Michael Noronha

Theoretical Neuroscience, Final Project

For this project, I looked at a perceptual decision task that was studied by Shadlen and Newsome.[1] Their study examined the neural basis for a simple visual decision-making task, and their findings showed that neural responses in the lateral intraparietal area (LIP) of the cerebral cortex integrated motion information (essentially) into a decision. I trained an RNN on this task to a target 90% accuracy, and then looked at differences between the performance and dynamics of this model compared to the original results.

1 Task and Model Details

1.1 Task Description

The studied task was a single-interval, two-alternative, forced choice discrimination of motion direction. Specifically, a collection of moving dots were shown within a 2-second interval, with a subset of them moving coherently in the same direction (either left or right), and the rest moving randomly (in any direction). The monkey had to choose left or right based on what direction of the coherent dot motion:

1. For 350ms, only a single point was visible at the center of the screen, that the monkey fixated on
2. For 500ms, the targets (dots on the left or right the monkeys would look at to indicate their choice)
3. For 2s, the dot motion was shown
4. There was a random delay for approximately 500ms (the fixation point remained visible)
5. The fixation point disappeared and the monkey indicated its choice by looking at the left or right target

1.2 Model Description

1.2.1 Model Architecture/Training

I didn't make any modifications to the pycog defaults. My network had an input dimension of 100, a hidden dimension of 50, and an output dimension of 2.

1.2.2 Model Inputs

There were a several different ways the input could've been presented

1. A sequence of images
2. A sequence of point locations

[1] Paper: `https://www.pnas.org/content/pnas/93/2/628.full.pdf`

3. A sequence of point velocities

4. A sequence of point directions

5. A sequence of direction and coherence levels (possibly with noise)

Because our goal wasn't to build a perception system for extracting semantic information from images, I didn't consider the first option. The remaining options are ordered in roughly decreasing order of intuitive difficulty. To perform the task effectively, the group of neurons need to learn to track the points well enough to tell how they're moving through time (i.e. understand velocity). Also, the speed isn't important, only the direction a point is moving. Based on this, the fourth option, inputting a sequence of directions, seems like a natural choice. The last option, which is what Song et al. did in their example code,[2] seemed easier but less interesting. When the input is a sequence of point directions, the network manages to derive this from the data, so I thought it would be more interesting to look at the dynamics of this network than one that started off with that information.

This illustrates a general trade-off between having information encoded directly in the input (acting like a prior perception system had perfectly extracted it) and forcing the network to extract it itself. I input directions as an angle θ between $-\pi$ and π (so left is $-\pi$, and right is 0). Then, I sampled points moving in random directions uniformly from the interval $(-\pi, \pi)$. This worked well at a range of coherence levels, from as low as 12.5% (which demonstrated some discriminative power at $\sim 70\%$ accuracy) upwards. Given more time I would've liked to try out the other input types.

2 Model Results

2.1 Target Input and Output

I constructed the model input as a sequence of point directions in the interval $(-\pi, \pi)$. This meant that each point was completely described by a single point at each time-step. I included 100 points in the input, so the input vector had 100 entries. I used the same time intervals as described in the task description above,1.1 but I combined and shortened the first two windows (fixation and targets appearing) into one shorter interval.

Because we'd like our model to be able to do the task at a variety of coherence levels, I choose a coherence level uniformly from the interval $(0.125, 1)$ when generating each input.[3] To generate the entries at a set coherence level $p\%$, I randomly chose $p\%$ of the inputs and randomly chose between left and right. I set the directions of the coherently-moving indices to this direction for the duration of the input period, and sampled all others uniformly. This made the task intuitively easier than if the random dot motion was more continuous, since the random dots were changing direction *every* time-step, so it'd be interesting to relax this (eg. by sampling an incremental change in direction) as a follow-up. Note that because the coherent point indices were randomly selected in each trial, the model couldn't rely on only paying attention to a subset of the inputs to determine direction.

The target output was a 2-element vector with the first entry set during the decision period if the coherent motion was to the left, and the second set if the coherent motion was to the right.

