# 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:

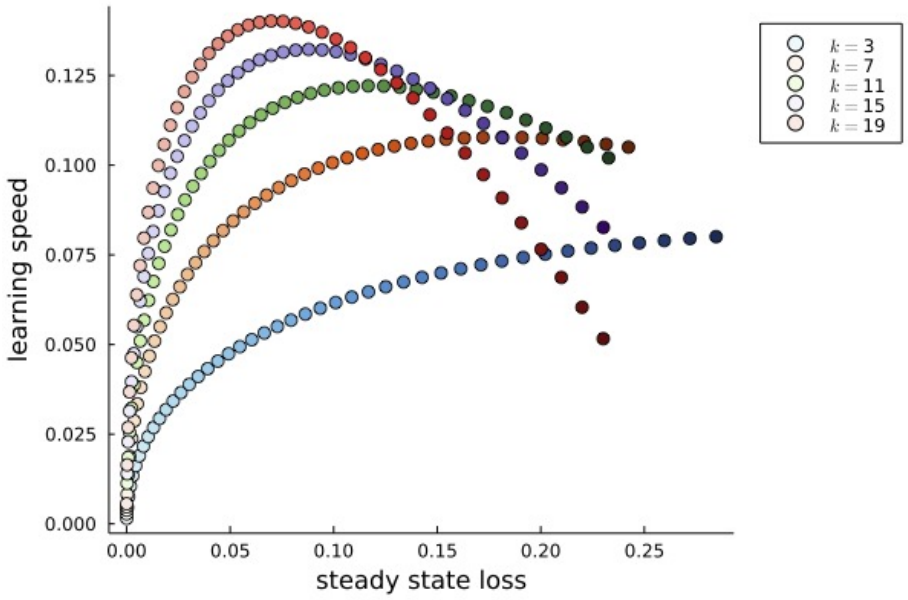


Figure 1: theoretical plot

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 $\gamma$. 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.

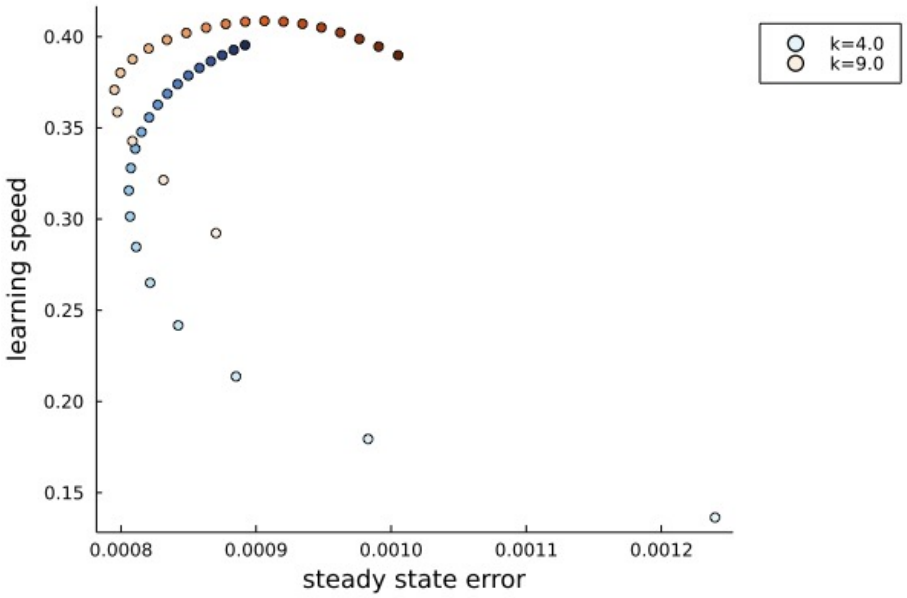


Figure 2: simulation plot

The main discrepancy is that in the simulation results, for small learning steps $\gamma$, the steady state loss $\xi$ initially decreases as we increase $\gamma$. Whereas, in theory we would expect the steady state loss to increase as we increase $\gamma$ unless learning with smaller $\gamma$ 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 $\gamma$ 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.

