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:

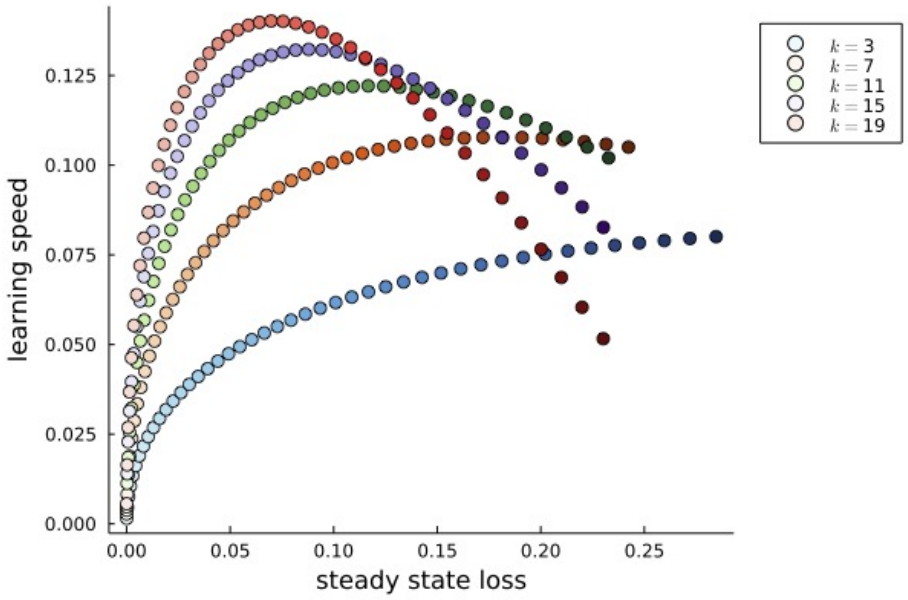

Figure 1: theoretical plot

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.

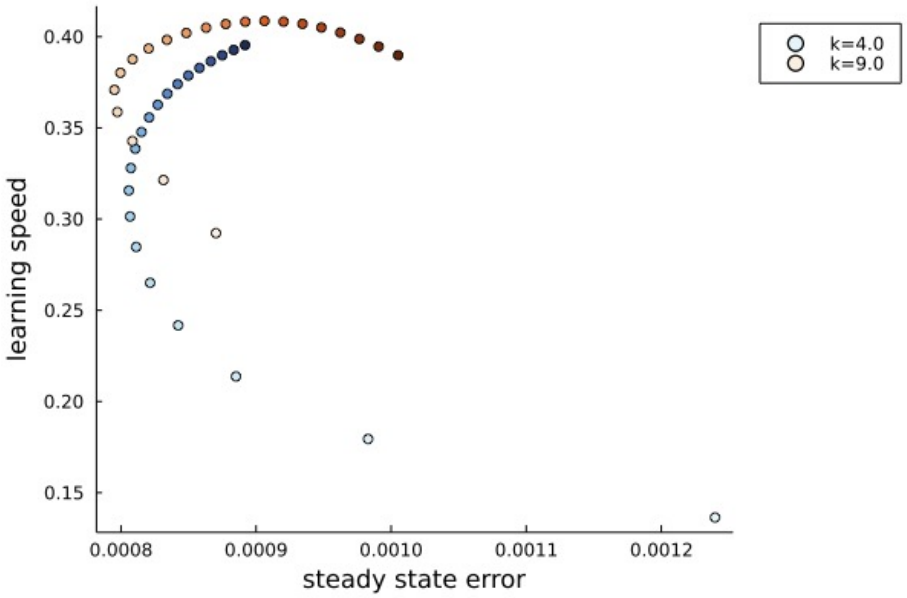

Figure 2: simulation plot

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.

Attempts

Attempt 1: - 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.

I redid the simulations, now I pretrain each network with LMS over 400s with a medium sized γ and take the final weights to be the *best weights* $\mathbf{w}^*$. Then for each different learning step, we initialise the weights at $\mathbf{w}^*$, train with lms with γ for 400s and take the average of the task loss over the last 100s as the steady state loss.

Unfortunately, the results in the figure above didn't change.

We look at the task loss when training to compute the steady state loss. We see that the task loss is still varying greatly and we can't really asses if we have actually reached steady state.

