Efficient Universal Computing Architectures for Decoding Neural Activity

Benjamin I. Rapoport[1,2,3], Lorenzo Turicchia[2], Woradorn Wattanapanitch[2], Thomas J. Davidson[4], Rahul Sarpeshkar[2]*

1 M.D.–Ph.D. Program, Harvard Medical School, Boston, Massachusetts, United States of America, **2** Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, United States of America, **3** Division of Health Sciences and Technology, Harvard University and Massachusetts Institute of Technology, Cambridge, Massachusetts, United States of America, **4** Department of Bioengineering, Stanford University School of Medicine, Stanford, California, United States of America

Abstract

The ability to decode neural activity into meaningful control signals for prosthetic devices is critical to the development of clinically useful brain– machine interfaces (BMIs). Such systems require input from tens to hundreds of brain-implanted recording electrodes in order to deliver robust and accurate performance; in serving that primary function they should also minimize power dissipation in order to avoid damaging neural tissue; and they should transmit data wirelessly in order to minimize the risk of infection associated with chronic, transcutaneous implants. Electronic architectures for brain– machine interfaces must therefore minimize size and power consumption, while maximizing the ability to compress data to be transmitted over limited-bandwidth wireless channels. Here we present a system of extremely low computational complexity, designed for real-time decoding of neural signals, and suited for highly scalable implantable systems. Our programmable architecture is an explicit implementation of a universal computing machine emulating the dynamics of a network of integrate-and-fire neurons; it requires no arithmetic operations except for counting, and decodes neural signals using only computationally inexpensive logic operations. The simplicity of this architecture does not compromise its ability to compress raw neural data by factors greater than 10^5. We describe a set of decoding algorithms based on this computational architecture, one designed to operate within an implanted system, minimizing its power consumption and data transmission bandwidth; and a complementary set of algorithms for learning, programming the decoder, and postprocessing the decoded output, designed to operate in an external, nonimplanted unit. The implementation of the implantable portion is estimated to require fewer than 5000 operations per second. A proof-of-concept, 32-channel field-programmable gate array (FPGA) implementation of this portion is consequently energy efficient. We validate the performance of our overall system by decoding electrophysiologic data from a behaving rodent.

Citation: Rapoport BI, Turicchia L, Wattanapanitch W, Davidson TJ, Sarpeshkar R (2012) Efficient Universal Computing Architectures for Decoding Neural Activity. PLoS ONE 7(9): e42492. doi:10.1371/journal.pone.0042492

Editor: Michal Zochowski, University of Michigan, United States of America

Received June 25, 2011; **Accepted** July 9, 2012; **Published** September 12, 2012

Funding: This work was supported by Grant NS056140 from the National Institutes of Health (www.nih.gov). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing Interests: The authors have declared that no competing interests exist.

* E-mail: rahuls@mit.edu

Introduction

Implantable Neural Decoding Systems for Brain–Machine Interfaces

Recent years have seen dramatic progress in the field of brain–machine interfaces, with implications for rehabilitation medicine and basic neuroscience [1–3]. One emerging goal is the development of an implantable system capable of recording and decoding neural signals, and wirelessly transmitting raw and processed neural data to external devices. Early versions of such systems have shown promise in developing prosthetic devices for paralyzed patients [4], retinal implants to restore sight to the blind [5,6], deep brain stimulators for treating Parkinson's disease and related disorders [7], and systems for predicting and preventing seizures [8]. Neural decoding has been essential to many of these systems, conferring the adaptive ability to learn to extract from neural data meaningful signals for controlling external devices in real time.

Electronics implanted in the brain must be sufficiently energy-efficient to dissipate very little power while operating, so as to avoid damaging neural tissue; conserving power also extends device lifetimes and reduces system size [9]. Yet experimental and clinical neuroscience demand ever-increasing bandwidth from such systems [10]: sampling rates on the order of 30 kHz per recording channel are commonly used in applications requiring discrimination of action potentials generated by individual cells, and while contemporary systems rarely record from more than hundreds of neurons simultaneously, much more extensive sampling will be required to probe the state of an entire human brain containing on the order of 10^{11} neurons. In this context, neural decoding can be viewed not only as a computational approach to extracting meaning from vast quantities of data [11], but also as a means of compressing such data. Neural decoding as a type of compression is 'lossy' in the formal sense that the decoding operation cannot be inverted to reproduce the original neural input signals, given only the decoder output. However, in neural prosthetics and related applications, not the neural signals but rather the information they encode is of primary interest. Indeed, the neural signals themselves can be viewed as constituting a redundant representation of underlying state information. In such contexts, the principal concern is not lost neurophysiologic information, but rather faithful reconstruction of encoded information, such as movement trajectories. The compression ratio, power consumption, and correlation of decoder output with encoded states and trajectories are relevant measures of performance. In previous work [12,13], we have shown that an implanted neural decoder can compress neural data by a factor of 100,000.

Considerable attention has been devoted to meeting the low-power operation constraint for brain implantation in the context of signal amplification [14,15], analog-to-digital conversion [16], power and data telemetry [17–20], neural stimulation [21–23], and overall low-power circuit and system architecture [9,24]. A small amount of work has also been conducted on power-efficient neural data compression [25]. However, almost no systematic effort has been devoted to the problem of power-efficient neural decoding [12,13].

Multiple approaches to neural decoding have been implemented by several research groups. As we have discussed in [12], nearly all of these have employed highly programmable algorithms, using software or microprocessors located outside the body [26–41]. An implantable, low-power decoder, designed to complement and integrate with existing approaches, would add the efficiency of embedded preprocessing options to the flexibility of a general-purpose external processor. As illustrated in Figure 1, our decoding architecture is designed to couple a power-efficient, bandwidth-reducing, implanted decoder, with an external unit that is less power-constrained and can therefore bear a computational load of greater complexity when postprocessing the decoded neural data. Being optimized for low power consumption, it sacrifices a small amount of algorithmic programmability—posing algorithmic challenges with which we deal in this paper—to reduce power consumption and physical size, and to facilitate inclusion of the decoder within an implanted unit.

As we have described in previous work [9,24], such an implanted unit consists of circuits for neural signal amplification, digitization, decoding, and near-field power and data telemetry. In association with an external unit that manages wireless power transfer and far-field data telemetry, such an implanted unit forms the electronic core of a brain– machine interface.

Biological and Universal Computing Primitives for Neural Decoding

Our computational architecture for neural decoding operates explicitly as a Turing-type universal computing machine, in which the decoding operation is programmed by selecting the rule array of the machine, which can also reprogram itself, resulting in an overall system that emulates the dynamics of a network of integrate-and-fire neurons. In contrast with existing approaches to neural decoding, this framework facilitates extreme power efficiency, *requiring no arithmetic operations except for counting.*

Our architecture decomposes the operation of neural signal decoding, allocating the computational load across two processing units: one implanted within the body, and therefore power-constrained, and the other located outside the body, and therefore less power-constrained. The overall architecture strategically imbalances the computational load of decoding in a way that leverages the relatively high computational power of the external unit to minimize power consumption in the implanted unit. Simultaneously, the system minimizes data throughput between the internal and external units in order to reduce the power costs of wireless communication between the two units.

Figure 1 shows the overall architecture of our neural decoding system. The architecture is decomposed into a set of operations implemented by Turing-type computing machines, shown as a collection of heads (data processing units) reading from and writing to a set of corresponding tapes (programs and data streams). Amplification and digitization of raw neural data, and decoding of that data, are performed by heads $\mathcal{N}$ and I, respectively, in the implanted unit. The computations of these two system components are streamed across a wireless data channel to an external unit, which performs more power-intensive external computations to postprocess the decoded output. In particular, further processing of the decoded data is performed externally by head E, and the final output of the system is reported by head O.

The core decoding function executed by the internal unit is an evaluation of the probability that the system is in each of its available states. At each time step, t, the internal unit reports a one-bit binary score $d_i(t)$, $i \in \{1 \dots n_s\}$ for each of the n_s possible states, based on neural data observed at each time step. The binary vector of scores, $\vec{d}(t)$, is processed by the external unit, which decides, on the basis of system history and *a priori* information, which single state is most probable. It then broadcasts its decision, for example to be used in controlling external devices.

The detailed operation of the internal unit is diagrammed in Figure 2, in which functional blocks are color-coded in accord with the scheme used in Figure 1. Neural inputs from an $(n=32)$-channel array are amplified and digitized, and the resulting digital bits are copied to the high-throughput neural data tape. As indicated by the red rectangle in Figure 2, these operations correspond to the function of the $\mathcal{N}$ head in Figure 1. The internal decoding computations implemented by the I head in Figure 1 are shown in detail within the green box in Figure 2. Digital circuits in this subsystem monitor each input channel during successive time windows of length t_w, counting the number of spikes whose amplitudes exceed channel-specific, programmable levels. The resulting spike counts are evaluated by a program stored in memory, which constitutes the core of the internal decoder. The program defines a set of rules, one or more for each possible state, that are configured during a learning period and that are then used to evaluate the scores d_i on the basis of the spike counts observed at each time step. Each rule also identifies the n_t channels that are most informative in decoding its corresponding state, and typically only these channels are used for computing d_i for state i. The spike-count thresholds for a given state-dependent rule are set through statistical learning such that the state may be discriminated from others with sensitivity and specificity that may be tuned to yield acceptable performance.