**Pre-training with LMS**

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

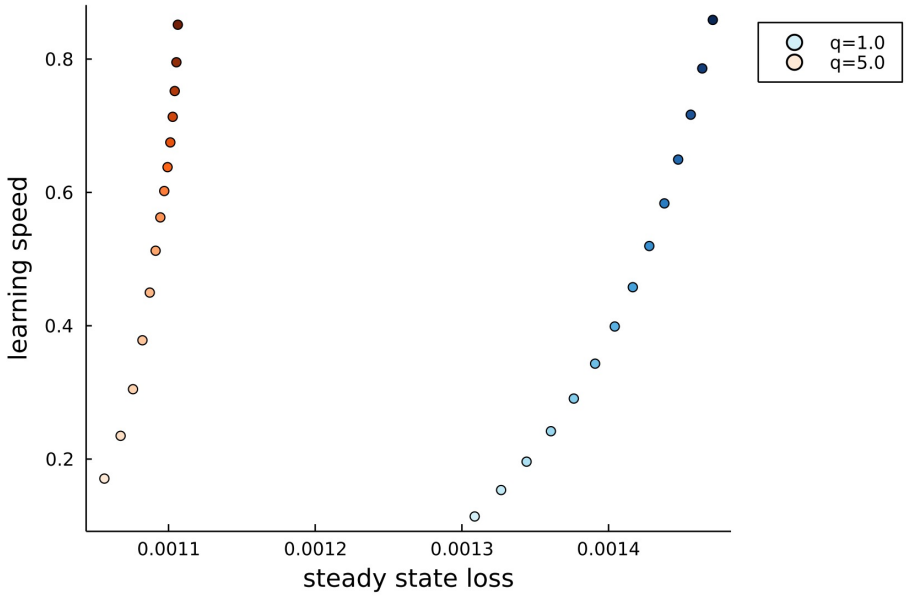


Figure 3: ss vs ls attempt 2

We see that now the steady state loss is increasing with $\gamma$ 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 converge to steady state any faster for small learning steps but we can use larger learning steps as there is not learning rule noise.

