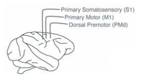
High-Fidelity RNN Approximations for Musculoskeletal Models of the Arm

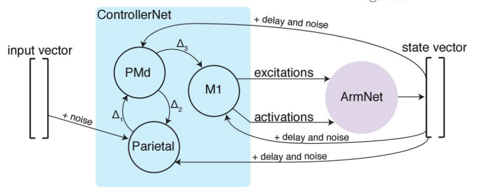
Iliana Bray[1] and Saurabh Vyas[2]
[1] Electrical Engineering, [2] Bioengineering, Stanford University, Stanford, CA
{ibray, smvyas}@stanford.edu

Motivation

The fundamental goal of our lab (advised by Prof. Krishna Shenoy) is to understand how patterns of neural activity in the motor areas of the brain give rise to movements. The specific goal of our research here is to construct a RNN (recurrent neural network) model that takes information about a particular task (e.g., the task goal of making a point-to-point reach to a target on a 2D plane) as input and generates realistic kinematics as output. We could then use neural population analysis techniques (e.g., dynamical system theory, fixed point analysis, etc.) to compare how well the activations in the RNN match real neural data collected from non-human primates from our research group [1]–[3]. We hope to build a model that will explain both the behavior of the animal and also the variance in recorded neural population data. This will then act in our research as another tool to understand how the brain performs motor control, by providing insight into the behavioral and computational control policies used in flexible feedback control and by generating predictions of how biomechanics influence behavior and neural dynamics.



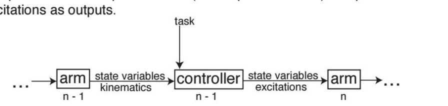
Models



Our model takes in an input vector of task goal and starting state. It outputs a state vector of 49 state variables, including kinematics. ControllerNet and ArmNet are both single layer RNNs with 1024 hidden GRUs (gated recurrent units) [4].

ArmNet is an RNN approximation to the OpenSim[5] arm model, which takes muscle excitations as inputs and produces kinematics as outputs.

ControllerNet takes in information about the task the network must accomplish and previous time-step kinematics (the output of ArmNet) and produces muscle excitations as outputs.



The models can be unrolled to produce the above computational graph. The output of ArmNet, which is a state vector including kinematics, is fed into ControllerNet in addition to task information. Then ControllerNet outputs excitations and activations which are fed into the next timestep of ArmNet.

ControllerNet can't directly produce the kinematics of the task. Therefore it must "indirectly" influence the kinematics, much like how the brain must learn to generate patterns of neural activity that will travel down the spinal cord and actuate the muscles.

Iterative Pipeline

Step 1:
Feed starting states of the arm and muscle excitations into OpenSim to produce kinematics and other output state variables
⟶ This produces a labeled dataset D = {input, output} = {excitations, state variables}

Step 2:
Train ArmNet using backpropagation through time (BPTT) in Tensorflow with a mean squared error (MSE) loss function across the full state vector
⟶ This produces "Generation 0 ArmNet"

Step 3:
Hold ArmNet constant and train ControllerNet using BPTT in Tensorflow with a loss function that compares how closely estimated kinematics match ground truth kinematics (which are trivially just a straight line)
⟶ This produces "Generation 0 ControllerNet"

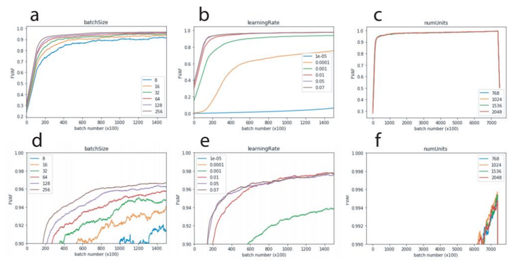
Step 4:
Generate new data using ControllerNet
Open loop: use ArmNet in addition to OpenSim to generate a new dataset
Closed loop: generate data as in Step 3 but using OpenSim rather than the constant ArmNet
⟶ This produces a new set of excitation and state variable pairs

Step 5:
Iteratively repeat Steps 3 and 4 to generate "Generation X ArmNet" and "Generation X ControllerNet"

Hyperparameter Tuning and Other Experimentation

Working in TensorFlow[6], we compared the use of LSTM (long short-term memory) units to GRUs in our model. We found that GRUs performed slightly better, so we chose to use them.

We performed a hyperparameter sweep across:
- batch size (ranging from 8 to 256 in powers of 2)
- learning rate (ranging from 1e-5 to 0.1 in log scale)
- number of hidden units (ranging from 512 to 2048 in powers of 2)

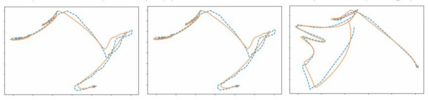


We chose to use a batch size of 128, learning rate of 0.07, and 1024 hidden units.

To deal with exploding gradients, we incorporated L2 regularization (with a parameter of 0.0001) and gradient clipping (set at a maximum of 10).

Results

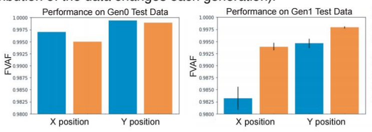
ArmNet acts as a good approximation for OpenSim. The output kinematics from ArmNet (dotted blue) closely approximate those from OpenSim (orange).



Performance of each generation of ArmNet on test set data from Generation 0 and Generation 1 also shows that ArmNet is a good approximation for OpenSim.

	X position (Gen 0 data)	Y position (Gen 0 data)	X position (Gen 1 data)	Y position (Gen 1 data)
ArmNet Gen 0	99.75%	99.85%	98.25%	99.50%
ArmNet Gen 1	99.50%	99.80%	99.50%	99.75%

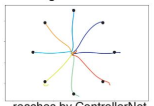
ArmNet improves across generations. ArmNet Generation K outperforms ArmNet Generation K-1 on Generation K data, while still performing well on Generation K-1 data (albeit not as well as ArmNet Generation K-1, which is expected because the distribution of the data changes each generation).

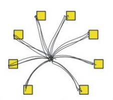


FVAF = fraction of variance accounted for

blue = ArmNet Generation 0
orange = ArmNet Generation 1

ControllerNet is able to perform the task (moving from the center of the screen to 8 radially arranged targets) and produce output kinematics similar to non-human primates performing the task.


reaches by ControllerNet


reaches by non-human primate

Future Work

- Separately tune hyperparameters for ControllerNet and ArmNet training and architectures
- Tune number of layers, regularization parameter, and gradient clipping maximum
- Compare activations for individual compartments within ControllerNet to electrophysiological recordings of neural data from non-human primates performing similar tasks [1]-[3]

References

[1] D. Sussillo and O. Barak, "Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks," Neural Comput., vol. 25, no. 3, pp. 626–649, Mar. 2013.
[2] V. Mante, D. Sussillo, K. V. Shenoy, and W. T. Newsome, "Context-dependent computation by recurrent dynamics in prefrontal cortex," Nature, vol. 503, p. 78, Nov. 2013.
[3] O. Barak, "Recurrent neural networks as versatile tools of neuroscience research," Curr. Opin. Neurobiol., vol. 46, pp. 1–6, Oct. 2017.
[4] K. Cho et al., "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," arXiv [cs.CL], 03-Jun-2014.
[5] S. L. Delp et al., "OpenSim: open-source software to create and analyze dynamic simulations of movement," IEEE Trans. Biomed. Eng., vol. 54, no. 11, pp. 1940–1950, Nov. 2007.
[6] M. Abadi et al., "TensorFlow: A System for Large-scale Machine Learning," in Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, Savannah, GA, USA, 2016, pp. 265–283.