Representing Behavior with Recurrent Neural Networks

Ishan Saran

April 7, 2020

1 Introduction

1.1 Defining the problem

With the advent of novel technologies, there has been a lot of work in the field of neuroscience to assemble and analyze large swaths of neural data [1–5]. The connectome of *Caenorhabditis elegans* has been fully realized [6] and that of *Drosophila melanogaster* is soon on its way [7], with models being built for other organisms as well [8, 9]. However, a rich description of the anatomy and physiology of the brain does not immediately translate to an understanding of the output of the brain. Namely, if we consider the input to be the set of neuronal firing patterns and environment an organism is placed in, the output would be something along the lines of behavior.

That a well-defined, precise, mathematically rich description of behavior is a necessary precursor to understanding a mapping from neural circuitry to behavior has been articulated by previous authors [10–12]. The problem lies in the fact that there is no single precise, meaningful description of behavior. Behavior operates on a variety of both time and length scales, it is context-dependent and the set external and internal stimuli which modulate behavior are not given equal weighting. They are hierarchically ordered and changing over time. What numbers can capture the essence of behavior across these scales while still being comprehensible to us? This is an open question, one which will require a bit of work on the theoretical level. As a result of this, the field as a whole has seen a continuum of approaches.

1.2 Traditional approaches to quantifying behavior

Traditionally, neuroscientists have studied animals in the lab setting. The classic picture that comes to mind is the rat-in-a-maze experiment, where an animal is put in an environment outside of its natural habitat and observed under a set of contrived conditions. While this approach promises reproducibility and interpretable results, the quantification of the behavior is defined into the experiment, it didn't fall out of observation of the animal. Moreover, the scope of the behavior is limited and there is no guarantee at all that the behavior an

animal is displaying is a part of its natural repertoire of actions. What if an animal develops a set of actions just to accomplish the task?

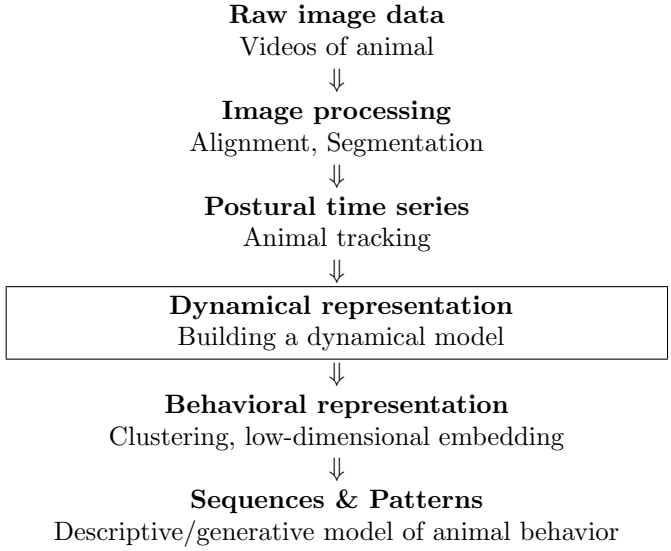
Even if an animal is studied in its natural habitat, what numbers of the animal would provide a meaningful representation of its behavior? If the goal is to study a specific behavior of an animal, such as bird singing, then sometimes a specialized set of numbers can describe the narrow repertoire of behavior, such as song pitch, and this has certainly been done [13, 14]. However, with other behaviors, such as aggression, the definition any researcher chooses, or the set of actions which they count as aggressive, will have naturally imposed researcher biases. This is true even if they wish to describe the entire repertoire of behavior. If a researcher wants to minimize bias, they must take themselves out of the analysis process. In other words, rather than themselves defining the behavior, it should fall out naturally from the behavioral analysis.

1.3 Stereotyped behaviors and modern approaches

Similar to how a phase space describes the set of all possible states of a system, we can define the notion of posture space as the set of all postures an animal is capable of being in. While this set of states is quite large, theoretically limited only by the biomechanical limits of its morphology, animals tend to occupy only a fraction of the total postural space. The movements that it spends a lot of time performing are known as "stereotyped behaviors" and are well-documented in the scientific literature. For example, head-bobbing in pigeons [15], cribbing in horses [16], and wing grooming in bees [17] are all well studied phenomena. Stereotyped behaviors offer a decomposable set of actions reproducible in a lab setting. It is imperative that whatever behavioral analysis one undertakes should not predefine the notion of stereotypy into the experiment, however. Rather, in understanding full repertoires of behavior, one would want the stereotyped behaviors to fall out naturally from the analysis. This ensures that stereotyped behaviors are a manner in which behaviors fundamentally decompose as opposed to merely being a convenient way in which they are represented.

Greg Stephens and colleagues were the first to conceptualize behavior as a trajectory through posture space with incredible success in *C. elegans*, deriving equations of motions for the worms' dynamics [18]. Thinking of behavior in this way is beneficial because behavior is intrinsically a dynamical variable and has led to significant discoveries into the nuances of animal behavior [19, 20]. Others have identified the internal states which drive social behavior, for example, with a mix of hidden Markov Models (HMMs) and generalized linear models (GLMs) [21].

If one wishes to extract stereotyped behaviors from data, they generally have to follow this pipeline [11]:

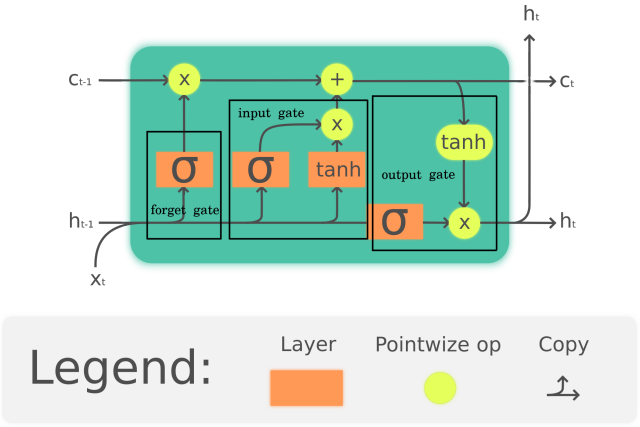


Traditionally, quite a bit of effort has been needed to take the raw image data to a postural time series, primarily because the tracking of individual animal body parts has been a difficult computer vision problem. Recently developed deep learning methods [22–24], however, have allowed high-throughput image analysis. My research takes the output from one of these tracking software algorithms applied to videos of freely moving fruit flies and works on the dynamical representation step of the pipeline. In my thesis, I utilize yet another way to quantify behavior: using recurrent neural networks to build a representation of the underlying neural dynamics.

1.4 Recurrent Neural Networks

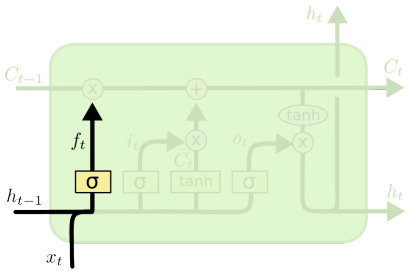
In a traditional feed-forward neural network, the information is solely propagated forward; i.e. the input layer has unidirectional connections to the first layer which has unidirectional connections to the second layer and so on and so forth until the output layer. Each layer corresponds to a vector, and the connections between the layers are contained within a weight matrix. This weight matrix contains the weights which multiply the incoming vectors as the information moves from layer to layer. Since information solely propagates forward, the input vectors are independent of one another. RNNs, on the other hand, have the ability to remember previous inputs. By introducing loops into the architecture, the network can learn temporal dynamics. This makes the RNN architecture particularly adept at handling time series and finding correlations across various time scales in the data. In terms of network architecture, I used a Long Short-Term Memory (LSTM) network, a popular RNN used to capture dynamics of the input vector both on the long- and short-term scale. LSTM networks have an architecture similar to traditional feed-forward networks with

two important specifications: (1) they contain feedback connections, such that the output of one cell is fed into the input of the next cell and (2) they have the ability to "remember" or "forget" the previous state depending on a set of rules. Below is a diagram of an individual LSTM cell (courtesy of Wikipedia, Creative Commons License).



There are three gates which control the flow of information in an LSTM cell and two avenues through which the information can go through. The two sequential sets of weights are $c(t)$, the running cell state and $h(t)$, the hidden cell state, which are collectively referred to as the hidden states of the RNN. The running cell state can be thought of as a weighting factor which stays relatively unchanged and is modulated by the hidden state. Both the hidden state and the running cell state are of the same length as the input time series and there is a $c(t), h(t)$ pair for every node in the network. The three gates are the *forget gate*, the *input gate*, and the *output gate*, which are all sigmoid functions [Note: the following figures of the individual gates are taken from Christopher Olah's blog post on LSTM networks titled "Understanding LSTM networks" [25]]. The sigmoid function is given by $\sigma(x) = \frac{1}{1+e^{-x}}$ and it maps x $\in \mathbb{R}$ onto the interval [0, 1]. The *forget gate* modifies the previous cell state based on the following equation:

$$f_t = \sigma(U_f h_{t-1} + W_f x_t + b_f)$$

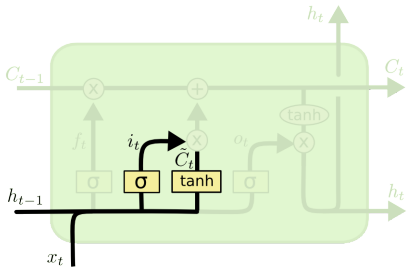


where W_f and U_f are the weight matrices associated with the forget gate, h_{t-1} is the previous hidden state, x_t] is the current input time series, and b_f is the constant term (vector). If the output of the forget gate is 1, the previous cell state is completely remembered (the values are retained)

and if the output is 0, the previous
cell state is completely forgotten (reset to 0).

The *input gate* selectively adds a value to the running cell state based on the following equation:

$$i_t = \sigma(U_i h_{t-1} + W_i x_t + b_i)$$

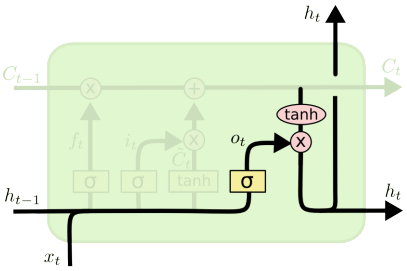


where W_i and U_i are the weight matrices associated with the input gate. If the output of the input gate is 1, then the new value proposed to the running cell state is added and if the output is 0, then the running cell state remains unchanged.

The output gate is one of the factors which is put into updating the hidden state for the time point, h_t, and is given by the equation:

$$o_t = \sigma(U_o h_{t-1} + W_o x_t + b_o)$$

where W_o and U_o are the weight matrix associated with the output gate. If the output of the output gate is 1, then the current cell state is passed onto the hidden state (c_t and h_t are equal) and if the output is 0, then the hidden state will be 0, in other words no information will be carried over to the next cell.



