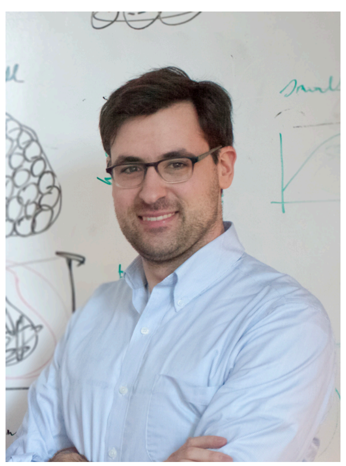
Composing graphical models with neural networks like chocolate and peanut butter

https://youtu.be/O7oD_oX-Gio


David Duvenaud


Alex Wiltchko


Matthew D. Hoffman


Dustin Tran


Scott Linderman


Sandeep Robert Datta


Ryan P. Adams

Google AI

Matthew J Johnson (mattjj@google.com)

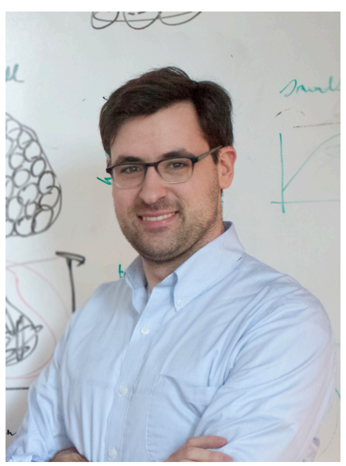
July 22 2019 @ UAI 2019

Composing graphical models with neural networks like chocolate and peanut butter

https://youtu.be/O7oD_oX-Gio

or

Graphical models and exponential families in the age of differentiable programming



David Duvenaud | **Alex Wiltchko** | **Matthew D. Hoffman** | **Dustin Tran** | **Scott Linderman** | **Sandeep Robert Datta** | **Ryan P. Adams**

Google AI

Matthew J Johnson (mattjj@google.com)

July 22 2019 @ UAI 2019

Goals

Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen

Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen
2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory

Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen
2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory
3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation

Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen
2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory
3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation
4. Make SVAEs and related PGM + DNN architectures super obvious so that you can invent better ones

Goals

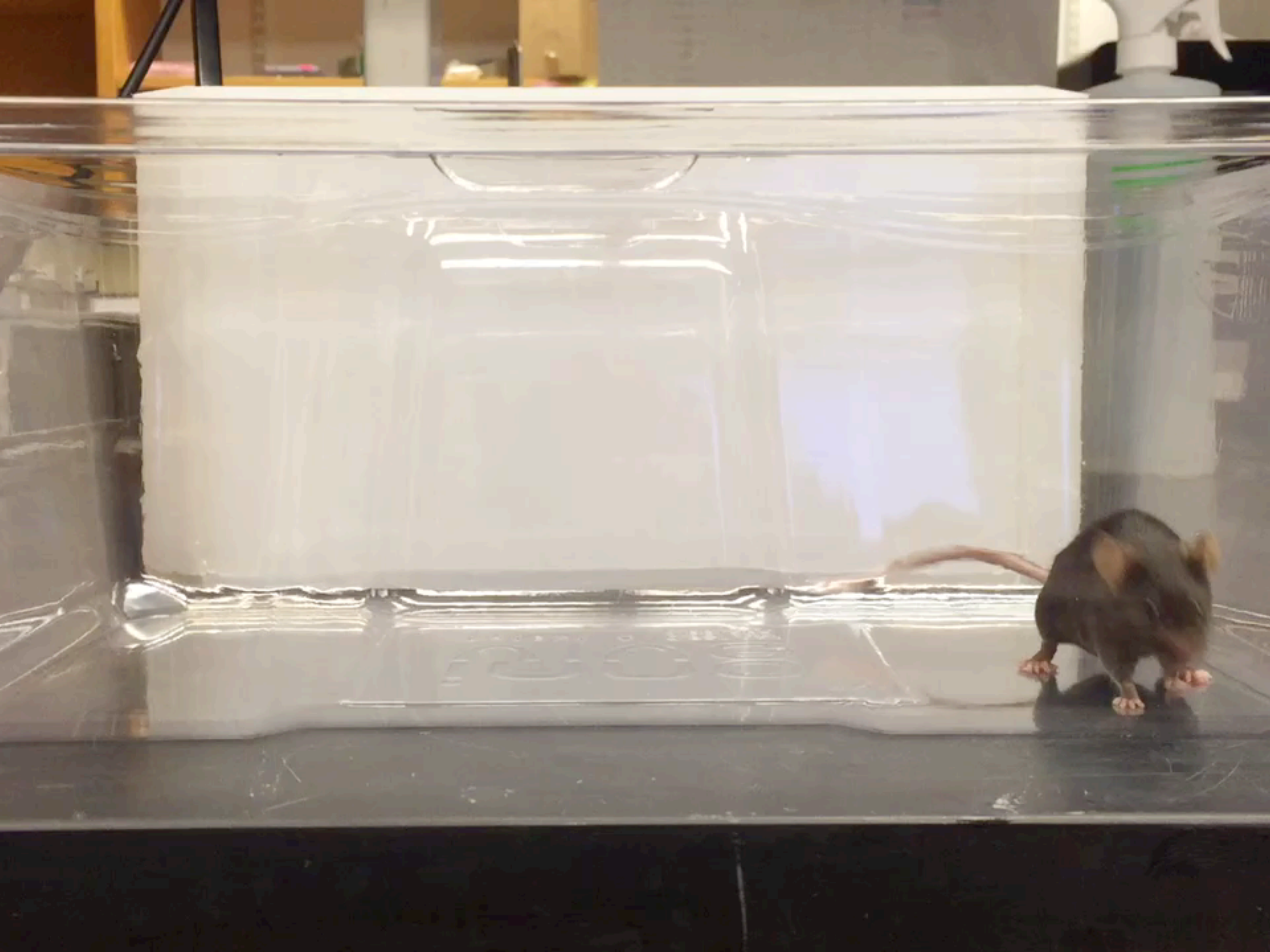
1. Motivate why PGMs + DNNs are a revolution waiting to happen
2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory
3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation
4. Make SVAEs and related PGM + DNN architectures super obvious so that you can invent better ones

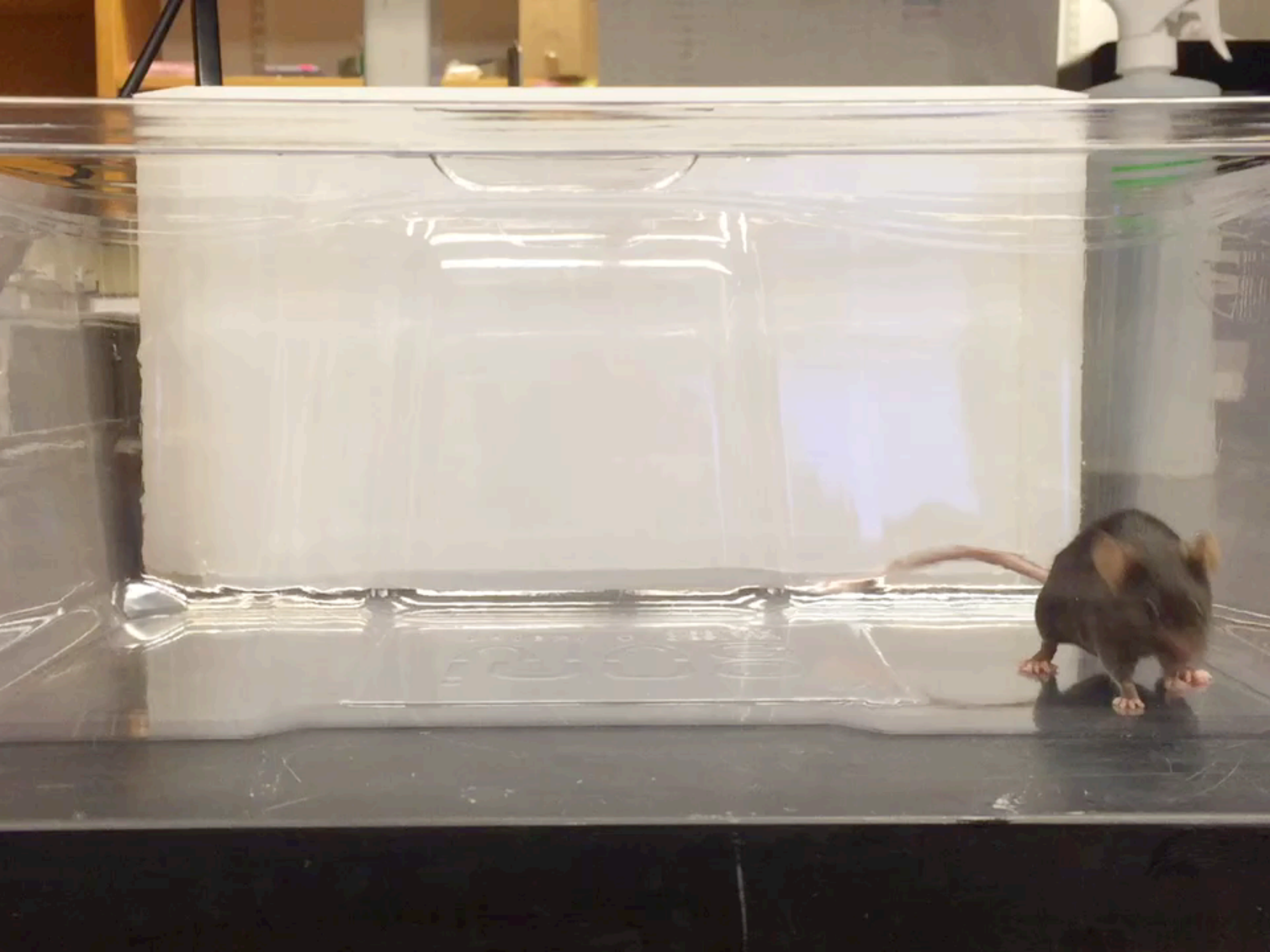
Non-goals

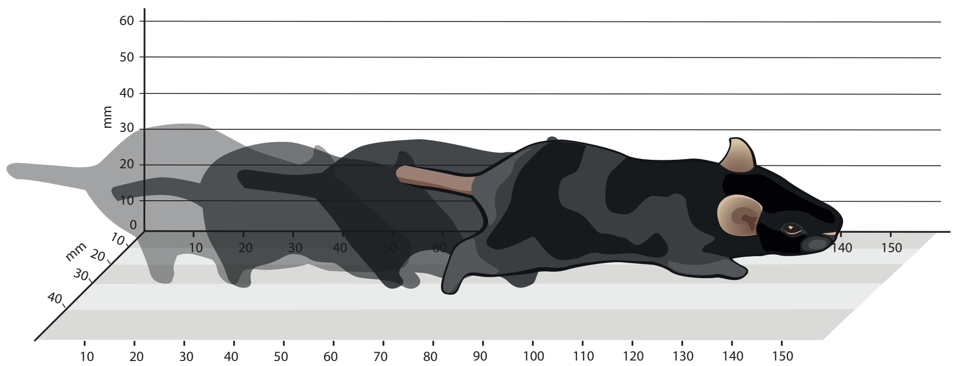
1. Cover the recent literature on PGMs + DNNs
2. Unpack all the technical details