We use the terms *sensitivity*, *specificity*, and later *positive predictive value*, as they are classically used in the context of binary classification [42]. As we have cast the decoding problem, the decoder must implement a binary classification function at each time step, for each state, in deciding whether or not the observed neural firing pattern encodes that state; the results of these classifications are recorded as the components $d_i(t)$. According to this framework, decoder sensitivity with respect to a given state, s_i, is defined as the proportion of cases in which state s_i is correctly decoded by $d_i(t)=1$; single-channel sensitivity is defined analogously, by restricting decoder input to a particular channel. Similarly, decoder specificity with respect to state s_i is defined as the proportion of cases in which states other than s_i, or the collective state $\bar{s}_i$ (*not-s_i*), is correctly classified with respect to s_i by $d_i(t)=0$; single-channel specificity is also defined analogously. The positive predictive value of decoding with respect to state s_i is

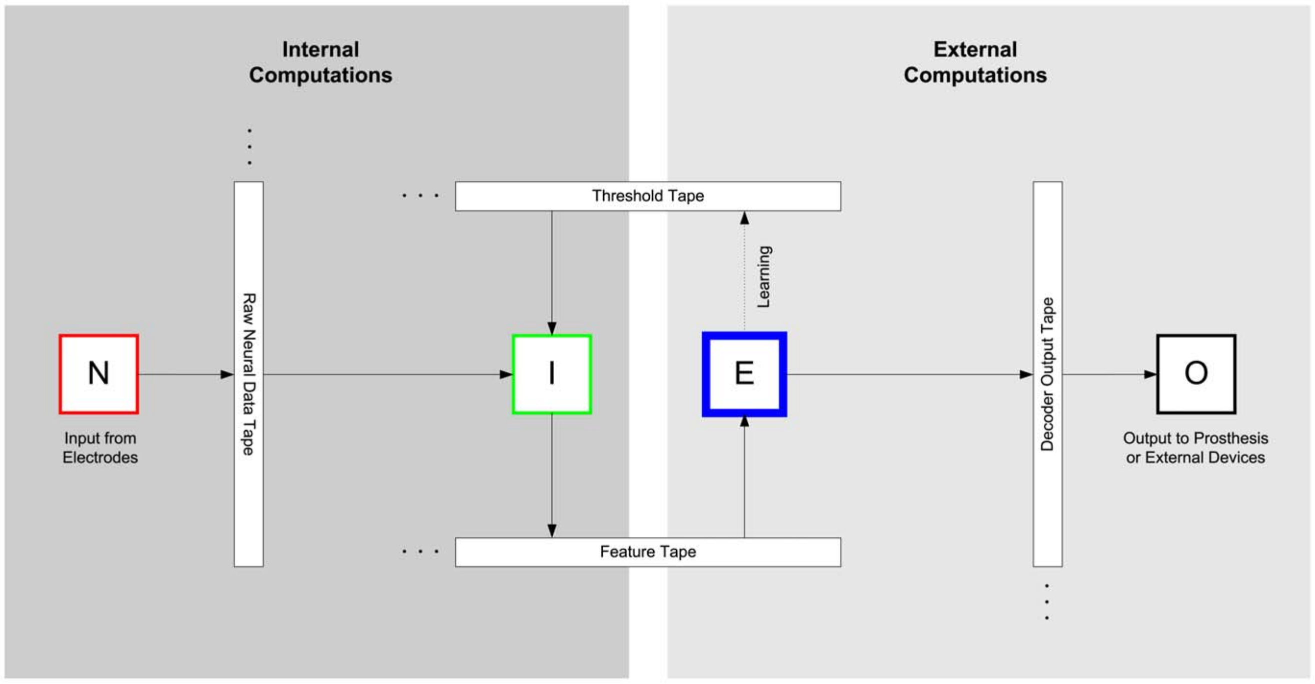

Figure 1. Universal Computing Architecture for Neural Decoding. The overall architecture of a neural decoding system is decomposed into a set of operations implemented by Turing-type computing machines, shown here as a collection of *heads* (data processing units) reading from and writing to a set of corresponding *tapes* (programs and data streams). Amplification and digitization of raw neural data, and decoding of that data, are performed by heads *N* and *I*, respectively, in a biologically implanted unit. The 'Internal Computations' of these two system components are streamed across a wireless data channel to an external unit, which performs more power-intensive 'External Computations' to post-process the decoded output. Further processing of the decoded data is performed externally by head *E*, and the final output of the system is reported by head *O*. The external system implements a learning algorithm that is used to write the program on the threshold tape, which is executed by the internal unit. doi:10.1371/journal.pone.0042492.g001

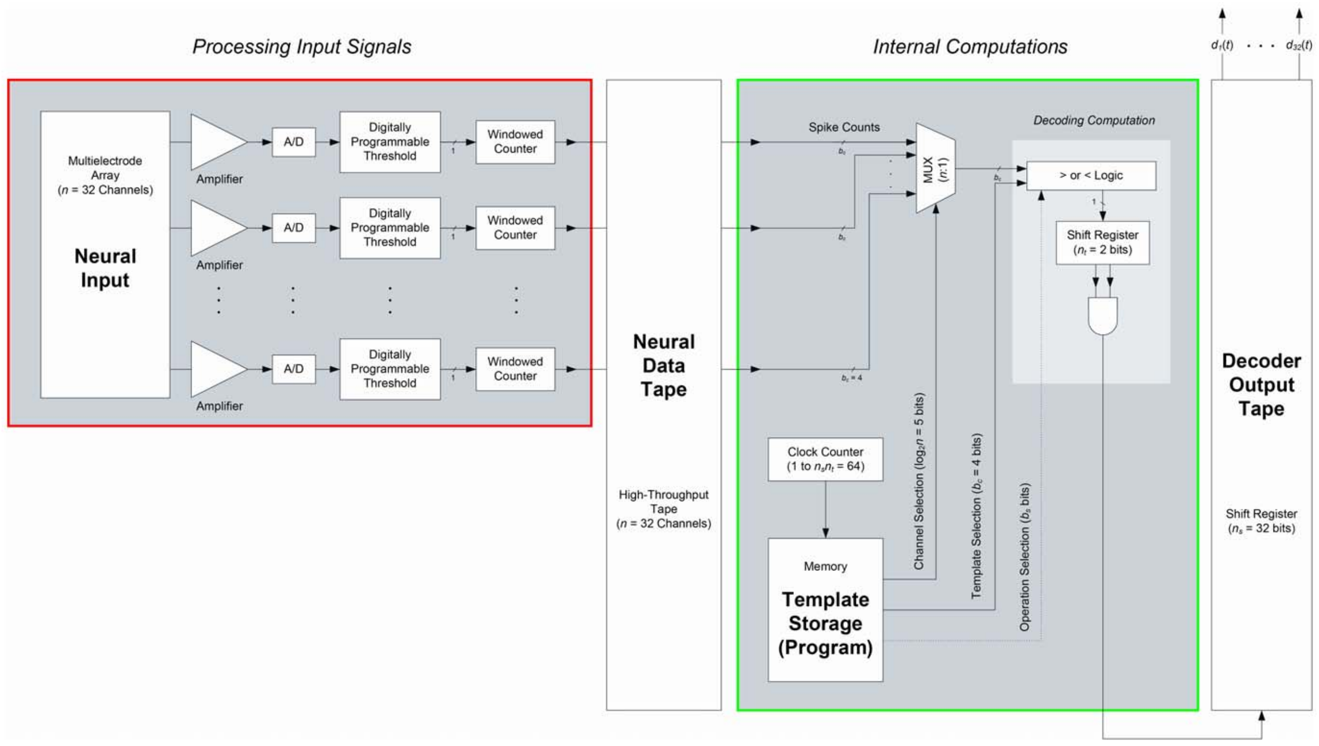
Processing Input Signals
Internal Computations
$d_1(t)$ · · · $d_{32}(t)$
Multielectrode Array (n = 32 Channels)
Neural Input
Amplifier
A/D
Digitally Programmable Threshold
Windowed Counter
Neural Data Tape
High-Throughput Tape (n = 32 Channels)
Spike Counts
b_c
$b_c = 4$
MUX (n:1)
Decoding Computation
> or < Logic
Shift Register (n_t = 2 bits)
Clock Counter (1 to $n_s n_t$ = 64)
Memory
Template Storage (Program)
Channel Selection ($\log_2 n$ = 5 bits)
Template Selection (b_c = 4 bits)
Operation Selection (b_s bits)
Decoder Output Tape
Shift Register (n_s = 32 bits)

Figure 2. Decoding Architecture. Block diagram of the low-power processing system of the internal component of our neural decoder, as implemented in one instantiation of our architecture. Functional blocks are color-coded in accord with the scheme used in Figure 1. doi:10.1371/journal.pone.0042492.g002

defined as the proportion of positive, $d_i(t)=1$, classification decisions that are correct; single-channel positive predictive value is again defined analogously. Finally, it is important to note that sensitivity and specificity are properties of the decoding function alone, independent of neural firing patterns, whereas positive predictive values depend on the distributions of recorded neural spikes [42].

The components of $\vec{d_i}(t)$ are written to the decoder output tape. The external unit examines the decoder output tape in a noncausal manner, postprocessing the decoded output generated by the internal unit to find a single most probable state. Our architecture permits a wide variety of postprocessing schemes, consistent with comparative studies of neural decoding algorithms and their underlying assumptions, which have formally and systematically demonstrated that movement smoothing is the most significant algorithmic factor influencing decoder performance [43]. We therefore implement a general-purpose decoding algorithm in the implanted system, while permitting application-specific choices in the external unit. Here we implement the postprocessing using a Viterbi algorithm.

Pattern matching algorithms conceptually related to the one implemented in our internal unit have previously been used to decode neuronal activity, notably in the context of memory replay during sleep [44,45], but until now the computational complexity of such approaches has limited their applicability to off-line, software-based implementations. Pattern matching systems have the useful property of being able to emulate receptive field structures—essential computational primitives of biological neurons—in a direct and intuitive way: they learn and store a set of *templates*, patterns corresponding to the activity of a given ensemble of neurons in response to a particular set of external states. Classical implementations of decoding by pattern matching function by comparing observed neuronal activity against stored templates (the system must store at least one template for each state to be decoded) and choosing a best match. This approach is typically computationally expensive for two reasons. First, the ability to quantify the degree to which observed neuronal activity matches a given template requires a defined metric, the value of which must be computed for every stored template at every time step of the decoder. And second, useful metrics themselves typically require computationally expensive operations, such as multiplication, root extraction, and division (or normalization). Computation of continuous-valued metrics in a digital context can also be accomplished only to a specified limit of precision.

