ARTICLES
https://doi.org/10.1038/s41551-021-00811-z

nature biomedical engineering

Rapid adaptation of brain–computer interfaces to new neuronal ensembles or participants via generative modelling

Shixian Wen[1]✉, Allen Yin[2], Tommaso Furlanello[1], M. G. Perich[3], L. E. Miller[4] and Laurent Itti[1]✉

For brain–computer interfaces (BCIs), obtaining sufficient training data for algorithms that map neural signals onto actions can be difficult, expensive or even impossible. Here we report the development and use of a generative model—a model that synthesizes a virtually unlimited number of new data distributions from a learned data distribution—that learns mappings between hand kinematics and the associated neural spike trains. The generative spike-train synthesizer is trained on data from one recording session with a monkey performing a reaching task and can be rapidly adapted to new sessions or monkeys by using limited additional neural data. We show that the model can be adapted to synthesize new spike trains, accelerating the training and improving the generalization of BCI decoders. The approach is fully data-driven, and hence, applicable to applications of BCIs beyond motor control.

A motor brain–computer interface[1] (BCI) is a system that enables users to control artificial actuators or even the contraction of paralyzed muscles[2,3], by decoding motor output from recorded neural activity. Many methods have been proposed to build such a decoder, including linear Wiener Filters[1,4], Kalman Filters[5,6], particle Filters[7,8], point process methods[9,10] and long short-term memory (LSTM)[11,12] networks. However, current BCI decoders face several limitations. First, the most powerful of these methods (LSTM) typically require large amounts of neural data to achieve good performance. Second, these decoders typically generalize poorly over time, requiring periodic recalibration. Lastly, decoders are user-specific and must be trained from scratch for each subject. This poses problems for clinical applications, where training data is difficult to acquire.

The applicability of BCI decoders could be greatly improved if they could generalize across recording sessions and to new subjects. In the cross-session scenario, a decoder trained with data from one recording session is expected to generalize to data from another session, despite possibly having different sets of recorded neurons. This scenario is important for designing BCI decoders that can maintain decoding performance despite the presence of glial scarring[13,14], relative motion between electrodes and brain or cell death[14], all of which may change the effective number and identity of recorded neurons from day to day. Even if those problems can be minimized, natural neural plasticity[15] might still require adaptation of the decoder over time. In the cross-subject scenario, a decoder trained with data from one subject would be expected to generalize to data from another subject, usually also with a different number of neurons, possibly after some limited additional training using data from the new subject. Leveraging data from the first subject (or subjects) would be beneficial, for example, when simultaneously obtaining sufficient neural data and covariates of interest from subsequent subjects is expensive, difficult or impossible[16] (for example, when paralyzed patients cannot generate motor outputs or when it is difficult to track complex-task variables). In addition, neural data from the first subject might be inherently easier to decode (for instance, the quality of the signal collected by the implanted electrode arrays might be better).

Even with ample available data, cross-session and cross-subject adaptation could leverage similar structures in the neural data despite differences in the recordings[17,18]. Recently, it has been shown[19] that a latent dynamic system trained on neural data from multiple sessions can predict movement kinematics from additional neural data of these sessions, but the study did not show generalization to completely new sessions or subjects. Another study[20] showed that adversarial domain adaptation of the latent representation of firing rates could predict movement kinematics from latent-signal inputs over many days using a fixed decoder. However, BCI decoders that can generalize to different sessions[21,22] or subjects without complete re-training have not yet been shown. We believe that this is because current approaches lack a principled representation of neural attributes (such as position, velocity and acceleration activity maps, velocity neural-tuning curves, distribution of mean firing rates and correlations between spike trains)[23].

Here we leverage state-of-the-art machine-learning techniques[24–29] to explore the interactions between a generative spike synthesizer and a BCI decoder to improve generalization across sessions and subjects. We trained a deep-learning spike synthesizer with one session of motor cortical neural population data recorded from a reaching monkey. The spike synthesizer can synthesize realistic spike trains with realistic neural attributes. With the help of the synthesized spike trains, our results show how the training of BCI decoders can be accelerated and how the generalization of BCI decoders across sessions and subjects can be improved. With small amounts of training data, we show modest yet significantly improved training of a BCI decoder in cross-session and cross-subject decodings. Furthermore, we show that our method can transfer some useful information and boost cross-subject decoding performance beyond the best achievable performance, by training only on data from the second subject.

[1]University of Southern California, Los Angeles, CA, USA. [2]Facebook, Menlo Park, CA, USA. [3]University of Geneva, Geneva, Switzerland. [4]Northwestern University, Chicago, IL, USA. ✉e-mail: shixianw@usc.edu; itti@usc.edu

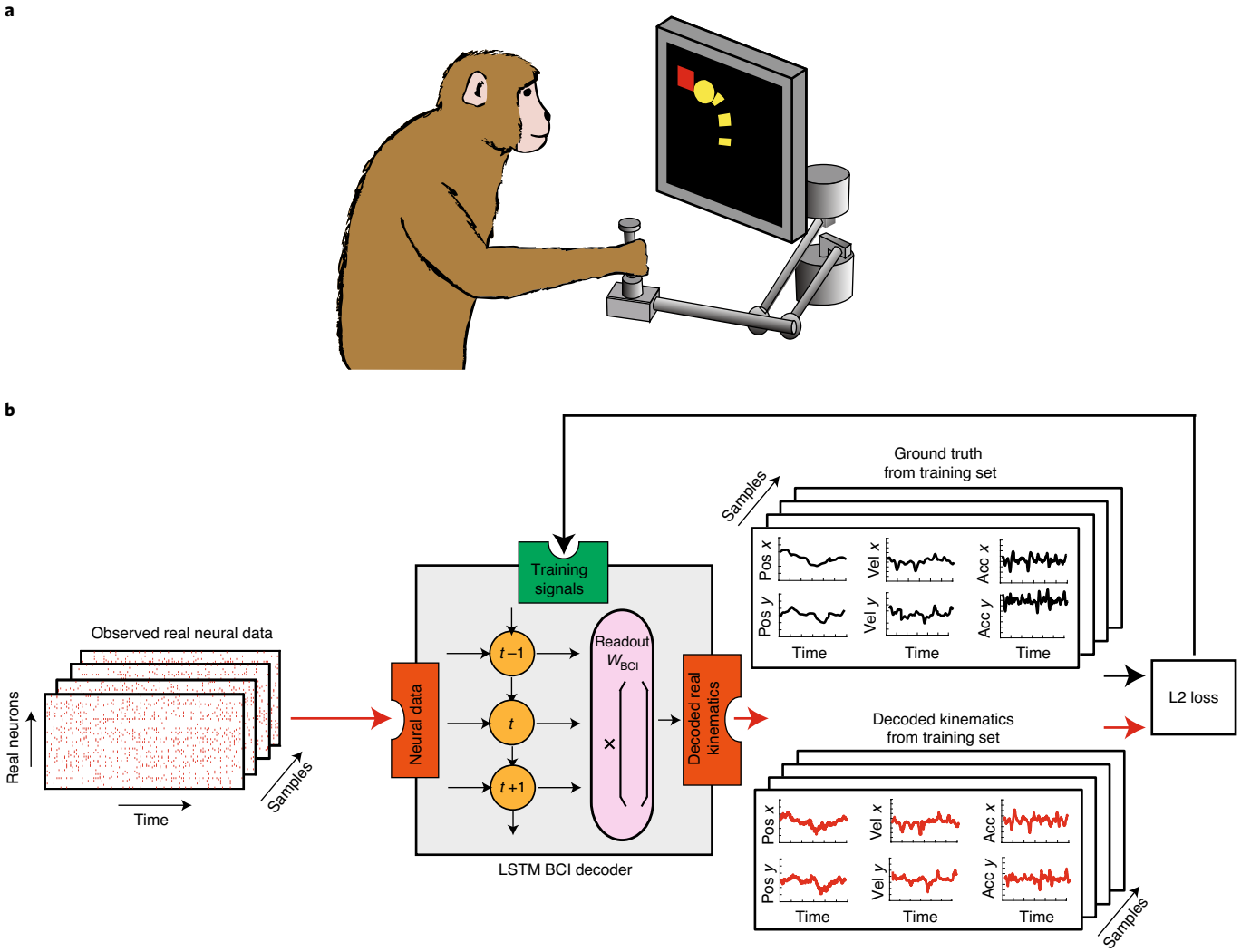


Fig. 1 | Experimental paradigm and training the baseline BCI LSTM decoder. a, Experimental paradigm: monkeys were seated in front of a video screen and grasped the handle of a planar manipulandum that controlled the position of a cursor. Monkeys made reaching movements to a sequence of randomly placed targets appearing on the screen while we recorded neural activity in the primary motor cortex using an implanted electrode array. **b**, Training the baseline BCI LSTM decoder. Recorded spike trains are input to the BCI LSTM decoder. The decoder outputs predicted kinematics by first learning a time-varying generalizable internal representation (symbols $t-1$, t, $t+1$), and then mapping it to kinematic space (using readout weights W_{BCI}). During training, actual kinematics (ground truth) were compared to the predicted ones using an L2 loss function (that is, a Euclidean distance comparison) and used to refine the decoder (Methods).

Results

Experimental setup and data preparation. Two monkeys (Monkey C and Monkey M) were chronically implanted with electrode arrays (Blackrock microsystems) in the arm representation of the primary motor cortex (M1). The monkeys were seated in front of a video screen and grasped the handle of a planar manipulandum that controlled the position of a cursor. We recorded neural spiking activity on each electrode while the monkeys made reaching movements to a sequence of targets appearing in random locations on the screen[30]. After the cursor reached a given target, a new target appeared, which the monkeys could reach immediately (Fig. 1a). We collected two sessions of neural data: one with 33 min and 69 neurons (session 1), and the other with 37 min and 77 neurons (session 2) from Monkey C. We also collected one session with 11 min and 60 neurons (session 1) from Monkey M. We parsed and binned all neural and kinematic data with 10 ms time resolution.

Spike synthesizer and general structure. Generative adversarial networks[24–29] (GAN) can learn, end-to-end, a mapping from a random noise input to an output point that belongs to a desired data distribution. The process of training a GAN can be thought of as an adversarial game between a generator and a discriminator. The role of the generator is to produce fake data that seem real, while the discriminator learns to recognize whether data are real or fake. Competition in this adversarial game can improve both components until the generated fake data are indistinguishable from real data. The random noise input allows the GAN to synthesize different instances or variations around the desired target point. For example, in computer vision, after training a GAN on images of people with various hairstyles and clothes, the generator can synthesize, for any new person, many realistic-looking images of that person with different hairstyles[25] or with different clothes[26]. Current deep generative models can only generate samples from the distribution they have been trained on. Thus, GANs cannot generalize to new kinds