Goals

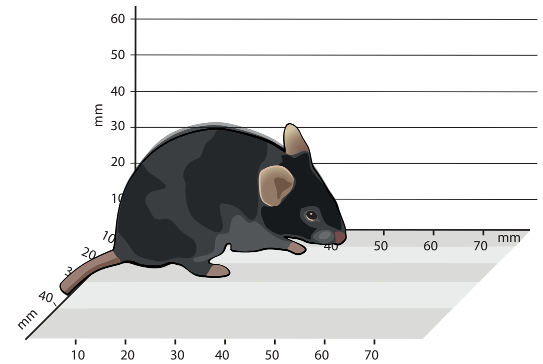
1. Motivate why PGMs + DNNs are a revolution waiting to happen
2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory
3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation
4. Make SVAEs and related PGM + DNN architectures super obvious so that you can invent better ones



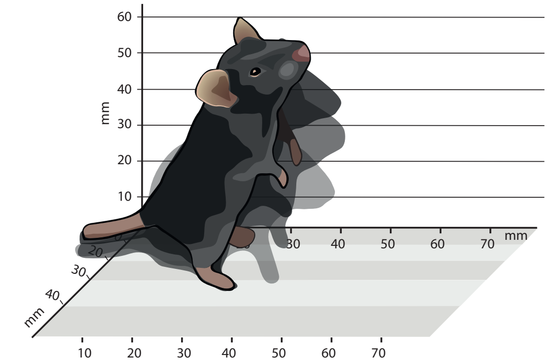




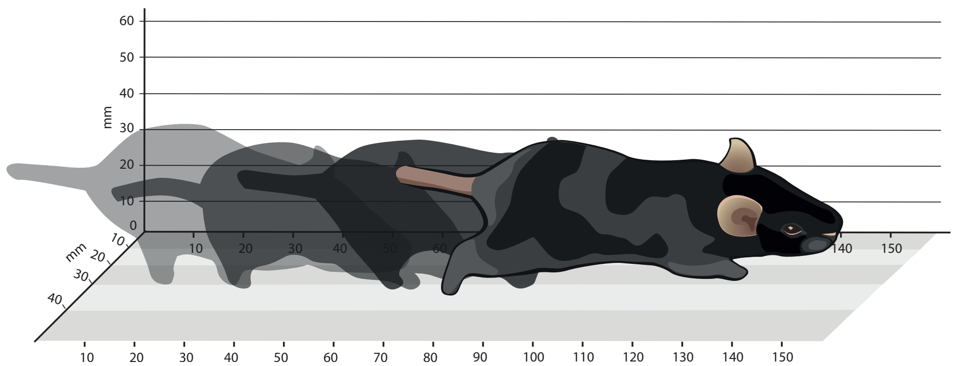
dart



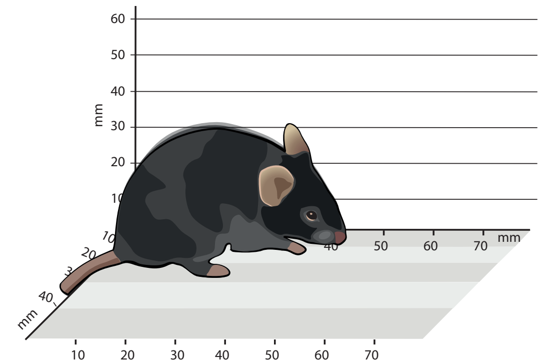
pause



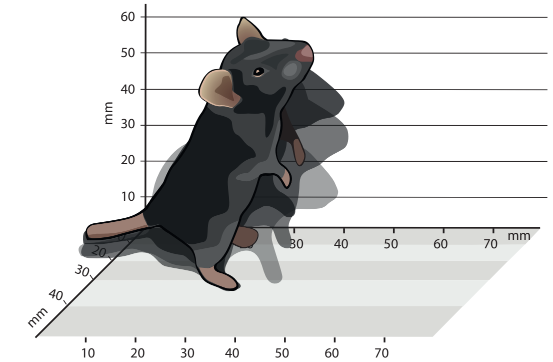
