Big data hidden Markov model for tracking neurons in *Hydra vulgaris* videos

Amnon Horowitz
Department of Computer Science
University of Washington
amnonh@cs.uw.edu

Abstract

Rafael Yuste's, Columbia university New York has collected many hours of video data of *Hydra vulgaris* genetically engineered to express a calcium indicator (GCaMP6) in neural cells. These cells fluoresce when they activate. Tracking neurons in the video is a manual, labor intensive process. Tracking is hard because cells are indistinguishable from background when they aren't active and because the *Hydra* moves around and deforms. We show a hidden Markov model based tracker that works successfully over this multi-gigabyte data-set.

1 The data set

The data set [1] is a collection of tiff files containing gray-scale, 16 bit deep, 1024 by 1024 video frames (figure 1). The frames are captured by a camera mounted on a scanning microscope, at 10-30 Hz. Under the microscope, inside a drop of water, sandwiched between a slide and a slide cover is a *Hydra* engineered to express a calcium indicator in its neuronal cell line. The *Hydra* can move around in the X-Y plane but the slide and cover limit movement in in the Z direction. The *Hydra* can deform its body, narrowing, widening, elongating, and shortening (figure 2).

When voltage controlled gates in the neuronal cell membrane open. Ionic calcium flows into the cell, binding to GCaMP6 and causing it to fluoresce. As the cell goes into its refractory period, the concentration of calcium in the cell drops and florescence stops.

2 Feature extraction

Given a sequence of images, $I_1, I_2, \dots I_n,$ we build a sequence of probability matrices $M_1, M_2, \dots M_n$ and of velocity matrices $\vec{V_1}, \vec{V_2}, \dots \vec{V_{n-1}}$. As shown below we will call M_k the neuron score and V_k the optic flow. Let $\vec{a} \in I_k$ be a pixel position. $M_k(\vec{a})$ is the probability that I_k contains an active neuron at $\vec{a}$. $\vec{V_k}(\vec{a}) \in \mathbb{R}^2$ is the velocity vector at $\vec{a}$ from I_k to I_{k+1}.

2.1 Computing M_k and $\vec{V_k}$

For a review of spot detection methods at the edge of the microscopic field see[2]. We use the morphological top hat method after blurring the image with a Gaussian filter ($\sigma = 2$). We use a disk of radius 5 as the morphology filter and set the threshold at the Otsu threshold of the filtered image. After identifying connected components in the thresholded image, we issue a neuron score at the center of mass of each component equal to the size, in pixels, of the component. We set the score at all other pixels in the matrix to some small constant ϵ.

We use a matlab implementation to compute optical flow obtained from [3]