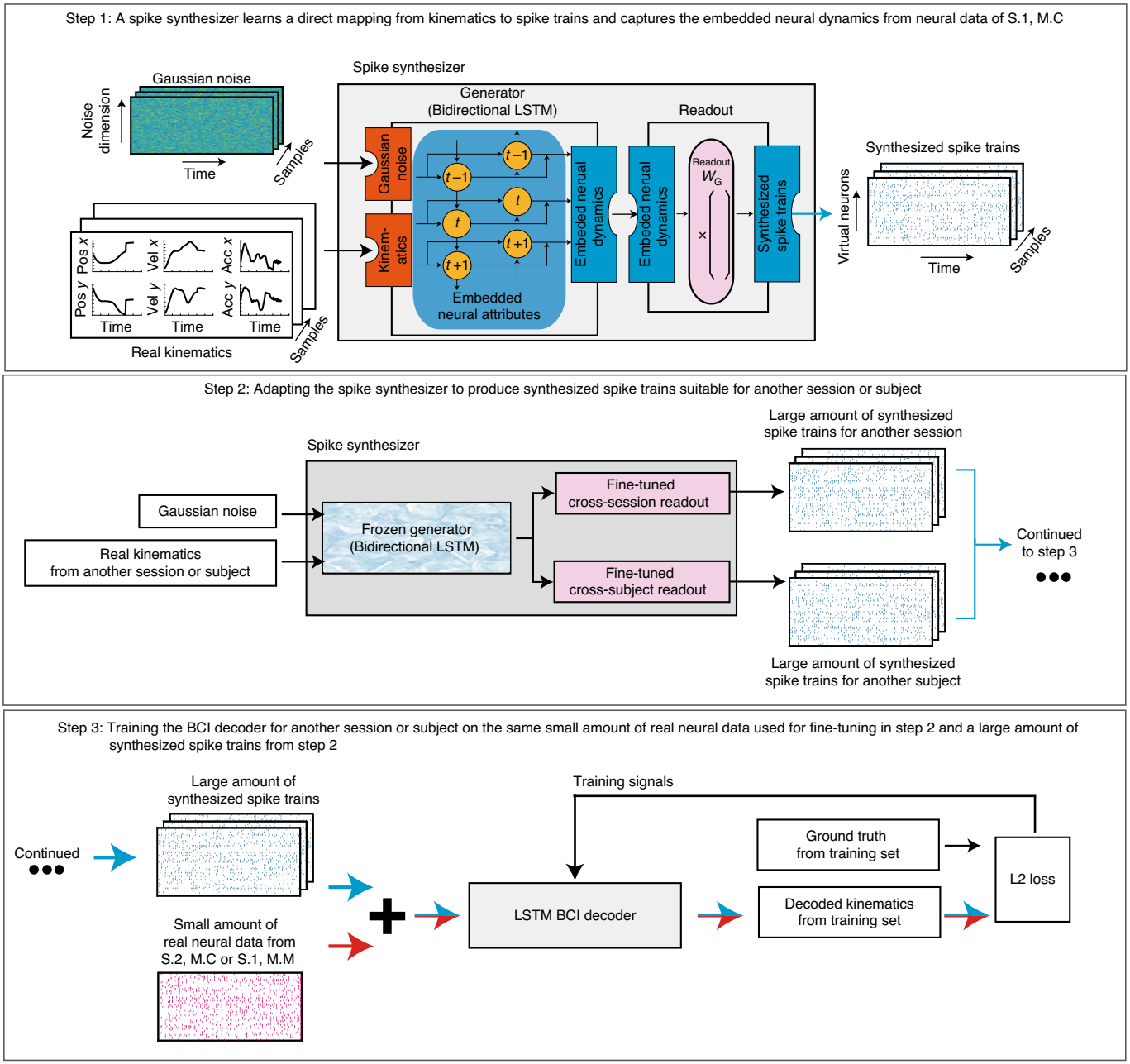


Fig. 2 | General framework. Step 1. Training a spike synthesizer on the neural data from session 1 of Monkey C (S.1, M.C) to learn a direct mapping from kinematics to spike trains and to capture the embedded neural attributes. Gaussian noise and real kinematics are input to the spike synthesizer (consisting of a generator and a readout). The spike synthesizer generates realistic synthesized spike trains by first learning the embedded neural attributes using a generator (a bidirectional LSTM recurrent neural network) through a bidirectional time-varying generalizable internal representation (symbols $t-1$, t, $t+1$). Different instances of Gaussian noise combined with new kinematics yield different embedded neural attributes that all have similar properties to those used for training. Then, the readout maps the embedded neural attributes to spike trains (using readout weight W_G). **Step 2**. Adapting the spike synthesizer to produce synthesized spike trains suitable for another session or subject from real kinematics and Gaussian noise. We first freeze the generator to preserve the embedded neural attributes or virtual neurons learned previously. Then, we substitute and fine-tune the readout modules using a limited neural data from another session or subject (session 2 of Monkey C (S.2, M.C) or session 1 of Monkey M (S.1, M.M)). The fine-tuned readout modules adapt the captured expression of these neural attributes into spike trains suitable for another session or subject. **Step 3**. Training a BCI decoder for another session or subject using the combination of the same small amount of real neural data used for fine-tuning (in **Step 2**) and a large amount of synthesized spike trains (in **Step 2**).

of data that they have never been trained with (for example, a GAN trained on images of shirts cannot generate images of pants).

Our synthesizer (Fig. 2, step 1, a sequential adaptation of a GAN) learns a direct mapping from hand kinematics to spike trains. This is achieved through the training of a new type of spike-based GAN (Methods and Extended Data Fig. 1, Step 1). After training, it can capture the embedded neural attributes. As with machine-vision GANs that cannot generalize from shirts to pants, here we expect that our model can synthesize new spike trains with good neural attributes for kinematics seen during training, but cannot generalize

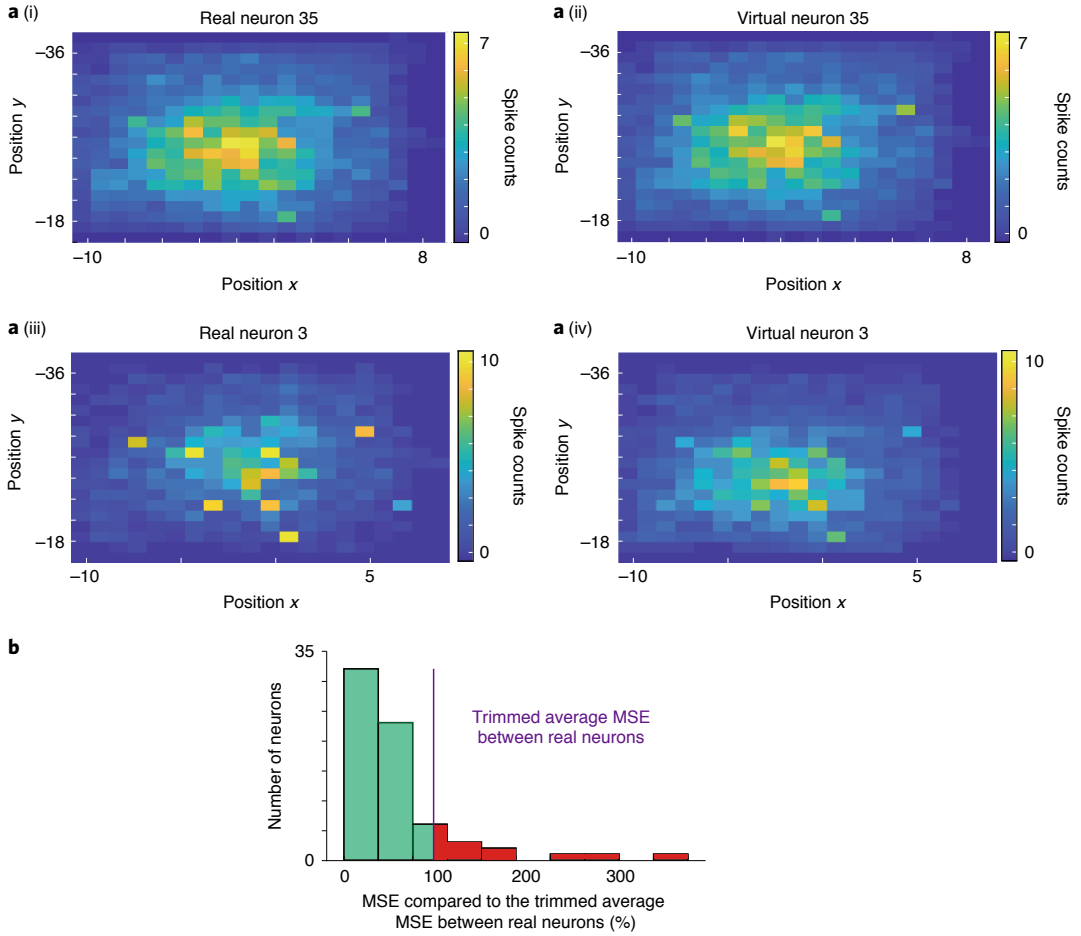


Fig. 3 | Normalized position activity map, constructed as the histogram of neural activity as a function of position. a, Position activity map for real neuron 35 (i) normalized across the workspace and corresponding position activity map for virtual neuron 35 (ii). Position activity maps for real (iii) and virtual (iv) neuron 3. **b**, Histogram of mean squared error (MSE) between the real and generated activity maps for all neurons. The purple line is the trimmed averaged MSE (based on 99% of samples, 0.13) between real neurons. It provides a reasonable bound for quantifying the difference between real and virtual neurons.The green (red) bars indicate that neurons have an error less (larger) than the average.

to new kinematics never encountered at training time. Next, new neural data from a second session, or a different subject, is split into training and test sets. The training set, which can be small (for example, 35 s data), is used to adapt[31,32] the synthesizer to the new domain (Fig. 2, Step 2). Once adapted, the synthesizer outputs spike trains that emulate the properties of the new data, and thus can be used to assist the training of a BCI decoder for that session. To train the BCI decoder (Fig. 2, Step 3), the synthesized spike trains are combined with the same limited training set that was used to adapt the synthesizer. We show that training on this combination of real and synthesized spike trains yields a better BCI decoder than training on the limited neural data alone, because the spike synthesizer notably increases the diversity and quantity of neural data available to train the BCI decoder. In essence, this achieves smart data augmentation[30,33], whereby a limited amount of new real neural data combines with and adapts a previously learned mapping from kinematics to spike trains, delivering an updated mapping and synthesizer that can generate sufficiently realistic data to effectively train the BCI decoder.

Validation of the spike synthesizer using a random reaching task. We trained the spike synthesizer on session 1 of Monkey C, and characterized both the recorded and virtual neurons using properties such as firing rates and position, velocity and acceleration activity maps. In addition, we measured the correlations between the recorded real and virtual neurons.

Can the spike synthesizer synthesize spikes with realistic position (velocity, acceleration)-related activity? To answer this question, we compared the position, velocity and acceleration activity maps built from synthesized spike trains to those of actual spikes. We counted the number of spikes for different hand positions and normalized them with respect to the averaged spike counts across the workspace. Figure 3a(i) shows the normalized position activity map for real neuron 35, and Fig. 3a(ii) shows its virtual counterpart. The mean square error (MSE) between the two maps is 0.0086, which in this example is lower than the trimmed average MSE between real neurons (0.13, based on 99% of samples). The position activity maps have similar light and dark spots. Figure 3a(iii) shows the normalized position activity map for real neuron 3, and Fig. 3a(iv) shows its virtual counterpart. The MSE between the two maps is 0.21, higher than the average MSE. In this example, the position activity maps exhibit similar overall features but differ slightly in the exact location of the peaks. Figure 3b shows a summary position histogram for all real–virtual neuron pairs. The histogram is left-skewed, around the mean of 0.13. Of 69 neurons, 61 (88%) had an error less than the trimmed average. Supplementary Fig. 5 (Supplementary Fig. 6)

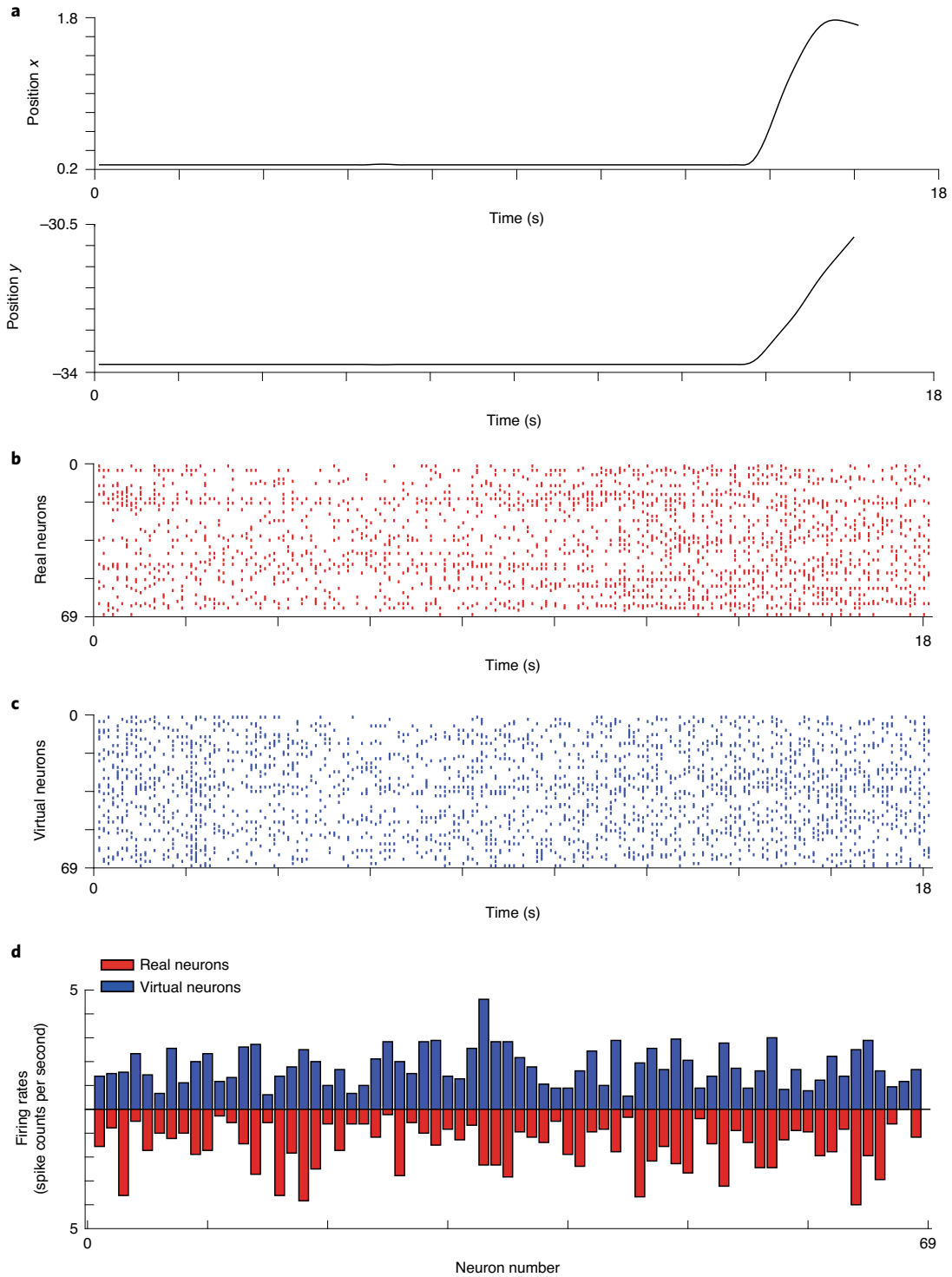


Fig. 4 | Synthesized spike trains generated from specific kinematics. a, Movement kinematics (position). **b**, Real neural data for the kinematics in **a**. **c**, Synthesized spike trains from the spike synthesizer for the kinematics in **a**. **d**, Firing rates (distribution of firing rates across all neurons) for the kinematics in **a** for real and virtual neurons.

shows a summary velocity (acceleration) histogram for all real–virtual neuron pairs. Those histograms are left-skewed, with a trimmed mean of 0.11 (0.10). Of 69 neurons, 60 (60) had an error less than the average. This shows that, with respect to position (velocity, acceleration)-related activity, the model has learned realistic virtual neurons.

