Contents

Overview

Three chapters about populations of neurons in the hippocampus coordinating their activity on fine timescales, and how that coordination relates to the larger context of hippocampal inputs, outputs, and experimental measurements. In the first chapter, explore the lack of synchrony in timing of the brain rhythms underlying excitatory drive in hippocampal place cells, and the impact of those differences on the ability of distant place cells to coordinate reliably during population-wide coding events. In the second chapter, describe the development of a tool for detecting place cell population coding events and using the information they contain to manipulate brain activity or behavior experimentally. In the third chapter, describe the unexpected transition of retrosplenial cortex into a slow-wave-sleep like state during reward consumption, and the timing relationship between hippocampal activity and retrosplenial cortex during these events. Conclude with a discussion about coordinated encoding that ties the three

chapters together - how does the sequential nature of coding in hippocampus relate to coding in the afferents and efferents - do they also encode information in spike sequences? What experiments using real-time ensemble decoding would help answer those questions?

General Background

Hippocampal anatomy and physiology relevant to information coding

A brief reveiw of cellular organization in the hippocampus is given to help the reader stay oriented during discussions of electrode placement and traveling wave propagation. We also describe the freely moving rat's local field potential signatures and single-unit spiking properties, which are central to the rest of the thesis.

Hippocampal anatomy: cell layer and dendritic layers

The rat hippocampus is a curved, complex, three-dimensional structure most easily thought of as a sheet, about 10 by 7 mm face-on and 1mm thick, folded into a 'C' shape first along its short axis, and again into a larger 'C' along its long axis. The face of the sheet is fully tiled by primarily excitatory pyramidal neurons. Their cell bodies of these neurons are collected into a thin (about 0.1mm) band whithin the sheet's 1mm thickness.

Basal dendritic arbors extending upward from the cell bodies toward the skull (the hippocampus is inverted relative to cortex) for 0.25mm form the stratum oriens. Apical dendrites extend downward for 0.5mm forming the stratum radiatum, and then branch widely to form the stratum laconosum moleculare.

After folding and curling, the far end along the longer dimension of the sheet terminates near the septal nuclei, and the other travels backward and bends down to embed itself in temporal cortex. The long axis of the hippocampus is referred to as the septo-temporal axis.

Hippocampal laminar anatomy: CA1, CA3, entorhinal cortex; their connectivity

The first folding of the sheet described above divides the proximal-distal axis into two parts, named CA3 and CA1 by Lorente de Nó [1], (CA stands for "cornu ammonis", or ram's horn, which is reminiscent of the shape of the hippocampus in cross section).

CA3 derdrites receive most of their synaptic input from the dentate gyrus, entorhinal cortex, and the axons of other CA3 neurons. [TODO]. CA1 receives most of its input from CA3 and entorhinal cortex, but does not project to itself [TODO]. These patterns of inputs are more restricted than in many other parts of the cortex and have lead to computational models that take advantage of a layer with recurrent connections (CA3) connecting to one without (CA1), but none have wide acceptance. We will see later that our understanding of information processing within a single layer is incomplete, and this makes it difficult to speculate on the nature information transmission between areas.

The natural coordinate frame for the hippocampus then are the proximal-distal, septo-temporal, and basal-apical

Hippocampal place cells

Pyramidal cells increase their firing rate dramatically when rats enter a particular part of a maze, as originally described by O'Keefe [TODO]. The region of space eliciting spikes is that cells's place field. Typical place fields are between 20 and 80 centimeters long [TODO], and different neurons have different place fields; recording about 30 neurons is enough to find an 3 meter track without any gaps unrepresented by at least one place field. Spiking rates outside of a neuron's place field are quite low - often less than 0.1 Hz, and in-field rates peak rates are reliable across trials, typically 10-30 Hz; a degree of activity modulation hard to find outside of the sensory periphery.

The behavior of place fields in response to rotations and distortions of the maze is a matter of human curiosity responsible for the demise of fandastical numbers of laboratory rats. This large body of work can be summarized in terms of map displacement, rate remapping and global remapping. Rate remapping refers

to a change in place field peak firing rate and global remapping a displacement in place field location not necessarily in agreement with the displacements experienced at the same time by other place cells. The rules governing which sort of remapping will result from which types of maze manipulation are baroque, to the point that the neurons themselves disagree on the rules in particular instances and may fall at the same time in different directions [TODO]. But in genaral, minor changes to the appearance of the maze tend to elicit rate remapping [TODO octagon] while radical ones scramble the locations of place fields and produce global remapping [TODO octagon and teleport].

A simular rule of thumb applies to most maze and cue displacement results: place fields tend to follow what we would expect of a rat's top-level model of where he is. Minor enlargements of the maze produce proportional stretching and displacement of the place fields. A rotation of enough maze cues such that North is falsely recognized as the old West will produce the appropriate rotation of place fields with respect to the polse of the earth (and a lack of displacement with respect to the cues)[TODO].

Theta oscillations and multiunit spike phase

Electrodes in the hippocampus pick up a 7-10 Hz, large-amplitude rhythm, in addition to somatic spikes. This is called the theta rhythm [TODO vanderwolf?] This rhythm is present when animals are running or in a stationary, attentive state, and during REM sleep [TODO]. Theta oscillations can be found throughout CA1 and CA1, as well as in the dentate gyrus and entorhinal contex, and a host of other cortical and subcortical areas. Collectively the areas expressing theta are known as the Papez circuit [TODO]. Incidentally a lesion to any component of the Papez circuit produces in humans strong anterograde amnesia (as reported in the hippocampal patient H.M. [TODO Squire], thought in fact HM's entorhinal cortex was far more damaged than his hippocampus [TODO Italian slice guy]).

The mechanisms of theta's expression is being explored on two levels: the level of the generation of rhythms in the neurons, and the level of the translation of neural rhythms to extracellular currents [TODO Buzsaki 2002 review]. Neither level is completely understood, despite a large number of studies lesioning or pharmacologically silencing specifically excitatory or inhibitory neurons in various brain regions.

What is known is that two sources of theta can be pharmacologically distinguished by atropine and NMDA antagonists [TODO Buzsaki atropine, Vertes?], and these two components are associated with different intrinsic frequencies and different behavioral states. Type 1 theta is sensitive to atropine delivered systemically [TODO Buzsaki atropine] or directly to the medial septum [TODO]; its intrinsic frequency is about 10 Hz and it is natuarlly elicited by running. Type 2 theta is sensitive to disruption of glutamate signaling and is naturally elicited by stationary attention [TODO - and fact-check]. The importance of the septum in theta generation is strongly suggested by the fact that lesions to it nearly eliminate the appearance of theta in the local field potential throughout the rest of the brain. But this is not the whole story, as a dissected hippocampus in a dish will spontaneously express theta after application of acetycholine [TODO]. Interestingly, if that same hippocampus is pharmacologically divided in two along its long axis by application of the GABA agonist muscimol, then the two halves will oscillate at different frequencies, the septal end closer to 10 Hz and the temporal end closer to 6 Hz [TODO - and fact-check].

Quite a few facts are also known about the connection between the theta-rhythmic excitation of neurons and neuropil, and the appearance of theta to an electrode in the form of a local field potential. These details are important and interesting because of a connection between the phase of the oscillation and the activities of place cells (which we will discuss soon), and the phase of such an oscillation is such a finicky thing. For one, applying different filters to the recorded signal is enough to significantly advance or delay theta's apparent phase [TODO digital signal processing]. For another, an electrode's view of the rhythm is determined by the dipole environment local to it, and different parts of a neuron's dendritic tree express different dipoles at different phases of theta; so that the perceived phase of theta changes by half a cycle as an electrode is moved through the thickness of the hippocampus.

Buzsaki [TODO fix name accents] in particular has done a lot of mapping of the electrical sources of theta, first by lowering a single electrode in consistent intervals [TODO], and later by using probes with large numbers of evenly spaced contacts and applying the current-source-density technique [TODO], which predicts the spatial sources of current from structured voltage measurements. To summarize his findings and related ones from other labs, the Type 1 theta currents come mostly from inhibitory conductances near the soma [TODO fact-check], and Type 2 currents are mainly due to excitatory input to the apical

dendrites. These two sources do not agree in phase, and the effects of each drop off with distance of the recording electrode from the source. This is why electrodes with different placement will report different theta phases - they are respectively closer to different theta sources. The combined effects of the multiple sources is still fairly sinusoidal, as the sources are fairly sinusoidal and equal in frequency, and sine waves of equal frequency but different phase generally sum to a sine wave of a new phase [TODO waves book].

Theta phase precession & theta sequences

Sleep states, cortical rhythms, hippocampal-cortical interactions

Sleep stages, cortical EEG correlates (spindles, delta, theta)

Up-down states in vitro, frames of cortical spikes during sleep in vivo

Hippocampal ripples and sleep replay, wake replay

Hippocampal-cortical coordination of ripples/spindles

Spatial content in CA1 and visual cortex

Haskell

What is functional programming

Functional programming is both a style of programming and a set of language features designed to make functional programs natural to write and performant. That style revolves around two novel notions of what a function is. First, functions in a functional programming language are analogous to functions in math - relationships between inputs in a domain and return values in a range; they are guaranteed to return the same result from the same inputs. Second, functions are themselves 'normal values' - they can be passed as arguments to other functions, or returned from other functions as return values.

Languages like c allow a programmer to use functions in this way but do not make it easy. C is modeled closely on computer hardware, a context that emphasizes allocating memory and manipulating it. These operations are not 'functional' in the mathematical sense, because they involve 'doing' things - fetching memory blocks, performing some activity that is dependent on what was found in the memory block, and modifying the memory block. Functions in math are relationships between values in a domain and a range; these relationships are not dependent on the state of things like memory blocks, and the evaluation of a function's result in math does not impact the world in a way that changes other mathematical equations.

