MAIN Tutorials

Variational autoencoders for neural data analysis

Blake Richards

Mila/McGill/CIFAR

Outline

1. Introduction to variational Bayesian methods
2. Variational autoencoders
3. Example use of VAEs for neuroscience: LFADS
4. Example code

Introduction to variational Bayesian methods

Latent variables

Variational autoencoders (VAEs) are a deep learning technique for inferring latent variables from data using a variational Bayesian approach.

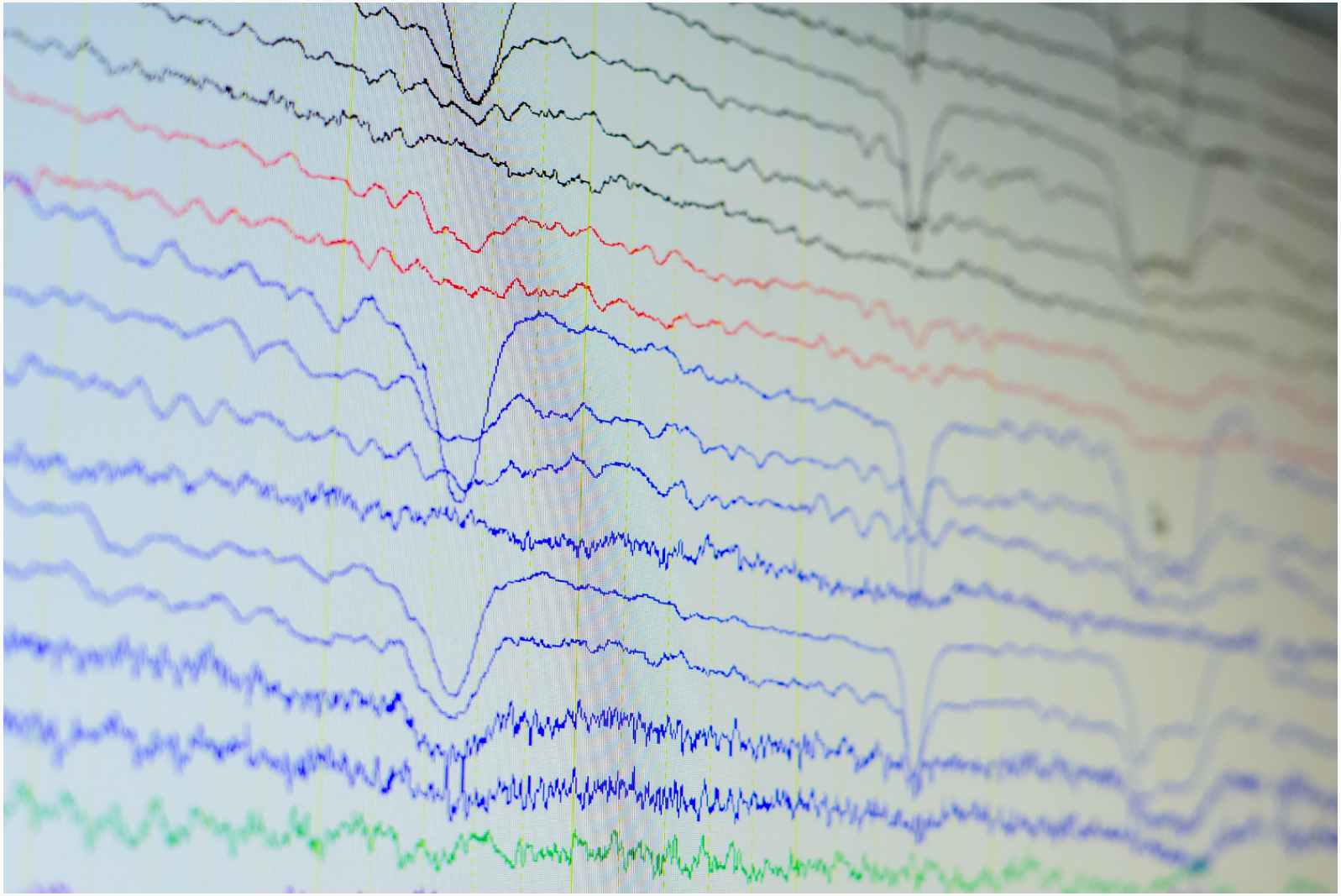
There's a lot to unpack there, let's start with the most important one:

What are latent variables?

Latent variables

Latent variables are variables that are not directly observable, i.e. most of everything we actually care about.

Latent variables



Latent variables

Since we can't observe latent variables, we must infer their values from the data we can observe.

Inferring latent variables

How should we infer latent variables?

Ideally, we would be good Bayesians about it.

Say we have data $\mathbf{x}$, that was generated from some latent variables $\mathbf{z}$ via a parameterized probability distribution $p_\theta(\mathbf{x}|\mathbf{z})$. We want to infer $\mathbf{z}$ using $\mathbf{x}$. So, we should use the posterior:

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})}$$

Inferring latent variables

How should we infer latent variables?

Ideally, we would be good Bayesians about it.