The hidden states, then, are functions of the input, output, and forget gates, as follows:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$h_t = o_t \odot \tanh(c_t)$$

where $\tilde{c}_t = \sigma(W_c x_t U_c h_{t-1} + b_c)$ and the $\odot$ operator represents an element-wise product such that the two vectors being multiplied retain the same shape. For completeness, the dimensions of the vectors are as follows [26]:

- $x_t \in \mathbb{R}^l$; input vector of size $l \times 1$
- $f_t \in \mathbb{R}^h$; forget gate activation vector of size $h \times 1$
- $i_t \in \mathbb{R}^h$; input gate activation vector of size $h \times 1$
- $o_t \in \mathbb{R}^h$; output gate activation vector of size $h \times 1$
- $c_t \in \mathbb{R}^h$; cell state vector of size $h \times 1$
- $h_t \in \mathbb{R}^h$; hidden state vector of size $h \times 1$

where l is the lookback parameter described later and h is the number of cells/neurons in the network. The dimensions of the weight matrices are:

- $W \in \mathbb{R}^{h \times l}$; weight matrix of the input connections of size $h \times l$
- $U \in \mathbb{R}^{h \times h}$; weight matrix of the recurrent connections of size $h \times h$
- $b \in \mathbb{R}^h$; bias vector of size $h \times 1$

1.5 RNNs to represent behavior

We develop a recurrent neural network (RNN) model to build a dynamical representation of the underlying forces driving the movement of *Drosophila melanogaster*. The idea is that by building an RNN with many more parameters than necessary to capture the fly's movements, what will result is a chiefly low dimensional representation embedded in a high dimensional space. Then, the structure of the resulting manifold can be visualized with dimensionality reduction techniques to analyze the clusters in the high dimensional space. Our hypothesis is that the representation that the RNN builds will be cleaner than the representation the raw postural data itself yields.

There are many reasons to believe an RNN will be able to capture the underlying dynamical system governing the fly's actions. For one, the underlying hidden states of the RNN are inherently dynamical - they vary as a function of time. Secondly, the "memory" associated with RNNs lends naturally to the fact that behaviors operate across scales - the RNN can learn connections on both short and long timescales as its architecture naturally lends to that. RNNs also act as denoising filters, reducing the jaggedness of the input time series when fed forward. Finally, and this is much more down the line, RNNs can be linked to each other sequentially to form an "encoder-decoder architecture"

where a much lower dimensional representation must be learned to translate the input to the output. A similar "encoder-decoder architecture" exists in the brain - there must be data compression of some sort from the brain to the set of movements, since the number of motor neurons and neurons in the neck are orders of magnitude fewer than those in the brain. Thus, one could hope an RNN could learn the method in which the brain encodes information as it is being sent down the bottleneck that is the neck. There is a sense in which behavior operates on many scales - the kinematics, the behaviors, and the "mood" of the animal. Another one of the benefits of linking RNNs together is each RNN could represent a different modality in which behavior operates.

2 Data

The data was taken from a data repository which contains the outputs of LEAP, LEAP Estimates Animal Pose, a tracking software to track selected points on an animal [22]. The tracking algorithm was applied to videos of freely moving fruit flies on a 2-d surface. There are 32 tracked points in total, described in Figure 1. There are a total of 30 prediction files, each with $N \sim 3.6 \times 10^5$ frames. For each of the 32 joints both x and y positions were tracked, leading to 30 (N x 2 x 32) time series arrays, on the order of 10^8 frames in total.

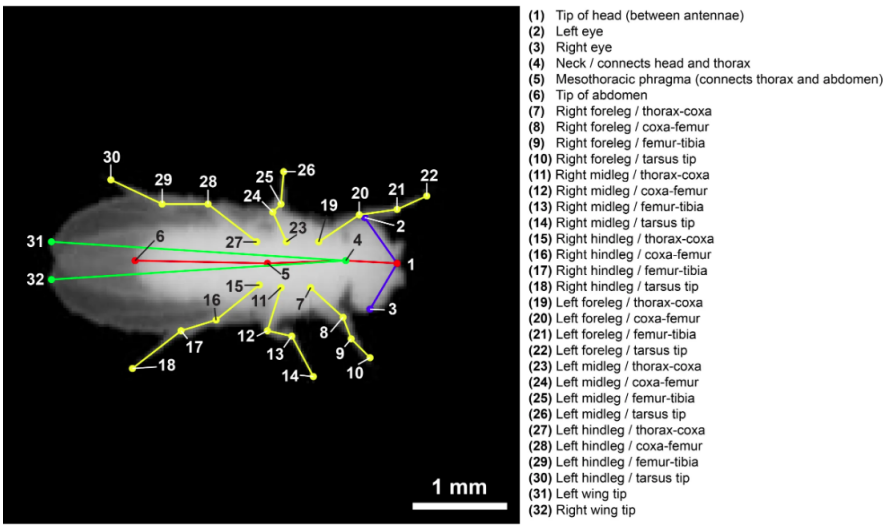


Figure 1: User-defined skeleton of fly with tracked points. The points of the fly were manually encoded for $\sim$ 1500 frames and LEAP (referred to above) generated predictions based on data. Figure courtesy of Pereira (2019) [22]

2.1 Fly recording experiments

The original videos of the freely moving fruit fly consisted of 59 male and 51 female *D. melanogaster* (Oregon-R strain) in a circular arena. The flies were restricted to a 2-D plane of movement and the arena was coated in a repellant silane compound to minimize long bouts of upside-down walking. The camera recorded at 100 Hz and the size of the frames were a 200 x 200 pixel square containing the fly. Each fly was recorded for periods of 1 hour, yielding 3.6×10^5 frames for each individual and 4×10^7 frames total. The flies were all of similar age (time after eclosion was controlled for) and the time (to control for circadian rhythm specific behaviors) and temperature (to control for temperature specific behaviors) at which the recording was conducted was held constant. For more information on the original fruit fly videos, including the arena and camera specification, see the imaging apparatus in [20].

2.2 Pose estimation with LEAP [22]

LEAP takes videos of animals as input, videos of the aforementioned freely moving fruit flies in my case, along with user annotated frames and predicts the animal pose for every frame given. LEAP uses a convolutional neural network (CNN) to generate an estimate and a confidence map for each of the joint position time series. The network architecture is simple, generalizable, and requires only a few labeled frames before the network generates reasonable estimations which can then be corrected by the user. LEAP consists of a 15-layer CNN, with a set of convolution-max pooling layers whose weights are updated during training. The strength of LEAP lies in the fact that only a relatively few number of iterations are required to obtain a set of initial values which are fairly good. Then, one can iteratively correct the generated labels with a far reduced labeling time per frame. The authors note the network is able to achieve <2.5 pixel error (2-3% of the flys body length) in 74% of the data with only ten labeled images, in some cases. The data set I used had 1,500 labeled images to train the network and <3 pixel error in 87% of the test set (N = 168 test frames, from 7 held-out flies).

For more information on the network architecture and training/benchmark methods, see [22].

3 Methods to fit data

3.1 Data pre-processing

3.1.1 Converting to joint angle time series

First, the position time series was converted to a joint angle time series. For each body point tracked except for points 4 and 5, the angle between two vectors: the mesothoracic phragma-neck vector (points 4 & 5 in Figure 1-2) and the mesothoracic-point in question vector was calculated. Converting from

positions to joint angles was beneficial because it reduced the dimensionality of the problem from tracking 64 coordinates (x & y positions of 32 points tracked) to 30 coordinates (the angles except for the mesothoracic phragma and neck, which redefined the x-axis), which led to quicker computations. Joint angles are also useful because they are independent of body orientation and body size and although the dataset generated from LEAP came with aligned data, in principle this is not required for angle calculation.

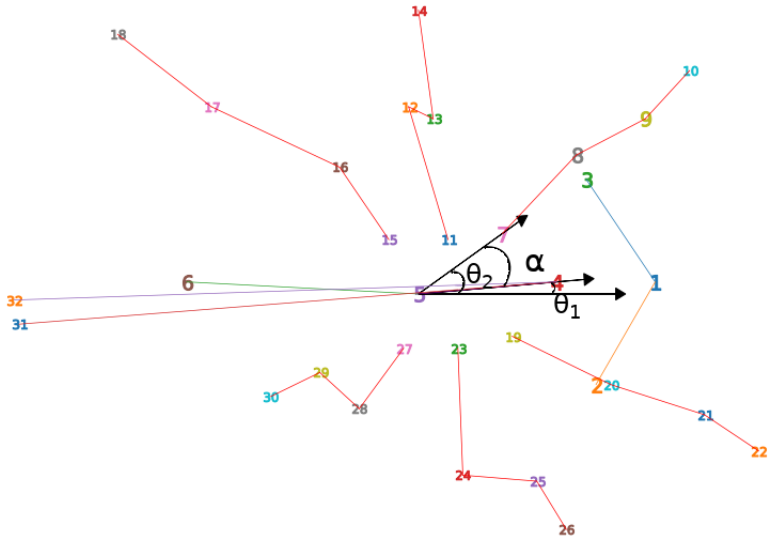


Figure 2: Example of defined angles for angle conversion calculation θ_1 corresponds to the angle of the meso-thoracic phragma neck vector to the x-axis of the image and θ_2 corresponds to the angle angle of the joint of interest to the x-axis. Then, the angle α can be determined from subtracting the two angles and applying a set of rules.

The angle calculation was done for each point (take, point 7, the right foreleg, for example) by defining two vectors: $\vec{v}_4$, the vector from the mesothoracic phragma (point 5) to the neck (point 4) and $\vec{v}_i$, the vector from the mesothoracic phragma to the point of interest ($\vec{v}_7$ for the right foreleg). Each of the angles with respect to the original coordinate plane's x-axis was calculated by taking the arctangent of the $\frac{y}{x}$ values for each of the vectors to get θ_1 and θ_2, as shown in Figure 2. Then, the angle α between the two vectors is

$$\alpha = \begin{cases} \theta_2 - \theta_1, & \text{if } \theta_2 - \theta_1 \geq 0 \\ \theta_2 - \theta_1 + 2\pi, & \text{if } \theta_2 - \theta_1 < 0 \text{ and } i \in \{8, 9, 10, 17, 18, 31\} \\ \theta_2 - \theta_1, & \text{if } \theta_2 - \theta_1 \leq 0 \\ \theta_2 - \theta_1 - 2\pi, & \text{if } \theta_2 - \theta_1 > 0 \text{ and } i \in \{6, 20, 21, 22, 29, 30, 32\} \end{cases}$$

where i corresponds to the i-th index of the vector $\vec{v}_i$, for each of the points tracked, going from 1 to 32, excluding 4 and 5 (since it wouldn't make sense to draw the vector over itself, the angle would just be 0). Arctangent has a range of $[-\pi, \pi]$ and for any vector above the x-axis would return a positive value from $[0, \pi)$ and any vector below x-axis would return a negative value from $(-\pi, 0]$. The indices in the second conditional correspond to the points on the right side of the fly and the indices in the fourth conditional correspond to the points on the left side of the fly. As seen in figure 2, though, sometimes the fly's wings would cross the x-axis defined in the original coordinate system, which would shift the value from π to $-\pi$ or vice versa rapidly, causing a discontinuity in the time series plots. This occurred for the wings, femur-tibia and tarsus tip connections (the points more lateral/further away from the body) and during grooming behaviors especially. In order to make the joint angle time series continuous, the angles which belonged to the right side of the fly were redefined to be 2π plus (or minus, if the points were on the left side of the fly) their initial value if they crossed the axis, which is what the second and fourth conditionals check for.