The efficiency of application-specific digital microcontrollers and digital signal processors (DSPs) arises in large part from their ability to identify and prioritize the computational primitives, such as Fourier transformation or specific kinds of filtering, that are of greatest importance in particular applications [46]. In seeking a minimal digital decoding system whose operation is consistent with the computing primitives of biological neural networks, we have retained pattern matching as an approach to embedding neuronal receptive fields within the decoding architecture. However, we have reduced the template-matching metric to a set of rules in programmable logic. The structure of these rules as implemented in the example system described here results in a decoding architecture that behaves like a network of integrate-and-fire neurons. However, the programmability of the system and its explicitly rule-based architecture ensure that its scope encompasses even complex, multimodal receptive fields, but is not limited to such emulations [47]; the decoding architecture presented here is an example of a universal computing machine customized for neural decoding.

This paper is structured as follows: In this Introduction Section and in the Discussion Section, we address the implications of this work in the context of implantable brain– machine interfaces for clinical applications and basic neuroscience. We present results illustrating the performance of our neural decoding architecture in an initial Results Section, which includes a subsection discussing techniques for noise reduction. In the Methods Section we describe the acquisition and format of our input signals, and develop the decoding and smoothing algorithms themselves. A Methods subsection describes in detail a concrete, hardware implementation of the neural decoding architecture in a low-power field-programmable gate array (FPGA).

Results

Neural Decoding

We applied our neural decoding system to decode head-position trajectories from place cell ensemble activity in the hippocampus of a behaving rat. Place cells in rat hippocampus exhibit receptive fields tuned to specific locations in the environment [48–50]. Our system was able to decode temporal firing patterns of ensembles of such cells in real-time simulations using recorded neural data. Spike train inputs were derived from spike-sorted tetrode recordings from the hippocampus of a rat bidirectionally traversing a maze for food reward, as described in [51]. In the example described here, the training phase consisted of a 4.5-minute interval during which the rat traversed the entire maze once in each direction. This training interval directly preceded the 23.5-minute testing interval, during which the rat traversed the maze three times in each direction.

In the context of our place-cell– based position decoding problem, the states to be decoded, s_i, $i \in 1 \ldots n_s = m = 32$, are 32 equally sized, discrete sections of a one-dimensional track maze, constituting an arbitrary discretization of the continuous, linear, 10-meter track. The track was unbranched but contained several right-angle turns.

Figure 3 illustrates the encoding of position in our ensemble of $n=32$ place cells. The columns of the color-coded array are normalized representations of spike activity for the place cells in the ensemble, with bins (columns) corresponding to discretized positions in the one-dimensional track maze. The rows have been sorted based on the locations associated with maximal spike activity. Figure 3 shows that the receptive fields within this ensemble of neurons are distributed over the available one-dimensional space, forming a basis for effective decoding.

Figure 4 graphically displays the data structure, g, in which the decoding templates have been stored. We used $n_t=2$, $t_s=0.5$, and $t_p=0.25$ to compute the decoding templates, where n_t refers to the number of most informative channels used to decode each state, and t_s and t_p respectively denote the global minimum thresholds for the sensitivity and positive predictive value of state decoding, as described in the Methods Section. In Figure 4, g_{s_ij} is displayed as an $(n_s=m=32)\times(n=32)$ array, in which column s_i contains the template used to establish the spike count thresholds for state s_i. Hence, each column contains $n_t=2$ maximally informative elements, color-coded in one of $2^{b_c=4}$ shades of gray, with dark values corresponding to a threshold spike count of 7 spikes counted within the integration window, light gray corresponding to a threshold spike count of 1 spike counted, and white corresponding to channels that are not used in the decoding of that state. The decoding rules displayed graphically and as fine print in Figure 4 are reproduced in Table 1.

Figure 5 displays the performance of our decoding algorithms at both the spike-to-state and state-to-state stages. At each time step,

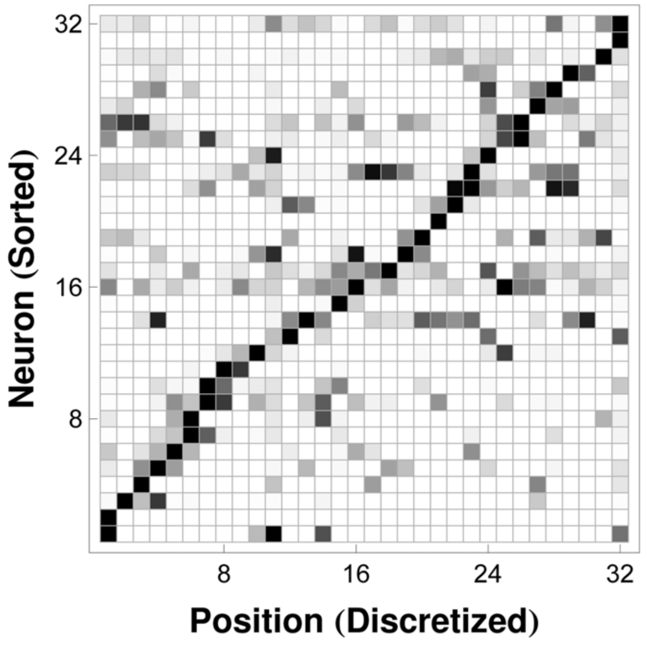

Figure 3. Encoding of Position by Place Cell Receptive Fields. Normalized spike rate for each of $n=32$ neurons in $n_s=m=32$ equal-length intervals along a one-dimensional track maze. Neurons (rows) have been sorted according to their positions of maximal activity to illustrate that the receptive fields of the place cells in this population cover the one-dimensional space of interest. Neuronal spike rates for each cell in each state (row elements) have been normalized to the highest spike rate (maximal row element) exhibited by the particular cell over all states. (Black: Maximal Spike Rate, White: Zero Spike Rate, Gray: Intermediate Spike Rates.).
doi:10.1371/journal.pone.0042492.g003

the output of the decoder is displayed vertically, with black pixels representing the ones in the binary vector $\vec{d}(t)$. The locations of those ones are observed to cluster along a trajectory reflecting the position of the rat in time, with stray ones due to noise and decoder error. The effects of noise and spike-to-state decoder errors are reduced by the state-to-state smoothing algorithm (in this case a Viterbi algorithm implemented by the external unit), whose output is also shown. Using a window length of $t_w=360$ ms, our decoded output matches the correct trajectory with a Pearson correlation coefficient of 0.70; the correlation rises to 0.94 when the time window for input spike counting is widened to $t_w=1440$ ms, as spike rate estimation is more accurate and smooth in longer time windows, enabling better interstate discriminability. Our performance is comparable to those of other implementations [29,52,53].

Compression Factor

Real-time decoding compresses neural data as it is acquired, by extracting only meaningful information of interest from a high-bandwidth, multichannel signal. For data acquired from an n-channel array and digitized to b_p bits of precision at s samples per second, our algorithm yields a compression factor of

$$c=\frac{nsb_p}{m/t_w}. \tag{1}$$

In our work, data are digitized to at least $b_p=8$ bits of precision, typically at a rate of 31,250 Hz; time windows are rarely shorter than $t_w=90$ ms. In an ($n=32$)-channel system decoding $m=32$ states, decoding therefore yields a compression factor of $c\geq 23{,}130$. In this example we have used $m=n$ to generate a conservative value for c; in practice, the number of states decoded is often fewer than the number of input channels, resulting in larger compression ratios.

Computational Efficiency

We explicitly calculate the computational efficiency of our decoding architecture in the Methods Section, following a detailed description of its operation. We find that the total computational load, L, associated with neural decoding, scales as

$$L=\beta m n_t f_w, \tag{2}$$

where $f_w\equiv\frac{1}{t_w}$, and $\beta=\frac{13}{2}$ for the implementation employed here, as explained in detail in the Methods Section. Thus, for $m=n_s=32$, $n_t=2$, and $f_w=\frac{1}{90\text{ ms}}$, $L\approx 4623$ operations per second, or approximately 4.7×10^{-3} MIPS (millions of instructions per second).

Noise Reduction and Cross-Validation

Input noise degrades the performance of the decoder, but our system has a number of mechanisms for mitigating the effects of noise. In considering the impact of noise on system performance, it is helpful to distinguish between correlated and uncorrelated noise, where the correlation is with reference to the collection of neural input signals, $\vec{h}(t)$, and $h_j(t), j\in\{1\dots n\}$, refers to the signal obtained from input channel j. More precisely, the covariance matrix for $\vec{h}(t)$, computed over a designated time interval, reflects the degree of correlation across input channels. In the context of neural signal recordings, cross-channel correlations (for electrodes spaced tens of micrometers apart) can in general arise from low-frequency components of the electroencephalogram (EEG) or from motion artifacts (movement of the recording array with respect to the brain, as may occur with head acceleration). Uncorrelated noise may be attributed to intrinsic properties of the recording system or to the biological signal itself, as discussed extensively in [9].

A useful feature of our internal decoding algorithm is its ability to explicitly suppress output noise and tune performance by adjusting t_w, the duration of the window over which neural data are integrated at each time step before making a prediction. In particular, t_w can be scaled in proportion to $\frac{1}{f_c}$, where f_c denotes a low-frequency cutoff in the noise spectrum. Lengthening t_w sacrifices system response speed for improved performance accuracy by integrating over more input data. Longer t_w intervals require more memory to store decoding templates of correspondingly higher resolution, so the total memory available to the implanted system (in our case, 18 kbits of RAM) places an upper bound on decoding performance. As we illustrate in the 'Neural Decoding' subsection of the Results section, we are able to reduce the effects of output noise (effectively representing aggregated correlated noise that affects all input channels simultaneously) by lengthening t_w. In particular, increasing t_w from 360 ms to 1440 ms results in an improvement in the correlation of decoded with observed output from 0.70 to 0.94.