rear



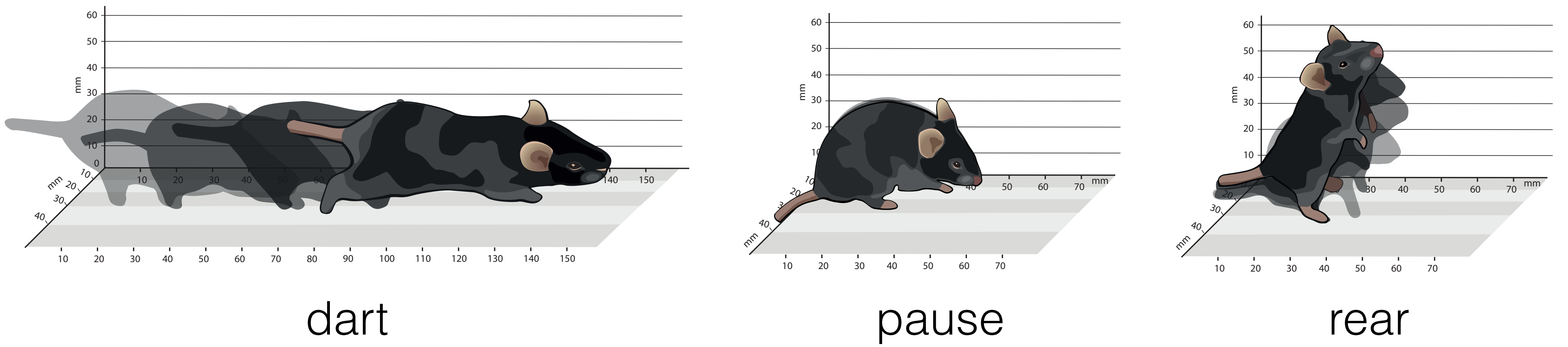
dart

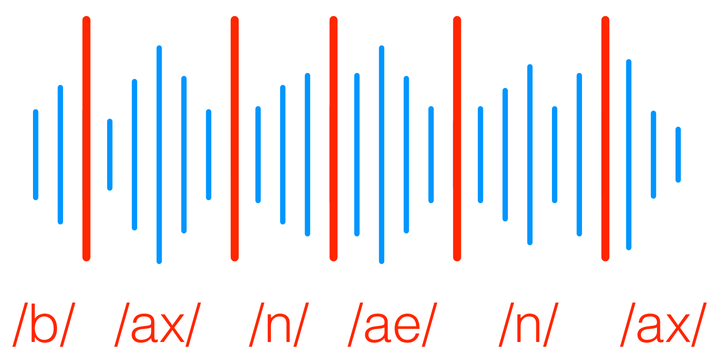


pause



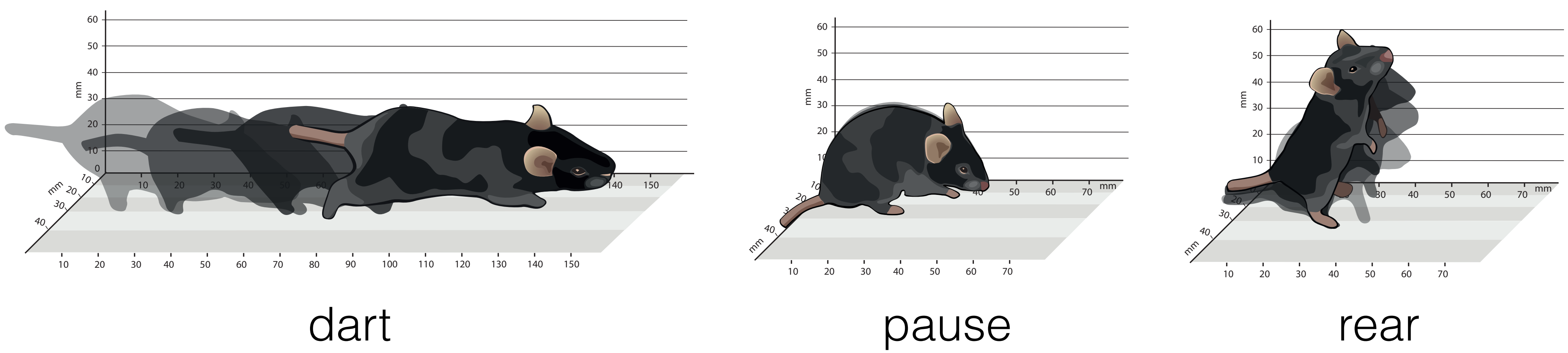
rear

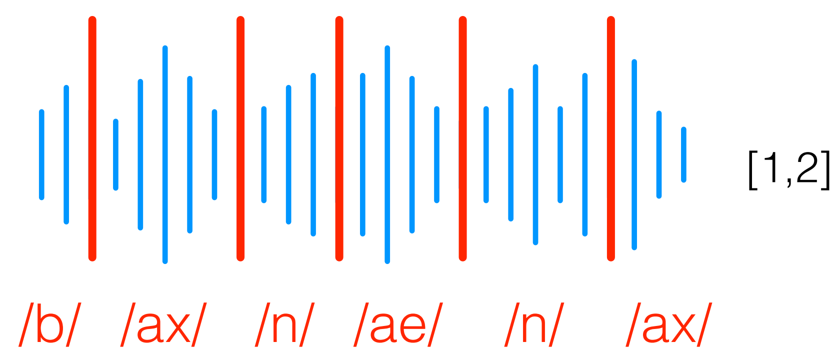


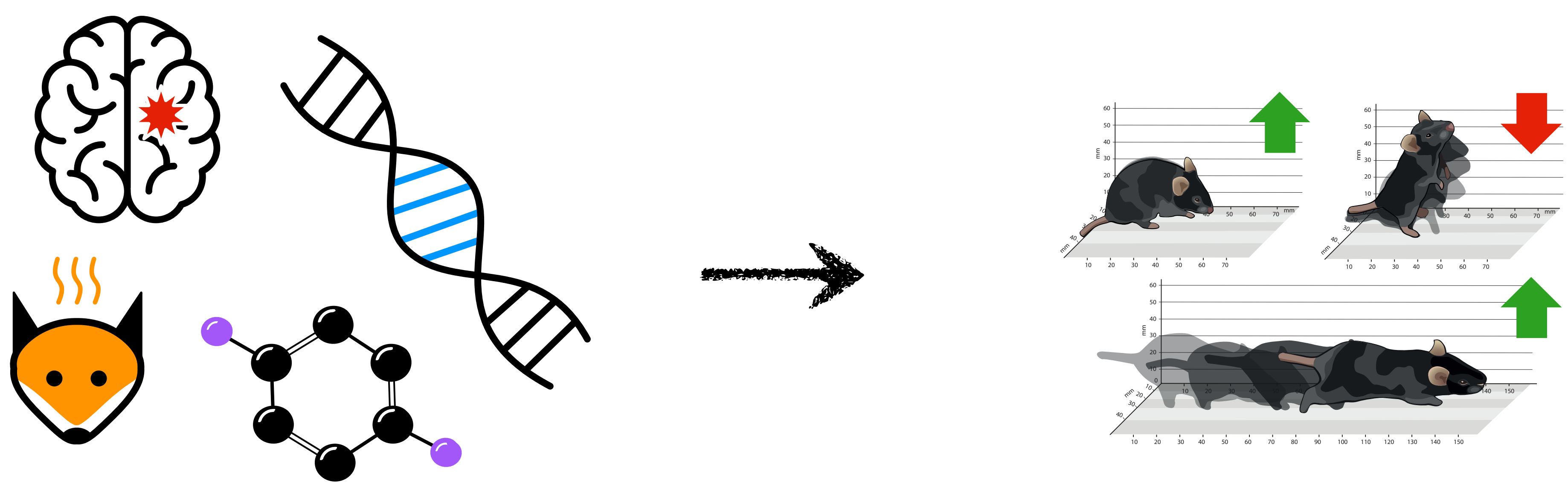


[1,2]

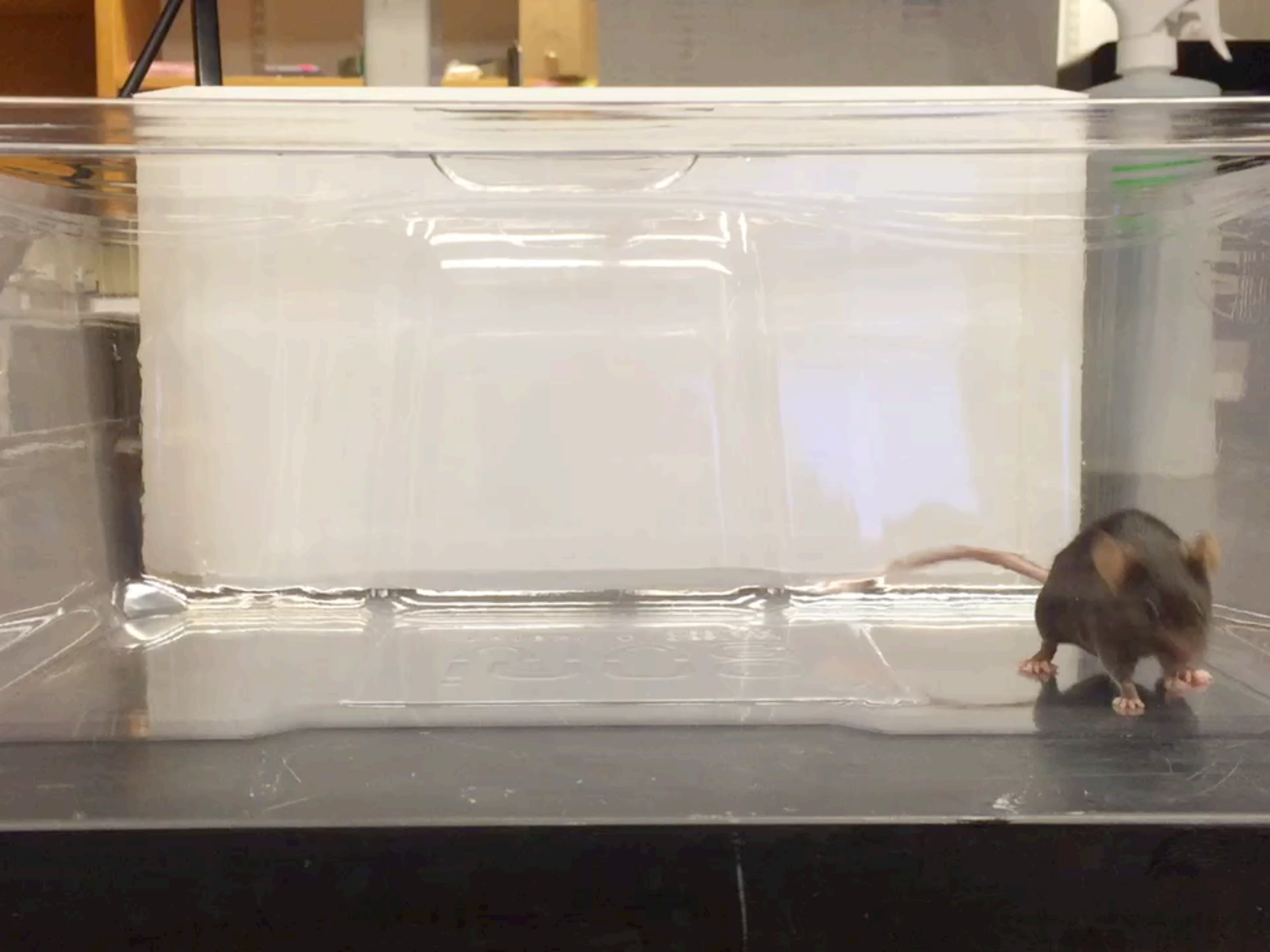
[1] Lee and Glass. A Nonparametric Bayesian Approach to Acoustic Model Discovery. ACL 2012.
[2] Lee. Discovering Linguistic Structures in Speech: Models and Applications. MIT Ph.D. Thesis 2014.