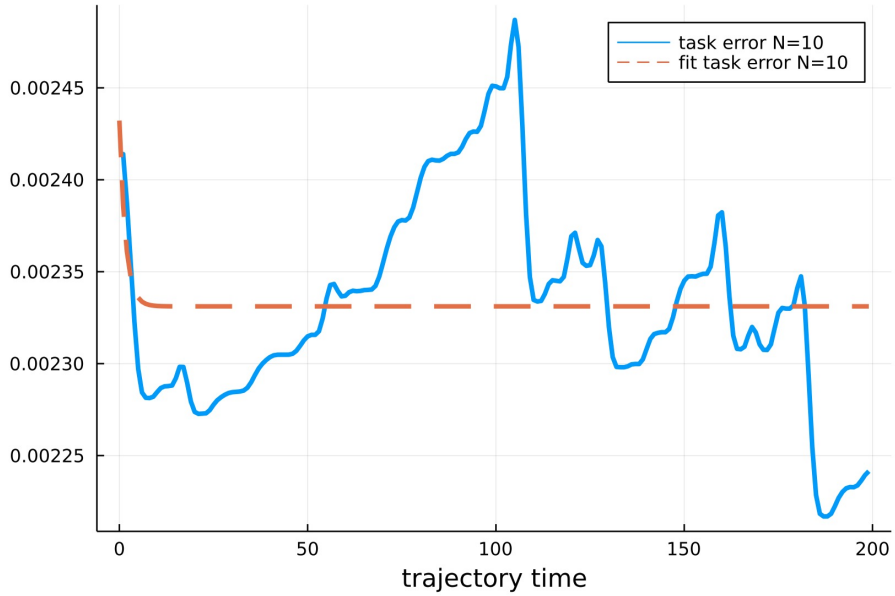

Figure 3: pre train short

Problems: - the weights $\mathbf{w}^*$ we have chosen are not close enough to local minimum - the training time is too short to see if the system has reached steady state - the reference trajectory is too stereotypical

Attempt 2: - 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.

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

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.

