Balancing memorization and generalization in RNNs for high performance brain-machine Interfaces

Supplemental Methods and Results

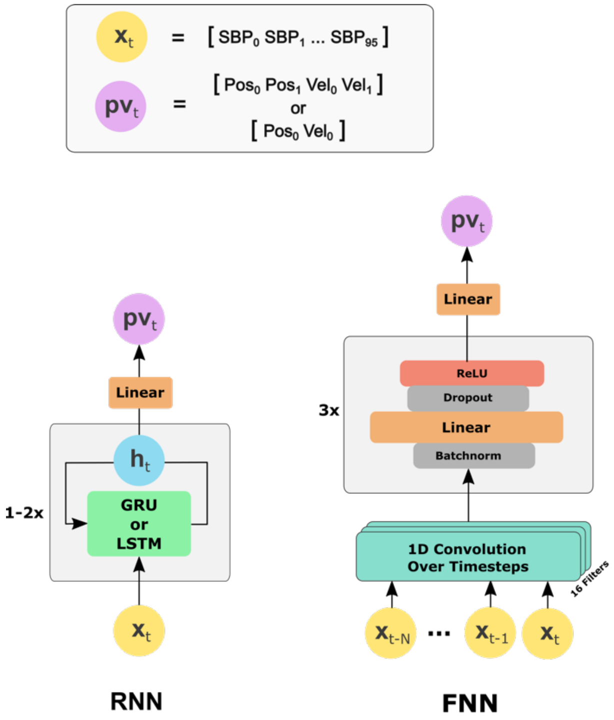
Supplemental Video Captions

Supplemental Video 1: Online decoder comparison. Example trials from Monkey N using an LSTM, FNN, TFM, and KF to control the virtual hand. All decoders were trained using 400 trials of random targets, and then tested online during the same experimental session.
Video corresponds to main text Results 3.1.

Supplemental Video 2: LSTM movement memorization. An LSTM decoder was trained on a reduced target dataset, and then tested online on the same set of targets (Monkey N). Example 1: 2 degrees-of-freedom with 4 target postures. Example 2: 1 degree-of-freedom with 7 target postures. In both cases, the decoder reaches the same performance as able-bodied control, with minimal overshooting or movement error.
Video corresponds to main text Results 3.2.

Supplemental Video 2: Performance recovery through memorization. First, we show Monkey N using an LSTM trained on random targets, which has poor online control of the index finger. Second, we train an LSTM on a modified task with random targets for MRS fingers but only 3 targets for index finger. This allows for memorization of the index finger movements while maintaining fully continuous control of MRS fingers, resulting in an improved success rate. Video corresponds to main text Results 3.3.

Neural Network Architectures



Supplemental Figure 1: Neural network architectures. Each decoder takes in binned neural features (RNN – the most recent bin, FNN & TFM – the most recent five bins) and predicts the current position and velocity of one or two finger groups. The FNN is a time-convolutional network introduced by Willsey et al. 2022 which learns convolutional features over the neural inputs for each channel and uses several feedforward layers. The TFM is a feedforward transformer (Vaswani 2017) network using positional encoding, multi-head attention, and standard feedforward layers.

Decoder Hyperparameters

To optimize decoder hyperparameters, we used the Optuna Python library to perform Bayesian optimization. For each monkey, optimization was independently performed for two datasets (from 2-3 months apart) and the final chosen hyperparameters minimized the MSE of position and velocity predictions for both days. All decoders were generally robust to the specific parameter choice (a large range of parameters achieved high offline performance). In the table below, "scheduler patience" corresponds to the learning rate scheduler and sets the number of training steps without loss improvement before reducing the learning rate.

	Monkey N				**Monkey W**			
	LSTM	**GRU**	**FNN**	**TFM**	**LSTM**	**GRU**	**FNN**	**TFM**
Number of Input Neural Time Bins	1	1	5	5	1	1	5	5
Convolutional Features	n/a	n/a	16	n/a	n/a	n/a	16	n/a
Num Layers	1	2	3	4 layers, 8 heads	1	1	3	5 layers, 8 heads
Hidden Size	300	250	400, 400, 100	1000	250	250	300, 100, 300	1400
Dropout	n/a	0.5	0.5	0.1	n/a	n/a	0.06	0.14
Learning Rate	2.00E-04	2.00E-04	2.00E-04	2.00E-04	2.00E-04	4.00E-04	8.00E-04	1.00E-04
Weight Decay	0.003	0.004	0.03	0.002	0.003	0.003	0.01	1.00E-04
Scheduler Patience	800	800	1000	800	800	800	600	800

Supplemental Table 1: Decoder hyperparameters for each monkey.