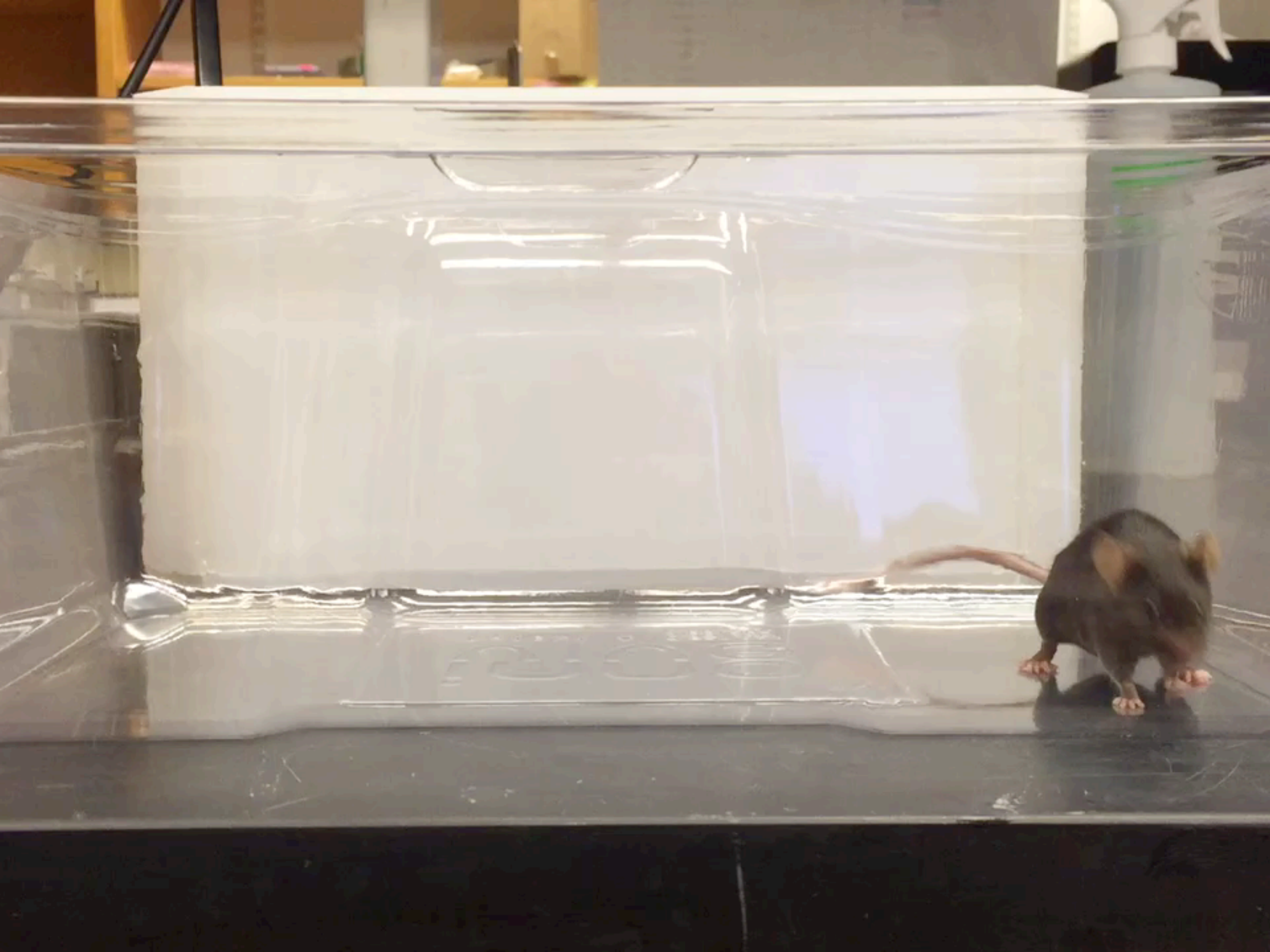


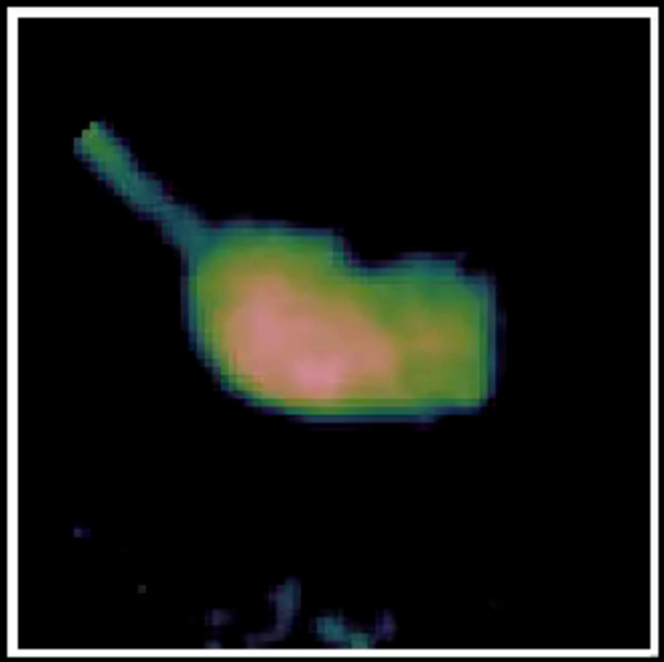




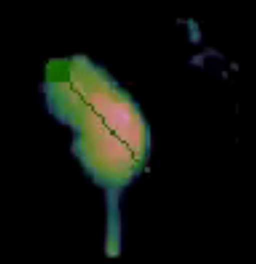
[1] Lee and Glass. A Nonparametric Bayesian Approach to Acoustic Model Discovery. ACL 2012.
[2] Lee. Discovering Linguistic Structures in Speech: Models and Applications. MIT Ph.D. Thesis 2014.





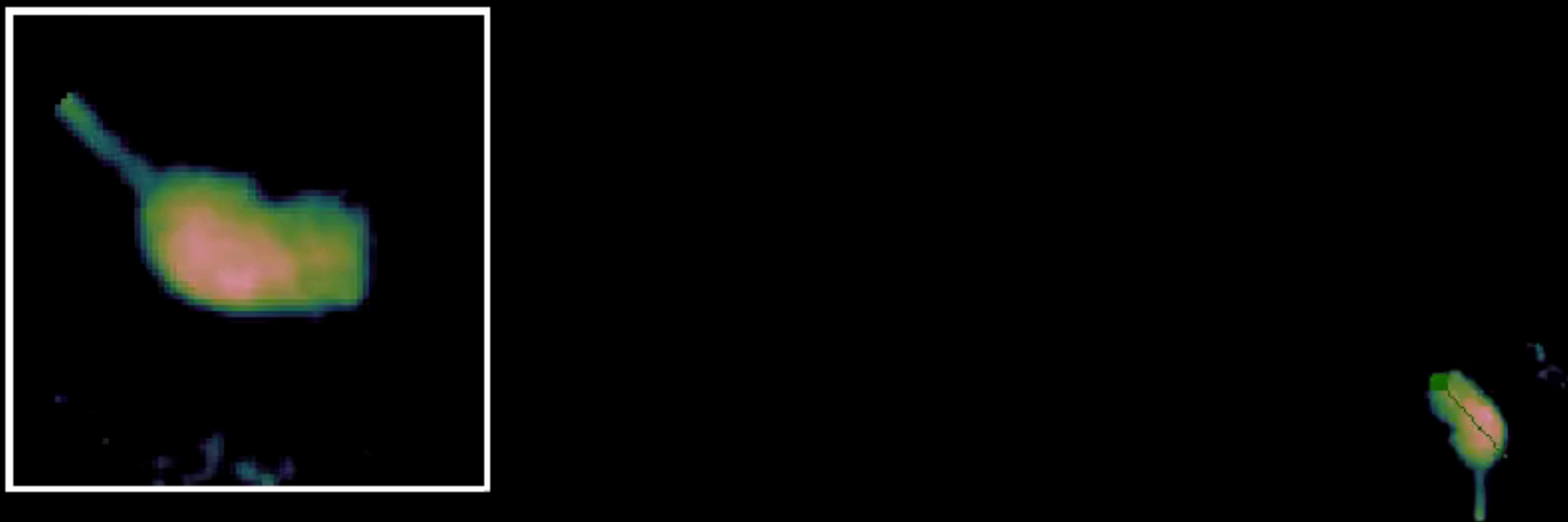


Frame 0



Alexander Wiltschko, **Matthew Johnson**, et al., Neuron 2015.

Frame 0



Alexander Wiltschko, **Matthew Johnson**, et al., Neuron 2015.

