Data-Driven Models for Zebrafish Motion

Lukas Krenz

June 7, 2018

The goal of the project is comparing different data-driven models for the behavior of juvenile zebrafish. The models should be able to predict and simulate the individual motion of an animal reacting to its environment. The environment in our case is another fish and the walls surrounding the arena.

We approximate the movement by a piece-wise linear model. The wall forces are fit using a force-based approach.

We present multiple models for the social behavior, starting with a linear receptive field model. This model is then enhanced by considering temporal dependencies and non-linear effects. The final model is able to capture the entire trajectory distribution conditioned on the surroundings of the fish for each linear segment. We achieve that using a mixture density recurrent neural network model.

Introduction

This paper is concerned with the development of models of the behavior of juvenile zebrafish. These models should be able to predict and simulate the individual motion of one animal reacting to its environment (i.e. another fish and a wall).

A use-case for this project is steering a virtual fish in a virtual reality environment for animals. It can be used to perform experiments that investigate causal relationships in animal behavior. Furthermore, we build our models such that they easily extend to larger groups.

The movement of zebrafish can be approximated by discrete models that assume that the fish moves in a piece-wise linear fashion. After choosing a heading direction, the fish kicks off and moves in an approximately straight line. We model the heading change (figure 1) and distance for each kick.

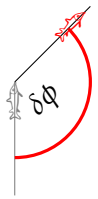

Figure 1: The heading change is the difference between the red and the gray heading.

We start with a simple model and refine it twice. Each modification drops assumptions about the behavior and thus creates a more flexible, data-driven model. The models developed in the earlier steps serve as baselines for the more complicated ones. We can thus see whether the assumptions are correct and how important each component of the model is.

We make the following contributions:

- We discuss a force-based model for the wall–fish interaction.
- We develop an data-driven receptive field approach for efficient spatial discretization.
- We evaluate whether including past trajectories improves the performance of the model. In other words, we evaluate whether the fish considers only its surroundings at kick-off time, or whether it posses a memory.

- We present a recurrent neural network that can handle multi-modal predictions and captures non-linear effects. This model describes the trajectory distribution directly instead of performing only a point estimate.

Pre-Processing

Before we can dive into the modeling, we need to massage the data into an appropriate format. This pre-processing stage is described in this section.

We first extract motion features from positions and orientations obtained by a video tracking software. Then we segment the continuous motion into discrete kicks. All steps mentioned thus far follow the protocol of Calovi et al.[1] with slight modifications. We then describe a simple wall model. This model is considered a part of the pre-processing stage because we use it to filter the data to areas with weak wall influence. We close this section with a description of our receptive field features that are a flexible representation of the surroundings of the animals.

[1] Daniel S Calovi et al. "Disentangling and modeling interactions in fish with burst-and-coast swimming reveal distinct alignment and attraction behaviors". In: *PLoS computational biology* 14.1 (2018), e1005933.

Experimental Setup & Input Data

We use data obtained from ten experiments. In each experiment two fish swam for one hour in a 30 cm $\times$ 30 cm tank, enclosed by four straight walls. The motion was captured by a camera and was tagged by a video tracking software. An example for the experimental situation can be seen in figure 2. The raw data consists of the positions of both fish, their orientation relative to the x-axis, and time. This coordinate system is used throughout the following sections.

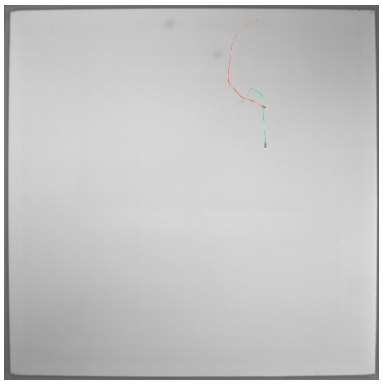

Figure 2: Example frame of the tracked video. Shown is the 30 cm $\times$ 30 cm large arena with two fish. The red and green lines show the fish trajectories that were estimated by the tracking software.

We mark frames that caused problems for the tracking system, for example when the identity of both fish was confused, as invalid. The video data has roughly 100 frames per second, the time between frames is slightly irregular. We convert the video to a constant frame rate by iterating over the frames and inserting frames between them if the time between frames differs by more than 0.005 s from a constant frame rate. These inserted frames are marked invalid as well. Invalid frames are later discarded.

The velocity is computed by finite differences between time-points. We then smooth the velocity by a Savitzky-Golay filter with a window size of 15 frames and a polynomial order of three. This filter uses a local polynomial approximation. We also use it to compute the derivative of the velocity. If fish move slower than 0.5 cm/s for 4 s, they are marked as stopped. Frames in that area are marked as invalid. We do this because we are only interested in swimming behavior.