- For some initialisations, (i.e. network weights and types of reference trajec-

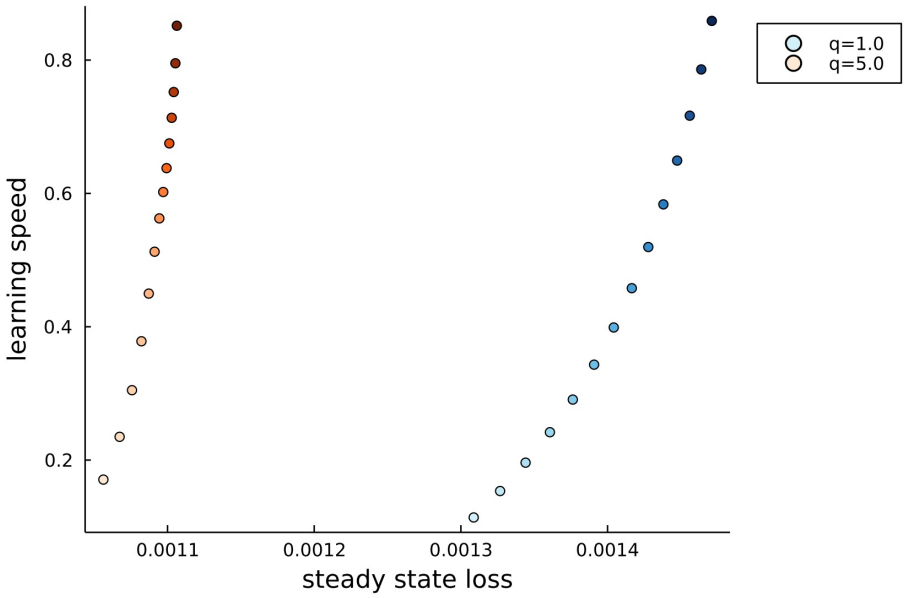

Figure 4: ss vs ls attempt 2

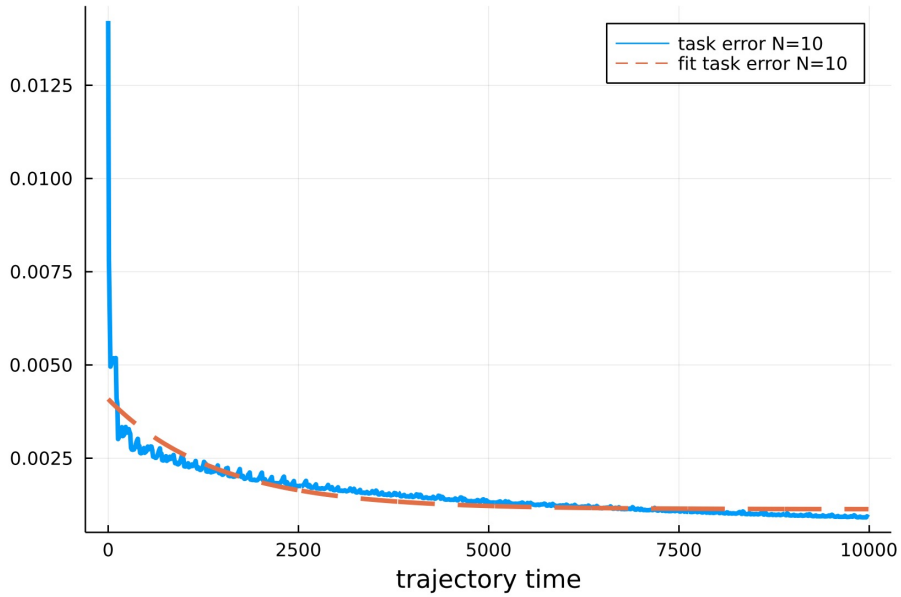

Figure 5: pre train lms long

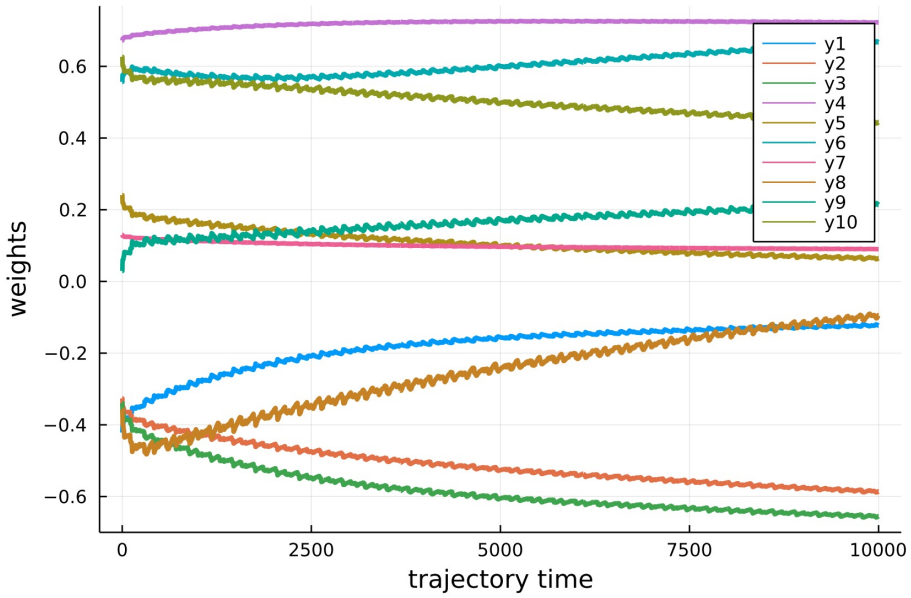

Figure 6: pre train lms long weights

tory) after 10000s of pre-training with LMS we can still not reach steady state. But with other initialisations, we reach steady state much faster. The main difference with the initialisations is if at initial points of learning the actual trajectory y heads in the same direction (positive or negative) as the reference trajectory r. If they head in the same direction, the training will be much slower than if they don't. It seems that having a large discrepancy at the beginning of the movement leads to a large weight in a "good" direction. We could filter this out by selecting just one type of the trajectories or training for even longer when heading in the same direction. Or by training with gradient descent which wouldn't have this problem.

Good initialisation

- Task loss during pre-training
- Weights during pre-training
- Initial parts of the trajectory

