Learning neural representations of complex motor behavior with recurrent neural networks

Xulu Sun
Department of Bioengineering
Stanford University
xulu2@stanford.edu

Mark H. Plitt
Department of Neurobiology
Stanford University
mplitt@stanford.edu

1 Abstract/Introduction

An increasingly popular approach in neuroscience is to train recurrent neural networks (RNNs) to perform tasks done by animals in the laboratory. The dynamics in these networks often resemble those seen in the real brain, and understanding how the RNN solves the task may provide mechanistic insight into how the real nervous system does the same. For our project, we trained RNNs to perform motor learning tasks in which an agent must control a cursor on a screen and move it to a target location. These RNNs are intended to serve as models for a large dataset of neural recordings in premotor and motor cortices (150-200 neurons recorded simultaneously) of two monkeys as they controlled a manipulandum to generate point to point arm movements. The animals had to subsequently learn to control the cursor after a curl force field (continuous force perturbations perpendicular to movement direction) was applied to perturb the cursor movement. In order to model this task as a supervised learning problem, we trained the RNN to generate the kinematics of smooth reaches to a target location. The inputs to our RNNs are three timeseries–the horizontal and vertical locations of the target as step functions (like a target appearing on the screen) and a pulsed "go cue" that indicates when to start the movement. The output of the networks are the vertical and horizontal components of the position, velocity, and acceleration for a smooth reach to the target that begins with the go cue (Fig 1). In addition, we explored training reinforcement learning agents to perform the same tasks (Fig 4).

2 Related work

A growing number of neuroscience studies train RNNs to model neural computations in cognitive or motor tasks (Mante et al., 2013; Hennequin et al., 2014; Sussillo et al., 2015; Michaels et al., 2016; Rajan, Harvey and Tank, 2016). Examples include modeling limb kinematics or electromyography during a motor task and psychometric curves during decision making tasks. Other RNN models, such as the latent factor analysis via dynamical systems (LFADS), are trained to infer latent dynamics from single-trial neural spiking data (Pandaraninth et al., 2018). Despite the growing use of these models in the neuroscience of motor control, to our knowledge, RNNs have not been built to perform a point-to-point reaching task, nor have they been used to learn to compensate for perturbations that alter the reaching behavior.

3 Dataset and Features

We are modeling the kinematics of idealized animal reaches, so we are able to generate synthetic data for training and testing. This gives us an essentially infinite train and test sets. Position traces were modeled as sigmoidal traces that saturated at the target location (Fig 1). The velocity and acceleration traces were the first and second derivatives of this position trace respectively. The activity of hidden units of the RNN were compared qualitatively to the activity of neurons recorded from the two monkeys.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

4 Methods

'*Standard' sequence model:* The target locations and go cue are fed into a recurrent neural network with 128 "ReLU" hidden unit activations. The output of the RNN is fed to a "dropout" layer (p=.3), and then one fully connected ReLU layer (128 units) before going to a linear output layer with six units, corresponding to the 6 kinematic output variables (Fig 1A). The network was trained to minimize the squared error between its outputs and the ideal kinematic timeseries. This model and all models below were trained using the Adam optimizer. We found that the inclusion of the fully connected layer after the RNN as well as the dropout prior were critical to performance of the network. In order to model force perturbations, the kinematics resulting from the force field were added to the output of the network before computing the loss. The model thus had to learn to output the compensated reach kinematics.

Sequence model with output buffer: This network received same three inputs described for the previous model as well as a history of it's recent outputs, called the "output buffer". We found that the four previous time points were sufficient input to get the network to make smooth reaches. Force perturbations, in this case, need to be added in real time. To make this more straightforward, the network simply outputs two dimensional accelerations for the hand. The force perturbation can be applied directly to these outputs as the addition of a constant when the acceleration places the hand in the force field. The network is then trained to minimize the error between the ideal cursor displacement time series and the double integral of the output layer acceleration timeseries.
We had to make several additional changes to the architecture in order for this model to work well. Two fully connected layers (128 neurons) with dropout (p=.3) were added before the recurrent layer. The hidden units of the RNN were also changed to GRU cells. This provided a modest improvement over ReLU or tanh hidden units (Fig 2A). Making these architecture changes to the 'standard' sequence model did not improve performance on the force perturbation trials.