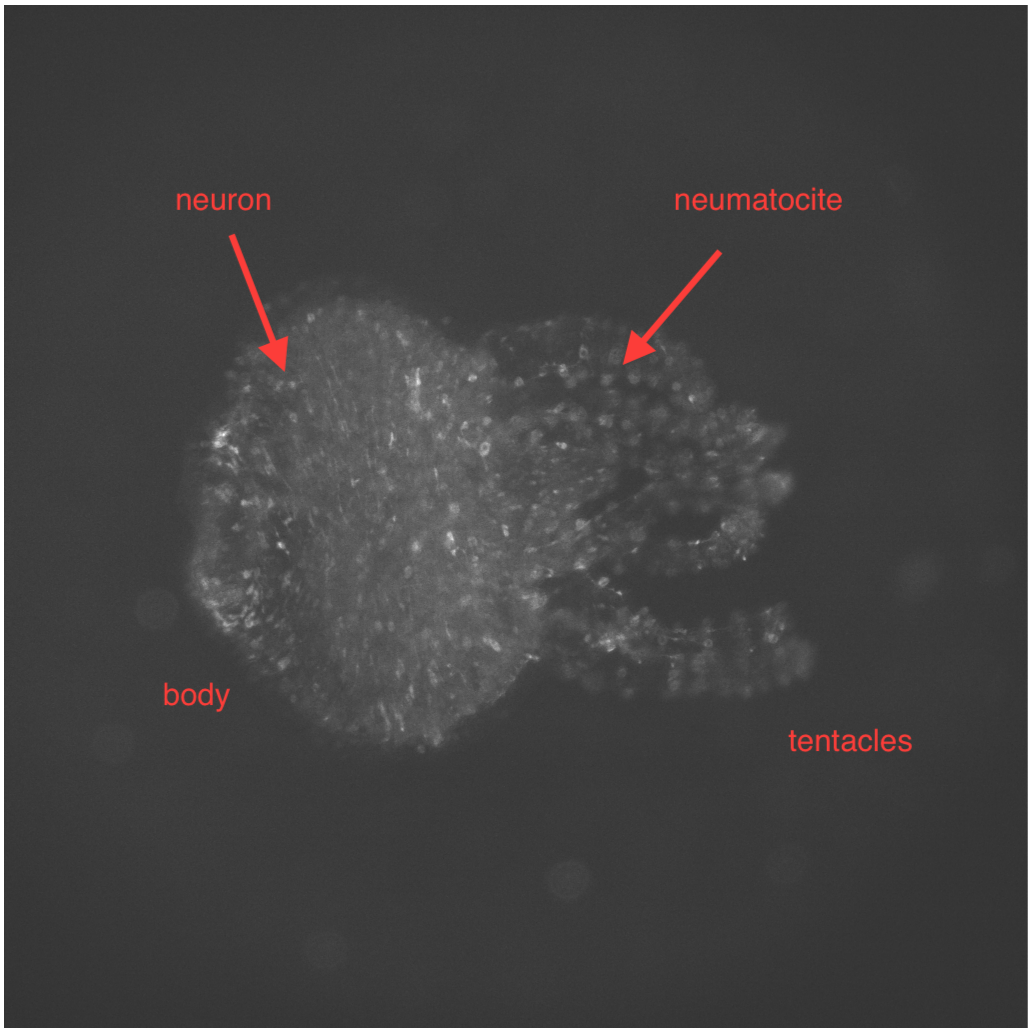


Figure 1: Hydra calcium Image

The M_k and V_k matrices for all images are computed ahead of time and stored in tiff files. The M_k matrices are stored in files with a ”-m” suffix, and the V_k matrices are stored in files with ”-vx” and ”vy” suffixes.

3 A hidden Markov model for Hydra

A hidden Markov model [4] is a generative model where a hidden state of the world generates phenomena we can observe. We can use these observations to deduce the likely state of the world. The Viterbi algorithm computes the maximum likelihood path of state transitions given the model and a sequence of observations. It runs in $\mathcal{O}(n \cdot \mid Y \mid^2)$ time where n is the sequence length and Y is the set of states in the model.

3.1 Model for a single neuron

Our states are vectors in $\mathbb{R}^4$. Each state describes a box (the top left and bottom right coordinates). In order to be a state, the box must satisfy the constraint that every point inside the box has to be closest to the same pixel. We use this construct in order to bridge between observations and optical flows, that are only available at discrete pixels positions, and between the non discrete values of the optical flow vector.

