# Developmental Interpretability in Toy Models of Superposition

Jessica Nunez

May 31, 2024

# Introduction

In the realm of machine learning, understanding the developmental trajectories of neural networks is crucial for humans to deem a model 100% safe, or at least as close as one can get to. It is extremely complex, however. Few success stories have risen from neural network interpretability. It's so new, there isn't a go-to textbook that I could use to help me understand these ideas better; only esoteric research papers and a handful of youtube videos. I say this not to discourage, but to emphasize how low-key this subfield is and how much room it actually has for growth and impact! Moving forward, the core of this analysis draws inspiration from developmental biology, where critical periods mark significant changes in neural architecture, we aim to uncover analogous phases in the training of artificial neural networks. Specifically, we investigate a toy model of superposition (TMS), a simplified neural network that allows us to observe fundamental learning dynamics without the obfuscating intricacies of larger models.

This work builds upon the foundational paper by Chen et al. (2023), "Dynamical versus Bayesian Phase Transitions in a Toy Model of Superposition," which demonstrates that in a two-dimensional hidden space, regular k-gons serve as critical points. These geometric configurations, they argue, are pivotal in determining phase transitions in both Bayesian and Stochastic Gradient Descent (SGD) learning paradigms.

The primary objective of this study is to extend Chen's work by examining how the developmental interpretability of our TMS changes under varying input conditions. We explore three key hyperparameters:

1. **Hidden Dimensionality**: By increasing the number of hidden dimensions from 2 to 3 and 4, we investigate how additional representational capacity affects the emergence and nature of developmental phases.

2. **Input Density**: Transitioning from sparse to dense inputs, with sparsity levels of 0.3, 0.5, and 0.7, we probe whether the richness of input data alters the model's learning trajectory.

3. **Feature Importance**: Introducing non-uniform feature weightings, we examine if the model's developmental phases are sensitive to the relative salience of input features.

Our analytical toolkit is diverse. We visualize weight snapshots to capture qualitative shifts in the model's internal representations. We compute Local Learning Coefficients (LLCs) to quantify learning speed at different stages. Most crucially, we perform covariance analysis on the model's weights, inspired by Freedman et al. (2023). By tracking the eigenvalues of the weight covariance matrix, we aim to identify discontinuities that signal developmental transitions, akin to the formation of neural circuits in biological systems.

Through this multifaceted analysis, we seek not only to characterize the developmental phases of our TMS but also to understand how these phases are modulated by the nature of the input data. Our findings have implications beyond this toy model, offering insights into the interpretability, robustness, and training strategies of more complex neural networks.

As a researcher new to this fascinating intersection of machine learning and developmental biology, I approach this study with humility, excitement, and an open mind. If I have an evident fallacy, please do kindly let me know. The journey through this analysis has been enlightening to me and I hope it will be for you as well, as it reveals the profound depths hidden even in the simplest of neural networks.

# Prima Example

In order to properly compare and contrast, we begin by showing and explaining the results for the development of the inital model, which has two hidden dimensions and eight features of equal importance.

Let's get right into it. Please refer to Anthropic's toy model of superposition at any point in time, if you're interested in reproducing it yourself.

**Algorithm 1 Prima Example** Loss and Weight Snapshots Code

```
NUM_FEATURES = 8
NUM_HIDDEN_UNITS = 2
NUM_SAMPLES = 1000
NUM_EPOCHS = 20000
INIT_KGON = 2
NUM_OBSERVATIONS = 50
STEPS = sorted(list(set(np.logspace(0, np.log10(NUM_EPOCHS),
NUM_OBSERVATIONS).astype(int))))
PLOT_STEPS = [min(STEPS, key=lambda s: abs(s-i)) for i in [0, 200, 2000,
10000, NUM_EPOCHS - 1]]
PLOT_INDICES = [STEPS.index(s) for s in PLOT_STEPS]
logs, weights = create_and_train(NUM_FEATURES,
NUM_HIDDEN_UNITS, num_samples=NUM_SAMPLES,
log_ivl=STEPS, batch_size=100, lr=0.01, num_epochs=NUM_EPOCHS,
init_kgon=INIT_KGON, init_zerobias=False, seed=1)
weights_to_plot = [weights[i]['embedding.weight'] for i in PLOT_INDICES]
losses = [logs.loc[logs['step'] == s, 'loss'].values[0] for s in STEPS]
plot_losses_and_polygons(STEPS, losses, PLOT_STEPS, weights_to_plot)
plt.show()
```

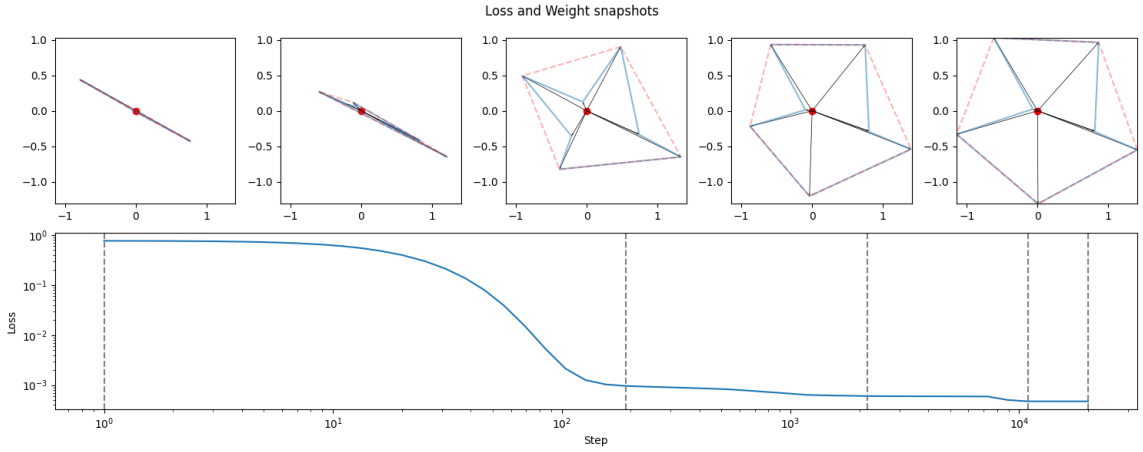



Figure 1: **Prima Example** Loss and Weight Snapshots

The image shows two types of plots that together provide insights into the training dynamics of the autoencoder TMS.

1. Loss Plot (Bottom):

This is a line graph with the x-axis showing the training steps on a logarithmic scale, and the y-axis showing the loss value, also on a logarithmic scale. The loss starts high (around 1) and decreases rapidly in the early stages of training (between steps 10 and 100). After step 100, the loss stabilizes around 10^-3, indicating that the model has largely converged. The vertical dashed lines correspond to specific steps where the model's weights (parameters) are visualized in the plots above.

2. Weight Snapshots (Top):

   There are five plots, each representing the state of the model's weights at different training steps, corresponding to the dashed lines in the loss plot. Each plot shows two weight vectors in 2D space, represented as lines from the origin (marked by a red dot). The x and y axes range from -1 to 1, hence they are normalized weights.

   Interpretation of weight snapshots:

   - First plot (leftmost): The weights form a simple line, indicating that the model is using a single dimension to represent the input features.
   - Second plot: Similar to the first, but with a slight deviation, suggesting the model is still in an early phase of learning.
   - Third plot: The weights form a "V" shape, indicating that the model has transitioned to using both dimensions to represent features. This transition corresponds to the steep drop in the loss plot.
   - Fourth and Fifth plots: The weights form a nearly perfect "L" shape (orthogonal vectors), suggesting that the model has learned to use both dimensions independently to represent different aspects of the input. This stable representation corresponds to the flat, low-loss region in the loss plot.

In summary, these plots reveal a fascinating aspect of neural network training dynamics:

1. The model starts by using a single dimension (a line) to represent all input features.

2. As training progresses, it transitions (around step 100) to using both dimensions (a V-shape), which dramatically reduces the loss.

3. Finally, it settles into using both dimensions independently (an L-shape), which allows it to represent the input features most effectively.

This transition from a line to a V to an L is a key insight in the paper on toy models of superposition. It suggests that neural networks go through distinct developmental phases, akin to critical periods in biological neural development, where the model's internal representations undergo qualitative changes. Understanding these transitions can be crucial for interpreting and improving the

robustness of more complex neural networks. If you're new to this, I highly suggest reading this article, also referenced below.

**Algorithm 2 Prima Example** Hyperparameter Sweep Results: Local Learning Coefficients

```
lambdahat_sweep_df = sweep_lambdahat_estimation_hyperparams(model,
dataset_double)
lambdahat_sweep_df
```

| | llc | llc/std | batch_size | lr | t_sgld | llc_type | loss |
|---|---|---|---|---|---|---|---|
| **0** | NaN | NaN | 1 | 0.00001 | 0 | mean | 0.062409 |
| **1** | -inf | NaN | 1 | 0.00001 | 0 | 0 | 0.000058 |
| **2** | inf | NaN | 1 | 0.00001 | 0 | 1 | 0.125561 |
| **3** | -inf | NaN | 1 | 0.00001 | 0 | 2 | 0.000058 |
| **4** | inf | NaN | 1 | 0.00001 | 0 | 3 | 0.093184 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **11995** | 3.258609 | NaN | 300 | 0.01000 | 99 | 0 | 0.109423 |
| **11996** | 3.408524 | NaN | 300 | 0.01000 | 99 | 1 | 0.112273 |
| **11997** | 6.431345 | NaN | 300 | 0.01000 | 99 | 2 | 0.169745 |
| **11998** | 4.512389 | NaN | 300 | 0.01000 | 99 | 3 | 0.133260 |
| **11999** | 2.536693 | NaN | 300 | 0.01000 | 99 | 4 | 0.095697 |

Figure 2: **Prima Example** Hyperparameter Sweep Results: Local Learning Coefficients

The table shows results from a hyperparameter sweep over different batch sizes and learning rates. For each combination, it runs multiple SGLD chains (num_chains) for a certain number of steps (num_draws).

For each SGLD step (t_sgld), it computes the LLC for each chain (llc_type), the mean LLC across chains (llc_type == 'mean'), and the corresponding loss. This allows you to see how the LLC and loss evolve over the course of SGLD sampling for different hyperparameter settings.

The presence of NaN (Not a Number) values in some columns suggests that there might be some issues with certain hyperparameter combinations or at certain steps in the SGLD process. These could be due to numerical instabilities, invalid computations (like division by zero), or other issues. An explanation for each column is written below.

1. llc (Local Learning Coefficient): This is the estimated local learning coefficient. It's a measure of how much the model is learning at a particular point in its training process. A higher value indicates faster learning.

2. llc/std (Local Learning Coefficient Standard Deviation): This is the standard deviation of the estimated local learning coefficient. It gives a sense of the uncertainty or variability in the LLC estimate.

3. batch_size: This is the number of samples processed together in one forward/backward pass of the model. In the table, it varies between 1, 10, and 300.

4. lr (Learning Rate): This is the step size at each iteration while moving toward a minimum of the loss function. In the table, it varies between 0.00001 (1e-5) and 0.01 (1e-2).

5. t_sgld (SGLD Time Step): This represents the time step in the Stochastic Gradient Langevin Dynamics (SGLD) process. SGLD is a variant of SGD that injects noise into the gradient updates to sample from the posterior distribution of model parameters.

6. llc_type: This indicates different types of LLC estimates. 'mean' refers to the average LLC across multiple chains (SGLD runs), while '0', '1', '2', etc., refer to individual chains.

7. loss: This is the value of the loss function (in this case, mean squared error) at the current state of the model. Lower values indicate better performance.

**Algorithm 3 Prima Example** Batch Size and Learning Rate Effect on Loss and the Local Learning Coefficient

```
plot_lambdahat_estimation_hyperparams(lambdahat_sweep_df)
```

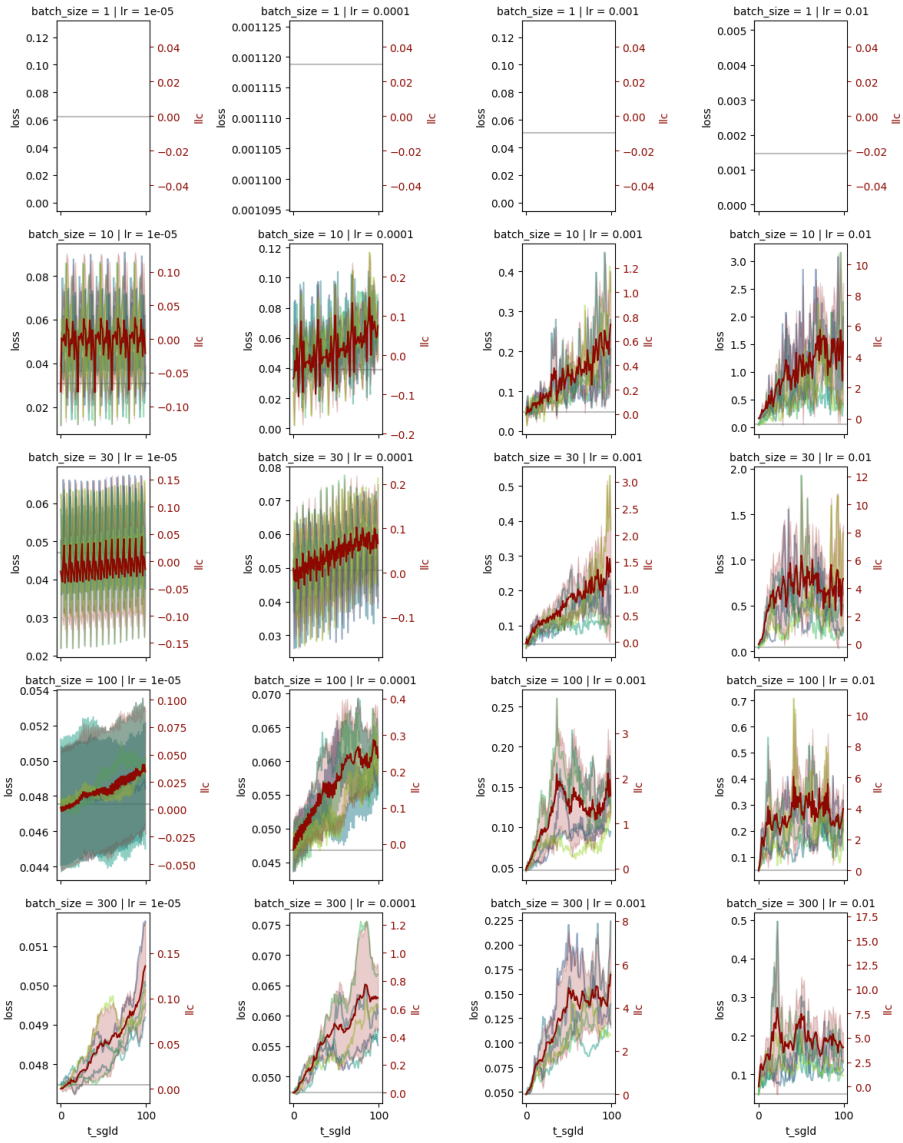



Figure 3: **Prima Example** Batch Size and Learning Rate Effect on Loss and the Local Learning Coefficient

From here on out, you will see slanted text inserts depicting explanations obtained from Timaeus's Dev Interp Library, aforementioned in the notebook reference. I think it is good to have in here for the reader to follow along with these details, at least for the prima example.

***Covariance Analysis***

*The idea behind covariance analysis is to look at the covariance between weights across a bunch of different trajectories and how this changes over time.*

*Based on literature from developmental biology (Freedman et al. 2023), we expect transitions to be associated with discontinuous increases in the maximal eigenvalues of the covariance matrix. Moreover, by looking at the associated eigenvectors, we can get a sense of how the weights are involved in these transitions, that is, "where the circuits are forming.*

***A note on tractability**: These models are small enough that we can handle the full covariance matrix, but in general we'll have to consider simplifications like looking at covariances within a specific layer. In the same vein, we will generally consider only a subset of the covariance eigenspectrum, e.g., the largest-K eigenvalues, which can be cheaply estimated using power iteration.*

***Versus SGLD estimation**: In principle, we could perform covariance analysis over the weights sampled by SGLD. Unlike SGLD-based sampling for lambdahat estimation, we'll want to include a larger thinning factor because weights at more of risk for autocorrelation. For reasons we'll flesh out in the future, we think it's probably best to do covariance across different trajectories at the same point in time. This means drawing only one sample per chain. We'll want to sample more and shorter chains.*

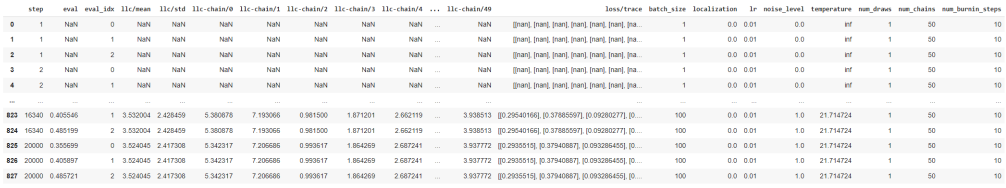
***Calibrating Covariance Estimation***

*As mentioned, we're currently interested in using covariance analysis over SGD trajectories rather than local posterior samples. By default, we recommend setting hyperparameters like sampling noise and localization strength to match the original SGD hyperparameters used during training (i.e., set them equal to zero).*

*First, let's see what difference it makes whether we use SGD or SGLD. We'll sweep over different batch sizes and interpolate between zero injected Gaussian noise (SGD) and standard SGLD (noise = 1.0). We'll use a localization strength of 0.0 for now.*

**Algorithm 4 Prima Example** Dynamic Evolution of Covariance Properties During Model Training

```
model = ToyAutoencoder(NUM_FEATURES, NUM_HIDDEN_UNITS,
final_bias=True)
covariance_estimation_sweep_df =
sweep_covariance_estimation_hyperparams(
model, dataset_double, STEPS, weights,
grid={"batch_size": [1, 10, 100], "noise_level": [0., 1.]}
)
covariance_estimation_sweep_df
```



| | step | eval | eval_idx | llc/mean | llc/std | llc-chain/0 | llc-chain/1 | llc-chain/2 | llc-chain/3 | llc-chain/4 | ... | llc-chain/49 | loss/trace | batch_size | localization | lr | noise_level | temperature | num_draws | num_chains | num_burnin_steps |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | NaN | 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 1 | 1 | NaN | 1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 2 | 1 | NaN | 2 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 3 | 2 | NaN | 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 4 | 2 | NaN | 1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 823 | 16340 | 0.405546 | 1 | 3.532004 | 2.428459 | 5.380878 | 7.193066 | 0.981500 | 1.871201 | 2.662119 | ... | 3.938513 | [[0.29540166], [0.37885597], [0.09280277], [0.... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 824 | 16340 | 0.485199 | 2 | 3.532004 | 2.428459 | 5.380878 | 7.193066 | 0.981500 | 1.871201 | 2.662119 | ... | 3.938513 | [[0.29540166], [0.37885597], [0.09280277], [0.... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 825 | 20000 | 0.355699 | 0 | 3.524045 | 2.417308 | 5.342317 | 7.206686 | 0.993617 | 1.864269 | 2.687241 | ... | 3.937772 | [[0.2935515], [0.37940887], [0.09328645 5], [0.... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 826 | 20000 | 0.405897 | 1 | 3.524045 | 2.417308 | 5.342317 | 7.206686 | 0.993617 | 1.864269 | 2.687241 | ... | 3.937772 | [[0.2935515], [0.37940887], [0.09328645 5], [0.... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 827 | 20000 | 0.485721 | 2 | 3.524045 | 2.417308 | 5.342317 | 7.206686 | 0.993617 | 1.864269 | 2.687241 | ... | 3.937772 | [[0.2935515], [0.37940887], [0.09328645 5], [0.... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |

Figure 4: **Prima Example** Dynamic Evolution of Covariance Properties During Model Training

This table provides a detailed look at how various aspects of the model's weight covariance matrix evolve during training. Each row represents a specific point in the training process, identified by the 'step' column. The table is rich with information about the covariance structure, learning dynamics, and the impact of hyperparameters.