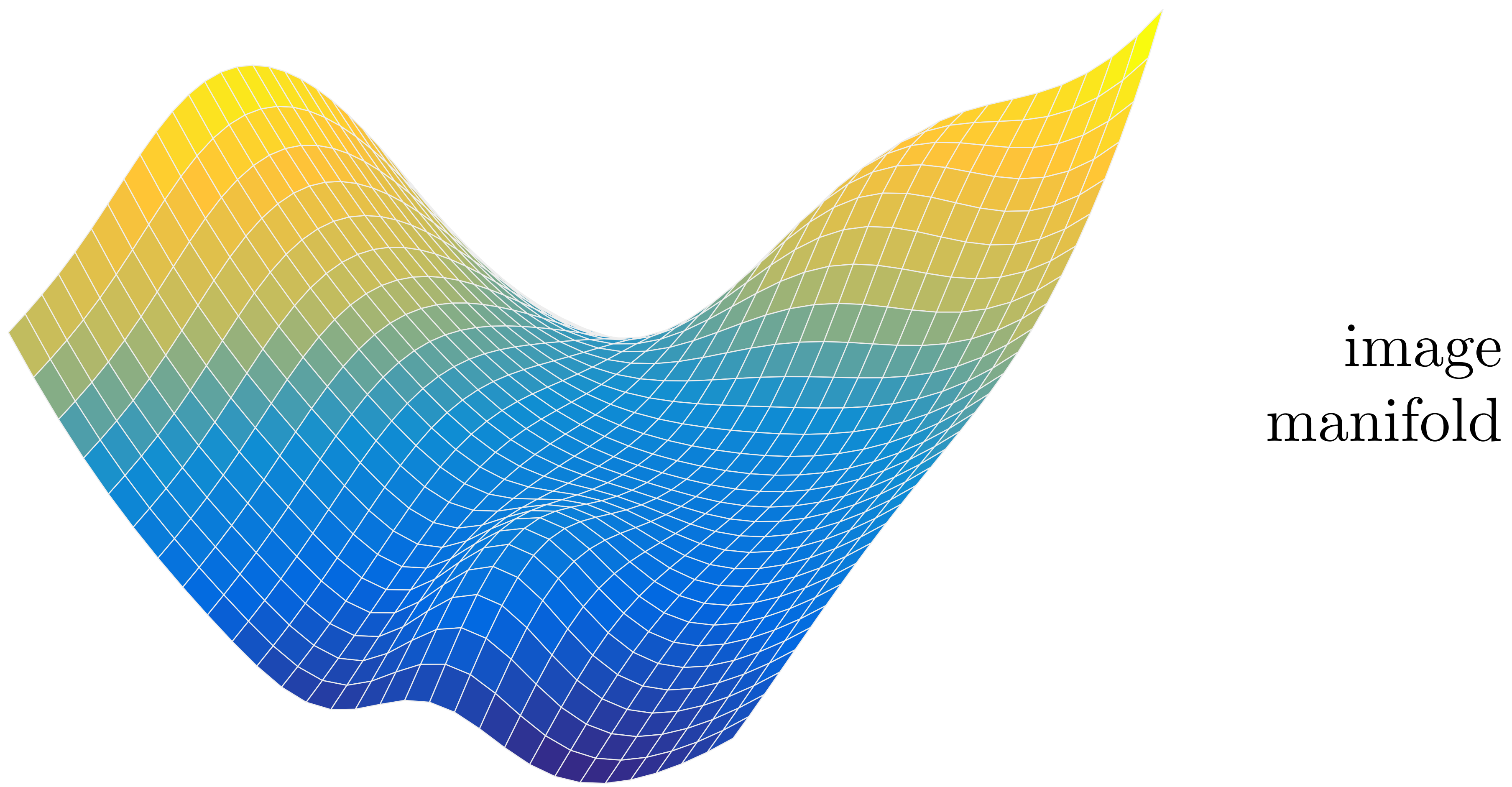
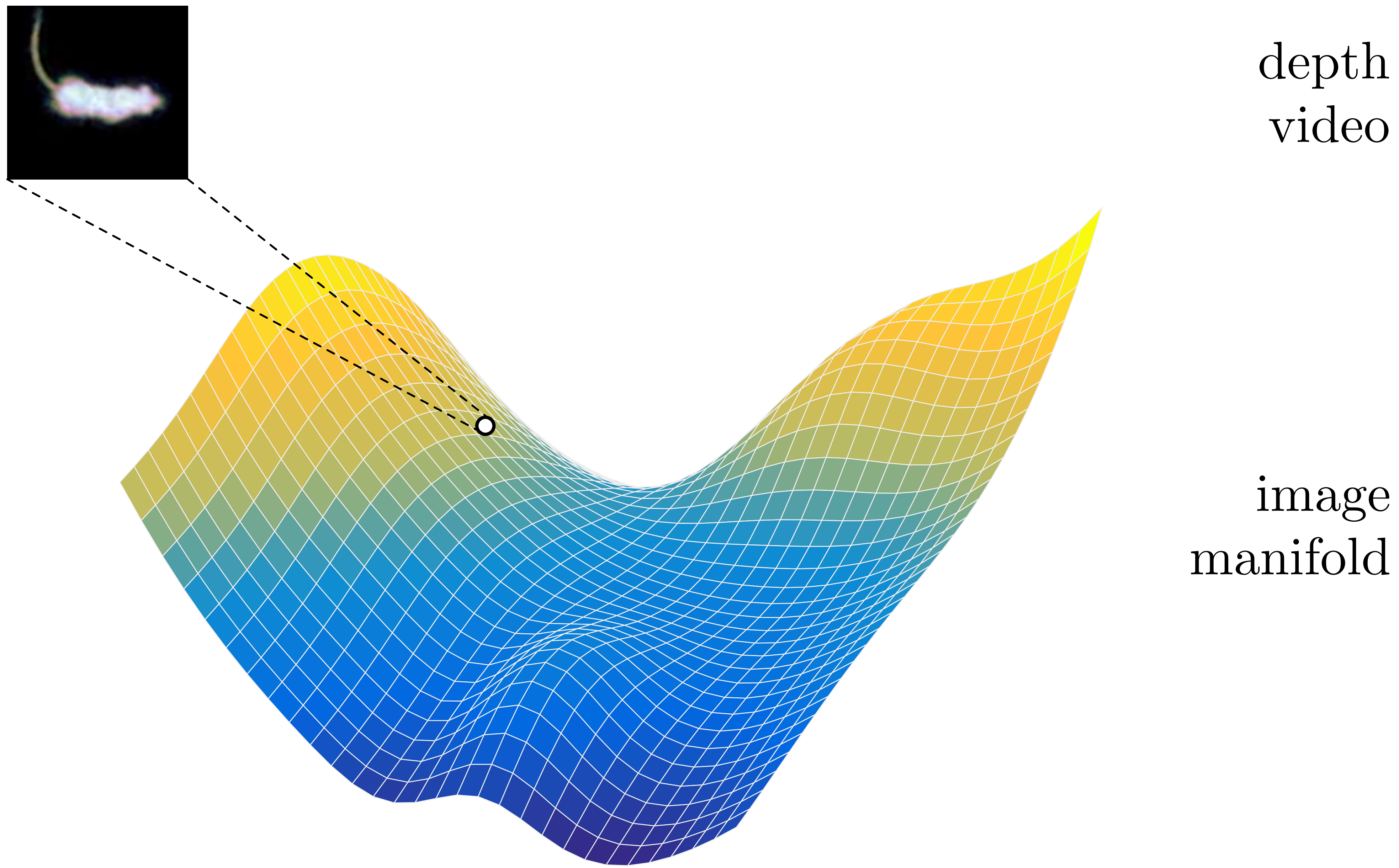
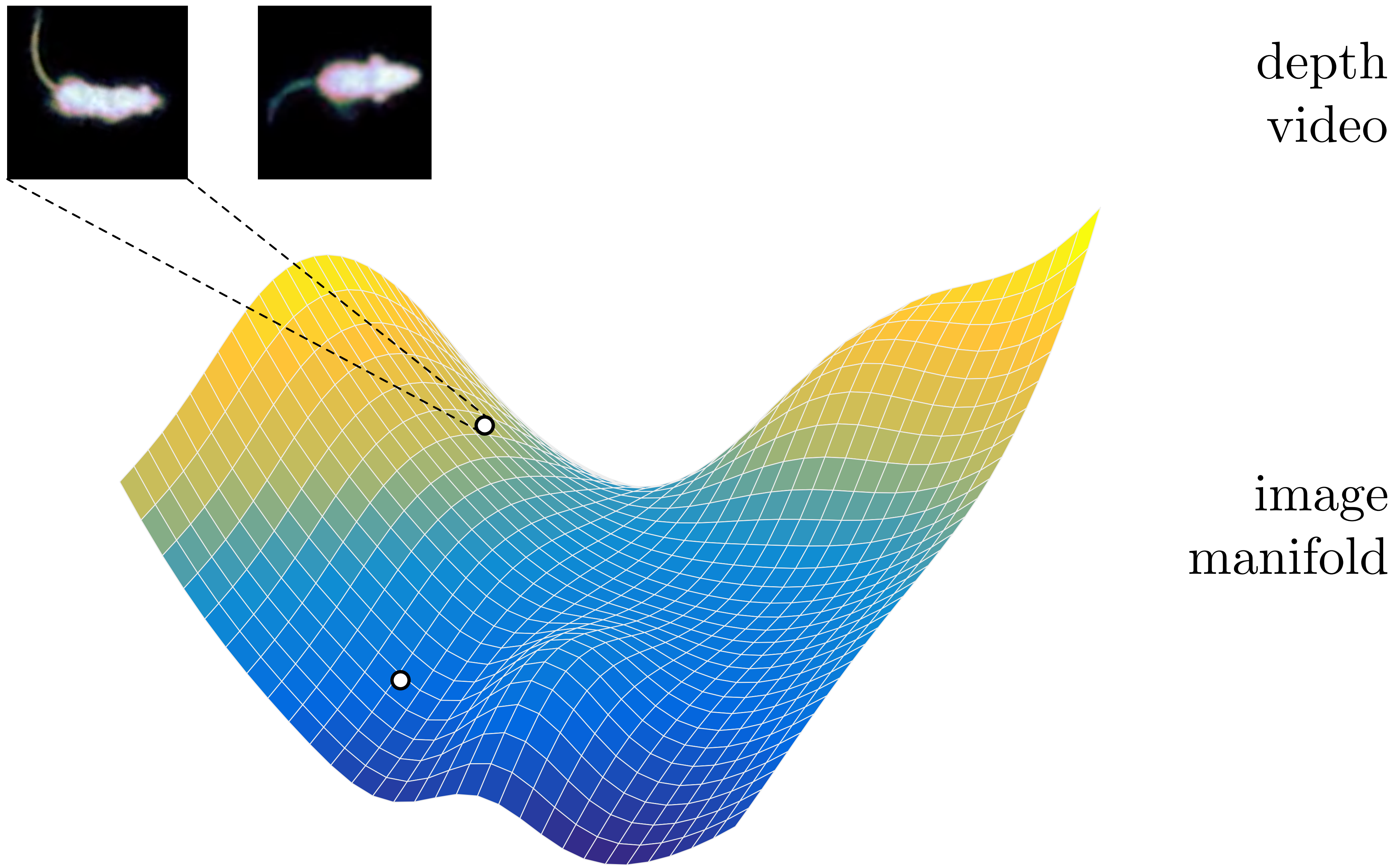
