Machine Learning Methods for Neural Data Analysis

LFADS

Scott Linderman

STATS 220/320 (NBIO220, CS339N).

Agenda

- LFADS: A stochastic RNN model for neural spike trains
- Recurrent Neural Networks (RNNs)
- Learning and Inference in LFADS
- Amortized Inference with “Recognition Networks”

Nonlinear models for time series data

- In neuroscience, we're often interested in **sequential data** $y_{1:T} = (y_1, \ldots, y_T)$. E.g., neural spike trains or behavioral time series.

- We could model each time point an an independent observation,

$$x_t \sim \mathcal{N}(0, I) \quad y_t \sim \mathcal{N}(f(x_t; \theta), \sigma^2 I)$$

  where x_t is a *latent state*, and $f(x; \theta)$ is a neural network with weights θ that maps latent states to observations.

- This captures nonlinear relationships between x_t and y_t, but how do we model dynamics over time?

Nonlinear state space models

- We could incorporate **temporal dependencies into the prior.** E.g., via an linear dynamical system prior,

$$p(x_{1:T}) = \mathcal{N}(x_1 \mid 0, Q_1) \prod_{t=2}^{T} \mathcal{N}(x_t \mid Ax_{t-1} + b, Q).$$

- More generally, we could have a **nonlinear dynamical system**,

$$p(x_{1:T}) = \mathcal{N}(x_1 \mid 0, Q_1) \prod_{t=2}^{T} \mathcal{N}(x_t \mid h(x_{t-1}; \theta), Q).$$

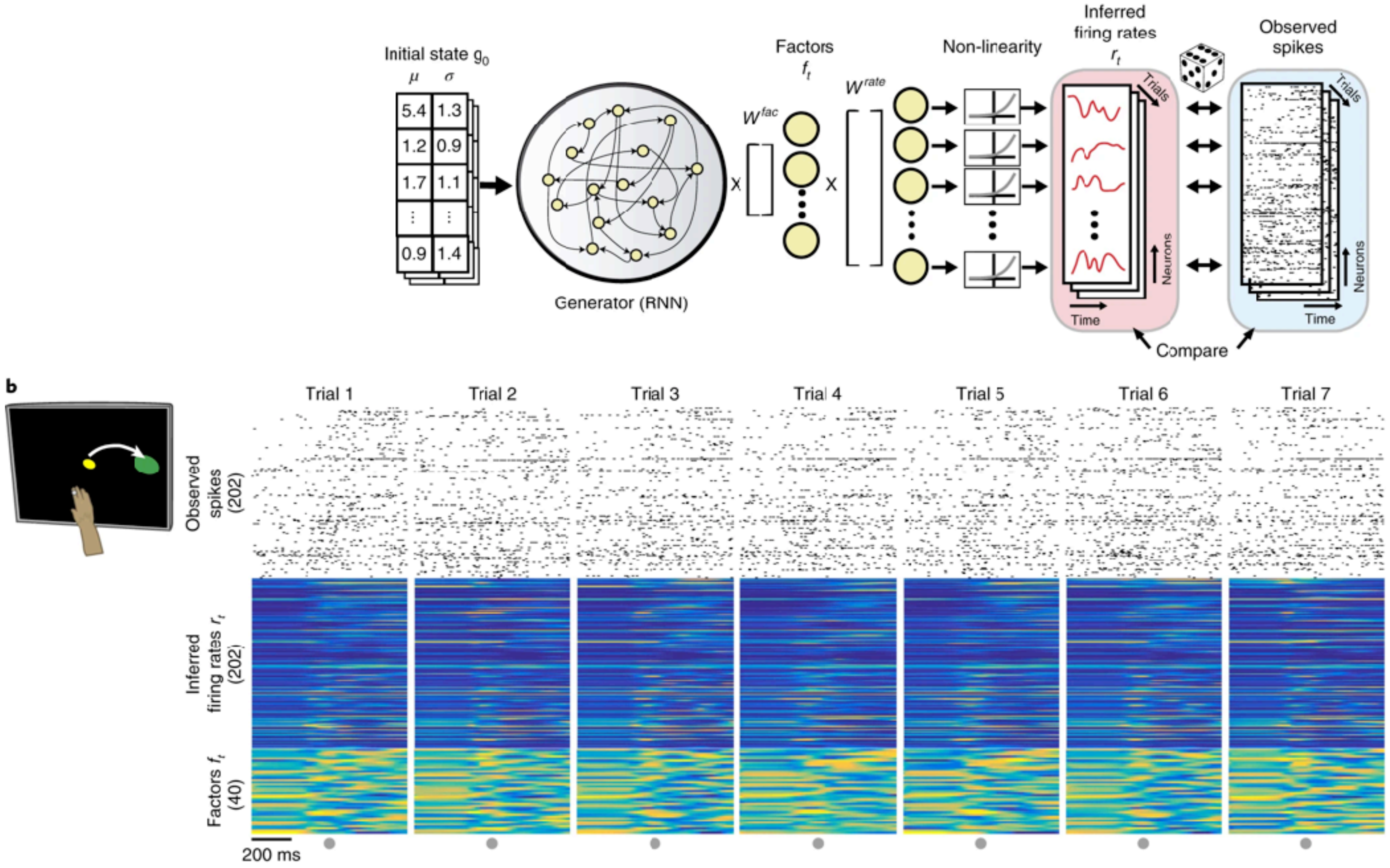
  where θ are the parameters of a neural network.

- For example, $h(x; \theta)$ could be a **recurrent neural network.**

LFADS: Latent Factor Analysis for Dynamical Systems

A Stochastic RNN model

- LFADS uses a recurrent neural network (the **generator**) to model nonlinear dynamics of neural activity.
- In the basic model, the RNN has **deterministic dynamics** with a **random initial condition.**
- The RNN state is mapped through a **GLM** to obtain firing rates for a **Poisson model.**

