Notes on calculating steady state loss in online training simulations

The problem

Our results are interested in the tradeoff between learning speed and steady state loss in online training.

In particular, we want to look at the steady state loss vs learning speed tradeoff in online learning, as a function of learning step for different granule cell layer expansions.

The figure obtained obtained from the theoretical analysis is the following:

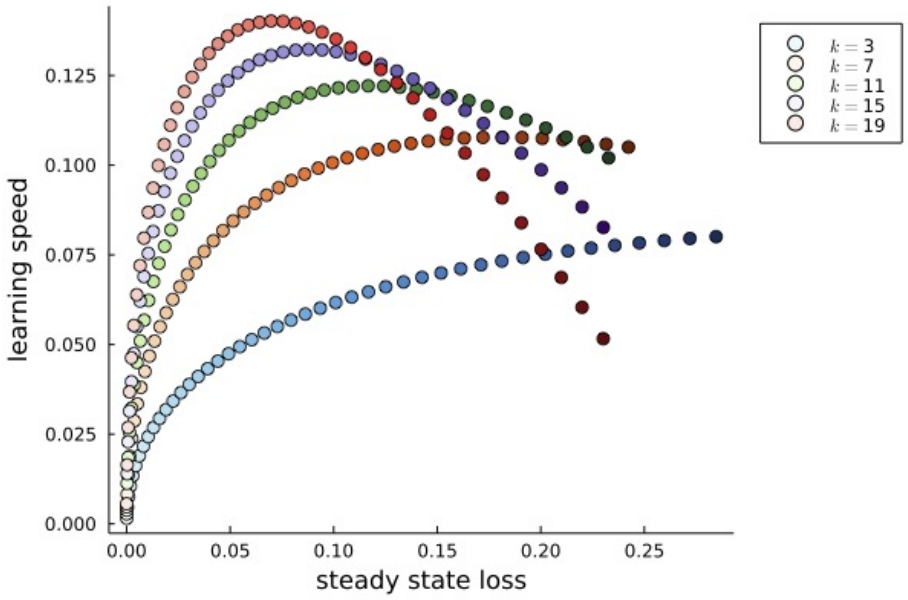

Figure 1: theoretical plot

We expect qualitatively a similar shape for the figure from the simulations. We get a different shape of plot when training with lms.

To produce this plot, we trained different systems with different sizes and different learning steps γ. For each network and each learning step, we compute the task loss at each weight update. The task loss $L[\mathbf{w}_t]$ is computed as the mse loss over the whole 400s trajectory if the weights $\mathbf{w}_t$ were frozen (i.e. no learning).

For this plot we computed the steady state loss, looking at the average of the task loss over the last 10 time-steps of learning. The system was trained with a ref trajectory, the sum of sin with random frequency under some cut-off frequency, with a length of 400s. The system was trained with lms every 1s.