Discrete Motion

As mentioned in the introduction, we approximate the fish-motion by a piecewise-linear trajectory. After choosing a heading direction, the fish kicks off and moves in an approximately straight line. We need to segment the fish motion into kicks. The segmentation for each fish is computed separately, the resulting kicks are concatenated.

We use the zero crossing of the smoothed acceleration to mark instances where the fish changes from accelerating to gliding phase or vice versa (figure 3). One acceleration and gliding phase is then combined to form a kick. If an acceleration or gliding phase is shorter than a threshold of 0.08 s the phase is merged with the previous one. We remove a kick if it contains a timestep which was marked as invalid. This segmentation procedure results in ca. 148000 kicks. We use the first 80% of each experiment for training, the remaining 20% are used as testing set.

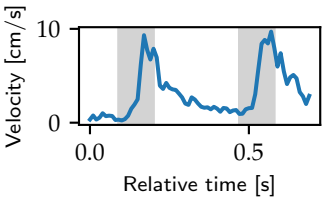

Figure 3: Example result of the segmentation procedure. Shown is (non-smoothed) velocity. Areas shaded in gray were marked as acceleration, others as gliding.

We then extract the heading change, length and duration for each kick. Additionally, we extract the positions and angles of both fish and their distance and angles towards the wall. We extract this data for the kick and timesteps before it. For our models we use time steps of 0 s, 0.05 s, ..., 0.35 s, which goes back roughly one mean kick duration (figure 4).

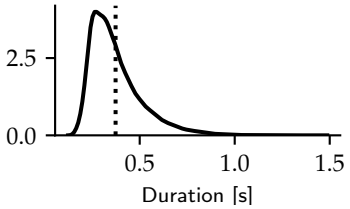

Figure 4: Kernel density estimate of training kick duration. Dotted line is the mean duration at 0.37 s.

This paper is concerned with the prediction of the relative change in orientation between the start and end of each kick (figure 1). The social models also predict the length of each kick, i.e. the total distance traveled from the start to the end of the kick. We do not model the duration of the kicks, we rather draw them from a Gaussian mixture kernel density estimate fit to the experimental data.

Wall Force

We are now ready to discuss the modeling of the wall influence. We follow the force-based modeling approach of Calovi et al. and describe the influence of a wall on the heading change $\delta\phi_w$ as

$$\delta\phi_w(r_w, \theta_w) = f(r_w)O_w(\theta_w),$$

where r_w corresponds to the fish and θ_w is its relative angle towards the wall[2]. The relative angle is computed by subtracting the orientation of the fish from the orientation of the wall. We split the wall influence into an exponentially decaying force term f_w and an odd angular-response function O_w

$$f(r_w) = \exp\left(-(r_w/l_w)^2\right),$$
$$O(\theta_w) = c\left(a_1\sin(\theta_w) + a_2\sin(2\theta_w)\right)\left(1 + b_1\cos(\theta_w) + b_2\cos(2\theta_w)\right),$$

where l_w, a_1, a_2, b_1, b_2 and c are parameters. Note that we use a different series expansion for the odd angular function O_w.[3] Finally,

[2] Daniel S Calovi et al. "Disentangling and modeling interactions in fish with burst-and-coast swimming reveal distinct alignment and attraction behaviors". In: *PLoS computational biology* 14.1 (2018), e1005933.

[3] We found that the functional form of Calovi et al. does not work as well in our case. The reason for this could be that they consider both a different species and a circular arena. We instead use a more generalized series expansion.

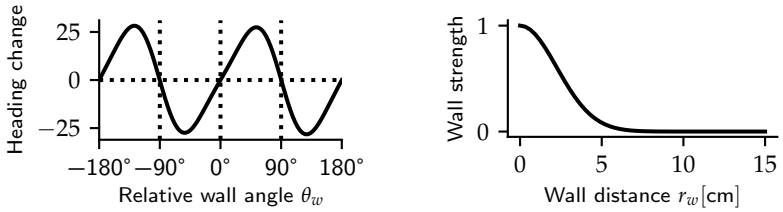

Figure 5: Our wall fit. The angular response function $O(\theta_w)$ is shown on the left, the exponentially decaying wall strength $f(r_w)$ on the right. Positions of fixed points are shown by dotted lines.

we consider the closest two walls and sum over their influences

$$\delta\phi_w^{\text{total}}(\boldsymbol{r_w}, \boldsymbol{\theta_w}) = \sum_{i \in 2 \text{ closest walls}} \delta\phi_w(r_w^i, \theta_w^i).$$

The parameters are fit by minimizing the mean-squared error of heading change prediction with the Levenberg-Marquardt algorithm using the implementation of **scipy**.[4] Our proposed functional forms result in a rapidly decaying wall influence (figure 5). For the sake of brevity, we do not investigate the wall model further as the focus of this work is modeling the social interactions.