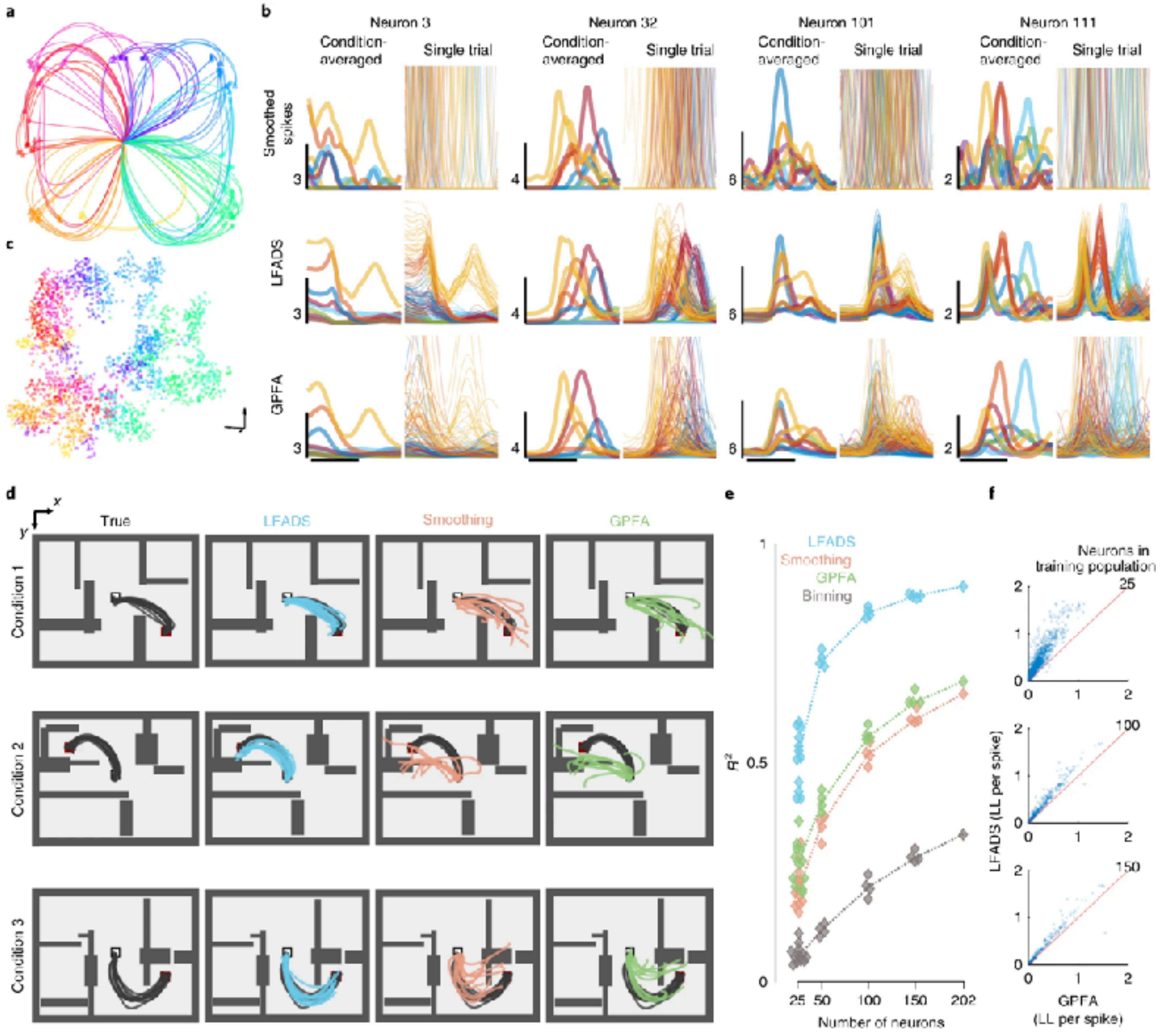


Pandarinath et al (2018)

LFADS: Latent Factor Analysis for Dynamical Systems

A Stochastic RNN model

- LFADS learns accurate **single-trial firing rates** and achieves excellent **decoding performance** on monkey reaching tasks (Recall Lab 5).



Pandarinath et al (2018)

The LFADS Generator

Stochastic dynamics vs stochastic inputs

- LFADS uses a slightly different formulation of the prior.
- Instead of having **stochastic dynamics**,

$$p(x_{1:T}) = \mathcal{N}(x_1 \mid 0, Q_1) \prod_{t=2}^{T} \mathcal{N}(x_t \mid h(x_{t-1}; \theta), Q).$$

  It uses **stochastic inputs** with **deterministic dynamics.**

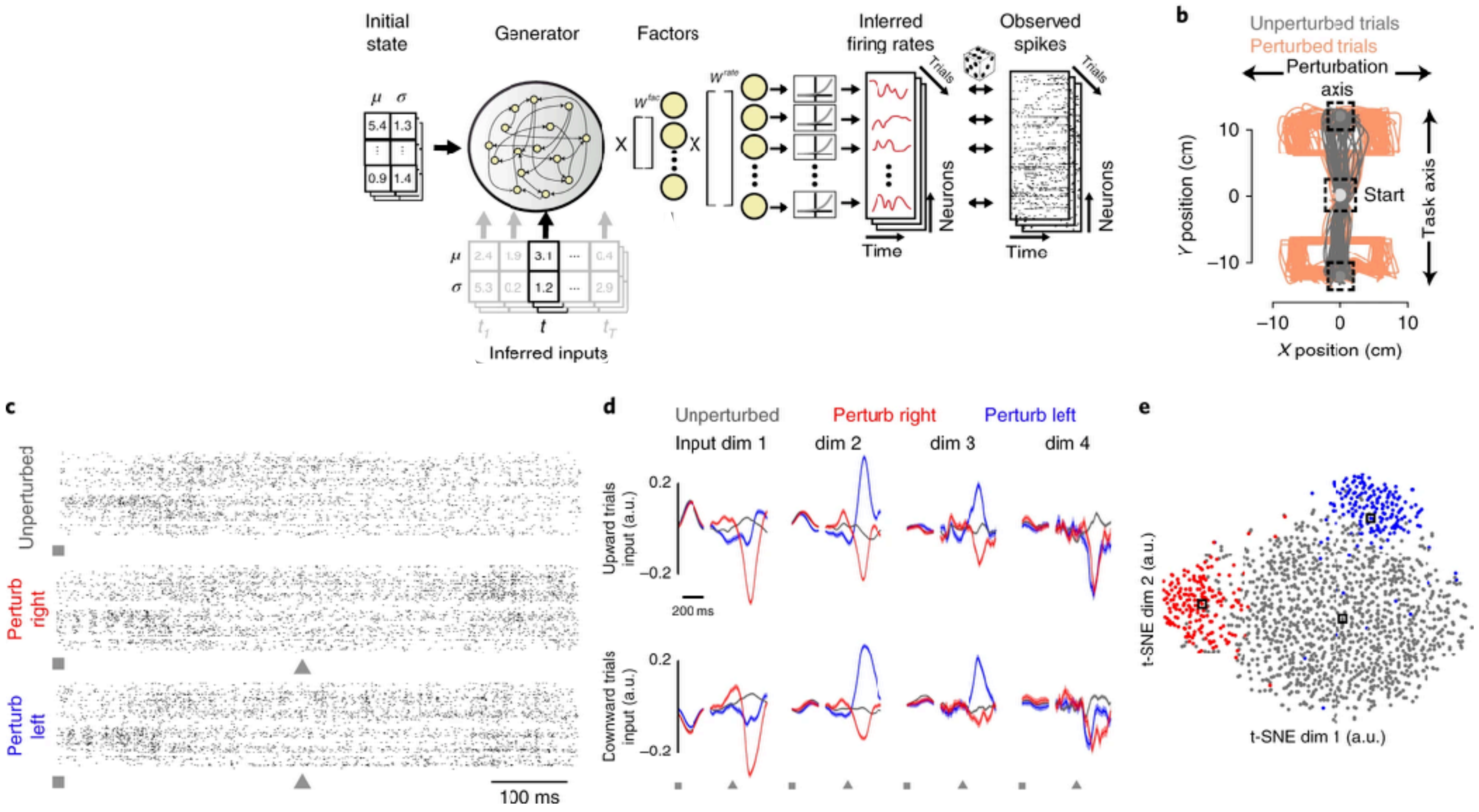
$$x_0 \sim \mathcal{N}(\ \mid 0, Q_1) \qquad u_t \overset{\text{iid}}{\sim} \mathcal{N}(0, I) \qquad x_t = h(x_{t-1}, u_t; \theta).$$

- This is just a **reparameterization**. It implies a distribution on $x_{0:T}$, but that distribution could be quite complex since h is nonlinear.

The LFADS Generator

Inferred Inputs

- The **inferred inputs** can suggest the presence, identity, and timing of **unexpected changes** in the dynamics.
- For example, in trials where the **cursor was randomly perturbed** to the right or left, inputs capture corresponding changes in neural activity.