Bad initialisation

- Task loss during pretraining
- Weights during pre training

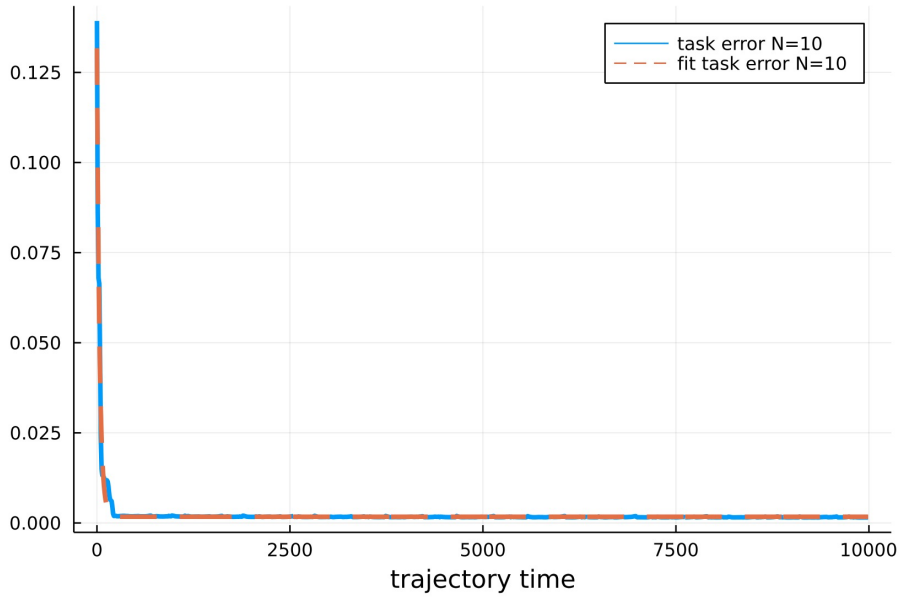

Figure 7: good init

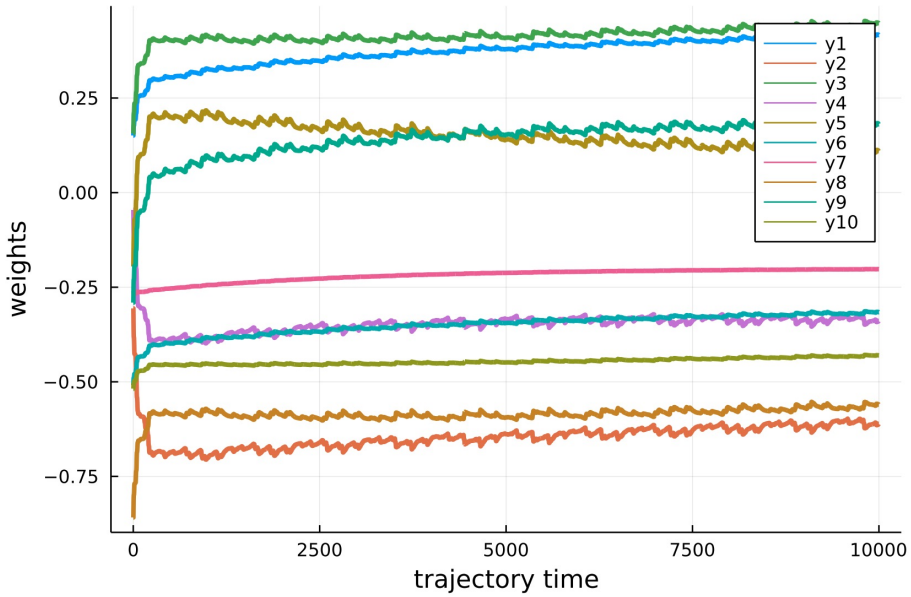

Figure 8: good init w

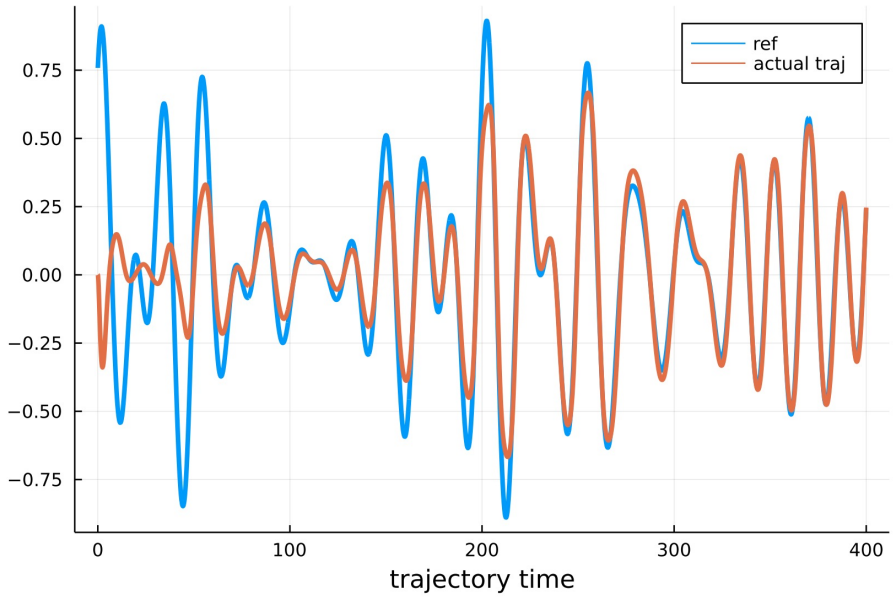

Figure 9: good init traj

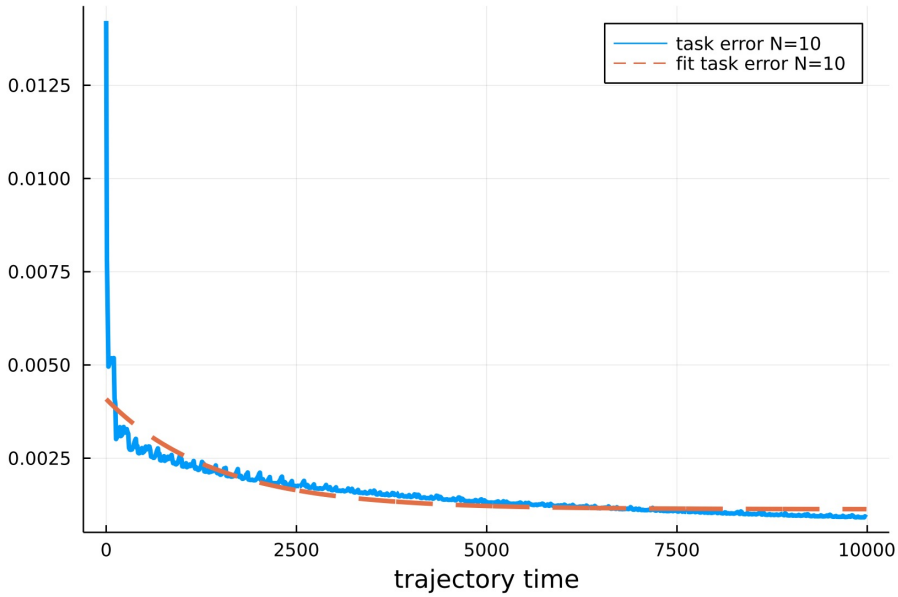

Figure 10: bad init