Say we have data $\mathbf{x}$, that was generated from some latent variables $\mathbf{z}$ via a parameterized probability distribution $p_\theta(\mathbf{x}|\mathbf{z})$. We want to infer $\mathbf{z}$ using $\mathbf{x}$. So, we should use the posterior:

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{\int p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})d\mathbf{z}}$$

Inferring latent variables

Problem

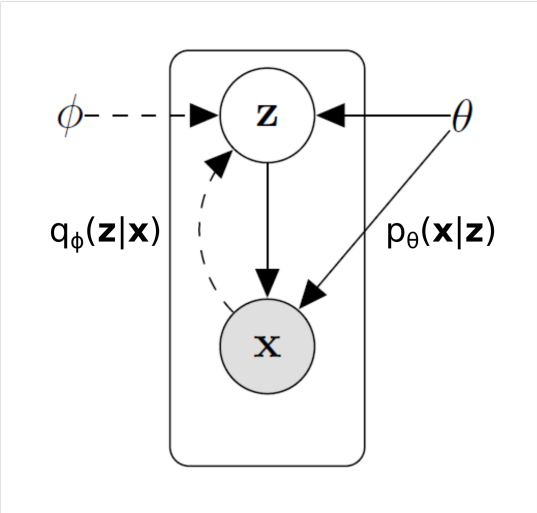
If $\mathbf{z}$ is a continuous variable, and $p_\theta(\mathbf{x}|\mathbf{z})$ is a complicated function, then $\int p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})d\mathbf{z}$ is intractable. So, we have no way of estimating our posterior directly:

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{\int p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})d\mathbf{z}}$$

Solution

We can get around our intractable posterior by using a tractable, variational approximation, $q_\phi(\mathbf{z}|\mathbf{x})$ instead (ϕ refers to the parameters for this approximation). Our goal then becomes:

$$q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{\int p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})d\mathbf{z}}$$