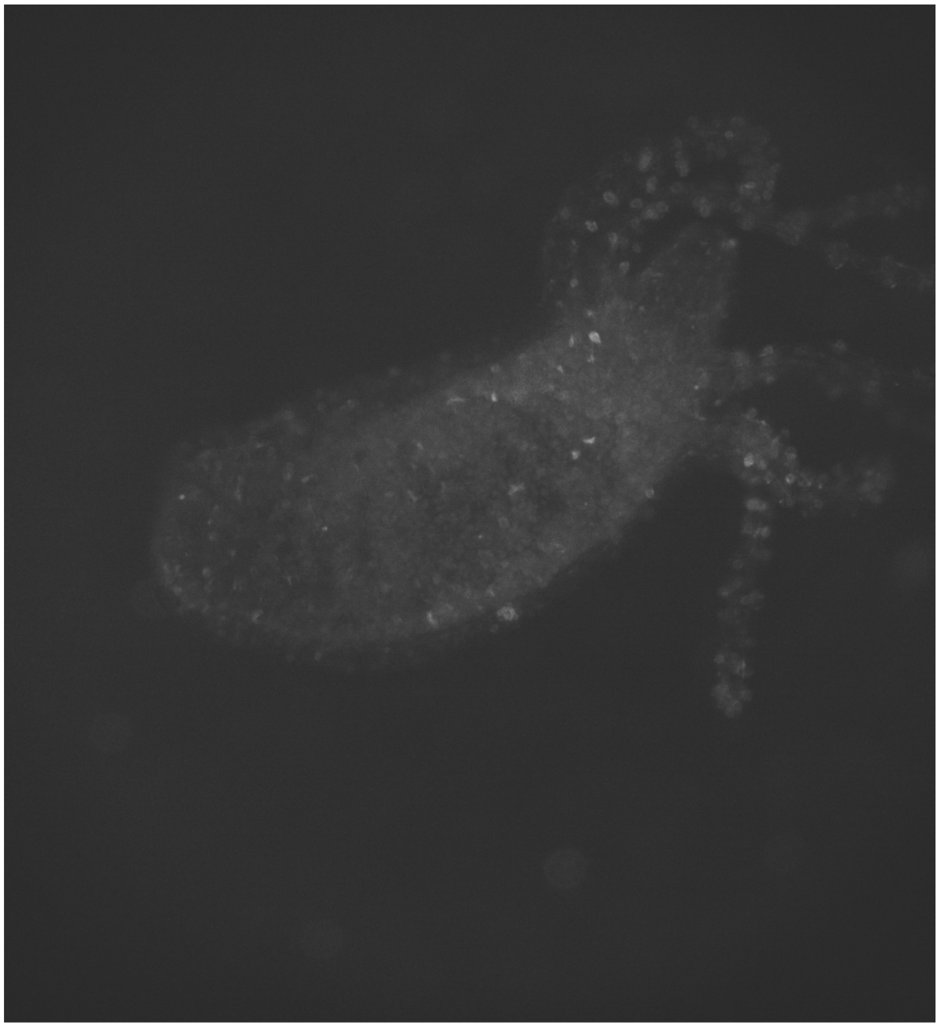


Figure 2: Hydra elongated

State transition probabilities are computed as shown in figure 3. Suppose the model is the state described by red box. Because of the constraint, there is only a single pixel position we have to worry about. We follow the optical flow vector from that pixel (black arrow) and arrive at the black box. Note that the black box no longer satisfies our constraint, and is therefore not a state. We now increase the size of the black box by some error margin, representing our uncertainty in the precision of the optic flow vector and obtain the green box. We slice the green box into multiple boxes that satisfy our constraint. That process defines the set of reachable states from our starting state, and we set the transition probability as uniform in that set.

The model for observations is simple - the probability for the model to generate a neuron score p is exactly p. Note that to be truly probabilities, we would need to scale the scores by a constant factor, but since the Viterbi algorithm is invariant to this scaling, we do not.

The definition of the starting state distribution is outside the scope of this work.For our purposes here we will assume that we are given a single starting state, based on the input of a human observer.

3.2 Extending the model to multiple neurons

It is easy to extend the model to n neurons. Each state is an array of n sub-states from the single neuron model. The transition probabilities between two states would now be the product of the