The effects of correlated noise can also be suppressed in postprocessing, by the nonimplanted component of the system designated 'External Computations' in Figure 1. By virtue of its ability to use *a priori* constraints, such as those imposed by

C_1	7	3	5	20	3	27	24	25	16	16	14	10	20	17	15	24	14	1	11	14	1	11	11	4	7	7	8	15	15	10	1	1
θ_1	7	3	4	7	3	4	4	2	3	3	2	3	7	4	3	4	2	4	2	2	4	2	2	1	7	7	3	3	3	3	4	4
C_2	17	31	31	31	19	32	26	26	17	17	16	20		32	17	27	24	15	29	17	16	16	20	10	17		13	16	16	29	3	
θ_2	4	3	3	3	3	1	4	4	4	4	3	7		1	4	4	4	3	4	4	3	3	7	3	4		1	3	3	4	3	

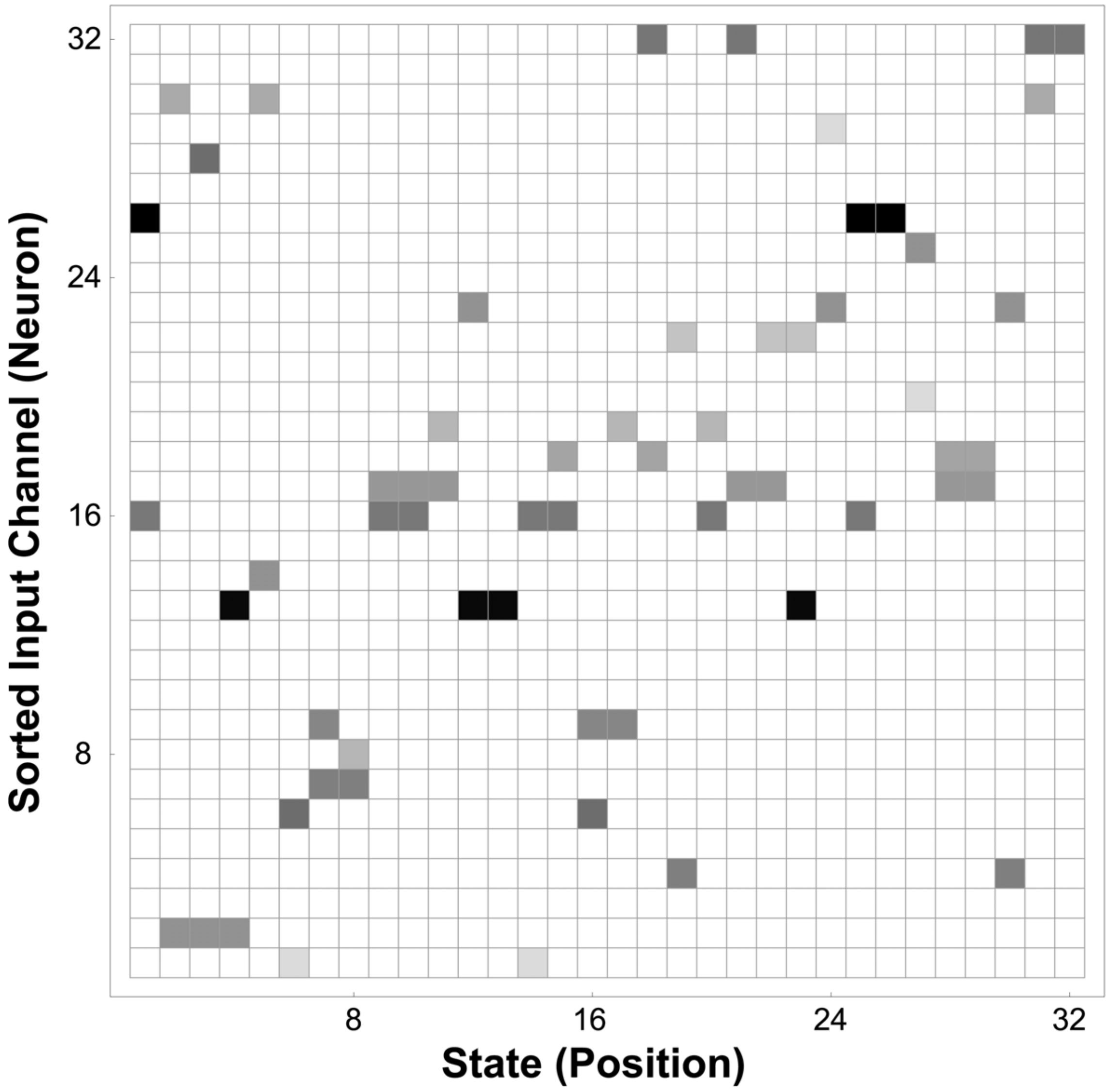

Figure 4. Decoder Logic Program: Finite-State Automaton Rules for Neural Decoding. Decoding template array g stored in system memory, resulting in the output shown in Figure 5, using the $n_t = 2$ most informative threshold values for each position state. Some elements of the rule table (three pair) are empty, with corresponding columns having fewer than $n_t = 2$ nonwhite elements, because the associated states had fewer than $n_t = 2$ channels able to satisfy $t_s = 0.5$ and $t_p = 0.25$, the minimum sensitivity and positive predictive value, respectively, for state decoding. (White: Unused, Light Gray: 1 Spike per 1440-ms Window, Black: 7 Spikes per 1440-ms Window.) Intuitively, this set of templates can be understood as the tape-reading rules for a Turing machine, whose symbols are generated by the time-windowed spike counts on neural input channels, and whose states correspond to a discretized set of position states encoded by the underlying neuronal populations. At each time step, the neural decoder scans down each column in the array to determine the states, if any, whose rules have been satisfied; the decoded output elements $d_i(t)$ are set to 1 for those states, and to 0 otherwise. The rules displayed graphically in the rectangular array are encoded numerically in the table displayed above the array (which is reproduced in Table 1). The columns of the table are aligned with the states in the array for which they contain decoding data, comprising the indices of the two most informative channels, C_1 and C_2, and the corresponding spike thresholds, θ_1 and θ_2.
doi:10.1371/journal.pone.0042492.g004

continuity as well as the physical properties and relationships of the states being decoded, the performance of the external system is not entirely limited by the statistical properties of the input signals and their associated noise levels.

Uncorrelated noise restricted to individual channels, associated with low channel-specific signal-to-noise ratios (SNRs), can be handled in the context of our decoding architecture by tuning the value of n_t on a channel-by-channel basis. The system is robust to noise even in the absence of such channel-by-channel tuning, however. In order to assess the robustness the decoder to channel noise, we constructed a cross-validation analysis, in which we evaluated the performance of the decoder when the input

Table 1. Tape-Reading Rules for a Turing Machine Decoder.

C_1	1	1	10	15	15	8	7	7	4	11	11	1	14	11	1	14	24	15	17	20	10	14	16	16	25	24	27	3	20	5	3	7
Θ_1	4	4	3	3	3	3	7	7	1	2	2	4	2	2	4	2	4	3	4	7	3	2	3	3	2	4	4	3	7	4	3	7
C_2		3	29	16	16	13		17	10	20	16	16	17	29	15	24	27	17	32		20	16	17	17	26	26	32	19	31	31	31	17
Θ_2		3	4	3	3	1		4	3	7	3	3	4	4	3	4	4	4	1		7	3	4	4	4	4	1	3	3	3	3	4

The decoding rules displayed graphically and as fine print in Figure 4 are reproduced here for clarity. The *j*th column of the table corresponds to the *j*th position state (as reflected by the vertical alignment in Figure 4), and contains decoding instructions for the *j*th state that are specified by the indices of the two most informative channels, C_1 and C_2, and their corresponding spike thresholds, Θ_1 and Θ_2. (As indicated in the context of Figure 4, some elements of the rule table are empty because the associated states had fewer than $n_t = 2$ channels able to satisfy the minimum sensitivity and positive predictive value for state decoding.)
doi:10.1371/journal.pone.0042492.t001

consisted of 31 of the 32 available channels of neural data, with the remaining channel replaced by noise. In this leave-one-out cross-validation scheme, we systematically replaced each of the 32 channels, one channel at a time, with a noise signal constructed to have the same spike rate as the original signal, but with spikes occurring at random times. Whereas the Pearson correlation between decoded (predicted) and experimentally observed animal position is 0.94 when all 32 channels contain valid neural signals, the mean correlation falls to 0.83 ± 0.13 (maximum 0.94) when only 31 channels contain valid neural signals and one is replaced by noise in the fashion described. Thus, although system performance is somewhat degraded in the presence of this form of uncorrelated, single-channel noise, the overall performance, as measured by a mean correlation of 0.83 with observed animal position, nevertheless remains consistent with the levels of performance described elsewhere in the literature [29,52,53].

Discussion

Generalizations

Our system for neural decoding, implemented as described here, can be generalized in several ways.

First, consider the combinatorial logic we use to interpret spike train data. We designed the decoding architecture to facilitate applying primitive operations derived from the structure of neuronal receptive fields. We model the receptive field of a neuron with respect to a set of states as the probability distribution of action potential firing as a function of state index. Discretizing this model to a finite level of precision transforms the frequency distribution to a histogram over states. The general problem of evaluating the degree to which input spike counts represent evidence of particular states has often been cast in terms of Bayesian analysis [54]. In the context of a discrete-time digital implementation, however, the most general analysis of a set $\vec{h}(t)$ of input spike counts can be conducted by comparing each element $h_j(t)$ to a finite set of thresholds. As the set of all possible composite results of all such comparisons (made over all channels and all states) is enumerable and finite, discrete-time digital decoding can be implemented by a finite set of threshold-comparison rules. In particular, although the template-matching algorithm we describe here consists of logical conjunction operators applied to single, unidirectional threshold crossings on designated sets of channels, our decoding architecture is capable of using much more general combinational logic in analyzing $\vec{h}(t)$. We note, further, that both the versatility of pattern matching [55] and the theoretical limitations of template matching [56] in the context of learning algorithms for computational neuroscience have been explored and reviewed in depth.