More natural support for functional programming is available in many higher-level languages, for instance python has the builtin functions map, which takes a function and a list and returns a list with the function applied to each element. We can write a function that modifies a single number and apply that function to a list of numbers using map.

```
  def topLimit(x):
      if x > 10:
          return 10
      else:
          return x

  print map(topLimit,[1,15,20,3,-2,5])
```

```
[1, 10, 10, 3, -2, 5]
```

The map function in Haskell looks very similar, except that there are no parentheses used in applying a function to its arguments. The first line defines the function $topLimit$ as a mapping from number to number, and the second line uses map to apply $topLimit$ to a list of numbers.

```
module Main where

  topLimit x
    | x > 10    = 10
    | otherwise = x

  main = print( map topLimit [1,15,20,3,-2,2] )
```

[1	10	10	3	-2	2]

What are types

Types are sets like $Integer$ or $String$ whose elements are values, like $\{0, 1, -1, 2, -2, ...\}$ and $\{'Greg', 'Hello\ neuron\backslash n', ...\}$ respectively [TODO]. Their role is to annotate data in a program, which would otherwise exist only as 0 s and 1 s whose identity would need to be tracked by the programmer. These annotations ensure that functions and data are paired in the correct way - for example preventing the programmer from attemping to take the square root of a string.

That basic motivation for types has been taken much further in the design of different programming languages. The nature of types is the main feature distinguishing programming languages [TODO]. Type systems divide languages into classes like dynamically typed languages (e.g. python, javascript, lisp), in which values can adopt a new type if the context demands it; and statically typed languages (e.g. c++, Java, Haskell), in which they can't. The term 'object oriented programming' refers to one style of type system, in which smaller types can be collected into a larger type called a 'class', and classes can be derived from a parent class [TODO]. The typical example is a Car class that has associated data, such as a $String$ to identify its owner, a pair of $Number$ s to indicate its location, and a $Number$ to indicate its speed. Another class $Truck$ could be derived from Car, and the $Truck$ type would inherit the Car s associated data. We can add additional associated data, like a $Number$ type to indicate the maximum payload and a $Number$ to store the tow rating of its trailer hitch. Individual Car s would be constructed in the program with concrete values in all the associated data fields. The goal in an object-oriented design is to build a heirarchy of sets (types, classes) that reflects the heirarchy of labels of objects. Internal properties of the objects being modeled are 'inside' the types, and running a program involves modifying these interval values. Consequently, the style is very noun-oriented [TODO - Yegee post].

An alternative foundation is to model types around logic, capturing ideas like mutual exclusion, associated data, and value-to-value relationships in the types. We need an example here:

```
data Coord = C Double Double  -- (1)
```

```
  ptA = C 0.1 0.1 :: Coord       -- (2)
  ptB = C 1.1 0.1 :: Coord
  ptC = C 0.5 2.1 :: Coord

  data SpikeRegion =             -- (3)
      Box       { corners :: (Coord,Coord), bChans :: (Int,Int)}
    | Polygon   { polyPoints :: [Coord],    pChans :: (Int,Int)}
    | Union     SpikeRegion SpikeRegion
    | Intersect SpikeRegion SpikeRegion
    | Diff      { rBase :: SpikeRegion, rDelete :: SpikeRegion}

regionA :: SpikeRegion           -- (4)
regionA = Box {corners = (ptA, ptB), bChans = (1,2)}

regionB :: SpikeRegion
regionB = Polygon { polyPoints = [ptA,ptB,ptC], pChans = (2,3)}

regionC :: SpikeRegion
regionC = Intersect regionA regionB
```

- (1): We first define our own set $Coord$, the set of all pairs of $Double$ s (real numbers). The C is a 'constructor' that can be used to build a $Coord$.
- (2): ptA, ptB and ptC are each particular $Coord$ s, built from C and a pair of real numbers.
- (3): We define a more complicated type, $SpikeRegion$, the set of amplitude regions that could be used to spike-sort tetrode data. A spike region could take one of five forms. The definition of each form is separated by a | and a new line. The Box constructor builds a Spike Region from a pair of $Coord$ s and the pair of electrode channels used for sorting. The terms 'corners' and 'bChans' here are not important - they are just labels for accessing the Box s internal data later. Alternatively, the $Polygon$ constructor can be appleid to a list of $Coord$ s. $Union$ is different; it is built from its constructor and a pair of other $SpikeRegion$ s. Its meaning in our program is: 'the space that is in either of the associated bounding regions'.
- (4): We define three different regions. The first is a rectangular region defined for tetrode channels 1 and 2. The second is a polygonal region defined by our three $Coord$ s on channels 2 and 3. The third is the intersection of the first two regions. $regionC$ is a typical sort of region used in manual cluster-cutting: the intersection of regions drawn on two projections. A region that uses a third projection to further restrict $regionC$ could be constructed simply as $Intersect\ regionC\ anotherRegion$.

To declare five mutually-exclusive sorts of regions in Python, we have two options, neithof of which are as intuitive as the Haskell type above.

First, we could write one class with an associated string that we set to 'box', 'polygon', etc, as well as the sum of all the associated data for all of the possible sorts of regions. This solution allows a programmer using a $SpikeRegion$ type to accidentally use value that does not belong with that sort of region. If we try to refer to one of the $Intersection$'s sub-regions when our region is a Box, our program will crash at some time in execution. An more serious issue would arise if data were used in a way that disagrees with the meaning of the type but does not cause a crash. It would be a very innocent mistake for a programmer to accidentally make use of the $bChans$ data when working with a $Union$ region, believing that they are taking the union of two projections instead of a union within the full 4 channels of a tetrode. This is a silent bug; the program will run but produce incorrect results. In the best case, a user will notice this and the bug will be fixed; in the worst case the error will propagate into the experimental conclusions.

Alternatively, we could use object-oriented style in Python to enfoce the invariant that Box and $Union$ and the others are associated with different sorts of data. The approach would be to define a $GenericRegion$ class with no assaciated data, and one associated 'stub' function for checking whether the generic region contains a spike (the stub will not be implemented - it's a placeholder). Then five new classes can be written for the five types of region. The derived Box class will have fields for the corners of the box and for the channels of the electrode. The derived $Intersection$ class will have two references to other $SpikeRegion$ s. This solution enforces our invariant nicely, but it forces the functions that use a subtype of $SpikeRegion$ to resolve the actual type; and it cost us a lot of boiler-plate code defining all of our subtypes. Aditionally, we have no way to keep track of whether additional classes will be derived in distant files that are part of the same program.

The mechanism of defining data types in Haskell allows (in fact, forces) the programmer to enumerate the variants of a type in one place, circumventing the issues discussed in the context of Python's types. Additionally, because the definition of our data is collected into one place, the compiler can know enough about our type check its use during compilation, before the program is ever run. It would be impossible for the programmer to accidentally refer to the sub-region of a $Polygon$ and produce an error in running code, for example, because the compiler would recognize this as a contradiction in terms ($Polygon$ has no associated sub-region data) and refuse to produce an program from the faulty code. The compiler

can also ensure than any function operating on $SpikeRegions$ has handled every possible case of $SpikeRegion$. This checking is a tremendous source of help to a programmer experimenting with changes in the data model. Without this checking, bringing the rest of a program into alignment with a change to a data definition is often done by running the program to the point of failure, finding and fixing one bug at a time.

Functions in Haskell are values and therefore have types. Their types indicate their domain and range. $length$ is a function from $[a]$ (list of any type) to $Integer$. We will also define a tetrode spike and a function for judging whether it is in a region.

```
length :: [a] -> Int
length =  undefined   -- we'll implement length soon

data TetrodeSpike = TS [Double]

regionContainsSpike :: SpikeRegion -> TetrodeSpike -> Bool
regionContainsSpike =  undefined
```

The type of $regionContainsSpike$ looks strange to normal programmers because there is not a clear distinction between arguments and return values. However there is something interesting happening. The $\rightarrow$ in a type is right associative, so $a \rightarrow b \rightarrow c$ is synonymous with $a \rightarrow (b \rightarrow c)$. $regionContainsSpike$ is in fact a function that takes a $SpikeRegion$ and returns a $(TetrodeSpike \rightarrow Bool)$, a function. We can apply this new function to a $TetrodeSpike$ to get a $Bool$. Surprisingly, all functions in Haskell are 'actually' functions of one argument. Multiple-function arguments can always be simulated in terms of single-argument functions that return new functions taking one argument fewer.

Declarative programming

Another side of the story of how Haskell facilitates writing code with fewer bugs is immutability - the notion that variables are fixed at a single value throughout a program. The rationalle is that the behavior of a program is much harder to reason about when variables are allowed to change value. All modern

languages have facilities for limiting the access of particular variables to specific other regions in the source code, to make this reasoning easier. Haskell goes to the extreme by forbiding any variable from changing.

Removing the ability to change a variable is obviously an enormous restriction on the flexability and usefullness of a language, and it's not immediately clear how many types of programs we could recover in this regime. In fact, this aspect of Haskell was very unflattering during its early history [TODO - lazy with class]. But a great deal of research and practice have resulted in new programming tools, styles and idioms that bridge the gap. After importing something called a monad from abstract mathematics, the notion of change could be integrated into the type system in a highly principled way [TODO Wadler], and now Haskell is an exceptionally good language for coordinating programs with moving parts and uncertainty from the data in the world.