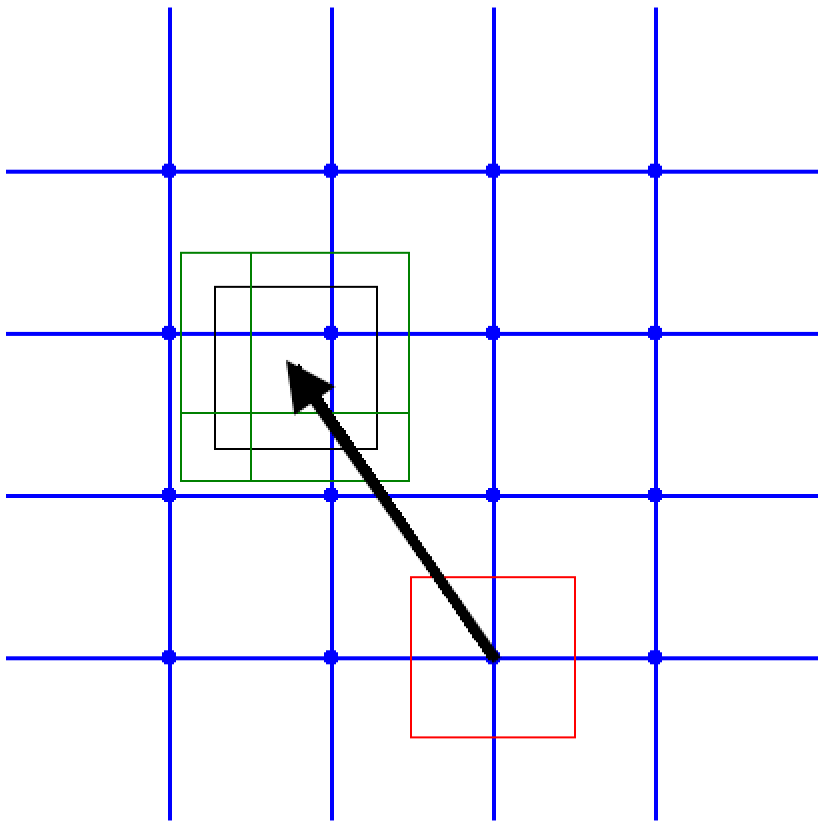


Figure 3: State transitions

transition probabilities between the two corresponding sub-states. In order to make sure we don't end up with two neurons in the same position, we zero out the probability of transitioning to state that has two sub-states surrounding the same pixel.

Similarly, we extend the probability of the model to generate observations. A state will generate a set of observations, each observation at each sub-state position, and each with the corresponding sub-state observation probability p.

4 Tackling the size of the model

The model as we've defined it actually has an infinite number of states, because there are an infinite number of boxes around a pixel. The classic Viterbi algorithm takes time and memory quadratic with the number of states which clearly would not work here.

4.1 Sparse Viterbi

We take advantage of the sparsity of the model to implement a sparse version of the Viterbi algorithm that only tracks reachable states. We maintain an array of maps instead of a matrix. Each element a_i of the array is map of the reachable states in the ith step in the Viterbi path. For each state, the map maintains the probability of the best Viterbi path seen so far, leading to that state. The map also maintains the back pointers required to traverse the path backwards after finding the state with the maximum probability in the final step of the path.

We initialize the array with a map containing the starting state (or states). Then, for each step i in the Viterbi path, we compute the transitions and probabilities from each state in the map a_{i-1}. We find the maximal path probability for each state s in a_i by maximizing the probability of the the path to the previous state in a_{i-1}, multiplied by the transition probability to s and the probability of the observation generated by s.

The original Viterbi algorithm takes as input matrices defining the transition and observation probabilities. This wouldn't work here either. We are providing the algorithm with functions that return these probabilities. The algorithm then calls the functions as needed.

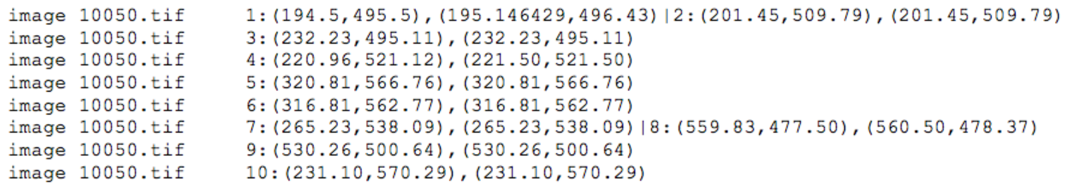
The reason we use maps instead of arrays or simple lists is that our transitions generate many duplicate states. There are many different Viterbi paths that lead to the same state. That issue does not exist in the original Vierbi algorithm because every state has a unique cell in the Viterbi matrix that we can find in constant time. Because of the time required to search the maps, the complexity of our algorithm is now $\mathcal{O}(\cdot \mid V \mid log \mid V \mid)$ where V is the set of reachable states.

