Learning the Motor Transfer Function of a Robot Arm with Neural Networks

Timothy M. Shea

Overview: To investigate computational challenges related to prediction in the neural control of movement, I trained a series of neural networks to predict endpoint trajectories of a simulated robotic arm as a function of a motor command. Although I succeeded in identifying features of training that were essential for good performance either locally or globally, none of the trained models was qualitatively successful even on a highly controlled dataset. This may have implications for theoretical descriptions of motor control which depend on highly accurate motor prediction.

Background: Forward models are an important theoretical construct in neuroscience. Although originally defined in terms of predicting factory output with production delays of weeks or months, forward models have been readily adapted to describe predictions on the order of milliseconds in robotic and biological control processes. Theoretically, the benefit of a forward model is the ability to adapt a noisy control signal before sensory feedback has been processed (**Figure 1**). Forward models are considered essential for the acquisition and production of smooth motor control (Marr 1969, Doya 2000). Neural circuits dedicated to predicting the sensory outcomes of actions are generally associated with the cerebellum and olivary nucleus.

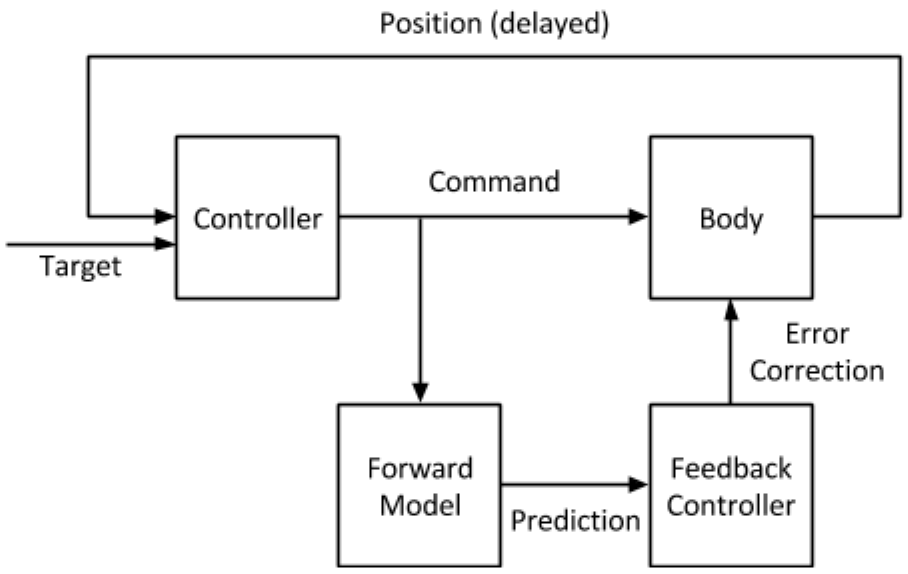


Figure 1: *Forward models in motor control allow for error correction based on predictions. If the quality of predictions is good, this can reduce the latency of the feedback control process and largely eliminate oscillations around a target, particularly when it is not possible to learn a "perfect" controller.*

Problem: Learning a predictive model is generally considered an "easy problem" in motor control[1]. As commands are sent to the body, copies of the commands are sent to the forward model, which generates predictions. The predictions and the observed positions from the body can then be compared to generate a prediction error.

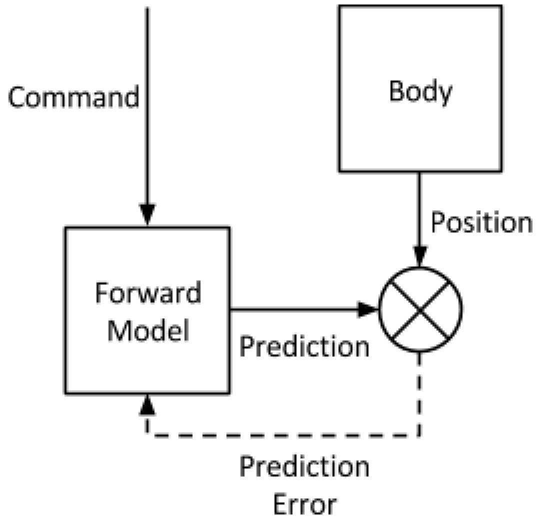


Figure 2: *Forward models are considered easier to learn than controllers (inverse models) due to the directly observable training signal.*