But before resorting to monads, it is usefull to see how many values can be computed without making use of changing variables, using pure mathematical equations instead.

```
length :: [a]    -> Int
length    []     =  0
length    (x:xs) =  1 + length xs
```

In this listing, the function $length$ is defined by parts. The length of the empty list ([]) is 0. The length of any list that can be broken into its first element and a remainder is 1 more than the length of the remainder. Evaluating the right-hand-side in the non-empty-list case involves a recursive call to length on xs. The term $(x : xs)$ on the left-hand side is a way of naming different parts of the input value passed to the function that makes this kind of recursive definition (the definition of functions in general) convenient. These names only apply within the body of the function, they aren't permanently stored or passed into subfunctions. So when we recursively descend into length, the name xs is a different variable in each context, respectively being bound to a smaller sublist. This is easier to see with the names removed completely:

```
  length "Rat"           -- matches (R:"at")
= 1 + length "at"        -- matches (a:"t")
= 1 + 1 + length "t"     -- matches (t: [])
= 1 + 1 + 1 + length []  -- matches []
```

```
= 1 + 1 + 1 + 0
= 3
```

Using recursion, we described the length of the list, rather than computing it with iteration as we would in Python:

```
def listLength(x):
    nElem = 0
    for i in x:
        nElem = nElem + 1
    return nElem

print listLength('Hello iteration.')
```

```
16
```

The differences between declarative and traditional styles becomes more clear when we combine pieces together into larger programs. Let's try to take the product of the first 10 elemnts of the Fibbonacci sequence.

```
def listProduct(xs):
    acc = 1
    for n in xs:
        acc = acc * n;
    return acc

def makeFibbonacci(nElems):
    fibs = [1,1]
    for n in range(2,nElems):
        fibs.insert(n,fibs[n-1] + fibs[n-2])
    return fibs

print listProduct(makeFibbonacci(10))
```

```
122522400
```

This works, but it's a little unsatisfying to have to say how to build an array filled with Fibbonacci numbers, instead of describing the series itself, and that the definition is tangled up with an arbitrary detail, the

length of the list we want to produce. What would happen if we needed the 1000000000th number, and the array didn't fit it memory? Should we have used something more complicated like a generator, to produce Fibbonacci numbers without using increasing amounts of memory? This would force any users of $makeFibbonacci$ to consume streams. In Haskell, we use recursion to declare what Fibbonacci numbers are:

```
fibs :: [Integer]
fibs = [1,1] ++ (zipWith (+) fibs (tail fibs))
```

Translating this into English, $fibs$ is the list [1,1] followed by the list-wise sum of $fibs$ with $fibs$ less its first element. The surprising fact that we can define a value in terms of itself comes from the fact that, in math, we don't need to know the entire list $fibs$ in order to apply a function to $fibs$, we only need to know enough about $fibs$ to satisfy what will be used by the function. Here $fibs$ is defined as a seed and a function that only needs the seed in order to produce the next value. But this is an implementation detail. From the point of view of the programmer, we have our hands on a value $fibs$ that is indistinguiquasble from the entire infinite series. Trying to take the product of the list will take infinite time, not because our definition is infinite, but because we are doing something infinite. On the other hand, we can do something finite with something infinite, and it will take only finite time.

```
module Main where

fibs = [1,1] ++ (zipWith (+) fibs (tail fibs))
prodFibs n = product (take n fibs)
main = print(prodFibs 10)
```

```
122522400
```

This property of being able to apply functions to arguments when the arguments aren't fully known is called $laziness$, and it is one of the main features allowing declarative programming style and the combination of diverse software components into large programs. Combining infinite things with other infinite things in finite time, and decoupling mathematical models from details about how many elements to generate, are central to that. For an excellent discussion of functional programming's role as a glue layer between large components see Why Functional Programming Matters [2].

Avoiding bugs by lifting program logic into types, compiler catches mistakes early

Concurrency - difficulty of running multiple threads simultaneously

Software transactional memory

Coordinated information coding in a desynchronized network

Abstract

Brain areas involved in mnemonic and spatial processing are locked to an underlying 8-12 Hz oscillation known as the theta rhythm. Different layers of entorhinal cortex, subcortical areas, and hippocampal subfields are each maximally activated during different theta phases; even within field CA1, theta-locked excitation is offset in a gradient manner. In addition to pacing cell excitability, theta influences spatial information processing by organizing the timing of place cell ensembles into temporally precise sequences. We sought to determine the impact of theta timing offsets on the coordination of spatial representations in ensembles of place cells in different brain areas.

Along the CA1 septal-temporal axis, we found spatial information content to be synchronized to within 5 ms (TODO), despite a time offset in theta on the order of 30 ms (TODO). Offsets in excitability manifest as a subtle tendency for different brain areas to be more active at different phases of the theta cycle, but this is independent of the encoded spatial information. The same degree of inter-area synchrony is apparent in the hippocampus of stationary rats, which sporadically replays sequences of spatial locations.

This observed information synchrony in the context of desynchronized excitation provides a constraint for future models of fast-timescale space coding. Integrating these findings into a model of theta phase encoding can accoun for previous observations of diverse It also has implications for the integration of spatial signals in downstream structures, which may be befuddled by anything short of a coherent message from converging inputs.

This finding is at odds with prior models that strictly link theta phase to place cell spike timing. Adding a

baseline excitatory drive to each area according to that area's phase offset brings the population information into synchrony and accounts for anatomical gradations in spatial receptive field shape. These results show that fine-timescale information coding can be decoupled from underlying differences in timing of excitatory drive.

Introduction

CA1 place cell excitation is timed by 10Hz oscillation - theta rhythm

In many brain areas associated with spatial coding and episodic memory, neural activity is modulated by an 7-12 Hz oscillation called the theta rhythm [3,4]. The influence of theta on spatial and mnemonic information processing has been appreciated at two levels. On the global level, theta is thouht to coordinate activity between connected brain regions [5–8]. Locally, theta shapes the fine-timescale properties of information coding within brain areas, by way of theta phase precession [9].

Tension between hypothesized roles in gating communication channels and encoding

These two roles for theta oscillations are difficult to unify, because they make conflicting demands on the details of how neurons interact with the oscillation. Mizuseki et. al. [10] point out that time offsets theta oscillations across brain regions fail to match the sequence of time offsets predicted by monosynaptic delays between connected areas. For example, pikes should only take 10ms (TODO CHECK) to move from entorhinal cortext to the dentage gyrus, but the peak activation times of these two areas differs by about 97 ms (over half of a theta cycle). This phenomenon is acceptable to the global account of theta; it allows for the opening of 'temporal windows' of processing between sequential anatomical processing stages. But it is at odds with intuitive and formal [11,12] models of the fine timescale spiking of place cells. Place cells in dentate gyrus are generated from the summed spiking activity of their inputs, and are expected to follow the timing of their inputs closely (TODO CITE).

Theta as traveling wave, excitatory time offsets over hippocampal CA1

Colgin et. al. present data in support of a model associating particular phases of the theta oscillations of CA1 with the opening of specific communication channels to either CA3 or the entorhinal cortex [8]. The tension between theta's local and global roles is apparent here, as well. To the extent that CA3-CA1 and entorhinal-CA1 communication is limited to narrow windows of theta phase. Contrary to this, place coding in CA1 involves a smooth transition through cell ensembles that extends over much of the theta cycle [13,14].

Theta-dependent phenomena: locally modulated or globally synchronized?

[5] presented a novel finding and a supporting model that suggest a unification of global and local roles for the theta rhythm. Using large grids of tetrodes carefully positioned a uniform distance from the hippocampal cell layer, and sampling a large extent of the length of the hippocampus, they showed that the theta rhythm is not synchronous within hippocampal CA1. Instead, theta at the septal pole of CA1 are advanced in phase, theta in more posterior parts of CA1 are phase delayed, and theta measured inbetween has a graded delay. The combined activity of these delays roughly resembles a traveling wave with a peak of excitation that 'moves' down the hippocampal long axis once for every cycle of theta. The characteristics of this wave vary from cycle to cycle, but tend to have a spatial wavelength of 10mm (TODO check) and a preferred direction 45 degrees off of the septo-temporal axis.

Theta sequences: locally paced or globally synchronized?

If theta phase precession conforms to the anatomicaly sweeping of peak excitation, then theta sequences composed of sets of cells from different regions of CA1 would be similarly offset in time. The periodic replay of spatial sequences would begin slightly earlier in septal CA1 ensembles, and ensembles near intermediate CA1 would begin the same sequence about 30ms later (TODO check). Though this time shifting may seem to complicate attempts to square theta sequences with anatomical communication. However, it leads to an interesting prediction: that local regions of hippocampus begins a representation trajectory at

offset times. Because of this, a downstream structure observing a snapshot of the spiking activity across the whole hippocampus would see different parts of the track encoded at different anatomical locations.

Alternatively, place cells may not conform to the timing offsets suggested by the traveling theta wave, and the encoded information may be temporally synchronized over large anatomical distances, despite the presumed timing differences in their underlying drive.

We set out to measure the timing relationship between theta waves and place cell sequences in order to begin to unify anatomical and population-coding accounts of information timing. We characterized the impact of spatial tuning and anatomical distance on the cofiring of pairs of place cells, as well as the timing relationships of population-encoded trajectories recovered from anatomically distinct groups of cells, both across CA1 and between CA1 and CA3. We found that in most cases, timing offsets in theta sequences were significantly more synchronized than the temporally offset excitatory waves that modulate them. We suggest that information synchrony may be decoupled from the mechanisms that modulate excitation. This decoupling could be achived in a trivial way, by stipulating that phase precession begins and ends according to an underlying source that is in fact synchronized across hippocampus; or it could be achieved through an active mechanism that supplies extra excitation to the regions that would otherwise be temporally delayed by the traveling theta wave.

Materials & Methods

Subjects

All procedures were approved by the Committee on Animal Care at Massachusetts Institute of Technology and followed US National Institutes of Health guidelines. Tetrode arrays were assembled and implanted according to the procedure in Nguyen et. at. (TODO 2008) and Kloosterman et. al (TODO 2008). We made several modifications to the materials and proceduces to improve our multi-cell sampling. First, we glued several hundred half-inch pieces of 29 guage and 30 guage hypodermic tubing into rows about 6 mm long, then stacked and glued the rows together to form a honeycomb patterned jig, for organizing the tetrode guide-tubes that would eventually inhabit the microdrive. Second, we developed the ArtE

recording system (TODO detailed in Chapter 2) to run in parallel with our usual usual tetrode recording rig. The broader goals of the ArtE project are to enable real-time data analysis and feedback, but in this experiment we used it merely to increase the number of simultaneously recorded tetrodes.