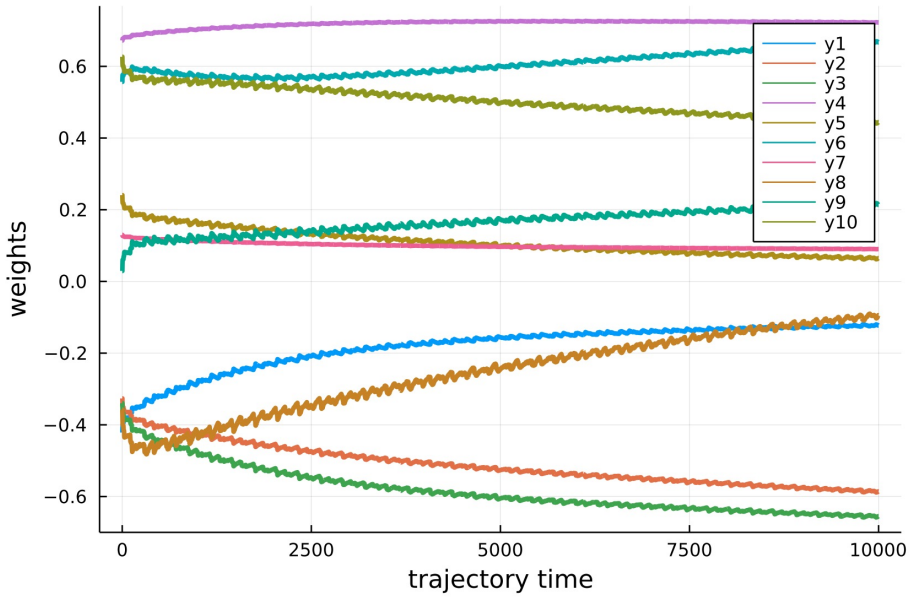

Figure 11: bad init w

- Beginning of the trajectory

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 large net size) as it needs to compute the gradient over the whole system for a very long time.

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 can 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.