3.1.2 Median-filtering the data

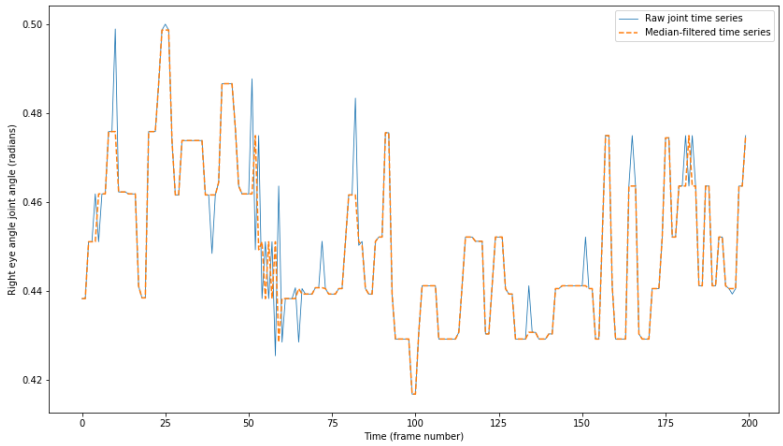


Figure 3: Median-filtering gets rid of some peaks, which corresponds to tracking errors. The x-axis is in units of number of frames. The video is taken at 100 Hz so each frame corresponds to 1/100ths of a second

Next, the joint angle time series was passed through a median filter of window length 3 time points. A median filter is a common data pre-processing technique whereby a moving window is run through a time series and at each point, the

median of the set of points in the window is calculated. The median filter is a great way to reduce noise, or tracking errors from LEAP. This is especially important because even if the network makes substantial tracking errors 0.5% of the time, this still corresponds to $\sim 5\times10^5$ problematic frames. Median-filtering allows us to greatly reduce the frequency of these erroneous frames. Figure 3 demonstrates some of the peaks (in blue) which get glossed over with a median filter.

With median-filtering, there is a trade-off between denoising the data, and getting a representative chunk of the data. The larger the kernel window, the more data is getting glossed over, which means there are less points to work with, but a smaller kernel size means more errors. Given that the RNN is hypothesized to build an internal representation of the underlying neural dynamics, it should also denoise the input time series. Since this is the case, a smaller kernel window can be used in hopes of finding a denoised output time series (which is confirmed later) but still helping alleviate tracking errors.

The median filter was performed with SciPy's signal Median Filter function.

3.1.3 Splitting the data into train-validation set

The data were concatenated into a training set and validation set. Because the network architecture is stateful - which just means the internal states are kept across batches as opposed to reset to 0 after every batch (stateless) - the angle time series was shaped in such a way that the batch-size b and lookback l parameters were taken into account. The batch-size parameter is the number of input vectors which are fed into an RNN at the same time and the lookback parameter is the number of time points the RNN has access to, in other words the size of the input vector. For the purposes of the RNNs built, $b = 32$ and $l = 100$. The data were concatenated as diagrammed in Figure 4, where $\{\alpha, \beta, ..., \omega\}$ are the set of angles measured; α corresponding to the first point, the tip of the head, β corresponding to second point, the left eye, and so forth.

The 32 individual time series of size $N_l \times n_a$ - where $n_a = 30$ is the number of joint angles and $N_l = N(\frac{l}{b})$; lookback and batch-size are RNN parameters and $N = 10^5$ - were shaped into a multidimensional array of size $N \times l \times n_a$ which was split into a final training and validation set, of sizes $0.8N \times l \times n_a$ and $0.2N \times l \times n_a$, respectively. The data was shaped in a way such that every 32nd value corresponded to the same angle, or $\{0, 32, 64, ..., 99968\}$ correspond to the α angles, $\{1, 33, 65, ...99969\}$ correspond to the β angles, and so on. More concisely, the indices which correspond to α can be defined as $i_\alpha = \{n \mid n \mod 32 = 0\}$ and so on.

3.1.4 Standardizing the data

In order to train the network more efficiently, the joint angle time series was standardized. This was done by subtracting the mean and dividing by the standard deviation of every point for each individual time series, or z-scoring the joint angle series. In order to ensure no information from the validation set

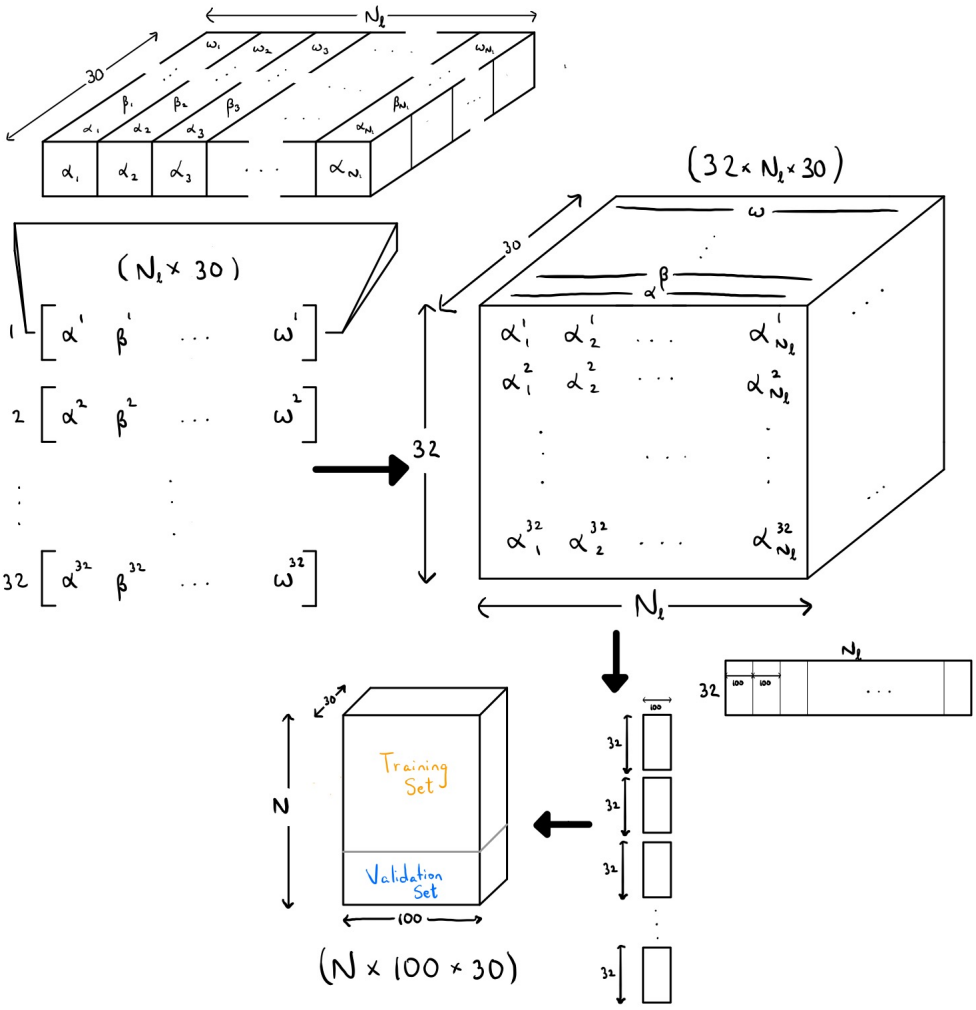


Figure 4: Concatenation of time series into training data set. $N_l = Nl$, where $l = 100$ is the *lookback* parameter for our RNN, a measure of how many points previous in the time series it looks to generate its future prediction

seeped into the model (and more for good practice), the mean and the standard deviation of the training set was applied to both the training and validation set.

3.2 Building a dynamical model: feeding the data into an RNN

3.2.1 RNN Hyperparameters

The hyperparameters listed here are not exhaustive, but they are representative of the tuning factors and if there were hyperparameters not mentioned here, they were kept the same across all models. The first parameter in the RNN is the batch-size. The batch-size is the number of input vectors which get fed into the RNN. A batch-size of 32 was used for all of the RNNs.

I used the Adam learning rate optimizer, which is the default optimizer for the learning rate. The Adam optimizer is an adaptive learning parameter, which means it has a running learning rate for each of the parameters. The Adam optimizer also modifies the learning rate according to the first and second moments of the gradient, where the moments are the same as in probability and statistics:

$$n\text{-th moment: } m_n = E[X^n]$$

In terms of network architecture, I used a Long Short-Term Memory (LSTM) network, a popular RNN used to capture dynamics of the input vector both on the long- and short-term scale. LSTM networks have an architecture similar to traditional feed-forward networks with two important specifications: (1) they contain feedback connections, such that the output of one cell is fed into the input of the next cell (there is nt necessarily a sequential order of cells, the output can be fed into many cells) and (2) they have the ability to "remember" and "forget" the previous state depending on a set of rules.

The input vectors are of size $l \times n_a$, where l is the lookback parameter and n_a is the number of angles, 30. The lookback parameter is a measure of how large the input time series is and it tells the network how far to "look back," or the amount of temporal information provided to the network. Since the data taken was at 100 Hz, the lookback corresponds to a time of 1 second. This means that the network takes information from up to one second ago and predicts the next instance of the fly's movement.

We tested both stateful and stateless LSTM networks. Stateless networks are those where the initial internal states between every batch, $h_0 and c_0$ are set to 0. Stateful neteworks are those where the initial internal states of one batch are the internal states of the previous batch, in other words the hidden and cell states carry forward. Stateful networks turned out to build a better representation, because of the fact that the sequences were made in such a way that each batch held consequtive time points. This meant that the sequences were highly correlated with one another, and stateful LSTMs work best when the sequences are correlated with one another (time series, especially). An epoch is when the entire dataset has been passed through the RNN forwards and backwards once.

3.2.2 1-layer RNN hyperparameters

Batch-size: 32
Number of LSTM cells/neurons in network: 128
Number of training epochs: 126
Number of principle components which explained 95% variance: 55

3.2.3 2-layer RNN hyperparameters

Batch-size: 32
Number of LSTM cells/neuron per layer: 64
Number of training epochs: 255
Number of principle components which explained 95% variance:

Layer 1: 35, Layer 2: 32

3.2.4 3-layer RNN hyperparameters

Batch-size: 32
Number of LSTM cells/neuron per layer: 64
Number of training epochs: 497
Number of principle components which explained 95% variance:

Layer 1: 37, Layer 2: 32, Layer 3: 24

3.3 Building a behavioral representation

3.3.1 Extracting hidden states from RNN

Once the RNN was trained, the hidden states of the RNN were extracted. The hidden states $\vec{h}(t), \vec{c}(t) \in \mathbb{R}^{\mathrm{h}}$ are also vector time series of the size $N_{tot} \times 1$ and there are n_c total hidden state vectors - one corresponding to each neuron. Thus, the total number of hidden state vectors can be concatenated into a matrix of size $N_{tot} \times 2n_c$, where n_c is the number of neurons, $2n_c$ because there are both $\vec{h}(t)$ and $\vec{c}(t)$ weights. Note that for training, $N = 10^5$ samples were used but the hidden states were extracted by externally driving the RNN on all 30 input time series of $N \sim 3.6 \times 10^5$ for a total of $N_{tot} \sim 1.08 \times 10^8$. The "external driving" consists of having the RNN predict the next time series value as diagrammed in figure 5 and updating the hidden states based on the error.

The $\delta_i' s$ were based on the difference between the prediction and the actual value, $\alpha_i' - \alpha_i$. The actual values predicted were less important than driving the hidden states to values for analysis and the result were vectors concatenated into a matrix of all the hidden states. Because analysis of the layers were done separately - meaning behavior maps were created for each layer - the size of the hidden weights matrix was different for the 1-layer RNN to the 2- and 3- layer RNN, but the analysis remained the same.