Tabular Q-Learning: Allowing the network to view it's recent outputs and the targets is effectively letting the network view the "state" of the hand and make the best movement to the target. This problem statement is perhaps more natural to approach from a reinforcement learning perspective. To get a baseline for how a reinforcement learning agent would perform the task, we trained an agent using tabular Q-learning. The 'state' is the location of the agent and the location of the reward. Adding multiple reward locations without doing any function approximation is equivalent to training multiple agents, so we keep the reward location fixed. The agent was allowed to move a maximum of one square in each direction, giving a total of 8 possible actions. To motivate direct movements toward the reward location, the agent receives a reward of -1 for every time point until it reaches the target where the reward is 0. To model force perturbations, we deflected the movements of the agent by one square when the agent was in the force field. The agent can move in a straight line perpendicular to the field by making diagonal steps.

Deep Q-Network (DQN): We also trained a DQN to perform this task (Mnih et al, 2015). The problem is set up as above, but we also change the target location randomly on each trial. For settings where the number of available positions was large (i.e. large screen size), the agent receives very sparse informative feedback. This dramatically slowed training and seemed to add instabilities to TD updates. To deal with this scenario, we performed graded training in which the radius around the target that was rewarded started large and gradually decreased until the agent had to reach the target exactly.
We trained a model to perform this task directly from pixels. We found that this actually learned faster than trying to train from a parameterized version of the task. The input to the network were 16 x 16 images with two color channels. The target location was encoded as a one-hot matrix in the first channel, and the agent's location was encoded as a one-hot matrix in the second channel. These inputs were given to two convolutional layers with eight channels and 16 channels respectively (kernel size= 3). The outputs of the convolutional layers were then fed to two fully connected layers (128 neurons each). A linear layer was used to project down to the eight action values.

5 Experiments/Results/Discussion

In a reaching task with no force perturbations, the standard sequence model was able to reproduce the target kinematics timeseries with acceptable accuracy (Fig 1B & C). This network, however, was unable to learn to compensate for force perturbations (Fig 1D & E).