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

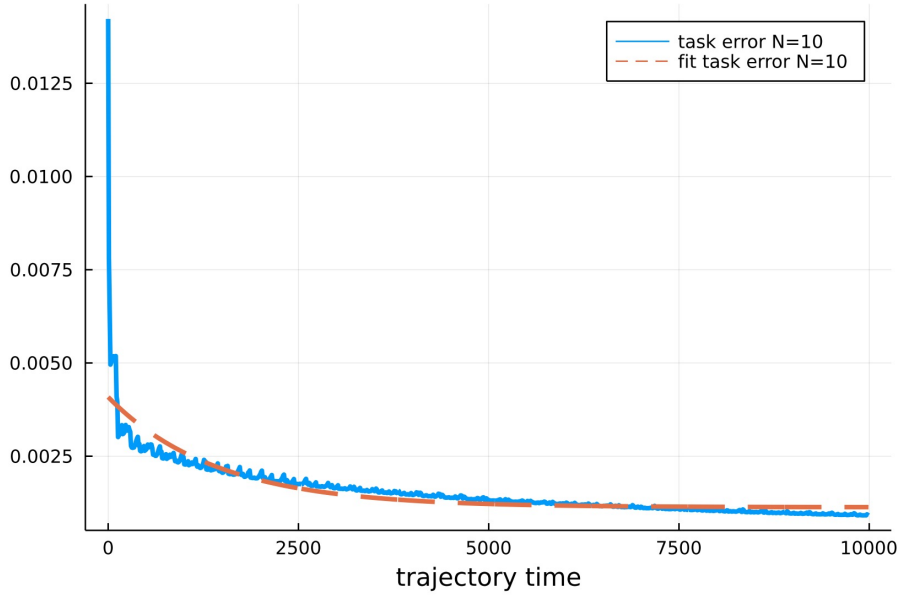


Figure 4: pre train lms long

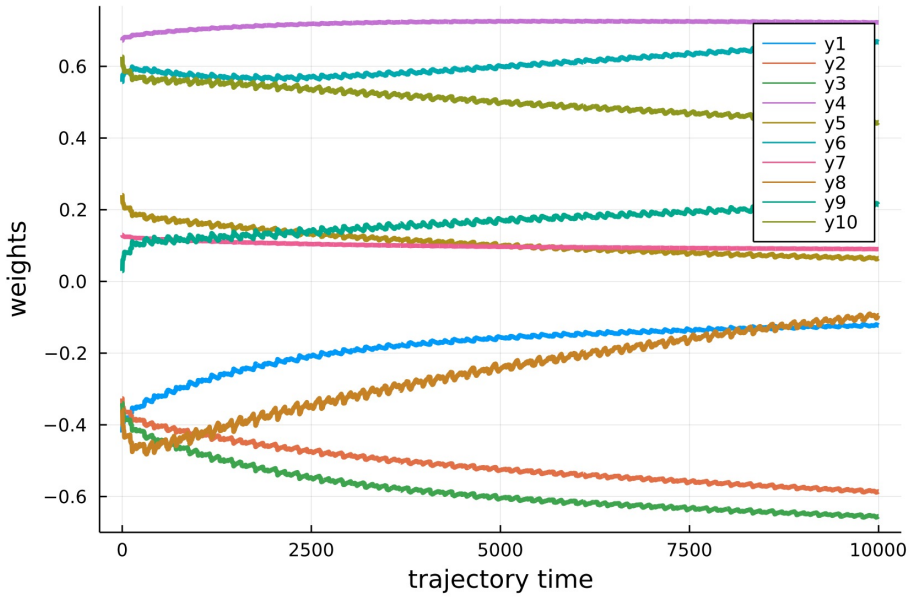


Figure 5: pre train lms long weights

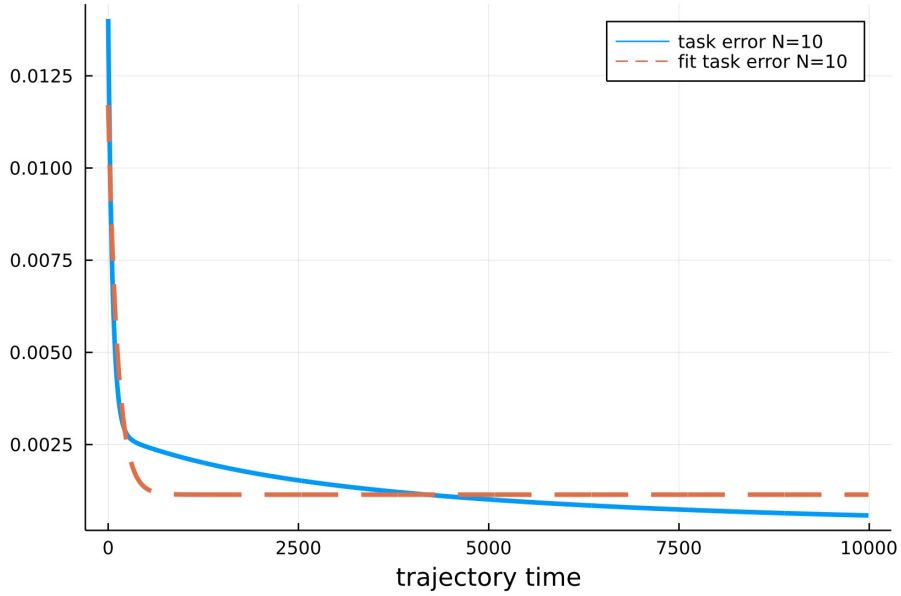


Figure 6: grad descent pre train

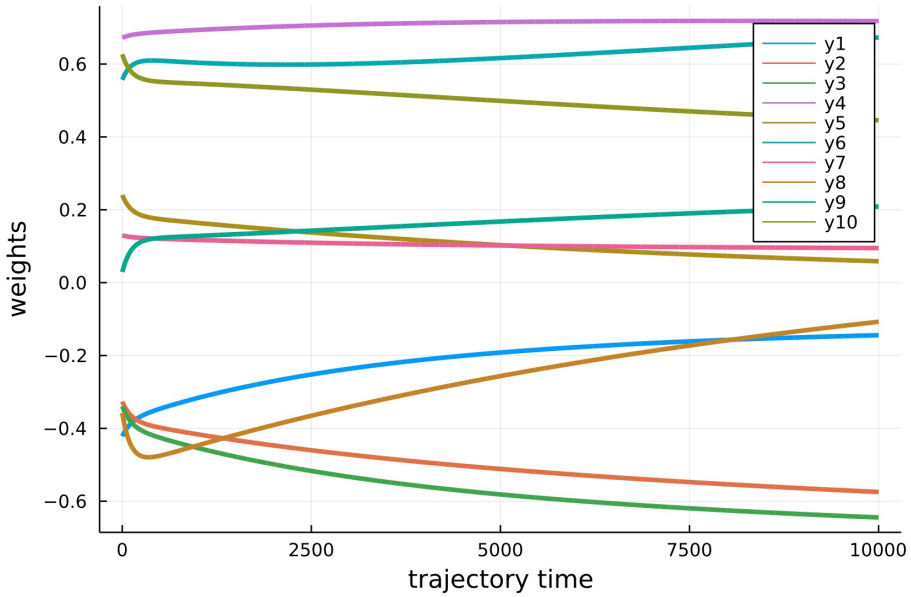


Figure 7: grad descent pre train

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.

