: Neural Network MSE Comparison by Layer Configuration

We can observe from the figure that all tested architectures exhibit a similar performance pattern: they all perform well in homogeneous scenarios, poorly in cross-validation scenarios, and show intermediate performance when trained on combined data. The tendency indicates that, no matter the neural network architecture, the cross-validation results are better than homogeneous validation but worse than with mixed-data validation. This reinforces the conclusion that the model's ability to generalize is the key challenge. Another trend to notice is that in certain scenarios, there appears to be a positive correlation between model complexity and performance. For instance, in homogeneous validation and mixed-data validation, more complex architectures such as `Layers: [100, 10]` tend to achieve a lower MSE than many simpler configurations. However, this performance advantage decreases significantly in the cross-validation scenarios. Especially in the Bottleneck → Corridor scenario, nearly all models, regardless of their complexity, exhibit similarly high MSE values.

4.4 Comparison of Weidmann and Neural Network Performance

Bringing together Figures 8 and 9, we get Figure 10. This gives us an understanding of the performance of the Weidmann model across various scenarios compared to the neural network with various different hidden layers and neurons in the same scenarios.
It is clear from Figure 10 that the general tendency of the MSE values for various layers and Weidmann is consistent throughout the cross-validation across scenarios. The training and testing with data from the same experiments (Corridor → Corridor and Bottleneck → Bottleneck) gives better prediction than training with a combined dataset and testing on either the combined, bottleneck, or corridor. The worst, however, is training with one dataset and testing with the other.
Going deeper into the various layers and neurons shows that the Weidmann Model performs on par with simple structures with few neurons. When layers with a higher number of neurons are used, the error is significantly smaller than the one seen with the Weidmann model. However, it's important to note that performance alone isn't the only factor to keep in mind while selecting hyperparameters to train a neural network; other factors, such as speed of training, also play an important role when making decisions.

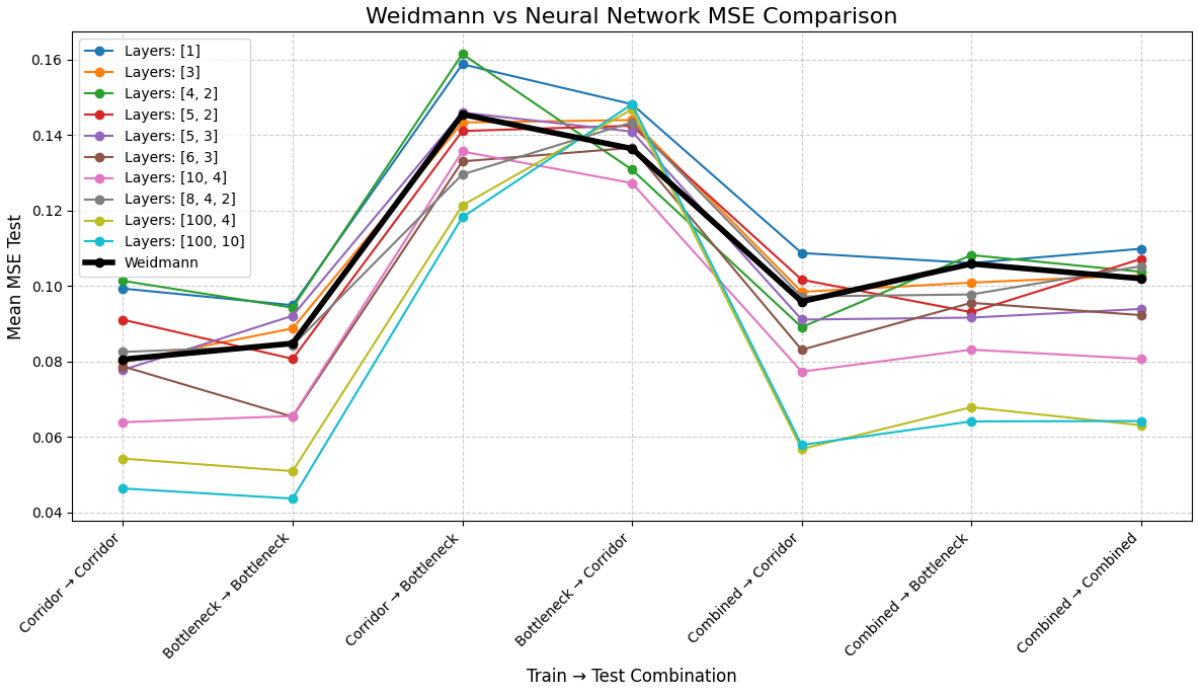

Figure 10: Neural Network MSE Comparison by Layer Configuration and Weidmann Model in the same plot

4.5 Best performing layer

Although the MSE is lower for more complex layers, the training time is significantly higher. Figure 11 shows a comparison between test and train MSE for the various layers as well as the time vs MSE for training these neural networks.