Can the spike synthesizer synthesize spikes with realistic firing patterns? We then asked whether the spike synthesizer can learn to synthesize spike trains from specific kinematics (Fig. 4a), with realistic firing rates (Fig. 4b,c). We found that the spike synthesizer produces firing rates (distribution of firing rates across all neurons) that are not different from the those of real neurons (Fig. 4d;

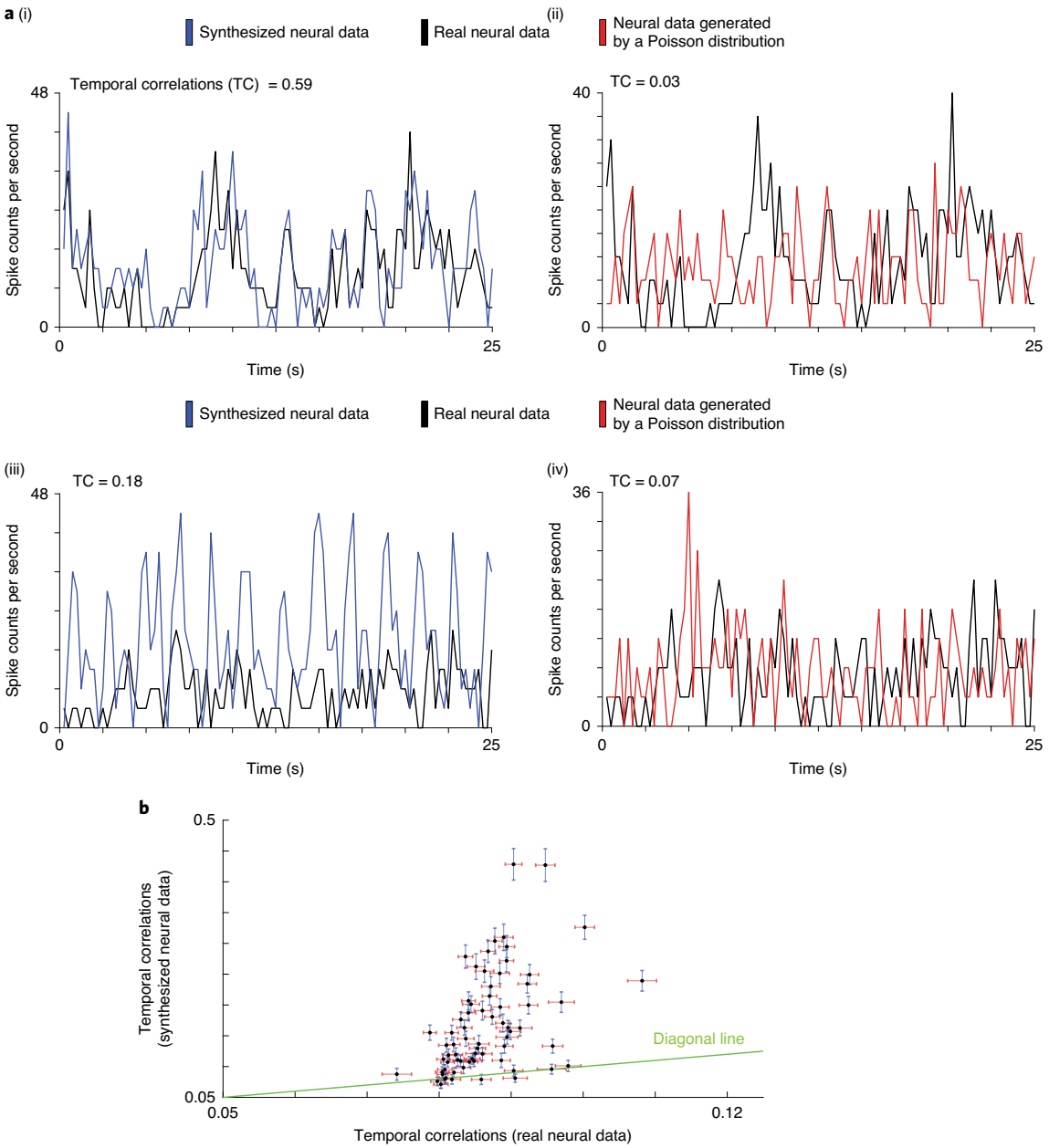


Fig. 5 | Correlation between synthesized and real spike trains. a, Time series of binned spike counts for neuron 8 (good example, left portions of Supplementary Fig. 6b). Correlation between synthesized and real neural data is 0.59 (i). By contrast, correlation between generated (red, from a homogeneous Poisson distribution) and real (black) neural data is 0.03 (ii). Time series of binned spike counts for neuron 59 (bad example, right portion of Supplementary Fig. 6b). Correlation between synthesized neural data and real neural data is 0.18 (iii), while correlation between generated neural data from Poisson distribution and real neural data is 0.07 (iv). **b**, Scatterplot of synthesized neural data versus randomly shuffled real spike trains baseline. Each black point represents a neuron. The vertical axis is the correlation between synthesized and real neural data across all neural spike train samples for each neuron, with blue standard error bar (mean ± S.E., $n = 63$). The horizontal axis is the correlation between neural data of randomly shuffled neurons across all neural spike train samples, with red standard error bar (mean ± S.E., $n = 63$).

Kolmogorov–Smirnov test: the samples are not from different distributions with $p = 0.1536$).

Then, we asked whether the synthesized spike trains are correlated with real spike trains more than would be expected by chance. To assess this, we used pairwise correlations computed after placing the spikes in 250 ms time bins[34], a time scale relevant to behavioural movements. We compared the correlation coefficients between pairs of real and synthesized spike trains to those between real spike trains and those generated by a homogenous Poisson process, an estimate of chance level (see Methods for an alternate measure that uses randomly shuffled real spike trains). The correlation coefficient is equal to one if spike trains are identical, and zero if they are independent. Figure 5a(i) shows a time series of binned spike counts for neuron 8 (an example from the left portions of Supplementary Fig. 3b), with a correlation between synthesized and real neural data of 0.59. In comparison, the correlation between the neural data from a homogeneous Poisson distribution and real neural data is 0.03 (Fig. 5a(ii)) . Figure 5a(iii) shows similar plots for neuron 59 (an example from the right portions of Supplementary Fig. 3b). Here, the correlation between actual and synthesized data is 0.18, whereas

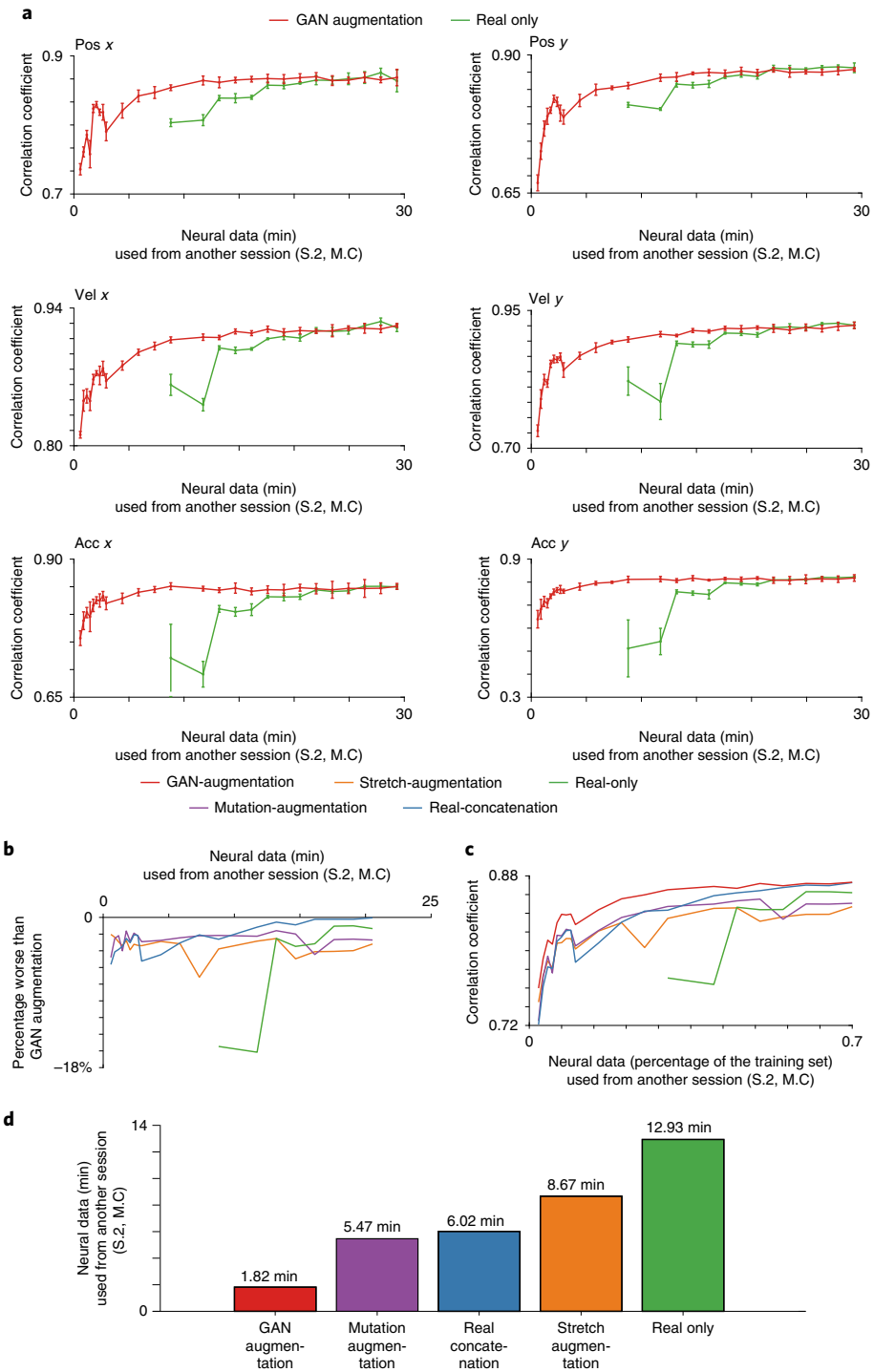


Fig. 6 | Cross-session decoding. We used six methods: GAN augmentation (red), mutation augmentation (purple), stretch augmentation (orange), real augmentation (blue) and real only (green). **a**, The performances for each kinematic in 5-fold cross-validation. The horizontal axis is the number of minutes of neural data from session 2 of Monkey C. The vertical axis is the correlation coefficient (mean ± s.d., $n = 5$ folds) between the decoded kinematics and real kinematics on an independent test set from session 2 of Monkey C. Synthesized spike trains that capture the neural attributes accelerate the training of a BCI decoder for the cross-session decoding. **b**, Average percentage performances worse than that of the GAN-augmentation method. The horizontal axis is the number of minutes of neural data from session 2 of Monkey C. The vertical axis is the average performance for all six kinematic signals of each method worse than the average percentage performance of the GAN-augmentation method. **c**, Average performances presented similar to ROC curves. The horizontal axis is the percentage of neural data from the training set of session 2 of Monkey C. The vertical axis is the same as in **a**. The cut-off time for **b** and **c** is 20.53 min—the minimum amount of time needed for other methods to achieve a performance comparable to that of GAN augmentation. **d**, Amount of additional neural data (S.2, M.C) needed for each method to achieve accuracy saturation (≥95% of the peak for the real-only method training on all neural data from S.2, M.C).

that between real neural and homogeneous Poisson data is 0.07 (Fig. 5a(iv)). In summary, for 91% (63 out of 69) of the neurons (Fig. 5b), the correlations of the actual spike trains with synthesized trains were higher than the correlations between neural spikes of randomly shuffled neurons.