Single-unit tetrode recording

Microdrive arrays were implanted with the center of the grid of tetrodes overlying dorsal CA1 (TODO A/P -4.0, M/L 3.5), spanning 3 mm of hippocampus in the septotemporal dimension and 1.5 mm proximo-distal. In two rats (TODO correct?), tetrodes were lowered into the pyramidal cell layer of CA1 over the course of 2 to 3 weeks and left there for several more weeks of recording. In two more rats, tetrodes were first lowered into CA1, and later a subset of those was moved further to record simultaneously from field CA3. In each cell layer, we sought to maximize the number of neurons recorded and to minimize within-experiment drift, so closely tracked the shape of sharp wave ripples (which undergo characteristic changes during approach to the cell layer) and later the amplitudes of burgeoning clusters. If either of these factors changed overnight to a degree greater than expected, the tetrode was retracted by 30 - 60 micrometers.

Behavioral training

Behavioral training began when nearly all tetrodes exhibited separable spike clusters, and consisted of rewarding rats for simply running back and forth on a curved 3.4 meter linear track, or running continuously counter-clockwise on a 3.4 meter long circular track, with rewards given for every 360 degrees of running for the first 3 laps and for every 270 degrees thereafter. Food deprivation began one or two days prior to the beginning of acquisition, with rats receiving 30 grams of food per day, adjusted up or down depending on the rat's motivation to run and level of comfort (assessed by the amount sleep taken before the running session). The target food-deprived weight was 80% of free-feeding weight, but we rarely achieved this without disrupting the sleep of the animals, so body weights tended to be 90% of the free-feeding weight or more, especially after rats learned the simple rules of the task. Additionally, we provided large rewards

throughout training (2-5 grams of wetted powdered rat chow per lap), to encourage the long stopping periods during which awake replay can be observed (TODO Foster, 2006). Under these conditions, rats run for about 20 laps or 30 minutes before becoming satiated and ignoring rewards.

Electrophysiological Characterization

Spikes and local field potentials were voltage buffered and recorded against a common white-matter reference, at 32 kHz and 2kHz respectively, and position was tracked at 15 Hz through a pair of alternating LED's mounted on the headstage, as in (TODO) Davidson et. al. (2009). Spikes were clustered manually using the custom program, xclust3 (M.A.W. TODO). Place fields were computed for each neuron as in Brown Sejnowski et al (TODO), by partitioning the track into 50 to 100 spatial bins, and dividing the number of spikes occurring with the rat in each spatial bin by the amount of time spent in that spatial bin, in each case only counting events when the rat was moving at least 10 cm/second around the track. Direction of running was also taken into account, allowing us to compute separate tuning curves for the two directions of running, which we label 'outbound' and 'inbound'.

To characterize the phase differences among tetrodes in CA1, a simple spatial traveling wave model was fit to the theta-frequency filtered LFP signals and the theta-filtered multiunit firing rate in turn, as in Lubenov and Spapas [5]. TODO expand on this.

Theta sequences

Two complementary techniques were used to assess the relationship between phase offsets between tetrodes and timing offsets in spatial information encoding. First, in CA1-only recordings, a pairwise regression was performed similar to that in Dragoi and Buzsaki (TODO 2006), measuring the dependence of short-timescale peak spike time differences on the distance between the peaks of that pair's place fields. We added a second inedpendent variable to this regression: the anatomical distance between each pair of place cells. The result is a model that predicts the average latency between any pair of cells, given that pair's place fields, that pair's anatomical separation, and the parameters of the traveling wave pattern of

phase offsets.

Second, Bayesian stimulus reconstruction (TODO Zhang et. al., 1998) was carried out independently using place cells from thre groups of tetrodes at the most septal end, the middle, or the most temporal end of the 3mm recording grid. Unlike the case for large populations of neurons, reconstructions from smaller anatomical subsets are considerably more noisy and do not reliably yield theta sequences. Session-averaged theta sequences were recovered by aligning the reconstructed position according to a shared theta phase and the rat's position on the track at that time. In both raw and session-averaged reconstruction cases, 2d autocorrelograms were taken to quantify the time-delay and space-delay between pairs of tetrode subgroups.

Results

Theta phase spatial properties and timing offsets: 20ms delay per mm

We first characterized the timing of the local-field potential (LFP) theta rhythm within a ~3mm long, 1.5mm wide strip dorsal CA1, in electrodes embedded near the pyramidal cell layer. A traveling wave model was fit to the theta-filtered and Hilbert-transformed signals from 16 to 24 tetrodes, in 0.25 second segments, resulting in a timecourse of traveling theta wave parameters (Figure 1 TODO link). We focus on the parameters that characterize the desynchronization: spatial wavelength, wave propagation direction, and temporal wavelength.

TODO

TODO

Each of these parameters vary on a short timescale, but are fairly consistent between animals when averaged across time. Theta frequency during running (which has been reported many times before) varies from 7 $\pm$ 3 (Hz $\pm$ standard deviations) TODO. The spatial wavelength is 10.2 mm $\pm$ with standard deviation of 5 mm, and the dominant propagation direction is 0.3 $\pm$ 20 degrees (mean $\pm$ standard deviation) TODO. The fit of the model was not sigficantly higher during running (r^{2}^ = 0.7 $\pm$ 0.3) than during stopping peri-

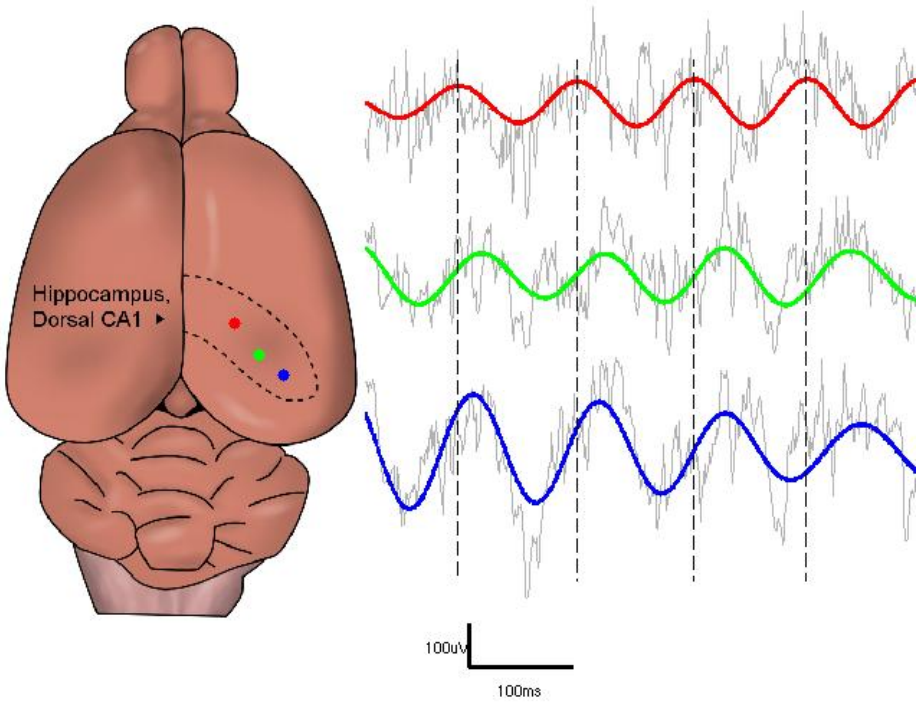

Figure 1: This is the caption for the next figure link (or table)

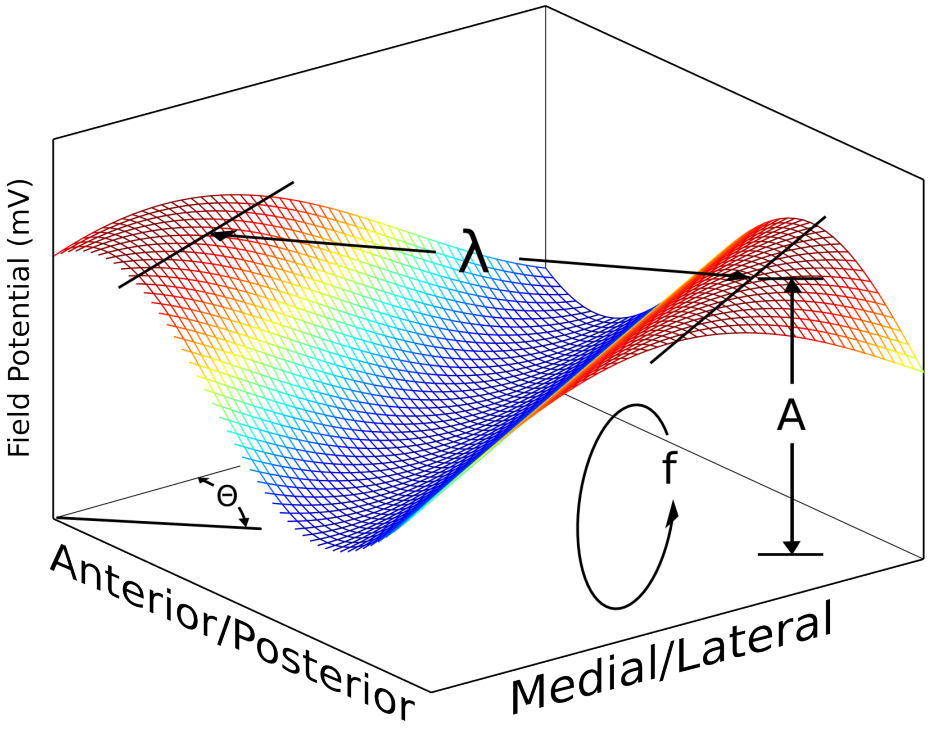

Figure 2: This is the caption for the next figure link (or table)

ods (0.75 ± 0.3) when theta amplitude is low, suggesting that traveling waves are a broad enough family to fit many patterns of data (in fact, a traveling wave model will perfectly fit a set of perfectly synchronized oscillators; the spatial wavelength in this case would be infinity).