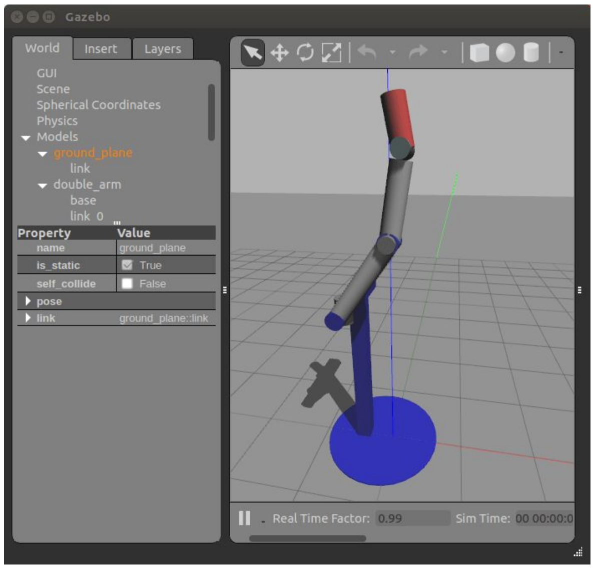
Nevertheless, for a motor system with many nested degrees of freedom, the effect that a motor command will have on the body varies across time and space. Even in the case of a fully deterministic motor transfer function (i.e. the function computed by the plant to transform the command into the position), there may be many sources of nonlinearity which interact in complex ways. Learning a general predictive model of such a system may not be useful unless it can capture both the global and the local characteristics of the transfer function.

I trained a series of dense neural networks of varying complexity on a motor prediction task using a dataset of simulated robot arm movements. During an initial phase of model exploration, I evaluated the performance of all models under a variety of training conditions to identify the models and hyperparameters with the greatest potential performance. In a second model tuning phase, I explored many more combinations of hyperparameters for two of the models.

[1] Relative to learning a control function, or an action selection policy, neither of which generally has a unique solution or a fully observable teaching signal.

Dataset: I developed a robot arm model in gazebo. Gazebo is a physically-realistic robot simulation platform in which a robot is specified in terms of 3d shapes and physical properties (Koenig & Howard 2004). The arm had a fixed based supporting 3 limb segments, each attached by a 1-axis revolving joint to the prior segment (i.e. shoulder -> elbow -> wrist). Each joint angle was clamped to approximately the range of a human arm and each joint applied a constant friction of 1 Nm. The hand of the robot was therefore only capable of motion in the X-Z plane. Gravity was disabled for all arm segments.

The robot arm model was programmatically driven by an arm controller script written in python and interfaced via Robot Operating System. The arm controller generated pseudo-random values for joint efforts (torques) and applied the efforts at fixed, 20 ms intervals to the simulation. The controller applied joint efforts in 5 s runs and recorded the position of the hand at each timestep. After each run, joint effort and hand position matrices were saved and the simulation was reset to initial conditions prior to the next run. 5000 runs were performed.



Property	Value
name	ground_plane
is_static	✓ True
self_collide	False
pose	
link	ground_plane::link

Real Time Factor: 0.99 Sim Time: 00 00:00:0

Figure 3: *Recording of the gazebo simulation while the arm controller is running. The hand jumps back to the starting position between runs. See the ipython notebook for animation.*

In principle, the motor transfer function of the robot arm model should be similar to the following:

$a_i(t)$ is joint angle for joint i at time t

$j_i(t)$ is joint effort for joint i at time t

$f(j)$ is friction, if $|j| < k$, then $f(j) = j$, else $f(j) = k \,\mathbf{sign}\,(\mathrm{j})$, k is a constant

$da_i(t) = (j_i(t) - f(j_i(t)))/m_i$, m_i is the radial inertia around joint i

$a_i(t) = a_i(t-1) + da_i(t) = sum_{0:t}[da_i(.)] + a_i(0)$

l_i is length of segment i, c_x and c_y are the constant offsets for x and y

$$x(t) = l_1 \cos(a_1(t)) + l_2 \cos(a_1(t) + a_2(t)) + l_3 \cos(a_1(t) + a_2(t) + a_3(t)) + cx$$

$$y(t) = l_1 \sin(a_1(t)) + l_2 \sin(a_1(t) + a_2(t)) + l_3 \sin(a_1(t) + a_2(t) + a_3(t)) + cy$$