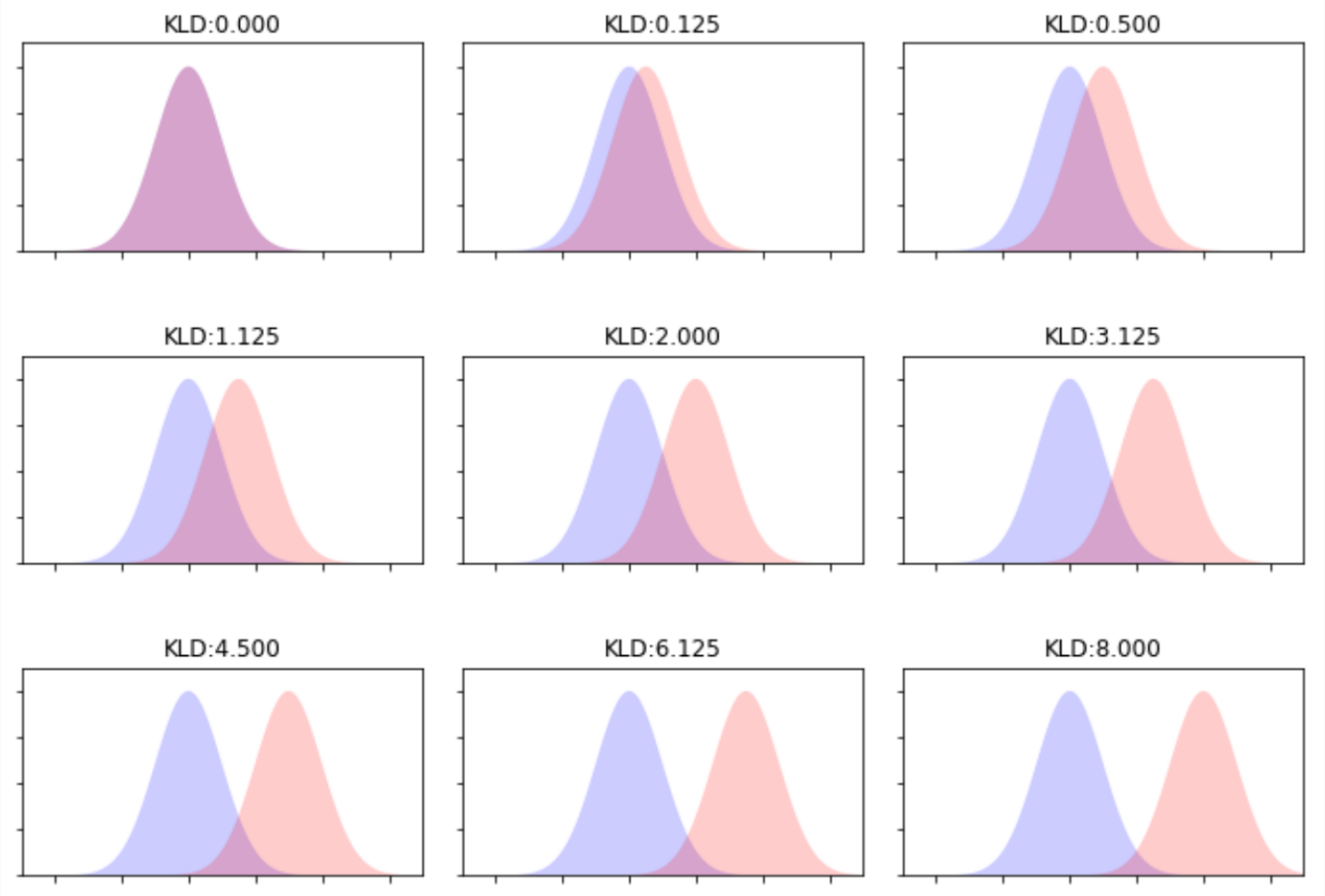
Variational Bayesian methods

We measure the dissimilarity between $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{z}|\mathbf{x})$ using the Kullback-Leibler divergence (D_{KL}):

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) = \int q_\phi(\mathbf{z}|\mathbf{x}) \log(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}) d\mathbf{z}$$

Kullback-Leibler divergence

$D_{KL}(q||p)$ measures the number of bits that would be required to encode data drawn from q using p. It is zero if and only if $q = p$, and rises as they become less similar:



Variational Bayesian methods

So, to get $q \approx p$ we want to reduce $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))$:

$$\begin{aligned} D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}) d\mathbf{z} \\ &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})}) d\mathbf{z} + log(p_\theta(\mathbf{x})) \end{aligned}$$

Giving us:

$$\begin{aligned} log(p_\theta(\mathbf{x})) &= D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) - \int q_\phi(\mathbf{z}|\mathbf{x}) \log(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})}) d\mathbf{z} \\ &= D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) - \mathcal{L}(\phi, \theta; \mathbf{x}) \end{aligned}$$

Variational Bayesian methods

$\mathcal{L}(\phi, \theta; \mathbf{x})$ is our variational lower bound on the *evidence* ($\log(p_\theta(\mathbf{x}))$). Because $\log(p_\theta(\mathbf{x}))$ is fixed w.r.t. q, we can reduce $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))$ by maximizing $\mathcal{L}(\phi, \theta; \mathbf{x})$. Thus, we want to maximize:

$$\begin{aligned}\mathcal{L}(\phi, \theta; x) &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})}) d\mathbf{z} \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log(p_\theta(\mathbf{x}|\mathbf{z}))] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \\ &= \text{Expected log-likelihood} - \text{Deviation from prior}\end{aligned}$$

Interim summary

Here's what we've covered so far:

- Latent variables, $\mathbf{z}$, are unobservable variables that cause the data we can observe, $\mathbf{x}$
- Inferring latent variables using standard Bayesian techniques can be intractable
- Variational Bayesian methods make it tractable by approximating the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$, with a variational approximation $q_\phi(\mathbf{z}|\mathbf{x})$
- To make our approximation better, we optimize our variational lower bound: $\mathcal{L}(\phi, \theta; \mathbf{x})$

Variational autoencoders

Variational autoencoders: the basic idea

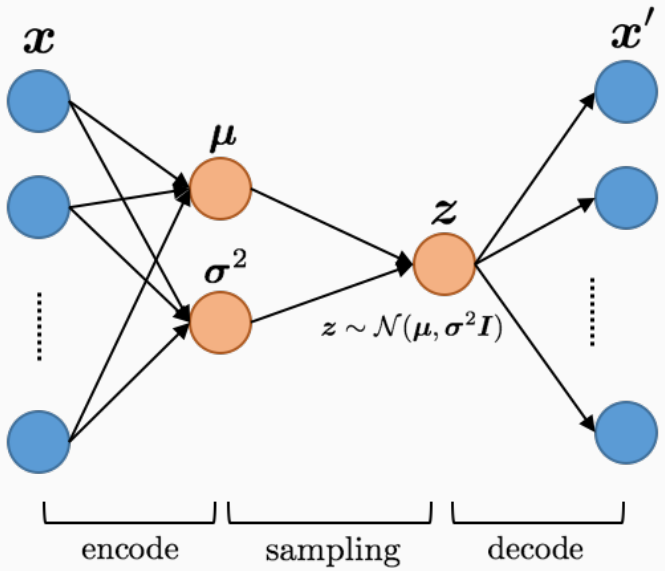
Optimizing the variational lower bound, $\mathcal{L}(\phi, \theta; x)$, traditionally involved computationally expensive techniques, such as Markov-Chain Monte-Carlo. This limited the usefulness of variational Bayesian methods for large-scale datasets.

Kingma & Welling (2014; arXiv:1312.6114) came up with a solution using deep neural networks: the **variational autoencoder (VAE)**.

Variational autoencoders: the basic idea

The basic idea is this, make a deep neural network with an autoencoder-like architecture:

- The input is $\mathbf{x}$
- Early layers compute $q_\phi(\mathbf{z}|\mathbf{x})$ (encoding)
- Use this to get a sample of $\mathbf{z}$ from $q_\phi(\mathbf{z}|\mathbf{x})$ (sampling)
- Later layers compute $p_\theta(\mathbf{x}|\mathbf{z})$ (decoding)
- Then, use gradient descent to minimize $-\mathcal{L}(\phi, \theta; \mathbf{x})$



The reparameterization trick

Cool idea, but...