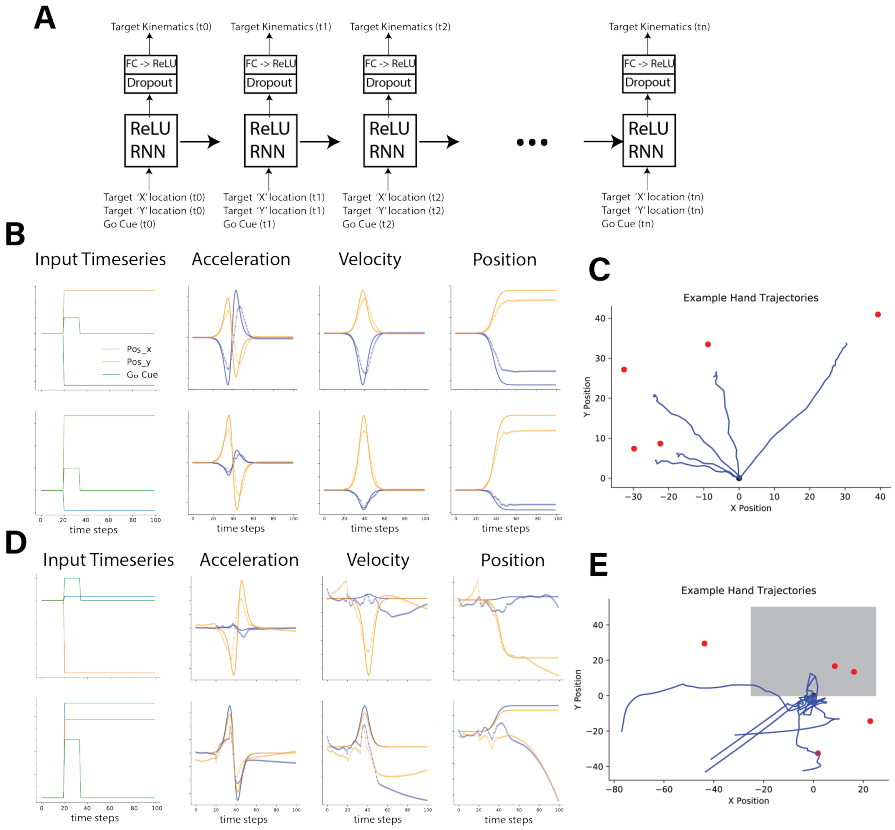

Figure 1: Performance of the 'Typical' sequence model on reaching task with and without perturbations. **A** Schematic of sequence model **B** Example input timeseries and output kinematics of RNN trained without force perturbations. *Left* Input timeseries to RNN (blue-target x location, orange-target y location, green - go cue). Target kinematics (bold; x component - blue, y component - orange) and the outputs of the model (shaded) are shown to the right. **C** Example hand trajectories from model. Red dots indicate target locations and position timeseries are shown in blue. **D** Same as **B** after the network was trained to compensate for force perturbations. Note network outputs fail to match target timeseries. **E** same as **C** but for the network shown in **D**. Shaded region indicates the location of the force perturbations

It is perhaps expected that the standard RNN above would not be able to compensate for location specific force perturbations. That network has no natural way of representing locations where the force perturbation will occur. In order to address this issue, we designed a network that could more easily detect force perturbations during a reach by allowing the network to receive a recent history of it's outputs, which will be affected by force perturbations (Fig 2A). This sequence model with an "output buffer", performed accurate reaches to target locations with and without force perturbations (Fig 2B & C). Performance actually increased during force perturbations. We suspect that this is due to the network upweighting the recent history of outputs as they become more relevant for compensating for perturbations. When the force field was removed the network quickly recovered performance and made more accurate reaches than before force perturbations.

For the "output buffer" network, we compared activity of single units in the hidden layer to activity of single neurons recorded in vivo (Fig. 3). Both biological neurons and RNN hidden units changed activity levels significantly after the "go cue" turns on, and show different levels of activity for the different reaching directions. RNN units tended to have multi-phasic oscillations that are not quite typical of actual neural activity; however, the peristimulus time histograms show qualitative similarities. In future analyses, we would like to investigate the extent to which the low dimensional dynamics of the entire network are similar to low dimensional dynamics of the neural population by performing dimensionality reduction (see Mante et al (2013) for a similar approach).

We next investigated the extent to which reinforcement learning agents were able to solve the same tasks. In a discretized 10x10-grid environment, a tabular Q-learning agent can quickly learn to make