TODO

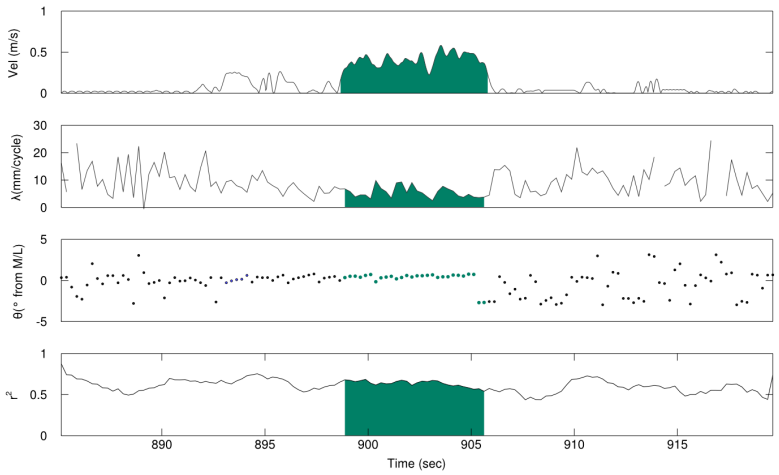

Figure 3: This is the caption for the next figure link (or table)

To put the parameters of the traveling wave into concrete terms, the peaks of theta in CA1 occur at earlier times in tetrodes closer to the midline, and at later times in more lateral tetrodes. Every millimeter of movement along this axis produces approximately 15 ± 5ms (TODO) of LFP delay.

Using multiunit firing rate (MUA) instead of LFP to estimate the underlying rhythm produced qualitatively similar results, with some quantitative differences. The goodness-of-fit across many tetrodes to a single traveling wave model was lower for MUA than LFP (r^2 = 0.9 ± 0.1 (LFP) vs. 0.7 ± 0.1 (MUA) (TODO FIX)), and the propagation direction was 30 ± 10 degrees (TODO), more anterior than the LFP. Importantly, the spatial wavelength was approximately half of that estimated from the LFP (5.2 ± 3.1 mm). These parameters together result in a 30 ± 10 ms wavefront delay for each millimeter of travel along the direction of wave propagation.

TODO figure

Due to the differences between LFP and MUA estimates, and the tendency for the phase of an individual tetrode to deviate from others unpredictably when tetrodes are near the pyramidal cell layer, we focus the rest of our analysis on the relationship between spike times and a single "reference theta".

Place cell pairs are synchronized across anatomical space

We directly measured the relationship between anatomical spacing and spike timing in pairs of place cells. If two cells with the same place field and phase precession profile are separated by a spatial interval corresponding to a 20ms (TODO) delay between theta peaks, two fast-timescale timing relationships are possible. Either phase precession is locked to to the local theta oscillation, and spikes from the cell 1mm 'downstream' with respect to the traveling wave will occur 20ms (TODO) later than those of the upstream cell (Figure 2A). Alternatively, if phase precession disregards the anatomical delays of theta phase, then spikes from the two cells should fire roughly in synchrony. Other timing relationships are possible of course, but it is not clear what they would imply mechanistically.

These predictions can be generalized beyond cell pairs with perfectly overlapping fields. Field separation will result in a theta-timescale timing shift due to phase precession. The virtual speed of the rat encoded in theta sequences is about 10 m/s, so a cell with a field peaking 0.5 meters beyond that of another cell will tend to spike 50 ms later, simply due to phase precession. If phase precession is paced against the local theta, then anatomical separation on the axis of the traveling wave should add to this delay linearly. We can use simple linear regression to estimate the independent effects of place field spacing and anatomical spacing on spike timing (Figure 2C).

TODO

Pooling across several recording sessions with large numbers of simultaneously recorded cells, anatomical distance contributes 0.7 $\times\pm\times$3.3 ms per medial-lateral mm. Pooling cell pairs across rats, we estimate each meter of place field distance to contribute 20 $\pm$ 6 ms (TODO) of delay and each mm of anatomical spacing along the traveling wave axis contributing 2 $\pm$ 5 ms (TODO). In other words, place cells fire with temporal delays that reflect spatial relationships on the track, and these spiking events are tightly

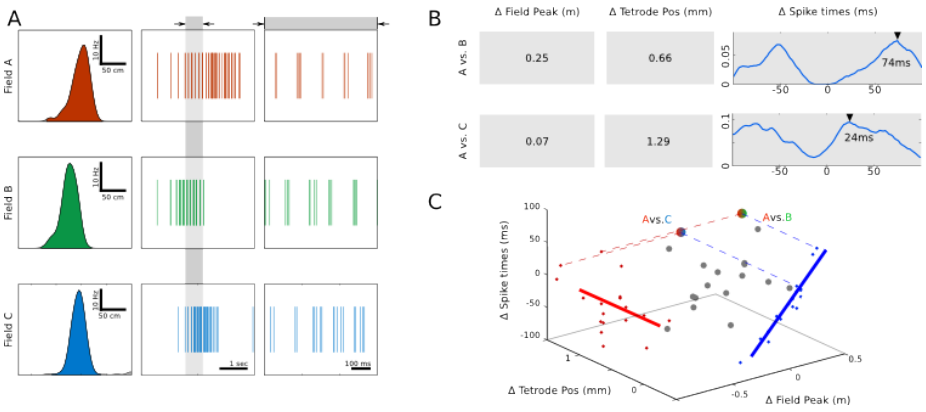

Figure 4: This is the caption for the next figure link (or table)

coordinated throughout the measured extent of CA1 (about 3 mm).

TODO

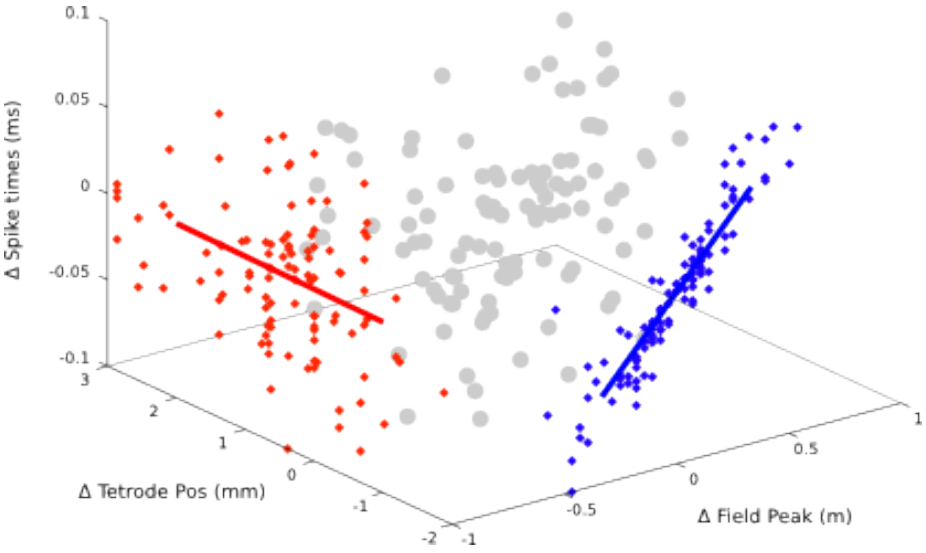

Figure 5: This is the caption for the next figure link (or table)

Table 2: This is the caption for the next table (or link)

Session	Anatomical (ms/mm)	Field (ms/m)	Offset (ms)	# of Pairs
Yolanda A	-7.0 $\pm$ 13.9	101.2 $\pm$ 20.0	7.3 $\pm$ 11.3	31
Yolanda B	-1.2 $\pm$ 16.4	199.1 $\pm$ 40.9	1.2 $\pm$ 11.0	18
Morpheus	0.9 $\pm$ 3.3	163.1 $\pm$ 20.7	-2.1 $\pm$ 5.2	38
Caillou	18.6 $\pm$ 12.8	198.1 $\pm$ 25.1	6.4 $\pm$ 7.9	19
Total	**0.7 $\pm$ 3.3**	**147.4 $\pm$ 14.2**	**-0.4 $\pm$ 3.5**	**106**

Ensemble theta sequences are synchronized

To assess the impact of anatomical distance on spatial representations from another angle, we turned to population decoding, which provides a direct view of theta seuences as well as spontaneous spatial replay events.

Within CA1, we partitioned cells into three groups according to the tetrode they were recorded on, then discarded the middle group, leaving two groups separated by a millimeter at their closest point, two millimeters on average. We then reconstructed the rat's location twice, once from each set of tetrodes, at a 10ms (TODO) temporal scale suitable for observing theta sequences. This division of units into independent groups drastically degrades the appearance of ongoing theta sequences, because the reconstruction process at such short timescales requires input from a large number of neurons. But clear theta sequences can be recovered by combining segments of the position reconstruction, aligned in time by peaks of the theta rhythm, and in space by the rat's current track position.

Using cross-correlations between the reconstructed theta sequences from different parts of hippocampus, we asked whether theta sequences are aligned with one another, or separated in time by a degree suggested by the timing of the local underlying theta rhythms. The peak of this cross correlation occurs when septal CA1 leads temporal CA1 by 10 $\pm$ 5 ms in time and lags 0.1 $\pm$ 0.01 (TODO) meters in space.

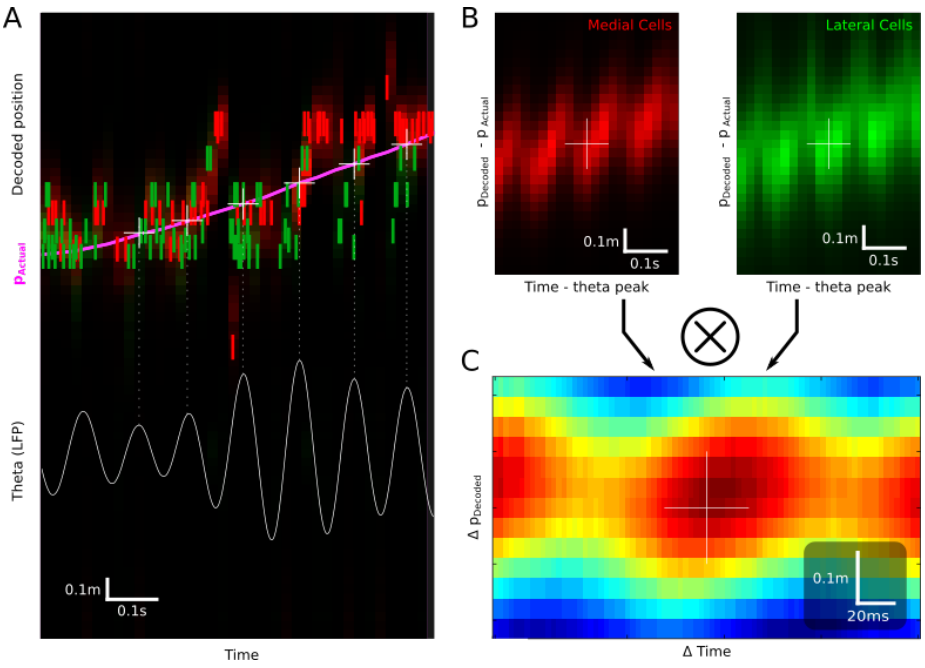

Figure 6: This is the caption for the next figure link (or table)

(TODO) Characterize time cross-correlation upward-diagonal skew.