In theory, the hidden states have created a representation of the dynamics governing the fly's output (behavior) in some high-dimensional space, $\mathbb{R}^{2\mathrm{n_c}}$;

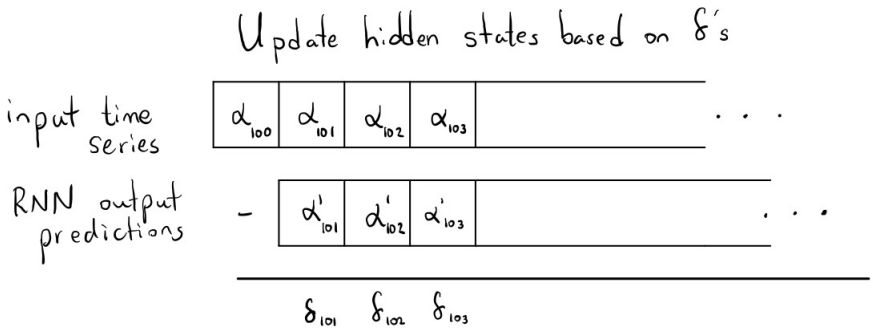


Figure 5: Hidden states were driven by feeding in time series, where weight updates are proportional to prediction differences

the dimension being $\mathbb{R}^{256}$ for the 1-layer RNN and $\mathbb{R}^{128}$ per layer for the 2-layer RNN. However, as stated earlier, animals tend to occupy ony a small fraction of this space - meaning that the actual dimensionality of the behavioral representation is one of much lower dimension. In other words, the actual posture space is a low dimensional manifold living in this high dimensional state space. The goal now is to reduce the dimensionality via the following steps.

One more thing to note: the $\vec{c}(t)$ hidden cell state is generally tanh'd to keep the values between -1 and 1. Otherwise, these values are not comparable to the hidden state values, which are naturally kept on the interval [0, 1]. So, the matrix actually consists of $\vec{h}(t)$ and $\tanh(\vec{c}(t))$.

3.3.2 PCA on hidden states of RNN

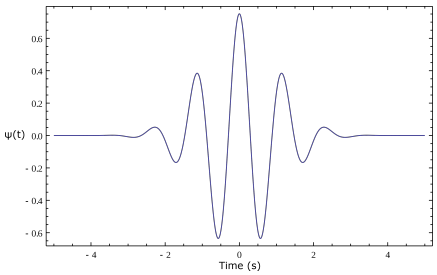
Principle component analysis (PCA) is a technique used to find the axes along which the most variance is explained. After finding the axes along which the most variance is explained, PCA redefines the input data based on the new components, the "principle components." PCA ensures that the new components it finds are linearly uncorrelated, however they can and usually are a linear combination of the original components. So, if the original components are $\{h_1(t), h_2(t), ..., h_{128}(t), c_1(t), c_2(t), ..., c_{128}(t)\}$, then a new first component could be $pc_1(t) = a_1h_1(t) + a_2h_2(t) + ... + b_1c_1(t) + b_2c_2(t) + ...$ where a_1, a_2, b_1 and b_2 are all constants. Performing PCA on the weight matrix will return a matrix of the same size, but in principle, only a small fraction of the total components are necessary. Rather than holding a constant number of principle components however, we just see how many principle components are necessary to cross the 95% variance explained threshold, this will be different for each of the layers.

Importantly, PCA captures linear correlations and does not capture corre-

lations that are non-linear. The way in which PCA maximizes the variance for each new component added is not discussed here, but has been elaborated on elsewhere (it essentially solves the eigenvalue-eigenvector decomposition problem for the covariance matrix). For our purposes, it is suffice it to say that the principle components are linear combinations of the original hidden states and the top principle components explain most of the variance and one can get arbitrarily close to explaining all the variance (all the principle components would explain all the variance, but then the dimensionality remains the same), where variance is the same variance as in statistics: $\sigma_x = \left\langle x^2 \right\rangle - \langle x \rangle^2$. A new matrix, $\overleftrightarrow{pc}$ of size $N_{tot} \times j$ is created, where again j is the number of principles components required to explain 95% of the variance.

3.3.3 Wavelet transform of principle components

A Fourier transformation on a function decomposes the function into its constituent frequencies. Similarly, a wavelet transform decomposes a function into its constituent frequencies, but it also captures frequency dynamics at multiple time scales. Whereas a Fourier transform sweeps different sine and cosine waves to reconstruct the original time series, the wavelet transform sweeps wavelets. The notable features of a wavelet are that it integrates to 0, and it can get arbitrarily close to 0 at $\pm\infty$.



Morlet Wavelet (figure courtesy of Wikipedia, Creative Commons License)

Since sine and cosine waves have the same structure across time, what wavelets add is the ability to resolve in time, which is why the wavelet transform is a function of both time and frequency whereas the Fourier transform is just a function fo frequency.

The wavelet we used is the real-valued Morlet wavelet, which is a sinusoidal wave - generally a sine or cosine wave - multiplied by a Gaussian function. The Gaussian function serves to localize the wave into a Gaussian wave packet and the wavelet can be defined as:

$$\psi(\eta) = \pi^{-\frac{1}{4}} e^{i\omega\eta} e^{-\frac{\eta^2}{2}}$$

Then, a wavelet transform is the following:

$$W(s,\tau)[pc_i(t)] = \frac{1}{\sqrt{s}} \int_{-\infty}^{\infty} pc_i(t) \psi^*(\tfrac{t-\tau}{s}) dt$$

We perform the wavelet transform on 25 frequencies over a range of 0.5 to 50 Hz, dyadically spaced (dyadically spaced just means that the frequencies aren't equally spaced but rather more are concentrated near 0.5). This leads to a new vector of the same size as the previous vector for each frequency, the resulting

matrix is a spectrogram of size $N_{tot} \times 25j$. This is a pretty large matrix, if there are 30 principle components which explain 95% of the variance, then the spectrogram is a subset of $\mathbb{R}^{750}$. Again, we rely on the fact that although this spectrogram lives in a high-dimensional space, it only occupies a small region - it is chiefly low-dimensional.

3.3.4 Creating a behavior map with t-SNE

Finally, we apply a dimensionality reduction technique known as t-distributed Stochastic Neighbor Embedding (t-SNE)[27]. Considering data in high-dimensional space, one way we can categorize structure is global versus local. Global structure would be structure in data which is describes far away from each other. The opposite of global structure would be local structure, which describes points which are close to one another. Most dimensionality reduction techniques - PCA, Isomap, multi-dimensional scaling, etc. - preserve the global structure of the data at the cost of local structure. This translates to points which are close together in high-dimensional space being potentially separated in the low-dimensional representation. However, t-SNE preserves local structure at the cost of global structure (and one other thing) - which is precisely what we want, since postures which are "close" together in high-dimensional space should be clumped together and we are not too concerned with postures which are very much different being kept in the same global orientation - as long as they are far apart in the low-dimensional representation, it does not matter where. t-SNE is a great candidate for visualizing the spectrogram behavior space, then. What t-SNE preserves is the transition probability defined as

$$p_{j|i} = \frac{\exp(-\frac{d(t_i,t_j)^2}{2\sigma^2})}{\sum_{k\neq i}^{\sim 25} \exp(-\frac{d(t_i,t_k)^2}{2\sigma^2})}$$

where $d(t_i, t_j)$ is the distance between the two points which is defined as the Kullback-Leibler (KL) divergence between two feature vectors. The transition probability of going from point t_i to t_j is a function of the distance and it is the probability that you would go from one point to another if you were to take a random walk from that point, assuming that the probability of transitioning to the point you are currently at is 0.

The other cost of performing the t-SNE algorithm is that it is computationally intensive - it scales as $O(N^2)$ with respect to memory. However, a clever sampling method developed previously reduces the scaling to $O(NlogN)$ by sampling representative points from each of the 30 datasets to create a full embedding from all the data. The resultant 2-d map is a representation of the high-dimensional spectrogram.

3.4 Identifying stereotyped behaviors

3.4.1 Peak-finding with Gaussian smoothing

Although the 2-d map generated from t-SNE is a representation of the behavior space, it is largely uninterpretable unless the clusters from the map can be

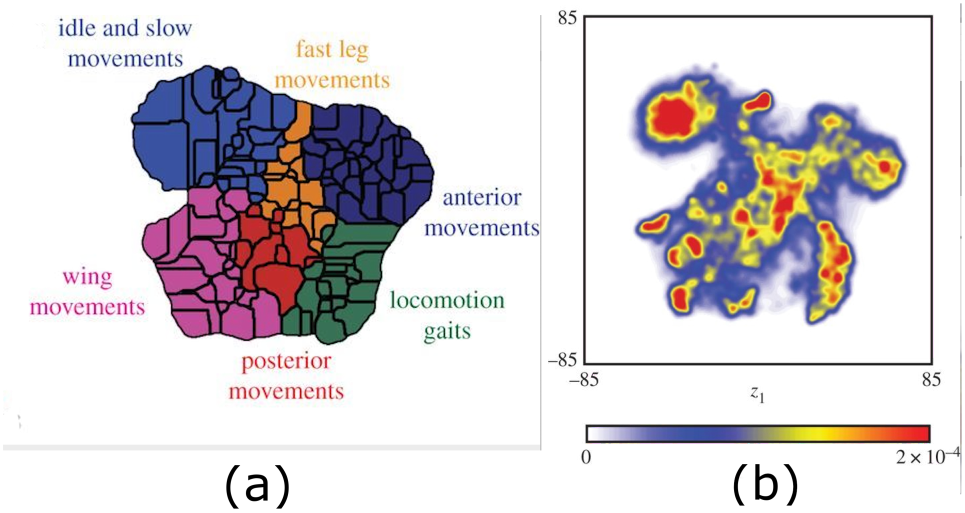


Figure 6: Behavior map of raw postural time series. (a) are the coarse-grained watershed regions and (b) is the t-SNE density map [$\sigma = 1.5$]. Figure courtesy of Berman (2014) [20]

distinguished from one another. Since the map contains millions of data points, however, where points are concentrated isn't immediately obvious. Turning the map into a probability density function allows us to visualize the peaks. A probability density function can be obtained by convolving every point in the dataset with a Gaussian function; depending on the width of the Gaussian, maps with different granularities will arise. A smaller σ value corresponds to a narrower Gaussian peak and therefore finds more peaks while a larger σ alue corresponds to a broader Gaussian with fewer overall peaks. The peaks can also be clipped by a value, c_{lim} to ensure that they do not dominate other peaks. This c_{lim} generally is somewhere on the interval [0.3, 0.7].

Figures 6b and 7a are the behavior maps generated from performing the aforementioned analysis on 50 fly PCA modes and raw postural time series. The first behavior map is a 1250-dimensional space (25 frequencies times 50 postural modes) so it gives a sense of the dimensionality of the behavior space. These maps offers a good comparison for the maps generated from the RNNs, as they also represent projections from a high-dimensional behavior space onto a 2-d plane.