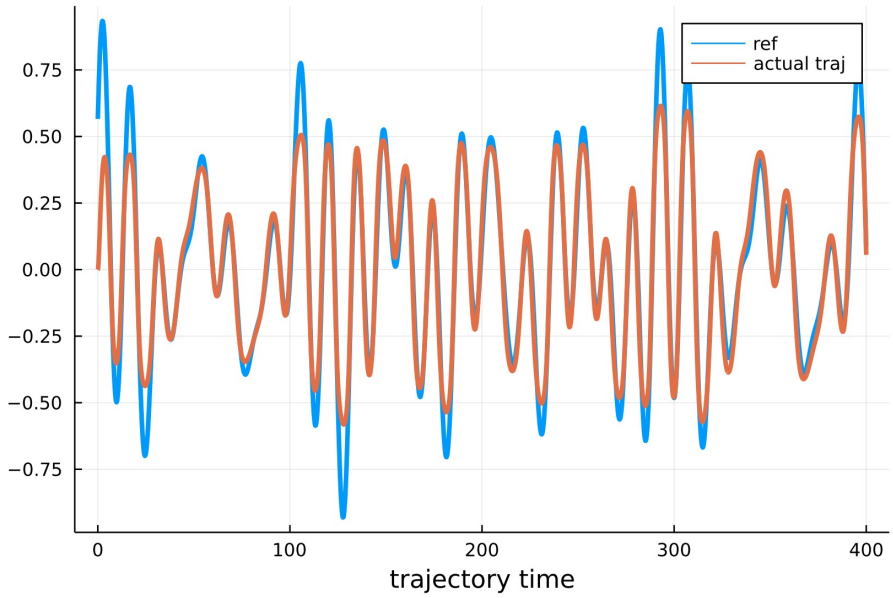

Figure 12: bad init traj

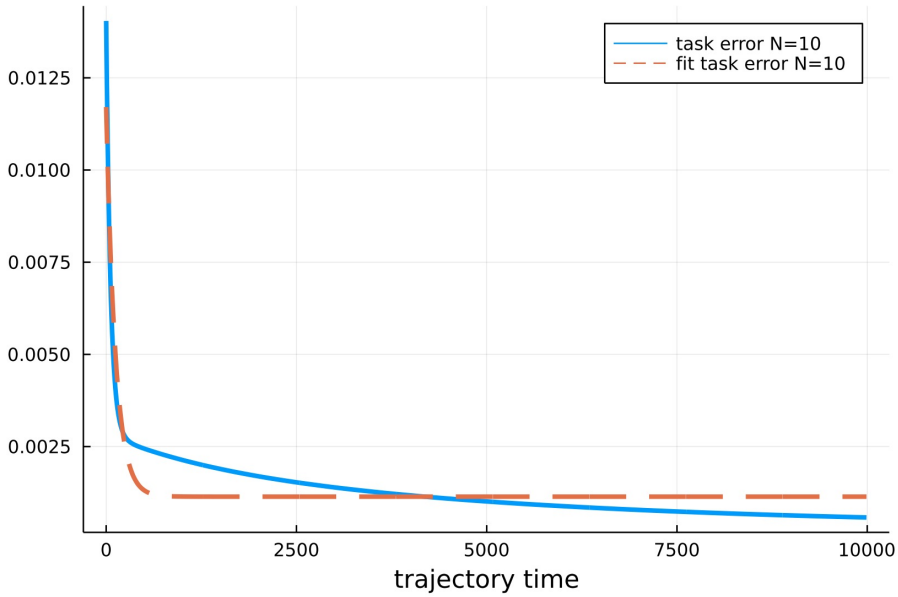

Figure 13: grad descent pre train

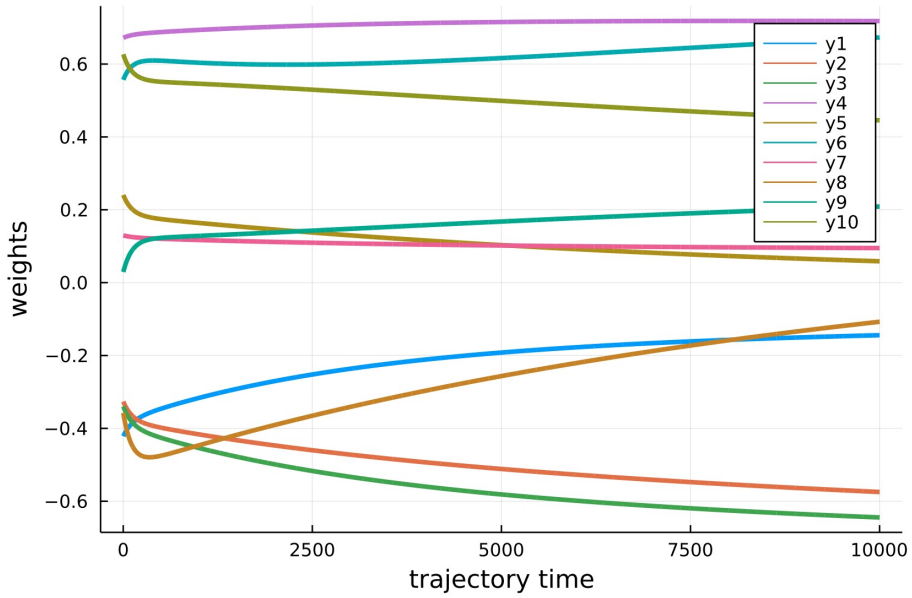

Figure 14: grad descent pre train

New Results

- 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. From one simulation where the pre-training worked we get the following
- **Issue 1** It seems that the optimal learning speed for each network size is equal.

From our model, we can get this case when there is no learning rule error $\eta^{lr} = 0$ and the spread of the hessian scales linearly $\rho = 1$. Then, it is possible that lms like learning rule in this case has essentially no learning rule error and the scaling of the hessian projection is linear with network expansion.

The theoretical analysis we carry out is static, in the sense that we determine learning speed by looking at the change in task loss for one weight update at initial stages of learning. In this paradigm, the optimal learning step is the learning step that would have to be taken at that point in weight space to reduce the task loss the most in a single step. This would be equivalent to one shot learning.

In the simulations, however, we are measuring a dynamic learning speed, i.e. the change in task loss over multiple weight updates at the initial points of learning. The optimal learning step for dynamic learning speed is the combination of the