The combinational logic function we implement explicitly in this work, described by Equation 3, is appropriate when decoding

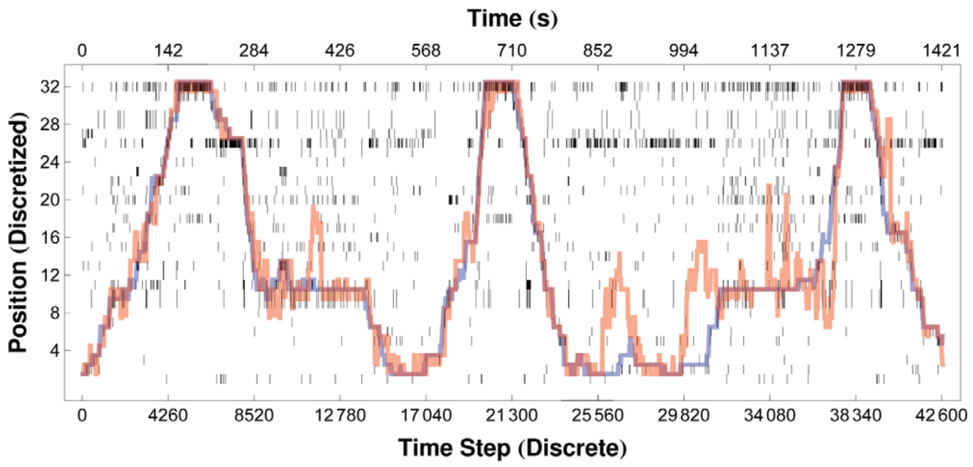

Figure 5. Decoder Output when Decoding Position from Hippocampal Place Cells. Our system decodes the location of a maze-roaming rat, from spike trains recorded from thirty-two hippocampal place cells. Raw output of the decoding algorithm, $\vec{d}(t)$, is shown as a raster array, with the output at each time step displayed as a vertical column of pixels (black pixels correspond to $\vec{d}_i(t) = 1$, white pixels to $\vec{d}_i(t) = 0$). Red lines show the trajectories obtained after applying our Viterbi algorithm to the raw decoder output, as described in the text. The actual trajectories of the rat are shown in blue. Decoding accuracy and decoder noise are affected by the length of the time window over which spikes are collected at each time step: $t_w = 2^{b_w = 4} \times 90$ ms ($t_w = 1440$ ms). Here the decoded trajectory matches the actual trajectory with a correlation coefficient of 0.94.
doi:10.1371/journal.pone.0042492.g005

from neuronal populations that encode information in receptive fields structured like those of the hippocampal place cells from which we demonstrate effective decoding here. The prototypical place cell has an approximately Gaussian, unimodal receptive field, which can be coarsely approximated by an impulse or threshold function that assumes a constant, nonzero value for states in which the cell is most active, and vanishes over all other states.

While such receptive-field encoding patterns are common, a more precise and more general model based on the one we describe here will have broader applicability. In a system with more memory or more logic circuits, a decoding scheme could use more logical functions with more conditions, operating on multiple upper- and lower-bound thresholds per channel. Such an approach could more smoothly model the state dependence of neuronal activity present in most receptive fields, through level-dependent conditions, where spike count levels are defined by pairs of upper- and lower-bound thresholds. This approach should be especially useful in decoding from multimodal receptive fields, or from input channels carrying unsorted multiunit activity, as in [57]. As indicated by the dashed 'Operation Selection' line in Figure 2, the most general decoder can employ different logical functions for each state.

Second, consider the information content of channel silence. Our decoding scheme takes action on the basis of observed spikes. Yet the absence of spikes also conveys information. A more general version of the system we describe here could exploit the information content of channel silence, constructing histograms and templates for time-windowed spike absence in analogy with those described here for observed spikes.

Third, consider cross-channel correlations in neural activity. The logical operations and probabilistic assumptions we have described here treat input channels, and indeed channel activity in each position, as independent. While it is convenient to assume such independence, neural activity across channels and positions may in general exhibit nonzero correlations. In a more elaborate version of our decoding system, such correlations could be exploited, for example by implementing combinational logic functions depending simultaneously on activity across multiple channels.

Interpretations

Our decoding system, programmed as we have described here, illustrates how a simple universal computing architecture can implement effective, biomimetic neural decoding. In particular, the rule-based decoding program we describe can be understood as implementing a two-layer network of digitized integrate-and-fire neurons. The first layer of this network consists of n input neurons, corresponding to the n input channels of the system. The second layer consists of m neurons, each connected to the n_t neurons in the first layer indicated by the decoding rules, and each synaptic weight set to the reciprocal of the corresponding threshold. Each second-layer neuron integrates its inputs for the duration of each time window, resetting to zero every t_w, and firing when it accumulates a value of n_t. The output of the neurons in this second layer constitute the decoder output we have designated $\vec{d}(t)$.

The particular scheme we describe here for programming our decoding architecture has an intuitive interpretation as an emulation of a network of integrate-and-fire neurons. However, the rule-based programming structure of the system is extremely versatile, and by no means limited to intuitive, biomimetic computations. Indeed, the computational universality of such rule-based systems has been explored and described in detail by several investigators, including Wolfram [47].

Methods

Ethics Statement

This study uses behavioral and neurophysiologic data acquired from a live rodent. The data were provided to the authors by collaborators (T. J. Davidson) at the Picower Institute for Learning and Memory at the Massachusetts Institute of Technology. Experimental protocols were designed in accordance with guidelines set forth by the Department of Comparative Medicine at the Massachusetts Institute of Technology in the 2010 Laboratory Animal Users' Handbook, and were consistent with the goals of minimizing suffering in all work involving nonhuman vertebrate animals. All experimental methods were approved by the Committee on Animal Care at the Massachusetts Institute of Technology under protocol 0505-032-08, 'Cortico-Hippocampal Ensemble Memory Formation.'

Spike Detection

Our neural decoder is designed to process full-bandwidth, multichannel neural signals in real time, by interfacing directly with the digitized output stream from an array of amplifiers and neural recording electrodes. Such a direct connection of the logic to the amplifier array facilitates great flexibility in the spike detection and decoding algorithms that the associated system can implement. In the present work we describe a system for detecting neuronal action potentials ('spikes') using a single-threshold method; individual thresholds are programmable, and can be fine-tuned channel by channel. In the tests of the decoding system described here, however, we used acausally filtered, prerecorded neural data, as described in this section.

More advanced spike-detection methods are also possible. One such approach would be to implement dual-threshold detection, with or without a refractory period. Another would be to use more computationally intensive detection criteria, such as nonlinear energy operators [58]. However, we have found that a basic single-threshold method works well in practice.

Importantly, we can also estimate the noise level on each channel in real time. Estimating the noise level is necessary in high-channel-count systems because in such systems it is typically desirable to set spike detection thresholds automatically. A common approach is to set the detection threshold to a multiple of the noise level, isolating spike events with approximate statistical confidence bounds.

The results presented here were obtained from neural data collected as follows: Spike train inputs were derived from recordings from the hippocampus of a rat, while the animal repeatedly, bidirectionally traversed a linear track for food reward, as described in [51]. Spikes from multiple neurons were recorded on each of several four-microwire tetrodes, and the spikes were manually sorted (by identifying isolated clusters in spike amplitude space). Decoder input waveforms were then constructed by thresholding the spikes observed in isolated clusters, with the additional constraint that only cells with clear spatial modulation were included.

Neural Decoding Algorithm

Overall Scheme of Operation. The objective in neural decoding is to infer an aggregate neural state, such as the intention to move a limb in a particular way, from signals sampled from a population of neurons encoding that state. The computational task of a neural decoding system operating in discrete time can therefore be described in the language of universal computing architectures, and our neural decoding architecture can be understood in terms of Turing-machine– like architectures with tapes and heads, as shown in Figure 1. At each time step, the system accumulates a sample of activity from a targeted neuronal population. This sample is interpreted as a symbol, or set of symbols, on the 'tape' read by our processor. The state of the system at that time step is inferred from the symbols by a discrete set of rules, stored in the memory of the system and used in reading the 'tape.'

The rules for symbol interpretation are derived using a statistical procedure that we describe here in detail. Intuitively, the procedure treats neural spike counts from each input channel as test statistics, used to evaluate the probability that the system is in each of the possible states. The distributions of these test statistics, conditioned on being in or out of each of the possible states, are approximated as histograms collected during a learning phase. Maximally informative spike-count values (thresholds) can therefore be derived and converted to logical rules involving only comparison operations.

Our decoding architecture operates as follows:

Neural spikes are detected on each of n input channels using digitally programmable thresholds, as described in the Methods Section under 'Spike Detection.' Spikes detected from each channel during an observation window of duration t_w (typically tens to hundreds of milliseconds) are registered in b_c-bit counters. At the end of each time window, the set of counter values is stored in an n-dimensional spike-count vector, $\vec{h}(t)$, describing neural activity across all channels. This vector is compared, component by component, to a set of m stored templates embodying the decoding rules. Each template g_i, $i \in \{1 \ldots m\}$ is also an n-dimensional vector, and its components g_{ij}, $j \in \{1 \ldots n\}$ constitute spike-count thresholds for corresponding channels $h_j(t)$ in $\vec{h}(t)$. The data structure g that contains the templates is therefore an $m \times n$ array, mnb_c bits in size; one such template array is illustrated in Figure 4. The set of templates contains at least one member for each of the n_s states to be decoded.