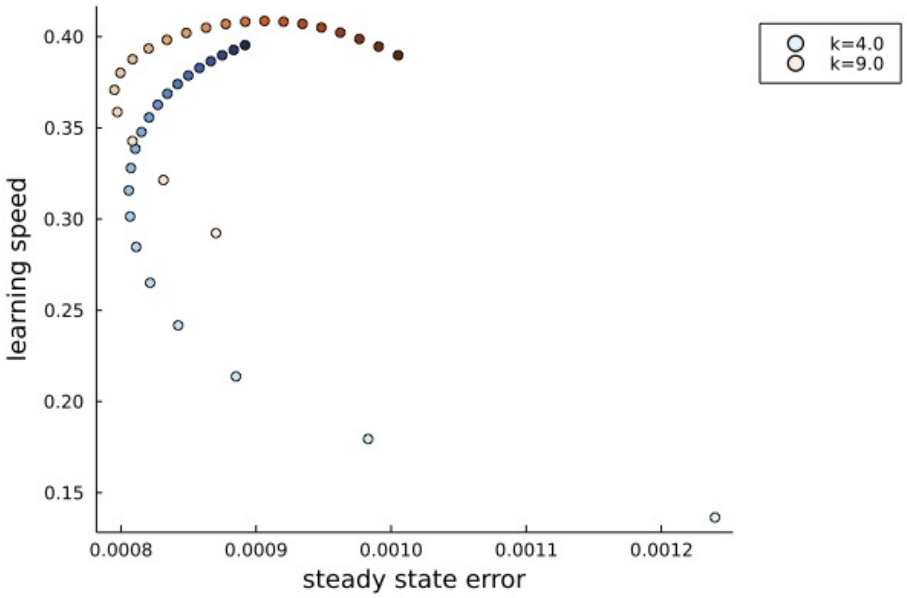

Figure 2: simulation plot

The main discrepancy is that in the simulation results, for small learning steps γ, the steady state loss ξ initially decreases as we increase γ. Whereas, in theory we would expect the steady state loss to increase as we increase γ unless learning with smaller γ has more task irrelevant plasticity (i..e worse learning direction).

What is causing the discrepancy? The main **hypothesis** is that we are not computing the steady state loss in the simulations correctly and that the systems with small γ have not reached steady state loss.

New simulations

- Consider computing the steady state by looking at the task loss when training from an initial weight state $\mathbf{w}^*$ very close to the local minimum (ideally the local minimum itself). In that case we would expect the system to begin at zero loss or close to zero, potentially increase loss until it settles at the steady state loss.
- Chose the weights $\mathbf{w}^*$ by pre-training over a much longer trajectory (between 1000s and 10000s). Also try pre-training with gradient descent instead of lms as it should reach a better solution faster.
- Change the reference trajectory to include random phase shifts as well. Potentially generate the reference signal by selecting frequencies in the Fourier series of a white noise signal. We chose to add random phase shifts

between 0 and $\pi/2$ to the reference trajectory as it is the easiest thing to try first computationally.

Need to make sure that the reference trajectory starts at zero because the plant initial state is zero. Otherwise we get the large initial transients when computing steady state loss as well. Even though the network is well trained, there is no way for it to fix the initial error when the plant will always output zero.

Pre-training with LMS

After these changes we find much better performance, we can get much better estimate of the steady state loss.

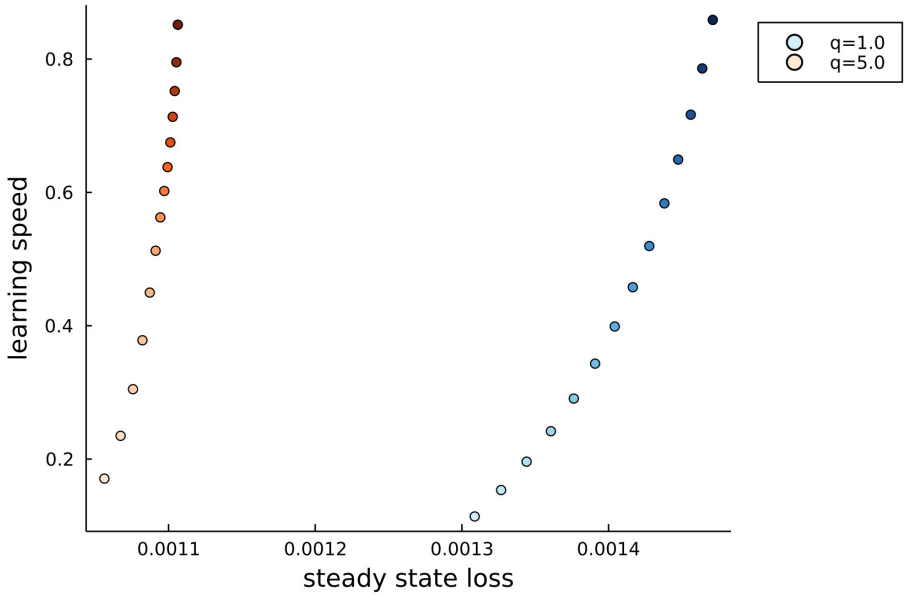

Figure 3: ss vs ls attempt 2

We see that now the steady state loss is increasing with γ instead of decreasing.