[4] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. [accessed 2018-05-22]. 2001–. URL: `http://www.scipy.org/`.

Receptive Fields

We now explain the input features of our social models. From here on, we restrict our dataset to kicks where the wall influence is negligible. Practically, we drop all kicks where the value of $f_w(r_w)$ is larger than 0.1 for any wall. This corresponds to a wall distance of ca. 4.8 cm. The number of kicks in the training set is reduced by this to ca. 19400. Note that we discard a majority of our data because juvenile zebrafish mostly swim near the boundaries of the arena.

Our social models use a receptive field (RF) as their input.[5] We construct this RF by rotating the coordinate system such that the fish we are considering is parallel to the x-axis and looks to the right. This corresponds to a heading of 0° in our coordinate system. We then shift the coordinate system such that our fish is at the center (figure 6). This is done for all timesteps.

We now shift our focus to the spatial discretization of all receptive fields (for all timesteps). The standard approach is to divide the space into a regular grid of bins with equal size. This approach has the disadvantage that some bins contain most of the kicks while others are nearly empty. One solution to lessen the impact of this problem is to introduce a cut-off range, after which the social forces are assumed to be negligible.

We do not follow this approach but rather use a data-driven binning scheme where we try to create bins that have a more balanced occupancy. This is not the only criterion for choosing a discretization approach. For interpretability, it is useful to be able to clearly distinguish situations where the other fish is behind or in front of the focal fish and whether it is on its left or right side. Our proposed algorithm to compute the bin edges is shown in algorithm 1.

[5] Roy Harpaz, Gašper Tkačik, and Elad Schneidman. "Discrete modes of social information processing predict individual behavior of fish in a group". In: *Proceedings of the National Academy of Sciences* (2017), p. 201703817.

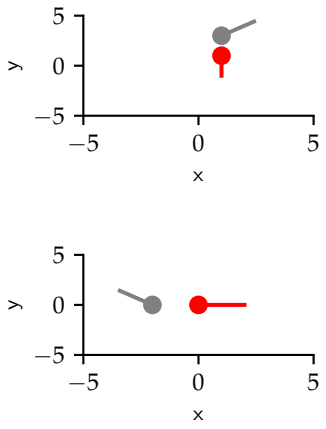

Figure 6: The receptive field transforms the situation depicted in the top image to the one in the bottom image. Note that the focal fish (red) looks to the right and is at the origin.

It first divides the x-axis (with symmetry around the origin) into bins that each have a roughly equal number of fish and then subdivides each of these bins in the same manner. We only show how to compute the bins for the training set, the bins for the testing set can be computed in the same manner but without recomputing the edges.

Algorithm 1: Data-driven Spatial Binning

Helper functions:

EDGES($\boldsymbol{x}, N$) computes the edges of a histogram s.t. all N bins have the same number of elements. If this is not possible, the first few bins have a size that is 1 larger.

BIN(edges, $\boldsymbol{x}$) computes the histogram bin from the bin edges with open boundaries on both sides.

function EDGES-1D($\boldsymbol{x}, N$)
  edges$^-$ $\leftarrow$ EDGES($\boldsymbol{x}[\boldsymbol{x} \leq 0], N$)
  edges$^+$ $\leftarrow$ EDGES($\boldsymbol{x}[\boldsymbol{x} > 0], N$)
  return CONCATENATE(edges$^-$, 0, edges$^+$)

Input: Positions of focal fish $(\boldsymbol{x}, \boldsymbol{y})$ and number of bins per axis N
bins $\leftarrow$ ARRAY(size = LEN($\boldsymbol{x}$))
edges-x $\leftarrow$ EDGES-1D($\boldsymbol{x}, N$) ▷ First bin x-axis
bins-x $\leftarrow$ BIN(edges-x, $\boldsymbol{x}, N$)
edges-y $\leftarrow$ ARRAY(size $= N$)
for bin-x $\in$ bins-x **do** ▷ Bin y-axis for each x-bin separately
  idx $\leftarrow$ (bins-x $=$ bin)
  $\boldsymbol{y}^{\text{cur}} \leftarrow \boldsymbol{y}$[idx]
  edges-y[bin-x] $\leftarrow$ EDGES-1D($\boldsymbol{y}^{\text{cur}}$)
  bins[idx] $\leftarrow$ bin-x $\cdot\, N +$ edges-y[bin-x]
return edges-x, edges-y, bins