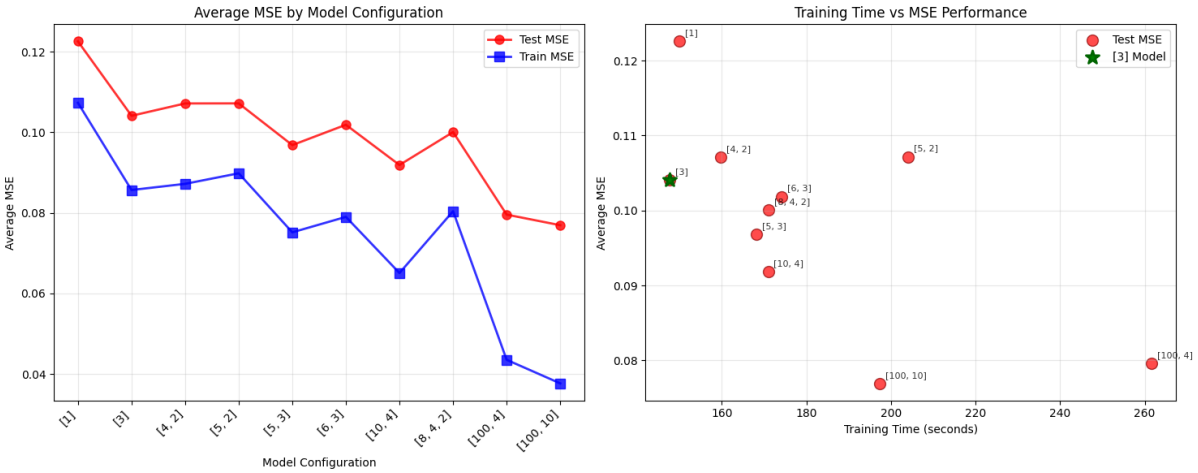

Figure 11: Comparison of MSE of various layers with model configuration and time

As already mentioned, the test MSE decreases with increasing model complexity, and so does the time taken to train the network. In [2], Tordeux et al. state that the layer [3] is well suited for the configuration of the neural network, as it's where the test MSE is the lowest. Although we don't see the exact replication of that in Figure 11, it's seen that in terms of time taken and mean MSE, it's one of the better ones. This discrepancy is highly likely due to the use of different parameters by Tordeux et al. and the fact that we use a sampled dataset for training and testing.

4.6 Further training with [3]

Since a simpler network structure is preferred given that the results are accurate enough, we explore further by changing some hyperparameters while keeping the hidden layer [3].

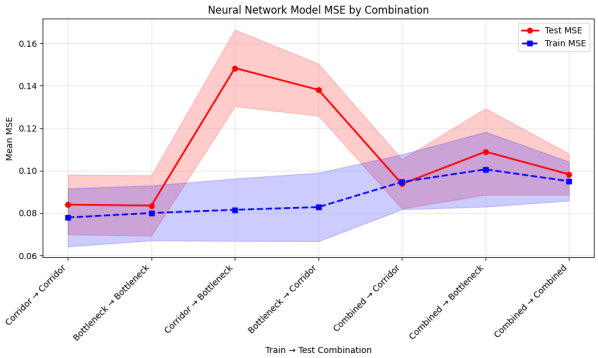

Figure 12: MSE of test and train data for NN with layer [3]

In Figure 12, the mean test and train MSE of the neural network is seen alongside the standard deviation. The tendency of MSE in regards to the scenario is, again, similar to earlier results. This is with a decreased batch size (256 to 128), increased epoch (5 to 10), decreased learning rate (0.01 to 0.001), and an increased bootstrap (10 to 50). This should result in smoother training curves and improved test MSE, amongst others, but with a larger training time. The exact effects of the changes are seen in Table 6

Parameter	**Change**	**Speed**	**Stability**	**Result Variability**
Batch Size	256 → 128	↓ Slightly	↑	↑ Slightly
Epochs	5 → 10	↓ (more compute)	↑	–
Learning Rate	0.01 → 0.001	↓	↑↑	↓ (controlled learning)
Bootstraps	10 → 50	↓ (longer analysis)	–	↓↓↓ (better confidence)

Table 6: Expected impact of hyperparameter changes on training behavior and performance.