In summary, the spike synthesizer learned to generate spike trains that are better correlated with real spike trains than one would expect by chance, although some differences are apparent between real and synthesized data. Therefore, we then assessed whether the synthesized spike trains are sufficiently realistic to effectively assist the training of a BCI decoder, which is the primary goal of this study.

Synthesized neural spikes can accelerate the training and improve the generalization of cross-session and cross-subject decoding. To test the utility of our spike synthesizer, we explored whether it can accelerate the training and improve the generalization of a BCI decoder across sessions and subjects. We trained the spike synthesizer from the data of the first session of Monkey C (S.1, M.C; Fig. 2, Step 1). Then we split new neural data from a second session, either from the same subject (session 2 of Monkey C; S.2, M.C) or from a different subject (first session of Monkey M; S.1, M.M), into training and test sets. We froze the generator of the spike synthesizer to preserve the embedded neural attributes. Then we fine-tuned the readout module of the spike synthesizer with as little as 35 s of the new training set to learn its dynamics. After training the BCI decoder with various combinations of real and synthesized data, we tested it on an independent test set from another session (S.2, M.C) or subject (S.1, M.M) and compared the decoding performance to other data augmentation methods (Supplementary Information). These included either using real neural data alone (real only) to train the BCI decoder, or three other data augmentation methods: real augmentation (duplicate and concatenate available real neural data), stretch augmentation (duplicate, randomly stretch and concatenate available real neural data) and mutation augmentation (duplicate, add noise and concatenate available real neural data).

Synthesized spike trains accelerate the training of a BCI decoder when the neural data from another session or subject is limited. We first tested whether synthesized spike trains can accelerate training for cross-session decoding. When using less than 17 min of new data from S.2, M.C to train the BCI decoder, our GAN augmentation method performed better than the other augmentation methods (Extended Data Fig. 2) and the real-only method (Fig. 6a). Without any data augmentation, the BCI decoder needed at least 8.5 min of training data to converge. However, using our spike synthesizer makes it possible to train the BCI decoder with substantially fewer real data. At the extreme, with only 35 s of new neural data (S.2, M.C), augmented by 22 min of synthesized data, cross-session decoding performance was better than for competing methods (used to extend the original 35 s of new data to 22 min). The averaged best performance for all kinematics (the lowest point for each curve in Fig. 6b) of the GAN-augmentation method was 7.2%, 4.8%, 5.6% and 16% better than the stretch-augmentation, mutation-augmentation, real-concatenation and real-only methods, respectively. Figure 6c shows general performance results for all kinematics variables over the percentage of new neural data used, demonstrated with a format similar to receiver operating characteristic (ROC) curves. The area under the GAN-augmentation curve was 0.030, 0.020, 0.015 and 0.33 greater than for the stretch-augmentation, mutation-augmentation, real-concatenation and real-only methods, respectively. To achieve accuracy saturation (Fig. 6d, ≥95% of the peak for the real-only method of training on all neural data from S.2, M.C), GAN augmentation only requires 1.82 min additional neural data from S.2, M.C, compared with 5.47, 6.02, 8.67 and 12.93 min (4.27×, 4.70×, 6.77× and 10.10×) for mutation augmentation, real concatenation, stretch augmentation and real only, respectively. The real-only method requires 8.8 min of additional neural data to converge, and achieves 0.77 average correlation coefficient across kinematics. In comparison, GAN augmentation requires only 0.67 min of additional neural data to achieve the same performance. In summary, our GAN augmentation is the best among all tested methods for cross-session BCI training.

Synthesized spike trains can also accelerate the training for cross-subject decoding. When using less than 9 min of new data from S.1, M.M to train the BCI decoder, our GAN-augmentation method performed better than the other augmentation methods (Extended Data Fig. 3) and the real-only method (Fig. 7a). Without any data augmentation, the BCI decoder needed at least 8.5 min of training data to converge. Figure 7b shows the general performance results for all kinematics variables over a wide range of new neural data. The averaged best performance for all kinematics (the lowest point for each curve in Fig. 7b) of the GAN-augmentation method is 60%, 17%, 11% and 6% better than the stretch-augmentation, mutation-augmentation, real-concatenation and real-only methods. Figure 7c shows general performance results for all kinematics variables over the percentage of new neural data used, demonstrated with a similar format as ROC curves. The area under the GAN-augmentation curve was 0.11, 0.052, 0.039 and 0.58 greater than the stretch-augmentation, mutation-augmentation, real-concatenation and real-only methods, respectively. None of the tested methods achieved performance comparable to GAN augmentation, since the GAN-augmentation method can transfer learned dynamics and improve the decoding performance beyond the best performance achievable on data from Monkey M alone. (Further explained in the following section). To achieve accuracy saturation, GAN augmentation only needs 0.1 min of additional neural data from S.1, M.M, compared with 1.25, 2.65, 4.02 and 8.67 min (12.50×, 26.50×, 40.20× and 86.70×) for real concatenation, mutation augmentation, stretch augmentation and real only, respectively. Thus, our GAN augmentation is again the best among all tested methods for accelerating cross-subject BCI training.

Transferring learned dynamics and improving beyond the best-achievable cross-subject decoding. Training a spike synthesizer that learns good neural attributes from Monkey C (with its better decoding performance) might transfer useful information to boost cross-subject decoding performance beyond the best performance achievable on data from Monkey M alone. When neural data are ample for both Monkey C (Fig. 6a) and Monkey M (Fig. 7a), the decoding performance on position and velocity are equally good for both monkeys, although the performance on acceleration of Monkey C is better than that of Monkey M (0.85 versus 0.43 for acc x; 0.82 versus 0.48 for acc y). The better decoding performance on acceleration data from Monkey C might come from the better quality of signals collected by the electrode arrays, or from the fact that neural data from Monkey C are inherently easier for the decoder than those from Monkey M. A lower quality of the collected neural data from Monkey M might decrease the decoding accuracy of accelerations because accelerations are usually the hardest to decode, but this could be improved through prior learning from monkey C. Since decoding performance on position and velocity for monkey M are already good, we wondered whether any useful information learned from neural data of Monkey C could improve the best achievable decoding performance on acceleration for Monkey M.

Here we compared the best achievable performance of all methods (the highest points for acceleration x and acceleration y curves in Fig. 7a and Extended Data Fig. 3). As a result, for acceleration y, the best achievable performance for GAN augmentation was 0.64, significantly better than the best achievable performance for real only (0.48; $P = 10^{-14}$, t-test), stretch augmentation (0.62; $P = 10^{-3}$,

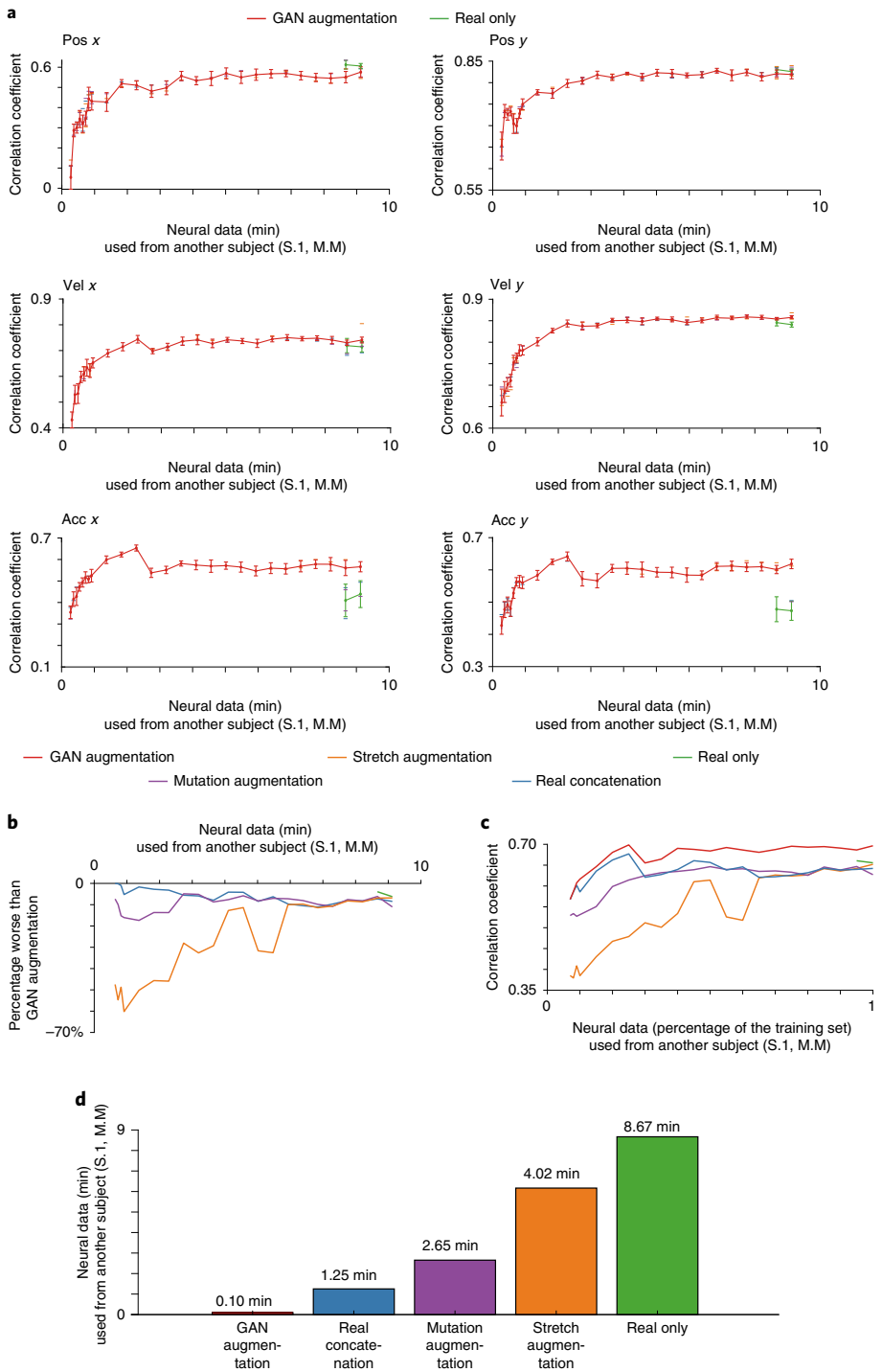


Fig. 7 | Cross-subject decoding. Methods and their notations are the same as in Fig. 6. **a**, Performances for each kinematic in 5-fold cross-validation. The horizontal axis is the number of minutes of neural data from Monkey M. The vertical axis is the correlation coefficient (mean ± S.D., $n = 5$ folds) between the decoded kinematics and real kinematics on an independent test set from Monkey M. When the neural data from another subject is limited, synthesized spike trains that capture the neural attributes accelerate the cross-subject decoding performance. In addition, synthesized spike trains have learned generalizable information that can boost cross-subject decoding performance on acceleration over the best achievable performance. **b**, Average percentage performances worse than that of the GAN-augmentation method. The horizontal axis is the number of minutes of neural data from session 1 of Monkey M. The vertical axis is the same as in Fig. 6b. **c**, Average performances presented similar to ROC curves. The horizontal axis is the percentage of neural data from the training set of session 1 of Monkey M. The vertical axis is the same as in **a**. **d**, Amount of additional neural data (S.1, M.M) needed for each method to achieve accuracy saturation (≥95% of the peak for the real-only method training on all neural data from S.1, M.M).

t-test), mutation augmentation (0.51; $P = 10^{-11}$, *t*-test) and real concatenation (0.59; $P = 10^{-4}$, *t*-test). The results for acceleration *x* are similar to that of acceleration *y*. Thus, even with ample neural data for both monkeys, the neural attributes learned from the first monkey can transfer some useful information to improve the best achievable decoding performance for the second monkey.

Discussion

The concept of this Article is that building a spike synthesizer that captures underlying neural attributes can accelerate the training and improve the generalization of BCI decoders. We have introduced a new spike synthesizer that can learn a mapping from kinematics to neural data. The spike synthesizer captures and reproduces the neural attributes in its training set. After the fine-tuning procedure, we showed that the synthesizer adapted to data from a new session or even a new subject, and can synthesize new spike trains. The synthesized spike trains can accelerate the training of a BCI decoder and improve its generalization across sessions and subjects.

We could interpret the improvement in the decoding performance from the perspective of statistics of movements. The understanding of the statistics of movements can help us in many situations where obtaining simultaneous recordings of both neural activity and kinematics is difficult. It has been shown[35] that distribution-alignment decoding (DAD) can leverage the statistics of movements to achieve semi-supervised decoder training. The authors built prior distributions of motor-output variables during a simple planar reaching task. DAD was then able to align the distributions of decoder outputs to kinematic distribution priors. However, it is not clear whether the approach would work in more complex movements, as one would need many prior distributions of motor-output variables (high-level templates). The complex movements might involve sub-movements such as holding still, rapid reaching and slow reaching. It might take a substantial amount of time to craft these templates manually, and it is hard to choose the right form of the templates for a task and to combine them properly. In comparison, our spike synthesizer recreated neural attributes, such as position, velocity and acceleration activity maps, and velocity neural tuning curves (low-level templates) directly from the data. For more complex kinematics, our spike synthesizer might synthesize spike trains drawn from different distributions for holding and rapid reaching by combining those neural attributes in an autonomous way. There is no need to handcraft high-level templates (prior distributions of motor-output variables) if they can be constructed from more fundamental low-level ones.