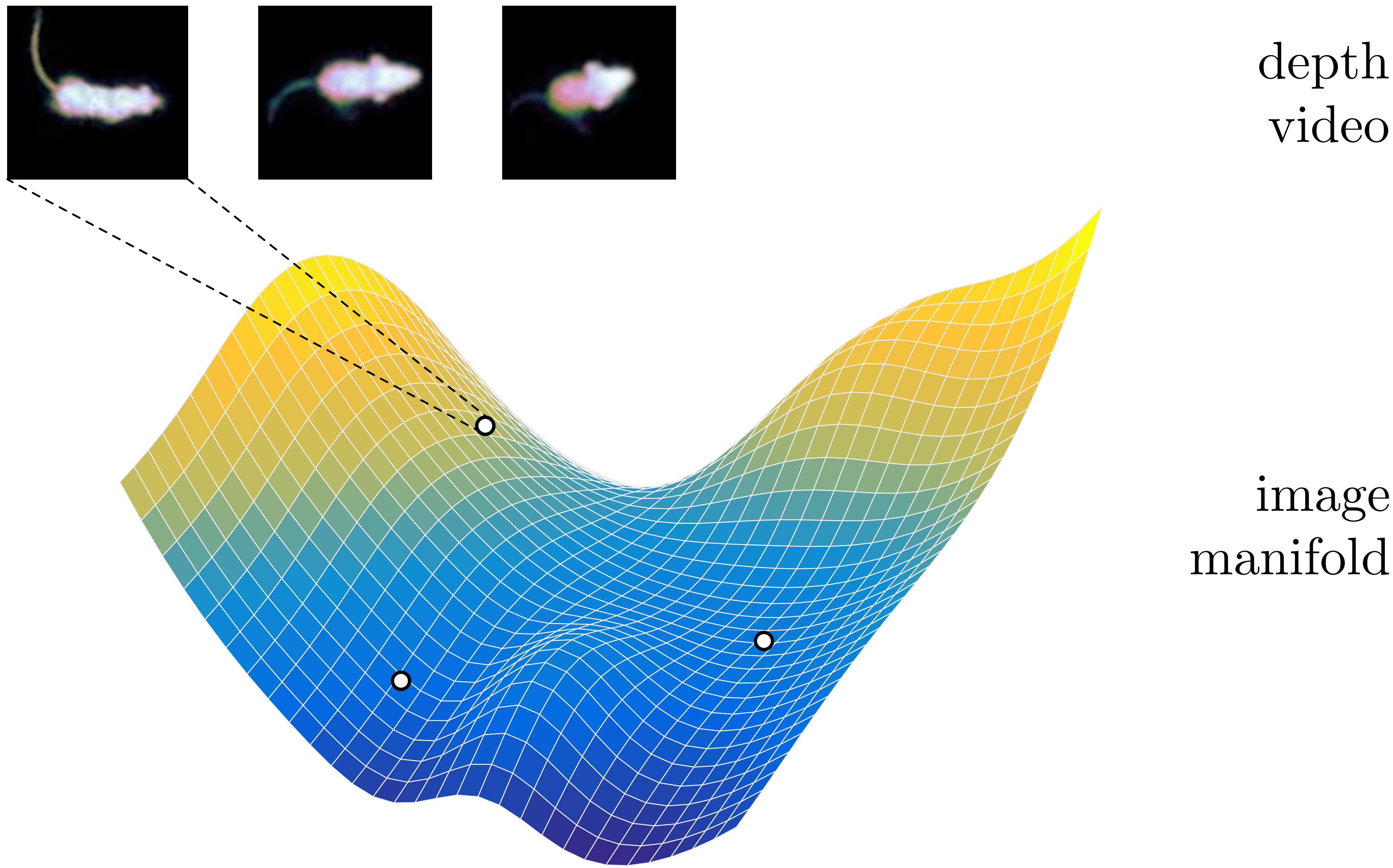
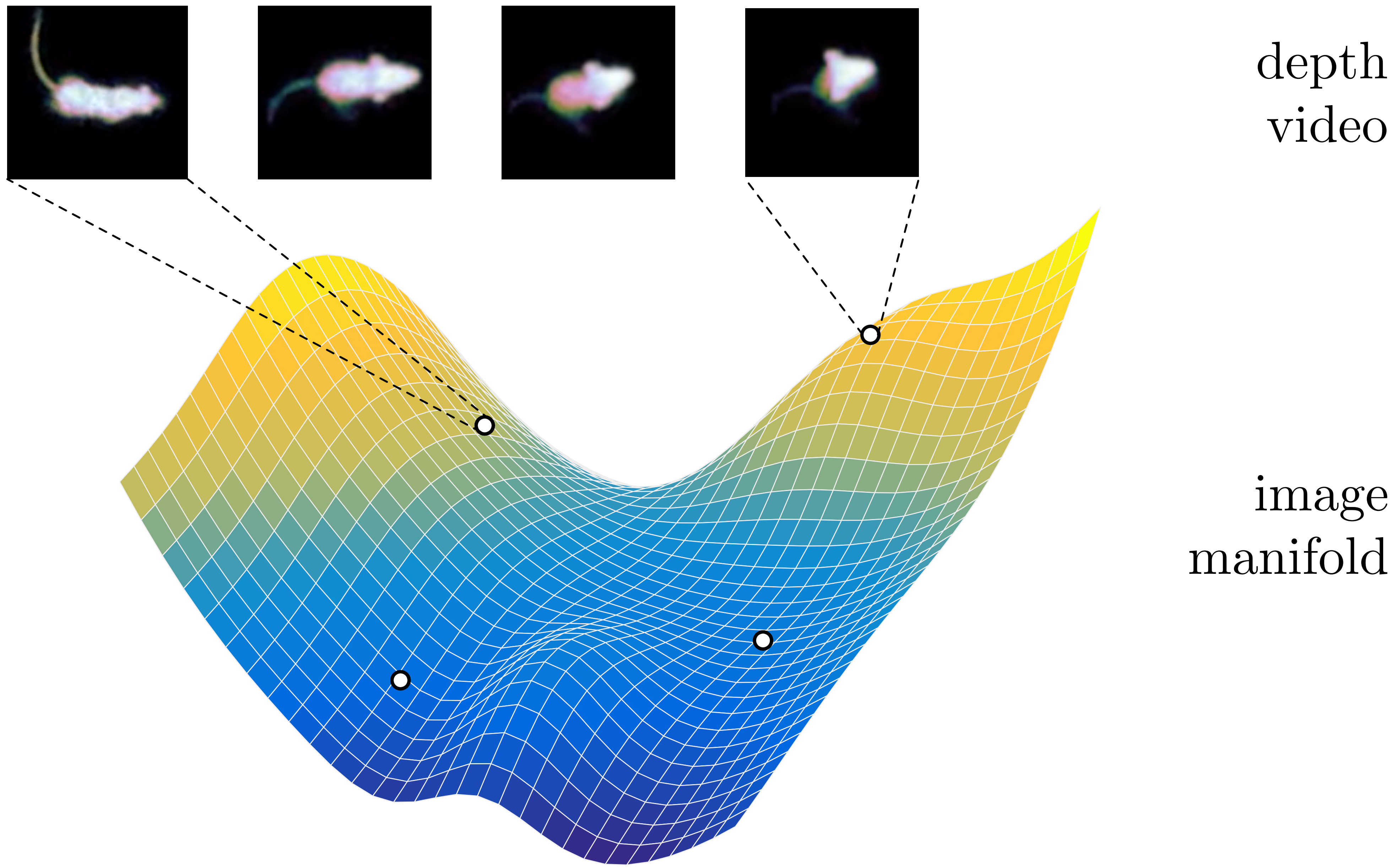