Decoder Number of Parameters

	LSTM	**GRU**	**FNN**	**TFM**	**KF**
Number of Parameters	479K	639K	821K	923K	9.7K

Supplemental Table 1: The number of parameters used for each decoding architecture, as optimized for Monkey N.

Decoder Training Optimizations

As noted in the main results (“Training Optimizations”), we found several techniques for improving online performance and are detailed here.

Modified loss function:
See section below.

Single-finger movements:
The 2-DoF random target task typically involves both fingers simultaneously moving to their respective targets. However, the task lacks examples of one finger holding still while the other moves to the target, which could be useful as decoder training examples. To provide more training examples of independent movement, we modified the task such that on 50% of trials only one finger had to move. When trained on this modified task, the resulting decoders had qualitatively more independent control of each finger.

Positional perturbations:
When using a BMI decoder, the user often has to make small correctional movements near the target in order to precisely land on the target. When training on able-bodied movement, however, movements are typically very precise and lack examples of these fine-tuning movements. To encourage more adjustment movements in the training dataset, we added perturbations to the position of the virtual fingers during offline training. Perturbations occurred on 50% of trials with a magnitude slightly larger than the target radius and a randomly chosen direction for each finger group. On perturbation trials, the perturbation was applied at 100 ms (near movement onset), 300 ms (during movement), or 1000 ms (during the hold period) from trial start, chosen with equal probability.

Neural Noise:
Willet et al. 2021 found that adding a small amount of noise to the neural data during training improved robustness to small shifts in neural tuning and magnitude over time. Here, we similarly added a random bias (std. of 0.1, normalized units; held constant across the multiple timesteps of each example) and random noise (std. of 0.2, normalized units) to each training example. Noise was drawn from a zero-mean normal distribution.

Loss Function to Penalize Finger Co-Dependence

In initial online tests with RNN decoders we found that the fingers tended to move together and were visually correlated (despite strong offline accuracy). To encourage finger independence, we used the standard MSE loss with an additional term to penalize finger correlations:

$$loss = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + k\sum_{j=1}^{M}\sum_{\substack{m=1, \\ m\neq j}}^{M}|\rho_{j,m}|$$

where y contains the true positions and velocities (matrix of shape [number of samples, number of fingers*2]), $\hat{y}$ contains the predicted position and velocities, N is the number of samples, M is the number of output variables (number of fingers*2), k is a hyperparameter to control the degree of correlation penalty, and $\rho_{j,m}$ is the correlation between the $j-th$ and $m-th$ output variables of $\hat{y}$. Qualitatively, for offline predictions, adding the correlation penalty acts to smooth out small-amplitude, high-frequency prediction noise that occurs across both fingers. The final decoder predictions were robust across a range of k values (0.01 to 100), so a value of 1 was used for decoder training.

RNN Hidden State Visualization

In Figure 4 we visualized the hidden states of an RNN decoder over time. To train the decoder, we trained a GRU (1 layer, 300 hidden units) on a simulated dataset with 100 neural channels, 1000 seconds of data, and 2 degrees of freedom where one DoF had 3 targets and the second DoF had random targets. We used a 1-layer GRU since its hidden state is a simple vector, whereas multi-layer GRUs or LSTMs have multiple hidden state vectors. To visualize the hidden state, we performed a principal component analysis (PCA) on the hidden state vector over time and plotted the first 3 components with highest variance. As in Figure 4, each point represents the hidden state at one time step, and the component axes with highest variance visually correspond with the decoder’s position output.

Simulated Datasets

For some analyses we also created simulated offline datasets of a virtual user performing the same target acquisition task. The goal of these simulations was to test the relative impact of amount of training data, number of DoFs, and number of inputs on decoder performance, rather than measuring absolute performance. The simulated user moved with a velocity proportional to the distance to the target along each DoF, with a random reaction time of 32-96 ms at trial onset. We generated artificial neural activity such that each channel had a random relationship with position, velocity, and acceleration, using the log-linear approximation suggested in Truccollo et al. 2008:

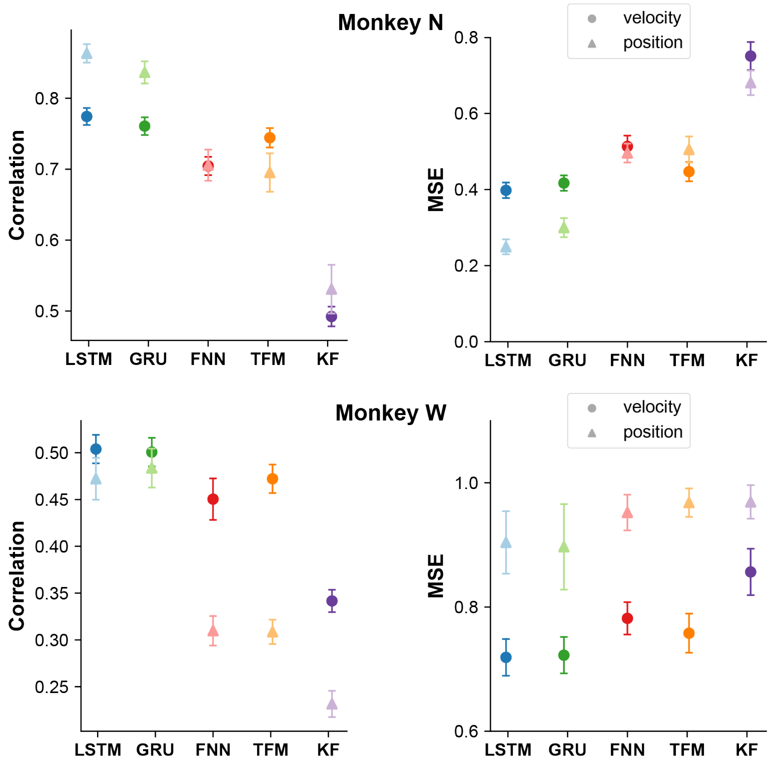
$$\log(Y_{avg}) = k * [\underline{1}, X, \frac{dX}{dt}, \frac{d^2X}{dt^2}] * W$$

where Y_{avg} is a vector of average neural channel values, X is a vector of the current positions, W is a matrix of uniform random values between -1 to 1 and defines the random tuning of each channel, and k is scaling constant adjusting the level of nonlinearity. At each time bin, the value of each channel was sampled from a normal distribution:

$$Y_t \sim Normal(Y_{avg}, diag(Y_{avg}/S))$$

where Y_t is a vector of values for each neural channel, and S is a constant that scales the noise standard deviation. Each channel has independent noise from other channels. We chose a value of $S = 10$ such that the resulting LSTM decoding had similar accuracy to Monkey N (correlation of ~0.8 at 100 channels). An additional lag of 32 ms was added to the final positions and velocities relative to the artificial neural data. To account for random variation in simulated channel tuning, we ran 5 simulations for each analysis with a separate decoder trained on each dataset.

Offline Decoder Performance for Monkey N and Monkey W



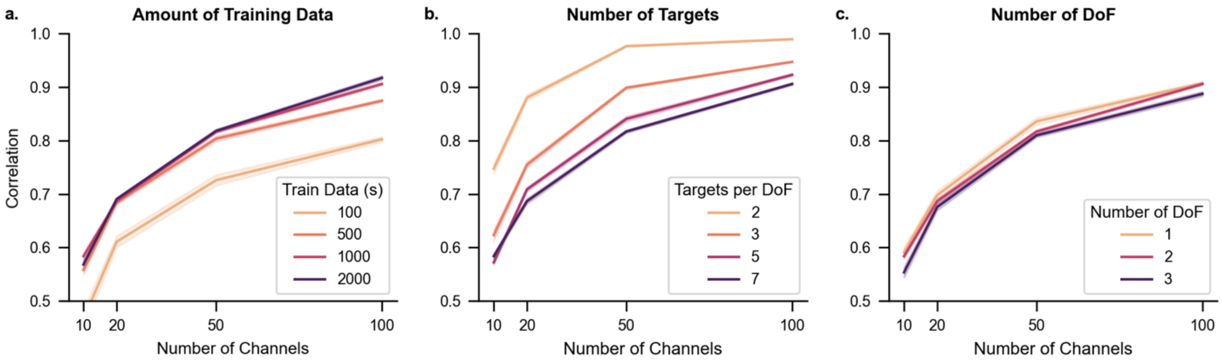
Supplemental Figure 2: Offline performance for each decoder. Top: Monkey N. Bottom: Monkey W. Correlation and MSE were calculated between the ground truth and predicted positions/velocities. Performance was averaged across ten datasets for each monkey performing a 2-DoF random task. Error bars denote the standard error of the mean.