Problem

If we sample $\mathbf{z}$ from $q_\phi(\mathbf{z}|\mathbf{x})$ stochastically, how can we calculate the gradient of our loss, $-\nabla_\phi \mathcal{L}(\phi, \theta; \mathbf{x})$?

Basically, what is the gradient of a random sample with respect to its distribution parameters?

The reparameterization trick

We can get around this using the "reparameterization trick":

Solution

Alter the way that we parameterize $\mathbf{z}$. Rather than saying:

$$\mathbf{z} \sim \mathcal{N}(\mu(\mathbf{x}; \phi), \sigma(\mathbf{x}; \phi)^2)$$

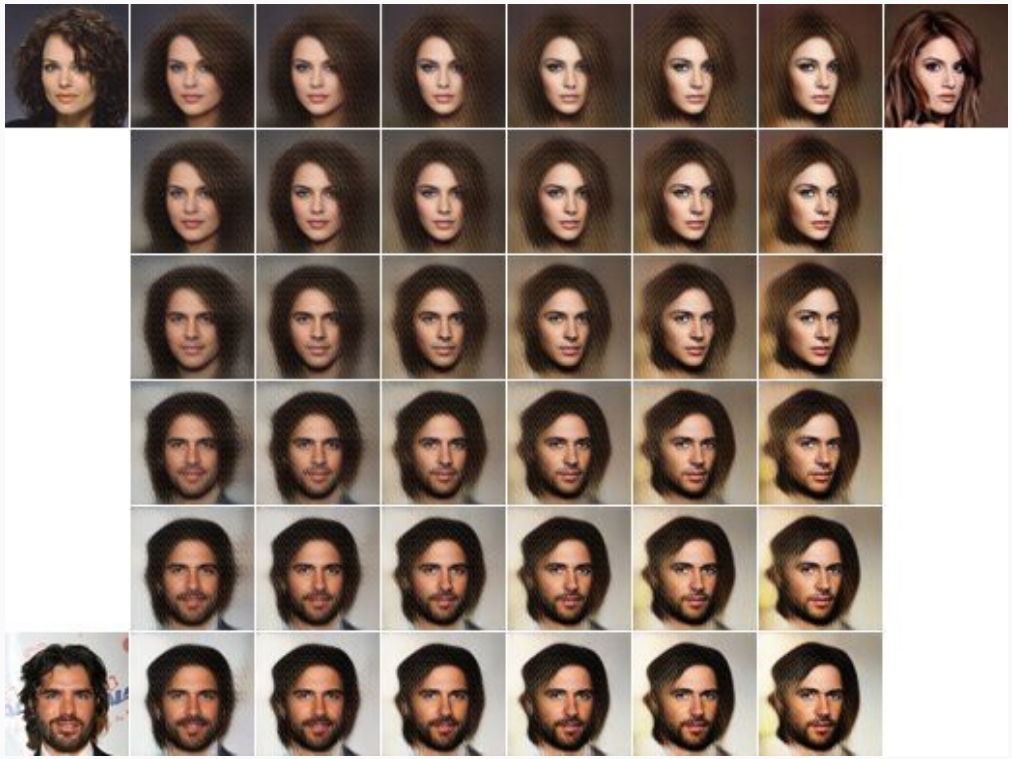
We say:

$$\mathbf{z} = \mu(\mathbf{x}; \phi) + \sigma\epsilon$$
$$\epsilon \sim \mathcal{N}(0, 1)$$

Now, $\mathbf{z}$ is a deterministic function of ϕ and $\mathbf{x}$, and the randomness comes only from ϵ. This allows us to calculate gradients on $-\nabla_\phi \mathcal{L}(\phi, \theta; \mathbf{x})$

