nature communications

Article

https://doi.org/10.1038/s41467-022-34452-w

Real-time brain-machine interface in non-human primates achieves high-velocity prosthetic finger movements using a shallow feedforward neural network decoder

Received: 16 March 2022

Accepted: 25 October 2022

Published online: 12 November 2022

Check for updates

Matthew S. Willsey[1,2], Samuel R. Nason-Tomaszewski[2], Scott R. Ensel[2], Hisham Temmar[2], Matthew J. Mender[2], Joseph T. Costello[3], Parag G. Patil[1,2,4,5] & Cynthia A. Chestek[2,3,4,6,7]

Despite the rapid progress and interest in brain-machine interfaces that restore motor function, the performance of prosthetic fingers and limbs has yet to mimic native function. The algorithm that converts brain signals to a control signal for the prosthetic device is one of the limitations in achieving rapid and realistic finger movements. To achieve more realistic finger movements, we developed a shallow feed-forward neural network to decode real-time two-degree-of-freedom finger movements in two adult male rhesus macaques. Using a two-step training method, a recalibrated feedback intention–trained (ReFIT) neural network is introduced to further improve performance. In 7 days of testing across two animals, neural network decoders, with higher-velocity and more natural appearing finger movements, achieved a 36% increase in throughput over the ReFIT Kalman filter, which represents the current standard. The neural network decoders introduced herein demonstrate real-time decoding of continuous movements at a level superior to the current state-of-the-art and could provide a starting point to using neural networks for the development of more naturalistic brain-controlled prostheses.

Brain–machine interfaces (BMIs) offer hope to the very high numbers of Americans (~1.7%) with sensorimotor impairments[1]. To this end, cortical BMIs have allowed human patients using brain-controlled robot arms to perform a variety of motor tasks such as bringing a drink to the mouth[2] or stacking cups[3]. Motor decoding algorithms are required to convert brain signals into a control signal, usually with position and velocity updates, for the prosthetic device. Despite the potentially non-linear relationship between neural activity and motor movements[4,5], linear algorithms—including ridge regression, Kalman filtering, and Poisson processes—represent state-of-the-art performance in motor decoding[2,6–8]. Even with the rapid progress, many recognize that further developments are necessary to restore quick and naturalistic movements[2].

Some gains in performance have already been achieved by adding non-linearities to classic linear decoders to leverage the likely non-linear relationship between neural activity and motor movements. For example, since the neural activity is markedly different when moving compared to stationary postures, decoders have been introduced to

[1]Department of Neurosurgery, University of Michigan, Ann Arbor, MI, USA. [2]Department of Biomedical Engineering, University of Michigan, Ann Arbor, MI, USA. [3]Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA. [4]Neuroscience Graduate Program, University of Michigan Medical School, Ann Arbor, MI, USA. [5]Department of Anesthesiology, University of Michigan, Ann Arbor, MI, USA. [6]Robotics Graduate Program, University of Michigan, Ann Arbor, MI, USA. [7]Biointerfaces Institute, University of Michigan, Ann Arbor, MI, USA. e-mail: cchestek@umich.edu

move a prosthesis only when the desire to move is detected[4,7,9]. To leverage the non-linear relationship between kinematics and motor cortex neural activity, the classic Kalman filter has been adapted by expanding its state space[10] or with Gaussian mixture models[11] so that the algorithm can adopt different linear relationships in different movement contexts. In a particularly novel implementation, Sachs et al.[12] implemented a weighted combination of two Wiener filters trained for either fast movements or near-zero velocities so that continuously decoded velocities largely draw upon the fast Wiener filter at the beginning of the trial and the slow-movement filter as the cursor approaches the target. However, for many of these approaches, performance is improved only for very specific tasks, and a general-purpose nonlinear approach is lacking.

Artificial neural network decoders, with their capability to model complex non-linear relationships, have long been thought to hold tremendous promise for brain-machine interfaces. They may ultimately also represent the most biomimetic motor decoder to transform motor cortex activity into realistic motor movements. However, early neural network decoders, prior to recent advancements in hardware, toolboxes, and training methods, were not found to improve performance over standard linear methods when decoding continuous motor movements[13,14]. Many advanced techniques employing recurrent neural networks and variational inference techniques show great promise for predicting prosthetic kinematics from brain signals (in offline testing). However, these techniques are often employed to perform classification[15], as opposed to continuous motor decoding, and are not used in real-time control of prosthetic devices (in online testing), likely because of the computational complexity[16,17]. Sussillo et al.[18], however, did demonstrate real-time control of a computer cursor with a recurrent neural network in a non-human primate implanted with motor cortex arrays. However, this did not outperform a ReFIT Kalman filter in the same animals[6,18]. George et al.[19] demonstrated control of hand and finger movements in human amputees with peripheral nerve interfaces using a convolutional neural network but again did not outperform a linear Kalman filter.

In this work, we demonstrate a ReFIT neural network for decoding brain activity to control random and continuous two-degrees-of-freedom movements in real time using Utah arrays in rhesus macaques. The ReFIT neural network is compared with the ReFIT Kalman filter, which we use to represent the current state-of-the-art in linear decoders. The ReFIT Kalman filter, conceptually similar to other multi-stage training/calibration procedures[20,21], is a two-step training process that first computes the weights of a classic Kalman filter and then modifies the weights when the prosthesis direction is not toward the actual target[6].

In this study, we find that the ReFIT neural network decoder substantially outperforms our previous implementation of the ReFIT Kalman filter[22–24] with >60% increase in throughput by utilizing high-velocity movements without compromising the ability to stop. The decision to explore shallow-layer artificial networks was inspired by the biological motor pathway from the pre-central gyrus to the spinal cord. These artificial neural networks may be a bridge toward more sophisticated and deeper neural network decoders and eventually lead to improved high-velocity, naturalistic robotic prostheses.

Results

Two adult male rhesus macaques were implanted with Utah arrays (Blackrock Microsystems, Salt Lake City, Utah) in the hand area of the primary motor cortex (M1), as shown in Fig. 1a. The macaques were trained to sit in a chair, and perform a finger target task in which a hand manipulandum was used to control virtual fingers on a computer screen in front of the animal. During online BMI experiments, spike-band power (SBP) was used as the neural feature. SBP is the time-averaged power in the 300–1000 Hz frequency band that provides a high signal-to-noise ratio correlate of the dominant single-unit spiking rate and usually outperforms threshold crossings as a feature[25]. A two-effector finger task was previously developed by Nason et al.[23], where the monkeys used two individual finger groups to acquire simultaneous targets along a one-dimensional arc. Although there are two degrees of freedom in this task, similar to two-dimensional cursor tasks, the animal must track two independent fingers, and the underlying neural mechanism may be inherently different than the hypothesized cosine tuning of the cursor task[23]. Monkey N used his index (D2) finger individually and his middle-ring-small (D3-5) fingers as a group, and Monkey W used D2 and D3 as one group and D4 and D5 as the second group. Unlike the previous task using center-out targets, where targets appeared in pre-defined positions[23], task difficulty was increased by placing targets at random positions within the one-dimensional active range of motion of each finger group. Further specifics of the task are available in the Methods. After a 400-trial calibration task, a decoder was trained to predict the velocity of both finger groups, as shown in Fig. 1b. We have recently demonstrated online real-time decoding of these 2 degrees of freedom using a ReFIT Kalman filter[23], and primarily compare our algorithm to that approach.

Offline analysis of the neural network architecture

Limited computational complexity was a design goal for the neural network to allow same-day training and testing. As most online decoders incorporate recent time history[2,3], the neural network was designed so that an initial time-feature layer constructed 16-time features per electrode from the preceding 150 ms of SBP (time feature layer in Fig. 1c). These time features were then input into 4 fully connected layers, where the first three output to a rectified linear unit (ReLU) activating function and the final layer outputs a velocity for each finger group. The number of fully connected layers and output time features were chosen to achieve a near-maximal correlation coefficient in offline performance using 400 trials of training data. As can be seen in Fig. 2a, increasing the number of neurons in hidden layers beyond 256 and the number of fully connected layers beyond 4 did not substantially improve the offline correlation. While these numbers of layers and hidden neurons were close to optimal given our amount of training data (400 trials), which allowed networks to be trained and tested on the same day, more training data would allow for larger neural network algorithms, as shown in Fig. 2a. On the other hand, less training data might suggest smaller networks. Furthermore, increasing the number of time features beyond 16 (Fig. 2b) also did not substantially improve the offline correlation. For notational simplicity, the neural network in Fig. 1c is abbreviated as NN.

While the primary endpoint of our offline analysis was to compare our final neural network (NN in Fig. 1c) with the Kalman filter, we also wanted to understand the impact on the performance of the individual network components, including the time feature input and a number of layers, assessed through an offline analysis based on 3 days of recorded spike-band power for each monkey during manipulandum-controlled finger task, where algorithms were trained and tested on the same day. Illustrative examples of predicted versus actual finger velocities for Monkey N using the manipulandum are given for neural networks of increasing complexity: 2 layers, 2 layers with time history, 4 layers, and 4 layers with time history (Fig. 2c). The correlation of each neural network decoder relative to the Kalman filter correlation is given in Fig. 2d by combining both fingers over all days for each monkey. The offline Kalman filter correlation averaged 0.59 ± 0.01 for Monkey N and 0.50 ± 0.02 for Monkey W.

The architectural features that improved offline correlation were the inclusion of time history and deeper networks. Correlations between predicted and actual movements are shown in Table 1 for a variety of comparisons. Our analysis included both a 2-layer network with and without regularization (i.e., batch normalization and dropout), which showed that these techniques increased the correlation across both animals ($P < 10^{-5}$). Moving from a 2-layer to a 4-layer

network also improved offline performance ($P = 2.6 \times 10^{-3}$) by 0.07 in Monkey N and 0.04 in Monkey W. The benefit of including time history is evaluated by comparing the 2-layer network with and without time history, which increases the correlation ($P < 10^{-5}$) by 0.09 in Monkey N and 0.08 in Monkey W.

We also performed an offline analysis to compare the correlation for the 4-layer network with time history (NN in Fig. 1c) with the Kalman filter. When doing so, NN also achieved a higher offline correlation by 0.08 in Monkey N ($P = 3.6 \times 10^{-4}$) and by 0.04 in Monkey W ($P = 0.016$). The total performance comparisons are summarized in Table 1. As a final control, to ensure that the offline dimensionality reduction benefit of the NN over KF is not derived entirely from the additional time features, we compared NN with a classic Kalman filter with the same number of time-lagged features (a total of 3-time bins). Across the 6 days in both monkeys, NN maintained a 0.035 ± 0.01 advantage in a correlation coefficient (6%) over the Kalman filter ($P = 1.9 \times 10^{-3}$).