3.4.2 Watershed transformation

The watershed transformation is a common image segmentation technique used to divide an image into different regions [28]. The number of watershed regions identified by the algorithm depends on the number of peaks present, so it is also a function of the Gaussian σ value, where a lower σ will find a larger

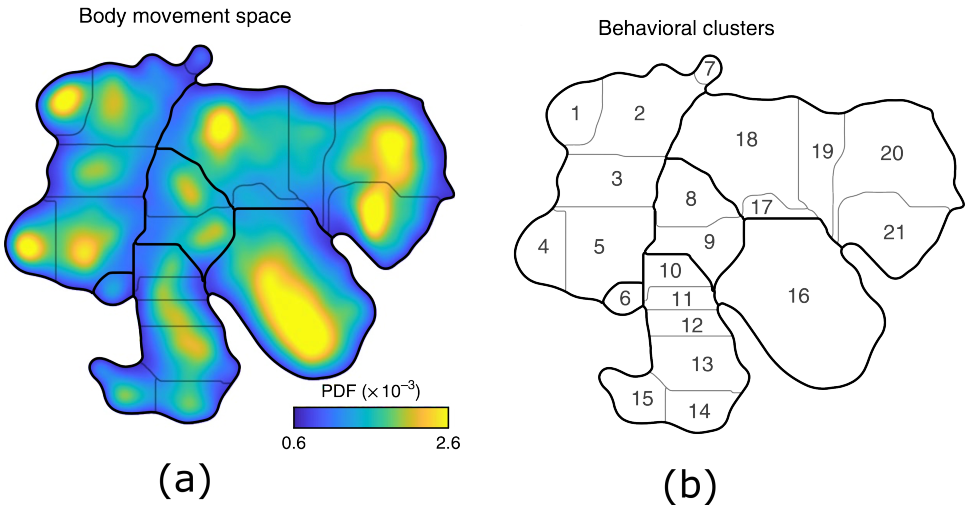


Figure 7: Behavior map of raw postural time series. (a) is the t-SNE density map [$\sigma = 0.65$]and (b) are the coarse-grained unlabeled watershed regions. Figure courtesy of Pereira (2019) [22]

number of watershed regions and a larger σ will find less watershed regions. If there are a larger number of watershed regions, each behavior region would be a coarser description of the animals movements at that time whereas more watershed regions would mean each the regions would have behaviors with finer detail separated from other regions. For our purposes, we have the watershed algorithm vary the σ value of the Gaussian until it finds somewhere between 125 and 150 regions, but in theory the number of watershed regions can be anything as well. Since σ itself is a parameter, one can derive arbitrarily subtle behaviors. This represents the transition from discrete behaviors to continuous behaviors.

Figures 6a and 7b are the watershed regions generated from [11] and [22] corresponding to the behavior maps in the previous section. We can see that more coarse behaviors can be studied by choosing a smaller number of watershed regions as in [22] or finer behaviors can be studied by choosing a larger number of watershed regions as in [11]. Moreover, the larger number of regions will encapsulate the smaller regions within it, as one would expect with coarse and finer behaviors.

3.4.3 Composite movies

Each point in the watershed (and t-SNE) map can be traced back to a single frame in one of the fruit fly videos. Points in a watershed region can be extracted;the frames ordered and a mini snippet in time can be constructed of the fly which corresponds to that region in the behavior map (and more generally, the behavior space). Based on those videos, then, the behavior space

can be retroactively segmented into descriptive regions. If we pull multiple videos of flies performing the same stereotyped behavior from one region and play them next to one another, we can visually analyze the videos and see the flies are doing similar things. Playing these videos side-by-side and in synchrony, we can create composite movies to see what the flies are doing to generate descriptive attributes in each of the watershed regions.

The composite movies for the joint angles, joint positions, and each of the RNN layers can be located by clicking here*. The folders are described by the total number of layers first and then the layer which the folder corresponds to; e.g. RNN-layer3.2 corresponds to the 2nd layer of the 3-layer RNN.

* https://www.dropbox.com/sh/r96cx3qz5v1mybd/AACQufWCJg93N1wo9XBi6mPma?dl=0

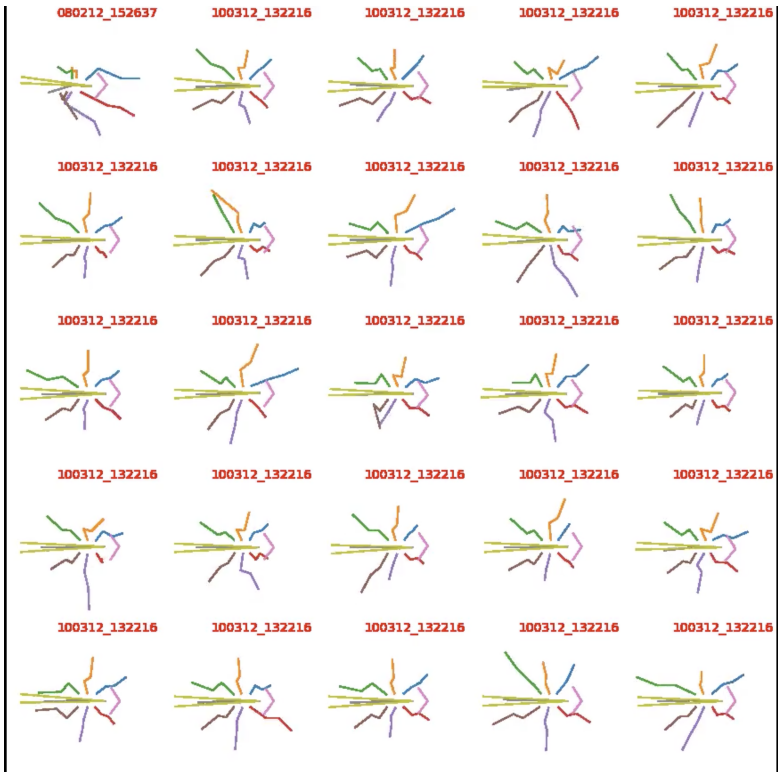


Figure 8: Screenshot of composite movie video. All flies are taken from a single region and placed side-by-side in a grid performing the same action

4 Results

We hypothesized that RNN representation would minimally produce a cleaner version of the map generated directly from postural data; RNNs should hopefully capture the essential temporal correlations in the time series data fed in and correspondingly the most important parts of the temporal dynamics would be extracted. When forced into a lower-dimensional space, then, the peaks would be more finely separated into distinct movements, and ideally more finer, subtler, and detailed behaviors could be realized via watershed transformation. The general behavior pipeline to get from outlined below.

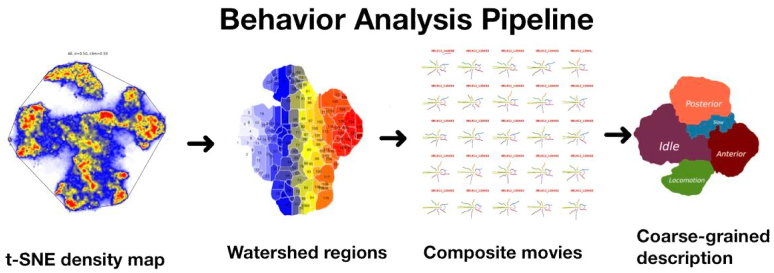


Figure 9: The pipeline for getting coarse-grained watershed regions from a t-SNE density map. Each frame in the t-SNE generated map corresponds to a single frame. Frames which exist in a similar region in the behavior map and are from the same video can be temporally ordered and played to create a movie. From analysis of the movies, coarse-grained behavior maps can be generated corresponding to the stereotyped behaviors the fly is performing at each region.

4.1 Model errors

We trained the RNN network to predict the output at the next time step of the fly; not the *behavior*, but the dynamics, in the hopes that we would glean the behavior from analyzing the interal states. One thing we can do is check exactly how good the model is at predicting the next time point. Say, in theory, the model was perfect at predicting fly behavior. Then the simulated fly would be as good as a real fly (a freely moving fruit fly, at least - nothing can be said about how it might react to stimulii) and the simulated fly could be studied for behavior dynamics, which would save hundreds of research hours in developing the tools to study fly behavior - building elaborate chambers, setting up high-speed cameras, tracking algorithms, etc. One way in which we can quantify the error of the model is by looking at the predictions of the RNN compared to the actual values of the fly's movement, an error which is often quantified by mean squared error. The mean squared error for the 1-layer, 2-layer, and 3-layer RNN are 1.53×10^{-3}, 1.69×10^{-3}, and 1.50×10^{-3} radians (0.088, 0.097, and 0.086

degrees) respectively. However, the mean squared errors tells nothing of the distribution of points.

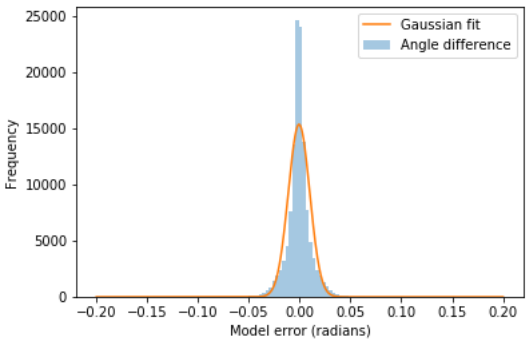


Figure 10: Error distribution for 2-layer RNN. Gaussian fit overlayed; μ and σ for this plot are 5.2×10^{-4} and 1.0×10^{-2}, respectively.

If the goal was to predict the future dynamics of the fly, then the mean squared error would be a good metric since mean squared errors tend to place more weighting on outliers - points which are far off represent a greater chunk of the mean squared error; outliers would greatly impact fly movement predictions if they were used as the input to drive predictions. Even small errors magnify over time, but they matter less if the error magnification can be constrained for the relevant time scale. However, the goal in general was not to maximize the prediction success but to build the interal states of the RNN for analysis. Therefore, we were more interested that most of the points were correctly categorized and less worried about outliers - which were bound to exist but did not affect analysis. Thus, another way to quantify the error of the RNNs was to fit a Gaussian to the distribution of errors and look at the width of the Gaussian. Figure 10 is an example of the typical error distribution for all the angles; the σ's for the RNNs are all $1 \times 10^{-}2$ radians to four decimal places, corresponding to around half a degree. For example, 98% of values lie in the $FWHM$ range $(2\sqrt{2ln2}\sigma)$, within ± 0.235 radians (1.346 degrees) of the true value.

4.2 Joint angle behavior map

First we look at the joint angle representation versus the joint position representation (with x- and y- coordinates). Figure 11 compares the two behavior maps generated for the joint angles and positions. Note the fact that the σ and $clim$ values are not the same is not a problem, since the values are based on the map generated - in order to best visualize the peaks. Regardless, we can see there are a couple of notable similarities and differences between the maps. For one, the joint angle time series is slightly more spread out. The peaks are also slightly more defined in the joint angle representation - presumably a combina-

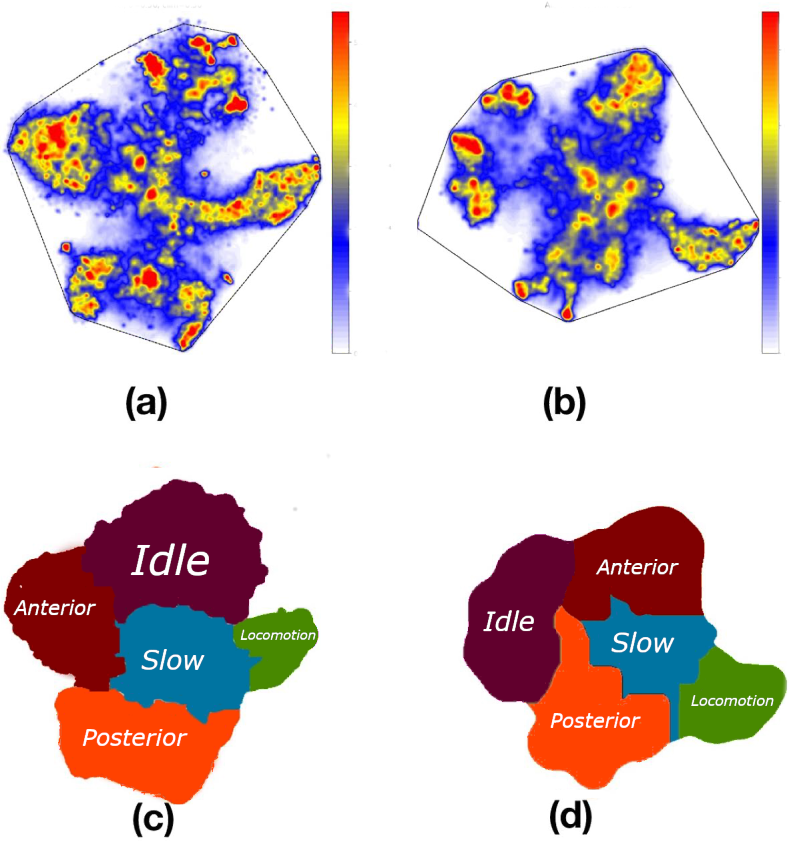


Figure 11: Behavior map generated from joint angles (left) versus positions (right) - no RNN involved yet. (a) [probability density parameters: $\sigma = 0.5; c_{lim} = 0.3; P(x) \in [0, 5 \times 10^{-4}]$] and (c) are the density plots which correspond to the joint angle representations and (b) [probability density parameters: $\sigma = 0.0.6; c_{lim} = 0.55; P(x) \in [0, 8 \times 10^{-4}]$ and (d) correspond to the joint position representations. Key features to look out for are dual peaks in the posterior region, the arc in the locomotion region, and the transition to larger velocities in moving from the slow to locomotion regions.

tion of the fact that the noise associated with two variables x and y is more than the noise associated with a single variable θ and the dimensionality of the joint angles space is significantly lower than the dimensionality of the joint positions space.