Here's a breakdown of the columns:

1. step: The training iteration.
2. eval, eval_idx: The eigenvalue of the covariance matrix and its index.
3. llc/mean, llc/std: The mean and standard deviation of the Local Learning Coefficient (LLC) across all chains.
4. llc-chain/0 to llc-chain/49: LLC values for individual SGLD chains (up to 50 chains). These give insight into the variability of learning across different stochastic trajectories.
5. loss/trace: The mean loss across all chains at this step, providing a measure of model performance.
6. Hyperparameters:
    - batch_size: Size of data batches (1, 10, or 100).
    - localization: A parameter controlling the SGLD dynamics (set to 0.0).
    - lr: Learning rate (0.01).
    - noise_level: Amount of noise added in SGLD.
    - temperature: Parameter for the SGLD sampling distribution.
    - num_draws: Number of SGLD draws per chain.
    - num_chains: Number of parallel SGLD chains (depends on input, up to 50).
    - num_burnin_steps: Number of SGLD steps before sampling (10).

Key Insights:

1. Eigenvalue Evolution: The largest eigenvalues (eval) and their indices (eval_idx) show how the dominant directions in the weight space change. A transition from one dominant eigenvalue to multiple could indicate a phase transition.

2. Learning Dynamics: The LLC values (llc-chain/*) show how fast the model is learning at each step. Changes in these values, especially increases, could indicate critical learning periods.

3. Hyperparameter Impact: Varying batch_size and noise_level allows us to see how these affect the covariance structure and learning dynamics. For example, higher noise might lead to more variability in LLC values across chains.

4. NaN Values: The presence of NaN (Not a Number) values, especially in early steps, suggests numerical instabilities or invalid computations that could be related to the model's initialization or early learning dynamics.

This table is a helpful tool for developmental interpretability. It allows us to pinpoint when and how the model's internal representations change, how learning speed varies over time, and how hyperparameters influence these dynamics. Such insights can guide model design, training strategies, and our understanding of how neural networks "develop."

*Let's plot the results using a seaborn FacetGrid. Each subgraph shows the values of the three largest covariance eigenvalues over training time (with indices in increasing order). Each subgraph corresponds to a unique combination of batch size and noise level. The left column represents SGD trajectories. The right column represents SGLD trajectories (without localization).*

*We see that indeed the SGLD-based covariance analysis seems to miss the last transition (while this shows up as a bump in the SGD-based analysis).*

**Algorithm 5 Prima Example** Impact of Batch Size and Noise on Covariance Eigenvalues

```
def plot_covariance_estimation_hyperparam_sweep(observations_df,
y="eval", row="batch_size", col="noise_level", hue="eval_idx"):
fig = plt.figure(figsize=(15, 6))
g = sns.FacetGrid(observations_df, col=col, row=row, hue=hue,
palette="viridis", sharey=False)
g.map_dataframe(sns.lineplot, x="step", y='eval')
g.add_legend()
g.set(xscale="log", yscale="log")
# g.set(xscale="log", yscale="linear")
plt.suptitle("Covariance estimation hyperparameter sweep")
g.fig.tight_layout()
plt.show()
covariance_estimation_sweep_df =
covariance_estimation_sweep_df.dropna()
plot_covariance_estimation_hyperparam_sweep(covariance_estimation_sweep_df)
```

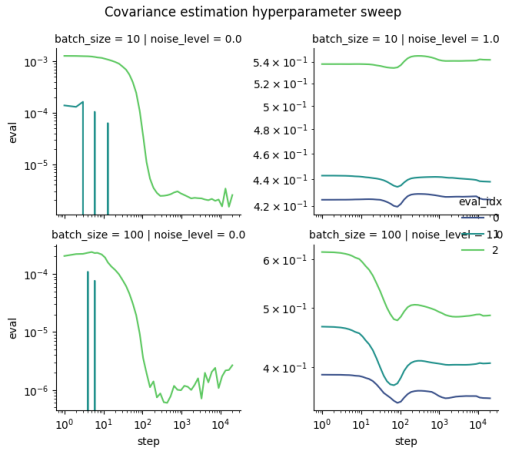



Figure 5: **Prima Example** Impact of Batch Size and Noise on Covariance Eigenvalues

Observe that this figure shows how two hyperparameters, batch size and noise level, affect the evolution of the covariance eigenvalues.

To understand this better:

1. There are four subplots, each a combination of batch size (10 or 100) and noise level (0.0 or 1.0).

2. The x-axis is the "step" on a logarithmic scale, representing the model's training progress.

3. The y-axis is the eigenvalue magnitude.

4. Lines represent different eigenvalues (eval_idx 0, 1, 2).

Insights:

- Batch Size (10 vs. 100): Larger batch sizes seem to lead to smoother, more stable eigenvalue estimates. This makes sense because larger batches provide more information for each covariance estimate.

- Noise Level (0.0 vs. 1.0):

  With no noise (0.0), the eigenvalues are more distinct, suggesting the model is using dimensions differently. With noise (1.0), the eigenvalues are closer in magnitude. This could mean the noise is "blurring" the distinctions between dimensions, or it's causing the model to use dimensions more uniformly.

- Both: The impact of noise seems more pronounced with smaller batch sizes, likely because the noise dominates the signal more easily in smaller samples.

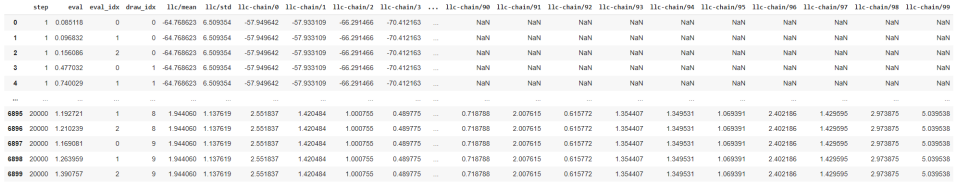
***Calibrating Burn-in Period by Timaeus***

*We'll want to use a burn-in period to avoid including the initial transient in our analysis. We'll want to use a burn-in period that's long enough to ensure that the weights have diverged from their initial values (and thus that there is any meaningful covariance to analyze).*

*To avoid having to repeat the covariance analysis for a bunch of different burn-in periods, we'll follow an online approach where we compute a novel covariance estimate for increasing numbers of sampling time steps.*

**Algorithm 6 Prima Example** Online Covariance Evolution Under Varying SGLD Sampling Parameters

```
model = ToyAutoencoder(NUM_FEATURES, NUM_HIDDEN_UNITS,
final_bias=True)
online_covariance_estimation_sweep_df =
sweep_online_covariance_estimation_hyperparams(
model, dataset_double, STEPS, weights,
loader_kwargs={"batch_size": 10},
sampling_kwargs={"num_burnin_steps": 0, "num_steps_bw_draws": 5},
grid={"num_draws": [10], "num_chains": [5, 10, 20, 50, 100]}
)
online_covariance_estimation_sweep_df
```



| | step | eval | eval_idx | draw_idx | llc/mean | llc/std | llc-chain/0 | llc-chain/1 | llc-chain/2 | llc-chain/3 | ... | llc-chain/90 | llc-chain/91 | llc-chain/92 | llc-chain/93 | llc-chain/94 | llc-chain/95 | llc-chain/96 | llc-chain/97 | llc-chain/98 | llc-chain/99 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0.065118 | 0 | 0 | -64.768623 | 6.509354 | -57.949642 | -57.933109 | -66.291466 | -70.412163 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 1 | 1 | 0.096832 | 1 | 0 | -64.768623 | 6.509354 | -57.949642 | -57.933109 | -66.291466 | -70.412163 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 2 | 1 | 0.156086 | 2 | 0 | -64.768623 | 6.509354 | -57.949642 | -57.933109 | -66.291466 | -70.412163 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 3 | 1 | 0.477032 | 0 | 1 | -64.768623 | 6.509354 | -57.949642 | -57.933109 | -66.291466 | -70.412163 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 4 | 1 | 0.740029 | 1 | 1 | -64.768623 | 6.509354 | -57.949642 | -57.933109 | -66.291466 | -70.412163 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 6895 | 20000 | 1.192721 | 1 | 8 | 1.944060 | 1.137619 | 2.551837 | 1.420484 | 1.000755 | 0.489775 | ... | 0.718788 | 2.007515 | 0.615772 | 1.354407 | 1.349531 | 1.069391 | 2.402186 | 1.429595 | 2.973875 | 5.039538 |
| 6896 | 20000 | 1.210239 | 2 | 8 | 1.944060 | 1.137619 | 2.551837 | 1.420484 | 1.000755 | 0.489775 | ... | 0.718788 | 2.007515 | 0.615772 | 1.354407 | 1.349531 | 1.069391 | 2.402186 | 1.429595 | 2.973875 | 5.039538 |
| 6897 | 20000 | 1.169081 | 0 | 9 | 1.944060 | 1.137619 | 2.551837 | 1.420484 | 1.000755 | 0.489775 | ... | 0.718788 | 2.007515 | 0.615772 | 1.354407 | 1.349531 | 1.069391 | 2.402186 | 1.429595 | 2.973875 | 5.039538 |
| 6898 | 20000 | 1.263959 | 1 | 9 | 1.944060 | 1.137619 | 2.551837 | 1.420484 | 1.000755 | 0.489775 | ... | 0.718788 | 2.007515 | 0.615772 | 1.354407 | 1.349531 | 1.069391 | 2.402186 | 1.429595 | 2.973875 | 5.039538 |
| 6899 | 20000 | 1.390757 | 2 | 9 | 1.944060 | 1.137619 | 2.551837 | 1.420484 | 1.000755 | 0.489775 | ... | 0.718788 | 2.007515 | 0.615772 | 1.354407 | 1.349531 | 1.069391 | 2.402186 | 1.429595 | 2.973875 | 5.039538 |

Figure 6: **Prima Example** Online Covariance Evolution Under Varying SGLD Sampling Parameters

The table produced provides insights into how the covariance structure of the model's weights evolves as we accumulate more samples in an online estimation process. It's particularly focused on understanding how different Stochastic Gradient Langevin Dynamics (SGLD) sampling parameters affect this evolution.

Here's what each column represents:

1. step: The training iteration, indicating the model's progress.
2. eval, eval_idx: The eigenvalue of the covariance matrix and its index. These eigenvalues are computed cumulatively, so eval at a given step represents the eigenvalue computed using all samples up to that step.
3. llc/mean, llc/std: The mean and standard deviation of the Local Learning Coefficient (LLC) across all chains. These are not directly relevant to the covariance analysis but provide context about the learning dynamics.
4. llc-chain/*: LLC values for individual SGLD chains. Again, not directly used in covariance analysis but provide context.
5. loss/trace: The mean loss across all chains, giving a sense of model performance over time.
6. Hyperparameters:
   - batch_size: Fixed at 10 for all runs.
   - localization: A parameter controlling the SGLD dynamics (not varied here).
   - lr: Learning rate (fixed).
   - noise_level: Amount of noise added in SGLD (fixed).
   - temperature: Parameter for the SGLD sampling distribution (fixed).
   - num_draws: Number of SGLD draws per chain (fixed at 10).
   - num_chains: Number of parallel SGLD chains, varied across 5, 10, 20, 50, and 100.
   - num_burnin_steps: Set to 0, meaning no burn-in period.

- num_steps_bw_draws: Set to 5, meaning 5 SGLD steps are taken between each draw.

Key Notes:

1. Online Covariance Estimation: By fixing num_steps_bw_draws and varying step, we see how the covariance estimate (via eigenvalues) changes as more samples are incorporated. This tells us how many samples are needed for the estimate to stabilize.

2. Impact of num_chains: This is the primary hyperparameter being varied. More chains mean more independent trajectories are used to estimate the covariance. We can see how increasing num_chains affects:

   - Stability of eigenvalues: Do they converge faster or more reliably with more chains?
   - Eigenvalue magnitudes: Does the relative magnitude of eigenvalues change, indicating a shift in how dimensions are used?

3. No Burn-in (num_burnin_steps=0): This means we're using the SGLD samples directly, without discarding any. This could introduce bias if the chains haven't properly mixed, but it also gives us insight into the transient dynamics.

4. Intra-Sampling Dynamics (num_steps_bw_draws=5): Five SGLD steps are taken between each sample. This adds some independence between samples but doesn't waste too much computation.

5. NaN Values: Any NaN values, especially in early steps or for certain num_chains values, could indicate numerical issues or that the covariance estimate is ill-conditioned early on.

This table is particularly valuable for understanding:

1. How quickly the model's internal representations (as captured by the covariance) stabilize. Whether more independent samples (higher num_chains) lead to qualitatively different representations or just faster convergence.

2. The interplay between the model's training dynamics (captured by step) and the SGLD sampling dynamics.

By comparing this to the previous tables and plots, we get a richer picture of how the model's developmental phases are influenced by both training progress and the specifics of how we probe its internal state.

**Algorithm 7 Prima Example** Covariance Eigenvalue Evolution Across Online Estimation Steps

```
online_covariance_estimation_sweep_df =
online_covariance_estimation_sweep_df.dropna()
plot_covariance_estimation_hyperparam_sweep(online_covariance_estimation_sweep_df,
col="draw_idx", row="num_chains")
```

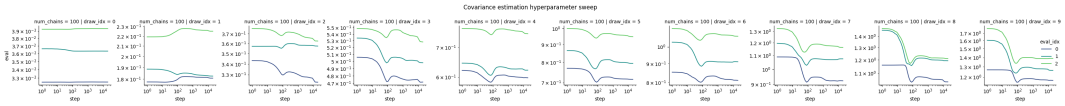



Figure 7: **Prima Example** Covariance Eigenvalue Evolution Across Online Estimation Steps

This figure is a grid of line plots, each representing how the largest eigenvalues of the weight covariance matrix change as we accumulate more samples (steps) in an online estimation process.

To understand this better:

1. Each subplot corresponds to a different "draw_idx" (draw index), which likely represents different points in the model's training process.
2. The x-axis is the "step" on a logarithmic scale, representing how many samples have been used to estimate the covariance.
3. The y-axis is the eigenvalue magnitude, also on a logarithmic scale.
4. There are three lines in each plot, corresponding to the three largest eigenvalues (eval_idx 0, 1, 2).

Takeaways:

- The eigenvalues generally stabilize as more steps are included, suggesting that the covariance estimate converges.
- The relative magnitudes of the eigenvalues can tell us about the dimensionality of the weight space being used. For example, if one eigenvalue is much larger than the others, it suggests the model is primarily using one dimension.
- Changes in the eigenvalue patterns across different "draw_idx" values could indicate phase transitions in the model's development, like those observed in the previous figure.

Overall, these figures provide insights into how the model's internal representations evolve during training (last figure) and how this evolution is affected by training hyperparameters (previous figure). This is crucial for developmental interpretability, as it helps identify critical periods of change and understand how different settings might encourage or inhibit the model's ability to develop distinct, meaningful representations.

# Hidden Dimensions

In this section, we go over results after having increased the amount of hidden dimensions and kept everything else equal. Recall that the first example has 2 hidden dimensions only. Please note that I only increased this parameter up to 4 due to current lack of computational power and time. I'd suggest the reader to try going up even higher (e.g. 5, 10) and observe how the results to be mentioned could be emphasized.

### 3 Hidden Dimensions

**Algorithm 8 3 Hidden Dimensions** Loss and Weight Snapshots

```
NUM_FEATURES = 8
NUM_HIDDEN_UNITS = 3 # r increased
NUM_SAMPLES = 1000
NUM_EPOCHS = 20000
INIT_KGON = 2
NUM_OBSERVATIONS = 50
STEPS = sorted(list(set(np.logspace(0, np.log10(NUM_EPOCHS),
NUM_OBSERVATIONS).astype(int))))
PLOT_STEPS = [min(STEPS, key=lambda s: abs(s-i)) for i in [0, 200, 2000,
10000, NUM_EPOCHS - 1]]
PLOT_INDICES = [STEPS.index(s) for s in PLOT_STEPS]
logs, weights = create_and_train(NUM_FEATURES,
NUM_HIDDEN_UNITS, num_samples=NUM_SAMPLES,
log_ivl=STEPS, batch_size=100, lr=0.01, num_epochs=NUM_EPOCHS,
init_kgon=INIT_KGON, init_zerobias=False, seed=1)
weights_to_plot = [weights[i]['embedding.weight'] for i in PLOT_INDICES]
losses = [logs.loc[logs['step'] == s, 'loss'].values[0] for s in STEPS]
plot_losses_and_polygons(STEPS, losses, PLOT_STEPS, weights_to_plot)
plt.show()
```

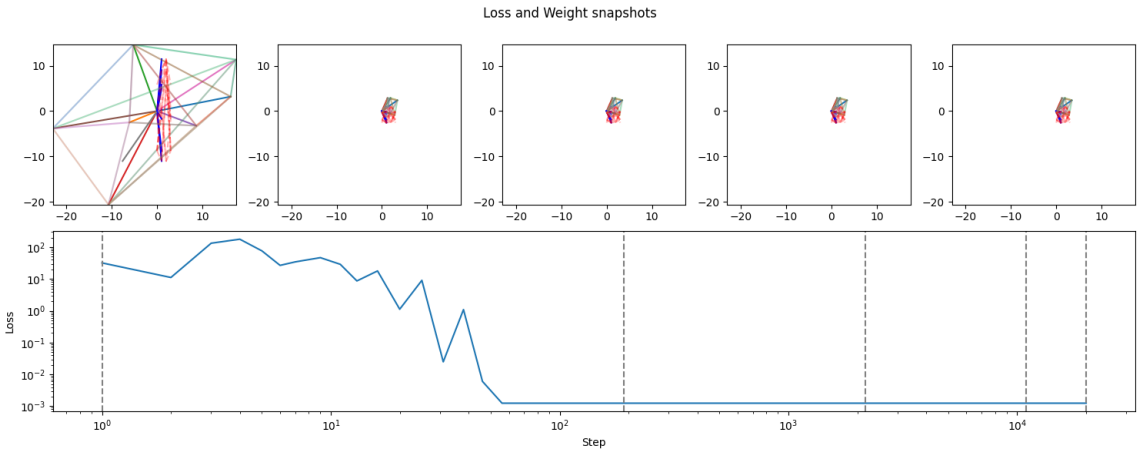



Figure 8: **3 Hidden Dimensions** Loss and Weight Snapshots

We will now compare and contrast our new model findings with that of the prima example's.

**Loss and Weight Snapshots (Figures 1 vs. 8):**

- Loss Curve: Both models show a rapid decrease in loss early on, followed by stabilization. However, the 3D model stabilizes a bit faster at less than 10^2 steps while the 2D model stabilizes over that step quantity, although more smoothly. Also note the 2D model stabilizes at around 10^-3 as does the 3D model, or at a marginally lower value. All this suggests the extra dimension allows for better fitting of the data.