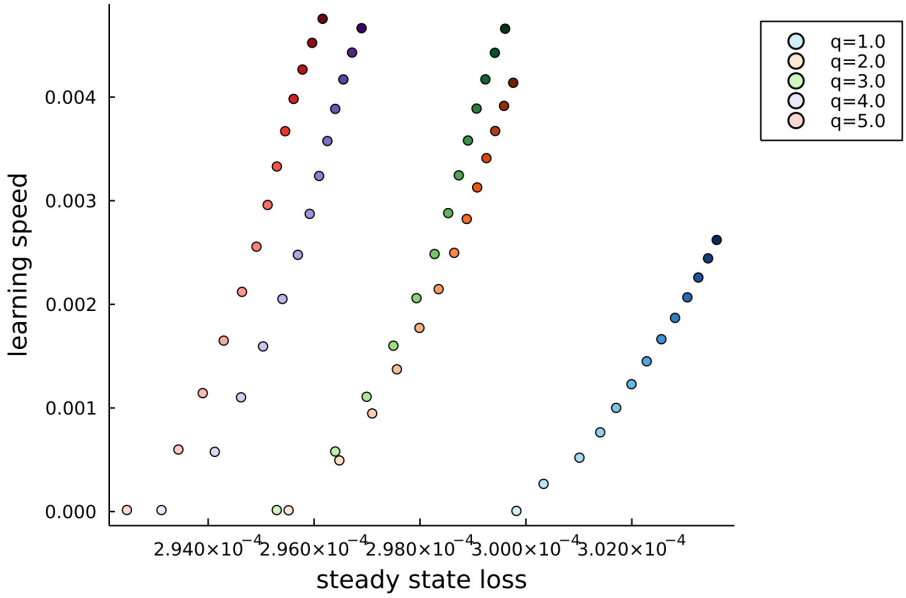


Figure 8: new tradeoff

## Observations

- We consider small learning steps for from the "optimal learning speed" where the learning starts decreasing. It is not relevant to look at the regime of maximal learning speed. This would correspond being in a regime near one shot learning.

The figure from the theoretical results in this case are shown in fig 9.

- 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).