Neural network decoder outperforms ReFIT Kalman filter decoder in real-time tests

In two non-human primates (NHP), Monkeys N and W, neural network decoders outperformed a ReFIT Kalman filter (RK) during real-time (online) testing, and the performance results are summarized in Table 2. In Monkey N, a neural network decoder outperformed the RK 13 mos after implantation in 2 days of testing over 1080 total trials, regardless of which algorithm was used first. The NN decoder improved the throughput over the RK by 26% with 2.15 ± 0.05 bits per second (bps) for the NN and 1.70 ± 0.03 bps for RK ($P < 10^{-5}$). The acquisition time was lower at 1240 ± 40 ms for the NN and 1550 ± 40 ms for the RK ($P < 10^{-5}$). NN had 3/543 unsuccessful trials while RK had 1/537 unsuccessful trials. In Monkey W, NN and RK decoders were compared 2 mos after implantation on one-day testing over 412 trials. As graphically depicted in Fig. 3a, the NN decoder improved the throughput over the RK by 46%, with 1.23 ± 0.09 bps for the NN and 0.84 ± 0.04 bps for RK ($P < 10^{-5}$). The acquisition time was lower at 2680 ± 160 ms for NN and 3310 ± 130 ms for RK ($P < 2.5 \times 10^{-3}$). NN had 26/133 unsuccessful trials while RK had 113/279 unsuccessful trials. Figure 3a illustrates the throughput of each trial and the mean value for each run.

ReFIT neural network decoder outperforms both the original NN and RK decoders

The ReFIT innovation was applied to the neural network in a similar manner as it was used with the Kalman filter. Essentially, after completing trials using the NN decoder, the NN learned weights were further updated whenever the predicted finger direction was oriented away from the true targets, as described in the Methods. The ReFIT neural network (RN) decoder improved performance across all metrics when compared with the original NN in both monkeys (illustrated in

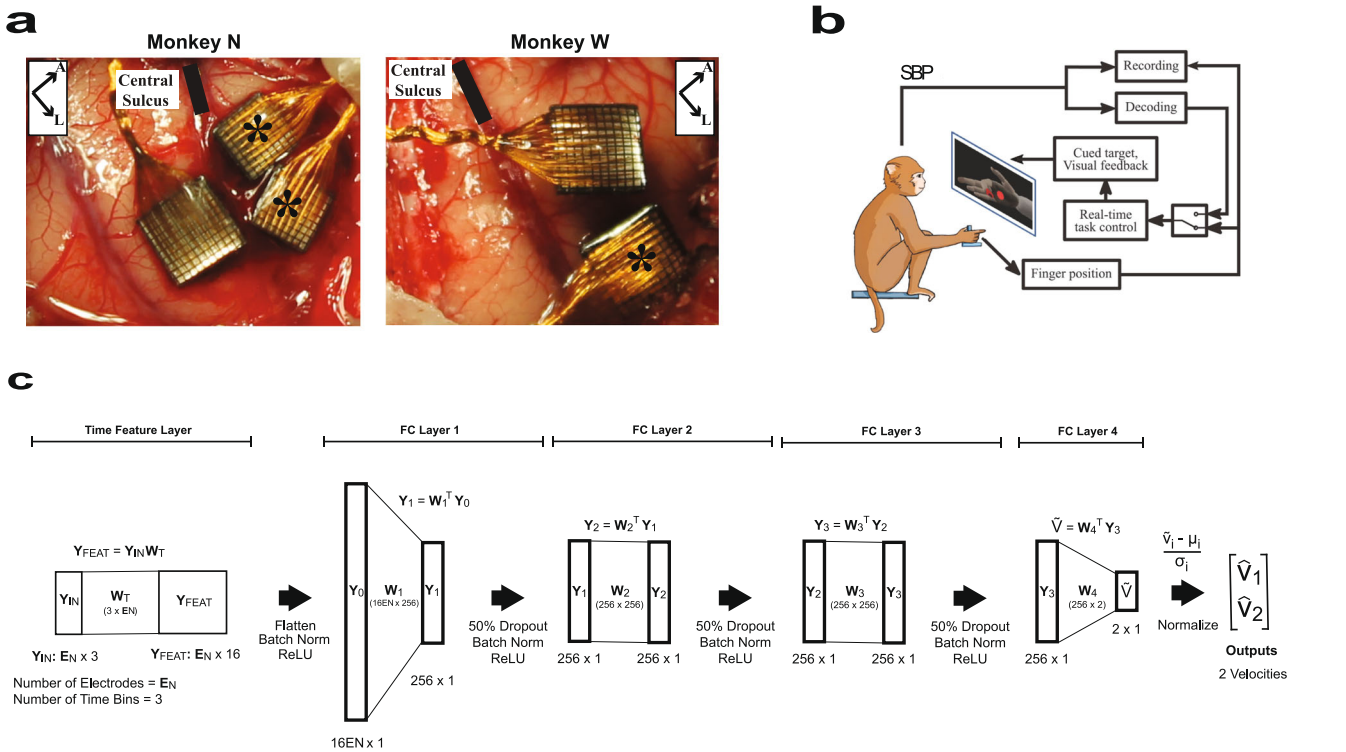

Fig. 1 | Neural network velocity decoder. a Image of Utah array implants for Monkeys N (left) and W (right). In Monkey N, two split Utah arrays were implanted in the primary motor cortex immediately anterior to the central sulcus and denoted with asterisks (*). The array in the primary somatosensory cortex was not used in this analysis. In Monkey W, two 96-channel arrays were implanted and the analysis herein uses the lateral array. **b** Experimental setup. The NHP is controlling the virtual finger with the hand manipulandum in manipulandum-control mode or using spike-band power (SBP) to control the virtual finger in brain-control mode. **c** NN architecture. The network consists of five layers. The input to the network is Y_{IN} which is a $E_N \times 3$ data matrix that corresponds to the number of input electrodes and the 3 previous 50-ms time bins. The time feature layer converts the last three 50-ms time bins for all the electrodes into 16 learned time features for each electrode. The equation representing the operation is given above the graphical description of the layer. The arrow indicates that the elements undergo batch normalization and pass through a ReLU function and are then flattened to an $16E_N \times 1$ array. The remaining four layers are fully connected layers with an associated weight matrix, denoted by W. The first three layers consist of 256 hidden neurons and process the hidden neuron output first with 50% dropout, then batch normalization, and finally with a ReLU function. The fourth and final fully connected layer, FC-Layer 4, has two neurons – that are normalized—and represents the final velocity estimates of the two fingers, $\dot{v}_1$ and $\dot{v}_2$. Panel **b** was adapted from Vaskov AK, Irwin ZT, Nason SR, Vu PP, Nu CS, Bullard AJ, Hill M, North N, Patil PG and Chestek CA (2018) Cortical Decoding of Individual Finger Group Motions Using ReFIT Kalman Filter. Front. Neurosci. 12:751. doi: 10.3389/fnins.2018.00751 and licensed under CC BY 4.0 (https://creativecommons.org/licenses/by/4.0/); "spikes" replaced with SBP.

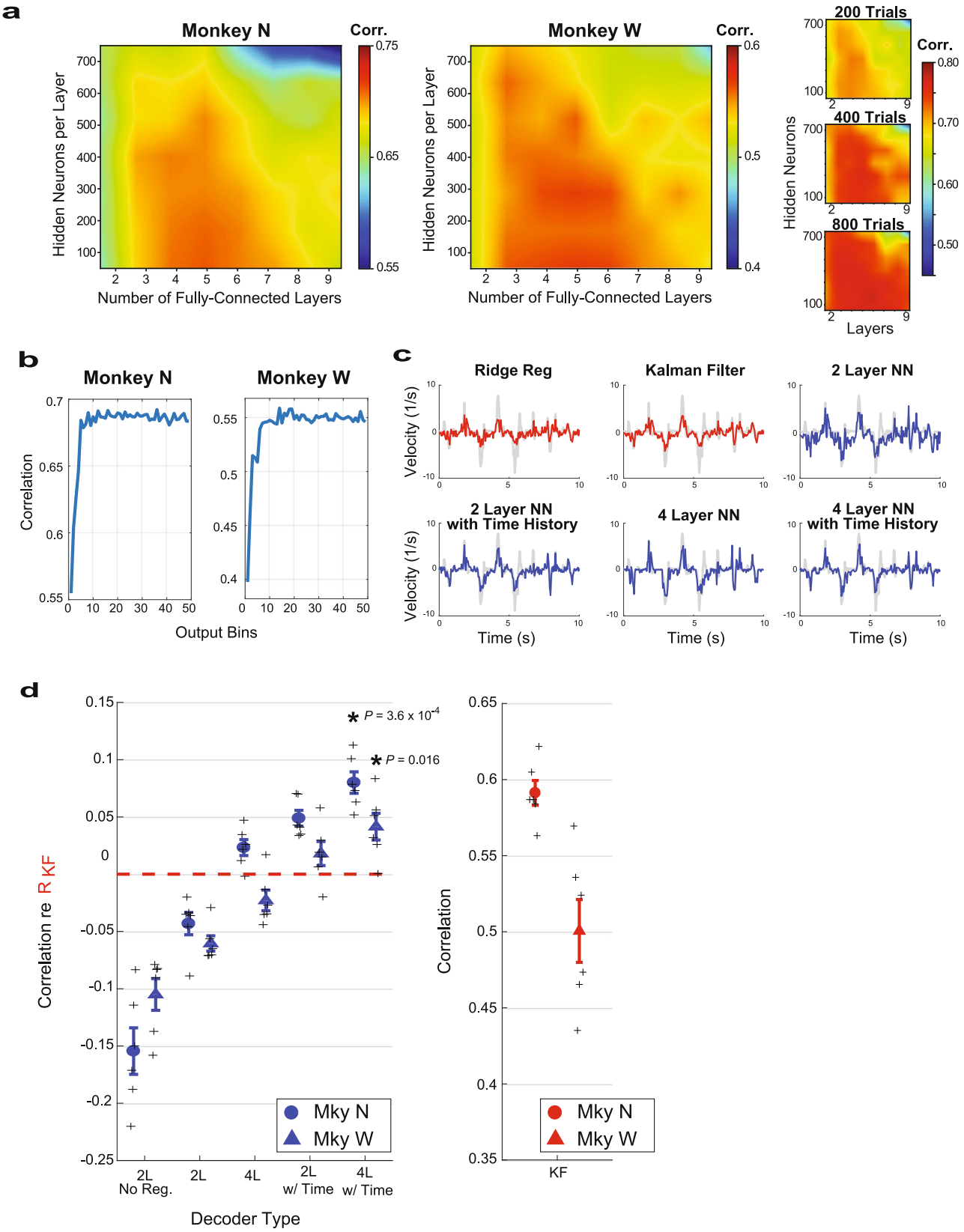

Fig. 3b and Table 2). These tests for Monkey N were conducted at 19 mos post-implantation, and the decoding performance of all decoders had declined from earlier tests at 13 mos.