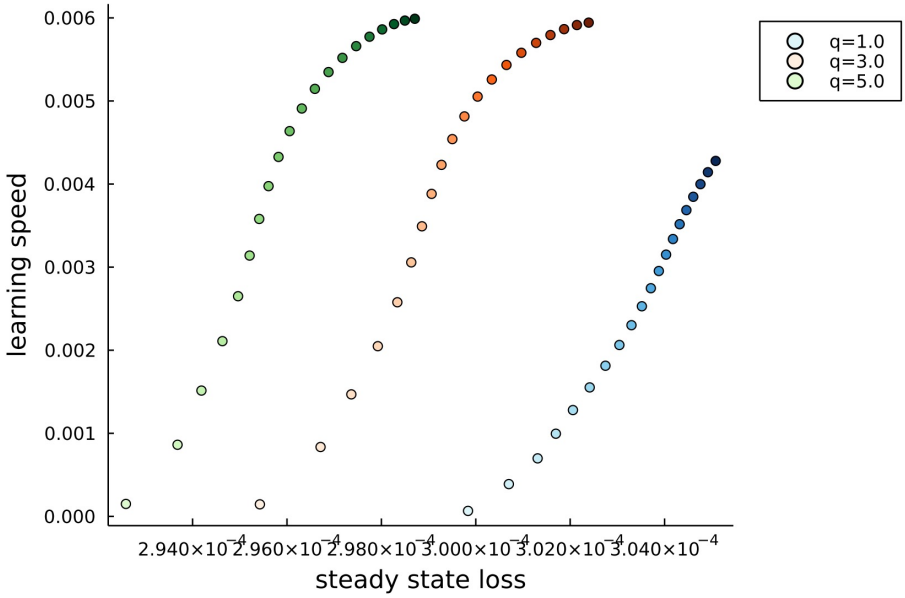

Figure 15: new tradeoff

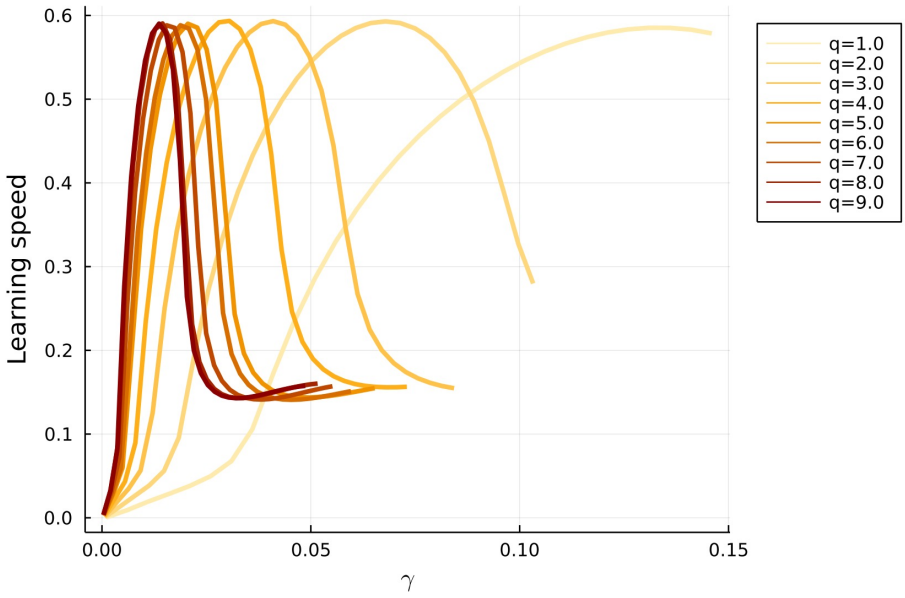

Figure 16: new tradeoff

optimal learning step for each one of the weight updates we are considering. Indeed, after each weight update, the gradient and curvature is different. So there will be discrepancies between the theoretical analysis and the simulations in the regime close to "optimal learning step" (i.e. for larger learning steps).

This is okay because in any case the biologically relevant regime is with smaller learning step. It is not relevant to look at the regime of maximal learning speed. This would correspond being in a regime near one shot learning.

This problem wouldn't be solved by just doing more simulation runs as the inconsistencies are within each simulation run and there is a bias towards larger network sizes not being pre-trained well enough.

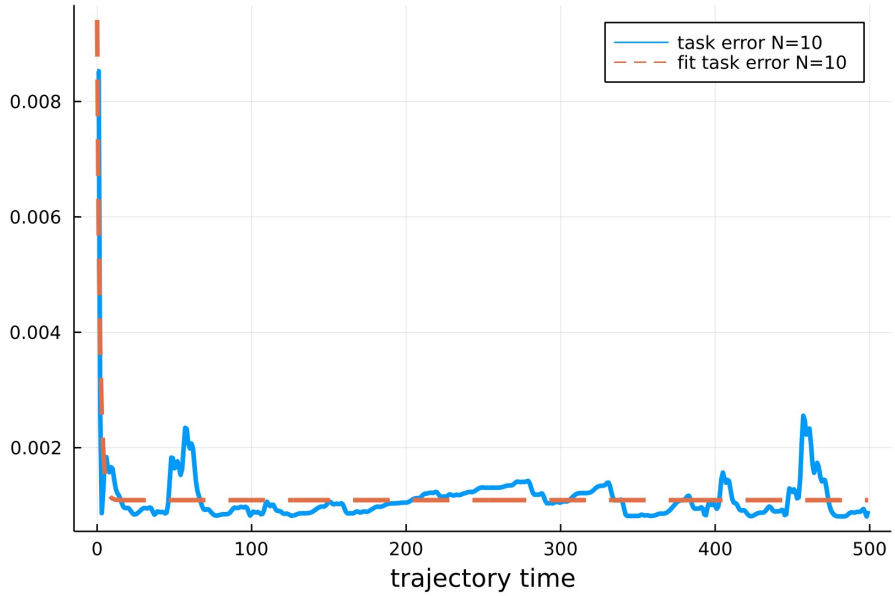

Figure 17: task loss

Solution only look at relatively small learning steps away from "optimal learning speed".