At the end of every time window, $\vec{h}(t)$ is compared component-wise to each of the m templates, g_{s_k}, $k \in \{1 \ldots m \geq n_s\}$, $s_k \in \{1 \ldots n_s\}$; we index the templates in this way to make clear that we allow states to be encoded by multiple templates. The decoder output vector, $\vec{d}(t)$, is then defined as follows:

$$d_{s_k} = \bigwedge_{j=1}^{n} h_j(t) > g_{s_k j}, \tag{3}$$

where the condition specified in Equation 3 is the logical *and* of all the component-wise comparisons between the spike counts in the input at time t, and the corresponding stored thresholds in the templates encoding a given state, s_k. Thus, at every time step, the decoder independently evaluates the strength of the evidence that the state s_k is encoded by the neural population over the time window $[t - t_w, t)$. The decoder output, $\vec{d}(t)$, is an m-dimensional binary vector whose components coarsely encode the corresponding estimates as probable, $d_i(t) = 1$, or improbable, $d_i(t) = 0$.

In practice, as discussed in the Methods Section under 'FPGA Implementation,' the decoder output can be computed by implementing a reduced version of the full n-input logical *and*. This is accomplished by storing the templates g_{s_k} not as n-component vectors, but as $(n_t < \frac{n}{2})$-component vectors containing the n_t most informative threshold values for each state, paired with pointers to their corresponding input channels. This simplified version of the full decoding algorithm is useful in practice because it reduces the amount of memory and the amount of logic, and consequently the total power, required to implement the algorithm. The reduction is possible because in most practical applications the number of highly informative channels for any state is considerably smaller than the full number of channels in the system. The decoding algorithm itself permits selection of any value $n_t \leq n$, and the selection $n_t = n$ would correspond to the full decoding algorithm as reflected in Equation 3. In simplifying the implementation of the algorithm to save memory, logic, and power, a value of $n_t < n$ can be selected empirically to ensure that decoding performance remains sufficiently comparable to that achieved in the case of $n_t = n$. As we discuss in the Results Section under 'Generalizations,' and as indicated by the dashed 'Operation Selection' line in Figure 2, a more general form of the decoder architecture can also store a pointer to a logical operation to apply to the indicated channel thresholds. This pointer would select among alternatives to the operation described in Equation 3, which we apply uniformly across all channels. In a more general decoder, a more flexible scheme along these lines might be appropriate, in which optimized, channel-specific logical operators are selectively applied to each channel.

The decoding algorithm itself makes no *a priori* assumptions about the nature of the encoded states. As a result, it may be equally well suited to decoding tasks involving continuous trajectories (controlling a computer mouse, for example, or a prosthetic limb) and to those involving discrete and discontinuous decisions (such as typing on a computer keyboard, controlling the click-state of a mouse, or selecting from among preprogrammed grip states of a prosthetic hand). Decoder output predictions of multiple simultaneous states, null predictions, and state-to-state transitions that are implausible due to physical or other constraints, are all handled by the smoothing algorithm described in the Methods Section under 'Trajectory Smoothing Algorithm.'

Computational Efficiency

In order to determine the total computational load associated with our decoding architecture, we account for the number of operations required in each computational frame, defined as a single cycle through the input channels, in which a single spike-count vector, $\vec{h}(t)$, is acquired. Each spike-count vector must be compared against n_t thresholds in each of m stored templates. As illustrated in Figure 2, each comparison is associated with the set of

$$\beta = 5 + \beta_L + \frac{1}{n_t} \tag{4}$$

operations enumerated in Table 2, where the parameter β denotes the total number of basic operations per frame, and the parameter β_L denotes the number of basic operations associated with the combinational logic used to compare incoming spike-count vectors to each stored template. In the implementation we demonstrate here, that logic consists of a single comparison, and so $\beta_L = 1$. In general, however, when more intricate logic is used to evaluate $\vec{h}(t)$, such as when a sum-of-products scheme is used to account for receptive fields with multimodal activity patterns (or, equivalently, input channels carrying multiunit spike activity), β_L may be greater than 1.

Spike-count vectors $\vec{h}(t)$ are acquired at frequency f_w, so the total computational load, as indicated in Equation 2, is $\beta m n_t f_w$.

Table 2. Basic Operations per Computational Frame of the Decoding Architecture.

Operation	Instances per Frame
Clock Counter ($\log_2(n_s n_t) = 6$ bits)	1
Memory Access	1
Multiplexer ($n = 32$) : 1	1
Comparison (1 bit)	$\beta_L = 1$
Binary Logic (*and*)	$\frac{1}{n_t} = \frac{1}{2}$
Shift Register ($n_t = 2$ bits)	1
Shift Register ($n_t = 2$ bits)	1
Total	$\beta = 5 + \beta_L + \frac{1}{n_t} = \frac{13}{2}$

Detailed accounting of the basic operations required in each computational frame of neural decoding, corresponding to the system block diagram of Figure 2. The parameter β denotes the total number of basic operations per frame. The parameter β_L denotes the number of basic operations associated with the combinational logic used to compare incoming spike-count vectors to each stored template.
doi:10.1371/journal.pone.0042492.t002

Computation of Templates

Decoding templates are constructed so as to be, in a statistical sense, both sensitive and specific to their associated states. By *sensitive* and *specific* we mean, respectively, that each template should detect its corresponding state with high probability when that state occurs, and that templates should discriminate well between states.

The decoding templates are computed based on data gathered during a training period, as shown in Figure 6, and are based on the approximation that the system state is stationary over the chosen time window, t_w. From the training data the decoder learns the tuning and firing rate properties of each input channel with respect to the defined set of states, in the following way:

The template construction algorithm begins by computing two sets, H^+ and H^-, of joint frequency histograms for neural spike activity over all pairs of channels and all states. The sets H^+ and H^- are organized by state and by channel. More precisely, $H^+_{s_i}$, $s_i \in \{1 \ldots m\}$, is a data structure containing n joint frequency histograms, $H^+_{s_i j}$, $j \in \{1 \ldots n\}$, of the number of spikes detected on channel j in a time interval of length t_w during which the system encodes state s_i. Similarly, $H^-_{s_i}$ stores corresponding histograms of the number of spikes detected on channel j in a time interval of length t_w during which the system encodes a state *other than* s_i. As described in the Methods Section under 'Overall Scheme of Operation,' a template g_{s_k} consist of a set of n thresholds, corresponding to the minimum number of spikes that must be detected on each channel during a single time window in order to meet the criteria for state s_i.

We use the following heuristic for setting these thresholds, designed to yield decoding performance meeting at least a tunable minimum quality. We begin by setting two parameters, $0 \le t_s, t_p \le 1$, representing a global minimum sensitivity and a global minimum positive predictive value, respectively. We then consider each pair of histograms, $H^{\pm}_{s_i j}$, and for every possible threshold value from 0 to b_s we compute the sensitivity and positive predictive value with which the threshold discriminates between s_i and $\bar{s}_i$, where the overbar denotes the set complement ($\bar{s}_i$ signifies *not*-s_i, any state other than s_i). The template value $g_{s_i j}$ is

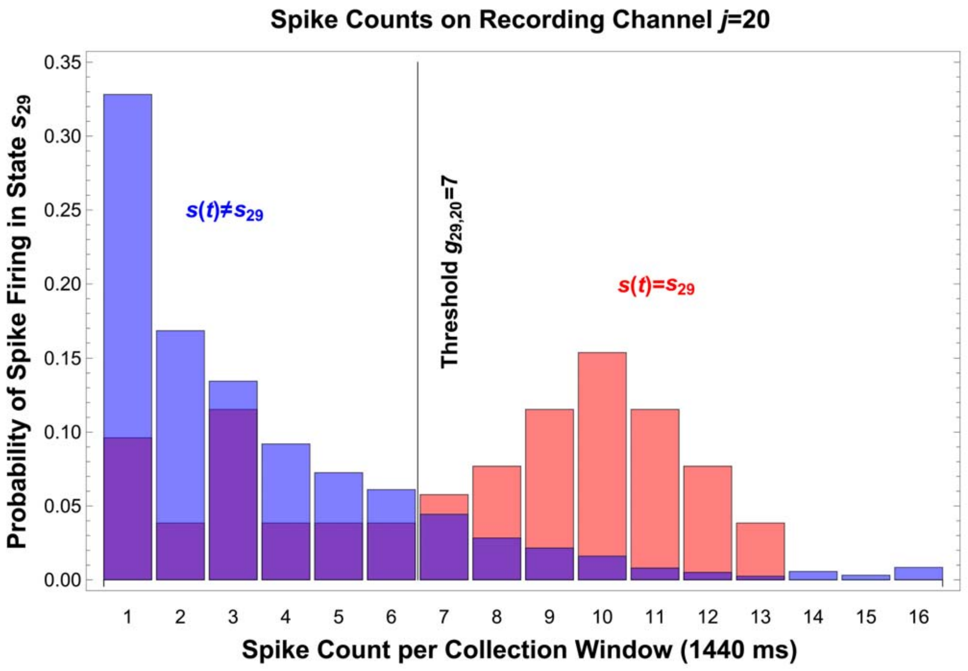

Figure 6. Histograms and Threshold. Histograms collected during the training phase of the decoding algorithm facilitate computation of thresholds for windowed spike activity, which are stored as templates in memory and used to discriminate between states. This histogram of spike activity, collected from recording channel 20, demonstrates that a threshold of $g_{29,20} = 7$ spikes per 16×90-ms window, on recording channel 20, is sensitive and specific for state s_{29} (Sensitivity: 0.63, Specificity: 0.85, Positive Predictive Value: 0.86). This threshold is written on the threshold tape used to program the internal unit of the decoder, and can be seen numerically in Figure 4 as the C_1 and θ_1 row entries of column 29, and graphically as the corresponding pixels in the rule array.
doi:10.1371/journal.pone.0042492.g006

set to the lowest threshold value yielding a discrimination sensitivity greater than or equal to t_s and simultaneously a positive predictive value greater than or equal to t_p. In the reduced implementations described in the Methods Section under 'Overall Scheme of Operation,' t_s and t_p can be tuned either globally or on a state-by-state basis until no template contains more than n_t predictive thresholds.