- Weight Snapshots Comparison: In the 2D model, the weights transition from a line to a V to an L, indicating a progression from using one dimension to using both independently. In this 3D model, the transition is more complex. It spreads out in 3D space, immediately. The model is using all three dimensions in an interrelated way. The final configuration isn't a simple orthogonal shape like the 'L' in 2D. The model isn't just using the three dimensions independently (which would look like a 3D 'L' or 'corner'), but is using them in a more entangled yet more careful way; observe how the weights and overall space being used has diminished. It is possibly leveraging interactions or correlations between these dimensions. This "entanglement" might allow for more complex representations or capture more nuanced features in the data, allowing for a slightly better or faster fitting observed in the loss curve.

| | llc | llc/std | batch_size | lr | t_sgld | llc_type | loss |
|---|---|---|---|---|---|---|---|
| **0** | NaN | NaN | 1 | 0.00001 | 0 | mean | 0.125000 |
| **1** | NaN | NaN | 1 | 0.00001 | 0 | 0 | 0.125000 |
| **2** | NaN | NaN | 1 | 0.00001 | 0 | 1 | 0.125000 |
| **3** | NaN | NaN | 1 | 0.00001 | 0 | 2 | 0.125000 |
| **4** | NaN | NaN | 1 | 0.00001 | 0 | 3 | 0.125000 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **11995** | 0.000000 | NaN | 300 | 0.01000 | 99 | 0 | 0.125000 |
| **11996** | -0.567291 | NaN | 300 | 0.01000 | 99 | 1 | 0.114214 |
| **11997** | 0.000000 | NaN | 300 | 0.01000 | 99 | 2 | 0.125000 |
| **11998** | 0.000000 | NaN | 300 | 0.01000 | 99 | 3 | 0.125000 |
| **11999** | -0.576393 | NaN | 300 | 0.01000 | 99 | 4 | 0.114041 |

Figure 9: **3 Hidden Dimensions** Hyperparameter Sweep Results: Local Learning Coefficients

**Hyperparameter Sweep (Figures 2 vs. 9):**

- LLC Values: In both cases, higher learning rates lead to higher LLCs (faster learning). However, the 3D model shows more NaN values, particularly at lower learning rates and smaller batch sizes. This increased occurrence of NaNs suggests that the higher dimensionality makes the model more susceptible to instabilities or ill-conditioning, especially when hyperparameters are not well-tuned.
- Loss: When the 3D model is stable (no NaNs), it generally achieves lower losses than the 2D model, especially with larger batch sizes. This indicates that the additional dimension, when properly leveraged, allows for better performance.

Overall Insight: The comparison reveals a trade-off. The 3D model's increased expressiveness allows for better performance (lower losses) when hyperparameters are well-tuned, especially with more data per update. However, this comes at the cost of increased sensitivity to hyperparameter choices, particularly learning rates and batch sizes, as evidenced by more NaNs. This underscores the importance of careful hyperparameter tuning, especially as model complexity increases.

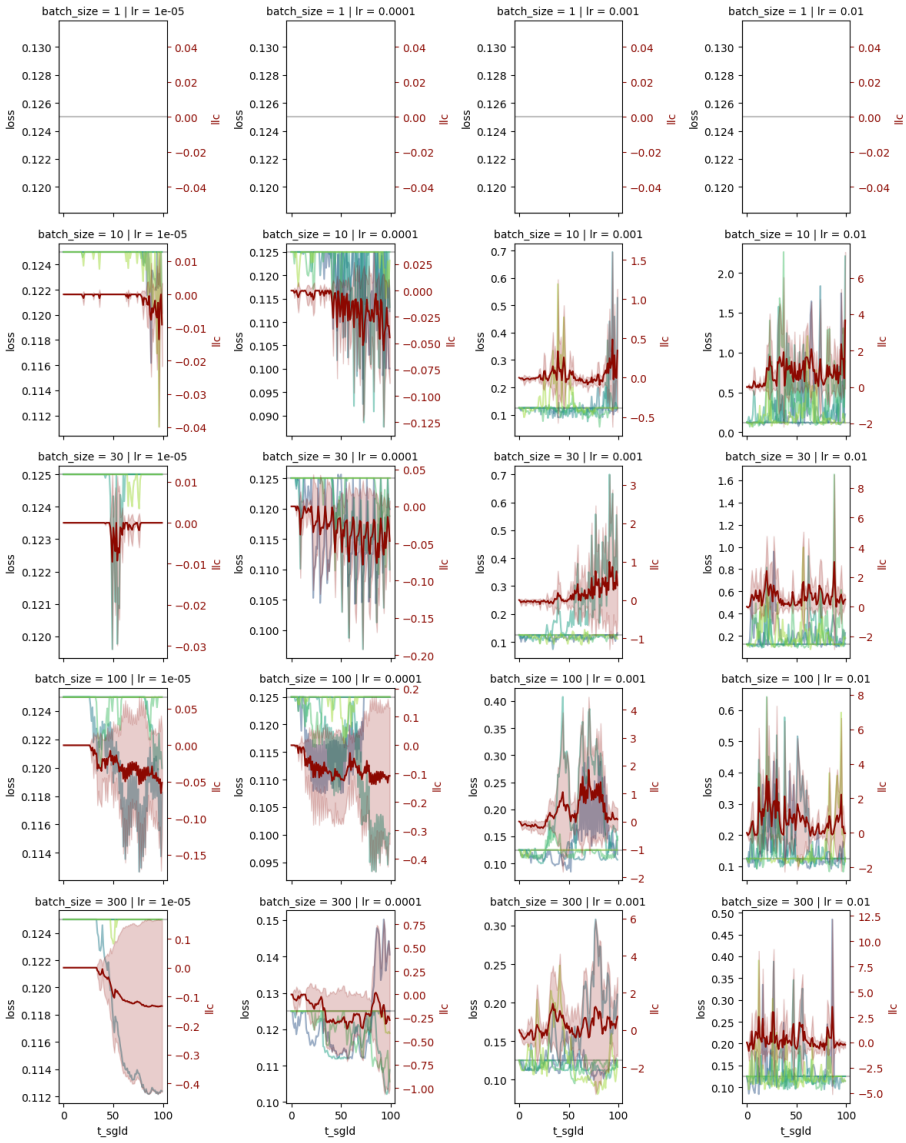
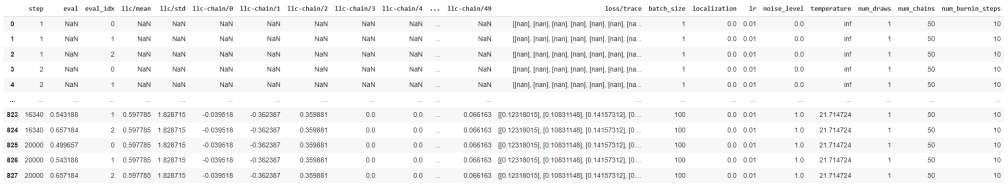



Figure 10: **3 Hidden Dimensions** Batch Size and Learning Rate Effect on Loss and the Local Learning Coefficient

**Batch Size and Learning Rate Effects (Figures 3 vs. 10):**

- LLC Peaks: Both models show peaks in LLC, suggesting critical learning periods. However, the 3D model's peaks are more pronounced and occur at different times, indicating that the developmental phases are altered by the extra dimension.

- Loss Curves: The 3D model's loss drops faster initially, likely due to the extra representational power. But it also shows more variability, suggesting a more complex loss landscape. Note this was also observed in Figure 8's loss description.



| | step | eval | eval_idx | llc/mean | llc/std | llc-chain/0 | llc-chain/1 | llc-chain/2 | llc-chain/3 | llc-chain/4 | ... | llc-chain/49 | loss/trace | batch_size | localization | lr | noise_level | temperature | num_draws | num_chains | num_burnin_steps |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | NaN | 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 1 | 1 | NaN | 1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 2 | 1 | NaN | 2 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 3 | 2 | NaN | 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 4 | 2 | NaN | 1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 823 | 16340 | 0.543188 | 1 | 0.597785 | 1.828715 | -0.039518 | -0.362387 | 0.359881 | 0.0 | 0.0 | ... | 0.066163 | [[0.12318015], [0.10831148], [0.14157312], [0.... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 824 | 16340 | 0.657184 | 2 | 0.597785 | 1.828715 | -0.039518 | -0.362387 | 0.359881 | 0.0 | 0.0 | ... | 0.066163 | [[0.12318015], [0.10831148], [0.14157312], [0.... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 825 | 20000 | 0.499657 | 0 | 0.597785 | 1.828715 | -0.039518 | -0.362387 | 0.359881 | 0.0 | 0.0 | ... | 0.066163 | [[0.12318015], [0.10831148], [0.14157312], [0.... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 826 | 20000 | 0.543188 | 1 | 0.597785 | 1.828715 | -0.039518 | -0.362387 | 0.359881 | 0.0 | 0.0 | ... | 0.066163 | [[0.12318015], [0.10831148], [0.14157312], [0.... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 827 | 20000 | 0.657184 | 2 | 0.597785 | 1.828715 | -0.039518 | -0.362387 | 0.359881 | 0.0 | 0.0 | ... | 0.066163 | [[0.12318015], [0.10831148], [0.14157312], [0.... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |

Figure 11: **3 Hidden Dimensions** Dynamic Evolution of Covariance Properties During Model Training

**Covariance Evolution (Figures 4 vs. 11):**

- Eigenvalue Magnitudes: In the last step, the 3D model has three significant eigenvalues of increased range magnitude (vs. 2D).
  - 3D:

| eval_idx | eval |
|---|---|
| 0 | 0.499657 |
| 1 | 0.543188 |
| 2 | 0.657184 |

  - 2D:

| eval_idx | eval |
|---|---|
| 0 | 0.355699 |
| 1 | 0.405897 |
| 2 | 0.485721 |

- LLC Patterns: The 2D model learns faster to begin and end with.

Further dissection can be obtained by running the script and viewing the ~800 rows/~50 columns. This isn't the core analysis, but still wanted to include this in here. I am learning as I go, but another noteworthy takeaway is that we have 0 values for llc-chains now as well as negative ones. We will see if this is complete randomness or if we observe this in the next highest dimension.

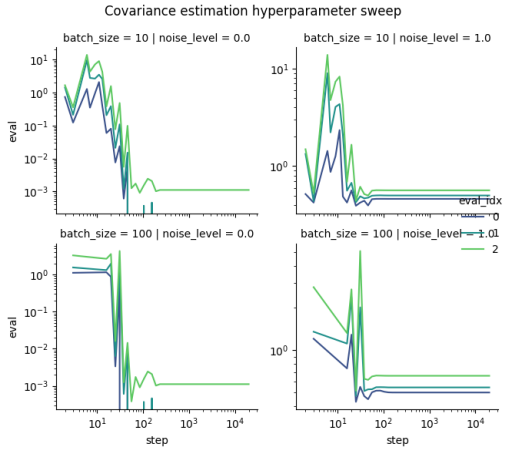



Figure 12: **3 Hidden Dimensions** Impact of Batch Size and Noise on Covariance Eigenvalues

**Batch Size and Noise Impact (Figures 5 vs. 12):**

- Eigenvalue Separation: In 2D, noise makes eigenvalues more similar. In 3D, this effect is less pronounced, suggesting that the extra dimension provides robustness against noise.
- Batch Size Effect: In both models, larger batch sizes lead to smoother eigenvalue curves. However, the effect is more dramatic in 3D, possibly because larger batches are needed to estimate the more complex covariance structure.

| | step | eval | eval_idx | draw_idx | llc/mean | llc/std | llc-chain/0 | llc-chain/1 | llc-chain/2 | llc-chain/3 | ... | llc-chain/90 | llc-chain/91 | llc-chain/92 | llc-chain/93 | llc-chain/94 | llc-chain/95 | llc-chain/96 | llc-chain/97 | llc-chain/98 | llc-chain/99 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 58.708069 | 0 | 0 | NaN | NaN | NaN | -142010.109375 | -144870.875000 | -143593.046875 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 1 | 1 | 247.684448 | 1 | 0 | NaN | NaN | NaN | -142010.109375 | -144870.875000 | -143593.046875 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 2 | 1 | 644.931885 | 2 | 0 | NaN | NaN | NaN | -142010.109375 | -144870.875000 | -143593.046875 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 3 | 1 | NaN | 0 | 1 | NaN | NaN | NaN | -142010.109375 | -144870.875000 | -143593.046875 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 4 | 1 | NaN | 1 | 1 | NaN | NaN | NaN | -142010.109375 | -144870.875000 | -143593.046875 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 6895 | 20000 | 1.595279 | 1 | 8 | 0.718332 | 0.784234 | 0.438817 | 0.623469 | -0.017982 | 0.000000 | ... | 0.828875 | 0.662783 | -0.00653 | 0.466373 | 0.051325 | -0.056593 | -0.008074 | 1.316041 | 0.805901 | 0.015925 |
| 6896 | 20000 | 1.693679 | 2 | 8 | 0.718332 | 0.784234 | 0.438817 | 0.623469 | -0.017982 | 0.000000 | ... | 0.828875 | 0.662783 | -0.00653 | 0.466373 | 0.051325 | -0.056593 | -0.008074 | 1.316041 | 0.805901 | 0.015925 |
| 6897 | 20000 | 1.591976 | 0 | 9 | 0.718332 | 0.784234 | 0.438817 | 0.623469 | -0.017982 | 0.000000 | ... | 0.828875 | 0.662783 | -0.00653 | 0.466373 | 0.051325 | -0.056593 | -0.008074 | 1.316041 | 0.805901 | 0.015925 |
| 6898 | 20000 | 1.685236 | 1 | 9 | 0.718332 | 0.784234 | 0.438817 | 0.623469 | -0.017982 | 0.000000 | ... | 0.828875 | 0.662783 | -0.00653 | 0.466373 | 0.051325 | -0.056593 | -0.008074 | 1.316041 | 0.805901 | 0.015925 |
| 6899 | 20000 | 1.767176 | 2 | 9 | 0.718332 | 0.784234 | 0.438817 | 0.623469 | -0.017982 | 0.000000 | ... | 0.828875 | 0.662783 | -0.00653 | 0.466373 | 0.051325 | -0.056593 | -0.008074 | 1.316041 | 0.805901 | 0.015925 |

Figure 13: **3 Hidden Dimensions** Online Covariance Evolution Under Varying SGLD Sampling Parameters

**Online Covariance Evolution (Figures 6 vs. 13):**

The 3D model starts off with more extreme eigenvalues and much larger absolute LLC values, indicating a more complex covariance structure, and a less stable starting structure, overall.

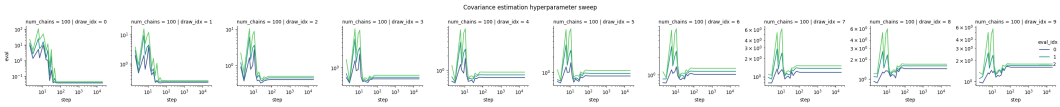



Figure 14: **3 Hidden Dimensions** Covariance Eigenvalue Evolution Across Online Estimation Steps

**Covariance Eigenvalue Evolution (Figures 7 vs. 14):**

- Developmental Phases: In 2D, there are clear phases where one eigenvalue dominates, then two. In 3D, the phases are less distinct. This suggests more nuanced, possibly overlapping developmental phases.
- Convergence: The 2D model's eigenvalues take longer to converge, especially the third eigenvalue, indicating a prolonged developmental period.

## Summary of Changes

1. Increased Performance: The extra dimension allows for better data fitting, as seen in lower loss values.
2. Complexity of Representations: In 2D, the model learns orthogonal representations. In 3D, the representations are more entangled, suggesting the model is capturing more complicated relationships.
3. Developmental Phases: The 3D model shows more complex, possibly overlapping phases.
4. Robustness and Instability: The extra dimension provides some robustness against noise but also makes the model more susceptible to instabilities, especially with certain learning rates.
5. Data Efficiency: The 3D model generally requires more data (larger batch sizes) and more samples for stable estimation, reflecting its increased complexity.

In summary, adding a third hidden dimension doesn't just provide more capacity; it fundamentally alters the model's developmental trajectory. The phases are more nuanced, the representations more complex, and the learning dynamics more variable. This suggests that as we scale to larger models, we might expect not just quantitative improvements in performance, but qualitative changes in how these models develop and represent information.

## 4 Hidden Dimensions

**Algorithm 9 4 Hidden Dimensions** Loss and Weight Snapshots

```
NUM_FEATURES = 8
NUM_HIDDEN_UNITS = 4 # r
NUM_SAMPLES = 1000
NUM_EPOCHS = 20000
INIT_KGON = 2
NUM_OBSERVATIONS = 50
STEPS = sorted(list(set(np.logspace(0, np.log10(NUM_EPOCHS),
NUM_OBSERVATIONS).astype(int))))
PLOT_STEPS = [min(STEPS, key=lambda s: abs(s-i)) for i in [0, 200, 2000,
10000, NUM_EPOCHS - 1]]
PLOT_INDICES = [STEPS.index(s) for s in PLOT_STEPS]
logs, weights = create_and_train(NUM_FEATURES,
NUM_HIDDEN_UNITS, num_samples=NUM_SAMPLES,
log_ivl=STEPS, batch_size=100, lr=0.01, num_epochs=NUM_EPOCHS,
init_kgon=INIT_KGON, init_zerobias=False, seed=1)
weights_to_plot = [weights[i]['embedding.weight'] for i in PLOT_INDICES]
losses = [logs.loc[logs['step'] == s, 'loss'].values[0] for s in STEPS]
plot_losses_and_polygons(STEPS, losses, PLOT_STEPS, weights_to_plot)
plt.show()
```

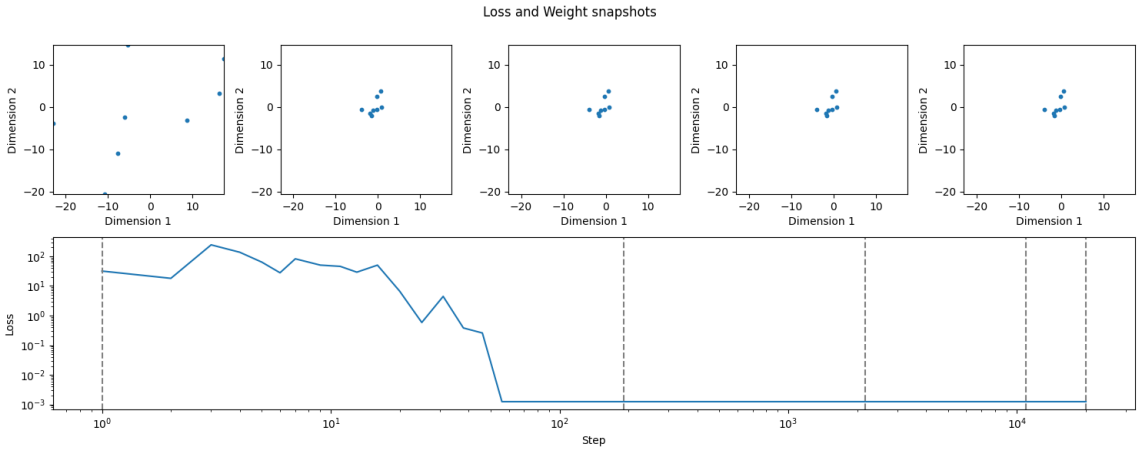



Figure 15: **4 Hidden Dimensions** Loss and Weight Snapshots

**Loss and Weight Snapshots (Figures 1 vs. 15):**

- Loss Curve: Both models show rapid early loss decrease, however the 4D model is faster again in doing so (at less than 10^2 steps). The 4D model also stabilizes at a loss marginally lower than the 2D and 3D models. This suggests that the extra dimensions allow for an even more precise fit to the data.
- Weight Snapshots:
  - 2D: Transitions from a line to a V to an L, indicating progression from one to two independent dimensions.
  - 4D: The transition is dramatically different. It quickly spreads out in 4D space, which is not-so-easy to plot. The final snapshots show a possibly complex structure. This could suggest the model is using all four dimensions in a highly interdependent manner, very different from the simple independent representations in 2D.

| | llc | llc/std | batch_size | lr | t_sgld | llc_type | loss |
|---|---|---|---|---|---|---|---|
| **0** | NaN | NaN | 1 | 0.00001 | 0 | mean | 0.125000 |
| **1** | NaN | NaN | 1 | 0.00001 | 0 | 0 | 0.125000 |
| **2** | NaN | NaN | 1 | 0.00001 | 0 | 1 | 0.125000 |
| **3** | NaN | NaN | 1 | 0.00001 | 0 | 2 | 0.125000 |
| **4** | NaN | NaN | 1 | 0.00001 | 0 | 3 | 0.125000 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **11995** | 0.000000 | NaN | 300 | 0.01000 | 99 | 0 | 0.125000 |
| **11996** | 0.000000 | NaN | 300 | 0.01000 | 99 | 1 | 0.125000 |
| **11997** | 0.000000 | NaN | 300 | 0.01000 | 99 | 2 | 0.125000 |
| **11998** | 0.000000 | NaN | 300 | 0.01000 | 99 | 3 | 0.125000 |
| **11999** | -0.497145 | NaN | 300 | 0.01000 | 99 | 4 | 0.115548 |

Figure 16: **4 Hidden Dimensions** Hyperparameter Sweep Results: Local Learning Coefficients

**Hyperparameter Sweep (Figures 2 vs. 16):**

- LLC Values: In both 2D and 4D cases, higher learning rates lead to higher absolute LLCs. However, this 4D model shows more NaN values at lower learning rates and smaller batch sizes just like the 3D model did. Again, this increased occurrence of NaNs suggests that the higher dimensionality makes the model more susceptible to instabilities or ill-conditioning, especially when hyperparameters are not well-tuned.
- Loss: Like the 3D model, when the 4D model is stable (no NaNs), it achieves lower losses than the 2D model with larger batch sizes. This indicates that the additional dimension, when properly leveraged, allows for better performance.