The neural attributes captured by the spike synthesizer (position, velocity and acceleration activity maps, and velocity neural tuning curves; Supplementary Figs. 1, 2, 5 and 6) have a special name in the literature[36–41]: motor primitives. Motor primitives are a generalization of movement components or the templates (kinematic distribution priors) used by DAD. It has been suggested that the motor cortex may control movement through flexible combinations of motor primitives, elementary building blocks that can be combined to give rise to complex motor behaviour. One study[40] defined a motor primitive as the directional neural tuning curve for each neuron, fitted by a Gaussian function. The authors built movement trajectories through linear combinations of those tuning curves. Related research[41] used gain patterns over neurons to predict movement trajectories. Another study[42] described the learning of a diverse set of motor primitives through a latent representation of a neural abstraction network from kinematics given in a demonstration. Then the authors split new complex kinematics into subparts and fit those subparts with the motor primitives. The key problem in motor-primitive research is how to learn the primitives, and then how to combine them to create complex motion. Here, our results suggest that our spike synthesizer learned motor primitives through its internal representation (similar to the work in ref. [42]) from both kinematics and neural data (in contrast to the work in ref. [42], where only kinematics was used) and their combination rules in an autonomous and principled way. In addition, from an extended version[40] of the definition of a motor primitive that includes both position and velocity tuning for each neuron, we can reproduce these motor primitives by reconstructing position and velocity tuning curves for each neuron from the internal representation.

A recent method, LFADS (latent factor analysis via dynamical systems)[19], can infer latent dynamics from single-trial neural spike data by leveraging a deep learning auto-encoder. Our method is complementary to this work in the following ways. First, LFADS focuses on how to construct a mapping from the neural data to low-dimensional latent variables (neural population dynamics), while simultaneously reconstructing the same neural data from a Poisson process parameterized by these low-dimensional latent variables. In contrast, our method uses a generative adversarial network to create a mapping directly from the kinematics to the neural data in an end-to-end manner. In addition, the mapping does not impose any prior distribution on the data.

Recent advances in the analysis of neural population activity, especially with the help of BCI methods, have revealed the importance of the covariance structure of neuronal populations in controlling movements and motor skill acquisition[43–45]. Although in our work the spike synthesizer was trained to learn a mapping between kinematic and neural data internally, the spike synthesizer maps the kinematics to a generalizable internal representation that captures embedded neural attributes, including the high-order statistics between neurons such as covariance. We expect that this covariance structure imposed by our generalizable internal representation between neurons can accelerate training for BCI decoders that even aim to generalize across tasks. Finally, our framework of learning the internal representation from raw data and additional synthesized data is general and fully data-driven, in contrast to neural encoding models[46–49] that assume Gaussian or Poisson distributions. Hence, our framework could be applied to other neuroscience encoding and decoding problems beyond motor control, with minimal too domain-specific modifications.

Methods

The BCI decoder. We used the state-of-the-art LSTM network[11,12] as the decoder. Recurrent neural networks can use their feedback connections to store a representation of recent input in the hidden states. However, with the traditional backpropagation through time to update the hidden states, they often suffer either a gradient-exploding or vanishing problem. LSTM creates an uninterrupted gradient flow and thus have a better performance. The structure of an LSTM cell can be formalized as

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \quad (1)$$

$$c_t = f \odot c_{t-1} + i \odot g \quad (2)$$

$$h_t = o \odot tanh(c_t) \quad (3)$$

where x_t is the input at time t and h_t is the hidden dimension at time t; W is the weight vector from h_{t-1} and x_t to gates; σ is the sigmoid function; i is the input gate, deciding whether to write to the cell; f is the forget gate, deciding whether to erase the cell; g is the gate gate, deciding how much to write to the cell; o is the output gate, deciding how much to reveal to the cell; and c_t is the middle variable. In the LSTM decoder case, we unroll our LSTM cell and consider 200 timesteps for each sample. The input dimension is (N, T, D), where N is the number of samples, T is the number of timesteps and D is the feature dimensions. Our input is batched neural spikes where the feature dimensions are 128 samples, 200 timesteps and the number of neurons is 69 for session 1, 77 for session 2 of Monkey C and 60 for session 1 of Monkey M. The hidden dimensions h is 200 for the LSTM decoder. So, we have an output dimension (N 128, T 200, H 200) from the LSTM decoder. We fed this output into a fully connected layer to produce the kinematics (dimension [128, 200, 6]). We applied dropout techniques[50] and learning rate decay[51] while training the LSTM decoder.

Bidirectional LSTM[52]. The output of a sequence at a current time slot relies not only on the sequences before it, but also depends on the sequences after it. So, to better capture the neural attributes, we used the bidirectional LSTM to build the generator and discriminator in our constrained conditional LSTM GAN (cc-LSTM-GAN) model. At each timestep t, this network has two hidden states,

one for left-to-right propagation and another for the right-to-left propagation. The update rule is

$$\overrightarrow{h_t} = g\left(\overrightarrow{W}x_t + \overrightarrow{V}\overrightarrow{h_{t-1}} + \overrightarrow{b}\right) \quad (4)$$

$$\overleftarrow{h_t} = g\left(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h_{t+1}} + \overrightarrow{b}\right) \quad (5)$$

where $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ maintain the left-to-right and right-to-left hidden states separately at time t; and g is the LSTM cell update function in equations (1), (2) and (3).

GAN. GANs[24] provide a tool to learn a map from a random noise input to the desired data distribution in an end-to-end way, updating its parameters via backpropagation[53]. Thus, it does not require any assumption about the data distribution. It is purely data-driven and does not need a strong prior model, which would limit generality. The process of training a GAN can be thought of as an adversarial game between a generator and a discriminator. The role of the generator is to produce fake currency and use it without detection, while the discriminator learns to detect the counterfeit currency. Competition in this adversarial game can improve both components' abilities until the counterfeits are indistinguishable from real currency. After this competition, the generator can take random noise (that provides variations) as input to output different kinds of realistic bills with different textures. Several approaches have been proposed for image synthesis using GANs, enhanced to be able to generate output images for a particular object class, such as conditional GAN[24], semi-supervised GAN[26], infoGAN[27], AC-GAN[25] and cGANs[28]. In this fake currency scenario, by injecting the conditions (denomination of each bill) into the input of GAN, we can select which kind of bills we want to generate (for example, a 100 dollar bill), while the noise provides the varied texture of the bills (for example, smooth or wrinkled).

cc-LSTM-GAN, the spike synthesizer. We propose the constrained conditional LSTM GAN to model the behaviour of M1 from the kinematics (Extended Data Fig. 4). A normal LSTM model takes input which has a dimension of (N, T, D), where N is the number of samples in one minibatch, T is the time horizon and D is the hidden dimension size. We chose 2 s as time horizon in our experiments; the first input dimension N is the number of batches; and the third input dimension D is the number of neurons for the discriminator or noise dimension for the generator. For each item in the batch, we had a 2 s slice of neural spikes from D neurons. Since the number of neurons is the third hidden dimension of the LSTM in the discriminator, our discriminator treated different neurons as individuals that have different neural tuning properties. Thus, the cc-LSTM-GAN encoding model is a multiple neural encoding model.

Training assistant LSTM decoder (GAN-ta LSTM decoder). We trained an LSTM decoder (hidden dimension h: 200) on neural data ((N,T,M), where N is the sample size (128), T is the time horizon (200) and M is the number of neurons (69 for S.1, M.C; 77 for S.2, M.C; 60 for S.1, M.M)) from one monkey and froze its parameters when we trained our bidirectional cc-LSTM GAN. This decoder applied constraints to the cc-LSTM-GAN. We wanted to maintain the decoding performance while we trained the spike synthesizer.

Bidirectional LSTM generator. The bidirectional LSTM generator takes Gaussian noise ((N, T, D), where N is the sample size (128), T is the time horizon (200), D is the dimension for Gaussian noise (6)) and real kinematics ((N, T, D), where N is the sample size (128), T is the time horizon (200), D is the dimension for kinematics (6)) as inputs and synthesizes the corresponding spike trains. We fed the outputs (dimensions (N, T, 2*H), where N is the sample size (128), T is the time horizon (200), H is the hidden dimension (200)) of the bidirectional LSTM into a fully connected layer to synthesize spike trains with the correct number of neurons (dimensions (N,T,M), where N is the sample size (128), T is the time horizon (200), M is the number of neurons (69 for S.1, M.C; 77 for S.2, M.C; 60 for S.1, M.M)). We applied $\frac{1}{2} \times tanh$ function[54] as the output layer of the fully connected layer, which maps a real value to (−0.5, 0.5) that gives us a probability representation of whether the current bin contains a spike event or not. For example, if the value in the current bin is −0.3, the probability that there is a spike event in this bin is 0.2 (−0.3 + 0.5). For analysis, we sampled the Bernoulli distribution to generate the neural firing patterns, while for decoding, the LSTM decoder directly took the probability as an input.

Bidirectional LSTM discriminator. The discriminator is a bidirectional LSTM. It takes the synthesized spike trains ((N,T,M), where N is the sample size (128), T is the time horizon (200), M is the number of neurons (69 for S.1, M.C; 77 for S.2, M.C; 60 for S.1, M.M)) and neural data ((N,T,M), N, T and M have the same meaning and dimension as in the synthesized spike trains) as inputs and learns to determine whether a sample is from the neuron data or synthesized spike trains. We fed the outputs (processed by a sigmoid activation) of the bidirectional LSTM into a fully connected layer to obtain a decision value (0,1), a probability that decides whether the current sample is real or fake. We fed the output of a bidirectional LSTM into another fully connected layer to get the decoded kinematics. This helped us to apply the category constraints.

Multiple neural encoding model (cc-LSTM-GAN). We used a conditional structure that has a GAN category loss to let the discriminator distinguish both data source distribution (neural spikes) and data label distribution (kinematics corresponding to these spikes). When the input of the bidirectional LSTM discriminator is the neural data (synthesized spikes trains), the real (fake) embedding features are the output of the bidirectional LSTM discriminator. GAN embedding category loss is the L2 loss between the real embedding features and the fake embedding features.

$$\begin{aligned}&\text{GAN embedding category loss}\\&= (\text{Real embedding features} - \text{Fake embedding features})^2\end{aligned} \quad (6)$$

When the input of the bidirectional LSTM discriminator is the neural data (synthesized spike trains), the real (fake) decoded kinematics are the output of the fully connected layer after the bidirectional LSTM discriminator. Real (fake) GAN decoding category loss is the L2 loss between real (fake) decoded kinematics and real kinematics. The GAN category loss is the average of real GAN decoding category loss, fake GAN decoding category loss and GAN embedding category loss.

$$\begin{aligned}\text{GAN category loss} = \frac{1}{3} \times (&\text{Real GAN decoding category loss} +\\ &\text{Fake GAN decoding category loss} +\\ &\text{GAN embedding category loss})\end{aligned} \quad (7)$$

To maintain the source distribution, our cc-LSTM-GAN needs to play the minimum maximum game. We need to minimize discriminator GAN loss (L_D) and generator GAN loss (L_G). The L_D and L_G are calculated as

$$L_D = -E_{x\sim p_{\text{data}}}[\log D(x)] - E_{z\sim p(z)}[\log(1 - D(G(z,k)))] \quad (8)$$

$$L_G = -E_{z\sim p(z)}[\log(D(G(z,k)))] \quad (9)$$

where z is the Gaussian noise, x is the neuron spikes, k is the kinematics, $p(z)$ is the noise distribution, p_{data} is the data distribution, L_D is the discriminator loss calculated by cross entropy loss function and L_G is the generator loss calculated by cross entropy loss function.

To further maintain the statistical structure of the real neurons, we wanted to minimize the negative of the inner product loss between the neural data and synthesized spike trains. This is achieved using the hyperparameter α, which adjusts the scale of inner product loss. We use 0.00001 in our implementation. Thus, we calculated our inner product loss as

$$\text{Inner product loss} = -\alpha \times \text{synthesized spike trains} \cdot \text{neural data} \quad (10)$$

The pre-trained GAN-ta LSTM decoder takes the neural data and synthesized spike trains as input and decodes the corresponding kinematics. The decoded generated kinematics are kinematics decoded from synthesized spike trains. The decoded real kinematics are kinematics decoded from neural data. We applied L2 loss between real kinematics and decoded generated kinematics. We also applied L2 loss between decoded generated kinematics and decoded real kinematics. The pre-trained GAN-ta LSTM decoder helped our generator synthesize more realistic spike trains in terms of the performance of the GAN-ta LSTM decoder.