A regular grid with 64 bins covering all seen positions (i.e. the minimum and maximum x and y coordinates are included in the grid) is used as a baseline. This grid can be created by binning the space symmetrically (around 0) for the x and y coordinates separately, e.g. the grid spans the range $(-\max(\text{abs}(x)) \ldots \max(\text{abs}(x)))$ for the x-axis. The bins are chosen such that they have equal size. Using this technique on our dataset results in a mean number of fish per bin of roughly 2428 ± 8483 (mean $\pm$ std.) for the training set and 555 ± 1926 for the testing set. In fact, 12 and 13 of the bins actually do not contain any fish at all for the training and testing set respectively! In contrast to that the bins of our proposed data-driven binning scheme contain 2428 ± 212 fish with zero empty bins for the training set. For the testing dataset the bins contain 555 ± 162 fish, again with zero empty bins (figure 7).

This categorical feature is converted using one-hot-encoding to a feature vector. We additionally use the standardized relative trajectory of the other fish multiplied with the aforementioned spatial feature[6]. This leads to $3 \times 64 = 192$ features for the discretization with 64 bins.

[6] This can be generalized to multiple fish by interpreting the features as number of fish per bin and mean relative trajectory per bin.

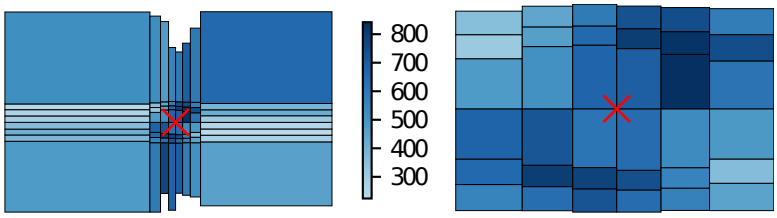

Figure 7: Number of data points in each bin obtained from data-driven binning scheme, evaluated on testing set. The color corresponds to the number of fish in per bin. The red cross marks the origin and thus the position of our focal fish. The left image shows all bins, the right image shows the 6x6 neighborhood of the origin. This plot is set in the local RF coordinate system, the focal fish looks to the right.

Social Models

We are now ready to model the social behavior of zebrafish. Note that the approaches developed here can all be easily extended to larger fish groups. Our target variable for all models is the relative kick trajectory in the local RF coordinate system described in the previous section. We thus model heading change and distance jointly.[7] We start with a simple linear model. This model is then refined twice by adding memory and non-linear effects.

[7] This approach has two advantages. Firstly, we avoid dealing with cyclic data (as angles are). Secondly, all target variables (i.e. vector components) have the same physical dimension and unit which makes the loss functions easier to interpret.

Linear receptive field model

The linear model without memory for the prediction of a component i of our 2-dimensional target vector $\boldsymbol{y}$ can be written as

$$\hat{\boldsymbol{y}}^i = \boldsymbol{X_0}\boldsymbol{\beta}_0^i + \text{intercept} + \varepsilon$$
$$\varepsilon \sim \mathcal{N}\left(0, \sigma^2\right),$$

where $\boldsymbol{X_0}$ is the design matrix at kick-off time and $\boldsymbol{\beta}_0^i$ is a vector of learned weights. The error ε follows the normal distribution $\mathcal{N}(0, \sigma^2)$ with mean zero and a residual variance of σ^2. This modeling situation is depicted in figure 8. We will drop the superscript for the component and describe the computation for one vector component in the following discussion of the linear models. The other component can be computed in the same manner. Furthermore all linear models discussed in this work are trained using a L_2 regularized mean squared error (MSE) loss. This type of model is known in statistics as ridge regression.

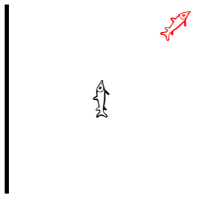

Figure 8: Situation considered for the models without memory: We model the trajectory vector of the red fish given the current position and angle of the other fish.

Time dependence

We now add a time-dependence to our model. To do this, we use all extracted timesteps leading to a kick (figure 9).

We consider two models here. For the first one we simply concatenate all features of the design matrix $\boldsymbol{X_t}$ for all timesteps t. This results in a very large number of parameters, consisting of one set of weights $\boldsymbol{\beta_t}$ per timestep.