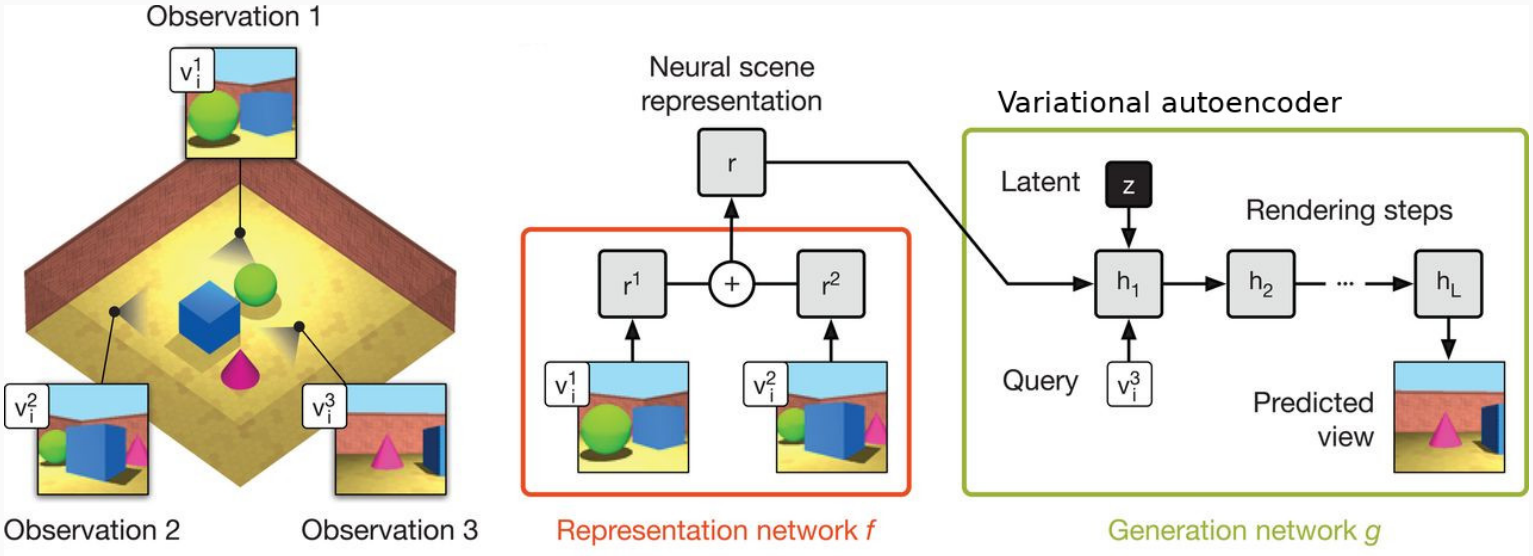
Optimizing VAEs

Thanks to the reparameterization trick, we can now employ all the standard tools of deep learning for optimization (ADAM, dropout, etc.), and standard differentiable architectures (convolutions, LSTMs, etc.) and learn on large datasets to great effect.

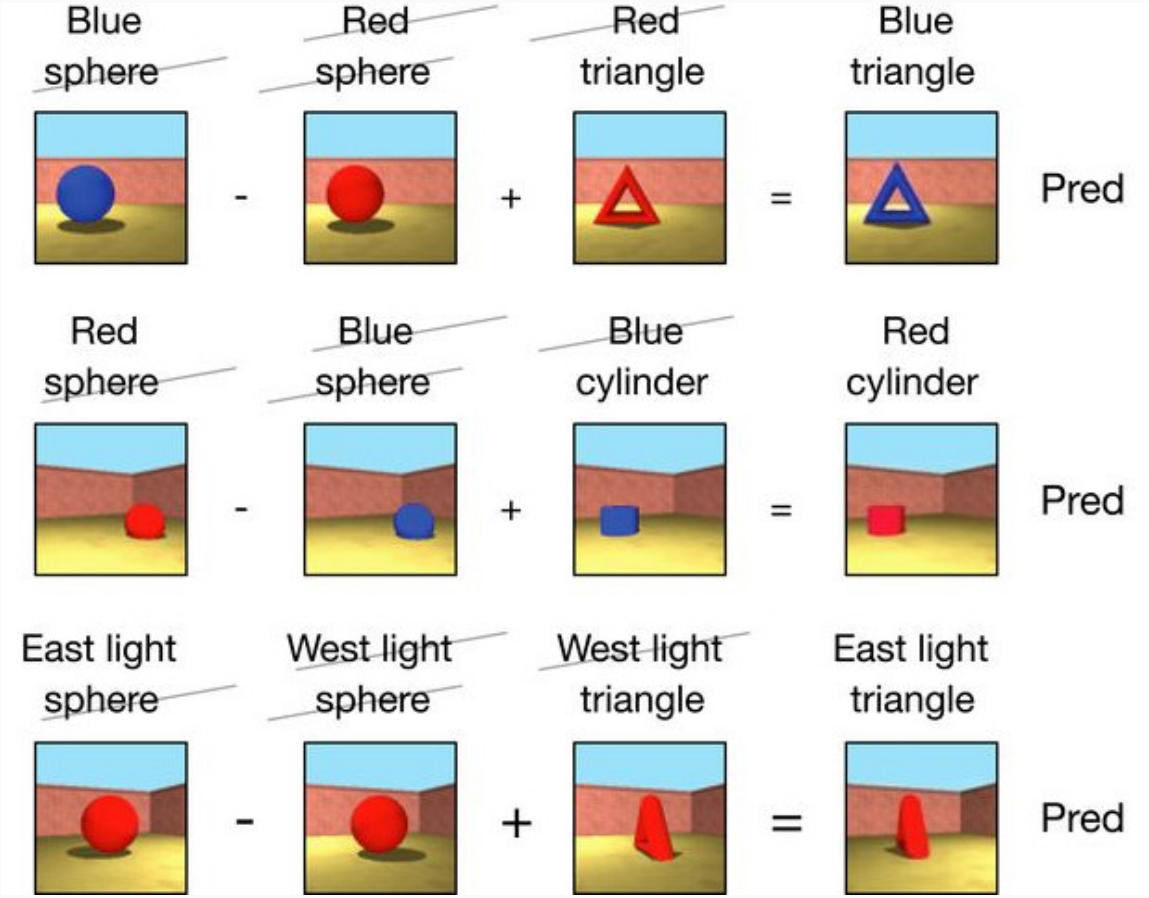


An example of latent variable learning in VAEs

A recent paper from DeepMind illustrates that power of VAEs (a conditional version) for learning appropriate latent spaces:



An example of latent variable learning in VAEs



Interim summary

Here's where we're at:

- Variational Bayesian methods were traditionally too computationally expensive
- VAEs provide a deep learning approach to Variational Bayesian methods with tractable optimization
- The key is the use of an autoencoder-like architecture and the reparameterization trick
- The reparameterization trick lets us use gradient descent to optimize $\mathcal{L}(\phi, \theta; \mathbf{x})$, which makes inference on hard data possible

Example use of VAEs for neuroscience: LFADS

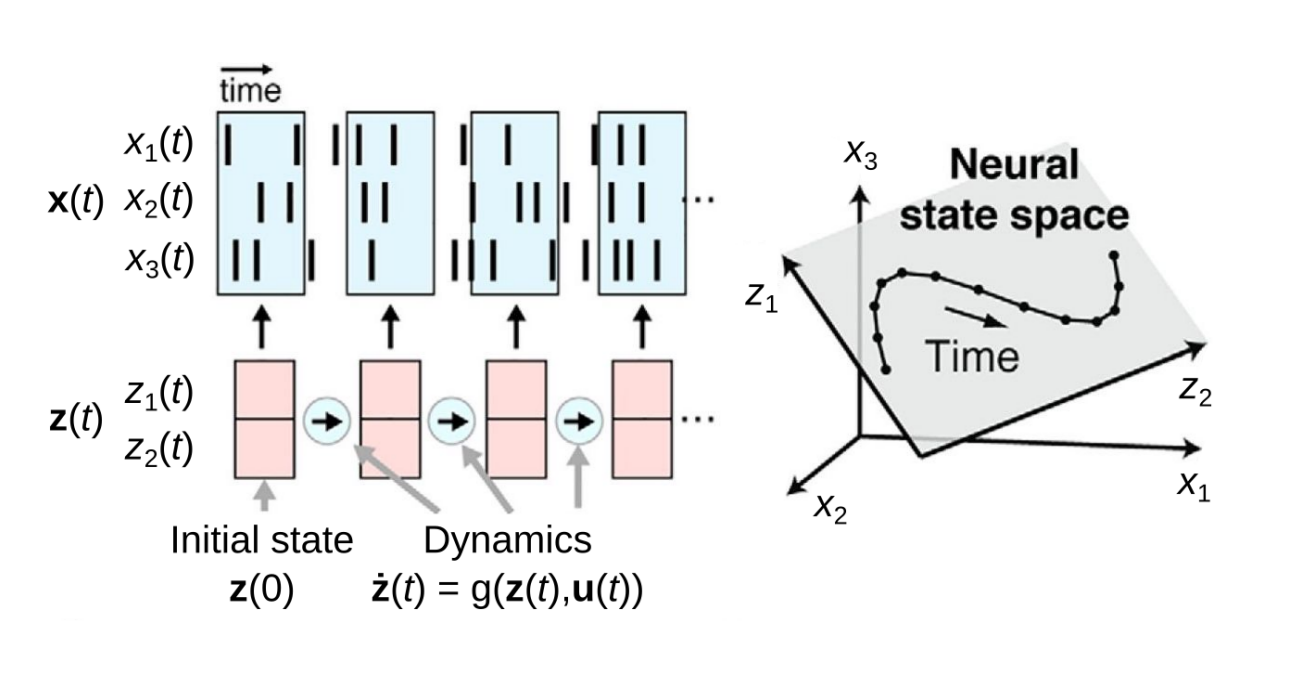
Latent Factor Analysis of Dynamical Systems (LFADS)

Latent Factor Analysis of Dynamical Systems (LFADS) is a VAE approach to analyzing data from large-scale neural recordings (see **`Pandiranath et al. (2018), Nature Methods, 15:805`**)

- It's based on the idea that the activity we record (e.g. spiking data from NeuroPixels) is actually just a reflection of latent variables composed of a lower-dimensional dynamical system.
- This lower dimensional system could reflect connectivity constraints, motor programs, etc.
- So, in order to get at the values we really care about, we need to do latent variable inference.