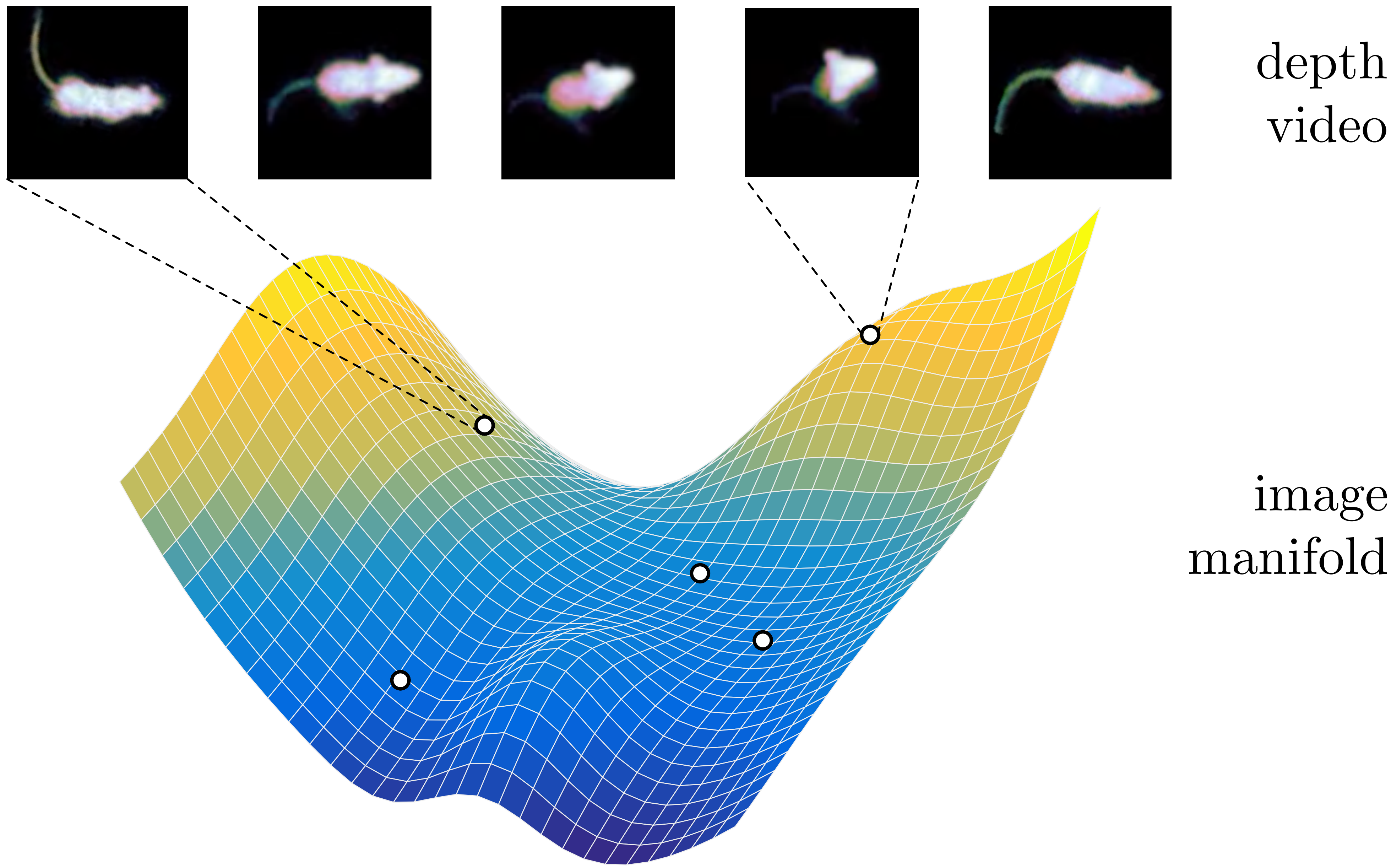
image
manifold

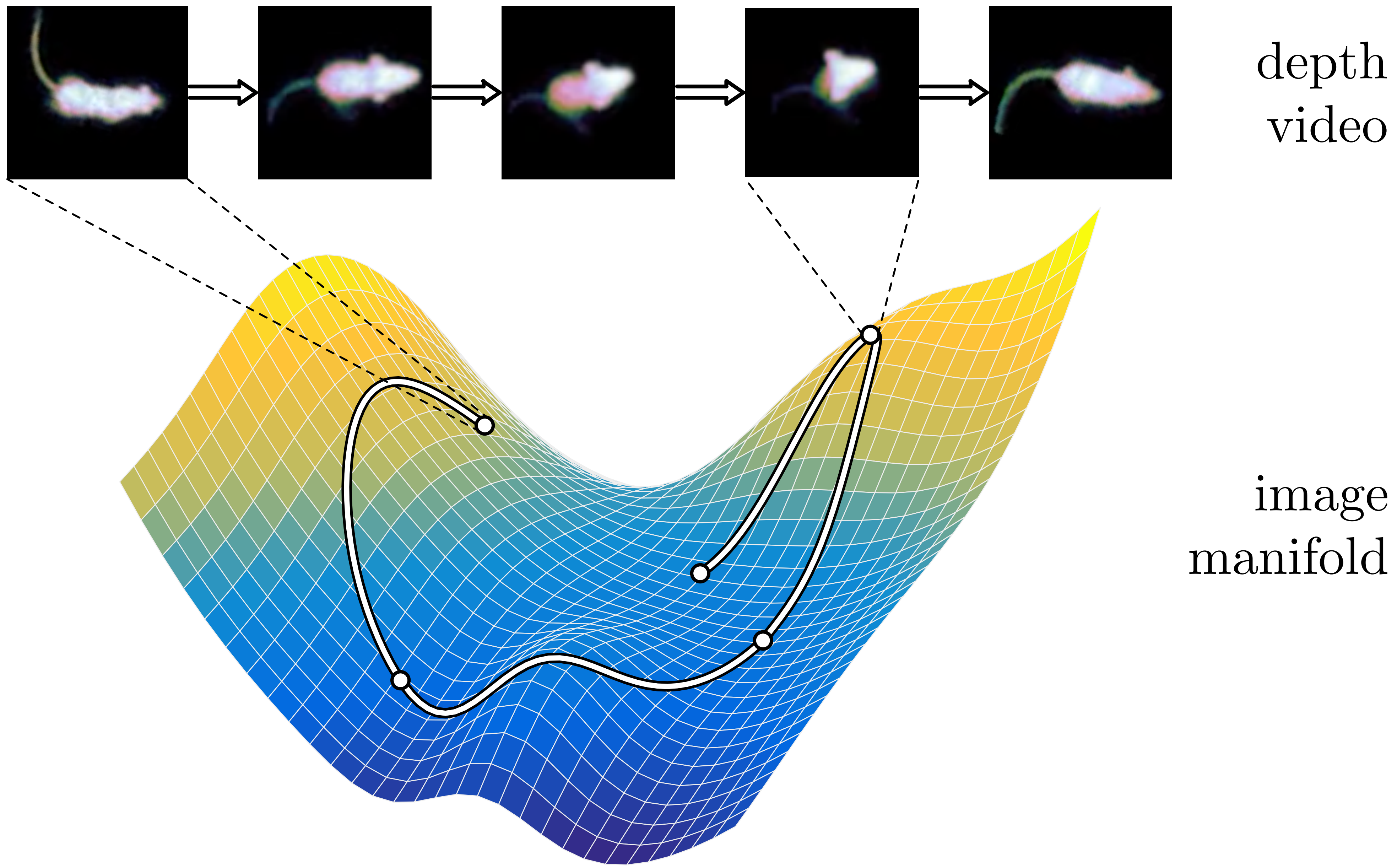

depth
video
image
manifold

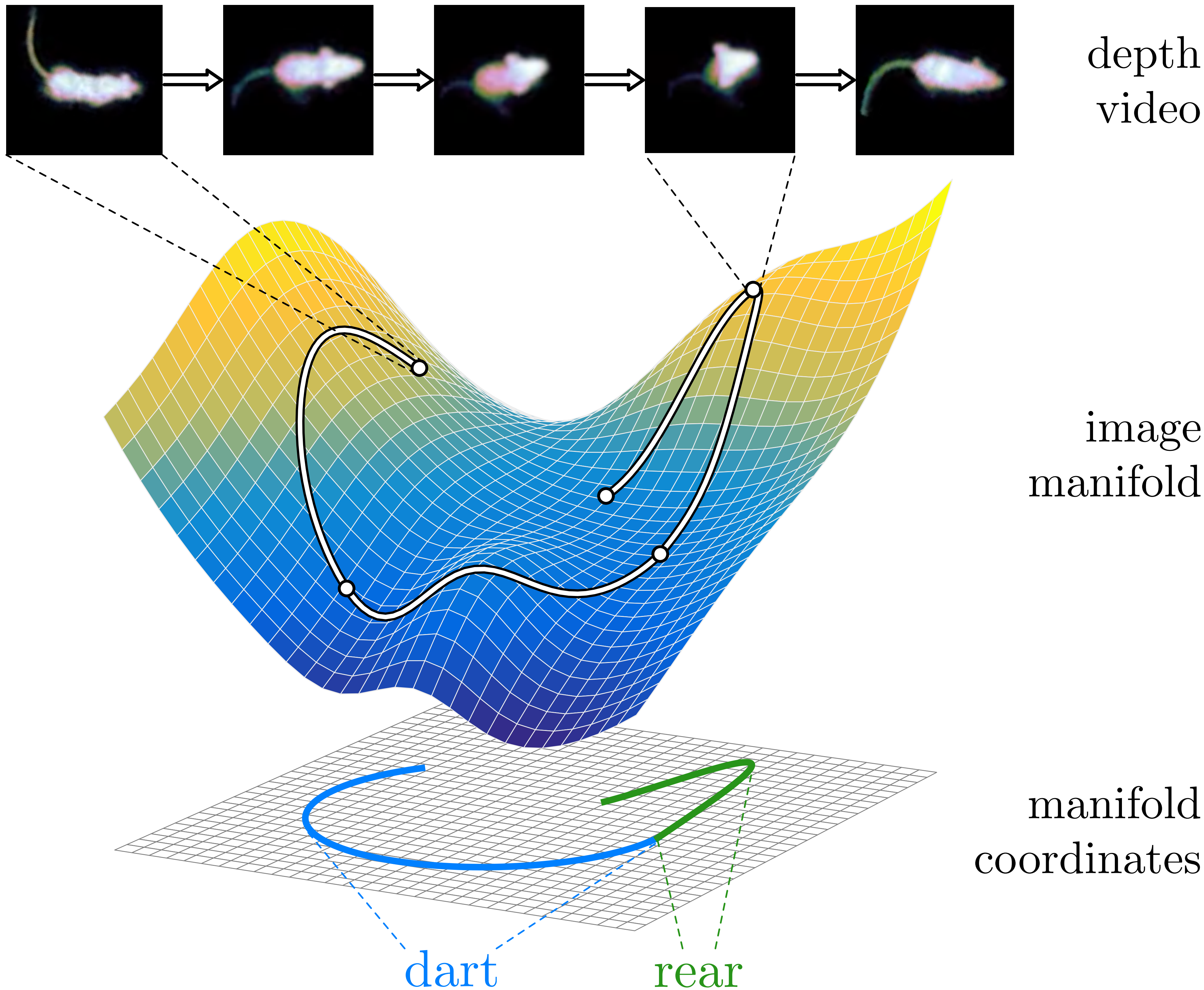

depth
video
image
manifold


depth
video
image
manifold


depth
video
image
manifold


depth
video
image
manifold


depth
video
image
manifold


depth
video
image
manifold
manifold
coordinates
dart
rear

Recurrent neural networks? [1,2,3]

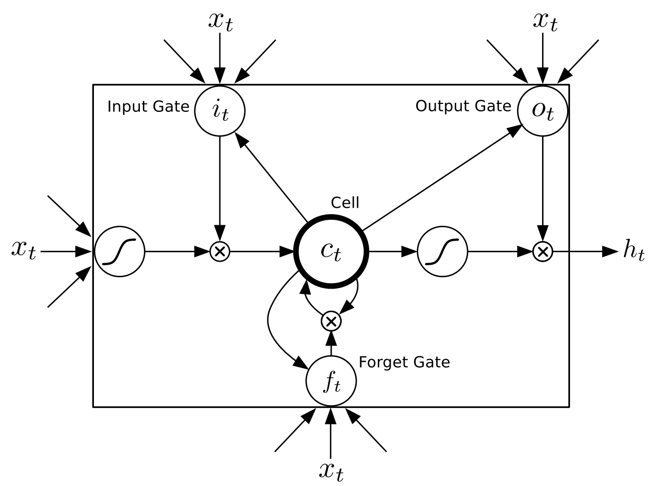


Figure 1. LSTM unit

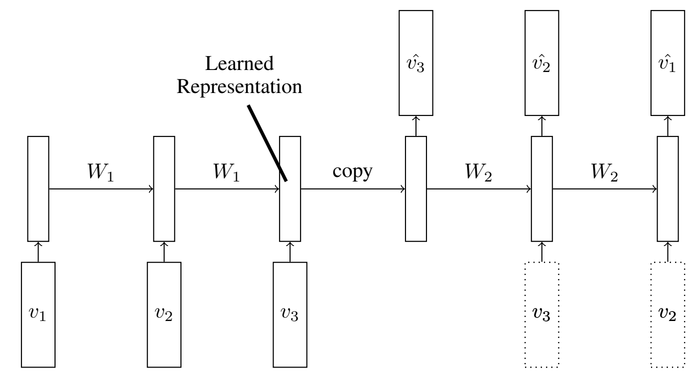


Figure 2. LSTM Autoencoder Model

[1] Srivastava, Mansimov, Salakhutdinov. Unsupervised learning of video representations using LSTMs. ICML 2015.
[2] Ranzato, MarcAurelio, et al. Video (language) modeling: a baseline for generative models of natural videos. Preprint 2015.
[3] Sutskever, Hinton, and Taylor. The Recurrent Temporal Restricted Boltzmann Machine. NIPS 2008.