That's not quite right for a couple of reasons:
(a) the simulation used to generate data is based on iterated differential equations with much greater temporal precision than the 10ms steps of our input/output, and the lagged cross correlations above show that the effects of effort are not simply delayed by 1 timestep
(b) the radial inertia m_i is itself a complex nonlinear function of the joint angles
(c) the joint angles are bounded, which means $da_i(t)$ has a nonlinear dependence on $a_i(t-1)$

Nevertheless, the dominant feature of the transfer function should be nested sinusoids, and the entire function is deterministic.

Model Exploration: In the initial phase of the project, I explored a series of models beginning with a very basic single hidden layer network and proceeding to a 2d convolutional network (**Table 1**). Many model variants were tested only once or twice, and abandoned if no significant improvement was observed in training.

Model	**Description**	**Qualitative Performance**
Dense1	250 ReLU	Converges quickly, overfits within 100 epochs
	1000 ReLU	Better test error, more overfitting
Dense3	3 x 80 ReLU	Converges quickly, better performance than Dense1, overfits
	3 x [250, 500, 1000] ReLU	Converges, better train error but same test error as Dense3
	5 x 250 ReLU	Converges with small learning rate, but asymptotic train error is poor
Conv1	16 1d Conv ReLU > 250 ReLU	Good train and test error, overfitting not as bad as above
Conv2	16 2d Conv ReLU > 250 ReLU	Best train and test error, does not generally overfit
	2 x [16, 32, 48] 2d Conv ReLU > [1, 2] x [250, 500] ReLU	Good train and test error, overfits quickly, poor predictions

Table 1: *Qualitative summary of model exploration.*

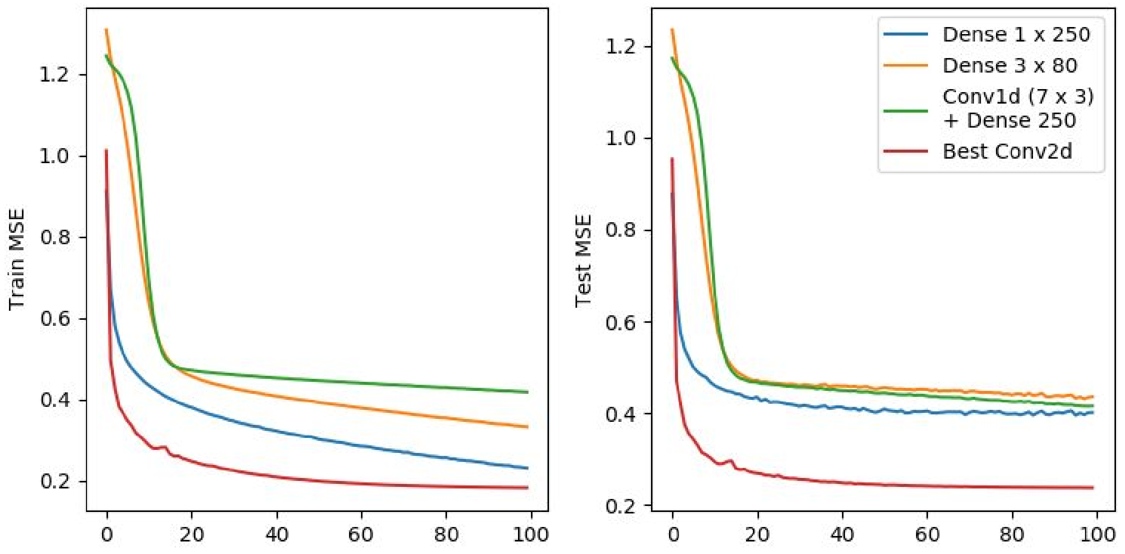


Figure 4: *Train and test error by epoch for several models during model exploration.*

While the best models in the model exploration phase performed reasonably well in terms of the loss function, the resulting predictions were not satisfying. Even overfit models produced trajectories which seemed to exhibit much more random, high-frequency variation than was present in the actual trajectories. The high-frequency noise allowed the envelope of the prediction to sometimes track the true position, even as the mean was far from accurate, thus some of these models perhaps captured the local structure but missed the global structure of the problem.