We make the following observations. - Training for a much longer period (10000s) we find some dynamics of the weights and task loss at much slower timescale that were not visible when we were only training for 400s (see fig 4 and 5).

Pre-training with gradient descent

- we tried pre-training with gradient descent instead of LMS and found that it is very very slow specially for large networks. It takes a couple of hours just to pre-train the networks. Furthermore, the task loss doesn't seem to

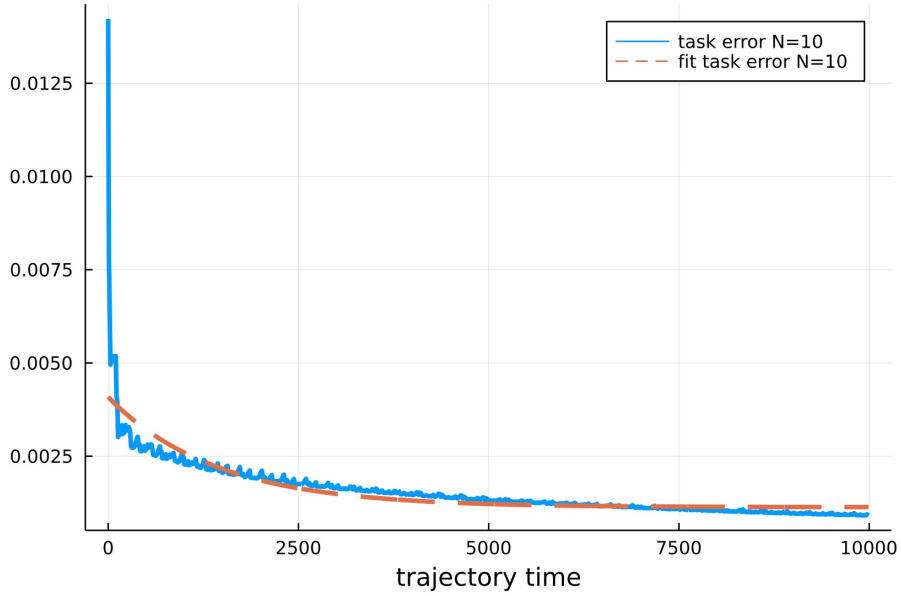

Figure 4: pre train lms long

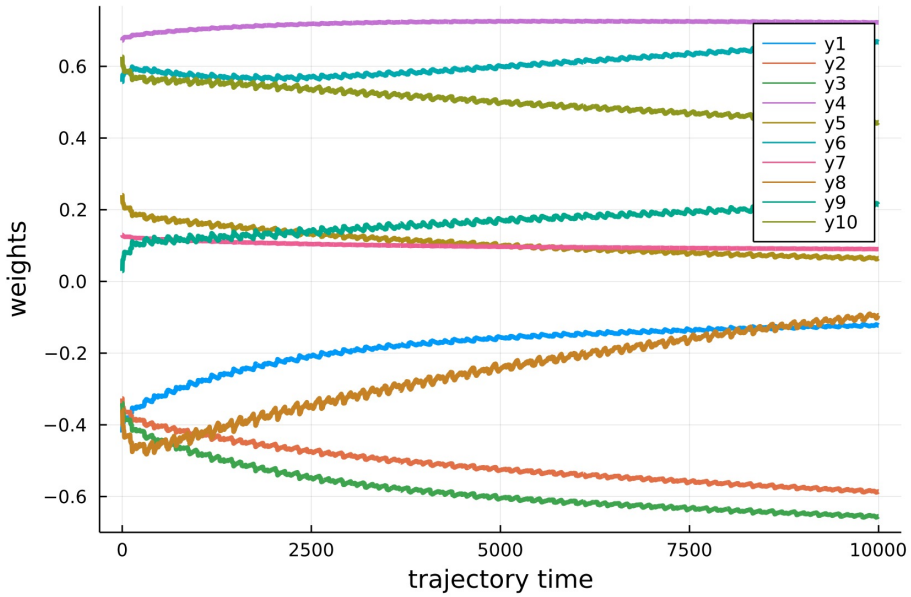

Figure 5: pre train lms long weights

converge to steady state any faster for small learning steps but we can use larger learning steps as there is not learning rule noise.