In Monkey N, who was capable of running a large number of consistent trials in 1 day, RN was compared directly with RK 19 mos after implantation in 2 days of testing with 1351 total trials (Fig. 3c and Table 2). RN improved the throughput over the RK by 62%, with 2.29 ± 0.05 bps for the RN and 1.41 ± 0.03 bps for RK ($P < 10^{-5}$). As illustrated in Fig. 3c, RN achieved a higher throughput on each day ($P < 10^{-5}$ for each day) regardless of which algorithm was used first. Average performance of each decoder for the random finger task is illustrated in Supplementary Movies 1 and 2. The acquisition time was

Fig. 2 | Neural network offline analyses. a Heat map illustrating the offline correlation (Corr.) between the number of fully connected layers versus the number of hidden neurons per layer for Monkeys N (left) and W (right) for NN in Fig. 1c. The three smaller maps on the right illustrate using a different day with a variety of training trials. **b** The correlation during offline training for Monkeys N and W as a function of the number of learned time features in the output from the time-history layer of Fig. 1c. **c** Examples comparing actual velocity (gray) and decoded velocity for linear decoders (red) and neural network decoders (blue) during 1 day of manipulandum-control tasks for Monkey N. The *Y*-axis is normalized by the standard deviation of actual velocities during the entire run. **d** Mean (and SEM) offline correlation difference between one of the neural network decoders and the Kalman filter, i.e., correlation of neural network decoder minus correlation of the Kalman filter. The circles denote the mean for Monkey N over both fingers over 3 days ($n = 6$ samples), and the triangles denote the mean for Monkey W over two fingers over 3 days ($n = 6$ samples). The individual samples are denoted by "+" marks. The primary endpoint of this analysis is to compare the offline correlation of the 4-layer neural network with time history (NN in Fig. 1c) with the Kalman filter. The *P*-values are listed and calculated from a one-sample, two-tailed *t*-test. Asterisks denote this statistical significance in both animals. 2L No Reg = 2-layer neural network without any regularization (i.e., batch norm or dropout); 2L 2-layer neural network, 2L w/ Time 2-layer neural network with a preceding time feature layer; 4L 4-layer neural network, 4L w/Time 4-layer neural network with a preceding time feature layer. Asterisks (*) denote statistically significant differences between the correlation of the neural network decoder and the Kalman filter. Source data are provided as a Source Data file.

1270 ± 30 ms for the RN and 1940 ± 50 ms for the RK ($P < 10^{-5}$). There were no unsuccessful trials (0/737) using the RN and 55/614 unsuccessful trials with the RK. Representative raw finger tracings are depicted in Fig. 3d for the RK and in Fig. 3e for the RN and depict a time segment with a throughput equivalent to the average throughput over both days. The tracings illustrate the higher target acquisition rate for the RN (30 targets in 50 s) compared to the RK (21 targets in 50 s). A graphical illustration for a data collection timeline is given in Fig. 3f, and data are summarized in Table 2.

Neural network decoders allow higher velocity decodes than the Kalman filter

To better understand why the RN outperformed the RK decoder, the mean velocity over all the successful trials was computed for each decoder for both monkeys. As seen in Fig. 4a, b, virtual fingers controlled by the RN and NN decoders achieved higher peak velocities and were more responsive for both monkeys than when the virtual fingers were controlled by the RK decoder. For Monkey N (Fig. 4e), the time to the mean velocity peak was 300 ms for RN, 350 ms for NN, and 450 ms for RK. The peak of the averaged velocity was 1.35 ± 0.03 u/s for RN, 1.00 ± 0.03 u/s for NN, and 0.55 ± 0.02 u/s for RK, where u denotes arbitrary units such that 1 was full flexion and 0 was a full extension. For Monkey W (Fig. 4a), the time to peak was 350 ms for RN, 400 ms for NN, and 800 ms for RK. The peak average velocity was 0.94 ± 0.04 u/s for RN, 0.76 ± 0.04 u/s for NN, and 0.39 ± 0.04 for RK u/s. Thus, in both monkeys, the peak value of the mean velocity improved with RN and NN compared with the standard RK decoder ($P < 10^{-5}$ for both monkeys). The high velocities achieved using RN are illustrated for a center-out task in Supplementary Movie 3.

To ensure the NN does not achieve high-velocity decodes at the expense of low-velocity decoding accuracy, which is important for stopping the prosthesis, the predicted velocity as a function of the true velocity was compared for the NN and Kalman filter (KF) in an offline analysis (graphically shown in Fig. 4b). The predicted velocity on the vertical axis is scaled so that when the true velocity is at zero, the standard deviation of the predicted velocity equals 1/2. In the high-velocity range (>1 u/s), the decoded NN velocity averages 157 ± 3% of the KF velocity for Monkey N and 122 ± 2% for Monkey W. Thus, after accounting for decoder performance at low velocities, the range of velocities that can be achieved is higher for the NN than the KF. As shown in the online analysis, higher velocities improved the performance of NN decoders.

Table 1 | Offline performance comparing Kalman filter and neural network (NN) decoders

Decoder	Correlation	
	Monkey N	Monkey W
Kalman filter	0.59 ± 0.01	0.50 ± 0.02
2-layer NN (no regularization)	0.44 ± 0.02	0.40 ± 0.02
2-layer NN	0.55 ± 0.02	0.44 ± 0.02
4-layer NN	0.61 ± 0.01	0.48 ± 0.02
2-layer NN with time history	0.64 ± 0.01	0.52 ± 0.02
4-layer NN with time history	0.67 ± 0.01	0.54 ± 0.02

ReFIT neural network decoder outperforms optimized RK decoder

Our implementation of the ReFIT Kalman filter for finger control utilizes a physiological lag and does not include hyper-parameter tuning (i.e., gain and smoothing parameters)[22–24]. Other work suggests RK performance can be improved without lag (providing the RK updates to the virtual fingers without delay)[26] and by optimizing the online gain and smoothing parameters for RK[27]. Furthermore, practicing with other linear decoders has also been found to improve performance[13]. Therefore, we optimized the Kalman filter in each of these areas, as in control tests described in "Optimizing the lag, gain, and scaling factors for real-time tests." Hyperparameter tuning increased throughput by 3.6% and average peak finger velocity by 16% and transitioning to zero-lag increased throughput by 16% and peak velocity by 13%. Using this optimized RK, RK_{opt}, with zero-lag, hyper-parameter tuning, and allowing for abundant practice, we then directly compared RK_{opt} to RN in 2 days of testing (1164 trials) at 29 mos post-implantation with Monkey N, using the same protocol as used above to compare RK and RN. In these tests, the throughput of RN of 2.41 ± 0.05 bps remained greater than that of RK_{opt} at 2.12 ± 0.05 bps ($P = 1.3 \times 10^{-5}$). RN's peak velocity of 1.29 ± 0.03 also remained greater than RK's peak velocity of 0.89 ± 0.02 u/s ($P < 10^{-5}$).

It is unlikely that animal practice caused the improvement of RN over RK because the animal has practiced more heavily with RK than RN. Between testing at 19 mos. post-implantation and these control tests at 29 mos. post-implantation, Mky N used a BMI decoder on 143 days of experiments unrelated to this work. Of these days, the Kalman filter was used in 124 days (87%) while the neural network decoders were used in only 28 days (20%).

Neural network merges decoders optimized for positive and negative velocities

Due to the network architecture itself, each node of the final hidden layer contributes either a positive or a negative velocity to the final prosthetic finger velocity. We explored whether this itself provides an example of how the fit is improved for different movement contexts, i.e. positive and negative velocities. Specifically, for finger 1, the sum of the product of N_k and $W_4^{(1,k)}$, overall k, determine $\hat{v}_1$, where N_k is the kth node of the final hidden layer and $W_4^{(1,k)}$ represents the learned weights (shown in Fig. 5a). Since each N_k is the output of the ReLU function, N_k is necessarily greater than or equal to zero. Thus, the nodal contribution of the kth node, $N_k\, W_4^{(1,k)}$, can be either positive (if $W_4^{(1,k)} > 0$) or negative ($W_4^{(1,k)} < 0$)—but not both positive and negative.

The nodal contributions of positive and negative nodes during day 1 for Monkey N are illustrated in Fig. 5b for positive velocities

Table 2 | Real-time performance comparison between ReFIT neural network (RN), neural network (NN), and ReFIT Kalman filter (RK) decoders

	Monkey N			Monkey W		
	RK	NN	RN	RK	NN	RN
NN vs. RK						
Throughput (bps)	1.70 ± 0.04	2.15 ± 0.05		0.84 ± 0.04	1.23 ± 0.09	
Acquisition time (ms)	1550 ± 40	1240 ± 40		3310 ± 130	2680 ± 160	
Time to target (ms)	1110 ± 30	950 ± 20		2410 ± 110	1680 ± 110	
Dwell time (ms)	440 ± 30	290 ± 20		790 ± 100	1000 ± 120	
Successful (total) trials	540 (543)	536 (537)		113 (279)	107 (133)	
Number of Test days	2			1		
Mos. post-implantation	13			3		
RN vs. NN						
Throughput (bps)		1.51 ± 0.04	2.15 ± 0.05		1.20 ± 0.06	1.43 ± 0.05
Acquisition time (ms)		1880 ± 50	1320 ± 30		2610 ± 120	2220 ± 70
Time to target (ms)		1230 ± 30	840 ± 20		1740 ± 80	1430 ± 50
Dwell time (ms)		650 ± 30	480 ± 30		860 ± 90	780 ± 60
Successful (total) trials		616 (629)	772 (772)		185 (237)	441 (483)
Number of Test days		2			1	
Mos. post-implantation		19			3	
RN vs. RK						
Throughput (bps)	1.41 ± 0.03		2.29 ± 0.05			
Acquisition time (ms)	1940 ± 50		1270 ± 30			
Time to target (ms)	1330 ± 30		790 ± 10			
Dwell time (ms)	610 ± 40		470 ± 30			
Successful (total) trials	559 (614)		737 (737)			
Number of Test days	2					
Mos. post-implantation	19					

Of note, there was a gap of 6 months in data collection between NN vs. RK and other tests with Monkey N because of a lab shutdown as a result of the COVID-19 pandemic.
bps bits per second.