Alternatively, we can use the same spatial weights for each timestep (i.e. $\forall t_1, t_2 : \boldsymbol{\beta}_{t_1} = \boldsymbol{\beta}_{t_2}$). We introduce a parameter c_t with one element per timestep and then use $c_t\boldsymbol{\beta}$ as our parameters. For consistency, we normalize all c_t such that they are positive and

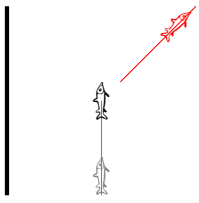

Figure 9: Situation considered for the models with memory: We model the trajectory vector of the red fish given the trajectory of the other fish. The trajectory of the red fish is implicitly encoded in the receptive field as we consider the position of the other fish in the local coordinate system at each point of red's trajectory.

sum to unity using the *softmax* non-linearity

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}, \tag{1}$$

where x_i is one component of the output vector.

The prediction is then given by

$$\hat{\boldsymbol{y}} = \sum_t c_t \boldsymbol{X_t} \boldsymbol{\beta} + \text{ intercept} + \varepsilon.$$

Mixture Density Networks: Bi-modal distributions & non-linearity

As a final model we present a neural network that predicts a full distribution for $(\boldsymbol{y}|\boldsymbol{X})$, which allows us to draw samples and estimate the predictive uncertainty. We start by describing encoders, which transform the input features to a hidden state, and decoders which then transform the hidden state into output values. The complete model is then an arbitrary combination of the encoders and decoders (figure 10).

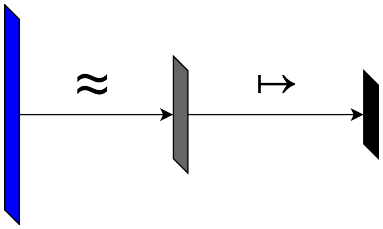

Figure 10: General encoder-decoder architecture. Symbols ≈ and ↦ correspond to encoding and decoding. Blue is input, gray hidden and black output.

We describe two simple encoders (figure 11):

- The simplest (non-linear) choice is a multilayer-perceptron (MLP). In our case we use two layers, each consisting of a linear transformation, followed by the *tanh* non-linearity and a dropout probability of 50%.[8] The input feature for this encoder is the receptive field at kick-off time.

- As a temporal model we consider a standard recurrent neural network (RNN) which has an linear transformation of the input and a recurrent hidden-to-hidden connection. For the hidden-to-hidden connection we apply recurrent dropout with a probability of 50%.[9] This means that we sample a dropout mask per example and apply this to the hidden-to-hidden connections for all sequence steps instead of sampling a separate mask for each sequence step. We denote this dropout operator as $d(\boldsymbol{x})$. During evaluation this operator scales the weights by a factor of 0.5. These two layers are added and transformed with the *tanh* non-linearity

  $$\boldsymbol{h_t}(\boldsymbol{X}) = \tanh\left(\boldsymbol{b} + \boldsymbol{w_{ih}}\boldsymbol{X_t} + \boldsymbol{w_{hh}} d(\boldsymbol{h^{i-1}})\right),$$

  where $\boldsymbol{w_{ih}}$ and $\boldsymbol{w_{hh}}$ are the weights for the input-to-hidden and hidden-to-hidden layers respectively. In this equation $\boldsymbol{X_t}$ is a tensor containing the input features for the entire batch at timestep t. The timestep that is the furthest away from kick-off time is denoted by $t = 1$. The parameter $\boldsymbol{b}$ corresponds to a vector of learned biases. We learn the initial hidden state $\boldsymbol{h_0}$ instead of using a zero-initialized state as it improved the speed of convergence. The last hidden state with $t = t_{\max}$ is used as an encoded representation of the data.

  The input features for this encoder is the complete sequence of receptive fields leading to a kick.[10]

[8] Nitish Srivastava et al. "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[9] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. "Recurrent dropout without memory loss". In: *arXiv preprint arXiv:1603.05118* (2016).

[10] We use a fixed input sequence length for two reasons: Firstly, this allows us to compare the time-dependent linear models and RNN-models directly. Secondly, we avoid the assumption that a fish only considers its surroundings in the time span between two kicks. Note that we assume that a kick is independent of the other kicks.

We did not observe significant better results using the rectifier linear unit activation function, in fact, it lead to divergence for the RNN model. Using more neurons in the hidden layer also did not achieve better results. For scenarios with less sparse feature vectors such as ones obtained with larger fish groups, a more complex representation might be beneficial. We hope that the combination of the simplicity of the models coupled with the dropout regularization leads to a highly representative compressed representations in the encoded vector. Note that both models (disregarding the learned initial RNN-state) have the same number of parameters.

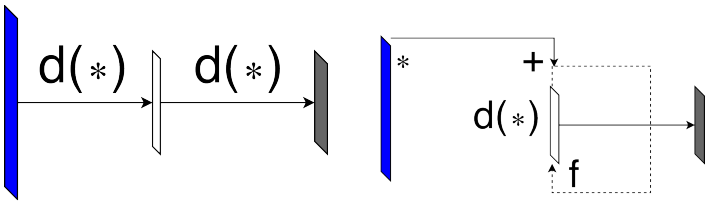

Figure 11: **Left**: Architecture of MLP encoder. Symbol $d(*)$ corresponds to linear layer, followed by tanh and dropout.