FPGA Implementation

Figure 2 shows our architecture for implementing the neural decoding algorithm described in the Methods Section under 'Overall Scheme of Operation.' As a practical demonstration of our computational architecture for neural decoding, we implemented our system in a low-power, $6\times6\times1\text{-mm}^3$, AGL060 FPGA of the Actel IGLOO Low-Power Flash family, which is tightly constrained by a number of parameters [59]. The measured power consumption of the decoding architecture, obtained as the difference between total power consumption of the FPGA (the product of operating current and the 1.2-V supply voltage) when it is programmed to contain or exclude the neural decoding architecture, is 537 μW. We note that the same architecture, implemented in an advanced-process application-specific integrated circuit (ASIC), should consume significantly less power, as even the lowest-power FPGAs are power-inefficient in comparison with custom ASICs. For example, custom digital architectures are often an order of magnitude more efficient than general-purpose architectures, such as those of FPGAs [9].

Our system accepts input from $n=32$ channels carrying digitized neural data from an array of recording electrodes and neural amplifiers [24]. The signal on each channel is thresholded to detect spikes, as described in the Methods Section under 'Spike Detection,' using a programmable comparator. Threshold-crossing events on each channel are registered by a $(b_c=4)$-bit windowed counter, and the counters on all channels reset synchronously after every interval of t_w. The set of counter values registered at the end of each time window constitute the input spike-count vector $\vec{h}(t)$ described in the Methods Section under 'Overall Scheme of Operation.'

As described in the Methods Sections under 'Overall Scheme of Operation' and 'Computation of Templates,' the decoding algorithm functions by evaluating a set of m logical expressions at each time step. The elements of these expressions are drawn from $\vec{h}(t)$ and the m templates, g_{s_i}; the templates thus implement programmable rules in combinational logic that are stored in the memory of the FPGA. In order to compress the decoding algorithm into the limited memory of our FPGA, we implement a reduced version of the template matching scheme, as described in the Methods Sections under 'Overall Scheme of Operation,' with $n_t=2$ rules per state; the thresholds and corresponding pointers are stored in RAM.

In our implementation we use $m=32$ reduced templates to decode $n_s=32$ states. At each time step, for each template, the $n_t=2$ counters identified by the template pointers are compared to the corresponding thresholds stored in the template. The logical results of these comparisons are stored in 1-bit shift registers, and each component of the decoder output, $d_{s_k}(t)$, is formed from the logical *and* of the associated $n_t=2$ registers. The m components of $\vec{d}(t)$ are saved in an $(n=32)$-bit shift register, and $\vec{d}(t)$ is transmitted every t_w, after all m templates have been applied.

Trajectory Smoothing Algorithm

Our decoding architecture separately implements routines based purely on *spike-to-state* correlation statistics, and routines based on physical or on other constraints on *state-to-state* transitions, independent of neuronal activity. This separation permits us to prioritize the former, the decoding computations most closely associated with raw neural activity and most critical for data compression, for execution within the implanted component of the system, and benefit from its inherent compression capabilities. The latter routines, described in this section, can then operate on the compressed data, outside the body. In the context of an implanted system with a small power budget, this scheme conserves power that would otherwise be spent in telemetry, by greatly reducing the bandwidth required for data transmission; and in computation, by offloading some of the decoding to the external components of the system. It also preserves algorithmic flexibility, as spike-to-state decoding, based on a generalized notion of neuronal receptive fields, is far less context-dependent than state-to-state decoding, which may employ a very broad range of constraints and context-specific statistical priors. For example, the priors employed to constrain letter-to-letter and word-to-word transitions in a typing task are very different from those used to smooth the trajectory of a prosthetic limb, yet both decoding problems can be addressed at the spike-to-state level by a pattern matching algorithm of the kind we describe here. The external unit, more accessible and less subject to power and size constraints, is better suited to tasks requiring more intense computation and frequent reprogramming.

We applied our decoding system to a problem involving the decoding of continuous movement, as described in the Results Section under 'Neural Decoding,' and so the state-to-state component of our decoding algorithm involves reconstruction of smooth trajectories from the spike-to-state predictions of the decoding algorithm described in the Methods Section under 'Neural Decoding Algorithm' and 'FPGA Implementation.'

The classic Viterbi algorithm is an efficient, recursive approach to state sequence estimation; it is optimal in the sense that it generates a maximum *a posteriori* probability estimation of the sequence of hidden states in a finite-state, discrete-time Markov process operating in memoryless noise [60]. In casting our decoding problem in terms of a discrete-time, finite-state Markov process, we can construct trajectories by treating discretized positions as states in a hidden Markov model, and applying a modified Viterbi algorithm to convert the decoded output into a maximum-likelihood trajectory.

In the context of our decoder, the observed states are the values generated at the output of the decoder, $\vec{d}(t)$, at each time step, t. The actual positions $s(t)\in\{s_i\}$ at the corresponding times are the associated hidden states. Implementing the Viterbi algorithm requires recursively computing

$$\hat{s}(t)=\max_i P\left(\vec{d}(t)|s_i(t)\right)P(s_i(t)|s(t-1)), \tag{5}$$

where $\hat{s}(t)$ is the optimal estimate for the trajectory position at time step t. In the following paragraphs we explain the other terms in Equation 5, as well as our approaches to computing them.

The first term in Equation 5 denotes the probability of obtaining the decoder output $\vec{d}(t)$ when the corresponding, true position is $s_i(t)$. We use the following method to estimate these conditional probabilities. As the decoder output $\vec{d}(t)$ consists of m single-bit indicators reflecting the probability that $s(t)=s_i$, we treat each component $d_i(t)$ of $\vec{d}(t)$ as being independent of the other $m-1$. Under this assumption of independence,

$$P\left(\vec{d}(t)|s_i(t)\right) \approx \prod_i P\left(\vec{d}_j(t)|s_i(t)\right), \tag{6}$$

where the index j in Equation 6 runs over all components i of $\vec{d}(t)$ for which $\vec{d}_i(t)=1$. We estimate the probabilities forming the individual terms in the product of Equation 6 empirically, by computing a confusion matrix, C, describing the performance of the decoder during its learning period. The element c_{ij} of this confusion matrix is the ratio of the number of time windows during which the decoder generated $d_i(t)=1$ when the correct state was s_j, to the total number of time windows during which the decoder generated $d_i(t)=1$; hence, c_{ij} approximates $P\left(\vec{d}_j(t)|s_i(t)\right)$.

The second term in Equation 5 represents a probability of transition from one state to the next; in our system, these transitions correspond to movements from one position to another. Such movements must obey physical constraints, and so the most probable transitions at adjacent time steps are those between each position and itself, and between a given position and its nearest neighbors. We therefore set the prior probability of transition between positions s_i and s_j according to the diffusion-like expression in Equation 7:

$$P\left(s_i(t)|s_j(t-1)\right) \equiv \frac{1}{Z_j} e^{-\alpha\frac{\left(\Delta(s_i,s_j)\right)^2}{\Delta t}} \tag{7}$$

$$Z_j \equiv \sum_{i=1}^{n_S} e^{-\alpha\frac{\left(\Delta(s_i,s_j)\right)^2}{\Delta t}}, \tag{8}$$

where $\Delta(s_i,s_j)$ denotes the physical distance between the indicated states, and Z_i is the normalization constant for transitions from s_j to s_i. The inverse proportionality to Δt, the time interval since the decoder generated at least one nonzero bit, permits transitions between more distant states with increased probability as time elapses, by broadening the probability distribution with elapsed time between informative observations; α is a constant, related in our system to mean speed of movement, that tunes the rate of this spreading. We set $\alpha=0.85$ to obtain the results shown in Figure 5.

Acknowledgments

Behavioral and neurophysiologic data used in the present work were collected in the laboratory of Matthew A. Wilson, Picower Institute for Learning and Memory, MIT Department of Brain and Cognitive Sciences.

Author Contributions

Conceived and designed the experiments: BIR LT RS. Performed the experiments: BIR LT WW TJD. Analyzed the data: BIR LT. Contributed reagents/materials/analysis tools: TJD. Wrote the paper: BIR LT.