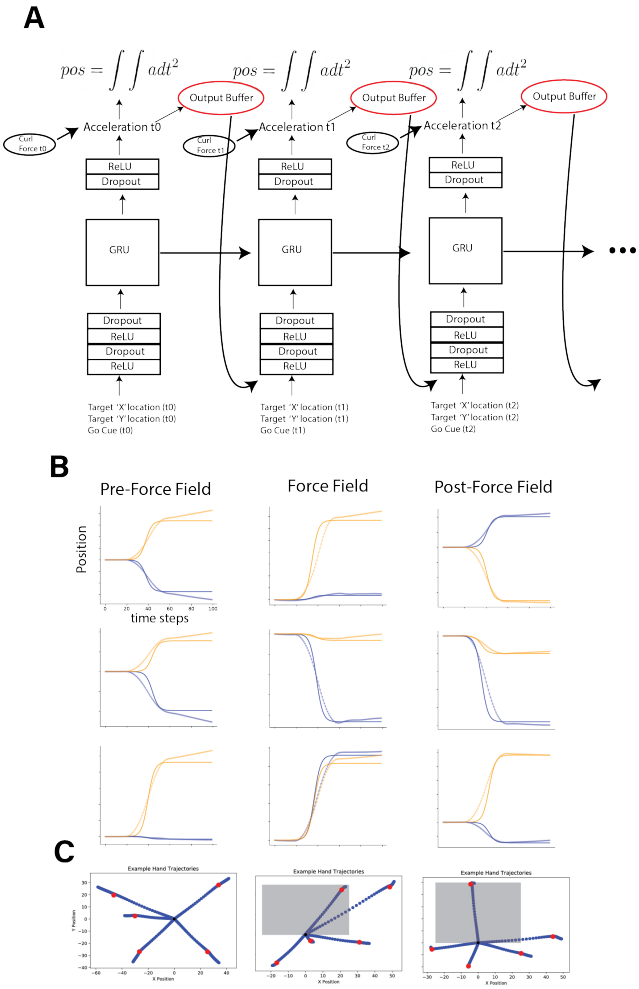

Figure 2: Performance of the sequence model with an output buffer on reaching task with and without perturbations. **A** Schematic of sequence model **B** *Left* target position timeseries (bold) and model estimated position timeseries (shaded) for a network trained without force perturbations.*Middle* Target and output position timeseries after model is trained with force perturbations. *Right* Target and output timeseries after force perturbations are removed. **C** Example hand trajectories for models shown in **B**. Shaded region indicates location of force perturbations

direct movements to a fixed reward location with or without force perturbations (Fig 4). We were also able to train a convolutional DQN agent to move a cursor to a randomly placed reward directly from pixels (Fig 4). This agent actually trained faster than a DQN trained from a parameterized version of the task. We believe the convolutional DQN was able to quickly find an efficient embedding of the task for which it could learn the action values more readily than our chosen parameterizations. For a variety of initial locations, the agent made direct movements towards the target. When comparing the performance of the network to an ideal agent that moves straight to the target with minimal moves, we see that we achieve ideal performance for even some very long trajectories, but that the agent typically makes a few superfluous moves before reaching the target. This may reduce with further training. We implemented force perturbations for the DQN but it was not able to learn to compensate well. Including the last several frames as additional inputs to the DQN did not dramatically improve performance. A DQN is likely capable of performing this task with a more thorough hyperparameter search.

6 Conclusion/Future Work

In this project, we trained four different model classes (a typical sequence model, a sequence model with an output buffer, a tabular Q-learning network and a DQN) to perform motor learning tasks in which an agent must control a cursor on a screen and move it to a target location. All four models were able to learn the basic point-to-point reaching task well; only the output buffer sequence model and the tabular Q-learning model could readily learn to correct for location specific force perturbations. The output buffer RNN could learn the basic center-out reach task and the force perturbations much

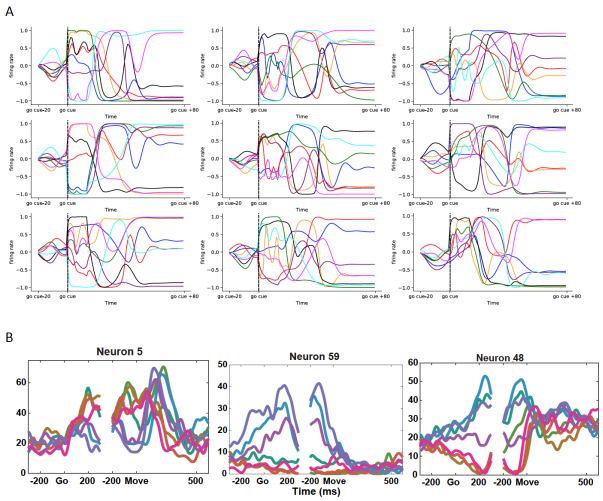

Figure 3: Example hidden unit activity (A) and recorded neural activity (B)