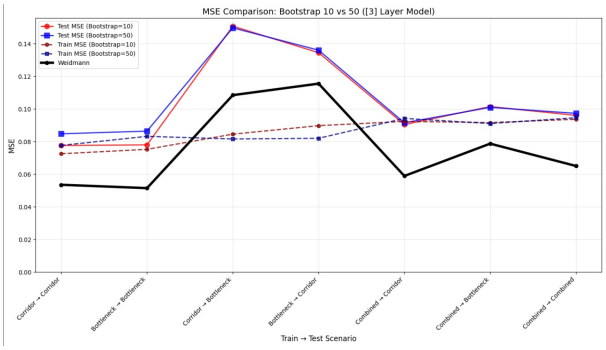

Figure 13: MSE comparison of NN with different hyperparameters and Weidmann

In Figure 13, we compare the results from Figures 12 and the results of layer [3] and the Weidmann model as seen in Figure 10. It is seen that the MSE is considerably lower for Weidmann compared to the neural network. In the neural network itself, the one with a bigger number of bootstraps seems to be performing worse than the one with fewer. This is certainly unexpected. The mean MSE got worse mainly because lowering the learning rate made the training slower, and even though the number of epochs was doubled, it was most likely not enough for the model to fully learn. Using a smaller batch size added more noise during training, which can sometimes help, but also made convergence slower here. On top of that, increasing the bootstrap count to 50 gave a more reliable average error, which might look worse compared to fewer bootstraps, which can be a bit optimistic. In essence, the model didn't

have enough training time to adjust well to these changes, leading to higher mean error.

The contrast to the value from the Weidmann model is most likely because we use a small fraction of the total data by randomly sampling the data points in neural network training, but the whole dataset for Weidmann. This was done to reduce the time to train and test the network. Using the entire set would have probably led to a different result in which the neural network would be better at predicting the speed of pedestrians, similar to the findings of Tordeux et al.[2]. Furthermore, a reason for the difference in the results from [2] is that the parameters used in the training are unknown, and the only way to replicate was trial and error. Finally, since a random sampling is done of the data, the same data points are not selected every time the program is run, making the resulting MSE vary. It was seen that whilst the results for the Weidmann model remained constant, some models of the neural network sometimes performed better and sometimes worse.

5 Task 5: Discussion of the Approach and Architecture

When it comes to predicting pedestrian behavior, the classical approaches, like Weidmann, which infers these dynamics with parameters of clear meaning, have lacked satisfactory results in complex geometries. Motivated by these limitations, we looked at the proposition of Tordeux et al. [2] in regards to using ANNs in predicting pedestrian behavior. This changed the paradigm towards a more data-driven alternative, capable of capturing more nuanced and context-dependent behaviors.

The testing across various scenarios revealed a more nuanced picture regarding the performance and limitations of both the classical Weidmann model and ANNs than is depicted in [2]. Contrary to initial expectations based on the related literature, the ANN doesn't consistently outperform the Weidmann model, but the performance strongly depends on the relevant settings such as data preprocessing, network complexity, and training configuration.

In [2], the scenario that gives the results presented in the paper isn't fully documented. Meaning the steps to preprocess the data, like if they actually combine all of them, use one `txt` file, which training hyperparameters are used, subsample sizes, etc., aren't specified. This made some of our results differ from those in [2], especially about the best ANN for prediction being a single-layer network with 3 neurons. We observed the prediction quality constantly increasing on the test set with larger network sizes, which seems like a more natural result considering even our largest networks $[100, 4], [100, 10]$ are relatively small.

Though some findings differ, there are still a lot of parallel results to [2] as well. In homogeneous scenarios (Corridor $\rightarrow$ Corridor and Bottleneck $\rightarrow$ Bottleneck), both models perform well, with simple ANN architectures yielding comparable performance, while deeper networks marginally improve results at the cost of increased training time. In contrast, both models suffer a sharp degradation in cross-validation scenarios (e.g., Bottleneck $\rightarrow$ Corridor), but the trend of larger networks achieving better results than the Weidmann model continues.

The most promising results we achieved for the ANNs were observed in mixed-data validation scenarios, where the network is trained on combined datasets and tested on individual or combined sets. In these cases, even the smallest ANN architectures showed promise, being just as good as the Weidmann model, while deeper architectures demonstrated significantly lower MSEs compared to Weidmann, showing that the ANN can exploit the added diversity of the training data to learn more general patterns.

In conclusion, the ANN approach showed promise, especially in heterogeneous or mixed-data contexts, where classical models like Weidmann are supposed to struggle. However, we found out that the success of the ANN isn't guaranteed and that careful preprocessing, training, and validation strategies need to be applied to unlock the ANNs' potential.

References

[1] Antoine Tordeux, Mohcine Chraibi, Armin Seyfried, and Andreas Schadschneider. Data from: Prediction of pedestrian speed with artificial neural networks. Zenodo, 2017. doi:10.5281/zenodo.1054017.

[2] Antoine Tordeux, Mohcine Chraibi, Armin Seyfried, and Andreas Schadschneider. Prediction of pedestrian speed with artificial neural networks. *arXiv preprint*, arXiv:1801.09782, 2018. arXiv:1801.09782, doi:10.48550/arXiv.1801.09782.

[3] Ulrich Weidmann. *Transporttechnik der Fußgänger: Transporttechnische Eigenschaften des Fußgängerverkehrs (Literaturauswertung)*. Schriftenreihe des IVT. ETH Zürich, Institut für Verkehrsplanung, Transporttechnik, Straßen- und Eisenbahnbau, 1993.