The total generator loss is the weighted average of the generator GAN loss (L_G), the GAN category loss, the inner product loss, the L2 loss between decoded generated kinematics and decoded real kinematics and the L2 loss between decoded generated kinematics and real kinematics.

$$\begin{aligned}\text{Total generator loss} = &0.7 \times L_G + 0.2 \times \text{GAN category loss} + 0.1 \times \text{inner product loss} +\\ &0.1 \times \text{L2 Loss between decoded generated kinematics and}\\ &\text{decoded real kinematics} + 0.1 \times \text{L2 Loss between decoded}\\ &\text{generated kinematics and real kinematics}\end{aligned} \quad (11)$$

The total discriminator loss is the weighted average of the discriminator GAN loss (L_D) and the GAN category loss. We trained this network with real GAN training set (a subset of the neural data from the first monkey) and minimized the total discriminator and generator losses.

$$\text{Total discriminator loss} = 0.8 \times L_D + 0.2 \times \text{GAN category loss} \quad (12)$$

Fine-tuning. We trained cc-LSTM-GAN on the session 1 data of Monkey C. In the fine-tuning process, we took a limited amount of neural data from session 2 of Monkey C (cross-session) or from Monkey M (cross-subject). We added another

fully connected layer (readout module) on top of the bidirectional LSTM generator and used it to generate the corresponding neural spikes with the same number of neurons as the added data. We froze the parameters of hidden units from the bidirectional LSTM generator and only trained this new fully connected layer with limited new data. The loss function of the fine-tuning process is the inner product loss in equation (10). Then we fed the kinematics corresponding to this limited amount of neural data into the generator multiple times to synthesize a large amount of spike trains.

Correlation. To calculate the correlations across neural spike train samples for each neuron, we parsed the synthesized and real neural spikes into 250 ms bins and set the time horizon to 25 s. We calculated the correlations for each sample and averaged the absolute value of correlations for each neuron. We compared the result against that of randomly shuffled real spike trains (compared the real neural data with shuffled neural data from other neurons) and that of the neural spikes generated from a homogeneous Poisson distribution. The homogeneous Poisson distribution assumes that the generation of each spike depends only on a homogeneous firing rate and is independent of all the other spikes. Thus, it is a good baseline for comparison.

Data augmentation. We used multiple data augmentation methods to train a BCI decoder and achieve a better decoding performance than training a BCI decoder on the neural data only. The start point of limited neural data is always selected from the start of the neural data in each fold. The end point of the limited neural data is the start point plus the length of the required amount of limited data.

Real-only method. Without any data augmentation, we trained the BCI decoder on the neural data only and tested on the neural data of an independent test set. This method required at least 8.5 min neural data in the training set to let the BCI decoder converge (The criterion for convergence is that the training loss would decrease slowly during training. If we have less than 8.5 min training data, the training loss does not decrease no matter how long we train the LSTM BCI decoder). If the BCI decoder fails to converge, the correlation coefficient is defined as correlation between randomly shuffled real kinematics and real kinematics (0.00027 chance level)

Real concatenation. We took a limited amount of neural data from the training set and concatenated it multiple times until it had equal or longer length than the whole training set. We trained the BCI decoder on the concatenated neural data and tested on the neural data of an independent test set.

Mutation augmentation. We flipped the value of the limited neural data with 5% probability (spike to non-spike or non-spike to spike). We repeated this step several times and concatenated the mutated neural data and its kinematics until they had equal or longer length than the whole training set. We trained the BCI decoder on the mutated neural data and tested on the neural data of an independent test set.

Stretch augmentation. We stretched the limited neural data by 10% and filled the empty stretched slots of the neural data by zero or one (50% probability). We calculated the averaged absolute gradients for each kinematic signal for each slot. We filled the empty stretched slots of the kinematics by adding or subtracting (50% probability) the value of the last slot with the averaged absolute gradients. We repeated the above steps several times and concatenated the stretched neural data and its kinematics together until it had equal or longer length than the whole training set. We trained the BCI decoder on the stretched neural data and tested on the neural data of an independent test set.

GAN augmentation. We combined the synthesized spike trains (22 min in our Article) with the augmented data from the real-concatenation method. We then trained the BCI decoder on the combination of synthesized spike trains and concatenated neural data, and tested on neural data of an independent test set.

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

Data availability

The raw and analysed datasets generated during the study are available on Github at https://github.com/shixianwen/Rapid-transfer-of-brain-machine-interfaces-to-new-neuronal-ensembles-or-participants.

Code availability

The codes used in this study are available on Github at https://github.com/shixianwen/Rapid-transfer-of-brain-machine-interfaces-to-new-neuronal-ensembles-or-participants.

Received: 13 June 2020; Accepted: 17 September 2021;
Published online: 18 November 2021

References

1. Wessberg, J. et al. Real-time prediction of hand trajectory by ensembles of cortical neurons in primates. *Nature* https://doi.org/10.1038/35042582 (2000).
2. Velliste, M., Perel, S., Spalding, M. C., Whitford, A. S. & Schwartz, A. B. Cortical control of a prosthetic arm for self-feeding. *Nature* https://doi.org/10.1038/nature06996 (2008).
3. Taylor, D. M., Tillery, S. I. H. & Schwartz, A. B. Direct cortical control of 3D neuroprosthetic devices. *Science* https://doi.org/10.1126/science.1070291 (2002).
4. Carmena, J. M. et al. Learning to control a brain–machine interface for reaching and grasping by primates. *PLoS Biol*. https://doi.org/10.1371/journal.pbio.0000042 (2003).
5. Li, Z. et al. Unscented Kalman filter for brain–machine interfaces. *PLoS ONE* https://doi.org/10.1371/journal.pone.0006243 (2009).
6. Wu, W. et al. Neural decoding of cursor motion using a Kalman filter. In *Proc. Advances in Neural Information Processing Systems 15* (eds Becker, S., Thrun, S. & Obermayer, K.) 133-140 (MIT Press, 2003).
7. Brockwell, A. E. Recursive Bayesian decoding of motor cortical signals by particle filtering. *J. Neurophysiol*. https://doi.org/10.1152/jn.00438.2003 (2004).
8. Gao, Y., Black, M. J., Bienenstock, E., Wu, W. & Donoghue, J. P. A quantitative comparison of linear and non-linear models of motor cortical activity for the encoding and decoding of arm motions. In *First International IEEE EMBS Conference on Neural Engineering* (IEEE, 2003); https://doi.org/10.1109/CNE.2003.1196789
9. Eden, U. T., Frank, L. M., Barbieri, R., Solo, V. & Brown, E. N. Dynamic analysis of neural encoding by point process adaptive filtering. *Neural Comput*. https://doi.org/10.1162/089976604773135069 (2004).
10. Eden, U. T. Point process adaptive filters for neural data analysis: theory and applications. In *Proc. IEEE Conference on Decision and Control* (IEEE, 2007); https://doi.org/10.1109/CDC.2007.4434708
11. Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural Comput*. https://doi.org/10.1162/neco.1997.9.8.1735 (1997).
12. Glaser, J. I. et al. Machine learning for neural decoding. *Eneuro* https://doi.org/10.1523/ENEURO.0506-19.2020 (2020).
13. Moeendarbary, E. et al. The soft mechanical signature of glial scars in the central nervous system. *Nat. Commun*. https://doi.org/10.1038/ncomms14787 (2017).
14. Kozai, T. D. Y., Jaquins-Gerstl, A. S., Vazquez, A. L., Michael, A. C. & Cui, X. T. Brain tissue responses to neural implants impact signal sensitivity and intervention strategies. *ACS Chem. Neurosci*. https://doi.org/10.1021/cn500256e (2015).
15. Duffau, H. Brain plasticity and reorganization before, during, and after glioma resection. *Glioblastoma* https://doi.org/10.1016/B978-0-323-47660-7.00018-5 (2016).
16. Tkach, D., Reimer, J. & Hatsopoulos, N. G. Observation-based learning for brain–machine interfaces. *Curr. Opin. Neurobiol.* **18**, 589–594 (2008).
17. Gallego, J. A., Perich, M. G., Chowdhury, R. H., Solla, S. A. & Miller, L. E. Long-term stability of cortical population dynamics underlying consistent behavior. *Nat. Neurosci.* **23**, 260–270 (2020).
18. Gao, P. et al. A theory of multineuronal dimensionality, dynamics and measurement. Preprint at *bioRxiv* https://doi.org/10.1101/214262 (2017).
19. Pandarinath, C. et al. Inferring single-trial neural population dynamics using sequential auto-encoders. *Nat. Methods* https://doi.org/10.1038/s41592-018-0109-9 (2018).
20. Farshchian, A. et al. Adversarial domain adaptation for stable brain-machine interfaces. *Seventh International Conference on Learning Representations* Paper 1437 (ICLR, 2019).
21. 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*. https://doi.org/10.1038/ncomms13749 (2016).
22. Degenhart, A. D. et al. Stabilization of a brain–computer interface via the alignment of low-dimensional spaces of neural activity. *Nat. Biomed. Eng.* **4**, 672–685 (2020).
23. Gerstner, W., Kistler, W. M., Naud, R. & Paninski, L. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition* (Cambridge Univ. Press, 2014); https://doi.org/10.1017/CBO9781107447615
24. Goodfellow, I. et al. Generative adversarial nets. in *Advances in Neural Information Processing Systems 27* (2014); https://papers.nips.cc/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html
25. Wei, L., Hu, L., Kim, V., Yumer, E. & Li, H. Real-time hair rendering using sequential adversarial networks. In *Computer Vision – ECCV 2018. ECCV 2018. Lecture Notes in Computer Science* Vol. 11208 (eds Ferrari V., Hebert M., Sminchisescu C. & Weiss Y.) 105–122 (Springer, 2018); https://doi.org/10.1007/978-3-030-01225-0_7
26. Jetchev, N. & Bergmann, U. The conditional analogy GAN: swapping fashion articles on people images. In *2017 IEEE International Conference on Computer Vision Workshops, ICCVW 2017* (IEEE, 2018); https://doi.org/10.1109/ICCVW.2017.269

27. Chen, Xi. et al. InfoGAN: interpretable representation learning by information maximizing generative adversarial nets. Preprint at https://arxiv.org/abs/1606.03657 (2016).
28. Mirza, M. & Osindero, S Conditional generative adversarial nets. Preprint at https://arxiv.org/abs/1411.1784 (2014).
29. Odena, A, Olah, C. & Shlens, J. Conditional image synthesis with auxiliary classifier GANs. *Proc. 34th International Conference on Machine Learning* (eds Precup, D. & Teh, Y. W.) 2642–2651 (PMLR, 2017).
30. Ho, D., Liang, D., Stoica, I., Abbeel, P. & Chen, X. Population based augmentation: efficient learning of augmentation policy schedules. *Proc. 36th International Conference on Machine Learning* (eds Chaudhuri, K. & Salakhutdinov, R.) 2731–2741 (PMLR, 2019).
31. Dai, W., Yang, Q., Xue, G.-R. & Yu, Y. Boosting for transfer learning. In *Proceedings of the 24th International Conference on Machine learning - ICML '07* (ACM, 2007); https://doi.org/10.1145/1273496.1273521
32. Arnold, A., Nallapati, R. & Cohen, W. W. A comparative study of methods for transductive transfer learning. In *Proceedings - IEEE International Conference on Data Mining, ICDM* (IEEE, 2007); https://doi.org/10.1109/ICDMW.2007.109
33. Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." arXiv preprint arXiv:1312.4400 (2013).
34. Tchumatchenko, T., Geisel, T., Volgushev, M. & Wolf, F. Spike correlations - what can they tell about synchrony? *Front. Neurosci.* https://doi.org/10.3389/fnins.2011.00068 (2011).
35. Dyer, E. L. et al. A cryptography-based approach for movement decoding. *Nat. Biomed. Eng.* https://doi.org/10.1038/s41551-017-0169-7 (2017).
36. Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P. & Schaal, S. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural Comput.* https://doi.org/10.1162/NECO_a_00393 (2013).
37. Schaal, S. "Dynamic movement primitives-a framework for motor control in humans and humanoid robotics." Adaptive motion of animals and machines. Springer, Tokyo, 2006. 261–280.
38. Poggio, T. & Bizzi, E. Generalization in vision and motor control. *Nature* https://doi.org/10.1038/nature03014 (2004).
39. Nuyujukian, P. et al. Performance sustaining intracortical neural prostheses. *J. Neural Eng.* https://doi.org/10.1088/1741-2560/11/6/066003 (2014).
40. Thoroughman, K. A. & Shadmehr, R. Learning of action through adaptive combination of motor primitives. *Nature* https://doi.org/10.1038/35037588 (2000).
41. Stroud, J. P., Porter, M. A., Hennequin, G. & Vogels, T. P. Motor primitives in space and time via targeted gain modulation in cortical networks. *Nat. Neurosci.* https://doi.org/10.1038/s41593-018-0276-0 (2018).
42. Shankar, T., Pinto, L., Tulsiani, S. & Gupta, A. Discovering motor programs by recomposing demonstrations. Conference paper 1246 in *International Conference on Learning Representations* (ICLR, 2020).
43. Costa, R. M., Ganguly, K., Costa, R. M. & Carmena, J. M. Emergence of coordinated neural dynamics underlies neuroprosthetic learning and skillful control. *Neuron* https://doi.org/10.1016/j.neuron.2017.01.016 (2017).
44. Golub, M. D. et al. Learning by neural reassociation. *Nat. Neurosci.* https://doi.org/10.1038/s41593-018-0095-3 (2018).
45. Sadtler, P. T. et al. Neural constraints on learning. *Nature* https://doi.org/10.1038/nature13665 (2014).
46. Gold, J. I. & Shadlen, M. N. The neural basis of decision making. *Annu. Rev. Neurosci.* https://doi.org/10.1146/annurev.neuro.29.051605.113038 (2007).
47. Felsen, G. & Dan, Y. A natural approach to studying vision. *Nat. Neurosci.* https://doi.org/10.1038/nn1608 (2005).
48. Paninski, L., Pillow, J. & Lewi, J. Statistical models for neural encoding, decoding, and optimal stimulus design. *Prog. Brain Res.* https://doi.org/10.1016/S0079-6123(06)65031-0 (2007).
49. Paninski, L. Superlinear population encoding of dynamic hand trajectory in primary motor cortex. *J. Neurosci.* https://doi.org/10.1523/JNEUROSCI.0919-04.2004 (2004).
50. 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).
51. Senior, A. et al. An empirical study of learning rates in deep neural networks for speech recognition. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. 6724–6728 (IEEE, 2013).
52. Schuster, M. & Paliwal, K. K. Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.* **45**, 2673–2681 (1997).
53. le Cun, Y. A theoretical framework for back-propagation. In *Proc. 1988 Connectionist Models Summer School* Vol. 1. (eds Touretzky, D., Hinton, G. & Sejnowski, T.) 21–28 (CMU, 1988).
54. LeCun, Y. A., Bottou, L., Orr, G. B. & Müller, K. R. In *Neural Networks: Tricks of the Trade* 2nd edn (eds Montavon, G., Orr, G. B. & Müller, K. R.) 9–48 (Springer, 2012); https://doi.org/10.1007/978-3-642-35289-8_3