Overall Insight: The comparison reveals the same trade-off, again, with the 2D and 3D one. Note, curiously, how the loss values between the 3D and 4D models are highly comparable. I wonder what would happen to these values as we go into the 5th dimension and higher.

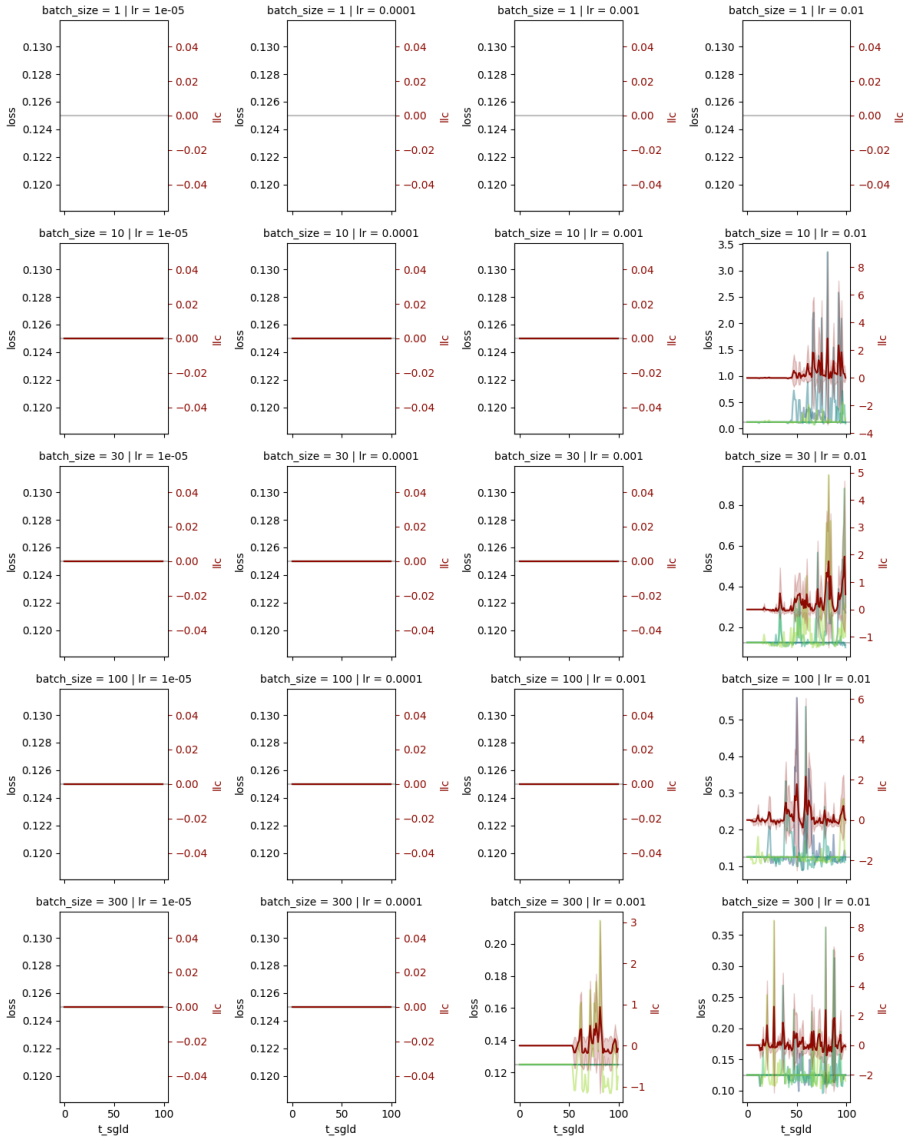



Figure 17: **4 Hidden Dimensions** Batch Size and Learning Rate Effect on Loss and the Local Learning Coefficient

**Batch Size and Learning Rate Effects (Figures 3 vs. 17):**

- LLC Peaks: The 4D model shows the most dramatic LLC peaks yet. These peaks are sharper and occur at different times compared to 2D and 3D, suggesting that the developmental phases in 4D are not just extended versions of lower-dimensional phases but qualitatively distinct.
- Loss Curves: The 4D loss drops precipitously fast, much faster than in 2D or 3D. However, it also shows the most variability, indicating an even more complex and possibly non-convex loss landscape.

| | step | eval | eval_idx | llc/mean | llc/std | llc-chain/0 | llc-chain/1 | llc-chain/2 | llc-chain/3 | llc-chain/4 | ... | llc-chain/49 | loss/trace | batch_size | localization | lr | noise_level | temperature | num_draws | num_chains | num_burnin_steps |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | NaN | 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 1 | 1 | NaN | 1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 2 | 1 | NaN | 2 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 3 | 2 | NaN | 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 4 | 2 | NaN | 1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 823 | 16340 | 0.593412 | 1 | -0.022229 | 0.081133 | 0.0 | 0.0 | 0.009757 | 0.0 | 0.0 | ... | 0.0 | [[0.125], [0.125], [0.12544931], [0.125], [0.1... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 824 | 16340 | 0.686349 | 2 | -0.022229 | 0.081133 | 0.0 | 0.0 | 0.009757 | 0.0 | 0.0 | ... | 0.0 | [[0.125], [0.125], [0.12544931], [0.125], [0.1... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 825 | 20000 | 0.565439 | 0 | -0.022229 | 0.081133 | 0.0 | 0.0 | 0.009757 | 0.0 | 0.0 | ... | 0.0 | [[0.125], [0.125], [0.12544931], [0.125], [0.1... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 826 | 20000 | 0.593412 | 1 | -0.022229 | 0.081133 | 0.0 | 0.0 | 0.009757 | 0.0 | 0.0 | ... | 0.0 | [[0.125], [0.125], [0.12544931], [0.125], [0.1... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 827 | 20000 | 0.686349 | 2 | -0.022229 | 0.081133 | 0.0 | 0.0 | 0.009757 | 0.0 | 0.0 | ... | 0.0 | [[0.125], [0.125], [0.12544931], [0.125], [0.1... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |

Figure 18: **4 Hidden Dimensions** Dynamic Evolution of Covariance Properties During Model Training

**Covariance Evolution (Figures 4 vs. 18):**

- I didn't expect this, but the LLC is lowering down to zero for even more llc_chain/* numbers. The rest of the values have decreased noticeably as well (compare and contrast to those in Figures 4 and 11). Let's take a closer look:
  - Recall that *llc-chain/0 to llc-chain/49* represent LLC values for individual SGLD chains, which give insight into the **variability of learning** across different stochastic trajectories. Hence, learning variability is actually decreasing for higher dimensions as the amount of steps increases.
  - Some chains even have negative LLCs, indicating periods of "unlearning" or reorganization, a phenomenon not seen in 2D, but noted in 3D; we can now discard this as being an offhand case.

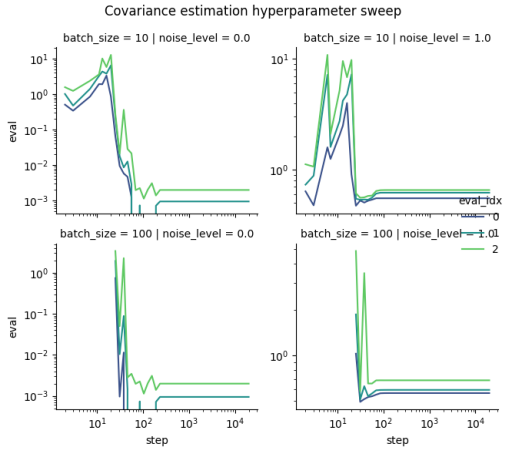



Figure 19: **4 Hidden Dimensions** Impact of Batch Size and Noise on Covariance Eigenvalues

**Batch Size and Noise Impact (Figures 5 vs. 19):**

- Eigenvalue Separation: In 2D, noise made eigenvalues more similar. In 4D, the effect is even less pronounced than in 3D. The eigenvalues remain distinct even with noise, suggesting that higher dimensions provide even more robustness.
- Batch Size Effect: The impact of batch size is most dramatic in 4D. Small batches lead to highly unstable eigenvalue estimates, while large batches are necessary for smooth curves. This underscores that estimating a 4D covariance is much more data-hungry.

| | step | eval | eval_idx | draw_idx | llc/mean | llc/std | llc-chain/0 | llc-chain/1 | llc-chain/2 | llc-chain/3 | ... | llc-chain/90 | llc-chain/91 | llc-chain/92 | llc-chain/93 | llc-chain/94 | llc-chain/95 | llc-chain/96 | llc-chain/97 | llc-chain/98 | llc-chain/99 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 105.477608 | 0 | 0 | NaN | NaN | NaN | NaN | -174678.640625 | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **1** | 1 | 1529.002563 | 1 | 0 | NaN | NaN | NaN | NaN | -174678.640625 | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **2** | 1 | 2141.168945 | 2 | 0 | NaN | NaN | NaN | NaN | -174678.640625 | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **3** | 1 | NaN | 0 | 1 | NaN | NaN | NaN | NaN | -174678.640625 | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **4** | 1 | NaN | 1 | 1 | NaN | NaN | NaN | NaN | -174678.640625 | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **6895** | 20000 | 1.749916 | 1 | 8 | 0.573014 | 0.638321 | 0.43449 | -0.025929 | 0.388708 | 0.064064 | ... | 0.057276 | 0.13934 | 1.984684 | 0.025876 | 0.087687 | 0.0 | 0.123116 | 2.932812 | 0.257771 | 0.039753 |
| **6896** | 20000 | 1.937240 | 2 | 8 | 0.573014 | 0.638321 | 0.43449 | -0.025929 | 0.388708 | 0.064064 | ... | 0.057276 | 0.13934 | 1.984684 | 0.025876 | 0.087687 | 0.0 | 0.123116 | 2.932812 | 0.257771 | 0.039753 |
| **6897** | 20000 | 1.774929 | 0 | 9 | 0.573014 | 0.638321 | 0.43449 | -0.025929 | 0.388708 | 0.064064 | ... | 0.057276 | 0.13934 | 1.984684 | 0.025876 | 0.087687 | 0.0 | 0.123116 | 2.932812 | 0.257771 | 0.039753 |
| **6898** | 20000 | 1.837693 | 1 | 9 | 0.573014 | 0.638321 | 0.43449 | -0.025929 | 0.388708 | 0.064064 | ... | 0.057276 | 0.13934 | 1.984684 | 0.025876 | 0.087687 | 0.0 | 0.123116 | 2.932812 | 0.257771 | 0.039753 |
| **6899** | 20000 | 2.008173 | 2 | 9 | 0.573014 | 0.638321 | 0.43449 | -0.025929 | 0.388708 | 0.064064 | ... | 0.057276 | 0.13934 | 1.984684 | 0.025876 | 0.087687 | 0.0 | 0.123116 | 2.932812 | 0.257771 | 0.039753 |

Figure 20: **4 Hidden Dimensions** Online Covariance Evolution Under Varying SGLD Sampling Parameters

**Online Covariance Evolution (Figures 6 vs. 20):**

The 4D model still requires many num_chains for eigenvalues to stabilize. Even with 100 chains, there's still some fluctuation, indicating a very complex learning structure. However, the ending LLC chain values are lower than those

of the 2D and 3D models, possibly indicating it has less to learn toward the end in contrast with the smaller-dimensional models. Note that its starting eigenvalues are also more extreme to begin with in comparison to the other models, but stabilize at the end as well.

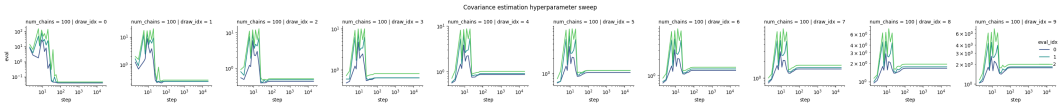



Figure 21: **4 Hidden Dimensions** Covariance Eigenvalue Evolution Across Online Estimation Steps

**Covariance Eigenvalue Evolution (Figures 7 vs. 21):**

- Developmental Phases: The 4D model shows the most intricate phase transitions. There are periods where one, two, three, or all four eigenvalues are significant, and these periods don't follow a simple sequential pattern. This suggests a highly nuanced developmental process with possibly recursive or branching phases.
- Convergence: Even by the end of the plots, the 4D eigenvalues haven't fully synced throughout the training process, especially the third and fourth. This indicates an extended, possibly open-ended developmental period, in stark contrast to the quick sync in 2D. However, all values converge in less than 10^2 steps for the 4D and 3D models.

# Summary of Changes

1. Performance Leap: The jump in performance (lower loss) from 2D to 4D is more significant than from 2D to 3D, suggesting superlinear returns to increased dimensionality.
2. Representational Complexity: The 4D representations are highly entangled and context-dependent. This is a qualitative shift from the independent dimensions in 2D or the partially entangled ones in 3D.
3. Developmental Richness: The 4D model exhibits the most complex developmental trajectory. Phases are not just extended but seem to interleave, branch, or even recur. This hints at a form of "developmental plasticity" not seen in lower dimensions.
4. Robustness and Challenge: Higher dimensions provide more robustness against noise but also make the model more susceptible to instabilities and harder to train (more NaNs). This suggests a tradeoff between representational power and training difficulty.

5. Data Hunger: The 4D model is the most data-hungry, requiring larger batches and more samples for stable estimates. This makes sense as more dimensions mean more parameters to estimate. Higher dimensions also have periods of negative LLC, suggesting times when the model "unlearns" or drastically reorganizes its representations. This could be analogous to synaptic pruning or major rewiring events in biological neural development.

In summary, the transition from 2D to 4D isn't just a quantitative improvement but a qualitative transformation in how the model develops and represents information. The 4D model shows a richer, more nuanced developmental trajectory with phases that interleave and possibly recur. It captures more complex relationships in the data but at the cost of being more challenging to train and understand. This suggests that as we scale to much higher dimensionalities of modern deep learning models, we might expect not just better performance but fundamentally different learning dynamics, possibly mirroring the intricate developmental processes of biological brains.

# Sparsity Changes

We will now use denser inputs with more non-zero elements and keep everything else equal as the paper focused on sparse inputs.

That is, we are keeping the same amount of hidden dimensions as the prima example (2D), assigning no feature importance, and changing sparsity to levels in the list: [0.3, 0.5, 0.7]. Observe how this affects the formation of superpositions and the critical points (k-gons).

### Sparsity=0.3

**Algorithm 10 0.3 Sparsity** Loss and Weight Snapshots

```
...
# old: creates sparse dataset
#dataset = SyntheticBinaryValued(num_samples, m, 1)
# new: creates dense dataset
dataset = SyntheticUniformValued(num_samples, m, 0.3) # s=0.3
...
NUM_FEATURES = 8
NUM_HIDDEN_UNITS = 2 # r
NUM_SAMPLES = 1000
NUM_EPOCHS = 20000
INIT_KGON = 2
NUM_OBSERVATIONS = 50
STEPS = sorted(list(set(np.logspace(0, np.log10(NUM_EPOCHS),
NUM_OBSERVATIONS).astype(int))))
PLOT_STEPS = [min(STEPS, key=lambda s: abs(s-i)) for i in [0, 200, 2000,
10000, NUM_EPOCHS - 1]]
PLOT_INDICES = [STEPS.index(s) for s in PLOT_STEPS]
logs, weights = create_and_train(NUM_FEATURES,
NUM_HIDDEN_UNITS, num_samples=NUM_SAMPLES,
log_ivl=STEPS, batch_size=100, lr=0.01, num_epochs=NUM_EPOCHS,
init_kgon=INIT_KGON, init_zerobias=False, seed=1)
weights_to_plot = [weights[i]['embedding.weight'] for i in PLOT_INDICES]
losses = [logs.loc[logs['step'] == s, 'loss'].values[0] for s in STEPS]
plot_losses_and_polygons(STEPS, losses, PLOT_STEPS, weights_to_plot)
plt.show()
```

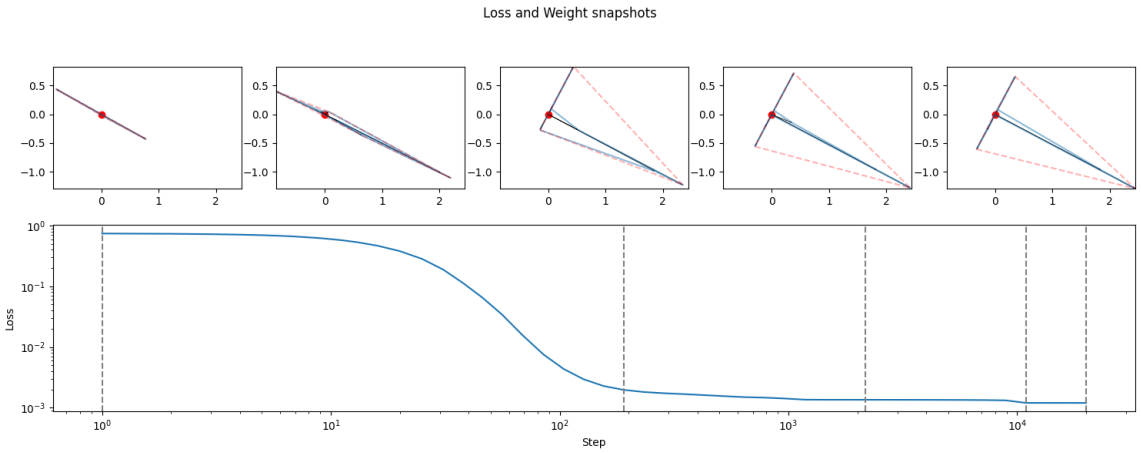



Figure 22: **0.3 Sparsity** Loss and Weight Snapshots

**Loss and Weight Snapshots (Figures 1 vs. 22):**

- Loss Curve: Both models show a rapid decrease in loss early in training, but the the prima example reaches a lower final loss ($<<$ 10^-3) compared to Sparsity=0.3 model ($<$ 10^-3). We will see if this holds true as we increase this parameter.
- Weight Snapshots: The prima example shows a clear transition from a line to a V-shape to an L-shape. In contrast, the Sparsity=0.3 model's transitions are less distinct. It starts with a line, but then quickly transitions to a more spread-out configuration, without the clear V and L shapes. This suggests that with denser inputs, the model uses both dimensions more uniformly from early on, rather than going through distinct developmental phases.