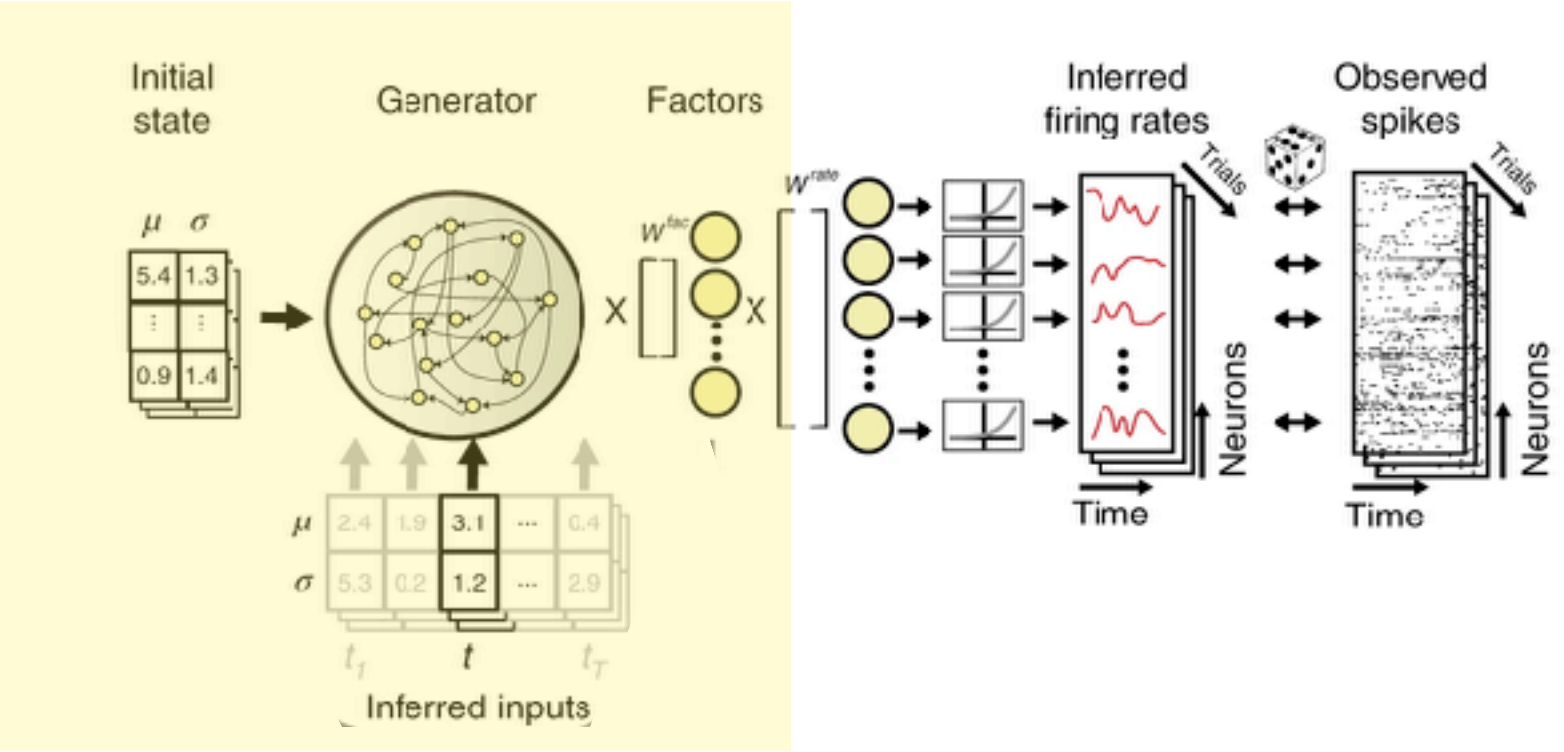


Pandarinath et al (2018)

The LFADS Generator

Reparameterizing the latent state

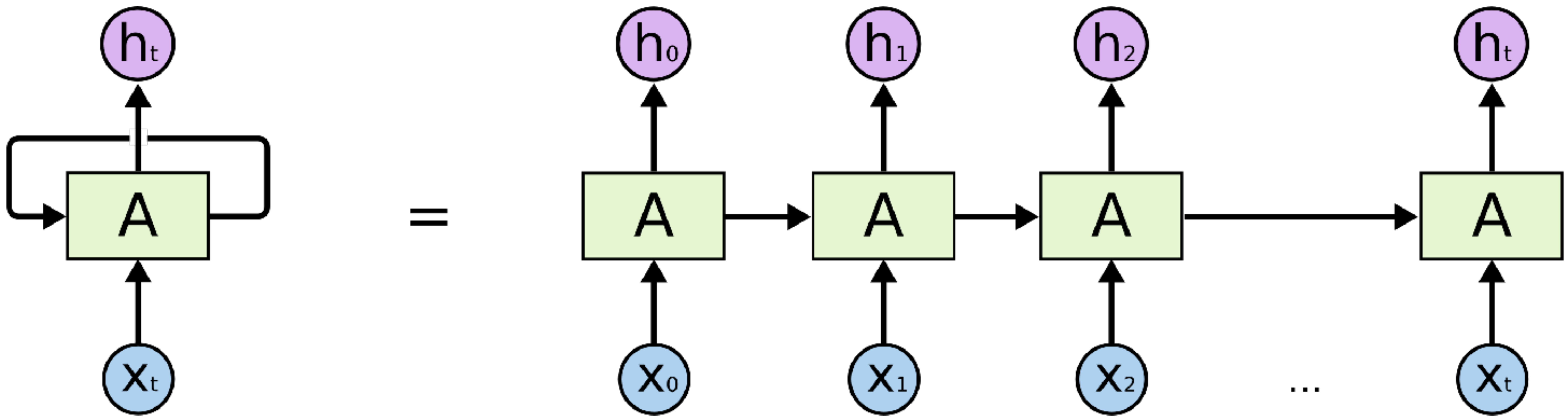
We can unwind the recursion to write the state at time t as a deterministic function of the initial condition and the inputs up to time t,



$$\begin{aligned} x_t &= h(x_{t-1}, u_t, \theta) \\ &= h(h(x_{t-2}, u_{t-1}, \theta), u_t, \theta) \\ &= h(\cdots h(h(x_0, u_1, \theta), u_2, \theta)\cdots) \\ &\triangleq h_t(x_0, u_{1:t}, \theta) \end{aligned}$$

Recurrent Neural Networks

“Vanilla” RNNs



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Networks

Vanishing and Exploding Gradients

- To optimize the ELBO, we'll need derivatives of the state with respect to the initial state,

$$\frac{\partial x_t}{\partial x_0} = \frac{\partial}{\partial x_{t-1}} h(x_{t-1}, u_t, \theta) \cdot \frac{\partial x_{t-1}}{\partial x_0}$$
$$= \prod_{s=1}^{t} \frac{\partial}{\partial x_{s-1}} h(s_{t-1}, u_s, \theta)$$

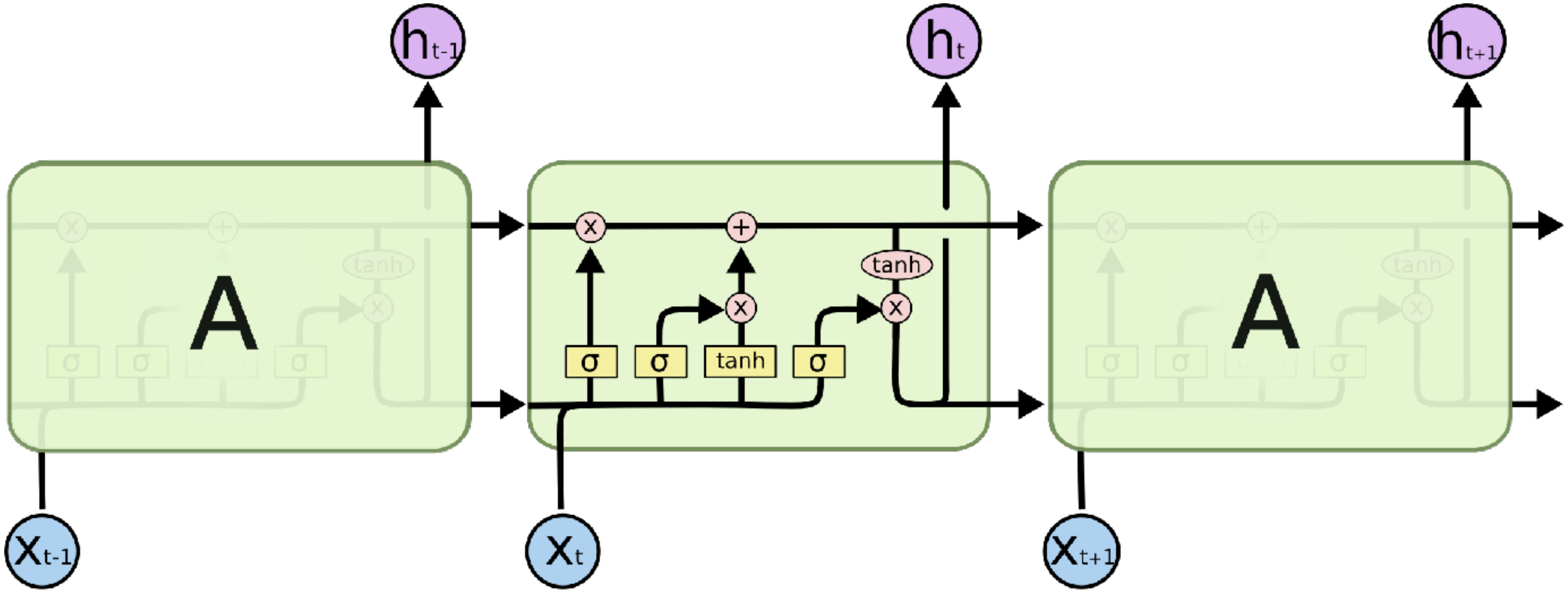
- In a vanilla RNN, $h(x, u) = \tanh(Wx + Bu)$, then,