Discussion

Theta traveling wave matches previous report: ~20ms/mm delay

Theta oscillations, the principal modulator of activity throughout the hippocampal network, are offset in time between CA3 and CA1; within CA1 theta oscillations are offset in time along the medial-lateral axis. Previous studies of theta oscillations generally rely on the simplifying assumption, justified by experimental evidence at the time (TODO cite Buzsaki linear probe), that theta within a givin brain region is synchronized (TODO, many examples). "What is the phase offset between CA3 and CA1" is no longer a well posed question; for any claim about CA1 theta phase, we must specify exactly which part of CA1 we are talking about, or else account for the phase gradient while generalizing across CA1.

Despite theta timing differences, information coding is synchronized

We reevaluate place cell's spiking relationship to theta in this context of unsynchronous theta. First we show that theta sequences, chains of place cell firing throught to be coordinated through their tight coupling to theta phase (TODO cite Mehta 2002), are tightly synchronized with each other, in spite of the desynchronization of the underlying theta rhythms. This information content synchronization exists between CA3 and CA1, which differ in theta timing by 10 to 40 ms (TODO), and between subsections of CA1 that differ in theta timing by by 10 to 15 ms (TODO).

Different parts of CA1 weakly preferentially carry most of the spike rate at different times

Model 1: Spatially graded, temporally constant compensating excitation

In Mehta & Wilson's (2002 TODO) model, theta phase controls the timing of place cell spikes in theta sequences. So if theta is desynchronized within CA1, how can theta sequences there be synchronized? We propose two models that could account for this. In the first, we propose a gradient of additional baseline excitation, greatest at the lateral pole of CA1 and least at the medial pole. Because spike times are locked to the moments when input excitation overcomes thata-rhythmic inhibition, extra excitation shifts these times to earlier phases. Applying greater excitation at points where theta is phase delayed would bring those otherwise-delayed spikes back into alignment with medial place cells, which experience less phase delay.

This model is not especially parsimonious, but it does make an testable prediction, which is borne out in the data. Under the excitatory input gradient gradient model in Mehta and Wilson (2002 TODO), additional uniform excitation should expose a greater extent of the subthreshold receptive field, resulting in longer place fields with more spikes in the 'anticipatory' part of the field and greater field asymmetry.

Model 2: Phase precession inherited from synchronized afferents

An alternative account for synchronized theta sequences throughout CA1 can be built around a less literal coupling between theta oscillations and phase precession. In this model, CA3 and entorhinal cortex (two of the known spatial-information carrying inputs to CA1) are modulated by a theta rhythm that is uniform within each respective area - the traditional view [10]. Theta recorded at any given point CA1 is inherrited from both of these areas and appear as a mixure of the two, in proportion to the relative strengths of the afferents at that point. But rather than organizing according to this local, mixed theta, CA1 spikes inherret their precise spike times directly from the spikes of the upstream brain areas. Without a traveling wave in CA3 or entorhinal cortex, all CA3 phase precession is synchronized and entorhinal cortex phase precession is synchronized; and for the sake of the model, CA3 phase precession is synchronized to entorhinal cortex phase precession. Now, the spikes of CA1 cells that are the result of either CA3 or entorhinal cortex input are aligned with respect to the spatial locations that the input units represent. What is offset in time is the phase-dependent modulation of spiking probability. Whatever the track position-by-phase relationship of a place cell, different phases of theta are associated with higher or lower spiking rates. In CA3, spike rates are higher during earlier phases of theta, and entorhinal cortex cells express higher firing rates at later phases.

This model accords with our findings in measuring place-cells: theta-timescale shifts in population firing rate, but maintained synchrony of the underlying information content. We shed the assumption of a perfectly balanced compensating excitation from the previous model, but pick up a new assumption: that positional information in entorhinal cortex is synchronized with that in CA3. This claim lacks empirical backing, and in fact it's not clear that such a timing comparison could even be made, because spatially selective neurons in entorhinal contex are grid cells (Hafting et. al. 2005 TODO), not place cells. However, theta phase precession is present (Hafting et. al. 2008 TODO) in most layer 2 entorhinal grid cells (these project mainly to CA3), but only sparsely in layer 3 grid cells (which project to CA1).

This does not necessarily contradict the Mehta and Wilson (2002 TODO) model, in which theta phase is not concretely linked to the spike times of any particular local group of neurons. On the contrary, if

the inhibitory oscillation that paces place cell spikes is derived from a single source like the GAGAergic theta cells of the septum (TODO Cite), then phase precession synchronization across brain areas would be expected, despite differences in the phases of the field potential theta recorded in those areas.

Information timing decoupled from bulk firing rate for globally coherent coding

(TODO)

Real time position decoding from populations of place cells

Abstract

Observational descriptions of hippocampal spatial encoding are outpacing our understanding of their underlying mechanisms and ties to behavior. The traditoinal manipulation techniques can not adequately target the richly choreographed spiking sequences increasingly recognized as an essential feature of spatial encoding. Some disruption specificity can be achieved by leveraging known statistical relationships between information content and the recency of spatial experience, and such experiments have provided the first evidence of a link between sequence replay and learning. But this method stops short of being able to distinguish among the diverse forms of spatial content known to be expressed in a single recording session.

A method of decoding spatial information content in real-time is needed. To do this, we are developing a multi-tetrode recording system focused on streaming representations of the processing stages typically used for offline spatial decoding: spike detection, neural source separation (cluster-cutting), position tracking, tuning curve extraction, and Bayesian stimulus reconstruction. We also extend a method for position reconstruction without human spike-sorting to operate in realtime. Our implementation makes critical use of Haskell, a programming language that aides software development by strictly separating a program's logic from its effects on program state, greatly simplifying code and eliminating large classes of common software bugs. We describe the capabilities and limits of our recording system, its implementation, and

routes for contributers to add functionality; and we survey the classes of questions that could benefit from real-time stimulus reconstruction and feedback.

Introduction

Theta sequences and sequence replay in place cells, phenomenology

Temporally compressed spike sequences are increasingly recognized as an essential feature of hippocampal encoding of space. Each increase in our ability to sample large numbers of cells in freely navigating rats has been accompanied by further support this claim [???].

Physiologists are aware of two forms of sequential encoding. The first occurs during active navigation. The majority ??? of spiking activity in the hippocampus is due to place cells ???, which spike only when the rat is within an approximately 1 meter span of the track particular to that place cell (the cell's "place field"). At any given time, the rat is within the partially overlapping place fields of many place cells. Rather than fire in random order, the spikes are arranged in precise sequences, with spikes from cells with place fields centered just behind the rat first, spikes from place fields centered ahead of the rat last, and a continuum between Reference 15. This sequence reflects the sequences of place field centers that the rat would encounter on the track, except it is sped up eight times and repeated once per cycle of the underlying 7-10 Hz "theta" oscillation in the local field potential Reference 16 [13].

A second form of sequenced spiking occurs while rats are paused on the track, consuming rewards or grooming. At these times, the hippocampus emits irregular, 100-500 ms bursts of local field potential "sharp wave-ripples"(SW-R's) and spiking activity, with spikes ordered in time according to the spatial ordering of their respective place fields [???]. These are known as 'sequence replay' events. Sequence replay often represents a track other than the track that the rat is currently running on ???; indeed it was first observed in sleeping rats ???.

Summary of semi-indiscriminant replay disruption studies

In contrast to the large number of studies exploring the phenomenology of theta sequences and sequence replay [???,??,??,14], interventional studies are rare, because any specific activity pattern of interest is embedded in a network also exhibiting off-target sequences, and sequences themselves are not apparent to the experimenter without extensive post-processing.

The content of sequence replay has a tendency to reflect recent experience, however. Some investigators using SW-R's as a trigger for immediate activity disruption have taken advantage of this to achieve some degree of stimulus selectivity in replay disruption. Ego-Stengel and Wilson ??? and Girardeau et. al. ??? used this paradigm to show that selective disruption of sleep sequence replay of one track can delay the acquisition of a spatial task on that track, relative to another track. And Jadhav et. al. ??? disrupted all awake sequence replay and showed that this impacts working memory performance.

Rationale for information-dependent replay manipulation

We would like to ask much more specific questions of sequence replay than whether or not it is needed for learning, of course. Does an individual replay reflect active cognitive processing of a route? Does a single theta sequence reflect a single conginitve sampling of a path ahead that may or may not lead to reward? Answering these questions means performing in realtime all of the processing steps from raw signal acquisition and sequence replay detection.

Here we report on two advances toward this goal. The first is a new system for simple acquisition, bandpass filtering, and multi-unit spike detection capable of running in tandem with our existing recording systems. The second is a proof-of-concept application that streams raw spike data and rat position data from the hard disk, performs source separation based on previously-determined waveform discrimation criteria, builds place field models, and performs the Bayesian inference to reveal sequence encoding, all in realtime.