**Algorithm 11 0.3 Sparsity** Hyperparameter Sweep Results: Local Learning Coefficients

```
# Sweep SGLD hyperparameters
NUM_SAMPLES_TEST = 200
NUM_DRAWS_SGLD = 100
NUM_CHAINS_SGLD = 5
# -- old
# Create a sparse dataset
# the SyntheticBinaryValued class creates sparse binary inputs where each element is either 0 or 1,
# with a sparsity level controlled by the sparsity parameter.
"'
dataset = SyntheticBinaryValued(NUM_SAMPLES_TEST, NUM_FEATURES, 1)
dataset_double = TensorDataset(dataset.data, dataset.data)
"'
# -- end old
# -- new
# Create a dense dataset
# The SyntheticUniformValued class generates inputs where each element is a uniform random value between 0 and 1,
# regardless of the sparsity parameter.
dataset = SyntheticUniformValued(NUM_SAMPLES, NUM_FEATURES, 0.3) # s
dataset_double = TensorDataset(dataset.data, dataset.data)
# Additionally, you can modify the SyntheticDataset class to create other types of dense inputs, such as Gaussian or non-negative inputs.
# -- end new
model = ToyAutoencoder(NUM_FEATURES, NUM_HIDDEN_UNITS, final_bias=True)
...
lambdahat_sweep_df = sweep_lambdahat_estimation_hyperparams(model, dataset_double)
```

| | llc | llc/std | batch_size | lr | t_sgld | llc_type | loss |
|---|---|---|---|---|---|---|---|
| **0** | NaN | NaN | 1 | 0.00001 | 0 | mean | 0.131014 |
| **1** | -inf | NaN | 1 | 0.00001 | 0 | 0 | 0.127174 |
| **2** | inf | NaN | 1 | 0.00001 | 0 | 1 | 0.162341 |
| **3** | -inf | NaN | 1 | 0.00001 | 0 | 2 | 0.133879 |
| **4** | -inf | NaN | 1 | 0.00001 | 0 | 3 | 0.078263 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **11995** | 11.022593 | NaN | 300 | 0.01000 | 99 | 0 | 0.334640 |
| **11996** | 8.014220 | NaN | 300 | 0.01000 | 99 | 1 | 0.277443 |
| **11997** | 6.525927 | NaN | 300 | 0.01000 | 99 | 2 | 0.249147 |
| **11998** | 4.910871 | NaN | 300 | 0.01000 | 99 | 3 | 0.218440 |
| **11999** | 3.978192 | NaN | 300 | 0.01000 | 99 | 4 | 0.200707 |

Figure 23: **0.3 Sparsity** Hyperparameter Sweep Results: Local Learning Coefficients

**Hyperparameter Sweep (Figures 2 vs. 23):**

- Both models show higher Local Learning Coefficients (LLCs) for larger batch sizes and learning rates. However, the Sparsity=0.3 model generally has higher LLC values, indicating faster learning. This is consistent with the lower final loss, as denser inputs provide more information per sample.

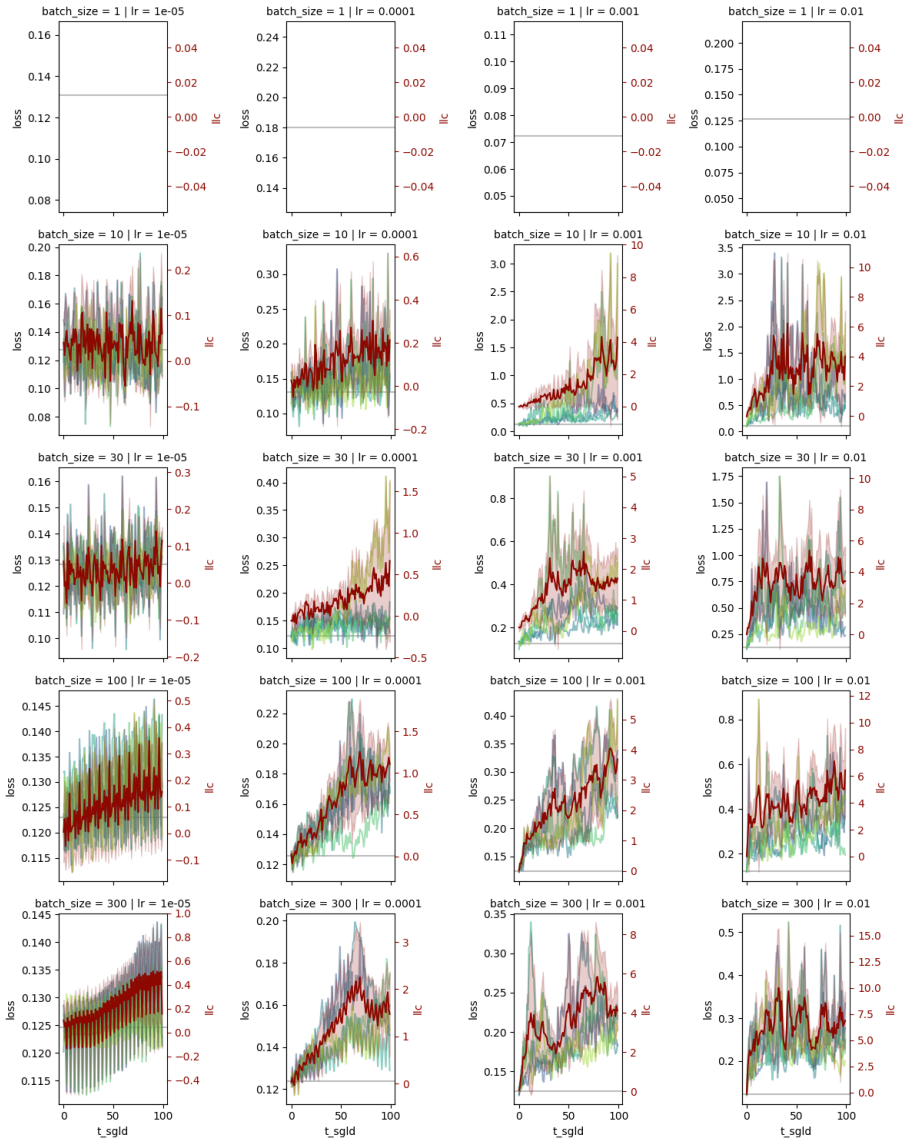



Figure 24: **0.3 Sparsity** Batch Size and Learning Rate Effect on Loss and the Local Learning Coefficient

**Batch Size and Learning Rate Effects (Figures 3 vs. 24):**

- Both models show that larger batch sizes and learning rates lead to faster convergence (steeper loss curves). However, the effect is more pronounced in the Sparsity=0.3 model, with the high learning rate curves dropping faster.

- The LLC plots for Sparsity=0.3 show higher peaks, especially for larger batch sizes, indicating periods of very rapid learning. This suggests that with denser inputs, the model can take better advantage of larger batches.

| | step | eval | eval_idx | llc/mean | llc/std | llc-chain/0 | llc-chain/1 | llc-chain/2 | llc-chain/3 | llc-chain/4 | ... | llc-chain/49 | loss/trace | batch_size | localization | lr | noise_level | temperature | num_draws | num_chains | num_burnin_steps |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | NaN | 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 1 | 1 | NaN | 1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 2 | 1 | NaN | 2 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 3 | 2 | NaN | 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 4 | 2 | NaN | 1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 823 | 16340 | 0.351483 | 1 | 4.158940 | 2.588783 | 8.557430 | 4.082306 | 7.949033 | 1.578098 | 2.856830 | ... | 2.314723 | [[0.51794916], [0.3118621], [0.48993146], [0.1... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 824 | 16340 | 0.434416 | 2 | 4.158940 | 2.588783 | 8.557430 | 4.082306 | 7.949033 | 1.578098 | 2.856830 | ... | 2.314723 | [[0.51794916], [0.3118621], [0.48993146], [0.1... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 825 | 20000 | 0.344714 | 0 | 4.160915 | 2.582109 | 8.441645 | 4.081877 | 8.100184 | 1.584952 | 2.849863 | ... | 2.352505 | [[0.5125502], [0.31177548], [0.49682537], [0.1... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 826 | 20000 | 0.351616 | 1 | 4.160915 | 2.582109 | 8.441645 | 4.081877 | 8.100184 | 1.584952 | 2.849863 | ... | 2.352505 | [[0.5125502], [0.31177548], [0.49682537], [0.1... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 827 | 20000 | 0.434879 | 2 | 4.160915 | 2.582109 | 8.441645 | 4.081877 | 8.100184 | 1.584952 | 2.849863 | ... | 2.352505 | [[0.5125502], [0.31177548], [0.49682537], [0.1... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |

Figure 25: **0.3 Sparsity** Dynamic Evolution of Covariance Properties During Model Training

**Covariance Evolution (Figures 4 vs. 25):**

The Sparsity=0.3 model shows higher LLC values across starting chains, consistent with faster initial learning. Eigenvalues are comparable with those of the prima example's but very different from those in the 4D model. Thus, changing dimensions might be more pronounced, here.

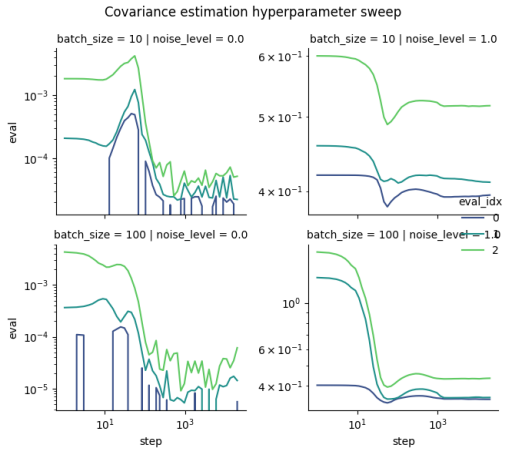



Figure 26: **0.3 Sparsity** Impact of Batch Size and Noise on Covariance Eigenvalues

**Covariance Eigenvalues (Figures 5 vs. 26):**

- The impact of noise is more pronounced in the Sparsity=0.3 model, suggesting that denser inputs make the model more sensitive to SGLD noise (at least in the start).

| | step | eval | eval_idx | draw_idx | llc/mean | llc/std | llc-chain/0 | llc-chain/1 | llc-chain/2 | llc-chain/3 | ... | llc-chain/90 | llc-chain/91 | llc-chain/92 | llc-chain/93 | llc-chain/94 | llc-chain/95 | llc-chain/96 | llc-chain/97 | llc-chain/98 | llc-chain/99 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0.084029 | 0 | 0 | -64.465393 | 5.308205 | -60.22963 | -58.841743 | -64.821426 | -66.229179 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 1 | 1 | 0.098296 | 1 | 0 | -64.465393 | 5.308205 | -60.22963 | -58.841743 | -64.821426 | -66.229179 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 2 | 1 | 0.158346 | 2 | 0 | -64.465393 | 5.308205 | -60.22963 | -58.841743 | -64.821426 | -66.229179 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 3 | 1 | 0.448311 | 0 | 1 | -64.465393 | 5.308205 | -60.22963 | -58.841743 | -64.821426 | -66.229179 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 4 | 1 | 0.663972 | 1 | 1 | -64.465393 | 5.308205 | -60.22963 | -58.841743 | -64.821426 | -66.229179 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 6895 | 20000 | 1.092776 | 1 | 8 | 2.590239 | 1.336141 | 2.15879 | 1.296228 | 2.525126 | 0.563324 | ... | 1.306979 | 3.099187 | 1.858244 | 1.383584 | 1.977158 | 3.842384 | 2.52422 | 1.768641 | 3.145313 | 3.013253 |
| 6896 | 20000 | 1.111740 | 2 | 8 | 2.590239 | 1.336141 | 2.15879 | 1.296228 | 2.525126 | 0.563324 | ... | 1.306979 | 3.099187 | 1.858244 | 1.383584 | 1.977158 | 3.842384 | 2.52422 | 1.768641 | 3.145313 | 3.013253 |
| 6897 | 20000 | 1.083243 | 0 | 9 | 2.590239 | 1.336141 | 2.15879 | 1.296228 | 2.525126 | 0.563324 | ... | 1.306979 | 3.099187 | 1.858244 | 1.383584 | 1.977158 | 3.842384 | 2.52422 | 1.768641 | 3.145313 | 3.013253 |
| 6898 | 20000 | 1.147112 | 1 | 9 | 2.590239 | 1.336141 | 2.15879 | 1.296228 | 2.525126 | 0.563324 | ... | 1.306979 | 3.099187 | 1.858244 | 1.383584 | 1.977158 | 3.842384 | 2.52422 | 1.768641 | 3.145313 | 3.013253 |
| 6899 | 20000 | 1.233712 | 2 | 9 | 2.590239 | 1.336141 | 2.15879 | 1.296228 | 2.525126 | 0.563324 | ... | 1.306979 | 3.099187 | 1.858244 | 1.383584 | 1.977158 | 3.842384 | 2.52422 | 1.768641 | 3.145313 | 3.013253 |

Figure 27: **0.3 Sparsity** Online Covariance Evolution Under Varying SGLD Sampling Parameters

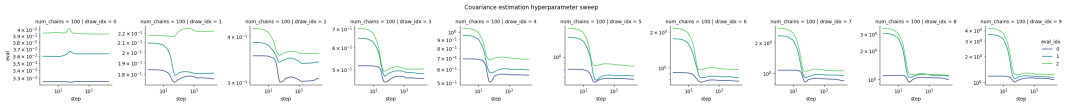



Figure 28: **0.3 Sparsity** Covariance Eigenvalue Evolution Across Online Estimation Steps

**Online Covariance Evolution (Figures 6-7 vs. 27-28):**

- In both models, increasing the number of chains leads to smoother eigenvalue estimates. However, in the Sparsity=0.3 model, the eigenvalues are closer in magnitude across all steps and chain numbers. This again suggests more uniform use of dimensions.

## Summary of Changes

In summary, reducing sparsity from 1.0 (prima example) to 0.3 leads to:

Surprisingly, not a lower final loss. Faster learning, as evidenced by higher LLC values. Less distinct developmental phases. Instead of clear line-to-V-to-L transitions, the model uses both dimensions more uniformly from early on. More uniform use of hidden dimensions throughout training, as shown by closer eigenvalue magnitudes. Increased sensitivity to SGLD noise, possibly because denser inputs provide more complex error surfaces.

These changes suggest that while sparser inputs lead to more distinct developmental phases (perhaps analogous to critical periods in biological neural development), denser inputs allow for faster, more uniform learning. This trade-off between structured development and learning speed could have implications for understanding and optimizing more complex neural networks.

### Sparsity=0.5

**Algorithm 12 0.5 Sparsity** Loss and Weight Snapshots

```
...
# old: creates sparse dataset
#dataset = SyntheticBinaryValued(num_samples, m, 1)
# new: creates dense dataset
dataset = SyntheticUniformValued(num_samples, m, 0.5) # s=0.5
...
NUM_FEATURES = 8
NUM_HIDDEN_UNITS = 2 # r
NUM_SAMPLES = 1000
NUM_EPOCHS = 20000
INIT_KGON = 2
NUM_OBSERVATIONS = 50
STEPS = sorted(list(set(np.logspace(0, np.log10(NUM_EPOCHS),
NUM_OBSERVATIONS).astype(int))))
PLOT_STEPS = [min(STEPS, key=lambda s: abs(s-i)) for i in [0, 200, 2000,
10000, NUM_EPOCHS - 1]]
PLOT_INDICES = [STEPS.index(s) for s in PLOT_STEPS]
logs, weights = create_and_train(NUM_FEATURES,
NUM_HIDDEN_UNITS, num_samples=NUM_SAMPLES,
log_ivl=STEPS, batch_size=100, lr=0.01, num_epochs=NUM_EPOCHS,
init_kgon=INIT_KGON, init_zerobias=False, seed=1)
weights_to_plot = [weights[i]['embedding.weight'] for i in PLOT_INDICES]
losses = [logs.loc[logs['step'] == s, 'loss'].values[0] for s in STEPS]
plot_losses_and_polygons(STEPS, losses, PLOT_STEPS, weights_to_plot)
plt.show()
```

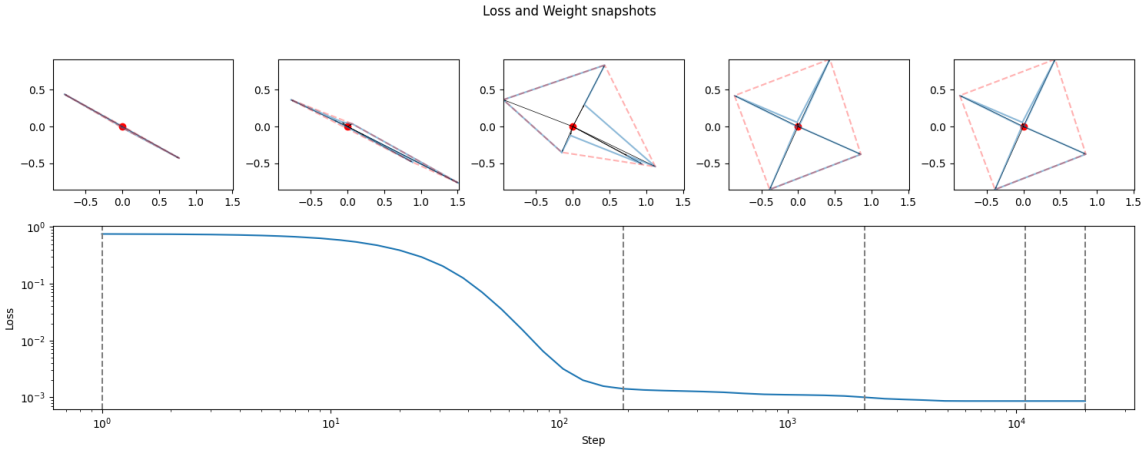



Figure 29: **0.5 Sparsity** Loss and Weight Snapshots

**Loss and Weight Snapshots:**

- Prima Example (Fig. 1): The model starts with a line, transitions to a V-shape around step 100 (steep loss drop), and finally settles into an L-shape (orthogonal vectors), indicating effective use of both dimensions.
- Sparsity=0.5 (Fig. 29): The overall pattern is similar, but there are notable differences:
  - The transition from line to V-shape happens earlier (around step 50), indicating faster adaptation to denser inputs.
  - The final weight vectors aren't perfectly orthogonal, suggesting that with denser inputs, the dimensions might not be as independently used.
  - The final weight vectors form a square instead of a pentagon. I've seen a pentagon more often in readings.