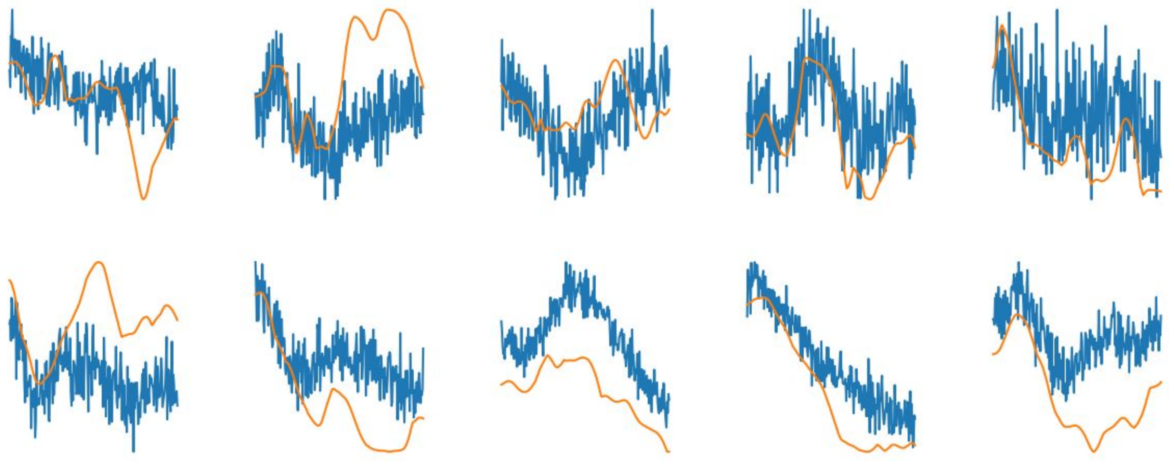


Figure 5: *Predictions (blue) made by an overfit Dense1 network are much noisier than actual trajectories (orange). The top row represents the X coordinate, the bottom row represents the Y coordinate. Each column is a randomly selected trajectory.*

At the end of the model exploration phase of the project I discovered that applying dropout to the inputs (not just the hidden layer activations) significantly altered the predictions of the models (**Figure 6**). These "de-noising" predictors were dramatically underfitting the problem, but were often more successful at predicting long-term movement of the hand.

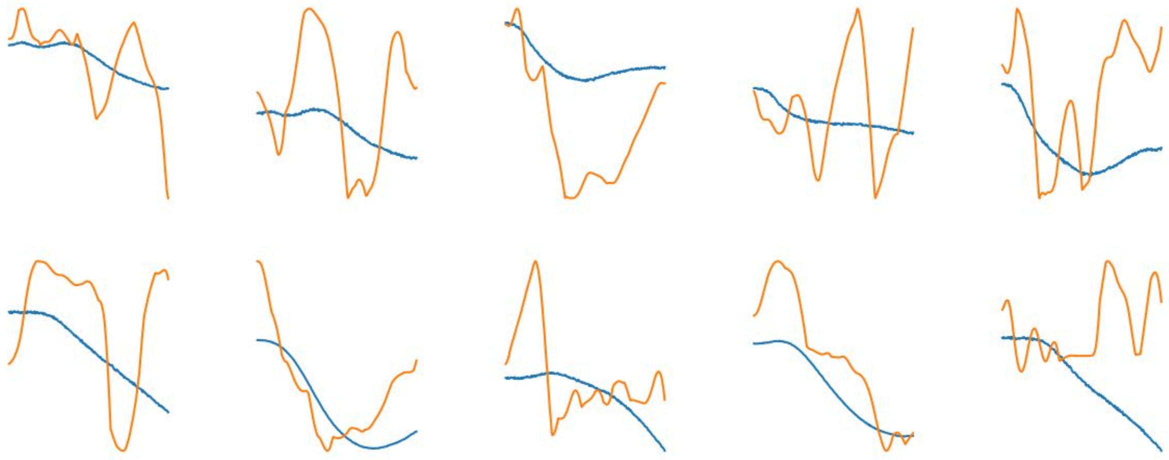


Figure 6: *Predictions (blue) made by an underfit Conv2 network are smooth, but lack many of the local fluctuations of the actual trajectories (orange).*

Model Tuning: To determine whether a model could be made to fit both the global smoothness and long-term structure and the local, rapid fluctuations in the trajectories, I focused on 2 models in greater detail.