$$\frac{\partial}{\partial x} h(x, u_t, \theta) = \mathrm{diag}(\mathrm{sech}^2(Wx + Bu_t))\, W$$

- Multiplying a bunch of these matrices together leads to **vanishing or exploding gradients**, depending on the eigenvalues of W.

Recurrent Neural Networks

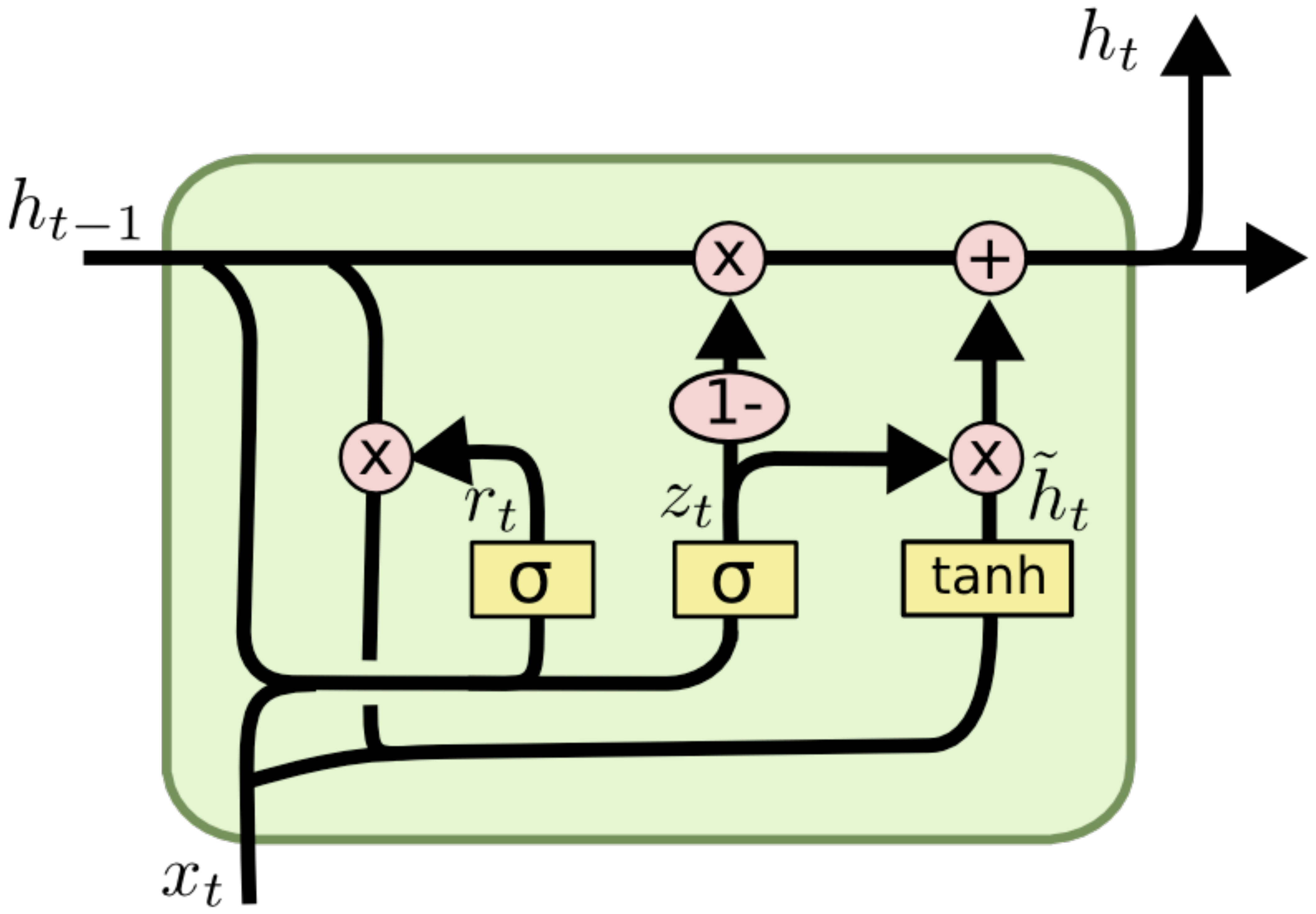
Long Short-Term Memory (LSTM) networks



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Networks

Gated Recurrent Units (GRUs)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