Acknowledgements

This work was supported by the National Science Foundation (grant no. CCF-1317433), C-BRIC (one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA), the Intel Corporation and the National Institutes of Health (grant nos. NIH NINDS T32 HD07418, F31 NS092356, NS053603 and NS074044). We affirm that the views expressed herein are solely our own and do not represent the views of the US government or any agency thereof.

Author contributions

M.G.P. and L.E.M. conducted the experiments. S.W., A.Y., T.F. and L.I. analysed the results. All authors reviewed the manuscript.

Competing interests

The authors declare no competing interests.

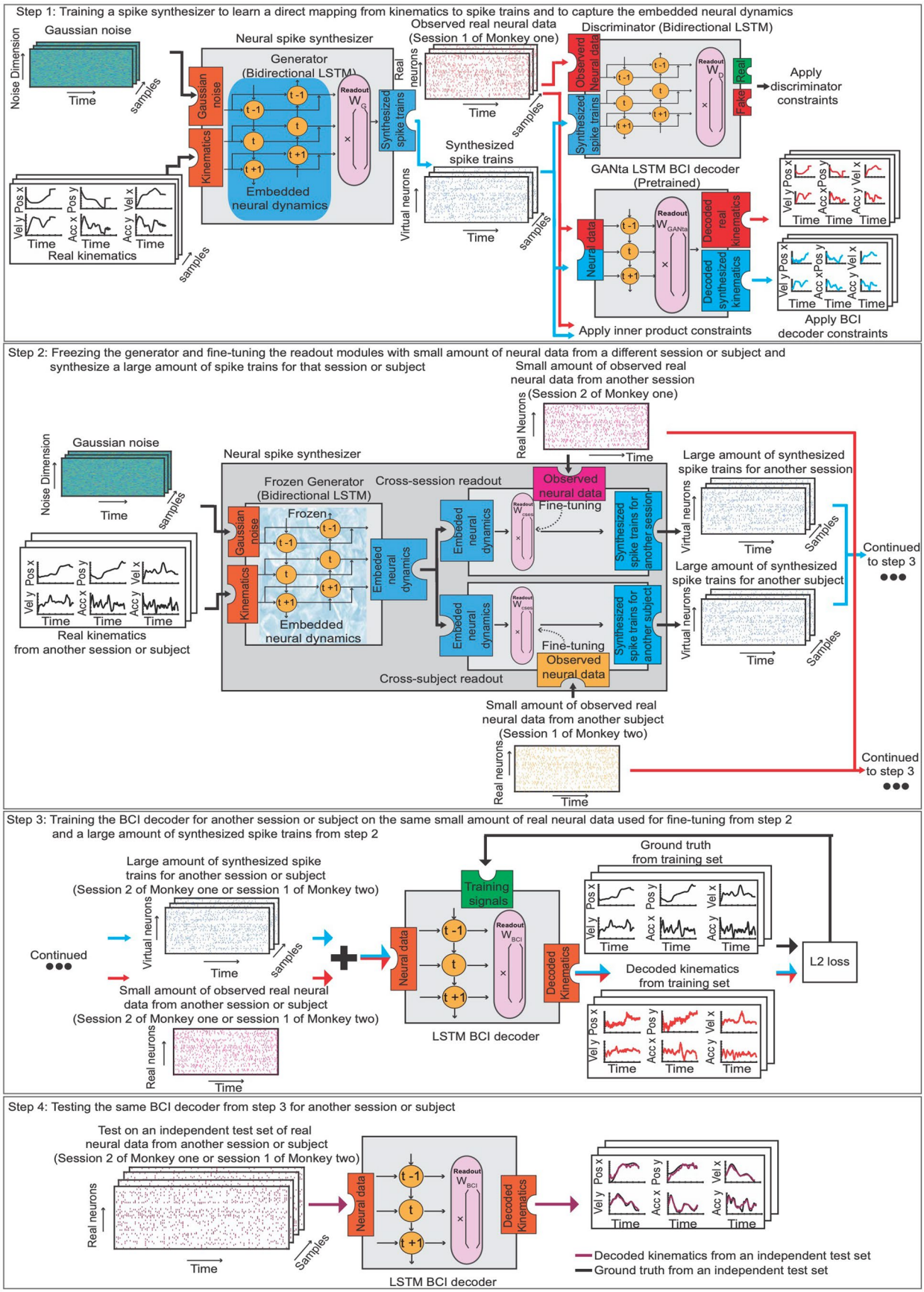
Additional information

Extended data is available for this paper at https://doi.org/10.1038/s41551-021-00811-z.

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s41551-021-00811-z.

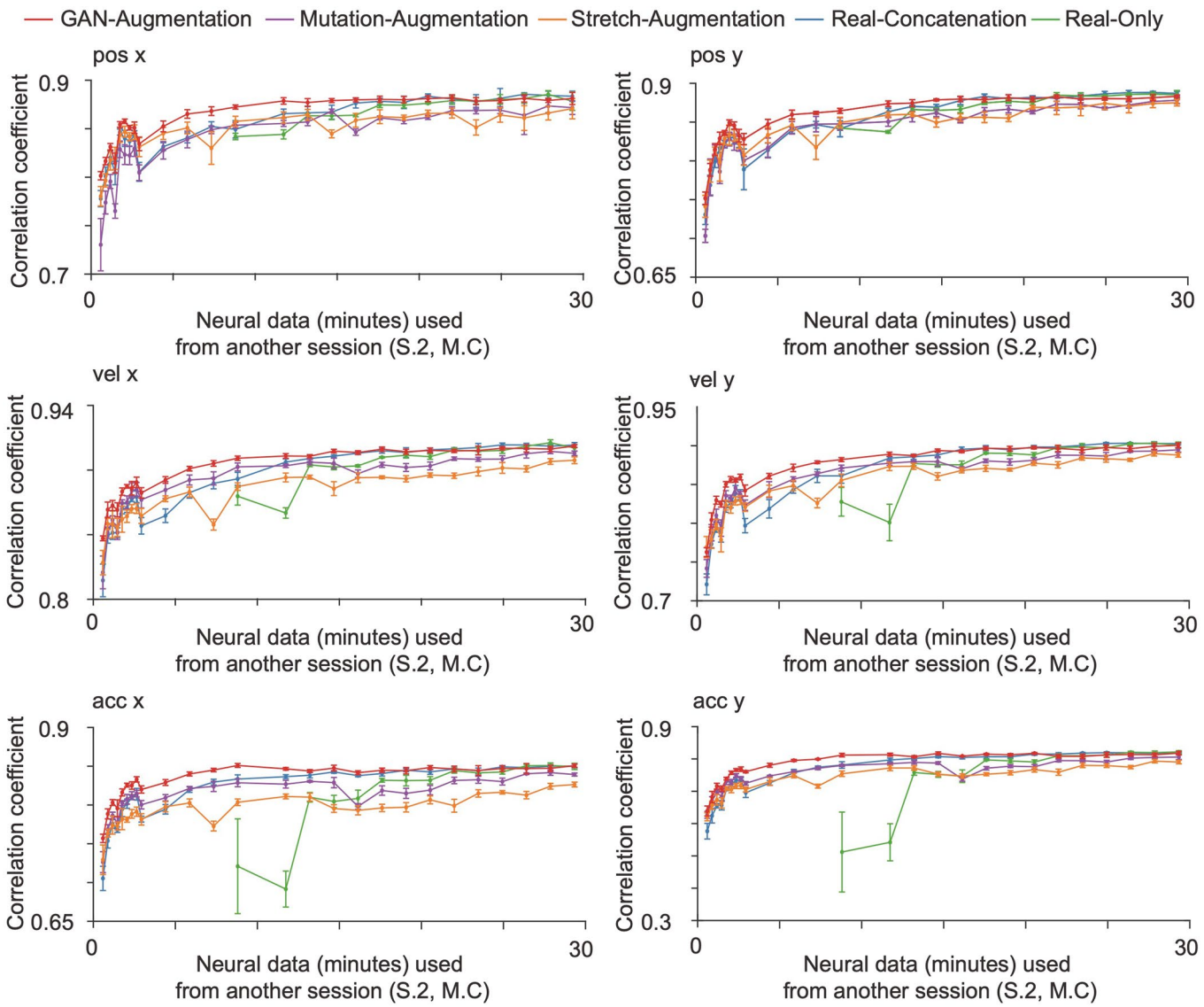
Correspondence and requests for materials should be addressed to Shixian Wen or Laurent Itti.

Peer review information *Nature Biomedical Engineering* thanks Jonathan Viventi and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.

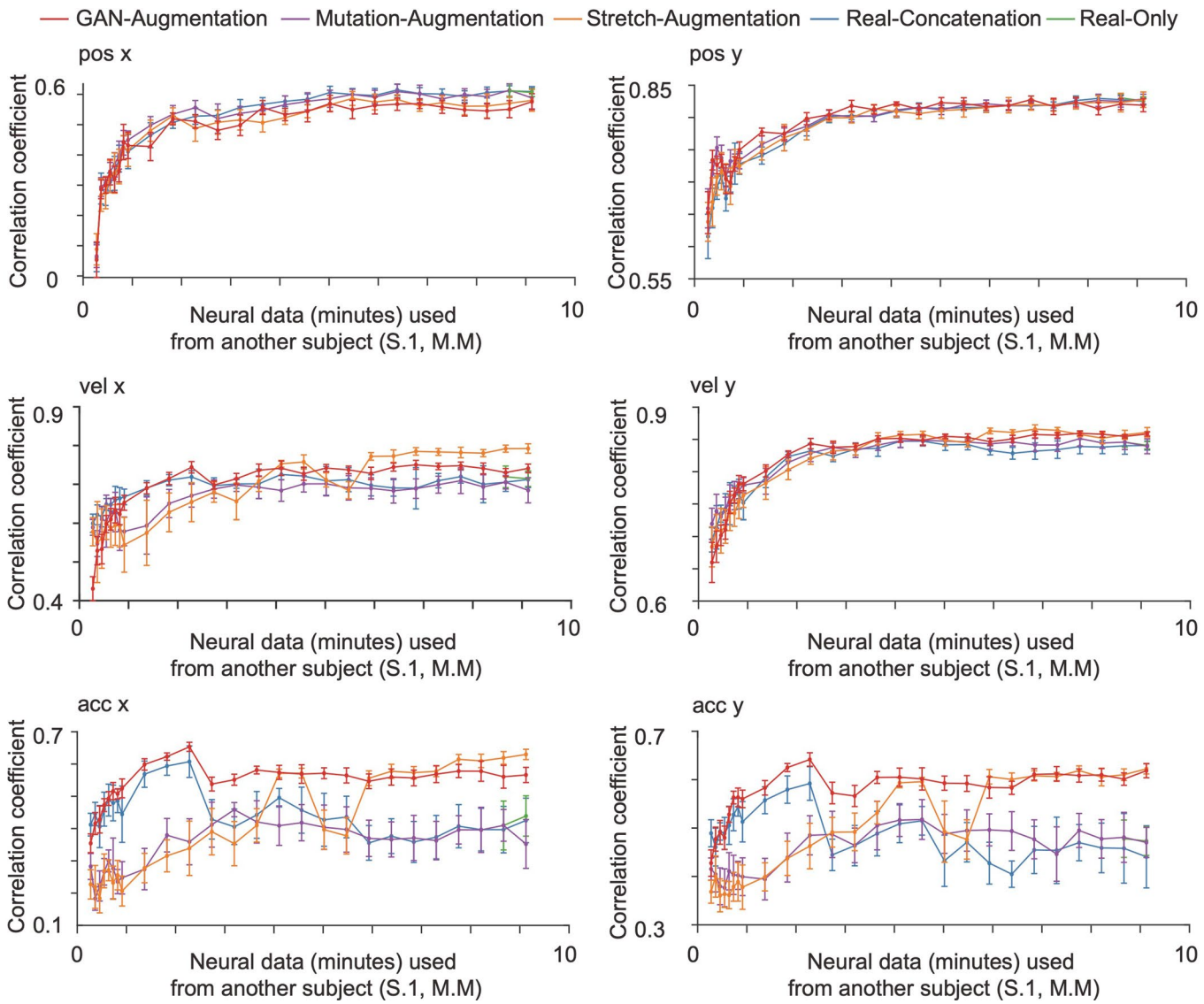


Extended Data Fig. 1 | See next page for caption.