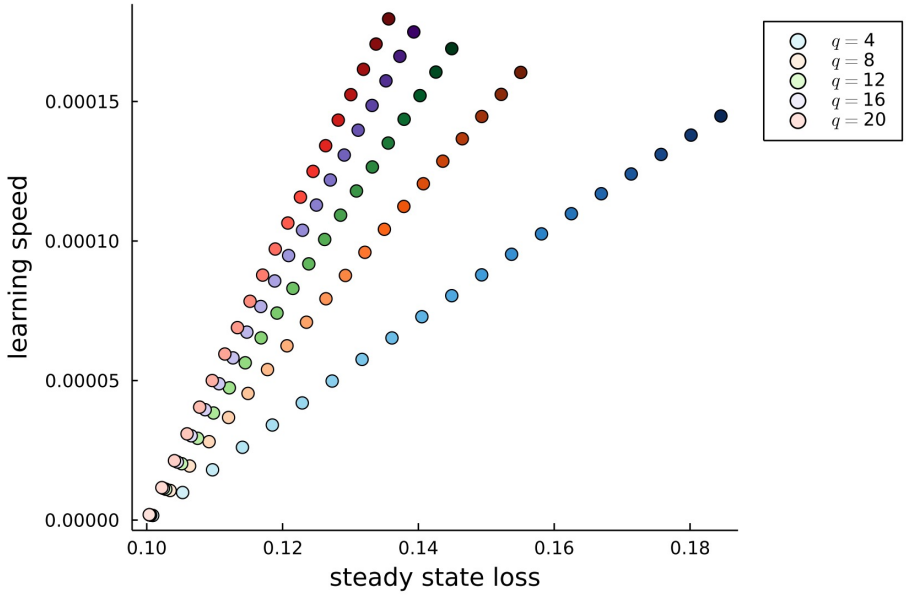


Figure 9: theory small gammas

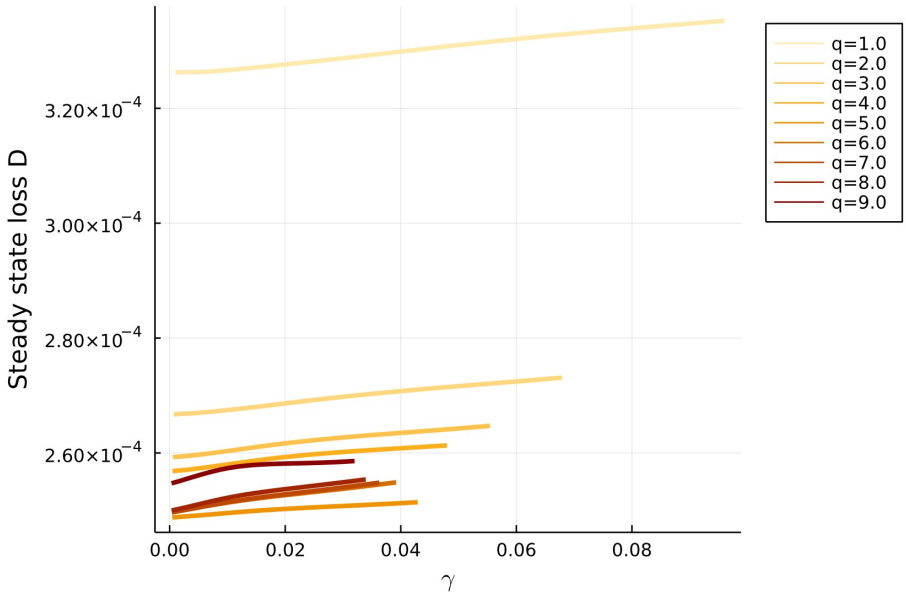


Figure 10: ss vs mu

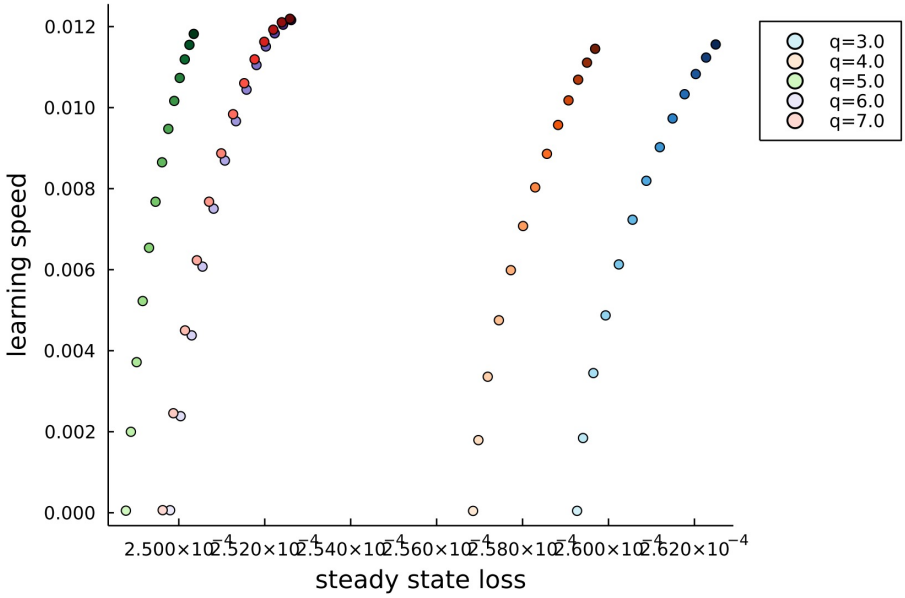


Figure 11: ss vs mu

We observe that for $q > 6$, the steady state loss increases with the expansion. 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 $3 \times 10^{-4}$ to $3.03 \times 10^{-4}$.
- 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.

## New fixes

- Need to make sure that the reference trajectory is starting 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 plan will always output zero.
- The issue with optimal learning speed being the same for all network sizes. We don't expect that to be the same. Especially if we are looking at the learning speed over the first learning step alone.
- The issue with steady state loss calculation and all the different network sizes having different losses after pre-training:
  - fix by stop pre-training in the large network when the loss reaches the same value as the smallest network after pre-training. This means that the pre-training sets all the network sizes to the same point. When we compute the steady state loss we expect the larger networks to decrease task loss but have enough time to reach steady state.
  - set steady state loss calculation trajectory time to 10000s at least and select loss of last 10 or so to get ss value.

Tried this but didn't work. Calculating the ss loss over 10000s is not enough for the larger networks to get to steady state loss (especially for small learning step). Which means that we don't recover the result that the steady state loss increases with learning step.

For some reason the pre-training was not good enough so all the steady state losses decrease network size. Indeed, we can look at the task loss over the trajectory when computing steady state loss. We observe that the loss is decreasing (for all different learning steps)

**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

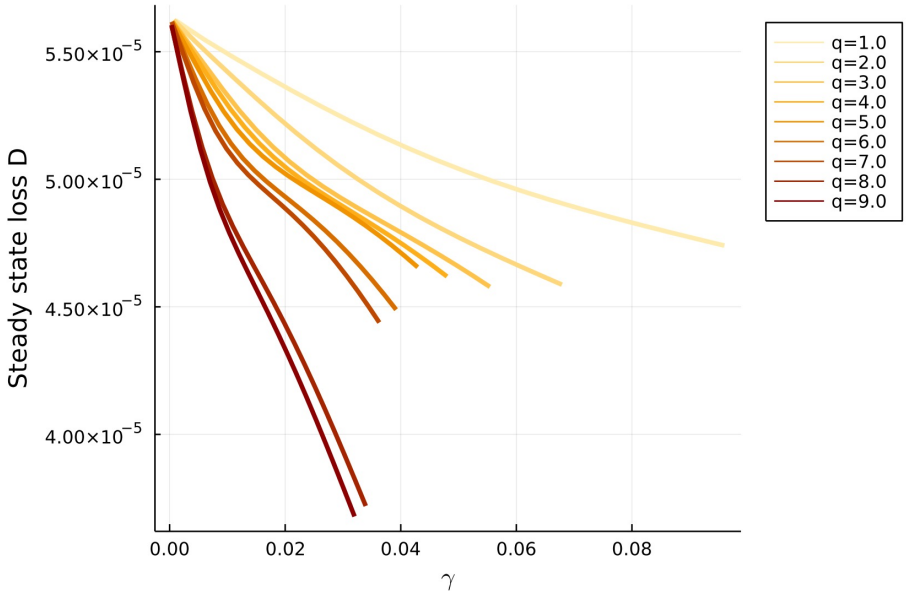


Figure 12: steady state loss vs learning step for different sized networks. when we pre-train different network sizes to have the same final loss after pre-training