Latent Factor Analysis of Dynamical Systems (LFADS)

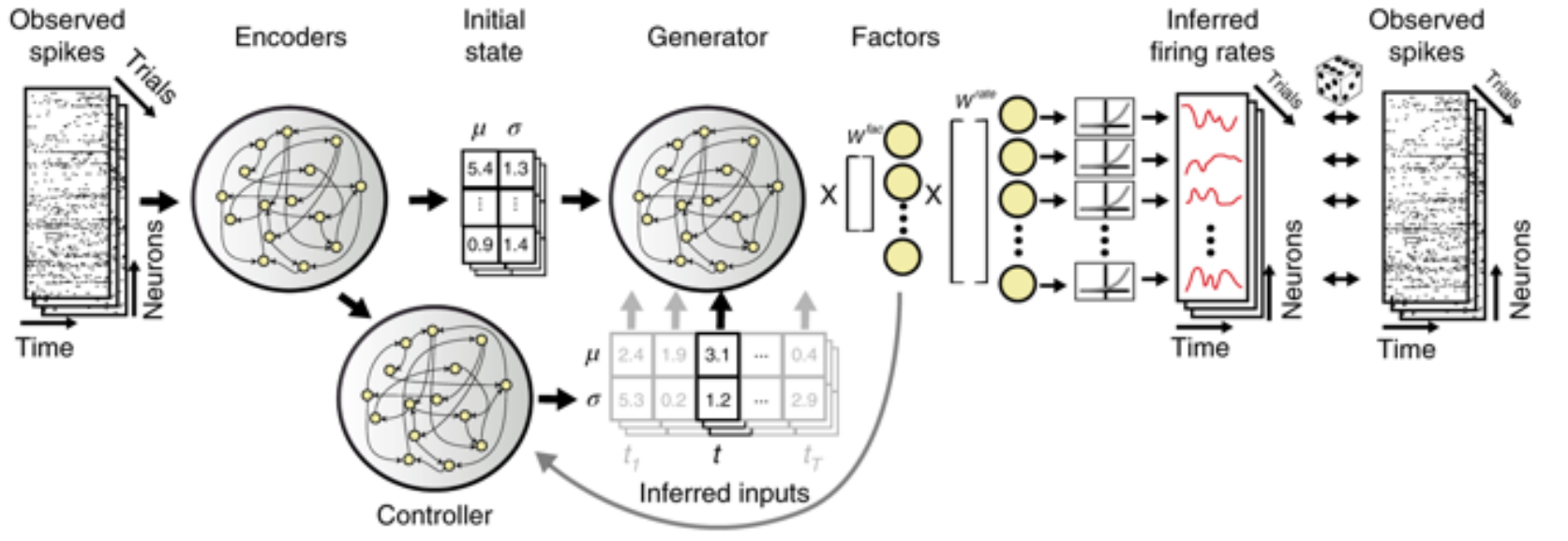
Basic idea: use a recurrent VAE to infer the initial state and internal dynamics of the latent space system, then analyse the latent space versions of the data, rather than the data itself.



Pandiranath et al. (2018), J. Neurosci., 38:9390

Latent Factor Analysis of Dynamical Systems (LFADS)

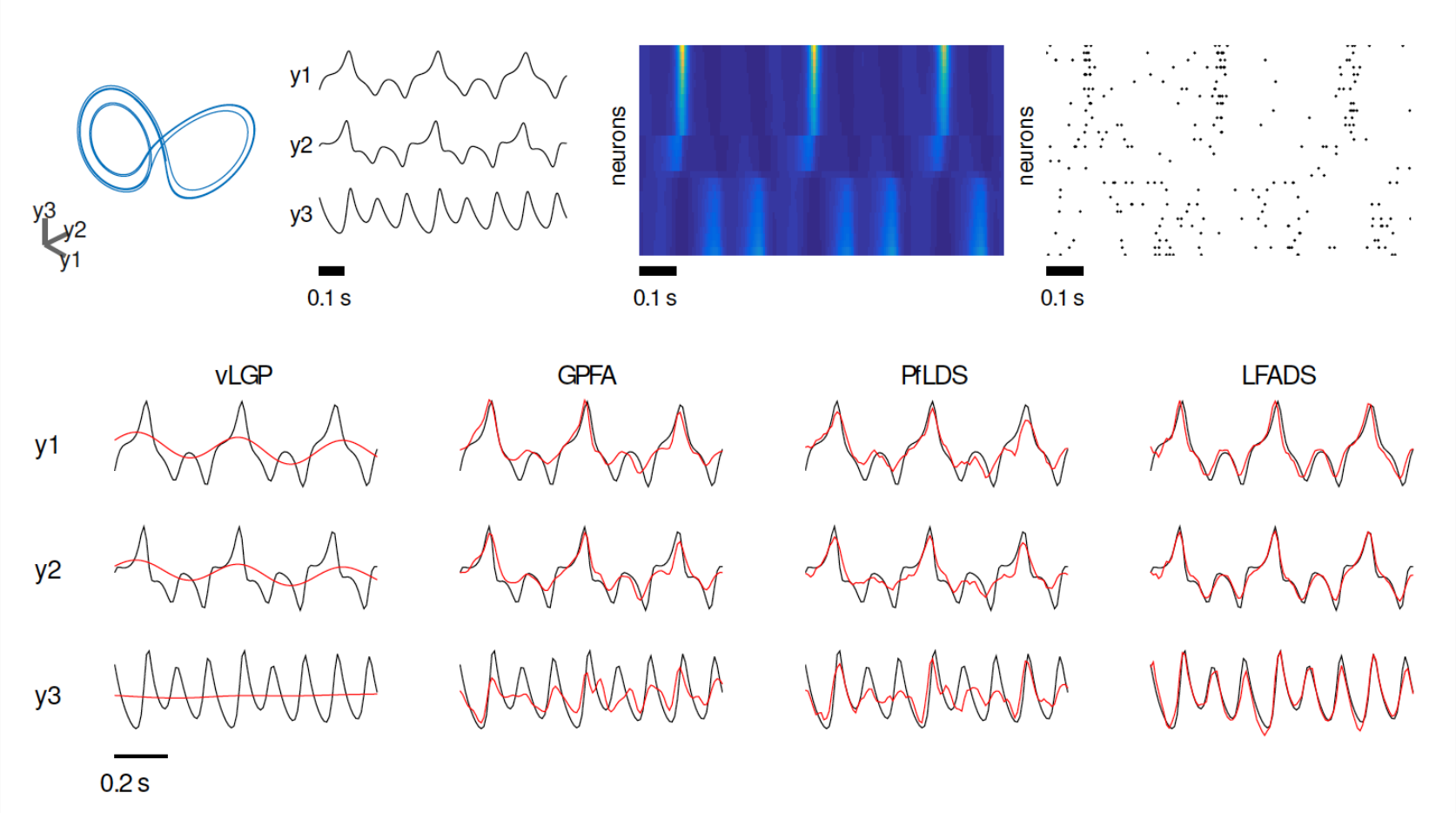
Here is the architecure of an LFADS network:



Pandiranath et al. (2018), Nature Methods, 15:805

Example use of LFADS on synthetic data

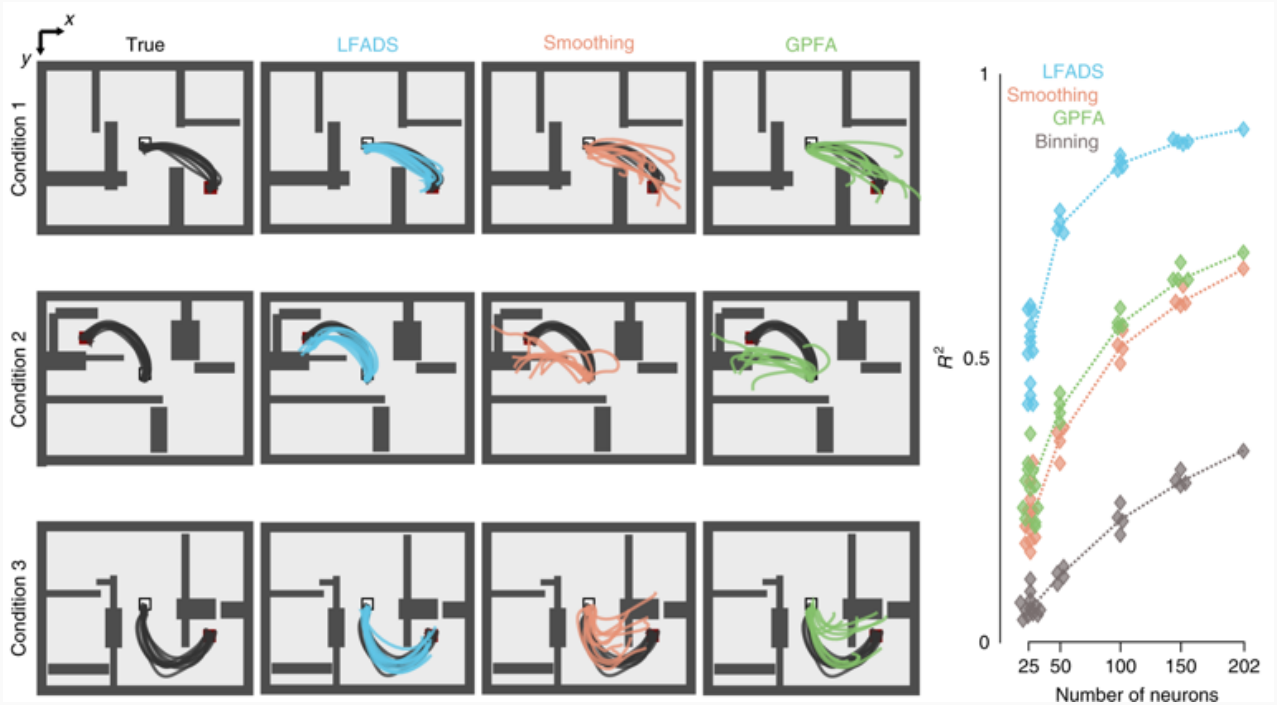
LFADS can infer the true latent variables in synthetic datasets:



Sussillo et al. (2016), arXiv: 1608.06315

Example use of LFADS on real data

Decoding behavior from LFADS latent variables can be better than standard techniques:



Pandiranath et al. (2018), Nature Methods, 15:805

Example code

Running LFADS in PyTorch

To the ~~Batmobile~~ `Jupyter Notebook`!

Code is available at
`https://github.com/lyprince/lfads_demo`