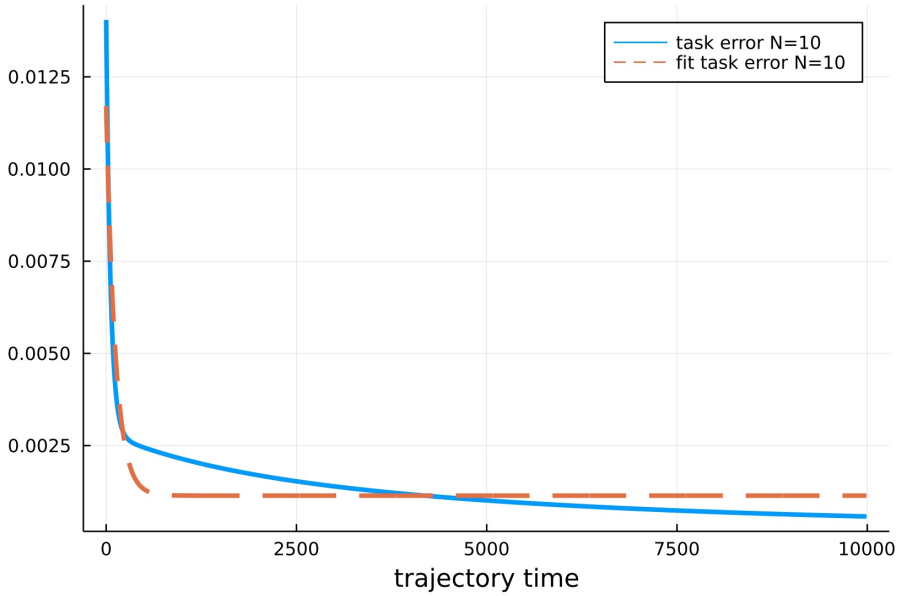

Figure 6: grad descent pre train

As expected there are no short timescale fluctuations (i.e. online learning error). Unfortunately, it doesn't seem to converge much faster than lms for the same learning step. Yet the computation time is much longer (order of hours for a large net size) as it needs to compute the gradient over the whole system for a very long time (see fig 6 and 7).

We train with gradient descent for a shorter time (5000), which is a bit more computationally feasible but the networks don't always reach steady state.

We try training with a slightly larger learning step that still allows convergence but faster. We also try training smaller networks for longer as the smaller the number of parameters, the shorter the computation time.

Observations

- There are inconsistencies in the pre-training quality between different sized networks within simulation seeds. There is a tendency for larger networks to not quite reach a good solution during pre-training. Which means that the steady state loss computed is higher for some of the larger network expansions (which we don't expect).

We observe that for $q > 6$, the steady state loss increases with the expansion.