- **Issue 2** 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. This is unexpected.

This could be a problem with our theoretical results, but it is more likely that it

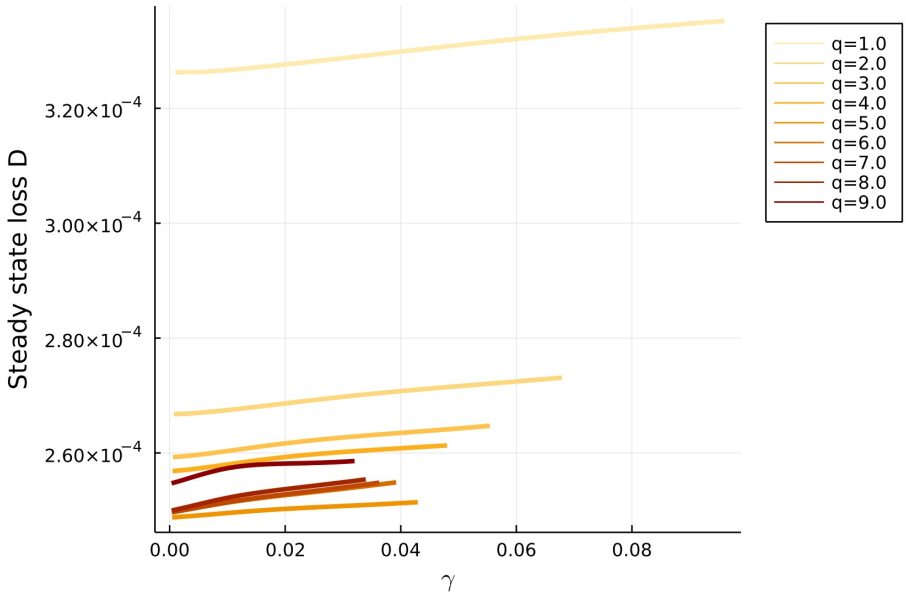

Figure 18: ss vs mu

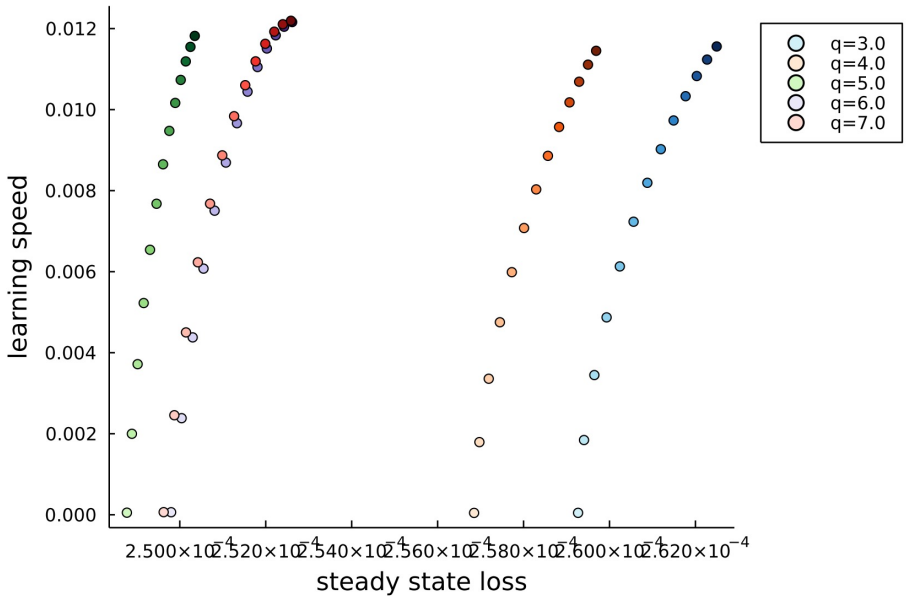

Figure 19: ss vs mu

is just a problem with the pre-training. Indeed the inconsistencies are different for different simulation seeds.

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.

Observations

- Should the task loss be computed over a very long trajectory (for example 10k sec)? Now that we are explicitly choosing the length of training for pre-training, learning speed calculation and steady state loss computation, the question arises of how long should be the reference trajectory over which we are computing the task loss. In the analysis, we take the task loss to be the mse loss over the whole reference trajectory with the only assumption that it is long enough to reach some form of steady state.

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.

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.

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.