($v_1 > \sigma$), negative velocities ($v_1 < \sigma$), and near-zero velocities ($-\sigma/4 < v_1 < \sigma/4$). During positive velocities, the final estimate, $\hat{v}_1$, of v_1 is dominated by positive nodes, which is illustrated with the dark blue line that depicts much higher nodal contributions for positive than negative nodes. The same is true of negative nodes during negative velocities. This trend is confirmed at a population level for both monkeys in Fig. 5c, where positive nodes dominate negative nodes at positive velocities and vice versa for negative velocities. Thus, the NN decoder is capable of optimizing for positive velocities by learning the weights of positive nodes and optimizing for negative velocities through the weights of negative nodes.

To understand whether the network is improving on the Kalman filter via separating movement contexts, we created an idealized decoder combining two separately trained Kalman filters: one for positive velocities (KF+) and one for negative velocities (KF−), as illustrated in Fig. 6a. The decoder in Fig. 6a assumes a perfect classifier that correctly chooses either KF+ or KF− depending on whether the true velocities are positive or negative. As can be seen for both monkeys in Fig. 6b, c, KF+ and KF− achieve higher velocity magnitudes closer to the NN decoder, and unlike the original Kalman filter, covers a wider range of velocities. This suggests that the NN allows for optimal fits within both of these contexts without overt switching.

Discussion

In 7 days of testing across two animals, neural network decoders (NN, RN), with higher-velocity and more natural appearing finger movements, achieved a 36% increase in throughput over the ReFIT Kalman filter (RK, RK_{opt}), with RK representing the current standard in finger control[22–24]. Even though not routinely performed in practice[25,28–30], optimizing the Kalman filter by allowing for 10 months of disproportionate practice and even empirically calibrating parameters of RK did not overcome the performance advantage of RN over RK, which maintained a substantial advantage of 45% in peak finger velocity and 14% advantage in throughput. In Monkey N, RN was directly compared against RK/RK_{opt} across >2500 trials, and RN had higher throughputs than RK/RK_{opt} every day. Also, in 6 days of testing across Monkeys N and W and >3600 trials, RN was indirectly compared against RK by first showing higher throughput of NN over RK on each day and then showing superior throughput of RN over NN (i.e., RN > NN, NN > RK). This improvement was driven by more accurately decoding higher velocities because the neural network better accesses high velocities and more accurately maps neural activity to intended finger velocities. More accurate high-velocity decodes were demonstrated in both online and offline analyses and may arise by separately training weights for either positive or negative velocities. The offline correlation analysis showing higher correlations for the neural network demonstrates, in both animals, that the neural network may better map neural activity to finger kinematics. Combined, these advantages may lead to more robust performance in various real-life tasks.

Neural network algorithms are loosely inspired by biological neural networks, and their use in BMIs has been explored for more than 20 years[13,14]. Decoding algorithms can be tested in open-loop or closed-loop mode. In open-loop testing, neural activity is obtained when the able-bodied animal performs a finger task with the hand manipulandum. From this pre-recorded neural activity, the position/velocity of the fingers is predicted and compared with the true finger position/velocity from the pre-recorded task, and many sophisticated non-linear and neural network architectures are known to show tremendous promise and even outperform linear algorithms in open-loop

https://doi.org/10.1038/s41467-022-34452-w

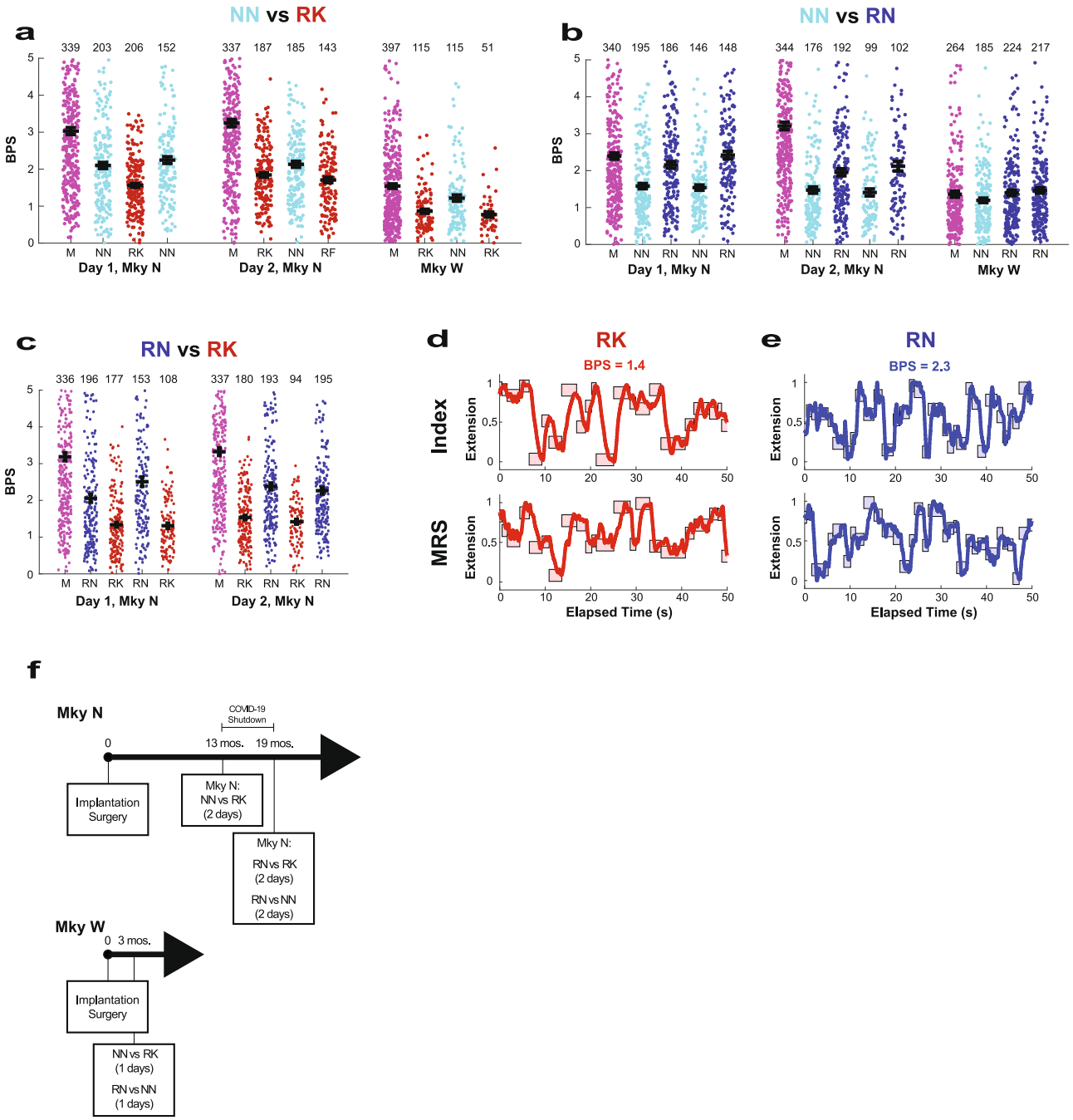

Fig. 3 | RN decoder outperforms RK during real-time tests. Throughput population data comparing decoders. For each run, the throughput is indicated with a dot, and the number of trials is printed above the data for each run. The few values greater than 5 bps are not shown. The black bars represent the mean and S.E.M. The manipulandum-control runs are indicated with "M" (magenta). **a** Throughput population data comparing the NN decoder (cyan) and the RK decoder (red) for 2 days of testing with Monkey N and 1 day of testing with Monkey W. **b** Throughput population data comparing the NN decoder (cyan) and RN decoder (blue) for 2 days of testing with Monkey N and 1 day of testing with Monkey W. **c** Throughput population data comparing the RN decoder (blue) and the RK decoder (red) for 2 days of testing with Monkey N. **d**, **e** Raw decoded position using the RK (red; **d**) and RN (blue; **e**) for the index finger (top pane) and middle-ring-small (MRS) fingers in Monkey N, which are locked together (bottom pane). The targets are represented as the shaded box. The x-axis denotes the elapsed time, 50 s, and the *y*-axis denotes the proportion of finger extension, i.e., 0 is fully flexed and 1 is fully extended. These time windows are representative of the average decoding performance as measured by throughput. **f** A timeline for data collection for Monkeys N and W. The data collection for Monkey N was interrupted by the COVID-19 lab shutdown. M manipulandum-control, NN neural network, RK ReFIT Kalman filter, RN ReFIT neural network, BPS bits per second. Source data are provided as a Source Data file.

mode[16,17,31,32]. In closed-loop mode, the decoding algorithm interprets neural activity in real-time and updates the position and velocity of the prosthesis. The closed-loop problem can be subdivided into two applications: classification and continuous motor decoding. In classification, neural activity is classified into a predefined set of prostheses positions, and in continuous decoding, neural activity is decoded onto a continuous range of positions/velocities that allow for arbitrary movements beyond a predefined set. Neural networks have been used previously in closed-loop classification[15,33,34]; however, in closed-loop continuous motor decoding, many attempts to improve performance over linear algorithms have been unsuccessful[13,14,18,19]. Sussillo et al.[30], which may be the most state-of-the-art demonstration of a neural