Right: Architecture of RNN encoder. Symbol $*$ is linear layer, $d(\,)$ is recurrent dropout, $+$ is summation and f is *tanh*.

Blue is input and gray is hidden state.

For the encoder we follow the ideas of *mixture density networks* (MDN) but use multivariate Gaussians with a non-diagonal covariance as mixture components.[11] To do this, we write our prediction as

$$p\left(\boldsymbol{y}|\boldsymbol{\beta}\right) = \sum_{i}^{n} \kappa_i \mathcal{N}\left(\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i}\right),$$

where $\kappa_i, \boldsymbol{\mu_i}, \boldsymbol{\Sigma_i}$ are the parameters (mixing coefficient, mean and covariance) for a mixture of Gaussians with n components. The function $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma_i})$ is the probability density of a multivariate normal distribution

These parameters are predicted for each target separately by our neural network. We now derive necessary conditions that have to hold for the parameters and describe how our network output satisfies them. Note that we can write the Cholesky decomposition[12,13] of the covariance matrix $\boldsymbol{\Sigma_i}$ as

$$\boldsymbol{\Sigma_i} = \boldsymbol{L_i}\boldsymbol{L_i^T} = \begin{bmatrix} a_i^2 & a_i b_i \\ a_i b_i & b_i^2 + c_i^2 \end{bmatrix}, \quad \text{with } \boldsymbol{L} = \begin{bmatrix} a_i & 0 \\ b_i & c_i \end{bmatrix}.$$

We then have to fulfill the following constraints:

- The diagonals of all $\boldsymbol{L}_i$ need to be larger than zero. To do this we use the *exponential function* as our non-linearity. We add a small $\varepsilon = \sqrt{10^{-6}}$ for numerical stability.[14] Using the fact that a matrix has an invertible Cholesky decomposition with positive diagonal elements iff. it is positive definite and Hermitian, the resulting matrix $\boldsymbol{\Sigma_i}$ is a valid non-singular covariance matrix.
- The mixing coefficients κ_i need to be positive and have to sum to one. This is achieved by using the *softmax* (equation 1) non-linearity.

All other parameters can have arbitrary values. We predict the Gaussian mixture parameters each by computing a linear transfor-

[11] Christopher M Bishop. *Mixture density networks*. Technical Report. 1994.

[12] William H Press et al. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[13] This decomposition always exists for invertible covariance matrices as they are by definition positive-semi-definite and symmetric.

[14] This regularization reduces the condition number of $\boldsymbol{\Sigma_i}$ by increasing the value of its minimal eigenvalue. Additionally it ensures that the variance is strictly positive and thus avoids degenerate distributions.

mation of the hidden state of the encoder followed by the appropriate non-linearity (figure 12). Using a full covariance matrix instead of a diagonal led to no improvement for our data. Because a diagonal covariance matrix is conceptually simpler, we set b_i to zero for all outputs.[15]

We minimize the negative-log-likelihood (NLL), which is given for a vector of examples $\boldsymbol{y}$ by

$$\mathcal{L}(\boldsymbol{y}; \boldsymbol{\kappa}, \boldsymbol{\Sigma}, \boldsymbol{\mu}) = -\log\left(\sum_i \exp\left(\log\left(\boldsymbol{\kappa_i}\mathcal{N}(\boldsymbol{y}; \boldsymbol{\mu_i}, \boldsymbol{\Sigma_i})\right)\right)\right) \tag{2}$$

with

$$\log\left(\mathcal{N}(\boldsymbol{y}; \boldsymbol{\mu_i}, \boldsymbol{\Sigma_i})\right) = -\log\left(\sqrt{\pi_i^2|\boldsymbol{\Sigma_i}|}\right) - 0.5\left((\boldsymbol{y}-\boldsymbol{\mu_i})^T\boldsymbol{\Sigma_i}^{-1}(\boldsymbol{y}-\boldsymbol{\mu_i})\right),$$

to train our models. In this equation, $|\boldsymbol{\Sigma_i}|$ denotes the determinant of $\boldsymbol{\Sigma_i}$[16]. While this optimization problem is mathematically correct, it is susceptible to underflow. To rectify this we follow common advice and apply the *log-sum-exp-trick* which is given by the identity

$$\log\left(\sum_i \exp(x_i)\right) = \max_i(x_i) + \log\left(\sum_i \exp(x_i - \max_i(x_i))\right)$$

to compute the sum in equation 2.[17]

[15] This does not limit the capabilities of our network because a mixture of Gaussians with diagonal covariance matrices is able to approximate all non-degenerate distributions, given a sufficient number of components.