References

1. Patil PG (2009) Introduction: Advances in brain-machine interfaces. Neurosurgical Focus 27: E1.
2. Nicolelis MAL (2003) Brain–machine interfaces to restore motor function and probe neural circuits. Nature Reviews Neuroscience 4: 417–422.
3. Schwartz A, Cui XT, Weber D, Moran D (2006) Brain-controlled interfaces: Movement restoration with neural prosthetics. Neuron 56: 205–220.
4. Hochberg LR, Serruya MD, Friehs GM, Mukand JA, Saleh M, et al. (2006) Neuronal ensemble control of prosthetic devices by a human with tetraplegia. Nature 442: 164–171.
5. Chader GJ, Weiland J, Humayun MS (2009) Artificial vision: needs, functioning, and testing of a retinal electronic prosthesis. Neurotherapy: Progress in Restorative Neuroscience and Neurology 175: 317–332.
6. Shire DB, Kelly SK, Chen JH, Doyle DP, Gingerich MD, et al. (2009) Development and implantation of a minimally invasive wireless subretinal neurostimulator. IEEE Transactions on Biomedical Engineering 56: 2502–2511.
7. Tarsy D, Vitek J, Starr P, Okun M, editors (2008) Deep brain stimulation in neurological and psychiatric disorders. Current Clinical Neurology. Totowa: Humana Press.
8. Salanova V, Worth R (2007) Neurostimulators in epilepsy. Current Neurology and Neuroscience Reports 7: 315–319.
9. Sarpeshkar R (2010) Ultra Low Power Bioelectronics: Fundamentals, Biomedical Applications, and Bio-Inspired Systems. Cambridge University Press.
10. Narasimhan K (2004) Scaling up neuroscience. Nature Neuroscience 7: 425.
11. Eliasmith C, Anderson CH (2003) Neural Engineering, MIT Press, chapter 4. pp. 112–115.
12. Rapoport B, Sarpeshkar R (2011) A Biomimetic Adaptive Algorithm and Micropower Circuit Architecture for Implantable Neural Decoders, IGI Global, chapter 10. pp. 216–254. In *System and Circuit Design for Biologically-Inspired Learning*, Edited by T. Temel.
13. Rapoport BI, Wattanapanitch W, Penagos HL, Musallam S, Andersen RA, et al. (2009) A biomimetic adaptive algorithm and low-power architecture for implantable neural decoders. Pro-ceedings of the 31st Annual International Conference of the IEEE Engineering in Medicine and Biology Society 2009: 4214–4217.
14. Holleman J, Otis B (2007) A sub-microwatt low-noise amplifier for neural recording. 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society 2007: 3930–3933.
15. Wattanapanitch W, Fee M, Sarpeshkar R (2007) An energy-efficient micropower neural recording amplifier. IEEE Transactions on Biomedical Circuits and Systems 1: 136–147.
16. Wattanapanitch W, Sarpeshkar R (2011) A low-power 32-channel digitally-programmable neural recording system. IEEE Transactions on Biomedical Circuits and Systems 5: 592–602.
17. Ghovanloo M, Atluri S (2007) A wideband power-efficient inductive wireless link for implantable microelectronic devices using multiple carriers. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications 54: 71–77.
18. Baker MW, Sarpeshkar R (2007) Feedback analysis and design of RF power links for low-power bionic systems. IEEE Transactions on Biomedical Circuits and Systems 1: 28–38.
19. Mandal S, Sarpeshkar R (2008) Power-efficient impedance-modulation wireless data links for biomedical implants. IEEE Transactions on Biomedical Circuits and Systems 2: 301–315.
20. Mohseni P, Najafi K, Eliades SJ, Wang X (2005) Wireless multichannel biopotential recording using an integrated fm telemetry circuit. IEEE Transactions on Neural Systems and Rehabilitation Engineering 13: 263–271.
21. Theogarajan LS (2008) A Low-Power Fully Implantable 15-Channel Retinal Stimulator Chip. IEEE Journal of Solid-State Circuits 43: 2322–2337.
22. Kelly SK, Wyatt JL (2011) A power-efficient neural tissue stimulator with energy recovery. IEEE Transactions on Biomedical Circuits and Systems 5: 20–29.
23. Arfin SK, Sarpeshkar R (2012) An Energy-Efficient, Adiabatic Electrode Stimulator with Inductive Energy Recycling and Feedback Current Regulation. IEEE Transactions on Biomedical Circuits and Systems 6: 1–14.
24. Sarpeshkar R, Wattanapanitch W, Arfin SK, Rapoport BI, Mandal S, et al. (2008) Low-power circuits for brain-machine interfaces. IEEE Transactions on Biomedical Circuits and Systems 2: 173–183.
25. Olsson RH, Wise KD (2005) A three-dimensional neural recording microsystem with implantable data compression circuitry. IEEE Journal of Solid-State Circuits 40: 2796–2804.
26. Warland D, Reinagel P, Meister M (1997) Decoding visual information from a population of retinal ganglion cells. Journal of Neurophysiology 78: 2336–2350.
27. Serruya M, Hatsopoulos NG, Fellows M, Paninski L, Donoghue J (2003) Robustness of neuropros-thetic decoding algorithms. Biological Cybernetics 88: 219–228.
28. Sanchez JC, Erdogmus D, Principe JC, Wessberg J, Nicolelis MAL (2005) Interpreting spatial and temporal neural activity through a recurrent neural network brain machine interface. IEEE Transactions on Neural Systems and Rehabilitation Engineering 13: 213–219.
29. Wu W, Black MJ, Mumford D, Gao Y, Bienenstock E, et al. (2004) Modeling and decoding motor cortical activity using a switching Kalman filter. IEEE Transactions on Biomedical Engineering 51: 933–942.
30. Wu W, Gao Y, Bienenstock E, Donoghue JP, Black MJ (2006) Bayesian population decoding of motor cortical activity using a Kalman filter. Neural Computation 18: 80–118.
31. Wu W, Shaikhouni A, Donoghue JP, Black MJ (2004) Closed-loop neural control of cursor motion using a Kalman filter. Proceedings of the IEEE Engineering in Medicine and Biology Society 26: 4126–4129.

32. Wu W, Hatsopoulos NG (2008) Real-time decoding of nonstationary neural activity in motor cortex. IEEE Transactions on Neural Systems and Rehabilitation Engineering 16: 213–222.
33. Brockwell AE, Rojas AL, Kass RE (2004) Recursive Bayesian Decoding of Motor Cortical Signals by Particle Filtering. Journal of Neurophysiology 91: 1899–1907.
34. Brown EN, Frank LM, Tang D, Quirk MC, Wilson MA (1998) A statistical paradigm for neural spike train decoding applied to position prediction from ensemble firing patterns of rat hippocampal place cells. Journal of Neuroscience 18: 7411–7425.
35. Eden UT, Frank LM, Barbieri R, Solo V, Brown EN (2004) Dynamic analyses of neural encoding by point process adaptive filtering. Neural Computation 16: 971–998.
36. Srinivasan L, Eden UT, Mitter SK, Brown EN (2007) General-purpose filter design for neural prosthetic devices. Journal of Neurophysiology 98: 2456–2475.
37. Sarma SV, Eden UT, Cheng ML, Williams ZM, Hu R, et al. (2010) Using Point Process Models to Compare Neural Spiking Activity in the Subthalamic Nucleus of Parkinson's Patients and a Healthy Primate. IEEE Transactions on Biomedical Engineering 57: 1297–1305.
38. Pesaran B, Pezaris JS, Sahani M, Mitra PP, Andersen RA (2002) Temporal structure in neuronal activity during working memory in macaque parietal cortex. Nature Neuroscience 5: 805–811.
39. Musallam S, Corneil BD, Greger B, Scherberger H, Andersen RA (2004) Cognitive control signals for neural prosthetics. Science 305: 258–262.
40. Santhanam G, Ryu SI, Yu BM, Afshar A, Shenoy KV (2006) A high-performance brain-computer interface. Nature 442: 195–198.
41. Shenoy KV, Meeker D, Cao SY, Kureshi SA, Pesaran B, et al. (2003) Neural prosthetic control signals from plan activity. Neuroreport 14: 591–596.
42. Japkowicz N, Shah M (2011) Evaluating Learning Algorithms: A Classification Perspective. Cam-bridge: Cambridge University Press.
43. Koyama S, Chase SM, Whitford AS, Velliste M, Schwartz AB, et al. (2010) Comparison of brain–computer interface decoding algorithms in open-loop and closed-loop control. Journal of Computational Neuroscience 1–2: 73–87.
44. Louie K, Wilson MA (2001) Temporally Structured Replay of Awake Hippocampal Ensemble Activity during Rapid Eye Movement Sleep. Neuron 29: 145–156.
45. Lee AK, Wilson MA (2002) Memory of Sequential Experience in the Hippocampus during Slow Wave Sleep. Neuron 36: 1183–1194.
46. Madisetti V, editor (2009) The Digital Signal Processing Handbook. Electrical Engineering Hand-book. Boca Raton, Florida: CRC Press, 2nd edition.
47. Wolfram S (2002) A New Kind of Science. Champaign, Illinois: Wolfram Media.
48. O'Keefe J, Nadel L (1978) The Hippocampus as a Cognitive Map. Oxford, England: Clarendon Press.
49. Best PJ, White AM, Minai A (2001) Spatial processing in the brain: The activity of hippocampal place cells. Annual Review of Neuroscience 24: 459–86.
50. McNaughton BL, Battaglia FP, Jensen O, Moser FI, Moser MB (2006) Path integration and the neural basis of the 'cognitive map'. Nature Reviews Neuroscience 7: 663–678.
51. Davidson TJ, Kloosterman F, Wilson MA (2009) Hippocampal replay of extended experience. Neuron 63: 497–507.
52. Hochberg LR, Serruya MD, Friehs GM, Mukand JA, Saleh M, et al. (2006) Neuronal ensemble control of prosthetic devices by a human with tetraplegia. Nature 442: 164–171.
53. Paninski L, Fellows MR, Hatsopoulos NG, Donoghue JP (2004) Spatiotemporal tuning of motor cortical neurons for hand position and velocity. Journal of Neurophysiology 91: 515–532.
54. Pouget A, Dayan P, Zemel RS (2003) Inference and computation with population codes. Annual Reviews of Neuroscience 26: 381–410.
55. Jain AK, Duin RP, Mao J (2000) Statistical pattern recognition: A review. IEEE Transactions on Pattern Analysis and Machine Intelligence 22: 4–37.
56. Bengio Y (2007) On the challenge of learning complex functions. Progress in Brain Research 165: 521–534.
57. Stark E, Abeles M (2007) Predicting movement from multiunit activity. Journal of Neuroscience 27: 8387–8394.
58. Rizk M, Wolf PD (2009) Optimizing the Automatic Selection of Spike Detection Thresholds Using a Multiple of the Noise Level. Medical and Biological Engineering and Computing 47: 955–966.
59. Actel Corporation. IGLOO Low Power Flash FPGAs with Flash*Freeze Technology. Data Sheet. November 2009.
60. Forney GD (1973) The Viterbi Algorithm. Proceedings of the IEEE 61: 268–278.