https://doi.org/10.1038/s41467-022-34452-w

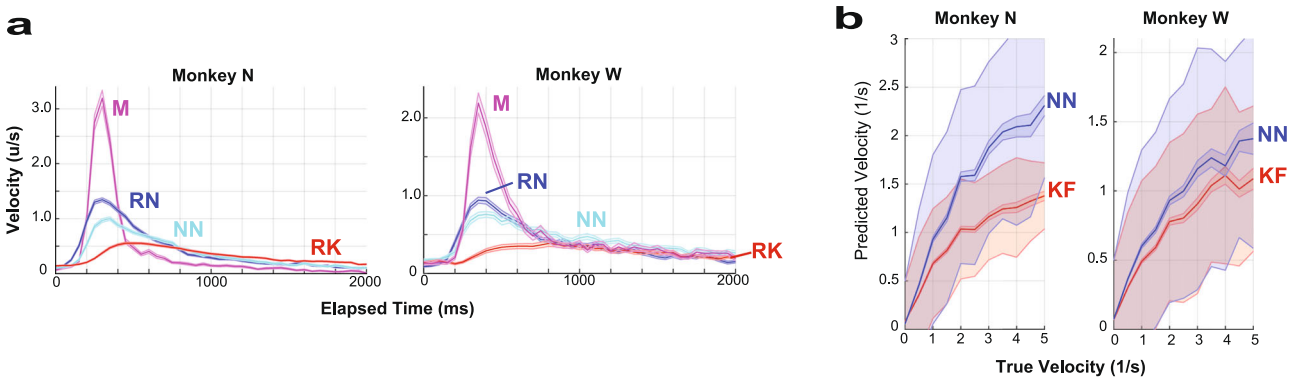

Fig. 4 | Higher decoded velocities using neural network decoders. a Online analysis. Virtual finger velocity for Monkey N (left pane) and Monkey W (right pane) for RN (blue), NN (cyan), RK (red), and manipulandum control (magenta). The plots indicate that the neural network decoders achieve higher peak velocities in real-time tests. In Monkey N, the RN, RK, and manipulandum -control data are taken from the days comparing RN vs. RK, while NN velocities were derived from the day comparing NN and RK. In Monkey W, RN data were derived from the day comparing RN vs. NN. RK was derived from the day comparing NN vs. RK. Manipulandum control and NN were derived from both days. The solid line indicates the mean value and the shaded region denotes the SEM. The shaded line tightly surrounds the mean, making these difficult to distinguish. If the trial was completed in <2000 ms, a velocity of zero was assigned extending from trial completion to 2000 ms. The unit, u, denotes arbitrary distance such that 1 was full flexion and 0 full extensions. **b** Offline analysis of true and predicted velocities for Monkey N (left) and Monkey W (right). The NN (blue) decodes higher velocities than the KF (red). The predicted velocity is normalized at a true velocity of zero so that one positive standard deviation of predicted velocities for both NN and KF is normalized to 0.5. The solid line indicates the mean value and the shaded region around denotes the SEM. The larger shaded region around the mean denotes the variance. NN neural network, RK ReFIT Kalman filter, RN ReFIT neural network, KF Kalman filter. Source data are provided as a Source Data file.

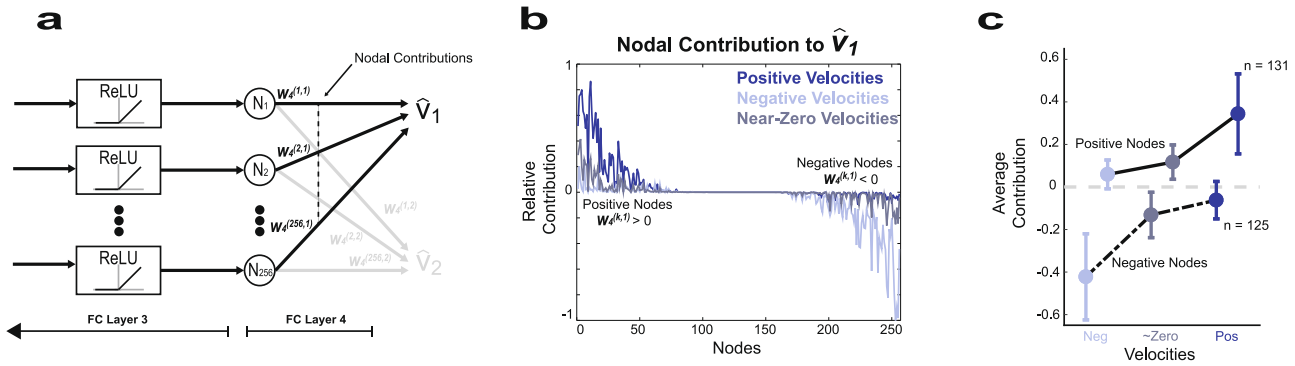

Fig. 5 | Neural network decoder functions as a weighted combination of a positive and negative velocity decoder. a Final neural network layer that converts 256 nodes into 2 finger velocities by matrix multiplication by a matrix $W_4^{(i,j)}$ of size 256 × 2, where i denotes the row and j denotes the column value. The value of the 256 nodes, N_k, of the final layer is all positive given that they are derived from the output of the preceding ReLU function. For finger 1, the kth node was considered a "positive node" if it contributes to positive—not negative—velocities and occurs when $W_4^{(k,1)} > 0$. "Negative nodes" contribute negative—not positive—velocities when $W_4^{(k,1)} < 0$. The nodal contribution from the kth node is defined as $N_k\, W_4^{(k,1)}$ and is the value of the output at the dashed line. **b** Example illustration of the mean nodal contribution from positive and negative nodes when the true velocity is positive (dark blue), negative (light blue), and near zero (gray) during 1 day of testing with Monkey N. Positive nodes are much higher than negative nodes during positive velocities, while negative nodes have a higher magnitude than positive nodes during negative velocities. Positive velocity is defined as $v_1 > \sigma$, negative velocity is defined as $v_1 < -\sigma$, and velocities near zero are defined as $-\sigma/4 < v_1 < \sigma/4$, where σ is the standard deviation of true finger velocity. **c** The nodal contribution during positive, negative, and near-zero true velocities illustrates that positive nodes largely determine the numerical value of positive velocities and negative nodes largely determine the value during negative velocities. The nodal contribution is averaged across both fingers during the 3 offline days for both Monkeys N and W. Thus, three operating regimes consisting of a decoder during positive velocities based on positive nodes, a decoder during negative velocities based on negative nodes, and a decoder using positive and negative nodes during velocities near zero. The dots indicate the mean and the error bars indicate the standard deviation. Source data are provided as a Source Data file.

network algorithm performing continuous online decoding, uses a recurrent neural network to demonstrate that day-to-day recording instabilities can be mitigated with the neural network. Although the RNN is compared with a ReFIT Kalman filter showing a small benefit, the neural network is allowed to train on substantially more data, and the Kalman filter implementation may not have been optimized through hyper-parameter tuning and animal practice.

Our approach differs from previous approaches mainly by using a shallow network architecture and the ReFIT innovation[6,24] to improve training data. Additionally, we utilize recently developed regularization techniques to prevent overfitting (i.e., batch normalization and dropout)[35–37] and also incorporate 150-ms time history of spiking band power (SBP) as input to the decoder instead of only one point in time. The shallow network architecture of 1 time-feature layer and 4 fully connected layers allows for the roughly 5×10^5 learned parameters to be trained with only 400 trials of same-day training data in about 1 min. In contrast, recurrent neural network architectures have combined training data across multiple days[15,16], and when implemented as continuous motor decoders may be too complex to run in real-time. An additional benefit of shallow feed-forward networks is that computing the velocity in real-time mode introduces only a 1–2 ms lag. Thus, through the use of a shallow network, limitations typical of more computationally expensive architectures are avoided. However, similar to other reports on neural networks[16,30], even this shallow neural

https://doi.org/10.1038/s41467-022-34452-w

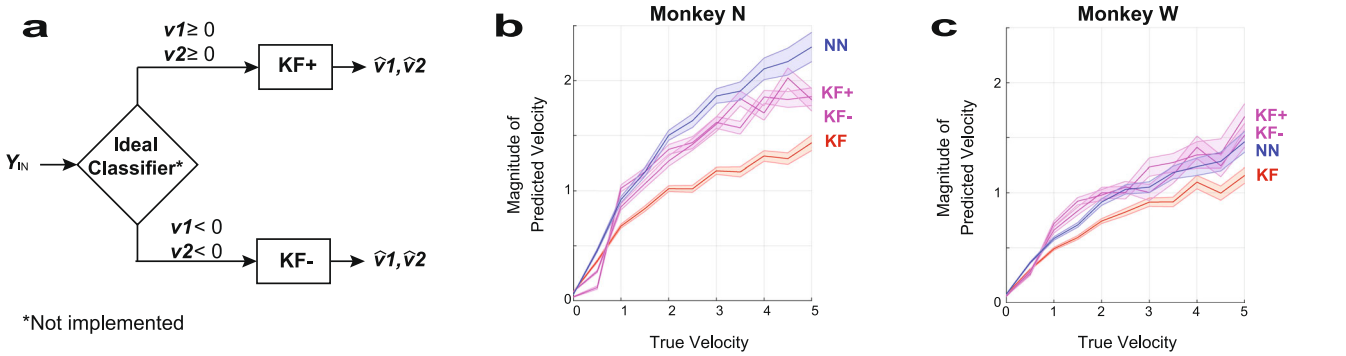

Fig. 6 | Hypothetical decoder for idealized decoder composed of a Kalman filter for positive and negative velocities. a Block diagram of a hypothetical decoder that uses two separate filters for positive finger velocities, $v_1 \geq 0$ and $v_2 \geq 0$, and negative finger velocities, $v_1 < 0$ and $v_2 < 0$. The Kalman filter for positive velocities, KF(+), was trained on velocities near zero and positive values ($>-0.5\sigma$), whereas the Kalman filter for negative velocities, KF(−), was trained on mainly negative velocities ($<0.5\sigma$). The ideal classifier is depicted only to illustrate the concept and was not implemented. **b**, **c** True versus predicted velocity magnitude during 3 days of manipulandum-control testing (3 days each for Monkeys N and W) for the neural network decoder (blue), Kalman filter (red), KF(+) (magenta), and KF(−) (magenta). The full-range Kalman filter predicted both positive and negative velocities, KF(+) predicted only positive velocities, and KF(−) predicted only negative velocities. The magnitude estimated velocities of KF(+) and KF(−) are shown to be higher than those of the full-range Kalman filter and illustrate that training and implementing the Kalman filter over restricted ranges would allow for higher velocities (assuming an ideal classifier). The solid line indicates the mean value and the shaded lines indicated the S.E.M. Source data are provided as a Source Data file.

network architecture may potentially achieve a performance boost from transfer learning if trained on previous days, which could reduce same-day training trials or may allow for deeper networks to be re-calibrated with a small amount of same-day data.

Incorporating a two-step, intention-based, re-training step is the fundamental innovation in improving the ReFIT over the classic velocity Kalman filter[6] and appears to have a similar positive effect on the NN. The intention-based retraining step was already known to improve finger decoding for a Kalman filter during center-out tasks[24]. Similar to the substantial improvement seen in the ReFIT Kalman filter[6], retraining the ReFIT neural network resulted in a substantial 35% improvement (3 days, 2 animals) in performance over the original neural network decoder. Despite the profound benefit of ReFIT, it is certainly possible that more sophisticated algorithms may be developed to better exploit transfer learning and allow for adaptive training algorithms, which may eventually reduce the training time and need for the ReFIT algorithm. However, further studies will be needed to characterize this tradeoff.