The most important hyperparameters for asymptotic performance of the models in this task are batch size, dropout rate, and kernel size for convolutional networks. Model architecture overall does not seem to be crucial, although larger models tend to be more susceptible to overfitting. The best performing model overall was a version of Conv2 which achieved test MSE of 0.12. The Dense1 model also performs quite well with high input dropout, generally reaching test MSE around 0.15. These models produce noticeably more accurate predictions in the best cases than the exploratory models, with reduced but not completely eliminated high-frequency variation (**Figure 7**).

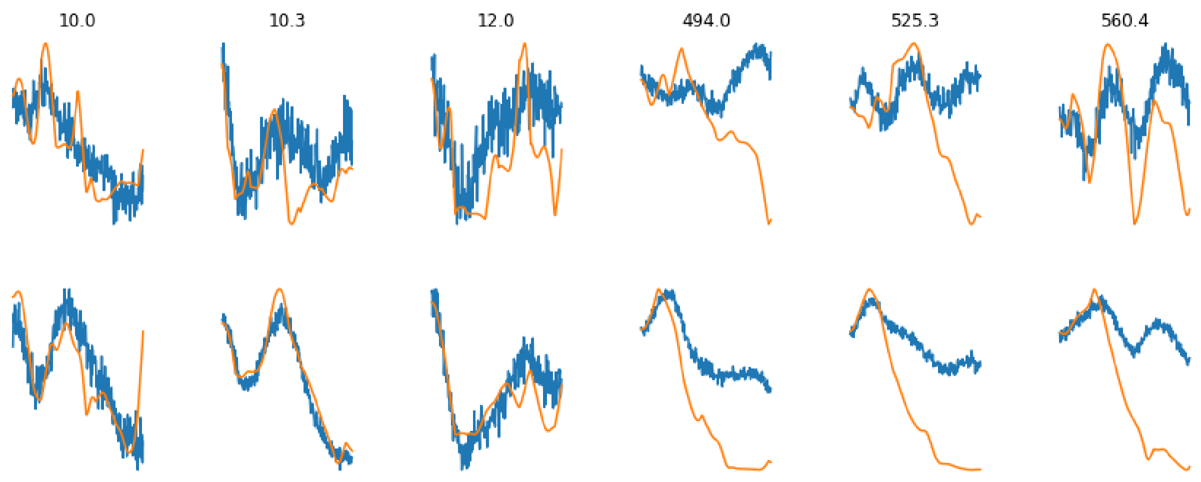


***Figure 7:** Best (left 3) and worst (right 3) predictions of the best tuned Conv2 model (Row MSE above). High-frequency variation is somewhere in between the examples above.*

However, neither of the tuned models is compelling on average. Many predicted trajectories still exhibit large deviations (sometimes late in the run), and qualitative fit is still poor.

Conclusions: It is likely that more advanced models, more finely tuned hyperparameters, or some combination would yield significantly better performance on this motor prediction task. However, the actual motor prediction task performed by animal brains must be many orders of magnitude more complex due to not just dozens of interacting skeletal joints but thousands of interacting muscles, nonstationary properties of the environment, interactions with objects, etc. If the solution to this problem were parameter tuning alone, the resulting system would likely be extremely unstable.

Instead, I would suggest that, akin to the multiple time-scales of plasticity present in neural systems, this robotic prediction problem calls out for distinct models of temporally and spatially local versus global behavior. Rather than trying to optimize a single model to fit all of the complexity in the problem, it might be preferable to train a slow-learning model on the large-scale and contextual variation, followed by a fast-learning model trained to predict only the residuals. This approach adds model complexity, but it might ultimately result in a more stable solution.