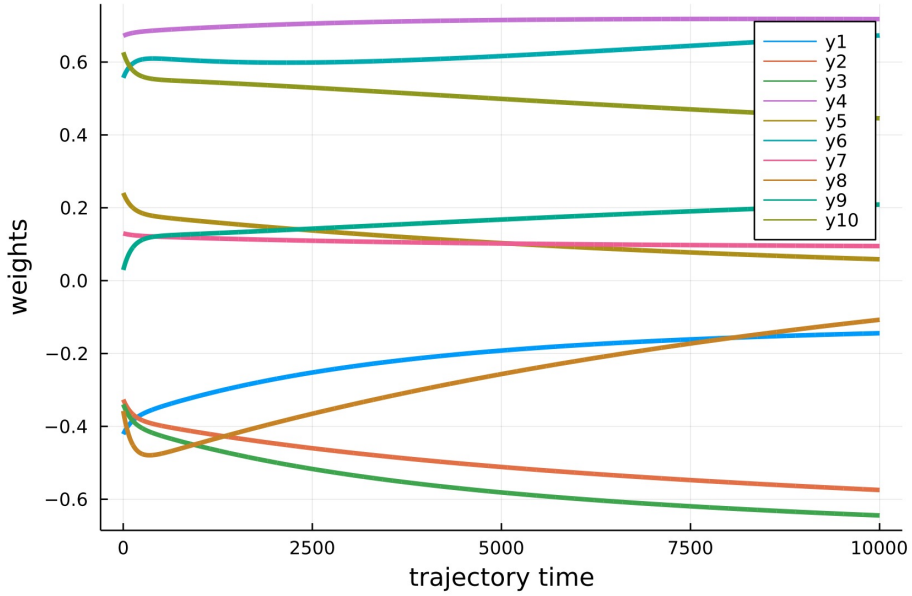

Figure 7: grad descent pre train

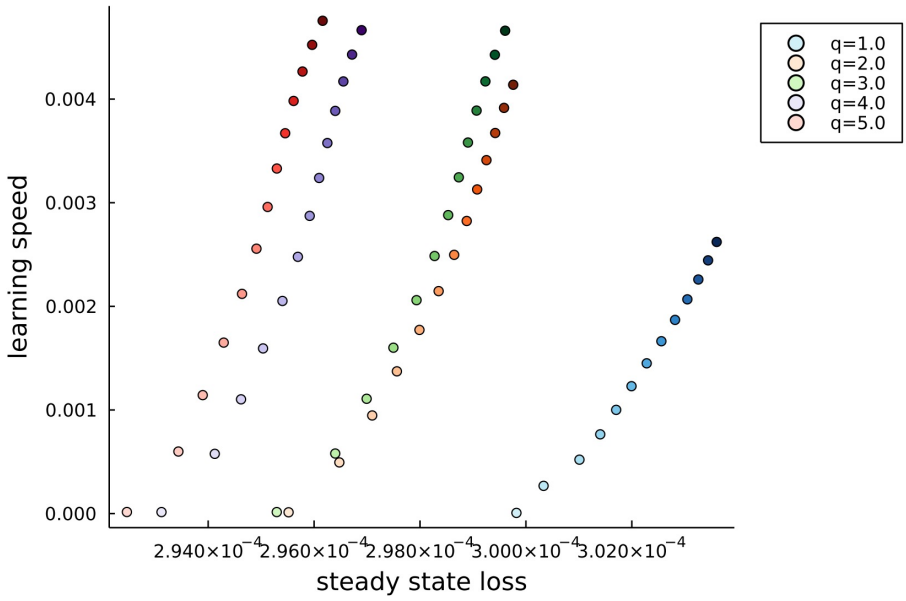

Figure 8: new tradeoff

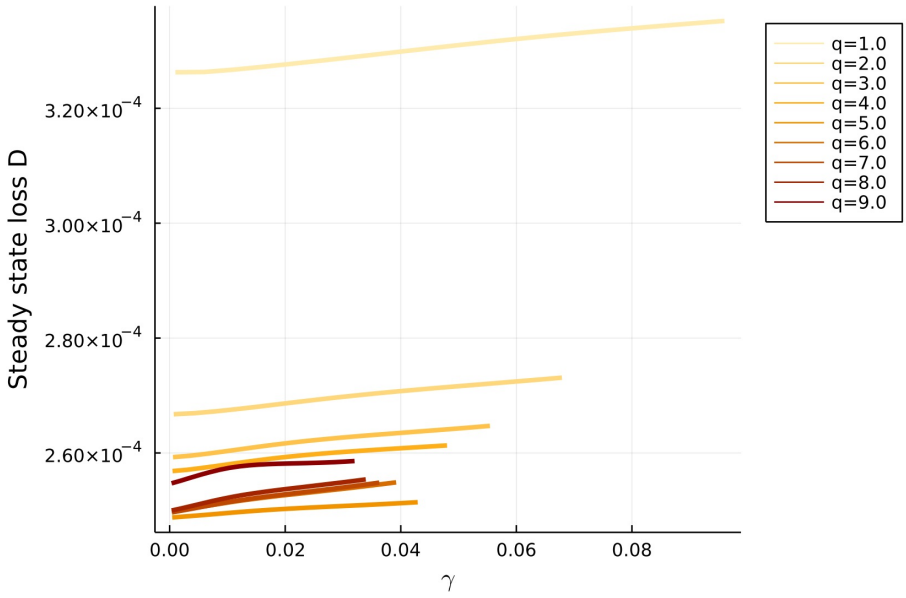

Figure 9: ss vs mu

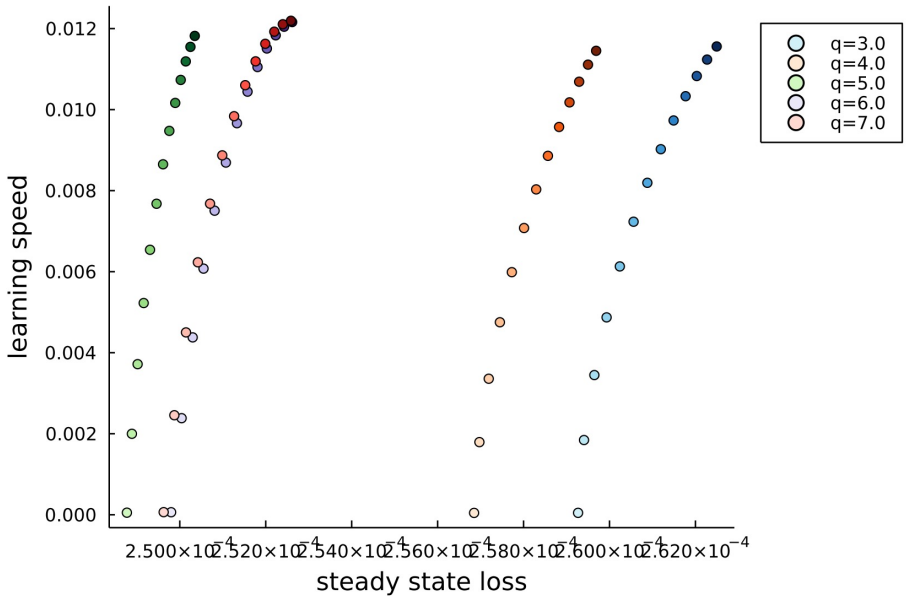

Figure 10: ss vs mu

This is unexpected.

This could be a problem with our theoretical results, but it is more likely that it is just a problem with the pre-training. Indeed the inconsistencies are different for different simulation seeds.

We need to make the pre-training more consistent to make sure that for all simulations and all network sizes we get close to minimum in the pre-training.

The only solution to this problem would be to do the pre-training differently, to get a better optimal weight state to compute steady state loss.

- From theoretical results, we expect the steady state loss to converge to zero for small enough learning step. It is hard to see this for simulations, we would have to look at much smaller learning steps and many more.
- The increase in steady state loss as we increase the learning step is relatively small. In fig 8, for $q = 1$, the steady state loss goes from 3×10^{-4} to 3.03×10^{-4}.
- Conceptually the task loss is modeling the performance on the learning goal which in this case is learning an inverse model of the plant. So ideally, the task loss would be computed over all the possible movements. This is not practical, so we will need to take a finitely long task loss. We can think the reference trajectory over which the task loss is computed as many movements put together. We chose this trajectory to be 1000s long.

Note that at this point, for a single simulation the trajectory over which we pre-train, train or compute task loss is the same. the only potential difference is how long we go for.

We still have the same issue as before. We just can't pre-train well enough!!! This effect is more visible as the tradeoff in this task between ss and ls as we vary learning step is small (i.e. steady state loss varies little). Also the benefit of adding granule cells is also small. Hence this problem is very sensitive to the pre-training and variability.

Adding gaussian noise

We tested adding gaussian noise to the learning rule with $SNR = 3$. Our theoretical results predict that having more learning rule error emphasizes the trade-off between ls and ss and the benefit of adding granule cells. Furthermore, larger learning rule error means the steady state loss is larger which should make our results less sensitive to pre-training or at least require less sensitive pre-training.

The new weight update rule is

$$\delta w_t = -\gamma \frac{1}{\Delta t_e - \Delta t_r} \int_{t-\Delta t_e}^{t-\Delta t_r} (y(t') - r(t'))dt' \mathbf{h}(t - \Delta t_h) + \eta\hat{\epsilon}$$

where $SNR = \frac{\gamma}{\eta}$ and $\hat{\epsilon}$ is drawn from a random normal distribution (and normalised for the weight update).