[2]Paper: `https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1004792`

[3]I would've gone lower, but I ran initial experiments to see if the network could learn lower coherence rates individually, and wasn't successful

Here's what these targets look like for a 50% coherence input example (the input has some baseline noise):

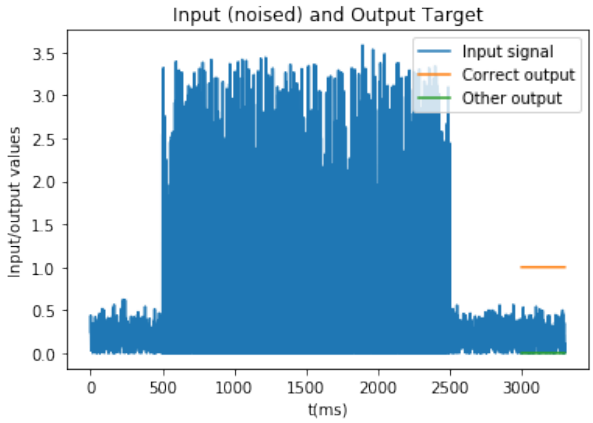

Figure 1: Constructed (target) input and output sequences

Looking at the input signal isn't very informative, since there's a lot of noise by construction since half the points were move randomly. If we ignore the random motion and think of an idealized, noise-free input, the input looks a lot like the output. It's helpful to visualize this idealized input/output on alongside neural activity from the RNN to see how it's achieving the training objective:

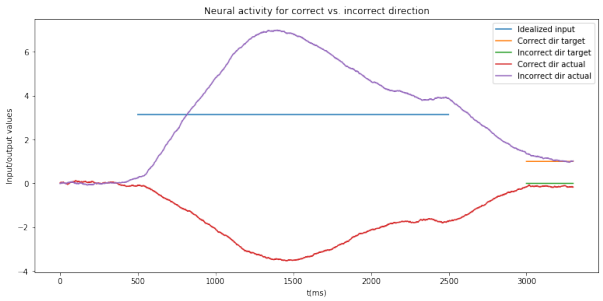

Figure 2: Neural activity of trained model through time

The plot shows the activations of the two output neurons. As we would hope, there's clear separation with high activation of the neuron corresponding to the target, and low activation of the neuron corresponding to the wrong direction choice. They converge towards the discrimination boundary, fairly quickly after the stimulus stops being shown (it's shown up until 2500ms, exactly when the drop-off begins). It seems like information about the stimulus is quickly "forgotten," but enough is retained to make a discrimination.

2.2 Model Dynamics

2.2.1 Neuron Activations

By looking at the activations through time of the recurrent neuron's, we can see how each neuron in the population responds to the input signal:

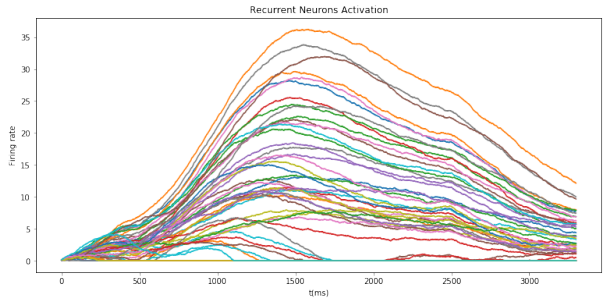

Figure 3: Model neuron activations

It looks like a large portion of the neuron's outputs respond to the input stimulus, each with roughly the same response curve (though different scaling). This makes sense since 80% of the neuron's are excitatory. One interesting question is why the response curves are all so similar – wouldn't it make sense for different neuron's to pick up on different things? I think there are two possible explanations:

1. We only trained on one very specific task, so there's no reason for neuron's to differentiate
2. The indices of the coherently-moving dots in the input were selected randomly, forcing symmetry

2.2.2 Neuron Connectivity

Looking at the network weights, we can clearly see the effect of constraining the network to satisfy Dale's law, with the intermediate layer weights partitioned based on whether they're excitatory or inhibitory.

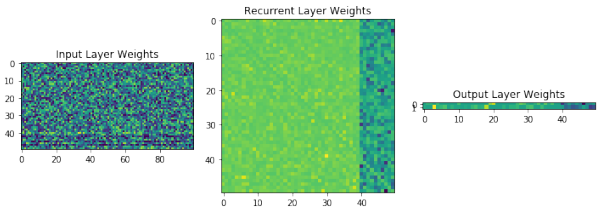

Figure 4: Network weights visualizations

It's also good to see that the input weight distribution doesn't have any structure. This is good because we constructed our input distribution to not be structured based on index.