The first system was written in a mix of c++ and Python, where data acquisition, signal processing, and networking are well-understood problems. The realtime decoding system presented more interesting

challenges, in terms modeling place fields, supporting infinite data streams, and concurrency. For this system, we turned to Haskell ???, a language optimized for ease of building composable abstractions Reference 2, through the marriage of a highly extensible static type system and functional purity. Haskell's type system enables the programmer to build custom times that capture the much of the intent of a model or algorithm, allowing the large classes of bugs to be eliminated by the compiler. Functional purity is an engineering discipline strictly enforced by Haskell that forbids variables to change their values during program execution. This restriction, thought apparently limiting, has many highly favorable consequences for managing complexity. These features fit together exceptionally well for designing highly concurrent programs, a notoriously difficult task in all programming languages [???].

Online replay decoding challenges

- Tracking rat, isolating units, computing place fields, and stimulus decoding all happen offline; need to happen online for streaming data
- Throughput requirements: must decode at least as quickly as data comes in
- Latency requirements: data -> decoding lag must be fast enough for behavioral feedback, preferably fast enough to disrupt an ongoing replay
- Asymptotic requirements: Decoding time must not increase with duration of experiment, or long experiments ruled out.
- Concurrency: Many sources of data (32 tetrodes, tracker, user input) all updating a single model

Minimizing human intervention: no time for manual spike sorting

Choosing the right language for implementation: Haskell

- Haskell types model domain very tightly, compiler checks program logic
- Types let compiler check whole codebase during code rewrites / code experiments
- Types tell runtime system which operations are pure (not-interacting), very nice property for concurrency

Materials and Methods

Backend signal acquisition and networking

Raw data is aquired simultaneously, at 32kHz, from 32 channels simultaneously on 2 NI PCI-6259 analog-to-digital converter cards (National Instruments), using the NIDaqMX c API. After passing data from the driver's memory to our program, samples are written into a circular buffer and passed through a 4th order Butterworth IIR filter. This choice of filter requires only two samples of history per channel, imposing a very short delay (< 1ms) between the collection of a given sample and subsequent processing. Spikes are detected by comparing each sample to a threshold, noting threshold crossings, and then waiting for one or a few cycles of acquisition until enough samples have been collected to meet the waveform length required by the user. Parameters like filter properties, spike threshold, and spike waveform length are initially set in a configuration file, and later modified through a networked API, so that the program can be run without an immediate graphical user interface - this is a preferable arrangement for a parallel, potentially distributed system, in which we may want a single command issued by the user to affect recording systems running on multiple computers.

Our previous recording system (AD. M.A.W. 1998 TODO date, cite?) also ran as a distributed collection of low-end acquisition computers receiving analog signals as input. In order to compare the recording quality and timing of our new system to the old system, we physically split sets of four analog inputs to two separat amplifiers - one serving each recording system. AD relies on hardware filtering of broadband data into the spike waveform band (300-6000 Hz) by a 3rd order Butterworth filter. ArtE reduces the hardware system requirements by digitally filtering a single broadband input into two signal bands - the spike band and the local field potential band (0.1 - 475 Hz), in each case using a digital filter designed to mirror the properites of AD's analog filters. Finally, using both systems in tandem required careful timebase coordination. Using standard computer system clocks is completely inadequate, as network delays between computers are on the order of several miliseconds, and can vary depending on system load. Instead, we route a diginal clock signal used to synchronize the AD computers into the ArtE system, and manually issue a counter resetting command to ArtE over the network while AD does the same for its own synchroization process.

This fairly hard-coded timebase integration is one problem that will have to be solved before ArtE can be used in isolatoin from AD, but not a very difficult one.

Isolated spike waveforms as well as downsampled, continuous local field potential signals are saved to disk in a different format from the one used in the rest of our cluster-cutting and analysis workflow. Until these tools are rewritten to work with the ArtE data format, we convert ArtE files into AD format, and continue with xclust (MAW - TODO) for cluster-cutting and MATLAB (Mathworks, Natick, MA TODO) for general analysis.

Offline position reconstruction

We compute fast timescale summaries of neural ensemble activicty through Bayesian stimulus decoding, as described in Zhang et. al. ???. Implementations of this procedure to date, including those used in our lab ??? are decidedly unfriendly to streaming, as they build models of place fields by sorting all spikes from the beginning of the recording session into the spatial bins partioning the track. This operation has time and space complexity linear in the number of recorded spikes, making it unsuitable for continuous streaming. Place field computations derived late in the recording would take longer than those computed at the beginning, and memory would be exhausted in finite time. These problems do not interfere with offline position decoding, because place fields may be computed once,slowly, and used repeatedly. The computation of many place fields that are synthesized into a single position estimate may be computed serially.

Online position reconstruction

- Manual spike sorting probably far too slow, use semi-automated or clusterless
- Choosing data structure for spike sorting & decoding with bounded memory & time use
- Likelihood functions have to be updated during experiment
  - By a lot of threads (~ 32 tetrodes * spike rate, plus current position)

  - Decoder also writes to likelihood function

- Use Haskell's concurrency library to coordinate many writing/reading threads

Modifying the place field models to update in constant time, rather than performing a linear-time recomputation for each incoming spike, is straightforward. Treatment of a large number of such models in parallel, rather than serially, is more challenging, because these models are ultimately combined into a single position estimate. Aditionally, the process of model update must run concurrently with graphic renderings, user input, and the regular computation of the position estimate itself.

To perform Bayesian decoding in realtime, we left the relative comfort of c++ and MATLAB for Haskell, on the promise that Haskell's type system and functional purity guarentees would simplify the static design of the model, and aid in the highly concurrent data flow.

Modeling place fields with Haskell data types

The phenomenology of place fields and the diversity of maze environments add complexity to the core notion of computing the place field, which is simply spike rate as a function of track position. These complexities are generally addressed in an ad-hoc way appropriate to each experiment. Due to the increased engineering effort involved in performing reconstruction in realtime, we aimed to anticipate as many of these issues as possible in the design of our stimulus model. We specify mazes as a collection of spatial bins, each with a user-assigned "outbound" direction and physical width. An animal's relationship to the environment is thus the combination of its relationship two each spatial bin in three respects, (1) physical proximity to the bin, (2) "outbound" or "inbound" heading with respect to the bin, and (3) position of the head with respect to the track width, either "centered" or "leaning over".

Matrix-based languages like MATLAB and c would suggest a representation of a place field as a three-dimentional array (with bin identity in the first dimension, the two possible heading directions in the second dimension, and head-overhang in the third dimension, for example). A particular position is referenced as an index into that array (for instance, the value at field[14,1,2] could correspond to a stored value related to

the 14th spatial bin, inbound running direction, head overhanging the edge). This is error prone. It requires the programmer to remember the mapping between maxrix dimenison and stimulus dimension, as well as a mapping between discreet values and stimulus levels (for example, than 1 means "inbound" and 2 means "outbound"). Naming the levels with variables does not solve the problem, because the variable "outboundLevel" and "headOverhanging" are both of the same type. Accidentally swapping the two (for example, writing field[14, headOverhanging, outboundDir] (TODO code formatting)) will result in code that compiles and runs, but produces incorrect output.

Haskell idioms are much safer. Instead of indexing into a matrix using three Integers, an idiomatic Haskell solution would be to use a tripple of indices with different types as the addressable space over which occupancy or a place field is defined. The use of distinct types for bin, direction, and alignment 'indices' allows the compiler to check the work of the programmer at every point where indexing happens. This small difference in approach eliminates a very large fraction of the bugs a codebase acquires as it changes and incorporates new features over time. If the matrix dimensionality were to change to accomodate a new feature, the Haskell compiler would enforce that this change is accounted for at every point where the code tries to access the matrix. This is in stark contrast to the flexible addressing of MATLAB and the untyped addressing of c/c++ arrays - in both of these cases the change may not result in any complaint from the program, but will instead happily deliver either noisy (or worse, unnaturally structured) data.

Our Haskell model of the track is the basis for the model of the rat's instantaneous "position", the model of accumulated time spend at each position (the "occupancy" function), and the model of a place field. At each point in time, we compute the animal's "position" as its relationship to each bin. In the simplest case, the bin that the rat occupies is given a score of 1.0, and all other bins scored 0.0; more typically, we assign graded scores to the bins according to their proximity to the rat; this method is favorable for smoothing noise in place field computations. For those time bins when the animal is running, this instantaneous position function added to an running tally of time spent at each position ("occupancy").

A place field is modeled in a similar mannar to the occupancy map - as a function from spatial bin to a number roughly equivalent to a "spike count" in that bin. Each time a neuron fires a spike, the instantaneous position map is added to the place field function accumulated so far. In the simple case when the

spatial bin containing the animal is assigned a 1.0, each spike adds an integer to that spatial bin in the place field. When position is taken by the more usual Gaussain-smoothed method, each spike adds a gaussian curve to the accumulated field. This procedure gives us constant-time, constant-memory spike-count functions that are simple to update, while respecting the complexity of the underlying behavior (the separate consideration for outbound vs. inbound running direction, and the consideration of whether the head is aligned with the track or leaning over the edge). When needed, the actual firing rate function can be computed, in constant time, by dividing the neuron's specific spike-rate function by the global occupancy function, at each spatial bin.

Managing concurrency and data streaming

To decode in realtime, we must simultaneously update place fields with information from new spikes, update the current postion of the rat, read the place fields and combine them into a single position estimate, handle user input, and render something to the screen. All of these operations interact with the same underlying data, and thus the problem is inherrently in a difficult programming regine (TODO CITE Concurrency difficulty paper). Due to strict enforcement of functional purity and immutable data, Haskell is in a special position to simplify concurrent computations. Indeed, the STM library provides a lockless concurrency scheme that allows multiple threads to simultaneously modify the same data if they wish (this genarally leads to data corruption), as long as the only variables modified are of a special type provisioned by the library, called TVars. STM tracks access to these veriables, detects when two threads have made conflicting changes, and roles both changes back, allowing the threads to attempt their modifications again.

We took advantage of the STM library to coordinate this concurrent read and write access to a single state value. This value was stored in one large TVar, which could be updated in the infrequent event of user input or the addition of new tetrodes. Within the enclosing state value, each place field is stored in its own TVar. In this scheme a very large number of spikes can be distributed to their respective place fields, and updates can be made without regard for the activity of other place field updates.