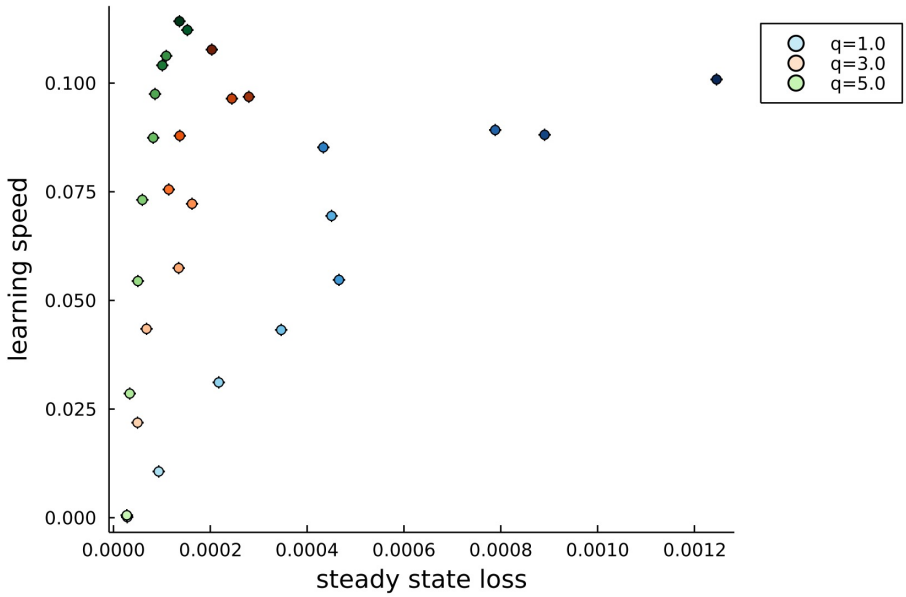

Figure 11: ss loss vs learning speed simulations with gaussian noise $SNR = 3$

As expected the trade-off holds more strongly. We get exactly what we predicted from the theoretical results. The only issue is that we need to run many simulations as the gaussian noise causes more variability in the results.

We perform 10 simulations with different reference trajectories and initial weights in the cerebellar like system. We take the mean of the learning speed and steady state loss over the simulations.

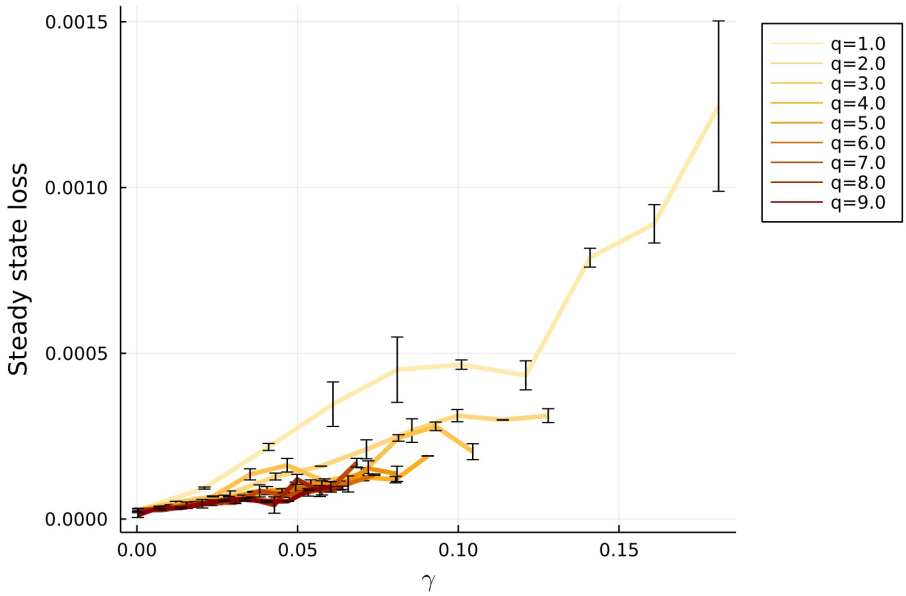

Figure 12: learning speed vs learning step simulations with gaussian noise $SNR = 3$