The LFADS Generator

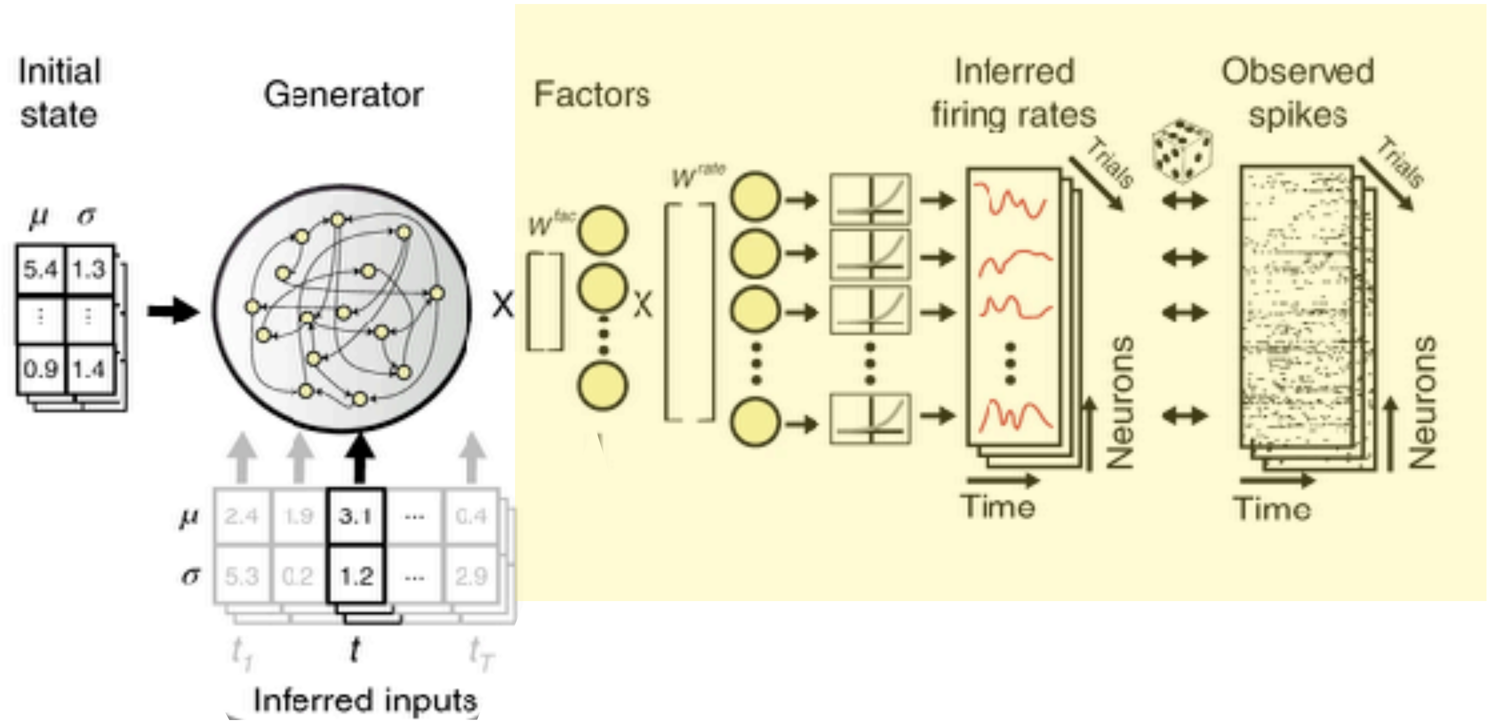
The emission model

- The output is modeled as a (typically simple) function of the latent state,

$$y_t \sim \mathrm{Po}\big(f(x_t)\big)$$

where, e.g.,

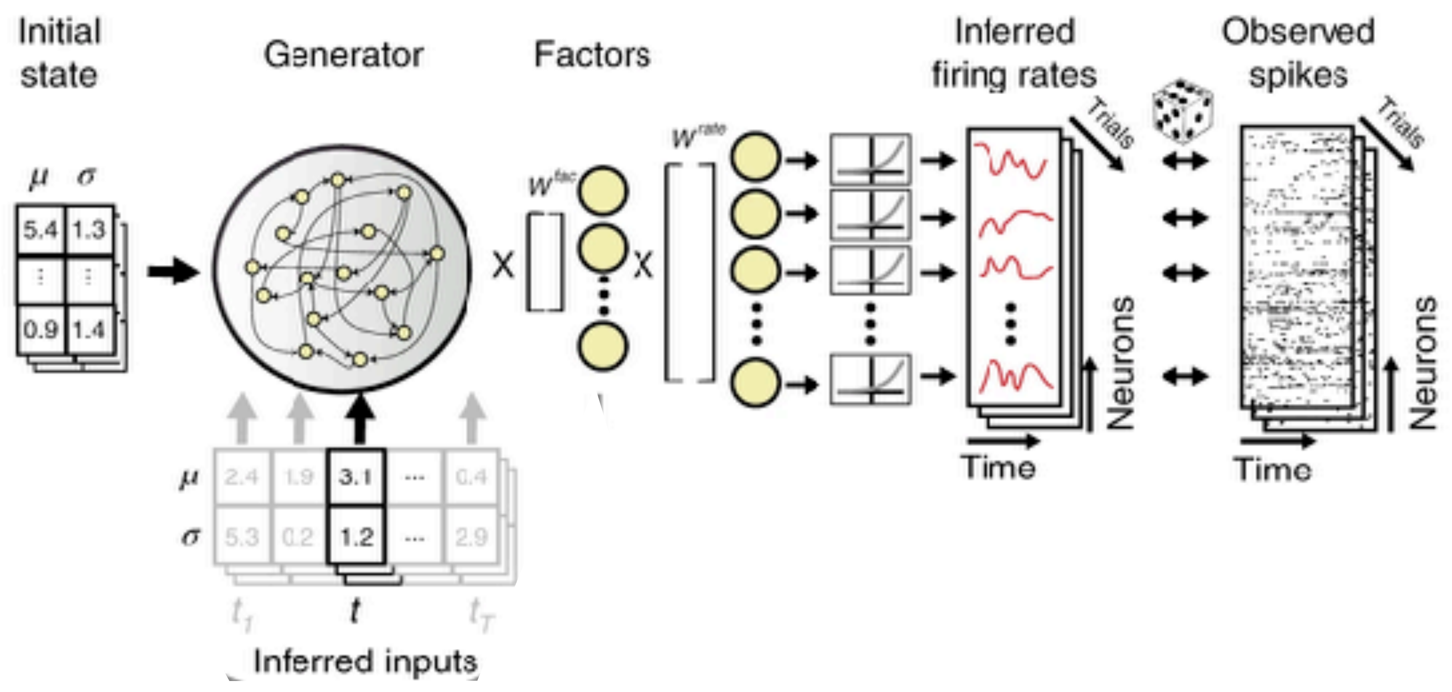
$$f(x_t) = \exp\left\{Cx_t + d\right\}.$$



The LFADS Generator

Joint distribution

- Assume the initial condition and inputs have standard normal priors.



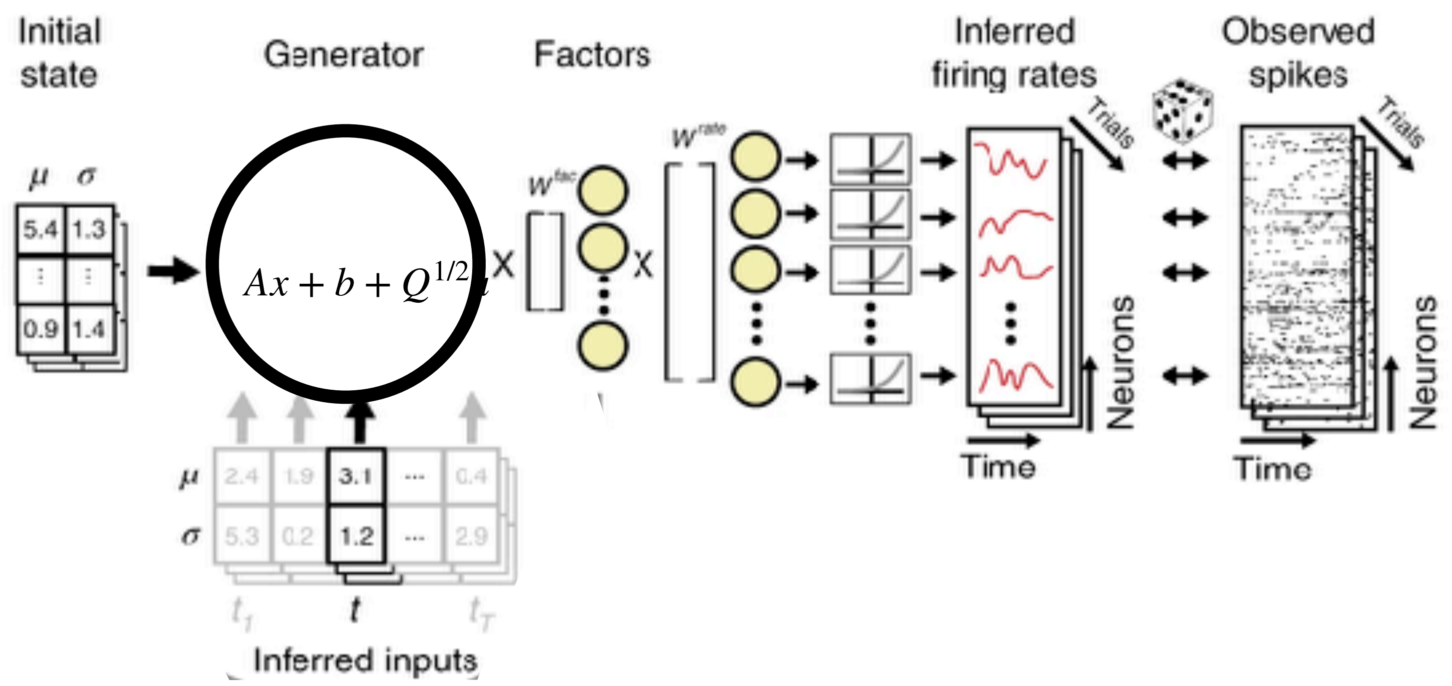
- The joint distribution is,

$$
\begin{aligned}
p(x_0, u_{1:T}, y_{1:T} \mid \theta) &= \mathcal{N}(x_0 \mid 0, I) \prod_{t=1}^{T} \mathcal{N}(u_t \mid 0, I)\, \mathrm{Po}(y_t \mid f(x_t)) \\
&= \mathcal{N}(x_0 \mid 0, I) \prod_{t=1}^{T} \mathcal{N}(u_t \mid 0, I)\, \mathrm{Po}\big(y_t \mid f(h_t(x_0, u_{1:t}, \theta))\big)
\end{aligned}
$$

The LFADS Generator

Poisson LDS as a special case of LFADS

- We can view the **Poisson LDS** (c.f. Macke et al, 2011) as a special case of LFADS with a **linear generator**.