Additionally, there seems to be an arc-like region shared by the two maps. Indeed, when the watershed regions are generated for the two maps, both of the arcs correspond to locomotion. In fact, starting medially from center of the map and following the arc outwards corresponds to faster and faster velocities of fly locomotion. While these are represented as discrete states in the behavior

map, it may be more apt to talk about this behavior in terms of a continuous stretch from slow behaviors to fast locomotion.

Another thing to notice - the joint position behavior map is split into around 20 different coarse-grained groups whereas the joint angles behavior map into around 120 groups - which caused the joint angle map to be much more jaggedy around the corners, or more finely delineated. This is an artifact of choosing a smaller σ value for the Gaussian smoothing, which led to more peaks being captured; both end up leading to the same coarsely defined regions.

Indeed, the regions generated by both the joint angles and joint positions contain characteristic dual peaks in the *posterior* region. What's fascinating about these peaks is that they presumably represent some symmetrical topology in the high-dimensional behavior space. A look at the corresponding regions with composite movies tells us that these regions correspond to right-wing grooming and left-wing grooming [composite movies 30 and 84 in the raw joint angle space correspond to the left and right wing grooming, respectively; composite movies 46 and 115 in RNN-1.1layer; composite movies 12 and 40 in RNN-2.2layer]. Grooming is a particularly abundant stereotyped behavior in fruit flies; flies spend a lot of time grooming. The symmetry of the peaks tells us something about the manner in which symmetric behaviors might show up in the behavior map. However, this does not necessarily mean that symmetric regions in the lower dimension space are a reflection of symmetries in higher dimensions, this may just be an artifact of the dimensionality reduction method.

One final thing to note is that there is no issue with the anterior and idle regions being swapped between the joint angle map and the joint positions map. t-SNE sacrifices global structure in order to preserve the local structure as it creates the lower-dimensional representation, so coarse regions such as posterior and idle may be swapped with one another. t-SNE is invariant to rotational and translational changes to data and with the right amout of rotating and inverting across axes, we could get the watershed maps to match up with the density maps by applying the appropriate transformations.

4.3 RNN-generated behavior maps

Armed now with landmarks in our fly behavior map to reference, we can look into the maps generated by RNNs. I'll start with comparing the 1-layer RNN to the joint angle map and other previous maps generated, followed by comparing the performance of the 1-layer RNN to the 2-layer RNN to the 3-layer RNN, and finish up with feedforward flies - driving the network with its own predictions and seeing the resulting dynamics.

4.3.1 1-layer RNN

Figure 12 is the behavior map and the coarse-grained map generated from the 1-layer RNN hidden states. This behavior map is exceedingly similar to the map generated with fly PCA modes, figure 7. There are a number of aforementioned notable features present: (1) the characteristic arc of locomotion, where as you

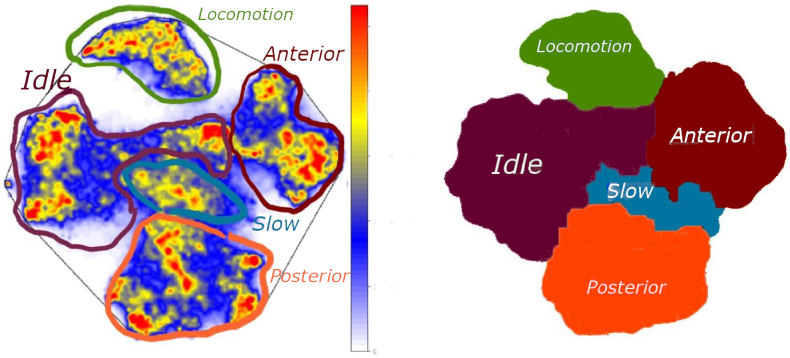


Figure 12: Behavior map and coarse-grained watershed regions generated from 1-layer RNN hidden states. The five coarse-grained regions are outlined in the density map and their positioning demonstrates rotational invariance - inversions across axes do not affect watershed region determination. [probability density parameters: $\sigma = 0.6; c_{lim} = 0.55; P(x) \in [0, 5 \times 10^{-4}]$]

sweep across the arc you go through a gradient of locomotion speeds; (2) the twin peaks in the posterior region corresponding to the left and right wing grooming; (3) multiple similar idle peaks. One of the benefits of having individual joints to work as opposed to raw images is the ability to describe behaviors in terms of individual body segments.

4.3.2 "Glitch" regions

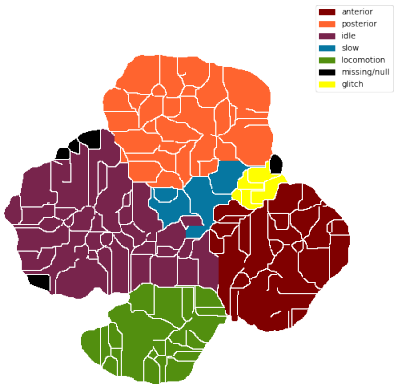


Figure 13: "Glitch Regions" are behavioral regions where the network messes up in a particular way. That glitch regions are clustered together in the behavior map indicates there is some similarity between the glitches aside from the glitch itself. Can the regions develop inslight into network prediction errors?

The watershed algorithm picks out regions based on Gaussian density peaks. Interestingly, some of the regions that the watershed algorithm picks out are spasms in the network - video snippets where the network trained by LEAP makes errors [see composite movie 11 in raw joint angles; composite movies 107, 110, 111, 114, 116 in RNN-layer1 for examples of this]. This ties back to a fundamental problem mentioned earlier: predictive power is heavily penalized by small network inaccuracies. Touting a network accuracy of even 99.99% still isn't enough; the network is prone to large swaths of mislabeled videos - for a dataset with $\sim 10^8$ data points there can be tens of thousands of mislabeled points. It seems, though, that in certain cases, the network makes errors in predictable and often enough manners that with a sufficiently narrow Gaussian peak it will get picked up as a behavioral region. In fact, if I go so far as to encode the "glitch" as a new coarse behavior (figure 13), the region corresponds to a significant portion of the behavior map and it is localized to one area. This suggests that the places where the network are messing up are localized in behavior space. One could imagine utilizing this fact to correct network errors - improving the accuracy of tracking software even more.

4.3.3 Comparing different RNN architectures - number of layers

Our visual system relies on hierarchical processing, the first set of neurons which obtains information from eyes captures a low-level set of features - color, edges and lines, those sorts of simpler features. The next set of neurons captures slightly more complex features, presumably combinations of the simpler features in the lower levels of neurons [29] .

Applying the same reasoning to neural network architecture, it was hypothesized and proven with great success that "deep learning" could develop along a similar set of rules, namely that by stacking layers, different sets of features would be extracted from layer to layer. Along this line of thinking, a multi-layer RNN should similarly, build representations where the earlier layers in the RNN capture simpler features of behavior and that by adding more layers the network could construct more complex behaviors. Thus, we hypothesized that when adding more layers, relatively simpler behaviors would arise in the first layer(s) and progress to more complicated behaviors. This is not so apparent in the 2-layer RNN (figure 14), where the first and second layer look fairly similar in terms of the behavioral repertoire, but in the 3-layer RNN the difference was noticeable, even in coarse-grained visualizations. In the 3-layer RNN (figure 15), sole wing movements became a separable simple behavior seen in the first layer [see composite movies 17, 18, 19, 24, 33, 37, 49, 50, and 51 in RNN-layer3.1] whereas they only showed up in combinations with other behaviors in the 1-layer and 2-layer RNN.

Additionally, more complicated behaviors are differentiated in later layers. For example, in the first layer of the 3-layer RNN, slow and quick grooming of the wings are mixed together [see composite movie 8 in RNN-layer3.1] whereas in the third layer the slow and quick grooming of the wings are separate regions [see composite movies 127 and 128 in RNN-layer3.3]. Similarly, the fly having

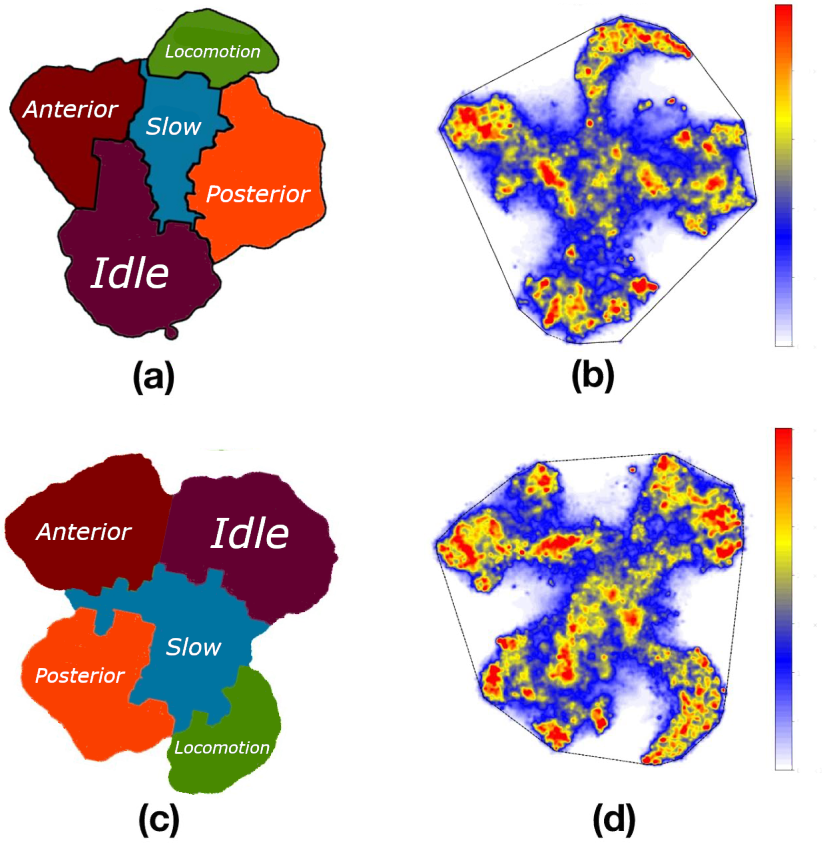


Figure 14: Behavior map and coarse-grained watershed regions generated from 2-layer RNN hidden states. (a) and (b) [probability density parameters: $\sigma = 0.5; c_{lim} = 0.35; P(x) \in [0, 4 \times 10^{-4}]$] are the maps for the first layer and (c) and (d) [probability density parameters: $\sigma = 0.5; c_{lim} = 0.5; P(x) \in [0, 3.5 \times 10^{-4}]$] are the maps for the second layer.

its wings open or closed are mixed in the first layer [see composite movies 25, 98, 104 in RNN-layer3.1] but separated in the third layer [see composite movies 65, 76, 90 in RNN-layer3.3]. This is also true for quick versus fast hindleg grooming [composite movies 13 and 36 in RNN-layer3.1 and 107 in RNN-layer3.3] It seems like more separation is still possible, however, and future work might look at additional layers to better decompose behavior.