**Algorithm 13 0.5 Sparsity** Hyperparameter Sweep Results: Local Learning Coefficients

```
# Sweep SGLD hyperparameters
NUM_SAMPLES_TEST = 200
NUM_DRAWS_SGLD = 100
NUM_CHAINS_SGLD = 5
# -- old
# Create a sparse dataset
# the SyntheticBinaryValued class creates sparse binary inputs where each element is either 0 or 1,
# with a sparsity level controlled by the sparsity parameter.
"'
dataset = SyntheticBinaryValued(NUM_SAMPLES_TEST, NUM_FEATURES, 1)
dataset_double = TensorDataset(dataset.data, dataset.data)
"'
# -- end old
# -- new
# Create a dense dataset
# The SyntheticUniformValued class generates inputs where each element is a uniform random value between 0 and 1,
# regardless of the sparsity parameter.
dataset = SyntheticUniformValued(NUM_SAMPLES, NUM_FEATURES, 0.5) # s
dataset_double = TensorDataset(dataset.data, dataset.data)
# Additionally, you can modify the SyntheticDataset class to create other types of dense inputs, such as Gaussian or non-negative inputs.
# -- end new
model = ToyAutoencoder(NUM_FEATURES, NUM_HIDDEN_UNITS, final_bias=True)
...
lambdahat_sweep_df = sweep_lambdahat_estimation_hyperparams(model, dataset_double)
```

| | llc | llc/std | batch_size | lr | t_sgld | llc_type | loss |
|---|---|---|---|---|---|---|---|
| **0** | NaN | NaN | 1 | 0.00001 | 0 | mean | 0.091511 |
| **1** | -inf | NaN | 1 | 0.00001 | 0 | 0 | 0.084254 |
| **2** | -inf | NaN | 1 | 0.00001 | 0 | 1 | 0.049026 |
| **3** | inf | NaN | 1 | 0.00001 | 0 | 2 | 0.195959 |
| **4** | -inf | NaN | 1 | 0.00001 | 0 | 3 | 0.013555 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **11995** | 6.266948 | NaN | 300 | 0.01000 | 99 | 0 | 0.208353 |
| **11996** | 8.356746 | NaN | 300 | 0.01000 | 99 | 1 | 0.248086 |
| **11997** | 6.731091 | NaN | 300 | 0.01000 | 99 | 2 | 0.217178 |
| **11998** | 4.116767 | NaN | 300 | 0.01000 | 99 | 3 | 0.167473 |
| **11999** | 5.490890 | NaN | 300 | 0.01000 | 99 | 4 | 0.193598 |

Figure 30: **0.5 Sparsity** Hyperparameter Sweep Results: Local Learning Coefficients

**Hyperparameter Sweep (Local Learning Coefficients):**

- Sparsity=0.5 (Fig. 30): Similar pattern to prima example, but:
  - Higher LLCs overall, suggesting faster learning with denser inputs. Although less than Sparsity=0.3.
  - Higher loss values than sparse model, but less than the Sparsity=0.3 model (makes sense as we increase sparsity).

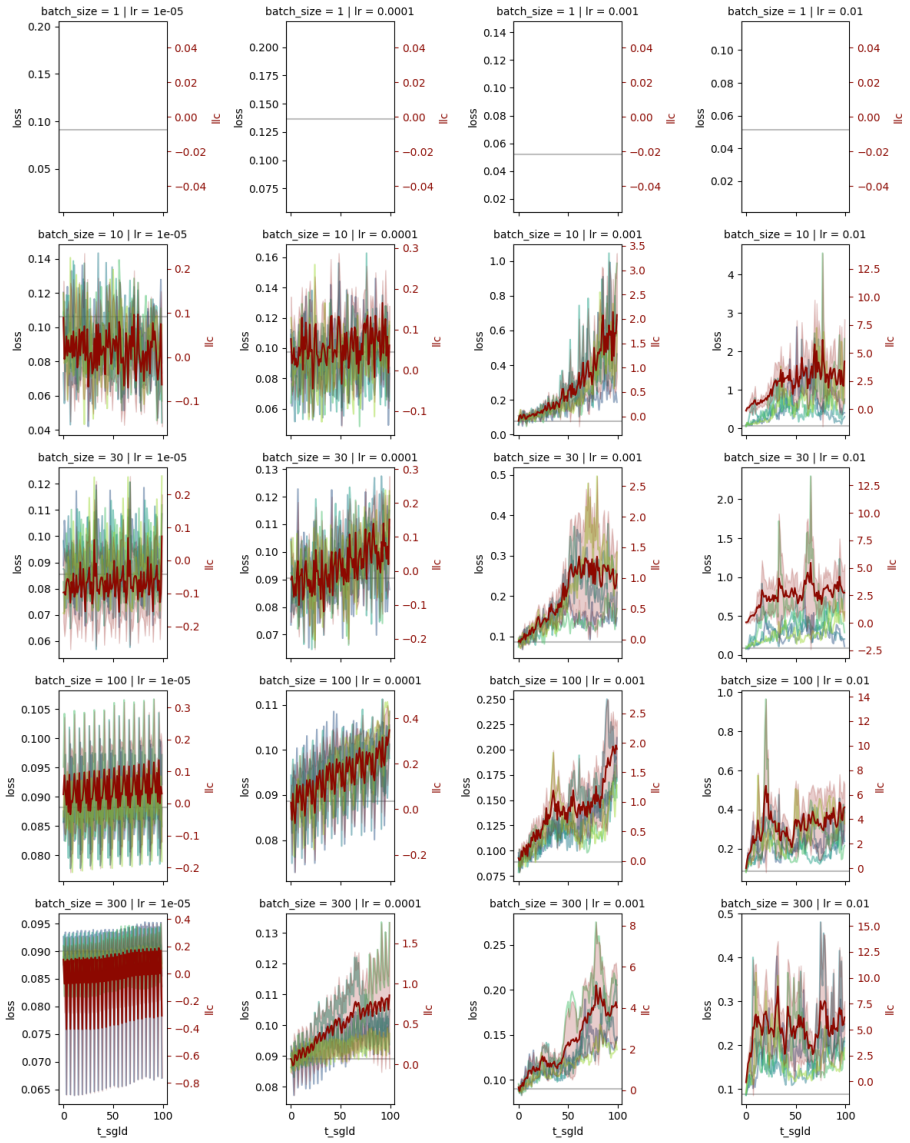


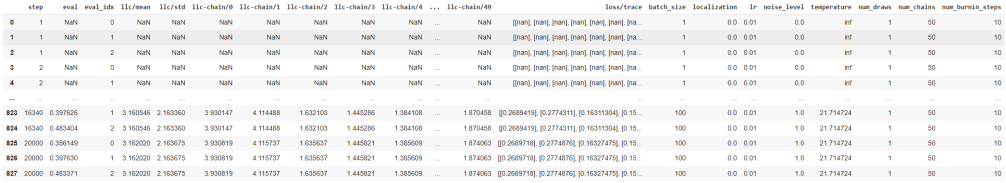

Figure 31: **0.5 Sparsity** Batch Size and Learning Rate Effect on Loss and the Local Learning Coefficient

**Batch Size and Learning Rate Effects:**

- Sparsity=0.5 (Fig. 31): Similar trends to sparse model, but:
  - Loss values are generally higher, reinforcing that denser inputs are more challenging.

– LLC values are higher, especially for smaller batches, suggesting that the model learns faster from denser, more informative batches.

– Note that these values are less pronounced than the Sparsity=0.3 model, again.



| | step | eval | eval_idx | llc/mean | llc/std | llc-chain/0 | llc-chain/1 | llc-chain/2 | llc-chain/3 | llc-chain/4 | ... | llc-chain/49 | loss/trace | batch_size | localization | lr | noise_level | temperature | num_draws | num_chains | num_burnin_steps |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | NaN | 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 1 | 1 | NaN | 1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 2 | 1 | NaN | 2 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 3 | 2 | NaN | 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 4 | 2 | NaN | 1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 823 | 16340 | 0.397626 | 1 | 3.160546 | 2.163360 | 3.930147 | 4.114488 | 1.632103 | 1.445286 | 1.384108 | ... | 1.870458 | [[0.2689419], [0.2774311], [0.163113 04], [0.15... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 824 | 16340 | 0.483404 | 2 | 3.160546 | 2.163360 | 3.930147 | 4.114488 | 1.632103 | 1.445286 | 1.384108 | ... | 1.870458 | [[0.2689419], [0.2774311], [0.163113 04], [0.15... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 825 | 20000 | 0.356149 | 0 | 3.162020 | 2.163675 | 3.930819 | 4.115737 | 1.635637 | 1.445821 | 1.385609 | ... | 1.874063 | [[0.2689718], [0.2774876], [0.163274 75], [0.15... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 826 | 20000 | 0.397630 | 1 | 3.162020 | 2.163675 | 3.930819 | 4.115737 | 1.635637 | 1.445821 | 1.385609 | ... | 1.874063 | [[0.2689718], [0.2774876], [0.163274 75], [0.15... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 827 | 20000 | 0.483371 | 2 | 3.162020 | 2.163675 | 3.930819 | 4.115737 | 1.635637 | 1.445821 | 1.385609 | ... | 1.874063 | [[0.2689718], [0.2774876], [0.163274 75], [0.15... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |

Figure 32: **0.5 Sparsity** Dynamic Evolution of Covariance Properties During Model Training

**Dynamic Evolution of Covariance (Figures 4 vs. 32):**
The Sparsity=0.5 model shows higher LLC values across starting chains in contrast with the sparse model, they are lower overall than those of the denser, Sparsity=0.3 model, however. This is consistent with our expectations.

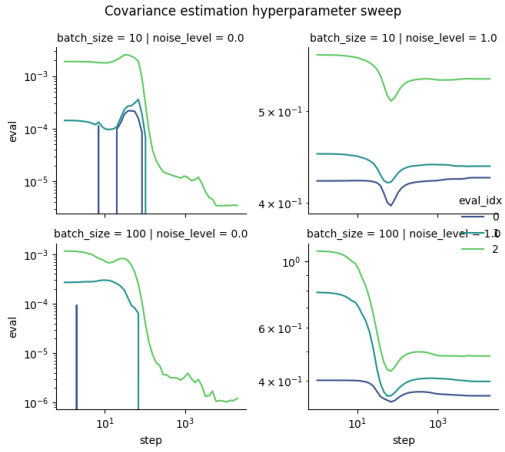



Figure 33: **0.5 Sparsity** Impact of Batch Size and Noise on Covariance Eigenvalues

**Impact of Batch Size and Noise on Covariance (Figures 5 vs. 33):**

- Prima Example: Clear distinction between eigenvalues, especially with no noise. Larger batches give smoother curves.

- Sparsity=0.: Eigenvalues are closer in magnitude, suggesting:
  - Denser inputs encourage more uniform use of dimensions.
  - The impact of noise is more pronounced in the Sparsity=0.3 model, and less so here. Again, the model is more sensitive to SGLD noise than the sparser model, but not as much as the previous example.

| | step | eval | eval_idx | draw_idx | llc/mean | llc/std | llc-chain/0 | llc-chain/1 | llc-chain/2 | llc-chain/3 | ... | llc-chain/90 | llc-chain/91 | llc-chain/92 | llc-chain/93 | llc-chain/94 | llc-chain/95 | llc-chain/96 | llc-chain/97 | llc-chain/98 | llc-chain/99 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0.083434 | 0 | 0 | -64.602211 | 5.414911 | -58.265015 | -59.751274 | -65.913063 | -68.296906 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 1 | 1 | 0.098928 | 1 | 0 | -64.602211 | 5.414911 | -58.265015 | -59.751274 | -65.913063 | -68.296906 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 2 | 1 | 0.159614 | 2 | 0 | -64.602211 | 5.414911 | -58.265015 | -59.751274 | -65.913063 | -68.296906 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 3 | 1 | 0.462323 | 0 | 1 | -64.602211 | 5.414911 | -58.265015 | -59.751274 | -65.913063 | -68.296906 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 4 | 1 | 0.691285 | 1 | 1 | -64.602211 | 5.414911 | -58.265015 | -59.751274 | -65.913063 | -68.296906 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 6895 | 20000 | 1.219280 | 1 | 8 | 1.909200 | 1.057315 | 1.661757 | 1.712940 | 1.633504 | 0.593781 | ... | 0.881036 | 1.60891 | 1.372497 | 1.260412 | 0.704683 | 1.158151 | 2.134599 | 2.034436 | 1.946654 | 4.66144 |
| 6896 | 20000 | 1.223005 | 2 | 8 | 1.909200 | 1.057315 | 1.661757 | 1.712940 | 1.633504 | 0.593781 | ... | 0.881036 | 1.60891 | 1.372497 | 1.260412 | 0.704683 | 1.158151 | 2.134599 | 2.034436 | 1.946654 | 4.66144 |
| 6897 | 20000 | 1.173231 | 0 | 9 | 1.909200 | 1.057315 | 1.661757 | 1.712940 | 1.633504 | 0.593781 | ... | 0.881036 | 1.60891 | 1.372497 | 1.260412 | 0.704683 | 1.158151 | 2.134599 | 2.034436 | 1.946654 | 4.66144 |
| 6898 | 20000 | 1.292375 | 1 | 9 | 1.909200 | 1.057315 | 1.661757 | 1.712940 | 1.633504 | 0.593781 | ... | 0.881036 | 1.60891 | 1.372497 | 1.260412 | 0.704683 | 1.158151 | 2.134599 | 2.034436 | 1.946654 | 4.66144 |
| 6899 | 20000 | 1.368300 | 2 | 9 | 1.909200 | 1.057315 | 1.661757 | 1.712940 | 1.633504 | 0.593781 | ... | 0.881036 | 1.60891 | 1.372497 | 1.260412 | 0.704683 | 1.158151 | 2.134599 | 2.034436 | 1.946654 | 4.66144 |

Figure 34: **0.5 Sparsity** Online Covariance Evolution Under Varying SGLD Sampling Parameters

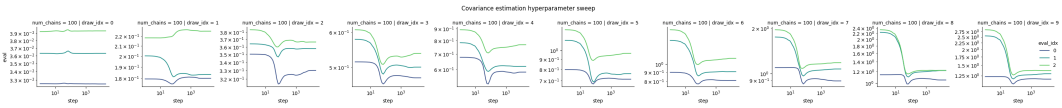



Figure 35: **0.5 Sparsity** Covariance Eigenvalue Evolution Across Online Estimation Steps

**Online Covariance Evolution (Figures 6-7 vs. 34-35):**

- Prima Example: Eigenvalues stabilize quickly, with a clear hierarchy (one dominant).
- Sparsity=0.5:
  - Faster stabilization of eigenvalues, indicating quicker convergence of internal representations with denser data.
  - Less hierarchy among eigenvalues, reinforcing the idea that dimensions are used more uniformly.

## Sparsity=0.7

I think we all know what results to expect here, given the patterns noted above. For completeness and curiosity's sake, I will leave these results below. Observe how closely it mimics the development of the initial, sparse model.

**Algorithm 14 0.7 Sparsity** Loss and Weight Snapshots

```
...
# old: creates sparse dataset
#dataset = SyntheticBinaryValued(num_samples, m, 1)
# new: creates dense dataset
dataset = SyntheticUniformValued(num_samples, m, 0.7) # s=0.7
...
NUM_FEATURES = 8
NUM_HIDDEN_UNITS = 2 # r
NUM_SAMPLES = 1000
NUM_EPOCHS = 20000
INIT_KGON = 2
NUM_OBSERVATIONS = 50
STEPS = sorted(list(set(np.logspace(0, np.log10(NUM_EPOCHS),
NUM_OBSERVATIONS).astype(int))))
PLOT_STEPS = [min(STEPS, key=lambda s: abs(s-i)) for i in [0, 200, 2000,
10000, NUM_EPOCHS - 1]]
PLOT_INDICES = [STEPS.index(s) for s in PLOT_STEPS]
logs, weights = create_and_train(NUM_FEATURES,
NUM_HIDDEN_UNITS, num_samples=NUM_SAMPLES,
log_ivl=STEPS, batch_size=100, lr=0.01, num_epochs=NUM_EPOCHS,
init_kgon=INIT_KGON, init_zerobias=False, seed=1)
weights_to_plot = [weights[i]['embedding.weight'] for i in PLOT_INDICES]
losses = [logs.loc[logs['step'] == s, 'loss'].values[0] for s in STEPS]
plot_losses_and_polygons(STEPS, losses, PLOT_STEPS, weights_to_plot)
plt.show()
```

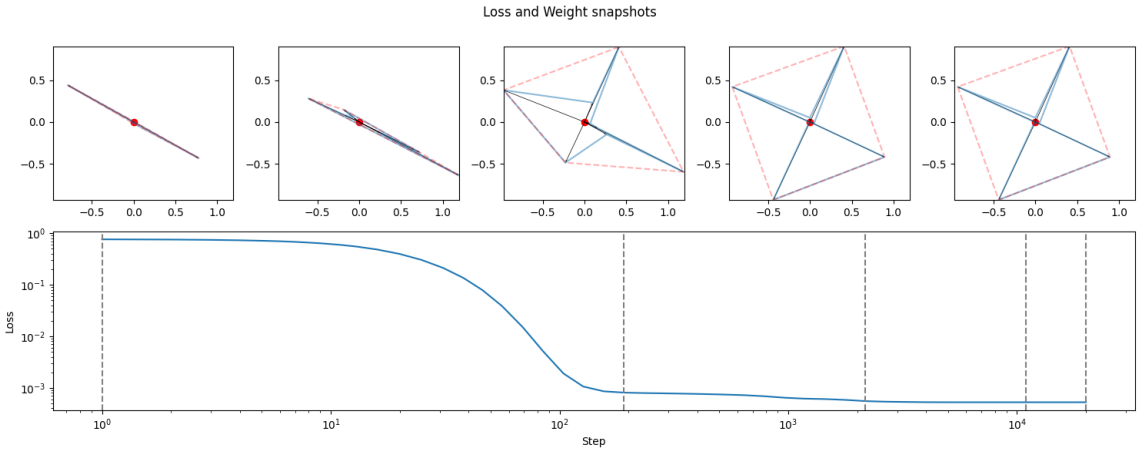



Figure 36: **0.7 Sparsity** Loss and Weight Snapshots

**Loss and Weight Snapshots (Figures 1 vs. 36):**

The prima example shows clear transitions: line → V-shape → L-shape. The Sparsity=0.7 model also starts with a line, but its subsequent transitions are less pronounced. It quickly spreads out without a clear V-shape, and the final configuration isn't a perfect L. This indicates that with denser inputs, the model uses both dimensions more uniformly from early on, without distinct developmental phases.

| | llc | llc/std | batch_size | lr | t_sgld | llc_type | loss |
|---|---|---|---|---|---|---|---|
| **0** | NaN | NaN | 1 | 0.00001 | 0 | mean | 0.036078 |
| **1** | inf | NaN | 1 | 0.00001 | 0 | 0 | 0.054543 |
| **2** | -inf | NaN | 1 | 0.00001 | 0 | 1 | 0.005244 |
| **3** | inf | NaN | 1 | 0.00001 | 0 | 2 | 0.089273 |
| **4** | -inf | NaN | 1 | 0.00001 | 0 | 3 | 0.025831 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **11995** | 4.985157 | NaN | 300 | 0.01000 | 99 | 0 | 0.148153 |
| **11996** | 9.601656 | NaN | 300 | 0.01000 | 99 | 1 | 0.235925 |
| **11997** | 3.429438 | NaN | 300 | 0.01000 | 99 | 2 | 0.118575 |
| **11998** | 3.030287 | NaN | 300 | 0.01000 | 99 | 3 | 0.110986 |
| **11999** | 4.626709 | NaN | 300 | 0.01000 | 99 | 4 | 0.141338 |

Figure 37: **0.7 Sparsity** Hyperparameter Sweep Results: Local Learning Coefficients