The problem is not amenable to processing by entirely independent threads ("embarassingly parallel"),

because the decoding step requires access to all place fields. In addition to place field updates, we accumulate spike-counts within short timewindows, and the decoding thread must reset all of these counts to zero each time a position estimate is produced. We group the resetting of all place field cell counts into a single atomic operation, to prevent the data inconsistencies that would inevitably arise if count-updating and count-resetting were interleaved. The grouping of actions into atomic blocks that can be retried upon collission is precisely the strength of the STM library that makes it so suitable for the structure of our decoding algorithm.

Clusterless decoding

It is often impractical to manually segment many tetrodes' spikes into putative single units, especially during a realtime experiment, when clusters need to be cut before any realtime feedback can be administered. Kloosterman et. al. [???] developed a method for Bayesian stimulus decoding from tetrode data without explicit spike sorting and provided an implementation in MATLAB. We extended this method by providing a new implementation that runs in bounded memory and time (Kloosterman's takes time and memory proportional to the number of spikes recorded, which makes it too slow for large-scale, long-running recordings). To restructure the alrogithm in a way that would continue to perform with potentially-infinite streams of data, we turned again to Haskell for its ease of use when working with custom data structures.

Kloosterman et. al.'s algorithm requires the comparison of recently-received spikes (the testing-set) to the amplitudes of all spikes received from the beginning of reconding (the training-set) along with the rat's track location during those training-set spikes. An estimate of the rat's position at testing-time is derived through Bayesian inference over a combination of the training-set spikes weighted by their amplitude-similarity to the testing-set spikes. A literal implementation of this algorithm has the disadvantage of making a larger and larger number of comparisons as the experiment progresses and the training-set grows. An obvious alternative would be to divide the space of spike amplitudes into a set of cubes, and update the cube into which each training-spike falls with the rat's current position. However, because amplitude space is four dimensional, the number of cubes required to tile amplitude space at a reasonable resulution is too large to store in computer memory. Sparse matricies and KD-trees are two good data structures

for holding multi-dimensional data in limited memory. We chose the reimplement clusterless decoding using the latter, at a slight performance penalty, because trees are somewhat more convenient to work with than matricies in Haskell. In order to accomotade new training-set spikes in bounded memory, when a new spike arrives less than some threshold distance from its nearest neighbor, the two are combined into one, and the payloads of the two (the place fields) are summed according to each point's weight.

Results

Decoding quality: theta sequences and replay

- Offline position reconstruction compared to online with clusters, online clusterless
- Tracking of rat's position
- Appearance of theta sequences
- Appearance of replay

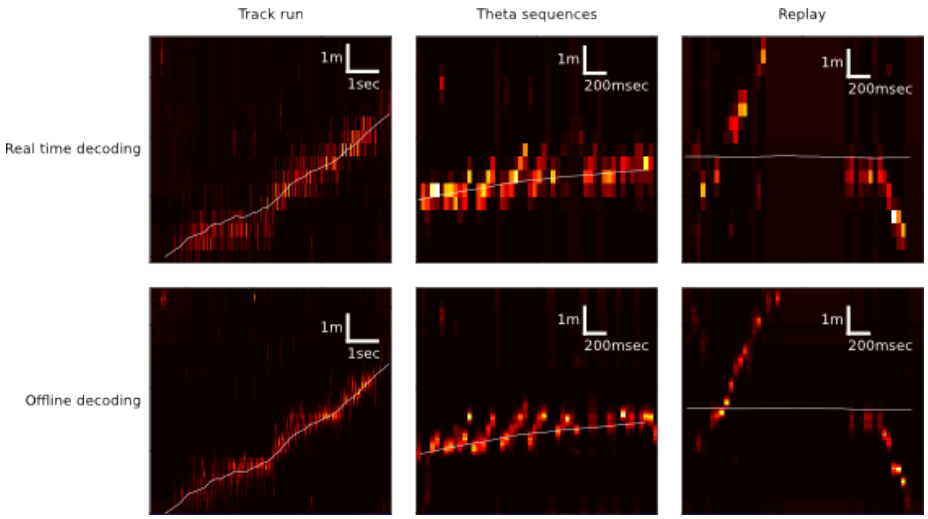

Figure 7: This is the caption for the next figure link (or table)

Decoding speed and realtime requeriments

Bugs, deadlocks, crashes and refactorings

Discussion

Recap: designed tool for decoding streaming place cell data

Remaining components needed to run experiments

- Networked rat tracker and track linearizer
- Online line-finding algorithm
- Combining estimates from multiple computers (for >16 tetrode case)

Experimental goals with sequence replay

Extension to non-hippocampal contexts

Retrosplenial slow-wave wake and interaction with hippocampus

Introduction

Cortico-hippocampal sleep interactions, possible role in memory

- Two phase consolidation model: encode at wake, burn-in during sleep
- HPC ripples correlated w/ sleep CTX sleep spindles - communication signature?
- Regular interval between hippocampal frame onset and cortical frame onset

Slow wave oscillations cleanly destinguish between sleeping and awake cortex

Ripples cleanly destinguish between 'online' and 'offline' hippocampus

Retrosplenial cortex unexpectedly follows HPC into SWS-like state during reward

Results

Characterizing slow-wave sleep (SWS) in cortex

- Examples of light sleep, spindles, frames and K-complexes in LFP, spiking
- Examples of deep sleep, frames and K-complexes in LFP, spiking
- Distribution of activity over all cortical electrodes
- Average up-state length, down-state length

Retrosplenial cortex enters SWS-like state during novelty / large rewards

- Examples
- Average up-state length, down-state length

RSC awake slow waves coordinate with hippocampal ripples

- 5-second window showing co-transition into SWS-like state (RSC frames, HPC ripples & replay)
- 200-second window showing behavioral-timescale relationship
- Cross-correlation of ripples & RSC frames similar between wake and SWS

RSC awake slow waves require large reward in well-trained rats

- Occur at most stopping points early in training
- After ~1 week, spontaneous frames & small-reward frames stop, but large-reward frames persist (for at least a month)

Anatomical restriction - nonparticipation in other cortical areas

- Simultaneously recorded somatosensory, motor, posterior parietal cortex have no frame-like activity (noticeable changes in spike rate or LFP) during RSC awake frames

Slow-wave wake not limited to times of sleepiness

- Awake SWS-like activity continues in both light and dark phases of light cycle
- Many SW's are flanked by fast running and chewing

Discussion

Recap: Awake slow-waves in RSC, coordinated with HPC, fully awake

In HPC-Cortex interaction, Online/offline vs. awake/asleep

Functional roles for HPC-Cortex coordination may apply to wake

New questions raised by SWW: mechanism and function

- New questions:
  - What other brain areas have SWWake? Papez circuit?
  - What's the mechanism for the switch from awake-aroused to SWW cortex?
  - What causes Slow Waves to traverse all of cortex during sleep, and not wake?
  - Is there information content in slow-wave frame spikes? Is it bounded by slow wave boundaries in an interesting way?

Materials & Methods

- 10 tetrodes in HPC, 10 tetrodes split between retrosplenial, somatosensory, motor, posterior parietal cortex
- Trained rats to run circular track for reward every 270 degrees CCW

Conclusion / Wrap-up

Brief summary of the role of populations of neurons in hippocampal spatial coding. Much more reliability in the timing of place cell spike sequences than there is in single cell measures like phase precession. We want to know if population sequences are an essential feature of coding, or just a means of denoising, and answering that question will involve manipulations that account for information content in and react to it in real time, as well as studies of how population sequences are interpreted by downstream cortical areas.

References

1. Lorente de Nó R (1934) Studies on the structure of the cerebral cortex. il. continuation of the study of the ammonic system. Journal für Psychologie und Neurologie.

2. Hughes J (1989) Why functional programming matters. The computer journal 32: 98–107.

3. Vanderwolf C (1969) Hippocampal electrical activity and voluntary movement in the rat. Electroencephalography and clinical neurophysiology 26: 407–418.

4. Buzsáki G (2002) Theta oscillations in the hippocampus. Neuron 33: 325–340.

5. Lubenov EV, Siapas AG (2009) Hippocampal theta oscillations are travelling waves. Nature 459: 534–539.

6. Jones MW, Wilson MA (2005) Theta rhythms coordinate hippocampal–prefrontal interactions in a spatial memory task. PLoS biology 3: e402.

7. Sirota A, Montgomery S, Fujisawa S, Isomura Y, Zugaro M, et al. (2008) Entrainment of neocortical neurons and gamma oscillations by the hippocampal theta rhythm. Neuron 60: 683–697.

8. Colgin LL, Denninger T, Fyhn M, Hafting T, Bonnevie T, et al. (2009) Frequency of gamma oscillations routes flow of information in the hippocampus. Nature 462: 353–357.

9. O'Keefe J, Recce ML (1993) Phase relationship between hippocampal place units and the eEG theta rhythm. Hippocampus 3: 317–330.

10. Mizuseki K, Sirota A, Pastalkova E, Buzsáki G (2009) Theta oscillations provide temporal windows for local circuit computation in the entorhinal-hippocampal loop. Neuron 64: 267–280.

11. Huxter J, Burgess N, O'Keefe J (2003) Independent rate and temporal coding in hippocampal pyramidal cells. Nature 425: 828–832.

12. Kamondi A, Acsády L, Wang X-J, Buzsáki G (1998) Theta oscillations in somata and dendrites of hippocampal pyramidal cells in vivo: Activity-dependent phase-precession of action potentials. Hippocampus 8: 244–261.

13. Foster DJ, Wilson MA (2007) Hippocampal theta sequences. Hippocampus 17: 1093–1099.

14. Gupta AS, Meer MA van der, Touretzky DS, Redish AD (2012) Segmentation of spatial experience by hippocampal theta sequences. Nature neuroscience 15: 1032–1039.

15. Skaggs WE, McNaughton BL, Wilson MA, Barnes CA (1996) Theta phase precession in hippocampal neuronal populations and the compression of temporal sequences. Hippocampus 6: 149–172.

16. Dragoi G, Buzsáki G (2006) Temporal encoding of place sequences by hippocampal cell assemblies. Neuron 50: 145–157.