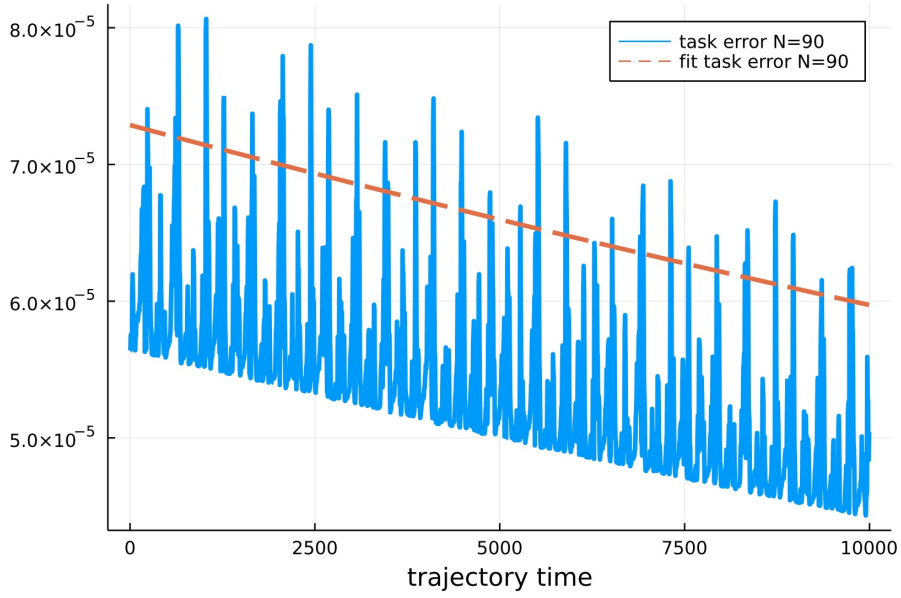


Figure 13: steady state loss vs training time for network with $q = 9$. when we pre-train different network sizes to have the same final loss after pre-training

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).

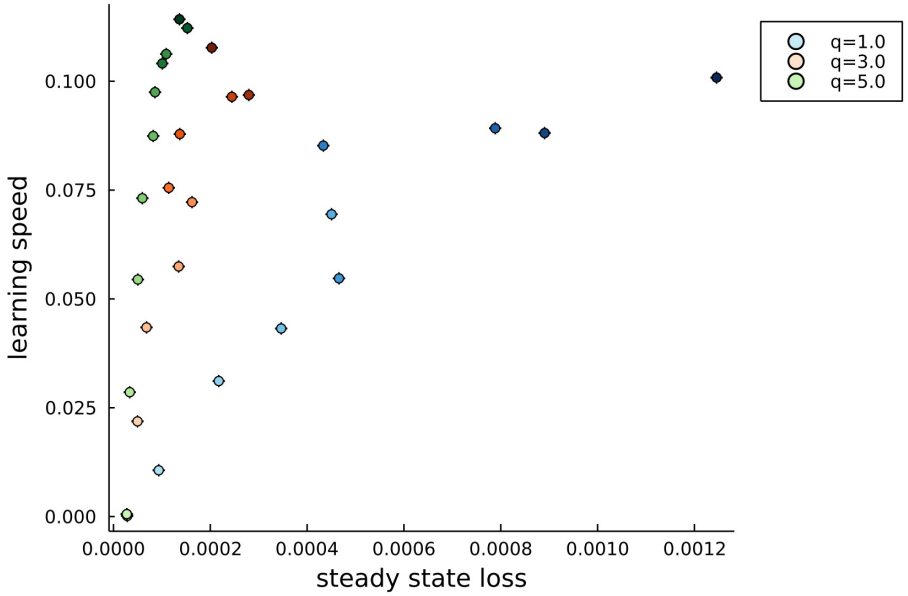


Figure 14: ss loss vs learning speed simulations with gaussian noise $SNR = 3$

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.

Adding gaussian noise to the learning rule might seem a bit artificial though. We could achieve a similar effect if we either modify the parameters of the learning rule to introduce more learning rule error or test a harder plant (for example non-linear) which would make the LMS-like learning rule perform worse.