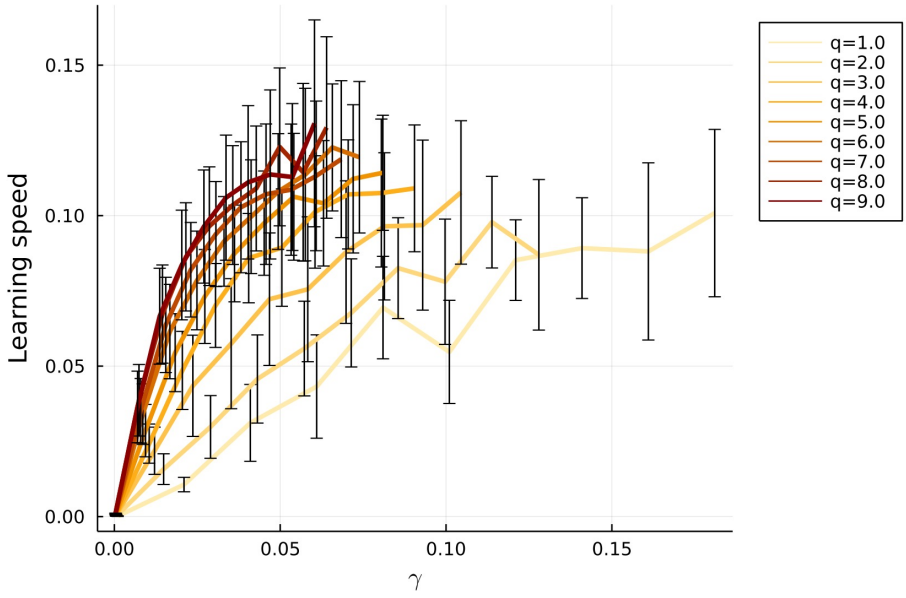

Figure 13: steady state loss vs learning step simulations with gaussian noise $SNR = 3$

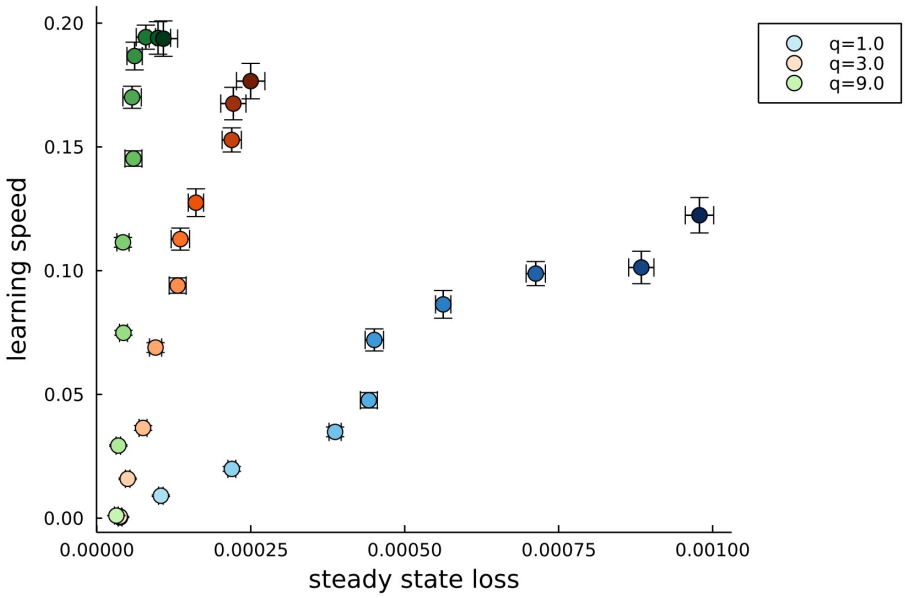

Figure 14: average learning speed vs steady state loss for 10 simulations with gaussian noise $SNR = 3$