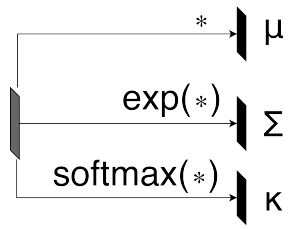

Figure 12: Architecture of MDN decoder. Symbol ∗ is linear layer. Gray is hidden state and black are the outputs.

[16] We compute the determinant directly from the Cholesky matrix $\boldsymbol{L_i}$. Furthermore we do not invert the covariance matrix but rather solve the equivalent linear system with forward-backward substitution using $\boldsymbol{L_i}$ again. For the diagonal case, both determinant and inverse are trivial to compute and numerically stable.

[17] Axel Brando. "Mixture Density Networks (MDN) for distribution and uncertainty estimation". MA thesis. 2017.

Implementation & Training Details

We used the data-driven discretization with 64 bins for all experiments.

The linear model without memory and the one with concatenated features were trained with the *RidgeCV* estimator from **scikit-learn**.[18] This implementation uses generalized-cross-validation over the training set to determine the best parameter for the regularization strength.

The linear model with static spatial weights and all neural network models were implemented in **PyTorch**. We set the number of components for the MDN-networks to 5. We use stochastic gradient descent with an initial learning rate of 0.1, a weight decay of 0.001 and a Nesterov momentum of 0.9 to optimize our networks. Encoders and decoders are jointly trained. We clip gradients such that they have a maximum absolute value of 100 to increase the stability of training. For initialization, weights are drawn from a normal distribution with mean 0 and standard deviation of 0.002 and all biases are set to zero. We train all models for 1000 epochs with a batch size of 128. The learning rate is multiplied by 0.99 after each epoch.

We implemented a program to output an animation that shows two fish swimming, each controlled by one of our implemented algorithm. The simulation uses periodic boundary condition to show the effect of the social models without the influence of walls.

[18] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

A possible use case for these models is steering a virtual fish in a virtual reality environment.[19] In this case periodic boundary conditions are possible but not meaningful. To rectify this, one can predict the heading change with our social models and then rotate the resulting trajectory by the prediction of the wall model. Alternatively it is possible to implement a virtual force that pushes our fish away from the wall. This is of course not physical but would allow us to observe the social behavior.

[19] John R Stowers et al. "Virtual reality for freely moving animals". In: *Nature methods* 14.10 (2017), p. 995.

Evaluation

We start by comparing the angle prediction from the wall model against the baseline of always predicting the mean angle. Evaluating these predictions with the MSE-loss or similar metrics is not meaningful because they do not take the cyclical nature of angles into account. For example, a prediction of 360° for an angle of 0° is perfect but incurs a large MSE-loss. We rectify this by using definitions from directional statistics.[20] An alternative approach would be considering the representation of the angles as unit vectors and computing their MSE. Computing the error on angles follows the optimization problem used to optimize the models more closely and is thus more appropriate. Following Batschelet, we can compute the mean of a vector of angles $\boldsymbol{\alpha}$ with

[20] Edward Batschelet. "Circular statistics in biology." In: *ACADEMIC PRESS, 111 FIFTH AVE., NEW YORK, NY 10003, 1981, 388* (1981).

$$\text{Mean-Angle}(\boldsymbol{\alpha}) = \arctan2\left(\sum_i \sin(\alpha_i), \sum_i \cos(\alpha_i)\right).$$

Due to symmetry, the mean angle of ca. 0.007 is nearly zero. One possible way of defining an error over angles is

$$\text{Angle-Error}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \text{Mean}\left(\arccos\left(\cos(\boldsymbol{\alpha} - \boldsymbol{\beta})\right)\right). \tag{3}$$

This error is zero if both angles are identical. Our wall model achieves a lower error than predicting the mean for all examples on both training and testing data (table 1).

Model	Angle-Train	Angle-Test
Mean	0.398	0.375
Calovi	**0.379**	**0.350**

Table 1: Results for angular model. Angle-{Train/Test} refers to equation 3. Best results are bold.

We compare the models using the MSE and the NLL on the training and testing set. As mentioned earlier, both datasets are now restricted to situations with negligible wall-force. Even though the linear models are not trained by optimizing the NLL, it is appropriate to use this metric for evaluation. The reason for this is that the regularized MSE loss function used for ridge regression can be derived by computing the NLL under the assumption that the likelihood is Gaussian and that we have a Gaussian prior on the parameters of the model.[21] This comparison captures the whole predictive distribution and not only the mean prediction. In this way we take the uncertainty of the predictions into account. Note that it is not straightforward to use more complicated methods such as log-likelihood ratios or the Akaike information criterion because we cannot estimate the number of degrees of freedoms for penalized recurrent neural network models.