2.2.3 Variable Waiting Time

To better understand the memory constraints of the model, I tried varying the length of the waiting time between the end of the stimulus and the decision time. I think it's extremely likely the way I trained the model over-fits, but this experiment showed that the framework is able to generalize to random waits. It looks like the model "hovers" around the target output value starting from the end of the shortest wait time it saw in training:

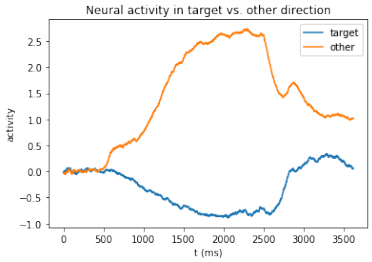

Figure 5: Response curve of model with random wait

2.3 Comparison to Shadlen and Newsome

Shadlen and Newsome's paper[4] included the following plot:

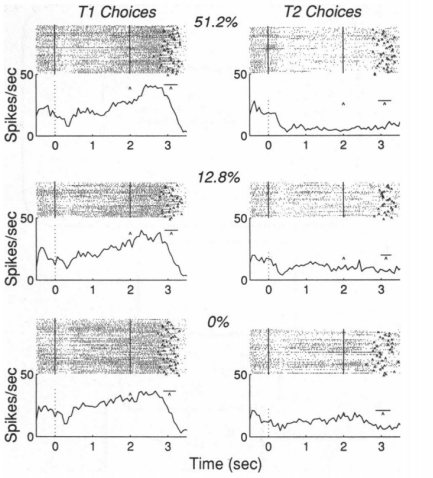

Figure 6: Original data: responses of a LIP neuron during task performance

What it shows is is there's a clear difference between neural responses when the monkey chose left and right, at different coherence levels. While we don't have access to spike data from our neuron, the trial-averaged firing rates correspond roughly with our earlier figure. 2 For a full comparison, we can look at the three coherence levels from the original figure:

45

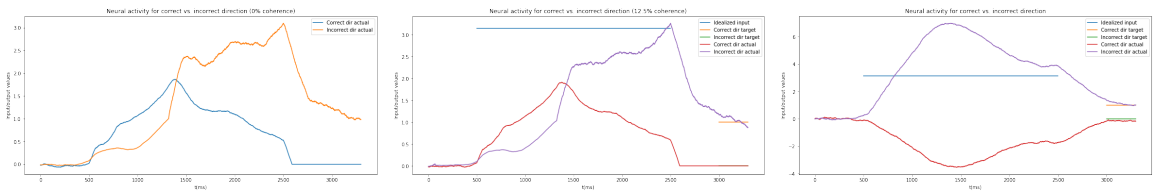

Figure 7: Response of neuron when choosing left versus right through time at 0%, 12.5%, and 51.2% coherence resp.

Note that the responses at 0% coherence and 12.5% coherence are exactly the same. This is because we sampled coherence levels from the interval $(0.125, 1)$, so there weren't any training examples covering the lower case. Since the unit activations are rectified linear, maybe the difference disappears due to thresholding.

Comparing to the Shadlen and Newsome paper, the response curves of the model look pretty different from the actual data. The rate of change appears to be larger in the model's neurons, especially at lower coherence levels. The real neuron's seem to change their firing rates more gradually, which might make sense biologically because inducing a really fast change might be less energy-efficient.

Another interesting thing is that the low-coherence examples don't follow the same response-curve pattern of either the high-coherence one or the original data. This trend didn't hold when I trained a networks at a fixed coherence level – in those cases, the responses looked more like the high-coherence case. This probably just means that the final network (which I trained on random coherence levels) converged even though it didn't generalize well. This is actually not that surprising, since high-coherence examples are much easier to learn, and the termination criteria was based on accuracy. Given more time, I'd try this again with a weighted termination criteria.

3 Discussion

Comparing the dynamics of was really interesting, but I think to draw more interesting conclusions I'd need to do more experiments. For example, we know that the monkeys were able to generalize to different coherence levels, so ideally we'd want a model that generalizes better than this final model. It'd also be interesting to look at how stable the responses are across training runs – anecdotally it seemed like high-coherence examples were easier to learn and resulted in more consistent models than low-coherence examples. That said, this was a super cool project and I really enjoyed working on it :)