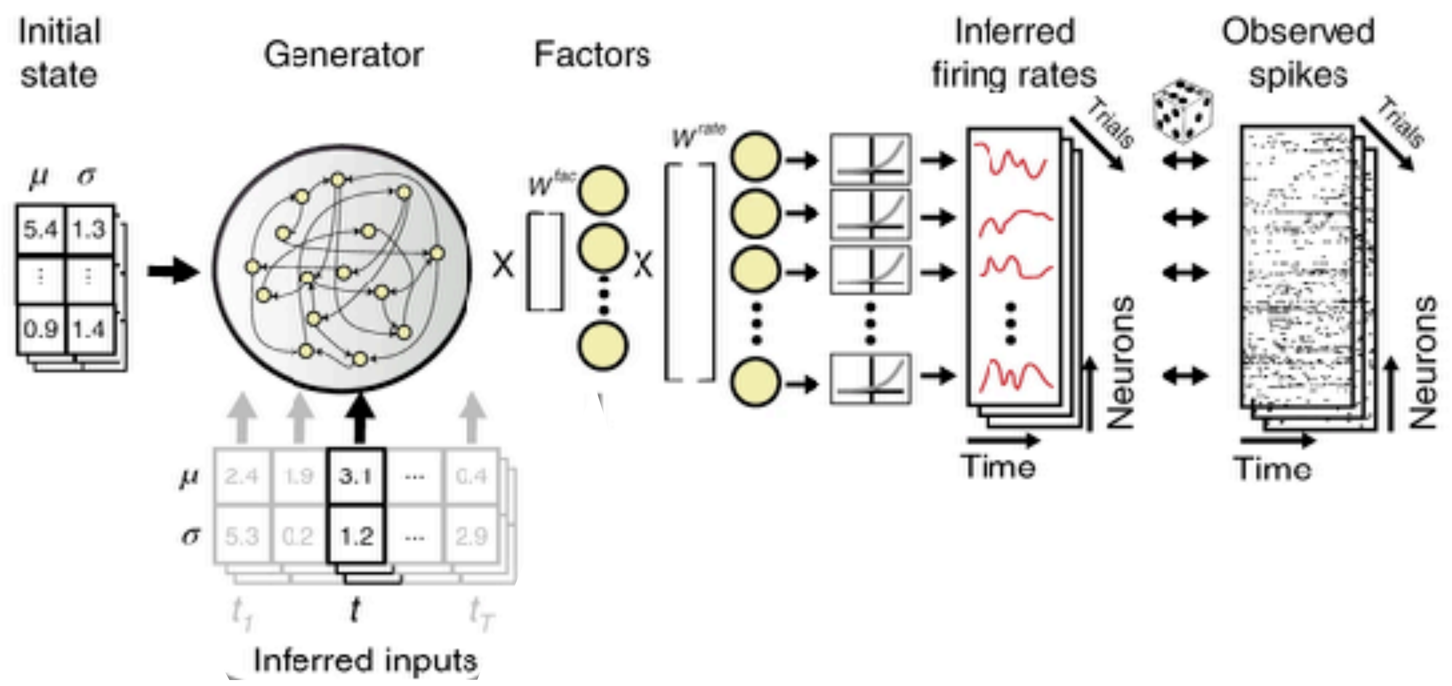


$$x_t \sim \mathcal{N}(Ax_{t-1} + b, Q) \qquad \Longleftrightarrow \qquad \begin{aligned} x_t &= h(x_{t-1}, u_t, \theta) \\ h(x_{t-1}, u_t, \theta) &= Ax_{t-1} + b + Q^{1/2}u_t \\ u_t &\sim \mathcal{N}(0, I) \end{aligned}$$

$$y_t \sim \mathrm{Po}\big(f(x_t)\big) \qquad\qquad y_t \sim \mathrm{Po}\big(f(x_t)\big)$$

Learning and Inference in LFADS

Variational EM



- How to learn the parameters θ and infer the latent variables $x_0, u_{1:T}$?
- **Variational EM**:
    - **E step:** Approximate the posterior with,

$$q(x_0, u_{1:T}) \approx p(x_0, u_{1:T} \mid y_{1:T}, \theta)$$

    - **M step:** Find parameters that maximize the ELBO

$$\mathscr{L}[q, \theta] = \mathbb{E}_{q(x_0, u_{1:T})} \left[\log p(x_0, u_{1:T}, y_{1:T}) - \log q(x_0, u_{1:T}) \right]$$

Learning and Inference in LFADS

Variational Approximation

- Let's assume a Gaussian form for each factor,

$$q(x_0, u_{1:T}; \lambda) = \mathcal{N}(x_0 \mid \tilde{\mu}_0, \tilde{\Sigma}_0) \prod_{t=1}^{T} \mathcal{N}(u_t \mid \tilde{\mu}_t, \tilde{\Sigma}_t)$$

- This approximation is parameterized by **variational parameters** $\lambda \triangleq \{\tilde{\mu}_t, \tilde{\Sigma}_t\}_{t=0}^{T}$.

- Let $\mathscr{L}(\lambda, \theta) = \mathscr{L}[q(x_0, u_{1:T}; \lambda), \theta]$ denote the ELBO as a function of the variational and generative model parameters.

Learning and Inference in LFADS

ELBO Surgery

ELBO Surgery*: we can rewrite the ELBO as,

$$\begin{aligned}
\mathscr{L}(\lambda, \Theta) &= \mathbb{E}_{q(x_0, u_{1:T}, \lambda)}\left[\log p(x_0, u_{1:T}) + \log p(y_{1:T} \mid x_0, u_{1:T}, \Theta) - \log q(x_0, u_{1:T}; \lambda)\right] \\
&= \mathbb{E}_{q(x_0, u_{1:T}, \lambda)}\left[\log p(y_{1:T} \mid x_0, u_{1:T}, \Theta) - \log \frac{q(x_0; \lambda)}{p(x_0)} - \sum_{t=1}^{T} \log \frac{q(u_t; \lambda)}{p(u_t)}\right] \\
&= \underbrace{\mathbb{E}_{q(x_0, u_{1:T}, \lambda)}\left[\sum_{t=1}^{T} \log p(y_t \mid x_0, u_{1:t}, \Theta)\right]}_{\text{expected log likelihood}} \underbrace{- \mathrm{KL}(q(x_0; \lambda) \,\|\, p(x_0)) - \sum_{t=1}^{T} \mathrm{KL}(q(u_t; \lambda) \,\|\, p(u_t))}_{\text{KL to the prior}}
\end{aligned}$$

*For more ways of rewriting the ELBO, see Johnson and Hoffman (2017)

Learning and Inference in LFADS

Gradients wrt θ

Gradient ascent on the ELBO:

$$\nabla_\theta \mathscr{L}(\lambda, \theta) = \mathbb{E}_{q(x_0, u_{1:T}, \lambda)} \left[\sum_{t=1}^{T} \nabla_\theta \log p(y_t \mid x_0, u_{1:t}, \theta) \right]$$

Since the generative parameters don't appear in q, we can **pull the gradient inside the expectation** and compute it with **automatic differentiation** for any $x_0, u_{1:t}, \theta$.