Online Decoder Comparisons – Monkey N

Monkey	Date	DoF	LSTM		GRU		FNN		TFM		KF	
N	7/27/22	2	-	-	2.33	100%	1.77	100%	-	-	-	-
N	10/18/22	1	2.33	100%	2.16	100%	1.77	100%	1.9	100%	1.73	100%
N	10/20/22	2	3.04	99%	2.78	99%	2.77	100%	2.1	100%	1.98	89%
N	3/15/23	2	2.31	97%	-	-	1.93	99%	-	-	-	-
			Average Bitrate	Success Rate								

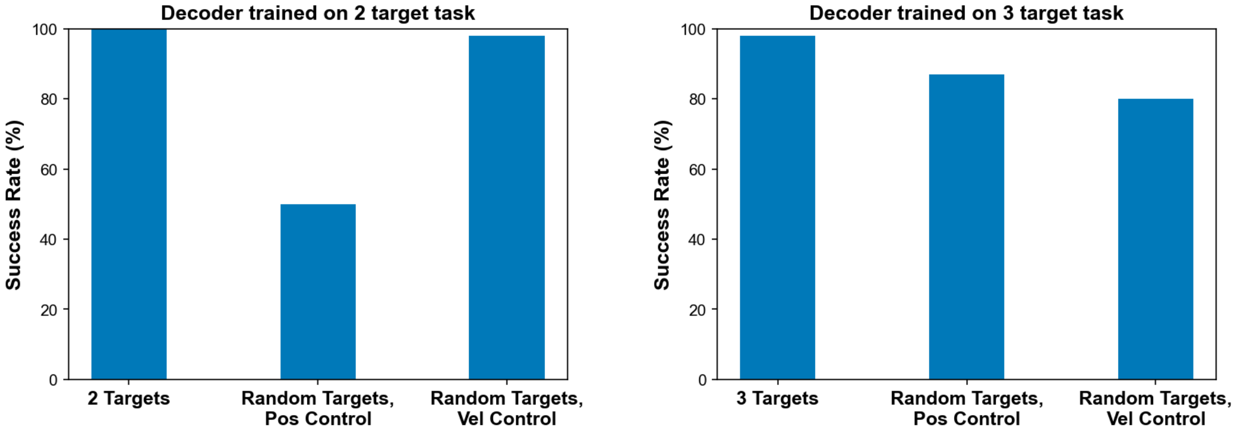
Supplemental Table 2: Online decoder performance for Monkey N. All decoders were tested on random targets.

LSTM Performance on Simulated Datasets with Varied Task Parameters



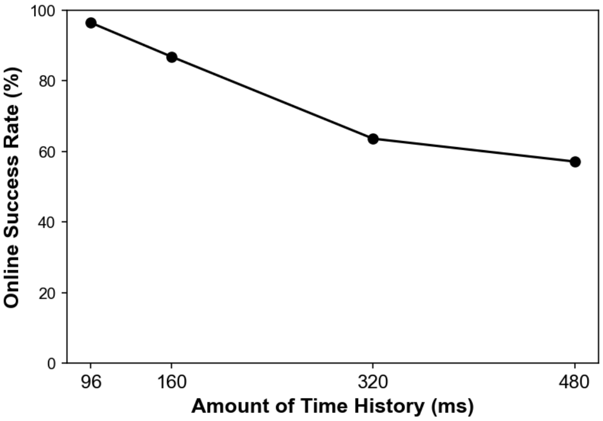
Supplemental Figure 3: Simulated decoder performance for varied task parameters. Simulated datasets were generated for each parameter combination and a GRU decoder was trained to predict positions and velocities. By default, each generated dataset used 2-DoF, 7 targets, 1000 seconds of training data, 50 ms bins, and 10 trials of each combination. In each plot we varied the number of neural channels and an additional parameter. The Pearson correlation was computed between the true and predicted kinematics, with higher values being better. (a) Varied amount of training data. Performance improves with additional training data and with additional channels. (b) Varied number of targets. Performance drops as more targets are added (requiring additional movements to be learned). (c) Varied number of DoF. There is only a small performance drop from adding more DoFs. All error bars denote the standard error of the mean.

LSTM Generalization to a More Complex Task



Supplemental Figure 4: Generalization to more complex tasks online. LSTM decoders were trained on a 1-DoF, 2-target (left) or 3-target (right) task and then tested online on the same task and the more complex random target task (see main text Results 3.4). "Pos Control" indicates the decoder used 50% position & 50% integrated velocity. "Vel Control" indicates the decoder used 1% position & 99% integrated velocity. Data corresponds to main text Results 3.4.

Transformer Performance vs Time History



Supplemental Figure 5: Transformer performance drops with more time history. Separate transformer (TFM) decoders were trained with a time history (sequence length) of 3, 5, 10, and 15 bins (96 to 480 ms using 32 ms bins), on a 2-DoF random target task with Monkey N. During online trials, success rate drops as more time history is added.