By training the neural network, the learned weights for either positive or negative nodes appear to be optimized for either the positive or negative velocity range. Similar to our results, Sachs et al.[12] showed that splitting the full velocity range into intervals subserved by separate Wiener filter decoders fine-tuned for either high or low velocities improved brain-machine interfaces for cursor control. Additionally, Kao et al.[7] improved performance by 4.2–13.9% over the ReFIT Kalman filter using a hidden Markov model to enable movement only when the decoder is in a "movement state," as neural activity is known to be different in movement and postural states[4]. The neural network architecture may be better able to discover these contexts without explicit classifiers or supervised training.

The variation of decoded velocities of the neural network appears to more closely mimic the range of velocities seen in native finger movements. In a tantalizing hypothesis, the decoder's naturalistic movements may be related to its shallow architecture, which may resemble true biological pathways. Specifically, there are only a few synapses between the neurons in the motor cortex and the α-motor neurons in the anterior horn of the spinal cord[38]. Although speculative, neural network architectures may perform well partly because motor cortex activity naturally controls the flexion and extension of antagonist muscle pairs. The neural network architecture may readily decode this flexion and extension into positive and negative velocities. Whether the similarity of neural network decoders to biological networks leads to more naturalistic motor control of many more contexts and simultaneous degrees of freedom was not investigated in this work and awaits further study.

As opposed to using our typical center-out finger task[23], we increased the difficulty to better challenge the decoders and elucidate differences between two well-performing decoders. Although these results could apply to a variety of real-world finger tasks, other tasks and prostheses (i.e., robotic arms) were not explicitly tested. While improvements in the Kalman filter implementation (including training and re-training procedures) may increase its performance, the neural network performance may also be further optimized in a similar way, such as by including position information into the decoder, optimally tuning its parameters, or by implementing it in a form similar to the steady-state Kalman filter by merging historical velocity and current updates (Eq. 2). Regardless, in online tests, the RN was found across all metrics to outperform RK, and offline tests confirmed a superior dimensionality reduction (as measured by correlation coefficient) and a higher dynamic range of predicted velocities.

A potential confounder of the improvement in using RN could be that the animals are more motivated to use RN over RK. To mitigate this confounder, the order of decoders used in online tests was reversed in multiple days of testing so that the animal used RK before RN. Additionally, animals had substantially more practice with RK from prior and parallel studies, which did not overcome the performance improvements of RN. Finally, motivational or decoder preferences are not applicable in offline testing where the animal controls virtual fingers using a hand manipulandum. In these tests, we found the neural network algorithm better predicts finger velocities and better models higher velocities than the Kalman filter.

A piecewise implementation of the Kalman filter could conceivably be used to achieve a similar range of predicted velocities but would require a sophisticated and generalizable switching algorithm to choose the appropriate Kalman filter. Furthermore, the neural network algorithms could similarly be constructed for distinct velocity ranges. Lastly, while the neural network does require increased computational complexity, it was optimized for performance and not to optimally trade off performance with computational complexity, which could certainly be accomplished.

This neural network decoder outperforms a current state-of-the-art motor decoder and achieves movements similar to naturalistic finger control. The architecture resembles biological motor pathways and may be amenable to further performance improvements.

Methods

Implantation procedure

The protocols herein were approved by the Institutional Animal Care and Use Committee at the University of Michigan. Two adult male rhesus macaques were implanted with Utah arrays (Blackrock Microsystems, Salt Lake City, Utah) in the primary motor cortex (M1). Under general anesthesia and sterile conditions, a craniotomy was made and M1 was exposed using standard neurosurgical techniques. The arcuate sulcus of M1 was visually identified, and the array was placed where this sulcus touches the motor cortex (Fig. 1a), which we have previously used as a landmark of the hand area in rhesus macaques. The incision was closed, and routine post-anesthesia care was administered.

Experimental setup and finger task

Both Monkeys N and W were trained to sit in a monkey chair (Crist Instrument, http://www.cristinstrument.com), with their head secured in customized titanium posts (Crist Instrument), while the Utah array was connected to the Cerebus neural signal processor (NSP, Blackrock Microsystems). The arms were secured in acrylic restraints. The hand contralateral to the motor cortex implant was placed in a manipulandum that translates finger position to a number between 0 (full extension) and 1 (full flexion). The impedance of a bend sensor was used to infer position and velocity. A computer monitor was in plain sight for the NHP and depicted a large virtual hand (Fig. 1b). The virtual finger could be controlled in either manipulandum-control mode or in brain-control mode (i.e., brain signals converted to updates for the virtual fingers). Brain-control mode is commonly denoted as either real-time, closed-loop, or "online" mode. Manipulandum-control mode is often described as "offline" mode. The two-dimensional finger task is identical to the task developed by Nason et al., except performed on random instead of center-out targets[23]. The finger task required placing either the virtual index and/or ring finger on the target for 750 ms during training mode and 500 ms during testing mode (testing vs. training modes will be explained in a subsequent section). The target size was 15% of the active range of motion. With target acquisition, apple juice was automatically administered through a tube placed in the animal's mouth. During experiments, animals were left alone in a dimly lit room and monitored on closed-circuit TV. They were free to move their hands within the acrylic restraints holding the arms, but the head was rigidly secured with the titanium posts. During experiments, the animals were cooperative and would participate in target acquisition. The animals would often move their hands during target acquisition, even when in brain control mode.

Front-end processing

The Utah array was connected to the Cerebus NSP (Blackrock Microsystems) through a cable. Although 96 channels were available, we accounted for changes in neuron count by only including channels that were not artifactual and had shown morphological neural spikes on the day of experiments or on previous days, leaving 54–64 channels for Monkey N and 50–53 channels for Monkey W. The Cerebus system sampled data at 30 kHz, filtered it to 300–1000 Hz, down-sampled it to 2 kHz, then transmitted it to the xPC Target environment version 2012b (Mathworks, Natick, MA). The xPC Target computer took the absolute value of the incoming data and then calculated each channel's mean in regular 50-ms time intervals. This binned value is referred to as spike-band power. We have previously shown that this band is highly correlated with and specific to the spiking rate of single units near the recording electrode[25]. A 50-ms time bin is a typical value used in the literature[6].

Software architecture

A separate computer with one 2070 super NVIDIA GPUs (NVIDIA, Santa Clara, CA) was connected to the xPC. This computing box was called the eXternal Graphic Processing Computer (xGPC). The xGPC executed commands in Python (v3.7, https://www.python.org/) using the PyTorch library (v1.4, https://pytorch.org/). Real-time performance was guaranteed in the following fashion. The xPC transmitted data to the xGPC with a timestamp and the xGPC calculated updates for the virtual fingers from the inputs (for all decoders) and transmitted the data back to the xPC along with the original timestamp. When the xPC received the data packet, the packet was logged with a new timestamp. Real-time performance was guaranteed given that the timestamp received from xGPC (the original timestamp sent by xPC) was within 50 ms of the current xPC timestamp and updates to the virtual fingers occurred every 50-ms time bin. The xGPC and xPC were assembled with the intention to use a much deeper neural network than was used in this work. The lag using this system with the neural network presented herein was only 1–2 ms.

ReFIT Kalman filter

The ReFIT Kalman filter (RK) was implemented, as we have done previously[39], for two-finger groups. for use with two fingers, as summarized by the equations below:

$$x_t = \begin{bmatrix} P_1 \\ P_2 \\ V_1 \\ V_2 \\ 1 \end{bmatrix} \tag{1}$$

$$\hat{x}_{t|t-1} = A\hat{x}_{t-1} \tag{2}$$

$$\hat{x}_t = \hat{x}_{t|t-1} + K_t(y_t - C\hat{x}_{t|t-1}) \tag{3}$$

$$A = \begin{bmatrix} 1 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & dt & 0 \\ 0 & 0 & A_{1V_1} & A_{V_2V_1} & 0 \\ 0 & 0 & A_{V_1V_2} & A_{V_2V_2} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

In Eq. (1), the kinematic state variable, x_t, is denoted and composed of the finger group positions (P_1 and P_2) and velocities (V_1 and V_2), and a value of 1 to account for the offset. The standard recursive implementation of the Kalman filter is described by Eqs. (2) and (3) where K_t is the Kalman gain (implemented without position uncertainty[24]), y_t denotes the SBP of the electrode array, and C denotes the weights calculated via linear regression to map kinematic variables to the array of SBP. A in Eq. (4) is implemented to obey the physical relationship between position and velocity where dt denotes the step size. Channels without morphological spikes were not used.

The trained KF was then used to perform closed-loop motor decoding. In closed-loop mode, the decoded position of the virtual fingers was calculated by adding the product of velocity and time step to the finger position at the previous time step. To train the RK, the target position for each finger is mapped to a two-dimensional space and the true velocity of each finger is scaled to be proportional to each finger's distance to the target while keeping the total velocity magnitude constant. This method of ReFIT was introduced by Nason et al.[23] and was not found to be statistically different from the ReFIT method in Vaskov et al.[24], where the finger velocity was modified by multiplying velocities by −1 when the velocity was oriented in the opposite direction as the target. The KF was then retrained using these new velocity values (for details see Nason et al.[23]).

Optimal lag is commonly implemented in KF motor decoders[40] to account for the physiologic lag between cortical activity and motor movement[41]. Thus an optimal time lag, calculated to be one 50-ms bin

for both Monkeys N and W, was applied when training and implementing the KF, as detailed in previous work[22,24]. Control tests comparing zero and one 50-ms bin lag are provided below (see Section "Optimizing the lag, gain, and scaling factors for real-time tests"). Additionally, the Kalman filter can be implemented as a steady-state Kalman filter with a gain and smoothing factor that can be optimized for online tests. Our implementation generally does not tune these parameters as the ReFIT training algorithm may determine near-optimal values for these parameters[42]. To validate this simplification, we compare our implementation of RK with one with optimal tuning (see Section "Optimizing the lag, gain, and scaling factors for real-time tests"). As will be explained below, we did compare RN with RK_{opt}, which uses zero lag and optimally determined gain/smoothing parameters, to ensure our results hold against a theoretically optimized RK, with the results presented in the section "ReFIT neural network decoder outperforming optimized RK decoder" of Results.

To determine whether the Kalman filter could better predict the high velocities if trained and used on restricted velocity ranges, we conducted an offline analysis using "KF+" and "KF−." KF+ was calculated with only positive and near-zero velocities, i.e., velocities greater than $-\sigma/2$, and KF− was calculated with velocities less than $\sigma/2$. These classic Kalman filters, as in Wu et al.[40], were implemented with a Kalman gain with position uncertainty, a fully learnable A matrix (Eq. (4)), and physiologic lag (calculated on that day).