**Increasing learning rule error in lms**

Reminder that the lms-like learning rule is defined as

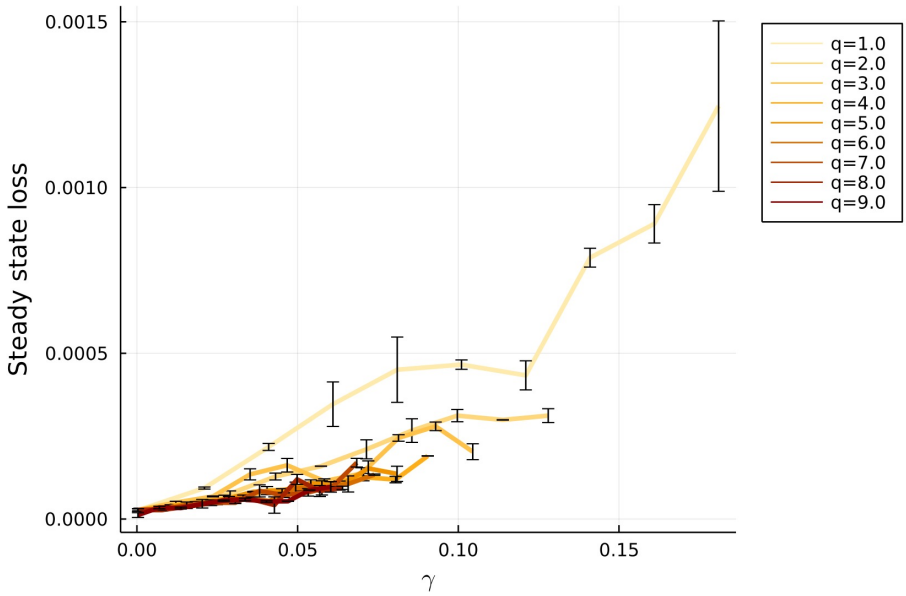


Figure 15: learning speed vs learning step simulations with gaussian noise $SNR = 3$

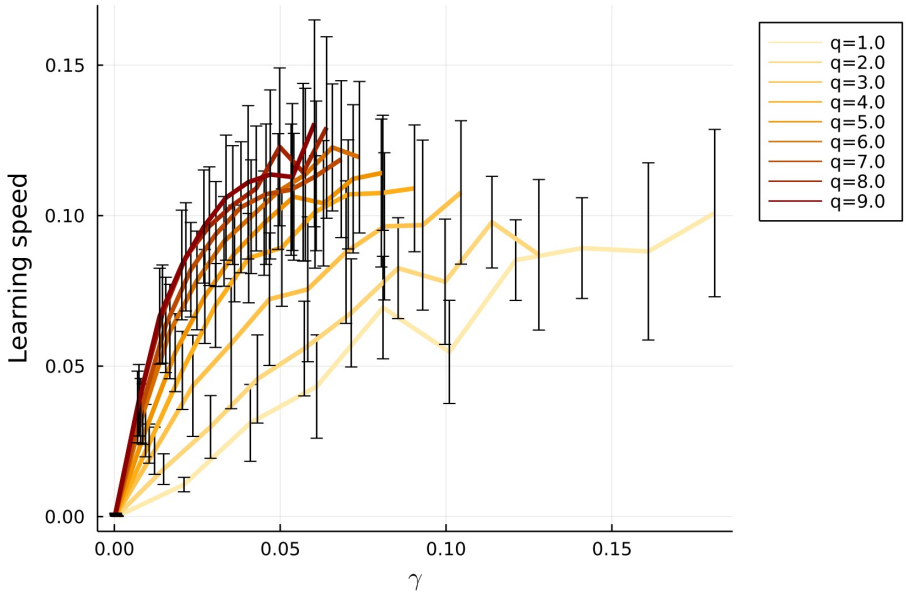


Figure 16: steady state loss vs learning step simulations with gaussian noise $SNR = 3$

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

**Changing parameters** There are some parameters in the lms-like learning rule that control how the accuracy of learning. In particular, introducing delays between the granule cell layer activity time $\Delta t_h$ and the reaction time (i.e. delay in the error information) $\Delta t_r$, usually decreases learning performance. We expect with the change in parameters that the results will be less sensistive to pretraining as the steady state loss will be higher.

We changed the parameters to $\Delta t_e = 4$, $\Delta t_r = 1$, $\Delta t_h = 0.1$. Now there is a 0.9 delay between the error information and the granule cell layer activity used for the learning rule. (Note that before we had $\Delta t_e = 1$, $\Delta t_r = 0.5$, $\Delta t_h = 0.1$.