4.2 Model segmentation and parallel processing with map reduce

A Hydra can have as many as 5000 neurons. The number of states in our model grows with the number of permutations between the sub-states because of the way the state transitions are defined. The number of permutations grows to be intractable with this many sub-states.

If two neurons are not close to each other, we can assume there will be no interaction between them. This allows us to segment the model into several sub models, each tracking a subset of the neurons, almost eliminating the permutation problem. Given a distance threshold d, we divide the neurons into groups where the neurons in each group are at most distance d from one another, and are at least a distance of $d + 1$ from every other neuron. Typically, we get groups of 1-4 neurons. We can then run Vierbi on each model in parallel.

The sub model structure lends itself naturally to a map reduce pattern. The mappers take a list of models and run sparse Viterbi to compute the neuron positions for the next frame. The mappers run in parallel (and can also run multiple models in parallel). The reducer takes the positions computed by the mappers and computes the sub-models for the next step, which is then fed into the next map-reduce cycle. Figure 4 shows a what a section of a map reduce input file looks like.



```
image 10050.tif      1:(194.5,495.5),(195.146429,496.43)|2:(201.45,509.79),(201.45,509.79)
image 10050.tif      3:(232.23,495.11),(232.23,495.11)
image 10050.tif      4:(220.96,521.12),(221.50,521.50)
image 10050.tif      5:(320.81,566.76),(320.81,566.76)
image 10050.tif      6:(316.81,562.77),(316.81,562.77)
image 10050.tif      7:(265.23,538.09),(265.23,538.09)|8:(559.83,477.50),(560.50,478.37)
image 10050.tif      9:(530.26,500.64),(530.26,500.64)
image 10050.tif      10:(231.10,570.29),(231.10,570.29)
```

Figure 4: Map reduce file

5 Implementation, Results, and future work

5.1 Implementation

The mapper and reducer are implemented in C++ (for speed). The tiff files with the images, neuron score matrices and optical flow matrices were uploaded into the Amazon S3 storage service. The mapper knows to retrieve image, neuron score or optical flow files from the S3 service when they are needed locally. The mapper and reducer were run using hadoop streaming on the Amazon Elastic Map Reduce service - EMR.

Several issues surfaced during the runs on AWS.

- The disk space available on each cluster node is limited and we run out of space. The mappers had to be modified to delete local files as soon as they were no longer needed. This way, the cluster nodes only needed enough storage to maintain the files for the current Viterbi path.
- Haddop breaks up the data into chunks and feeds each chunk into a separate mapper. Because Hadoop was designed for big data, it assumes the data is large and the chunk size is also large. The large data in our implementation is stored in S3 and retrieved by the mapper, and not by Hadoop. The actual data split by hadoop is small - and we ended up with all the data being given to a single mapper with no work left for the other mappers. The implementation had to be changed to not rely on Hadoop's native splitting.
- EMR allows only a limited number of map reduce steps to be scheduled. It's limited in the way it feeds the output of one step to the next and does not have good support for waiting until a step is completed before executing the next step. A controller shell script was implemented that ran outside of EMR, scheduled and synchronized the work and copied output files.

5.2 Results

A sample tracking video is attached in the material. A 100 frame run was completed on a 5 node cluster in approximately 2 hours.

5.3 Future work

We have a small set of labeled data - approximately 200 frames. We can calculate the squared loss of our implementation on this data. The next phase is to optimize the algorithm parameters and the neuron scoring algorithms to minimize this squared loss.

With that complete, we can start doing some actual neuro-science, and start looking for correlations and dimensionality reductions in the neuronal activity.

References

[1] C. Dupre and R. Yuste. The mind of a cnidarian: imaging neural networks in Hydra vulgaris . *Unpublished draft*, 2015.

[2] I. Smal, M. Loog, W. Niessen, and E. Meijering. Quantitative comparison of spot detection methods in fluorescence microscopy. *IEEE Transactions on Medical Imaging*, 29(2):282–301, Feb 2010.

[3] Deqing Sun, Stefan Roth, and Michael Black. Secrets of optical flow estimation and their principles, Jun 13 2010.

[4] L.R Rabiner and B.H. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, 3(January):4–16, 1986.