Neural network velocity decoder

The neural network velocity decoder was designed from preliminary offline experiments that explored various network architectures. The final network is given in Fig. 1c. The first layer was the time feature layer that constructs time features from 150 ms (three 50-ms bins) from the input electrodes. This layer was implemented in Pytorch, using the *torch.nn.Conv1d* module, i.e., as a one-dimensional convolution with a kernel size of 1 (H = W = 1) and 3 input channels (neural network channels, not electrode channels). Each channel corresponded to one 50-ms time bin. Although possible to construct a spatial convolution across electrodes, this was not performed because the spacing between electrodes was hypothesized to be distant relative to the size of the neurons being recorded. The output of the time feature layer provided 16 features per electrode and, when flattened, provided output 16 multiplied by the number of electrodes used as the output of the time feature layer, which equals 1536 if all 96 electrodes are used. These outputs then form the input to a series of fully connected layers. Regularization for fully connected layers 1–3 included 50% dropout[35] and batch normalization[36]. Fully connected layer 1 converted the output channels of the time feature layer (up to 1536 in number if all 96 electrodes are used) to 256, and the remaining layers had 256 hidden neurons. The sequence of the modules used as *torch.nn.linear*, *torch.nn.Dropout*, *torch.nn.BatchNorm1d*, and then finally *torch.nn.functional.relu*. The final layer implemented a matrix multiplication with *torch.nn.linear* to convert the 256 inputs to the two velocity estimates. The output of the network was normalized to zero mean and unit variance and roughly twenty times the magnitude of actual velocity peaks. This normalization was discovered to converge more quickly when training the RN than training without the normalization. The output of the neural network was scaled by an unlearned gain factor that equaled the average magnitude peaks of the actual velocity divided by the average magnitude peaks of the predicted velocity. No offset was applied to the final predicted velocity, leaving it a zero-mean signal. A diagram of the final neural network is given in Fig. 1c.

Prior to training the neural network, a training data set was collected in the manipulandum-control mode for roughly 400 trials with randomly appearing targets. A subsequent 100 trials were also performed and served as a validation set to ensure the network had converged. This validation set was also used to calculate the gain as described above. If there was a non-zero median, this was subtracted as well to approximate a zero-mean signal. The SBP and velocity data were assembled into data structures in Matlab (Mathworks). The data were randomized in two ways. First, the time data were randomized into batches of 64×3 time points: 64-time points with the corresponding value at time delays of 0, 50, and 100 ms. Second, a triangular distribution of velocities was imposed on the training data spanning the range of -4σ to 4σ, where σ was the standard deviation of the actual velocity. A total of 20,000 training samples were randomly chosen to achieve this velocity distribution. This velocity redistribution was anecdotally observed to improve performance on the finger task when the neural network was trained on a center-out finger task that led to a "sticky finger" behavior, in which the finger would often get close to but not quite all the way to the target. However, when the neural network was trained on random targets, the velocity redistribution was not observed to improve performance over non-redistributed data, but we describe it here for completeness. This redistribution of velocities was also used when training on random finger targets so that the decoder could easily be generalized to other training paradigms in the future.

In addition to the neural network used for online testing, several other neural networks were used to understand how individual components of the neural network affected offline performance. The networks included a network of only two layers and no regularization (no batch normalization, dropout, or output normalization). There were also regularized networks, including a 2-layer fully connected network (256 hidden neurons) with and without a preceding time feature layer (3 input channels for each electrode and 16 output channels), and a 4-layer fully connected network (256 neurons) with a time feature layer. These networks included regularization and parameters similar to Fig. 1c. The offline networks were compared with the classic Kalman filter (without the intention retraining step) and ridge linear regression without time history and with a regularization constant of $\lambda = 10^{-4}$. During offline tests, algorithms were trained only on the day they were tested on.

When training the network for online decoding, the neural network was optimized over 3500 iterations using the Adam optimization algorithm[43] (torch.optim.Adam) with a learning rate of 10^{-4}, weight decay of 10^{-2}, and momentum of 0.9. Each iteration consisted of a 64×3 mini-batch (64 random time steps with 3 samples of 150 ms of time history). We attempted to use a relatively large learning rate as larger learning rates provide additional regularization for the network[37]. On one day for Monkey W, 3000 iterations were used. The number of iterations was determined for each network from the first of three offline testing days and chosen so that the correlation between actual and estimated velocity (on the testing set) did not significantly change with additional training iterations (changes in correlation with additional iterations on the order of ~0.01). When generating weights for offline analysis, a learning weight of 2×10^{-5} allowed better comparisons between networks with different numbers of layers. Kaiming initialization was used to initialize the weights of each layer[44], and the bias terms were initialized to zero. The dropout level used was 50%[35]. On each day, a training set (~400 trials) and testing set (~100 trials) were collected, and performance on a testing set was characterized by the correlation of predicted and actual velocity.

When searching for the preferred number of layers, hidden neurons, and output time features (Fig. 2a, b), performance was characterized by the average of the maximum five correlations with the testing set overall training iterations using 400 trials of training data over three days for each animal. In this way, the optimal number of training iterations did not need to be calculated for different-size networks.

The weights for the ReFIT neural network were calculated by first using the NN decoder in brain-control mode. A truth signal was then constructed from the original NN output by flipping the velocity direction whenever the estimated finger velocity was directed away

from the target. Using this truth signal and the original neural network output, 500 further iterations of the Adam optimization algorithm were applied to further optimize the weights of the neural network. The gain factor for the neural network was calculated in the same manner as the original neural network except when comparing the NN to RN over 2 days, where the gain factor used during brain control with the NN was simply scaled by a factor of 0.75 on both days.

Testing protocols

Targets for the fingers were not allowed to be separated by >50% of the range. During training, the random targets spanned 100% of the finger flexion/extension range, but during online decoding, only 95% of the range was used. The Kalman filter was then used on a ~250 trial run from which the ReFIT KF coefficients were calculated.

For Monkey N, 8 online testing days were conducted. Offline testing was performed using manipulandum-control trials for 3 consecutive days. Two of these days were the manipulandum-control training trials from the online testing comparing NN and RK conducted 13 mos post-implantation. An additional day of manipulandum-control data at 13 mos post-implantation was also included. For Monkey W, 2 online testing days were conducted. The offline analysis included manipulandum-control trials from 3 days at 2 mos post-implantation that included the 2 online days and an additional day of manipulandum-control trials. To reduce confounders when comparing decoders for each monkey, decoders were compared in an alternating lineup: either A-B-A or A-B-A-B testing. When decoders were evaluated on the second day of testing, the order was reversed: B-A-B or B-A-B-A. In 1 day for Monkey W, NN was compared with RN without alternating the decoder. For Monkey N, the first 50 trials with each decoder were discarded in the analysis. For Monkey W, only the first trial was discarded as there were fewer total trials since W was less motivated to complete trials. To visually illustrate the peak performance of the RN on our typical center-out task[23], one additional day was included using the RN on this task. To prevent animal motivation from confounding the analysis, the run was terminated if the animal made no attempt to acquire targets. Only trials with targets that did not overlap with the previous trial's targets were included in the analysis.

Performance assessment and statistical analysis

Data analysis was performed using computers with built-in code and customized code in Matlab (Mathworks, Natick, MA) versions R2017a, R2018a, PyCharm 2020–2022, and Python v3.7/v3.9, PyTorch v1.5/v1.12.

Performance in online mode was characterized by Fitt's law throughput given below in Eq. (5), which accounts for both task difficulty and the time needed for completion. The variable D_k is the distance of the kth virtual finger to the center of the kth target at the start of the task, S is the target radius (equal in both fingers), and t_{acq} is the time to reach the target.

$$\text{Throughput} = \frac{\sum_k \log_2\left(1 + \frac{(D_k - S)}{2S}\right)}{t_{acq}} \tag{5}$$

While the throughput was the primary performance metric, acquisition times were also reported.

All velocities in the offline analyses were normalized by the standard deviation of the true velocity with 1 indicating the equivalent of 1 standard deviation of the actual velocity. The time plots depicting the actual versus predicted velocity were selected from one of the training days to illustrate the results (Fig. 2a). The correlation for each decoder was averaged over 2 fingers on 3 days (Fig. 2b). The plots of true versus predicted velocity were calculated by binning the magnitude of the actual velocity into bins of size 1.0 at intervals of 0.5 and averaging the magnitude of the predicted velocity in each respective bin.

A value of $\alpha = 0.05$ was used for statistical significance. The correlation coefficient was calculated with numpy.correlate and normalized to vary between 0 and 1. The primary endpoint for this analysis was whether the correlation of NN subtracted by the correlation of KF, was greater than zero, which was evaluated with a one-sample, two-tailed t-test (ttest.m). ANOVA post-hoc comparisons of each network (anova1.m) are also included to compare the effectiveness of various neural network features (i.e., network regularization, time history, number of layers). Performance metrics during real-time tests, including throughput and acquisition times, were compared with a two-sample, two-sided t-test (ttest2.m). Performance metrics are reported as mean value ± standard error of the mean (SEM).

Optimizing the lag, gain, and scaling factors for real-time tests

As explained above, we utilized physiologic lag similar to previous studies[40] in our implementation of RK for finger control[22–24]. Unless otherwise mentioned, our comparisons of RK and RN use RK with a 50-ms bin lag. To evaluate the effect of a 50-ms bin lag, RK with a lag of one 50-ms bin was compared to RK with a zero-lag implementation (where the Kalman filter predictions update the virtual hand as soon as they are available) in one day of testing over 273 trials with Monkey N. The zero-lag RK improved throughput by 16% and peak average velocity by 13%.

The Kalman filter estimates of position and velocity can be simplified (assuming a steady-state Kalman gain) as a weighted sum of two components: the previous time step's estimate of position/velocity and the current time step's estimate of position/velocity derived from the intra-cortical array as shown below in Eq. (6)[27].

$$\bar{x}_t = \alpha \bar{x}_{t-1} + (1 - \alpha)\beta D \bar{y}_t \tag{6}$$

In Eq. (6), $\bar{x}_t$ denotes a $2F_N \times 1$ column vector of position and velocity estimates for F_N fingers, D is a $E_N \times 2F_N$ matrix for E_N electrodes, α is the smoothing factor, and β is the gain factor. For the position-velocity Kalman filter, the smoothing factor and gain were only implemented for the velocity kinematics. To determine the optimal values for α and β, one day of testing with RK was dedicated to first tuning the gain, β, by increasing its value. Using the value of β giving the best performance, the smoothing factor, α, was then adjusted. The optimal values, based on throughput, were found by scaling trained values of α term by a factor of 1.25 and β by 1.2. The performance improvements from using a zero-lag RK and optimally tuned gain and smoothing factors were tested on the second day of testing. Using RK with optimally tuned parameters (475 trials) resulted in only a 3.6% increase in throughput and a 16% increase in average peak finger velocity.