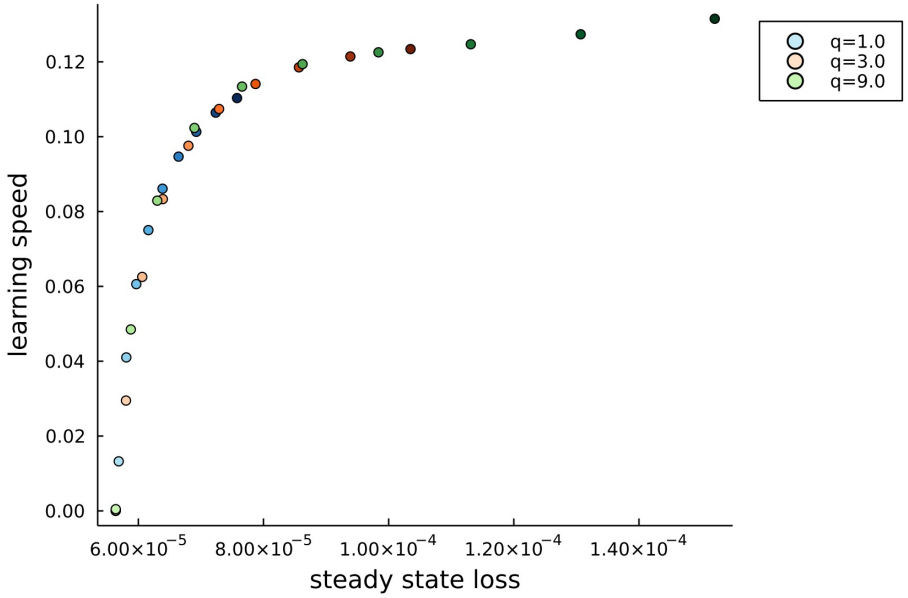


Figure 17: learning speed vs steady state loss simulations with change of lms parameters

In this case, as expected the results seems less sensitive to pretraining, and all the network sizes increase steady state loss as the learning step increases. However, there is no much change in the trade-off with the learning expansion. Indeed, the change in parameters makes learning worse but not by adding learning rule error. The error that is introduced is not be uncorrelated with the task loss hence the projection onto the hessian will not be uncorrelated and the network expansion won't be that beneficial.

**Adding vector noise** We introduce some noise at the granule cell layer activity. This would correspond to having some uncertainty in the direction of weight update for learning.

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

where $\epsilon$ is a random vector (elements drawn form random normal distribution)

Note that in this case, the noise will be multiplied by $\gamma$ and by the trajectory error integral $\frac{1}{\Delta t_e - \Delta t_r} \int_{t-\Delta t_e}^{t-\Delta t_r} (y(t') - r(t'))dt'$

hence the "signal to noise ratio" is

$$\sigma = \frac{1}{\eta \left[ \frac{1}{\Delta t_e - \Delta t_r} \int_{t-\Delta t_e}^{t-\Delta t_r} (y(t') - r(t'))dt' \right]}$$

which decreases close to steady state (when the trajectory error is small). The direction of this error should be uncorrelated with the task loss as it is determined by the random vector $\epsilon$ but the magnitude of the learning rule error is proportional to the amount of trajectory error and the learning step. Near steady state, the trajectory error will be smaller (by definition of steady state) than at the initial stages of learning. So the error will also be smaller. This kind of noise hence doesn't fall in the category of learning rule error we consider in the paper. It won't have a big effect on the steady state loss or on the benefit of the network expansion on the steady state loss.

**Adding scalar noise** Second we consider adding scalar noise to lms weight update. This corresponds to uncertainty to the trajectory error average.

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

In this case the "signal to noise ratio" is

$$\sigma = \frac{1}{\eta|\epsilon|||\mathbf{h}(t - \Delta t_h)||}$$

This quantity will be approximatively equal during learning. However, it will decrease with the network expansion as $||\mathbf{h}(t - \Delta t_h)||$ increases.

for $\sigma$ to be constant with network expansion as well, we need $|\epsilon|$ to decrease with network expansion as $\frac{1}{\sqrt{q}}$ we do so by changing the variance of the gaussian distribution from which we draw $\epsilon$, $\epsilon \sim \mathcal{N}(0, 1/\sqrt{N})$.

**Combination**

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

we have scalar noise in the error and vector noise $\epsilon^s \sim \mathcal{N}(0, 1/\sqrt{N})$ and $\epsilon^v \sim \mathcal{N}(0, 1)$ this is equivalent to having some error in the trajectory error and in the computation of the direction of weight change. It seems kind of arbitrary for the scalar error to have smaller variance as the network expansion increases. But it is the only way to make sure the signal to noise ratio is constant with network expansion.

$$\sigma = \frac{1}{\eta|\epsilon^s|||\mathbf{h}(t - \Delta t_h)|| + \eta^2 + \eta \left( \frac{1}{\Delta t_e - \Delta t_r} \int_{t-\Delta t_e}^{t-\Delta t_r} (y(t') - r(t'))dt' + \eta\epsilon^s \right)}$$

which at steady state is approximatively

$$\sigma \approx \frac{1}{\eta|\epsilon^s|||\mathbf{h}(t - \Delta t_h)|| + \eta^2}$$

## Explanation

We define learning performance with respect to a task loss $L$. The **learning speed** is a measure of how fast the task loss decreases *initially* during training. Whereas **steady state loss** is the loss reached when the system *stops* learning (i.e. the weight change is small on expectation over some training time).

In theory, at steady state, if there is no online error, or noise, the loss should be constant. However, because we are doing online learning (for example training with LMS-like learning rule), there is an online learning error that forces the weights and hence the task loss to keep changing, throughout training. We expect though that there will be period reached when, on average, the change in task loss will be zero. We can then define steady state loss as the average loss during a period of average change in task loss close to zero.

In practice, it is hard to define the length of period over which we want to average to get zero change in task loss.