**Hyperparameter Sweep: Local Learning Coefficients (Compare Figures 2 vs. 37)**

**Algorithm 15 0.7 Sparsity** Hyperparameter Sweep Results: Local Learning Coefficients

```
# Sweep SGLD hyperparameters
NUM_SAMPLES_TEST = 200
NUM_DRAWS_SGLD = 100
NUM_CHAINS_SGLD = 5
# -- old
# Create a sparse dataset
# the SyntheticBinaryValued class creates sparse binary inputs where each element is either 0 or 1,
# with a sparsity level controlled by the sparsity parameter.
"'
dataset = SyntheticBinaryValued(NUM_SAMPLES_TEST, NUM_FEATURES, 1)
dataset_double = TensorDataset(dataset.data, dataset.data)
"'
# -- end old
# -- new
# Create a dense dataset
# The SyntheticUniformValued class generates inputs where each element is a uniform random value between 0 and 1,
# regardless of the sparsity parameter.
dataset = SyntheticUniformValued(NUM_SAMPLES, NUM_FEATURES, 0.7) # s
dataset_double = TensorDataset(dataset.data, dataset.data)
# Additionally, you can modify the SyntheticDataset class to create other types of dense inputs, such as Gaussian or non-negative inputs.
# -- end new
model = ToyAutoencoder(NUM_FEATURES, NUM_HIDDEN_UNITS, final_bias=True)
...
lambdahat_sweep_df = sweep_lambdahat_estimation_hyperparams(model, dataset_double)
```

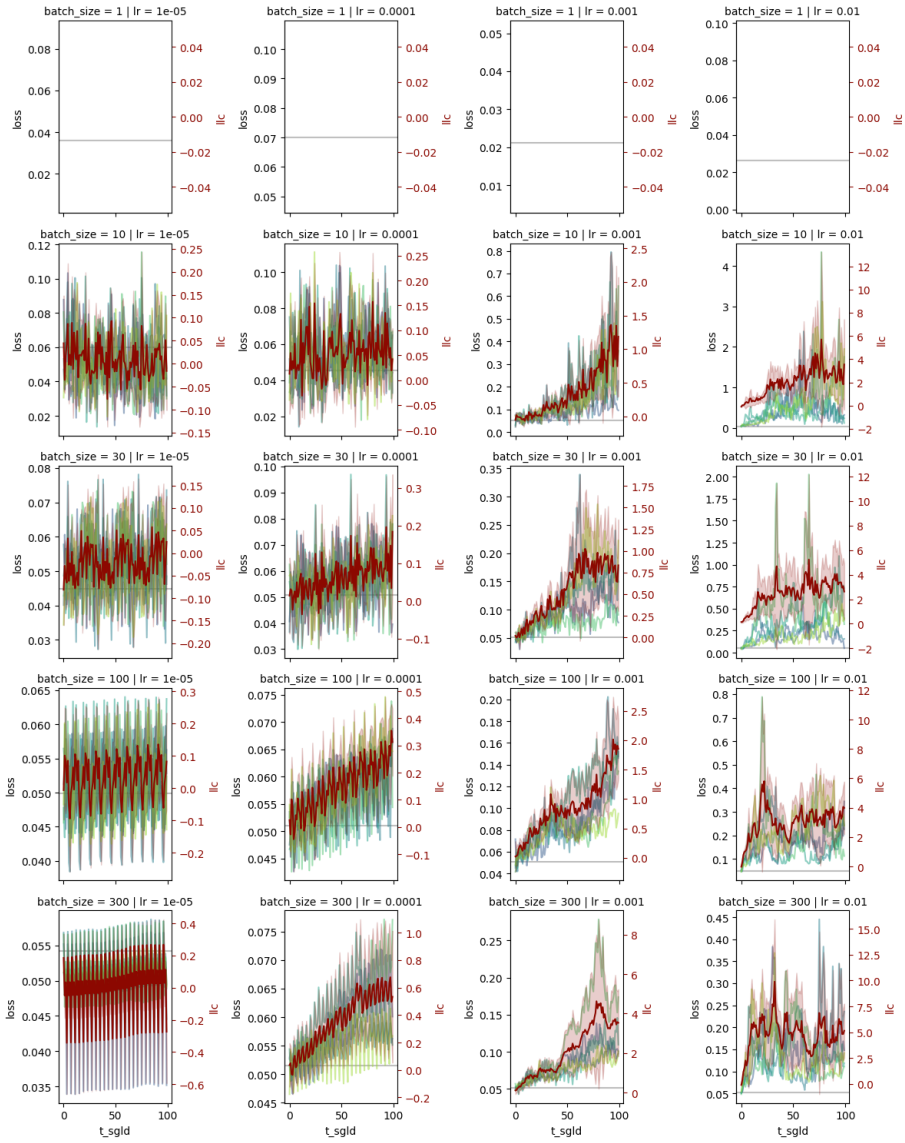



Figure 38: **0.7 Sparsity** Batch Size and Learning Rate Effect on Loss and the Local Learning Coefficient

**Dynamic Evolution of Covariance (Compare Figures 4 vs. 39)**

| | step | eval | eval_idx | llc/mean | llc/std | llc-chain/0 | llc-chain/1 | llc-chain/2 | llc-chain/3 | llc-chain/4 | ... | llc-chain/49 | loss/trace | batch_size | localization | lr | noise_level | temperature | num_draws | num_chains | num_burnin_steps |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | NaN | 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 1 | 1 | NaN | 1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 2 | 1 | NaN | 2 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 3 | 2 | NaN | 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 4 | 2 | NaN | 1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 823 | 16340 | 0.404138 | 1 | 2.603506 | 1.735864 | 6.177086 | 1.950513 | 2.389017 | 1.523358 | 1.420384 | ... | 1.318095 | [[0.33542687], [0.14078598], [0.16097984], [0.... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 824 | 16340 | 0.506060 | 2 | 2.603506 | 1.735864 | 6.177086 | 1.950513 | 2.389017 | 1.523358 | 1.420384 | ... | 1.318095 | [[0.33542687], [0.14078598], [0.16097984], [0.... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 825 | 20000 | 0.348492 | 0 | 2.604281 | 1.735896 | 6.180064 | 1.951340 | 2.388319 | 1.522528 | 1.421108 | ... | 1.318249 | [[0.3355632], [0.14082329], [0.16094692], [0.1... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 826 | 20000 | 0.404144 | 1 | 2.604281 | 1.735896 | 6.180064 | 1.951340 | 2.388319 | 1.522528 | 1.421108 | ... | 1.318249 | [[0.3355632], [0.14082329], [0.16094692], [0.1... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 827 | 20000 | 0.506033 | 2 | 2.604281 | 1.735896 | 6.180064 | 1.951340 | 2.388319 | 1.522528 | 1.421108 | ... | 1.318249 | [[0.3355632], [0.14082329], [0.16094692], [0.1... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |

Figure 39: **0.7 Sparsity** Dynamic Evolution of Covariance Properties During Model Training

**Dynamic Evolution of Covariance (Compare Figures 4 vs. 39)**

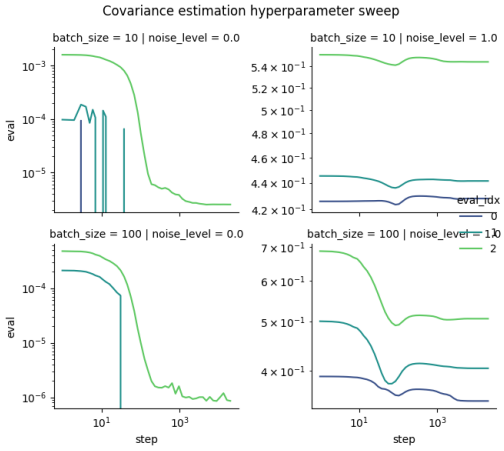



Figure 40: **0.7 Sparsity** Impact of Batch Size and Noise on Covariance Eigenvalues

**Impact of Batch Size and Noise on Covariance (Compare Figures 5 vs. 40)**

| | step | eval | eval_idx | draw_idx | llc/mean | llc/std | llc-chain/0 | llc-chain/1 | llc-chain/2 | llc-chain/3 | ... | llc-chain/90 | llc-chain/91 | llc-chain/92 | llc-chain/93 | llc-chain/94 | llc-chain/95 | llc-chain/96 | llc-chain/97 | llc-chain/98 | llc-chain/99 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0.083811 | 0 | 0 | -63.600121 | 5.556818 | -57.454704 | -57.936604 | -65.373375 | -68.327072 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 1 | 1 | 0.098263 | 1 | 0 | -63.600121 | 5.556818 | -57.454704 | -57.936604 | -65.373375 | -68.327072 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 2 | 1 | 0.158044 | 2 | 0 | -63.600121 | 5.556818 | -57.454704 | -57.936604 | -65.373375 | -68.327072 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 3 | 1 | 0.482913 | 0 | 1 | -63.600121 | 5.556818 | -57.454704 | -57.936604 | -65.373375 | -68.327072 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 4 | 1 | 0.725816 | 1 | 1 | -63.600121 | 5.556818 | -57.454704 | -57.936604 | -65.373375 | -68.327072 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 6895 | 20000 | 1.232444 | 1 | 8 | 1.560073 | 0.931168 | 0.762478 | 1.319593 | 1.490588 | 0.494455 | ... | 0.792295 | 1.401317 | 1.115182 | 1.164893 | 0.437991 | 0.863472 | 2.942998 | 1.739244 | 2.088552 | 3.826744 |
| 6896 | 20000 | 1.247154 | 2 | 8 | 1.560073 | 0.931168 | 0.762478 | 1.319593 | 1.490588 | 0.494455 | ... | 0.792295 | 1.401317 | 1.115182 | 1.164893 | 0.437991 | 0.863472 | 2.942998 | 1.739244 | 2.088552 | 3.826744 |
| 6897 | 20000 | 1.208567 | 0 | 9 | 1.560073 | 0.931168 | 0.762478 | 1.319593 | 1.490588 | 0.494455 | ... | 0.792295 | 1.401317 | 1.115182 | 1.164893 | 0.437991 | 0.863472 | 2.942998 | 1.739244 | 2.088552 | 3.826744 |
| 6898 | 20000 | 1.319814 | 1 | 9 | 1.560073 | 0.931168 | 0.762478 | 1.319593 | 1.490588 | 0.494455 | ... | 0.792295 | 1.401317 | 1.115182 | 1.164893 | 0.437991 | 0.863472 | 2.942998 | 1.739244 | 2.088552 | 3.826744 |
| 6899 | 20000 | 1.405367 | 2 | 9 | 1.560073 | 0.931168 | 0.762478 | 1.319593 | 1.490588 | 0.494455 | ... | 0.792295 | 1.401317 | 1.115182 | 1.164893 | 0.437991 | 0.863472 | 2.942998 | 1.739244 | 2.088552 | 3.826744 |

Figure 41: **0.7 Sparsity** Online Covariance Evolution Under Varying SGLD Sampling Parameters

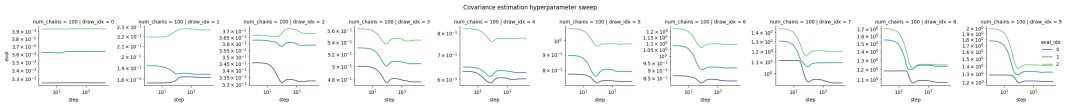



Figure 42: **0.7 Sparsity** Covariance Eigenvalue Evolution Across Online Estimation Steps

**Online Covariance Evolution (Figures 6 & 7 vs. 41 & 42):**

- Prima Example: Quick stabilization of eigenvalues, with a clear hierarchy (one dominant).
- Sparsity=0.7:
  - Eigenvalues are nearly identical from the start, reinforcing that both dimensions are used equally and consistently throughout training.

# Summary in Changes

1. Learning Speed: The model learns faster with denser inputs (Sparsity=0.3), as evidenced by higher LLCs and earlier transitions in weight snapshots. This suggests that more informative inputs allow for quicker adaptation.
2. Learning Difficulty: Despite faster learning, the overall loss is higher with denser inputs. This indicates that while the model can extract information more quickly, the task of representing denser data is inherently more challenging.
3. Dimensional Usage: In the prima example, the model clearly separates inputs into two orthogonal dimensions. With denser inputs, this separation is less stark. This could mean that:
   - Features in denser inputs have more complex inter-relationships, making clean separation harder.
   - The model uses both dimensions more uniformly to capture the richer information in denser inputs.
4. Critical Periods: Both models show clear developmental transitions (e.g., line to V to L), but these happen earlier with denser inputs. This suggests that the "critical periods" of neural development can be accelerated by more informative data.

In summary, increasing input density (reducing sparsity) leads to more stable learning, but also makes the learning task more challenging. The model's internal representations adapt to use dimensions more uniformly, possibly to

capture richer inter-feature relationships. These findings have implications for model design and data preparation: denser inputs might lead to quicker training and more robust models, but may also require more capacity (e.g., more hidden dimensions) to achieve low loss. It's important to note that these insights come from a toy model, and their applicability to large-scale models should be validated. However, they provide valuable intuitions about how data properties can fundamentally alter a neural network's developmental trajectory.

# Feature Importance

In this section, we go over results after assigning random weights to features, as previously all features had equal importance.

FEATURE_IMPORTANCE1 = [1.0, 0.8, 0.6, 0.4, 0.2, 0.1, 0.05, 0.01]
FEATURE_IMPORTANCE2 = [0.3, 0.41, 0.007, 0.44, 0.7, 0.89, 0.99, 0.23]

## Feature Importance Set 1

**Algorithm 16 1st Set Feature Importance** Loss and Weight Snapshots, part I

```
class SyntheticDataset(Dataset, ABC):
num_samples: int
num_features: int
sparsity: Union[float, int]
importance: Optional[float] # commented out in original script (no importance)
...
# new
self.importance = importance if importance is not None else np.ones(num_features)
self.data = self.generate_data() # Generate the synthetic data "'
# old
def generate_values(self):
raise NotImplementedError
"'
# new
"'
Here, we're generating random values between 0 and 1 and then multiplying them
element-wise with the importance array to scale the feature values according to their importance.
"'
def generate_values(self):
values = np.random.rand(self.num_samples, self.num_features)
return values * self.importance
...
)
```

**Algorithm 17 1st Set Feature Importance** Loss and Weight Snapshots, part II

```
def create_and_train(
...  # old
# dataset = SyntheticBinaryValued(num_samples, m, 1)
# new
# Create a dataset with varying feature importance
dataset = SyntheticUniformValued(NUM_SAMPLES, NUM_FEATURES,
0.5, importance=FEATURE_IMPORTANCE)
#dataset_double = TensorDataset(dataset.data, dataset.data)
...
) NUM_FEATURES = 8
NUM_HIDDEN_UNITS = 2 # r
NUM_SAMPLES = 1000
NUM_EPOCHS = 20000
INIT_KGON = 2
NUM_OBSERVATIONS = 50
# Define the importance of each feature
FEATURE_IMPORTANCE = np.array([1.0, 0.8, 0.6, 0.4, 0.2, 0.1, 0.05, 0.01])
# new
STEPS = sorted(list(set(np.logspace(0, np.log10(NUM_EPOCHS),
NUM_OBSERVATIONS).astype(int))))
PLOT_STEPS = [min(STEPS, key=lambda s: abs(s-i)) for i in [0, 200, 2000,
10000, NUM_EPOCHS - 1]]
PLOT_INDICES = [STEPS.index(s) for s in PLOT_STEPS]
logs, weights = create_and_train(NUM_FEATURES,
NUM_HIDDEN_UNITS, num_samples=NUM_SAMPLES,
log_ivl=STEPS, batch_size=100, lr=0.01, num_epochs=NUM_EPOCHS,
init_kgon=INIT_KGON, init_zerobias=False, seed=1)
weights_to_plot = [weights[i]['embedding.weight'] for i in PLOT_INDICES]
losses = [logs.loc[logs['step'] == s, 'loss'].values[0] for s in STEPS]
plot_losses_and_polygons(STEPS, losses, PLOT_STEPS, weights_to_plot)
plt.show()
```

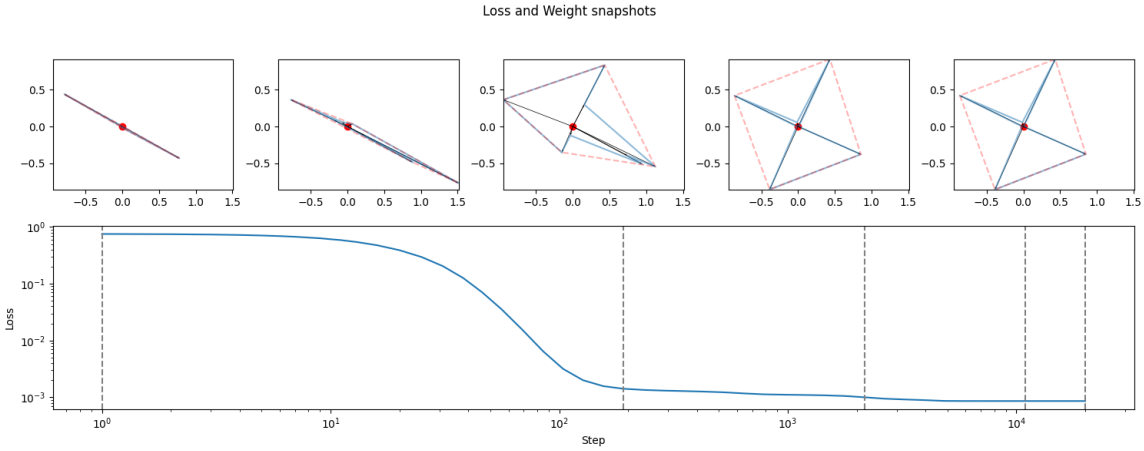



Figure 43: **1st Set Feature Importance** Loss and Weight Snapshots

**Loss and Weight Snapshots (Figures 1 vs. 43):**

- Loss Curve: Both models show rapid loss decrease early in training. However, the Feature Importance Set 1 model (FI-1) reaches a higher final loss compared to the prima example. This suggests that having a clear hierarchy of feature importance doesn't always allow the model to fit the data more accurately.
- Weight Snapshots: The prima example shows clear transitions: line → V-shape → L-shape. The Feature Importance Set 1 model also starts with a line, but its subsequent development is strikingly different:
  - It quickly transitions to a "fan" shape, with one vector (likely representing the most important features) dominating.
  - The final configuration is an uneven V, not an L. This indicates that with weighted features, the model prioritizes one dimension (for high-weight features) while using the other dimension less, possibly for low-weight features.

**Algorithm 18 1st Set Feature Importance** Hyperparameter Sweep Results: Local Learning Coefficients

```
# Sweep SGLD hyperparameters
NUM_SAMPLES_TEST = 200
NUM_DRAWS_SGLD = 100
NUM_CHAINS_SGLD = 5
# old
"'
dataset = SyntheticBinaryValued(NUM_SAMPLES_TEST, NUM_FEATURES, 1)
dataset_double = TensorDataset(dataset.data, dataset.data)
"'
# new
# Create a dataset with varying feature importance
dataset = SyntheticUniformValued(NUM_SAMPLES, NUM_FEATURES, 0.5, importance=FEATURE_IMPORTANCE)
dataset_double = TensorDataset(dataset.data, dataset.data)
model = ToyAutoencoder(NUM_FEATURES, NUM_HIDDEN_UNITS, final_bias=True)
...
lambdahat_sweep_df = sweep_lambdahat_estimation_hyperparams(model, dataset_double)
```

| | llc | llc/std | batch_size | lr | t_sgld | llc_type | loss |
|---|---|---|---|---|---|---|---|
| **0** | NaN | NaN | 1 | 0.00001 | 0 | mean | 0.091511 |
| **1** | -inf | NaN | 1 | 0.00001 | 0 | 0 | 0.084254 |
| **2** | -inf | NaN | 1 | 0.00001 | 0 | 1 | 0.049026 |
| **3** | inf | NaN | 1 | 0.00001 | 0 | 2 | 0.195959 |
| **4** | -inf | NaN | 1 | 0.00001 | 0 | 3 | 0.013555 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **11995** | 6.266948 | NaN | 300 | 0.01000 | 99 | 0 | 0.208353 |
| **11996** | 8.356746 | NaN | 300 | 0.01000 | 99 | 1 | 0.248086 |
| **11997** | 6.731091 | NaN | 300 | 0.01000 | 99 | 2 | 0.217178 |
| **11998** | 4.116767 | NaN | 300 | 0.01000 | 99 | 3 | 0.167473 |
| **11999** | 5.490890 | NaN | 300 | 0.01000 | 99 | 4 | 0.193598 |

12000 rows × 7 columns

Figure 44: **1st Set Feature Importance** Hyperparameter Sweep Results: Local Learning Coefficients

**Hyperparameter Sweep: Local Learning Coefficients (Figures 2 vs. 44):**

- Feature Importance Set 1: A notable shift in optimal settings:
  - Loss is overall higher, here.