Although most comparisons of RN and RK use RK without tuned hyperparameters, we did include a test RN and $\mathrm{RK_{opt}}$, which uses zero lag and optimal values of α scaled by 1.25 and β by 1.2, and validated our findings with a fully optimized RK at 29 mos post-implantation (see Results Section "ReFIT neural network decoder outperforms both the original NN and RK decoders").

Reporting summary

Further information on research design is available in the Nature Portfolio Reporting Summary linked to this article.

Data availability

The source data generated in this study have been provided within this paper, as supplementary data with this manuscript, and also available on the lab website [https://chestekresearch.engin.umich.edu/data-and-resources/]. The raw datasets used for this study are too large to be publicly shared, yet they are available for research purposes from

the corresponding author on reasonable request. Source data are provided with this paper.

Code availability

The python code for the neural networks is available on the lab website [https://chestekresearch.engin.umich.edu/data-and-resources/].

References

1. Armour, B. S., Courtney-Long, E. A., Fox, M. H., Fredine, H. & Cahill, A. Prevalence and causes of paralysis—United States, 2013. *Am. J. Public Health* **106**, 1855–1857 (2016).
2. Hochberg, L. R. et al. Reach and grasp by people with tetraplegia using a neurally controlled robotic arm. *Nature* **485**, 372–375 (2012).
3. Collinger, J. L. et al. High-performance neuroprosthetic control by an individual with tetraplegia. *Lancet* **381**, 557–564 (2013).
4. Kurtzer, I., Herter, T. M. & Scott, S. H. Random change in cortical load representation suggests distinct control of posture and movement. *Nat. Neurosci.* **8**, 498–504 (2005).
5. Bensmaia, S. J. & Miller, L. E. Restoring sensorimotor function through intracortical interfaces: progress and looming challenges. *Nat. Rev. Neurosci.* **15**, 313–325 (2014).
6. Gilja, V. et al. A high-performance neural prosthesis enabled by control algorithm design. *Nat. Neurosci.* **15**, 1752 (2012).
7. Kao, J. C., Nuyujukian, P., Ryu, S. I. & Shenoy, K. V. A high-performance neural prosthesis incorporating discrete state selection with hidden Markov models. *IEEE Trans. Biomed. Eng.* **64**, 935–945 (2016).
8. Shanechi, M. M. et al. A real-time brain–machine interface combining motor target and trajectory intent using an optimal feedback control design. *PLoS ONE* **8**, e59049 https://doi.org/10.1371/journal.pone.0059049 (2013).
9. Velliste, M. et al. Motor cortical correlates of arm resting in the context of a reaching task and implications for prosthetic control. *J. Neurosci.* **34**, 6011–6022 (2014).
10. Mulliken, G. H., Musallam, S. & Andersen, R. A. Decoding trajectories from posterior parietal cortex ensembles. *J. Neurosci.* **28**, 12913–12926 (2008).
11. Yu, B. M. et al. Mixture of trajectory models for neural decoding of goal-directed movements. *J. Neurophysiol.* **97**, 3763–3780 (2007).
12. Sachs, N. A., Ruiz-Torres, R., Perreault, E. J. & Miller, L. E. Brain-state classification and a dual-state decoder dramatically improve the control of cursor movement through a brain-machine interface. *J. Neural Eng.* **13**, 016009 (2015).
13. Carmena, J. M. et al. Learning to control a brain–machine interface for reaching and grasping by primates. *PLoS ONE* **1**, e42 https://doi.org/10.1371/journal.pbio.0000042 (2003).
14. Wessberg, J. et al. Real-time prediction of hand trajectory by ensembles of cortical neurons in primates. *Nature* **408**, 361–365 (2000).
15. Willett, F. R., Avansino, D. T., Hochberg, L. R., Henderson, J. M. & Shenoy, K. V. High-performance brain-to-text communication via handwriting. *Nature* **593**, 249–254 (2021).
16. Pandarinath, C. et al. Inferring single-trial neural population dynamics using sequential auto-encoders. *Nat. Methods* **15**, 805–815 (2018).
17. Hosman, T. et al. in *9th International IEEE/EMBS Conference on Neural Engineering (NER)* 1066–1071 (IEEE, 2019).
18. Sussillo, D. et al. A recurrent neural network for closed-loop intracortical brain–machine interface decoders. *J. Neural Eng.* **9**, 026027 (2012).
19. George, J. A., Brinton, M. R., Duncan, C. C., Hutchinson, D. T. & Clark, G. A. in *40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* 3782–3787 (IEEE, 2018).
20. Shanechi, M. M., Orsborn, A. L. & Carmena, J. M. Robust brain-machine interface design using optimal feedback control modeling and adaptive point process filtering. *PLoS Comput. Biol.* **12**, e1004730 (2016).
21. Taylor, D. M., Tillery, S. I. H. & Schwartz, A. B. Direct cortical control of 3D neuroprosthetic devices. *Science* **296**, 1829–1832 (2002).
22. Irwin, Z. et al. Neural control of finger movement via intracortical brain–machine interface. *J. Neural Eng.* **14**, 066004 (2017).
23. Nason, S. R. et al. Real-time linear prediction of simultaneous and independent movements of two finger groups using an intracortical brain-machine interface. Neuron 109, 3164–3177. e3168 (2021).
24. Vaskov, A. K. et al. Cortical decoding of individual finger group motions using ReFIT Kalman filter. *Front. Neurosci.* **12**, 751 (2018).
25. Nason, S. R. et al. A low-power band of neuronal spiking activity dominated by local single units improves the performance of brain-machine interfaces. *Nat. Biomed. Eng.* **4**, 973–983 (2020).
26. Willett, F. R., Suminski, A. J., Fagg, A. H. & Hatsopoulos, N. G. Improving brain–machine interface performance by decoding intended future movements. *J. Neural Eng.* **10**, 026011 (2013).
27. Willett, F. R. et al. A comparison of intention estimation methods for decoder calibration in intracortical brain–computer interfaces. *IEEE Trans. Biomed. Eng.* **65**, 2066–2078 (2017).
28. Pandarinath, C. et al. High performance communication by people with paralysis using an intracortical brain-computer interface. *Elife* **6**, e18554 (2017).
29. Shanechi, M. M. et al. Rapid control and feedback rates enhance neuroprosthetic control. *Nat. Commun.* **8**, 1–10 (2017).
30. Sussillo, D., Stavisky, S. D., Kao, J. C., Ryu, S. I. & Shenoy, K. V. Making brain–machine interfaces robust to future neural variability. *Nat. Commun.* **7**, 1–13 (2016).
31. Glaser, J. I. et al. Machine learning for neural decoding. *Eneuro* **7**, ENEURO.0506-19.2020 https://doi.org/10.1523/ENEURO.0506-19.2020 (2020).
32. Allahgholizadeh Haghi, B. et al. Deep multi-state dynamic recurrent neural networks operating on wavelet based neural features for robust brain–machine interfaces. *Adv. Neural Inf. Process. Syst.* **32**, (2019).
33. Skomrock, N. D. et al. A characterization of brain-computer interface performance trade-offs using support vector machines and deep neural networks to decode movement intent. *Front. Neurosci.* **12**, 763 (2018).
34. Schwemmer, M. A. et al. Meeting brain–computer interface user performance expectations using a deep neural network decoding framework. *Nat. Med.* **24**, 1669–1676 (2018).
35. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **15**, 1929–1958 (2014).
36. Ioffe, S. & Szegedy, C. in *2015 International Conference on Machine Learning* (eds Bach, F. & Blei, D.) 448–456 (PMLR, 2015).
37. Li, Y., Wei, C. & Ma, T. Towards explaining the regularization effect of initial large learning rate in training neural networks. In *Advances in Neural Information Processing Systems 32*. (eds H. Wallach et al.) (2019).
38. Rathelot, J.-A. & Strick, P. L. Subdivisions of primary motor cortex based on cortico-motoneuronal cells. *Proc. Natl Acad. Sci. USA* **106**, 918–923 (2009).
39. Nason, S. R. et al. Real-time linear prediction of simultaneous and independent movements of two finger groups using an intracortical brain-machine interface. *Neuron* **109**, 3164–3177. e3168 (2021).
40. Wu, W. et al. Modeling and decoding motor cortical activity using a switching Kalman filter. *IEEE Trans. Biomed. Eng.* **51**, 933–942 (2004).
41. Moran, D. W. & Schwartz, A. B. Motor cortical representation of speed and direction during reaching. *J. Neurophysiol.* **82**, 2676–2692 (1999).

42. Willett, F. R. et al. Principled BCI decoder design and parameter selection using a feedback control model. *Sci. Rep.* **9**, 1–17 (2019).
43. Kingma, D. P. & Ba, J. Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014).
44. He, K., Zhang, X., Ren, S. & Sun, J. in *Proc. IEEE International Conference on Computer Vision* 1026–1034 (IEEE, 2015).

Acknowledgements

We thank Eric Kennedy and Paras Patel for animal and experimental support. We thank Gail Rising, Amber Yanovich, Lisa Burlingame, Patrick Lester, Veronica Dunivant, Laura Durham, Taryn Hetrick, Helen Noack, Deanna Renner, Michael Bradley, Goldia Chan, Kelsey Cornelius, Courtney Hunter, Lauren Krueger, Russell Nichols, Brooke Pallas, Catherine Si, Anna Skorupski, Jessica Xu, and Jibing Yang for expert surgical assistance and veterinary care. We thank Tom Cichonski for his editorial review. This work was primarily supported by NSF grant 1926576 (M.S.W., H.T., M.J.M., P.G.P., C.A.C.); NIH grant F31HD098804 (S.R.N.); NSF GRFP (J.T.C.); the A. Alfred Taubman Medical Research Institute (P.G.P.); NIH grant R01GM111293 (P.G.P., C.A.C.); Craig H. Neilsen Foundation project 315108 (C.A.C.); MCubed project 1482 (C.A.C.).

Author contributions

M.S.W., S.R.E., H.T., M.J.M., and J.T.C. performed the NHP decoding experiments. M.S.W. conducted the data analysis and wrote the manuscript. M.S.W. and S.R.N. wrote and validated the analysis software. M.S.W. and S.R.E. wrote and validated the Python code. S.R.E. developed and tested the xGPC. M.S.W., P.G.P., and C.A.C. conducted the surgeries. P.G.P. and C.A.C. supervised this work. All authors reviewed and modified the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s41467-022-34452-w.

Correspondence and requests for materials should be addressed to Cynthia A. Chestek.

Peer review information *Nature Communications* thanks Christian Klaes and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.