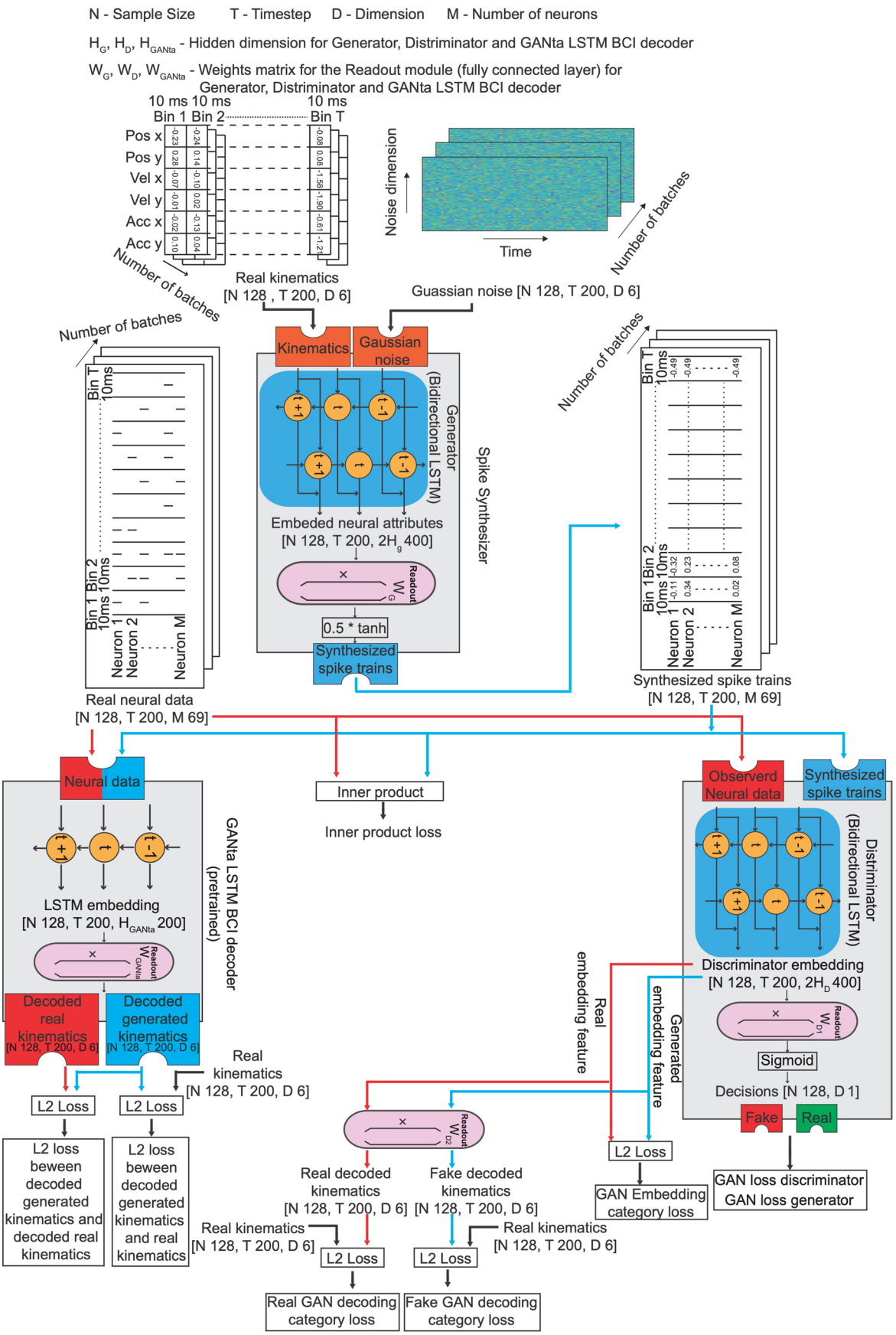
Extended Data Fig. 1 | General Framework. Step 1: training a neural spike synthesizer on the neural data from session one of Monkey one to learn a direct mapping from kinematics to spike trains and to capture the embedded neural attributes. Step 2: freezing the generator that captures the embedded neural attributes and fine-tuning the readout modules for different sessions or subjects to allow variations in neural attributes, using the neural data from session two of Monkey one or the neural data from session one of Monkey two. Then, synthesizing a large amount of spike trains that are suitable for another session or subject. Step 3: training a BCI decoder for another session or subject using the same small amount of real neural data used for fine-tuning (in step 2) and a large amount of synthesized spike trains (in step 2). Step 4: testing the same BCI decoder on an independent test set from another session or subject.



Extended Data Fig. 2 | Cross-session decoding. The GAN-augmentation, mutation-augmentation, stretch-augmentation, real-concatenation and real-only methods are shown in red, purple, orange, blue and green curves with an error bar in 5-fold cross-validation. The horizontal axis is the number of minutes of neural data from the session two of Monkey C used. The vertical axis is correlation coefficient between the decoded kinematics and real kinematics on an independent test set from the session two of Monkey C (mean +/ - S.D., n = 5 folds). Synthesized spike trains that capture the neural attributes accelerate the training of a BCI decoder for the cross-session decoding.



Extended Data Fig. 3 | Cross-subject decoding. The GAN-augmentation, mutation-augmentation, stretch-augmentation, real-concatenation and real-only methods are shown in red, purple, orange, blue and green curves with an error bar in 5-fold cross-validation. The horizontal axis is the number of minutes of neural data from Monkey M used. The vertical axis is the correlation coefficient between the decoded kinematics and real kinematics on an independent test set from the Monkey M (mean +/ - S.D., n = 5 folds). When the neural data from another subject is limited, synthesized spike trains that capture the neural attributes improve the cross-subject decoding performance on acceleration. Even with ample neural data for both subjects, the neural attributes learned from one subject can transfer some useful knowledge that improves the best achievable decoding performance on the acceleration of another subject.



Extended Data Fig. 4 | Detailed structure of the CC-LSTM-GAN.

nature portfolio

Corresponding author(s): Shixian Wen, Laurent Itti

Last updated by author(s): Jul 21, 2021

Reporting Summary

Nature Portfolio wishes to improve the reproducibility of the work that we publish. This form provides structure for consistency and transparency in reporting. For further information on Nature Portfolio policies, see our Editorial Policies and the Editorial Policy Checklist.

Statistics

For all statistical analyses, confirm that the following items are present in the figure legend, table legend, main text, or Methods section.

n/a	Confirmed
☐	☒ The exact sample size (*n*) for each experimental group/condition, given as a discrete number and unit of measurement
☐	☒ A statement on whether measurements were taken from distinct samples or whether the same sample was measured repeatedly
☐	☒ The statistical test(s) used AND whether they are one- or two-sided *Only common tests should be described solely by name; describe more complex techniques in the Methods section.*
☐	☒ A description of all covariates tested
☐	☒ A description of any assumptions or corrections, such as tests of normality and adjustment for multiple comparisons
☐	☒ A full description of the statistical parameters including central tendency (e.g. means) or other basic estimates (e.g. regression coefficient) AND variation (e.g. standard deviation) or associated estimates of uncertainty (e.g. confidence intervals)
☐	☒ For null hypothesis testing, the test statistic (e.g. *F*, *t*, *r*) with confidence intervals, effect sizes, degrees of freedom and *P* value noted *Give P values as exact values whenever suitable.*
☒	☐ For Bayesian analysis, information on the choice of priors and Markov chain Monte Carlo settings
☒	☐ For hierarchical and complex designs, identification of the appropriate level for tests and full reporting of outcomes
☐	☒ Estimates of effect sizes (e.g. Cohen's *d*, Pearson's *r*), indicating how they were calculated

Our web collection on statistics for biologists contains articles on many of the points above.

Software and code

Policy information about availability of computer code

Data collection	All neural and behavioural data were collected using a Cerebus system from Blackrock Microsystems (Salt Lake City, UT) and processed with custom software in Matlab (The Mathworks Inc.).
Data analysis	We used Tensorflow 1.13.1 and Matlab 2019a to process the data. The codes used in this study are available in Github at https://github.com/shixianwen/Rapid-transfer-of-brain-machine-interfaces-to-new-neuronal-ensembles-or-participants.

For manuscripts utilizing custom algorithms or software that are central to the research but not yet described in published literature, software must be made available to editors and reviewers. We strongly encourage code deposition in a community repository (e.g. GitHub). See the Nature Portfolio guidelines for submitting code & software for further information.

Data

Policy information about availability of data

All manuscripts must include a data availability statement. This statement should provide the following information, where applicable:

- Accession codes, unique identifiers, or web links for publicly available datasets
- A description of any restrictions on data availability
- For clinical datasets or third party data, please ensure that the statement adheres to our policy

The raw and analysed datasets generated during the study are available in Github at https://github.com/shixianwen/Rapid-transfer-of-brain-machine-interfaces-to-new-neuronal-ensembles-or-participants.

Field-specific reporting

Please select the one below that is the best fit for your research. If you are not sure, read the appropriate sections before making your selection.

☒ Life sciences ☐ Behavioural & social sciences ☐ Ecological, evolutionary & environmental sciences

For a reference copy of the document with all sections, see nature.com/documents/nr-reporting-summary-flat.pdf

Life sciences study design

All studies must disclose on these points even when the disclosure is negative.

Sample size	We used two subjects in our study. The most common number of subjects in a monkey-based study is two. In fact, in essentially computational studies such as this one, it is not unusual (although not ideal) to consider data from a single monkey. Thus, when setting up the experiment (which is challenging and costly, as it involves surgery and long recovery and training times), adding a third monkey would typically add up to a year of work. Monkeys are also valuable resources and these experiments are invasive and risky for the animals, so the number to be used must be well justified in terms of expected benefits. For this reason, usually two animals are considered adequate.
Data exclusions	No data were excluded.
Replication	We ran the codes multiple times with different seeds, and verified that the results can be replicated.
Randomization	The method was tested on the neural data collected from two monkeys. Randomization was not relevant to this study.
Blinding	Blinding was not relevant, because a blinding process wouldn't influence the sampling results.

Reporting for specific materials, systems and methods

We require information from authors about some types of materials, experimental systems and methods used in many studies. Here, indicate whether each material, system or method listed is relevant to your study. If you are not sure if a list item applies to your research, read the appropriate section before selecting a response.

Materials & experimental systems

n/a	Involved in the study
☒	☐ Antibodies
☒	☐ Eukaryotic cell lines
☒	☐ Palaeontology and archaeology
☐	☒ Animals and other organisms
☒	☐ Human research participants
☒	☐ Clinical data
☒	☐ Dual use research of concern

Methods

n/a	Involved in the study
☒	☐ ChIP-seq
☒	☐ Flow cytometry
☒	☐ MRI-based neuroimaging

Animals and other organisms

Policy information about studies involving animals; ARRIVE guidelines recommended for reporting animal research

Laboratory animals	This study used two monkeys (male, macaca mulatta). At the time of data collection, Monkey C was 6–8 years old and Monkey M was 6–7 years old.
Wild animals	The study did not involve wild animals.
Field-collected samples	The study did not involve samples collected from the field.
Ethics oversight	All surgical and experimental procedures were approved by the Institutional Animal Care and Use Committee (IACUC) of Northwestern University.

Note that full information on the approval of the study protocol must also be provided in the manuscript.