Then approximate the expectation with **Monte Carlo:**

$$\nabla_\Theta \mathscr{L}(\lambda, \theta) \approx \frac{1}{M} \sum_{m=1}^{M} \left[\sum_{t=1}^{T} \nabla_\Theta \log p(y_t \mid x_0^{(m)}, u_{1:t}^{(m)}, \theta) \right] \qquad x_0^{(m)} \sim q(x_0; \lambda),\ u_t^{(m)} \sim q(u_t; \lambda).$$

Learning and Inference in LFADS

The "reparameterization trick"

The gradients with respect to the variational parameters are a bit trickier:

$$\nabla_\lambda \mathscr{L}(\lambda, \theta) = \nabla_\lambda \mathbb{E}_{q(x_0, u_{1:T}, \lambda)}\left[\sum_{t=1}^{T} \log p(y_t \mid x_0, u_{1:t}, \theta)\right] - \nabla_\lambda \mathrm{KL}\big(q(x_0, u_{1:T}, \lambda) \,\|\, p(x_0, u_{1:T})\big)$$

Note that $x_0 \sim \mathcal{N}(\tilde{\mu}_0, \tilde{\Sigma}_0) \iff x_0 = \tilde{\mu}_0 + \tilde{\Sigma}_0^{1/2}\epsilon_0$ where $\epsilon_0 \sim \mathcal{N}(0, I)$.

Learning and Inference in LFADS

The "reparameterization trick"

The gradients with respect to the variational parameters are a bit trickier:

$$\nabla_\lambda \mathscr{L}(\lambda, \theta) = \nabla_\lambda \mathbb{E}_{q(x_0, u_{1:T}, \lambda)} \left[\sum_{t=1}^{T} \log p(y_t \mid x_0, u_{1:t}, \theta) \right] - \nabla_\lambda \mathrm{KL}\big(q(x_0, u_{1:T}, \lambda) \,\|\, p(x_0, u_{1:T})\big)$$

Note that $x_0 \sim \mathcal{N}(\tilde{\mu}_0, \tilde{\Sigma}_0) \iff x_0 = \tilde{\mu}_0 + \tilde{\Sigma}_0^{1/2} \epsilon_0$ where $\epsilon_0 \sim \mathcal{N}(0, I)$.

We can **reparameterize the model** in terms of an expectation wrt $\epsilon_{0:T}$ and then take the gradient inside the expectation, as before

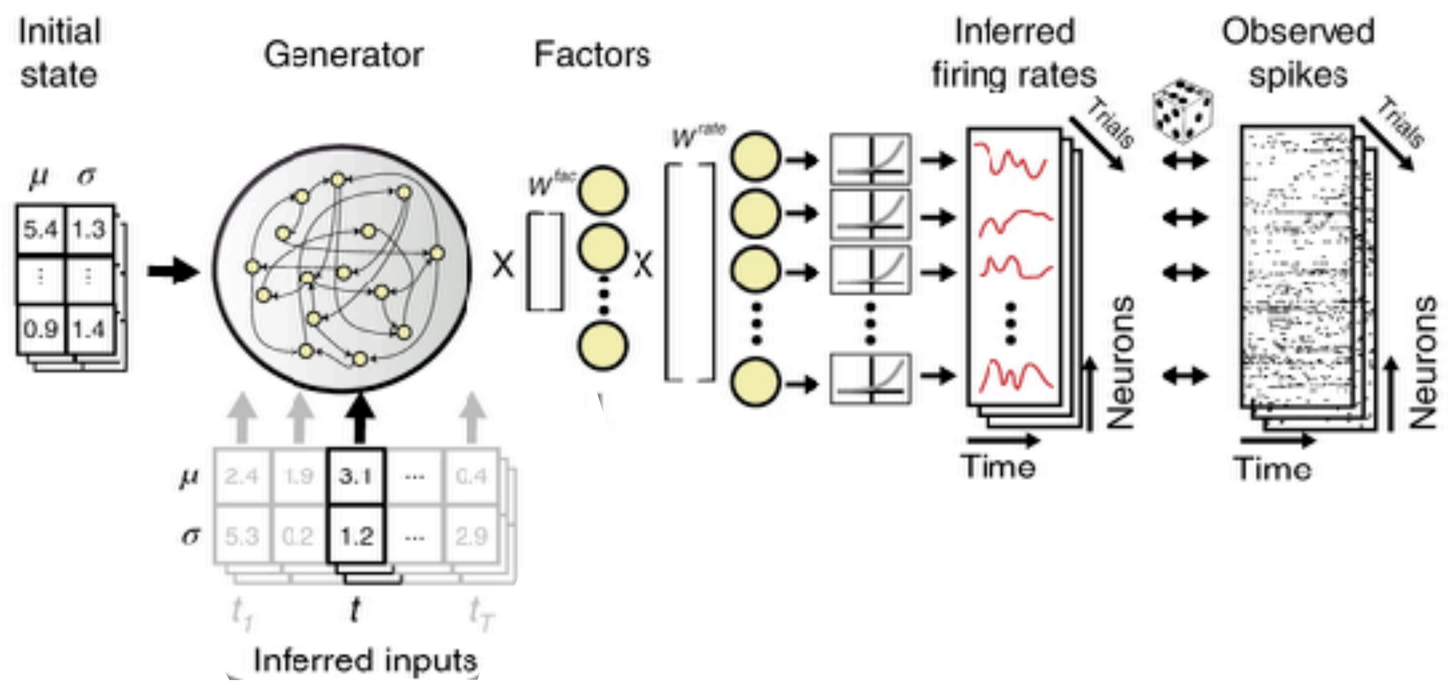
$$\nabla_\lambda \mathscr{L}(\lambda, \theta) = \mathbb{E}_{\epsilon_{0:T}} \left[\sum_{t=1}^{T} \nabla_\lambda \log p(y_t \mid x_0(\epsilon_0, \lambda), u_1(\epsilon_1, \lambda), \ldots, u_t(\epsilon_t, \lambda), \theta) \right] - \nabla_\lambda \mathrm{KL}\big(q(x_0, u_{1:T}, \lambda) \,\|\, p(x_0, u_{1:T})\big)$$

As before, we can approximate this with ordinary Monte Carlo.