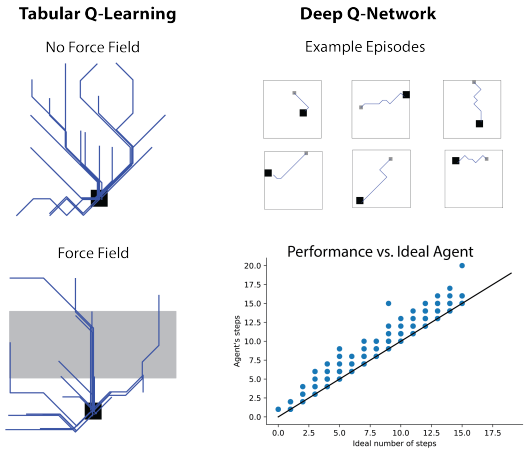

Figure 4: Performance of tabular and Deep Q Network reinforcement learning agents. *Top Left* - Example trajectories of tabular Q-learning agent trained to navigate to fixed goal location. Goal location is indicated by black square and blue lines show agent trajectories.*Bottom Left* - Example trajectories of tabular Q-Learning agent with force perturbations (shaded region). *Top Right* - Example trajectories of agent trained with a Deep Q-Network. *Bottom Right* Performance of DQN vs an ideal agent that takes a direct vector to the target. Unity line is shown for comparison.

better than the typical sequence model, with hidden layer units showing more biological neuron-like activity patterns.

There are three main thrusts to our next steps. First we could vary the task parameters, such as making the force perturbations non-static, to better simulate the actual motor task performed by animals and study how the neural networks learn the more complicated, new task. Second, it would be interesting to explore how agents trained to output continuous actions, such as DDPG networks (Lillicrap et al, 2019), could perform this task. In our current approach, the agent cannot directly overcome the force perturbation by making "stronger movements". It can only compensate by moving in a straight line perpendicular to the force field. In addition, the nervous system must act by controlling the magnitude of different muscle contractions, so an agent that must make continuous instead of discretized actions is likely a more relevant model for how the nervous system performs this task. Third, in this work, we compared units of the recurrent layer to single neurons recorded from the motor cortex during comparable reaches. The next step would be to compare the low dimensional activity of the RNN to the low dimensional activity of the neural population. We are particularly interested in how the RNN adjusts its activity to compensate for the force perturbations and how this compares to the neural data.

7 Contributions

MHP and XS conceptualized project and model classes and prepared manuscript. XS performed electrophysiology experiments, wrote implementation of initial sequence model using teaching signal (not shown), performed comparisons of hidden units to neural data, and performed hyperparameter

searches for 'typical' sequence model and 'output buffer' sequence model. MP wrote implementations for and trained initial instantiations of 'typical' sequence model, 'output buffer' sequence model, tabular Q-Learning model, and DQN.

References

All code used to generate results shown in this manuscript can be found at `https://github.com/markplitt/CS230Project`

All code was written in python and used standard packages from the Anaconda distribution. In addition, all models, except the tabular Q-learning agent, were trained using PyTorch

[1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609–616. Cambridge, MA: MIT Press.

[2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer–Verlag.

[3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.

[4] Mante, V., Sussillo, D., Shenoy, K. V., Newsome, W. T. Context-dependent computation by recurrent dynamics in prefrontal cortex. *Nature* vol. 503 78–84 (2013).

[5] Hennequin, G., Vogels, T. P., Gerstner, W. Optimal control of transient dynamics in balanced networks supports generation of complex movements. *Neuron* 82, 1394–1406 (2014).

[6] Sussillo, D., Churchland, M. M., Kaufman, M. T., Shenoy, K. V. A neural network that finds a naturalistic solution for the production of muscle activity. *Nat. Neurosci.* 18, 1025–1033 (2015).

[7] Michaels, J. A., Dann, B., Scherberger, H. Neural population dynamics during reaching are better explained by a dynamical system than representational tuning. *PLOS Computational Biology*, 12(11), e1005175. doi:10.1371/journal.pcbi.1005175 (2016).

[8] Rajan, K., Harvey, C. D., Tank, D. W. Recurrent Network Models of Sequence Generation and Memory. *Neuron* 90, 128–142 (2016).

[9] Pandarinath, C. et al. Inferring single-trial neural population dynamics using sequential auto-encoders. *Nat. Methods* 15, 805–815 (2018).

[10] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. *Nature* 518, 529–533 (2015).

[11] Lillicrap, T., Hunt, J., Pritzel, A., et al. Continuous control with deep reinforcement learning. *arXiv* :1509.02971 (2019).