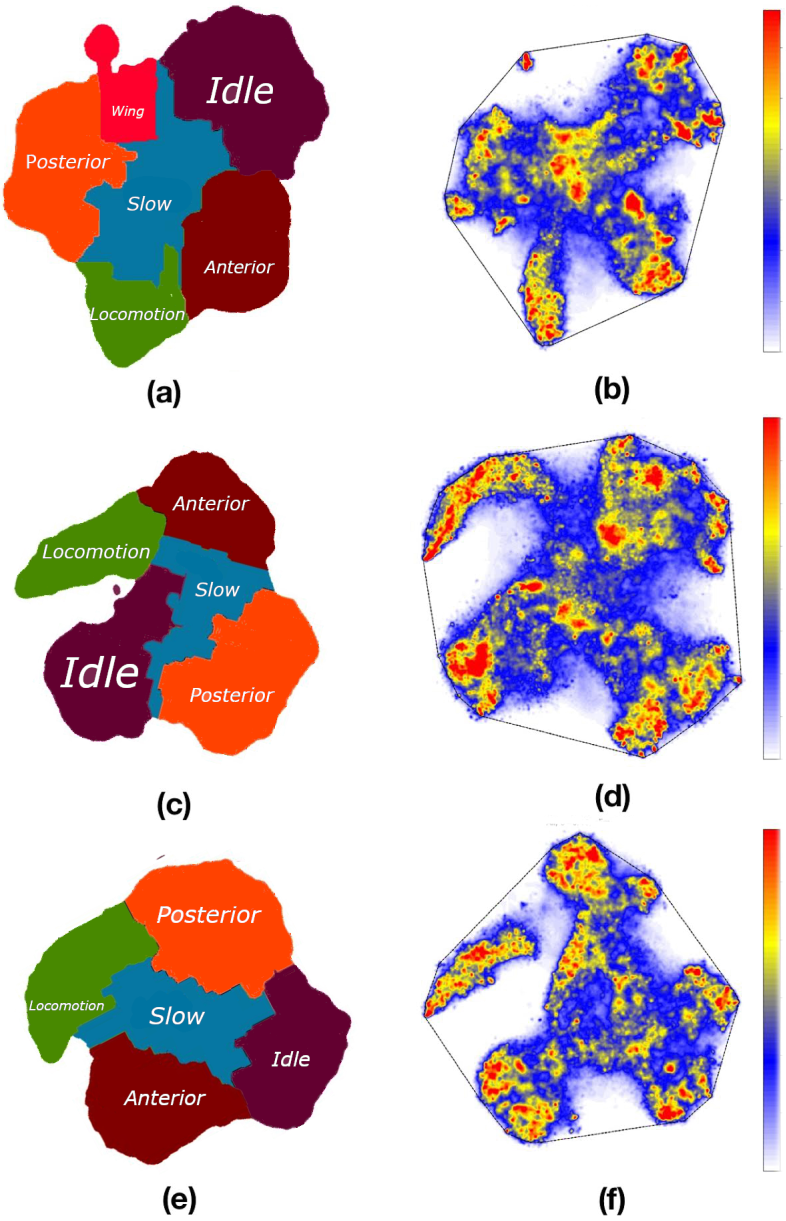


Figure 15: Behavior map and coarse-grained watershed regions generated from 3-layer RNN hidden states. (a) and (b) [probability density parameters: $\sigma = 0.4; c_{lim} = 0.45; P(x) \in [0, 3 \times 10^{-4}]$] are the maps for the first layer, (c) and (d)[probability density parameters: $\sigma = 0.3; c_{lim} = 0.25; P(x) \in [0, 2 \times 10^{-4}]$] for the second layer, (e) and (f)[probability density parameters: $\sigma = 0.4; c_{lim} = 0.55; P(x) \in [0, 2.5 \times 10^{-4}]$] for the third layer.

4.4 Closed-loop networks

As previously mentioned, we used the predicted outputs as the input for the next time point - known as a closed-loop or internally-driving the network. Then, we could analyze the resulting fly dynamics. Figure 16 are the steady state flies - the configuration of the fly once the network reaches a steady state value for each of the networks. For each of the RNNs, only one steady state fly was reached for all the initial conditions analyzed, although one could imagine having different steady state flies depending on the initial conditions. The steady state generated by the 1-layer RNN does not look like a fly, but the steady state generated by the 2-layer and 3-layer RNN look reasonable.

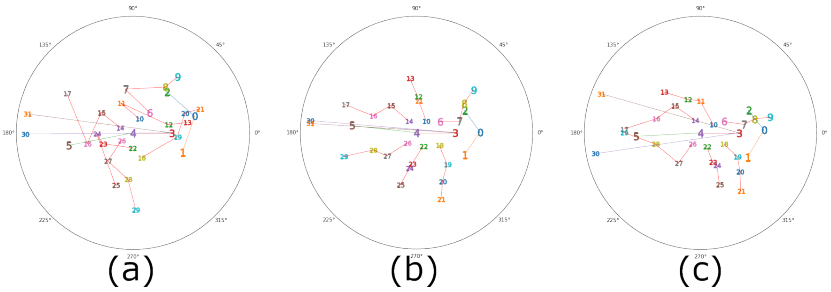


Figure 16: Steady state flies for each of the 3 RNNs. (a) is the 1-layer RNN, (b) is the 2-layer RNN, and (c) is the 3-layer RNN.

Movies for the internally-driven network are available here*. There are two timescales of particular importance to us: the time it takes for the fly to reach steady state τ_s and the time it takes for the fly to stop "looking like a fly" τ_f which could be thought of as the operational timescale at which the RNN can predict the fly's behavior, or the point at which the RNN-predicted fly is no longer a faithful representation of the fly. This is qualitatively defined for now but one could imagine specifying a concrete mathematical definition by, say, having a certain number of angles outside the regular distribution of those angles.

* https://www.dropbox.com/sh/r96cx3qz5v1mybd/AACQufWCJg93N1wo9XBi6mPma?dl=0

Figure 17 is an example plots of the time series plot for the internally-driven fly, with two sample angles - the left eye angle (joint 2) and the right forelimb angle (joint 7) - plotted from the 1-layer RNN driven forward. It takes approximately 8250 frames, or $\tau_s \sim 8.25$ seconds to reach a steady state value for all of the angles for the 1-layer RNN. Figure 18 is a similar plot for the interally-driven fly generated by the 2-layer RNN; steady state is reached much more quickly, $\tau_s \sim 1.9$ seconds. Figure 19 is the plot for the internally-driven fly generated by the 3-layer RNN. Now, steady state is not reached but rather, the predictions spike every so often. This tells us something interesting about the types of attractors for the networks; the 1-layer and 2-layer RNN generate

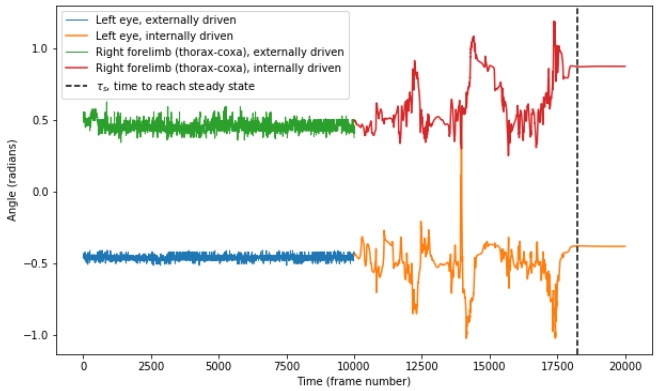


Figure 17: Plot of externally and internally driven 1-layer RNN network for two sample angles: left eye and right forelimb (thorax-coxa). The time it takes to reach steady state corresponds to around 8.25 seconds, markedly longer than the 2-layer RNN

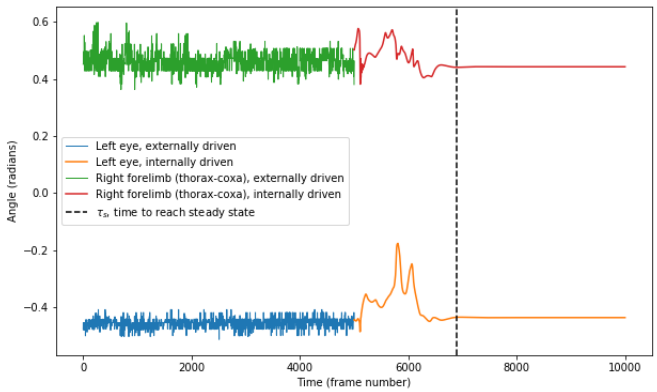


Figure 18: Plot of externally and internally driven 2-layer RNN network for two sample angles: left eye and right forelimb (thorax-coxa). The time it takes to reach steady state corresponds to around 1.9, much shorter than the 1-layer RNN

fixed attractors in behavior space whereas the 3-layer RNN generates limit cycle attractors.

The times it takes for the feedforward fly to stop looking like a fly are immediately for the 1-layer RNN, 0.4 seconds for the 2-layer RNN, and around 0.55

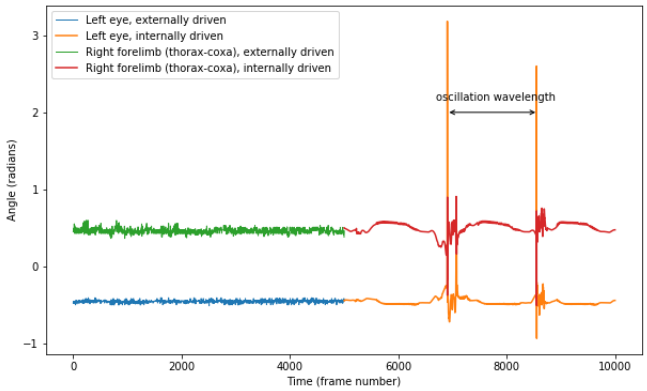


Figure 19: Plot of externally and internally driven 3-layer RNN network for two sample angles: left eye and right forelimb (thorax-coxa). Note that now no steady-state is reached! Instead, the network continuously spikes every so often, around 1.9 seconds

seconds for the 3-layer RNN. This depends, though, on the type of movement the fly is in directly before the driving occurs - for locomotion $\tau_f \sim 0.2$ seconds and the feedforward fly immediately quits locomoting.