Learning and Inference in LFADS

Stochastic Gradient Ascent on the ELBO

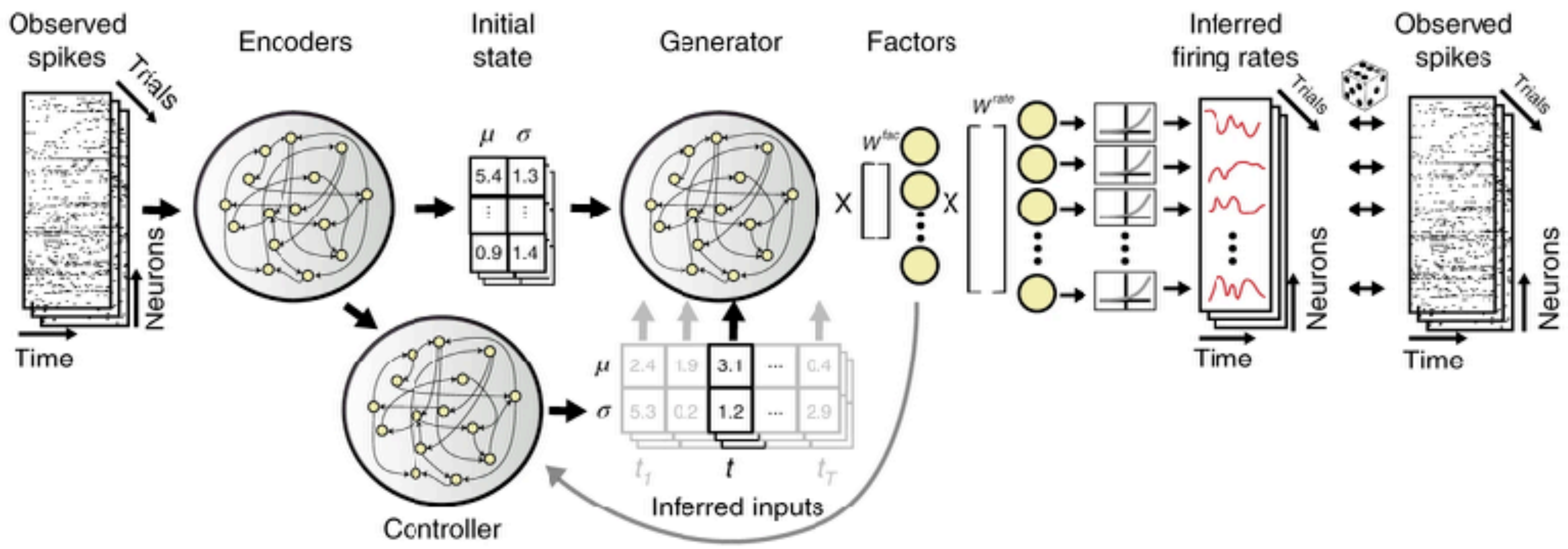
- **Variational EM** via gradient descent and the reparameterization trick,
  - **E step:**
    - Draw $\epsilon_t^{(m)} \sim \mathcal{N}(0, I)$ for $t = 0, \ldots, T$, $s = 1, \ldots, S$.
    - Use ϵ to approximate $\nabla_\lambda \mathscr{L}(\lambda, \theta)$ via Monte Carlo and the reparameterization trick.
    - Update $\lambda \leftarrow \lambda + \alpha \nabla_\lambda \mathscr{L}(\lambda, \theta)$
  - **M step:**
    - Use ϵ to approximate $\nabla_\theta \mathscr{L}(\lambda, \theta)$ via Monte Carlo.
    - Update $\theta \leftarrow \theta + \alpha \nabla_\theta \mathscr{L}(\lambda, \theta)$.



Learning and Inference in LFADS

Amortized inference with encoders / recognition networks

- With large datasets, we often work on one mini-batch at a time.
- In that setting, we need a way to quickly obtain a decent posterior approximation for that mini-batch.
- **Key idea**: the optimal $\lambda^\star$ is a function of the data $y_{1:T}$, so let's **use a neural network** to approximate the mapping from data to variational parameters.
- This is called **amortized inference.**
- The learned network is called an **encoder** or a **recognition network**.



Learning and Inference in LFADS

Amortization and Approximation gaps

- When we switch to nonlinear models, the posterior is no longer Gaussian ⇒ **approximation gap**

- Moreover, neural network encoder may not produce the best Gaussian approximation ⇒ **amortization gap**.

- Both lead to suboptimal inference and learning.

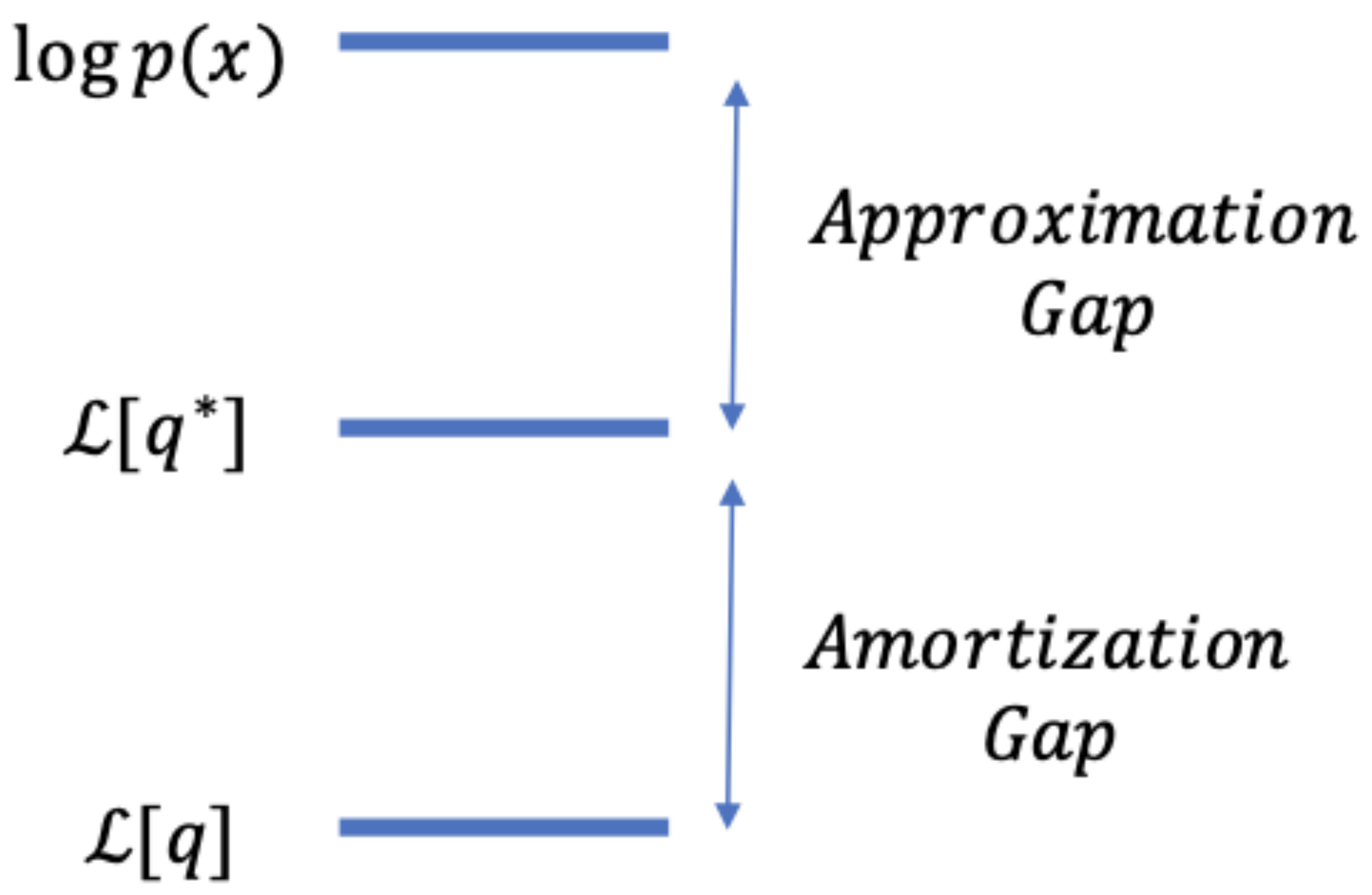


Figure 1. Gaps in Inference

Conclusion

- **Sequential VAEs** are latent variable models for time series data like neural spike trains and behavioral pose trajectories.
- **LFADS** is one such example that is popular in neuroscience. It uses recurrent neural networks to parameterize the nonlinear dynamics, and Poisson GLMs to model the spike count observations.
- **Learning and inference** are much the same as in standard VAEs —we just maximize the ELBO.
- It also uses an RNN for the **recognition network / encoder**, to estimate latent variables given observations.

Further Reading

- Pandarinath, Chethan, Daniel J. O'Shea, Jasmine Collins, Rafal Jozefowicz, Sergey D. Stavisky, Jonathan C. Kao, Eric M. Trautmann, et al. 2018. "Inferring Single-Trial Neural Population Dynamics Using Sequential Auto-Encoders." Nature Methods 15 (10): 805–15.