- Highest LLCs occur with larger batch sizes (300) and a wider range of learning rates.
- This suggests that with weighted features, the model learns faster when it can see a larger, more representative sample of the weighted data distribution.

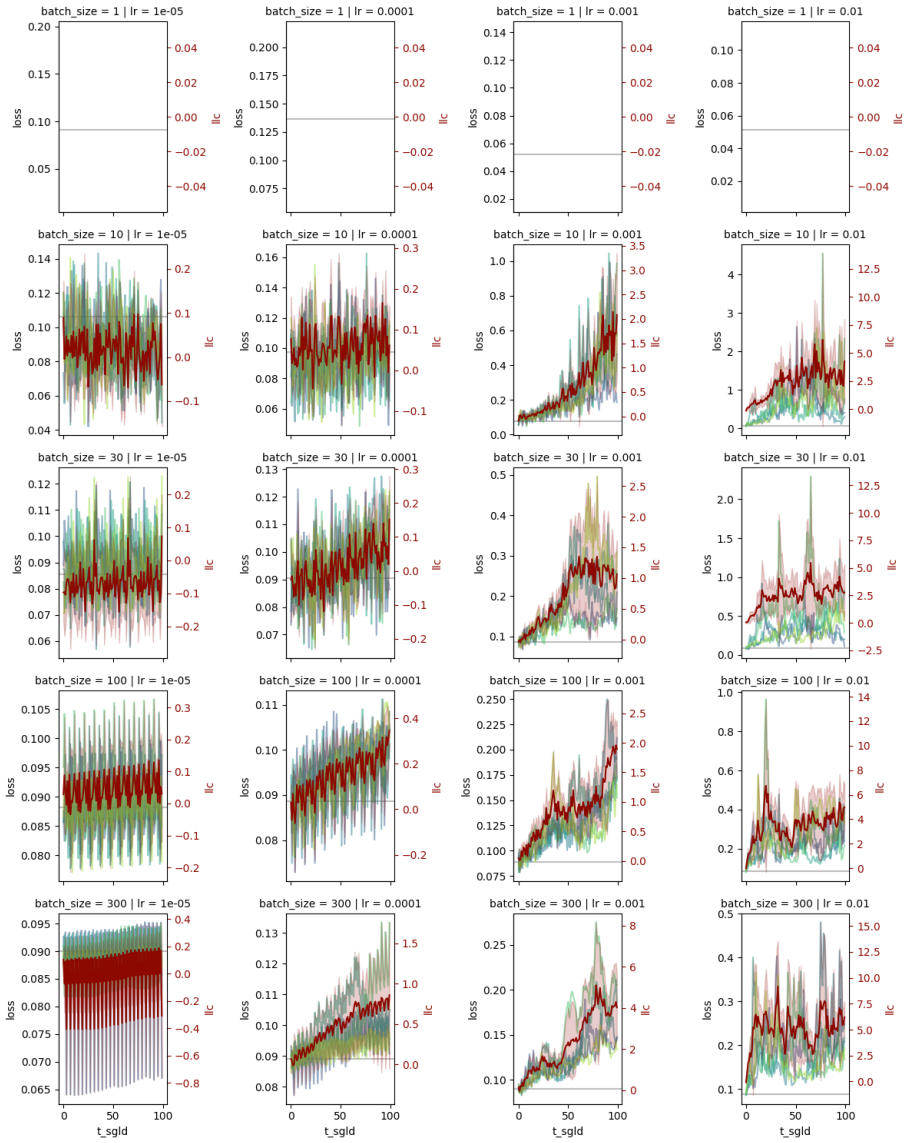



Figure 45: **1st Set Feature Importance** Batch Size and Learning Rate Effect on Loss and the Local Learning Coefficient

**Batch Size and Learning Rate Effects (Figures 3 vs. 45):**

- Prima Example: Smaller batches and moderate learning rates lead to lower loss and alternating LLCs.
- Feature Importance Set 1:

– The LLC curves show less pronounced peaks but higher overall values, suggesting that learning is fast and consistent when the model can leverage the feature importance hierarchy in larger batches. Loss is higher, though.

| | step | eval | eval_idx | llc/mean | llc/std | llc-chain/0 | llc-chain/1 | llc-chain/2 | llc-chain/3 | llc-chain/4 | ... | llc-chain/49 | loss/trace | batch_size | localization | lr | noise_level | temperature | num_draws | num_chains | num_burnin_steps |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | NaN | 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 1 | 1 | NaN | 1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 2 | 1 | NaN | 2 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 3 | 2 | NaN | 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| 4 | 2 | NaN | 1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | [[nan], [nan], [nan], [nan], [nan], [nan], [na... | 1 | 0.0 | 0.01 | 0.0 | inf | 1 | 50 | 10 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 823 | 16340 | 0.397626 | 1 | 3.160546 | 2.163360 | 3.930147 | 4.114488 | 1.632103 | 1.445286 | 1.384108 | ... | 1.870458 | [[0.2689419], [0.2774311], [0.16311304], [0.15... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 824 | 16340 | 0.483404 | 2 | 3.160546 | 2.163360 | 3.930147 | 4.114488 | 1.632103 | 1.445286 | 1.384108 | ... | 1.870458 | [[0.2689419], [0.2774311], [0.16311304], [0.15... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 825 | 20000 | 0.356149 | 0 | 3.162020 | 2.163675 | 3.930819 | 4.115737 | 1.635637 | 1.445821 | 1.385609 | ... | 1.874063 | [[0.2689718], [0.2774876], [0.16327475], [0.15... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 826 | 20000 | 0.397630 | 1 | 3.162020 | 2.163675 | 3.930819 | 4.115737 | 1.635637 | 1.445821 | 1.385609 | ... | 1.874063 | [[0.2689718], [0.2774876], [0.16327475], [0.15... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |
| 827 | 20000 | 0.483371 | 2 | 3.162020 | 2.163675 | 3.930819 | 4.115737 | 1.635637 | 1.445821 | 1.385609 | ... | 1.874063 | [[0.2689718], [0.2774876], [0.16327475], [0.15... | 100 | 0.0 | 0.01 | 1.0 | 21.714724 | 1 | 50 | 10 |

Figure 46: **1st Set Feature Importance** Dynamic Evolution of Covariance Properties During Model Training

**Dynamic Evolution of Covariance (Figures 4 vs. 46):**

The FI-1 model shows lower LLC values across starting chains. Eigenvalues are comparable with those of the prima example's. Moreover, adding feature importance produced a decreased learning rate, later on. Perhaps due to the early emphasis it made on certain features.

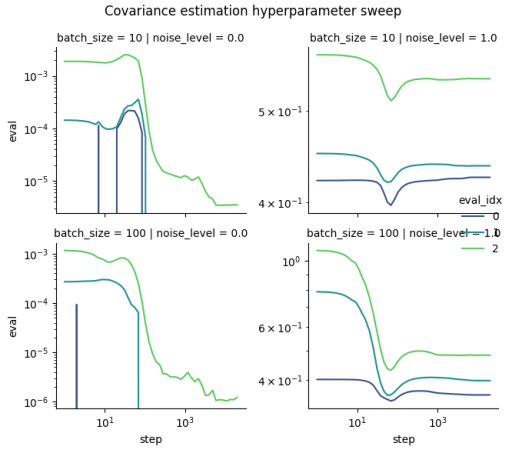



Figure 47: **1st Set Feature Importance** Impact of Batch Size and Noise on Covariance Eigenvalues

**Impact of Batch Size and Noise on Covariance (Figures 5 vs. 47):**

- Feature Importance Set 1 Differences:

– Consistently large gap between the first and other eigenvalues, regardless of batch size or noise. This underscores that feature weighting induces a persistent, dominant dimension in the model's internal representation.

– Noise emphasizes this dominance (view that larger range in the y-axis than that of the prima example's), suggesting that the feature importance signal is strong enough to overshadow stochastic effects.

| | step | eval | eval_idx | draw_idx | llc/mean | llc/std | llc-chain/0 | llc-chain/1 | llc-chain/2 | llc-chain/3 | ... | llc-chain/90 | llc-chain/91 | llc-chain/92 | llc-chain/93 | llc-chain/94 | llc-chain/95 | llc-chain/96 | llc-chain/97 | llc-chain/98 | llc-chain/99 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0.083434 | 0 | 0 | -64.602211 | 5.414911 | -58.265015 | -59.751274 | -65.913063 | -68.296906 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 1 | 1 | 0.098926 | 1 | 0 | -64.602211 | 5.414911 | -58.265015 | -59.751274 | -65.913063 | -68.296906 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 2 | 1 | 0.159614 | 2 | 0 | -64.602211 | 5.414911 | -58.265015 | -59.751274 | -65.913063 | -68.296906 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 3 | 1 | 0.462323 | 0 | 1 | -64.602211 | 5.414911 | -58.265015 | -59.751274 | -65.913063 | -68.296906 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 4 | 1 | 0.691285 | 1 | 1 | -64.602211 | 5.414911 | -58.265015 | -59.751274 | -65.913063 | -68.296906 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 6895 | 20000 | 1.219281 | 1 | 8 | 1.909200 | 1.057315 | 1.661757 | 1.712940 | 1.633504 | 0.593781 | ... | 0.881036 | 1.60891 | 1.372497 | 1.260412 | 0.704683 | 1.158151 | 2.134599 | 2.034436 | 1.946654 | 4.66144 |
| 6896 | 20000 | 1.223005 | 2 | 8 | 1.909200 | 1.057315 | 1.661757 | 1.712940 | 1.633504 | 0.593781 | ... | 0.881036 | 1.60891 | 1.372497 | 1.260412 | 0.704683 | 1.158151 | 2.134599 | 2.034436 | 1.946654 | 4.66144 |
| 6897 | 20000 | 1.173230 | 0 | 9 | 1.909200 | 1.057315 | 1.661757 | 1.712940 | 1.633504 | 0.593781 | ... | 0.881036 | 1.60891 | 1.372497 | 1.260412 | 0.704683 | 1.158151 | 2.134599 | 2.034436 | 1.946654 | 4.66144 |
| 6898 | 20000 | 1.292375 | 1 | 9 | 1.909200 | 1.057315 | 1.661757 | 1.712940 | 1.633504 | 0.593781 | ... | 0.881036 | 1.60891 | 1.372497 | 1.260412 | 0.704683 | 1.158151 | 2.134599 | 2.034436 | 1.946654 | 4.66144 |
| 6899 | 20000 | 1.368300 | 2 | 9 | 1.909200 | 1.057315 | 1.661757 | 1.712940 | 1.633504 | 0.593781 | ... | 0.881036 | 1.60891 | 1.372497 | 1.260412 | 0.704683 | 1.158151 | 2.134599 | 2.034436 | 1.946654 | 4.66144 |

Figure 48: **1st Set Feature Importance** Online Covariance Evolution Under Varying SGLD Sampling Parameters

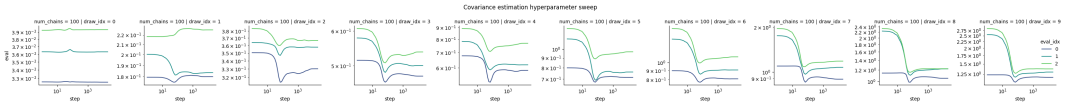



Figure 49: **1st Set Feature Importance** Covariance Eigenvalue Evolution Across Online Estimation Steps

**Online Covariance Evolution (Figures 6 & 7 vs. 48 & 49):**
Again, the range in eigenvalue valuations is more pronounced here, than in the prima example's eigenvalue indices. This is due to the feature importance values we assigned, most probably. We will see if this is still the case when we make another model with completely random weights.

## Feature Importance Set 2

I double checked our feature importance findings in this final model creation. Note that I got extremely similar results and therefore omitted the rest of this section. I am also afraid I did something wrong as I was expecting differences and will therefore dedicate more time into it before placing any strange results, here.

**Algorithm 19 2nd Set Feature Importance** Loss and Weight Snapshots

```
NUM_FEATURES = 8
NUM_HIDDEN_UNITS = 2 # r
NUM_SAMPLES = 1000
NUM_EPOCHS = 20000
INIT_KGON = 2
NUM_OBSERVATIONS = 50
# new
# Define the importance of each feature
FEATURE_IMPORTANCE = np.array([.3, .41, .007, .44, .7, .89, .99, .23])
STEPS = sorted(list(set(np.logspace(0, np.log10(NUM_EPOCHS),
NUM_OBSERVATIONS).astype(int))))
PLOT_STEPS = [min(STEPS, key=lambda s: abs(s-i)) for i in [0, 200, 2000,
10000, NUM_EPOCHS - 1]]
PLOT_INDICES = [STEPS.index(s) for s in PLOT_STEPS]
logs, weights = create_and_train(NUM_FEATURES,
NUM_HIDDEN_UNITS, num_samples=NUM_SAMPLES,
log_ivl=STEPS, batch_size=100, lr=0.01, num_epochs=NUM_EPOCHS,
init_kgon=INIT_KGON, init_zerobias=False, seed=1)
weights_to_plot = [weights[i]['embedding.weight'] for i in PLOT_INDICES]
losses = [logs.loc[logs['step'] == s, 'loss'].values[0] for s in STEPS]
plot_losses_and_polygons(STEPS, losses, PLOT_STEPS, weights_to_plot)
plt.show()
```

## Summary in Changes

In summary, transitioning from the weight-less, prima example to the Feature Importance Set 1 model reveals that explicit feature importance radically reshapes neural development. The model learns faster and better by immediately prioritizing a representational dimension for important features. This causes the classic developmental phases to collapse into a front-loaded learning process focused on high-weight features, followed by gradual refinement for less important ones.

These findings have profound implications:

1. Data Preparation: Feature weighting (e.g., through domain expertise or statistical analysis) could significantly speed up learning and improve performance.

2. Architecture Design: Models might benefit from architectures that explicitly allocate more capacity to important features, mirroring the dimensional prioritization seen here.

3. Developmental AI: The idea of uniform developmental phases might need revision. With structured data (like weighted features), development might be better characterized by immediate focus on high-value information, then gradual expansion.

4. Interpretability: The clear dimensional prioritization makes the model's behavior more interpretable. We can almost "read off" feature importance from the model's internal geometry.

In essence, this comparison suggests that when we provide models with explicit structure (here, feature importance), their development becomes less about discovering structure and more about efficiently leveraging it. This could point towards a synthesis of traditional "feature engineering" with modern deep learning, where we guide the learning process with domain knowledge without fully constraining it.

# Conclusion

In this paper, we have explored the developmental interpretability of a toy model of superposition (TMS) by examining how its learning dynamics and internal representations change under varying input conditions. It's been a roller coaster of discoveries.

Increasing the hidden dimensionality revealed a qualitative transformation in the model's learning trajectory. With more dimensions, the model exhibited complex, interleaved developmental phases, suggesting a form of representational plasticity. This came at the cost of increased training difficulty, highlighting a tradeoff between expressivity and stability.

Manipulating input density led to a spectrum of behaviors. Sparser inputs induced distinct, sequential developmental stages, while denser inputs encouraged more uniform dimensional usage and faster, but more challenging, learning. This suggests that input sparsity can regulate the structure and pace of neural development.

Finally, introducing non-uniform feature importance radically reshaped the model's learning dynamics. The model immediately prioritized dimensions corresponding to high-importance features, leading to a collapsed developmental trajectory focused on leveraging pre-specified structure. These findings have significant implications for both artificial and biological neural networks. They suggest that developmental phases, critical periods, and representational plasticity can emerge from simple learning algorithms, providing a bridge between AI and neuroscience.

As I wrap up this research, I feel like I've only scratched the surface of what's possible with developmental interpretability. It's a powerful lens that can help us peer inside the black box of neural networks and understand how they learn and grow. And in doing so, it can help us build AI systems that are not just powerful, but also transparent, robust, and trustworthy.

# Final Remarks and Next Steps

I have learned a lot after running this project sprint for BlueDot Impact's AI Alignment course. Interpretability seems to be at the heart of many AI safety problems. It's definitely not easy to get to and understand the answer, but it is not impossible. I hope that by using this toy model of superposition, it allows for the reader to have a better grasp on how models develop themselves after initializing them (which is really all I changed). How neurons take on mutliple meanings and increase in semantic dimensionality during the training process in order to fit data as best it can within its networks.

**Next Steps (Maybe *you!* could do)**

**Additional Variations:**

**Input and Output Dimension (c):** The paper uses c=6 (six-dimensional input and output). Explore different values of c. How does changing the input and output space impact the complexity of the TMS and the formation of superpositions?

**Non-linear activation functions:** Explore using different activation functions in the model beyond the linear ones used in the original paper.

The paper likely uses a standard non-linear activation function. Try using different activation functions and see how they affect the TMS behavior. Do some functions promote superpositions more than others?

**Batch size:** Experiment with different batch sizes during training and see how it affects the convergence and phase transitions.

**Training Dynamics:** Analyze the training dynamics of the TMS in more detail. How do the hidden dimensions evolve during training? Are there specific loss function behaviors associated with phase transitions? Are there critical points during training where complexity jumps occur?

By exploring these variations, you can gain a deeper understanding of how different factors influence the development of TMS and its ability to represent complex features. It might also provide insights into the connection between TMS behavior and real-world neural network training dynamics.

# References

**Paper Sources**

Chen, Zhongtian, et al. "Dynamical versus Bayesian Phase Transitions in a Toy Model of Superposition." *arXiv*, 10 Oct. 2023, arxiv.org/abs/2310.06301.

Lau, Edmund, et al. "Quantifying Degeneracy in Singular Models via the Learning Coefficient." *arXiv*, 23 Aug. 2023, arxiv.org/abs/2308.12108.

**Other Interpretability Sources**

Elhage, Hume, et al. "Toy Models of Superposition." *Transformer Circuits*, 14 Sept. 2022, transformer-circuits.pub/2022/toy_model/index.html.

Nanda, Neel. "Concrete Steps to Get Started in Transformer Mechanistic Interpretability." *Neel Nanda*, 25 Dec 2023, www.neelnanda.io/mechanistic-interpretability/getting-started.

Olah, Cammarata, et al. "Zoom in: An Introduction to Circuits" *Distill.Pub*, 10 Mar. 2020, distill.pub/2020/circuits/zoom-in/.

Olah, Chris. "Looking Inside Neural Networks with Mechanistic Interpretability." *YouTube*, 1 Sept. 2023, www.youtube.com/watch?v=2Rdp9GvcYOE.

Templeton, Conerly, et al. "Mapping the Mind of a Large Language Model." *Anthropic*, 21 May 2024, www.anthropic.com/research/mapping-mind-language-model.

Note that I used Anthropic's Claude 3 Sonnet at times to help me, too!