5 Discussion

5.1 RNNs on flies versus rats

Although the behavior maps generated from the postural data and the RNN hidden states were strikingly similar, it wasn't necessarily the case that it had to be so. For example, with rats the behavior map generated from RNN hidden states is markedly different than the behavior map generated from postural data (figure 20), with density peaks corresponding to different stereotyped behavior for rats. One could ask the question *a posteriori* why that might have been the case. One hypothesis is that fly dynamics are less correlated than rat dynamics. A fly is able to move one set of limbs fairly independently from another set of limbs. For example, the hindlimbs can be grooming the wings while the forelimbs of the fly are stationary. A rat, on the other hand has more correlations between the spatial dynamics of the data. The tracked points for a rat are on the body as opposed to the appendage and when a rat twists and turns its body, the points tend to move together. Thus, knowing the position of one or two of the points gives a lot more information about where the other points might lie, for example on the other end of an arc. The fundamental difference in the gaits of rats and flies could account for the similarilty or lack thereof in the behavioral

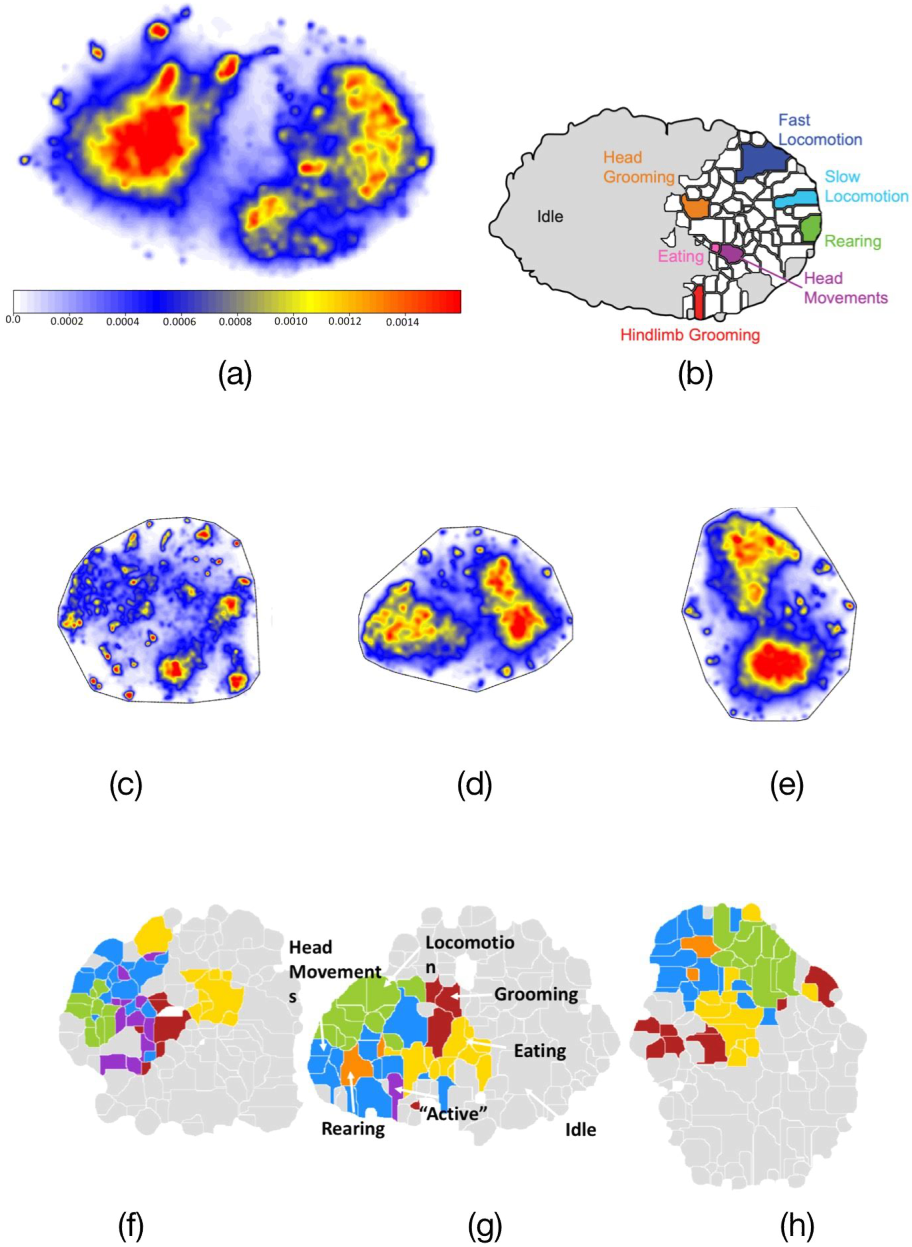


Figure 20: Behavior map generated for rat. (a) is the density map from postural time series, (b) are the watershed regions for the postural time series, (c) (d) and (e) are the density maps for the first, second, and third layer and (f) (g) and (h) are the corresponding watershed regions.

maps.

Another hypothesis is that the time correlations drawn by the RNN weren't as important as the spatial correlations. Upon performing a continuous wavelet transformation, which captured the temporal dynamics across a large range of frequencies, it is possible that whatever temporal dynamics the network had captured were not important compared to the temporal dynamics captured by the wavelet transform. In that case, what would be left is the spatial correlations between points, which could be the joint angle representation built by the RNN. Since the RNN is a highly overparameterized system, with many more variables than probably needed to capture the essence of motion, it is possible it devoted a small subset of parameters to representing the joint angles and another subset to temporal correlations which were then overshadowed by the temporal dynamics registered by the wavelet transform. There are ways in which we could test this. Namely, one could investigate whether the RNN representation of a rat reduces down to its joint angle representation. Or, staying with flies, one could see whether reducing the number of neurons in the network and therefore the number of parameters would lead to a different representation.

5.2 Future directions

One of the first things to do is to apply the idea of RNN representation in other animals. There are many animals for which the postural dynamics could be either similar or vastly different from it's RNN representation, but it is not known what exactly the rules are which govern similarity between representations. More examples between animals would help bring a deeper understanding of the types of correlations in postural dynamics. Another question one could ask is: when flies are no longer restricted to the 2-d plane, does the RNN representation still closely mirror that of the postural representation?

Additionally, if the RNN can predict behavior accurately, then one could feed-forward the RNN and create a virtual feedforward fly. Simulations now a significant chunk of experimentation - one could simulate fly behavior and analyze the resulting dynamics. This would save much experimental effort and cost dedicated to studying fly behavior: building elaborate chambers, setting up high-speed cameras, tracking algorithms, etc.

6 Conclusions

That RNN behavior maps recapitulate previous work is a great indication that the idea to use RNNs to capture behavioral dynamics could be fruitful. The real power of RNNs shines through when multiple RNNs are strung together. One could imagine ways in which combining RNNs would lead to deeper behavioral analysis. For one, the encoder-decoder architecture previously mentioned could come to a closer approximation of how the brain compresses information. By stringing together an ensemble of neural network models, the architecture could be molded into creating a representation the dimensionality reduction is

encoded into the architecture, rather than the researcher imposing the dimensionality reduction technique on it. The lower dimensionality representation would be analagous to how the brain must compress the high-dimensional information as it passes from the many orders of magnitude greater connections in the brain to a markedly fewer number of connections in the neck to a still fewer number of motor and behavioral outputs. There is a sense in which the representation built, as it goes from network to network, would capture dynamics at longer and longer time scales, going from kinematics on the order of micro- to milliseconds to behaviors on the order of seconds and even minutes, and even then "moods" on the order of hours and days. Thus, RNNs present an opportunities to study the non-Markovian timescale dynamics currently not available to us - opening avenues to quantitatively describe circadian rhythm effects and other such long timescale phenomena.

References

[1] Thomas Baumgartner et al. “The neural circuitry of a broken promise”. In: *Neuron* 64.5 (2009), pp. 756–770.

[2] Zhihao Zheng et al. “A complete electron microscopy volume of the brain of adult Drosophila melanogaster”. In: *Cell* 174.3 (2018), pp. 730–743.

[3] Aaron McKenna et al. “Whole-organism lineage tracing by combinatorial and cumulative genome editing”. In: *Science* 353.6298 (2016), aaf7907.

[4] Talia N Lerner, Li Ye, and Karl Deisseroth. “Communication in neural circuits: tools, opportunities, and challenges”. In: *Cell* 164.6 (2016), pp. 1136–1150.

[5] Reza Kalhor et al. “Developmental barcoding of whole mouse via homing CRISPR”. In: *Science* 361.6405 (2018), eaat9804.

[6] Barry Bentley et al. “The multilayer connectome of Caenorhabditis elegans”. In: *PLoS computational biology* 12.12 (2016), e1005283.

[7] C Shan Xu et al. “A connectome of the adult drosophila central brain”. In: *BioRxiv* (2020).

[8] Seung Wook Oh et al. “A mesoscale connectome of the mouse brain”. In: *Nature* 508.7495 (2014), pp. 207–214.

[9] David Grant Colburn Hildebrand et al. “Whole-brain serial-section electron microscopy in larval zebrafish”. In: *Nature* 545.7654 (2017), pp. 345–349.

[10] John W Krakauer et al. “Neuroscience needs behavior: correcting a reductionist bias”. In: *Neuron* 93.3 (2017), pp. 480–490.

[11] Gordon J Berman. “Measuring behavior across scales”. In: *BMC biology* 16.1 (2018), p. 23.

[12] Matteo Carandini. “From circuits to behavior: a bridge too far?” In: *Nature neuroscience* 15.4 (2012), pp. 507–509.

[13] Hans Slabbekoorn and Thomas B Smith. “Bird song, ecology and speciation”. In: *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences* 357.1420 (2002), pp. 493–503.

[14] Jeremy A Pfaff et al. “Song repertoire size varies with HVC volume and is indicative of male quality in song sparrows (Melospiza melodia)”. In: *Proceedings of the Royal Society B: Biological Sciences* 274.1621 (2007), pp. 2035–2040.

[15] BJ Frost. “The optokinetic basis of head-bobbing in the pigeon”. In: *Journal of Experimental Biology* 74.1 (1978), pp. 187–195.

[16] C Nicol. “Understanding equine stereotypies”. In: *Equine veterinary journal* 31.S28 (1999), pp. 20–25.

[17] Rudolf Jander and Ursula Jander. "Wing grooming in bees (Apoidea) and the evolution of wing grooming in insects". In: *Journal of the Kansas Entomological Society* (1978), pp. 653–665.

[18] Greg J Stephens et al. "Dimensionality and dynamics in the behavior of C. elegans". In: *PLoS computational biology* 4.4 (2008).

[19] Onno D Broekmans et al. "Resolving coiled shapes reveals new reorientation behaviors in C. elegans". In: *Elife* 5 (2016), e17227.

[20] Gordon J Berman et al. "Mapping the stereotyped behaviour of freely moving fruit flies". In: *Journal of The Royal Society Interface* 11.99 (2014), p. 20140672.

[21] Adam J Calhoun, Jonathan W Pillow, and Mala Murthy. "Unsupervised identification of the internal states that shape natural behavior". In: *Nature neuroscience* 22.12 (2019), pp. 2040–2049.

[22] Talmo D Pereira et al. "Fast animal pose estimation using deep neural networks". In: *Nature methods* 16.1 (2019), pp. 117–125.

[23] Alexander Mathis et al. "DeepLabCut: markerless pose estimation of user-defined body parts with deep learning". In: *Nature neuroscience* 21.9 (2018), p. 1281.

[24] Erick Moen et al. "Deep learning for cellular image analysis". In: *Nature methods* (2019), pp. 1–14.

[25] Christopher Olah. *colah's blog*. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[26] Wikipedia contributors. *Long short-term memory — Wikipedia, The Free Encyclopedia*. [Online; accessed 23-March-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Long_short-term_memory&oldid=946900344.

[27] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[28] Fernand Meyer. "Topographic distance and watershed lines". In: *Signal processing* 38.1 (1994), pp. 113–125.

[29] David H Hubel and Torsten N Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". In: *The Journal of physiology* 160.1 (1962), pp. 106–154.