[21] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. Springer series in statistics New York, 2001.

The best NLL-error was achieved by the RNN-MDN and MLP-MDN models on the training and testing set respectively (table 2). Interestingly, the RNN-MDN methods had the best testing MSE, even though we include models that were explicitly trained with the MSE-loss. All linear models had larger errors for all considered metrics but did not achieve a significantly worse MSE.

Model	NLL-train	NLL-test	MSE-train	MSE-test
Baseline (Train-Mean)	2.25	2.01	0.558	0.423
MLP-MDN	1.63	**1.60**	0.448	0.373
RNN-MDN	**1.43**	1.69	0.436	**0.371**
MLP-MSE	2.04	1.87	0.452	0.375
RNN-MSE	2.00	1.86	**0.432**	0.377
Linear (w/o time)	2.04	1.87	0.451	0.376
Linear (time conc.)	2.00	1.86	0.432	0.373
Linear (static spatial)	2.04	1.86	0.451	0.374

Table 2: Results for all social models on training and testing datasets. Linear models are linear without time, linear with entire sequence concatenated as features and linear with static spatial weights. The baseline model is always predicting the training mean trajectory. Best results are bold.

Conclusion & Future Work

In this work we presented a full modeling pipeline for discrete social models with temporal and non-linear effects. This pipeline starts by massaging the data into an appropriate format, using a wall model to filter timesteps with a strong wall influence, setting up a data-driven spatially discrete receptive field and finished with complex models implemented in a state-of-the-art neural network framework.

Even though extending the models with a temporal and non-linear components did not show significant improvements for our dataset, we think that our proposed methods are promising:

- The data-driven spatial discretization results in a more equal distribution of fish than a standard regular grid.
- Because the social models do not use properties of other animals directly but rather model the mean features per bin they extend trivially to larger animal groups, without changing the statistical models or the spatial discretization. This extension is harder for force-based models and usually requires extensive modifications of the core model. We believe that the models developed here would perform comparatively much better for larger group sizes than force-based models, as this would result in less sparse input vectors.
- The explicit modeling of probability distributions allows us to sample from the trajectory distribution conditioned on the social surroundings and is able to predict bi-modal distributions. This is biologically relevant, since we often observe non-Gaussian distributions in experiments. Force based models do not capture

these distributions. Additionally it allows us to quantify the uncertainty. This results in a smaller NLL.

These results can be used directly for large-scale modeling of fish-swarms. We recommend the collection of more data or increasing the data-efficiency of the models. This could be achieved by:

- Running more or longer experiments.
- Lessening the impact of the restriction to areas with negligible wall influence. This could be accomplished by using fish that do not swim close to the wall at most times, using different arenas or by using a model that incorporates both wall and social influences directly. The latter approach could also consider non-linear interactions between both kind of environmental forces.
- Using a continuous model instead of performing a segmentation into kicks. This could be problematic for juvenile zebrafish as naive RNN models might not be suited for the modeling of different states of motion.

References

Batschelet, Edward. "Circular statistics in biology." In: *ACADEMIC PRESS, 111 FIFTH AVE., NEW YORK, NY 10003, 1981, 388* (1981).

Bishop, Christopher M. *Mixture density networks*. Technical Report. 1994.

Brando, Axel. "Mixture Density Networks (MDN) for distribution and uncertainty estimation". MA thesis. 2017.

Calovi, Daniel S et al. "Disentangling and modeling interactions in fish with burst-and-coast swimming reveal distinct alignment and attraction behaviors". In: *PLoS computational biology* 14.1 (2018), e1005933.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. Springer series in statistics New York, 2001.

Harpaz, Roy, Gašper Tkačik, and Elad Schneidman. "Discrete modes of social information processing predict individual behavior of fish in a group". In: *Proceedings of the National Academy of Sciences* (2017), p. 201703817.

Jones, Eric, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. [accessed 2018-05-22]. 2001–. URL: `http://www.scipy.org/`.

Pedregosa, F. et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

Press, William H et al. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

Semeniuta, Stanislau, Aliaksei Severyn, and Erhardt Barth. "Recurrent dropout without memory loss". In: *arXiv preprint arXiv:1603.05118* (2016).

Srivastava, Nitish et al. "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

Stowers, John R et al. "Virtual reality for freely moving animals". In: *Nature methods* 14.10 (2017), p. 995.