Recurrent neural networks? [1,2,3]

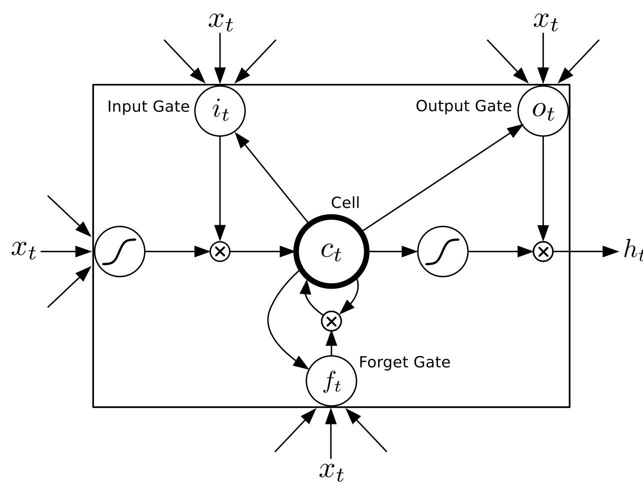


Figure 1. LSTM unit

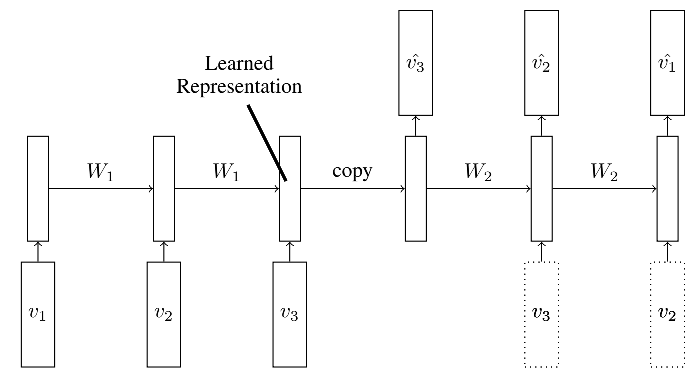
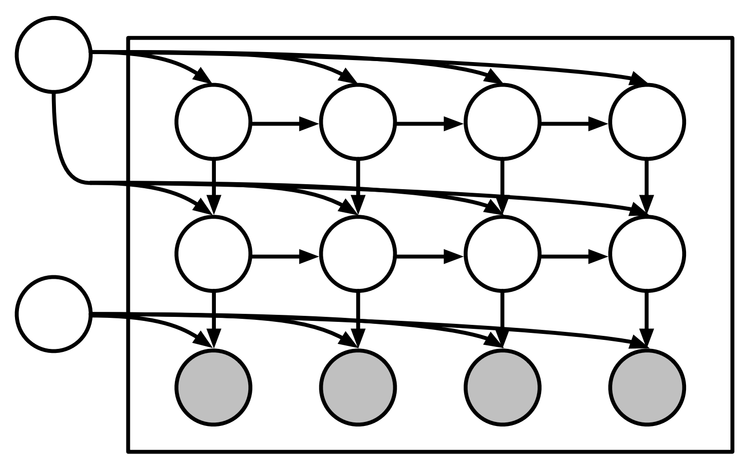
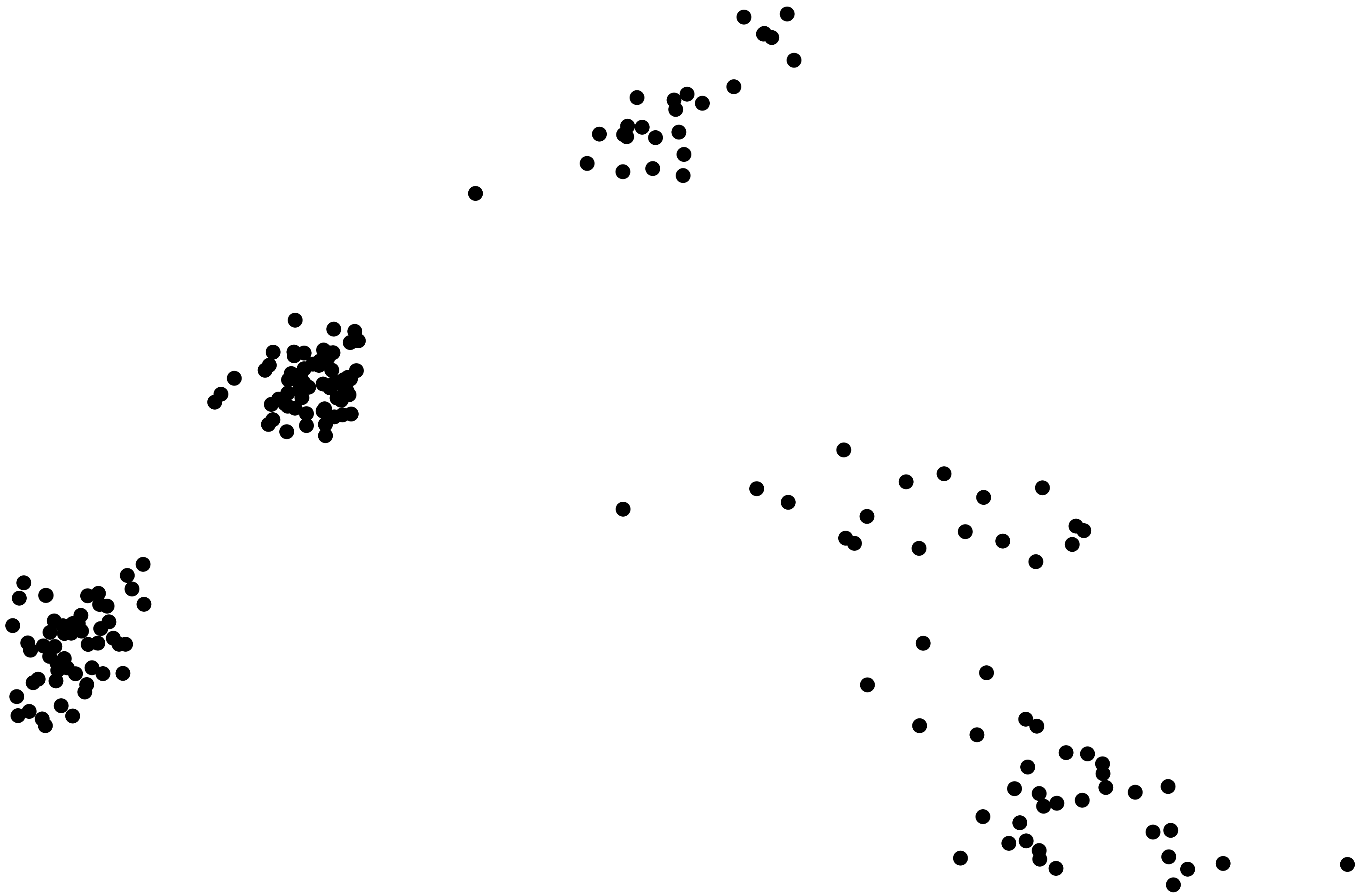


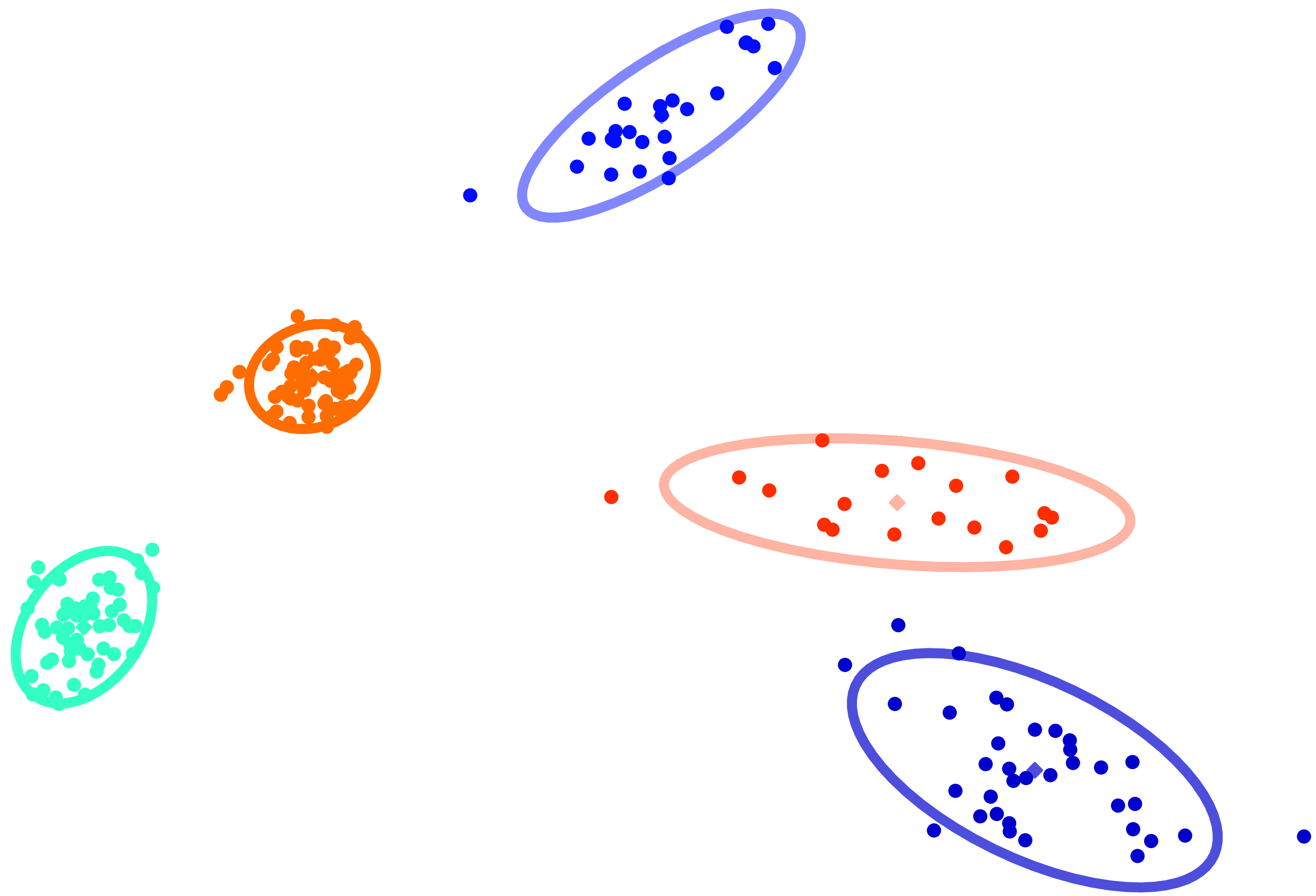
Figure 2. LSTM Autoencoder Model

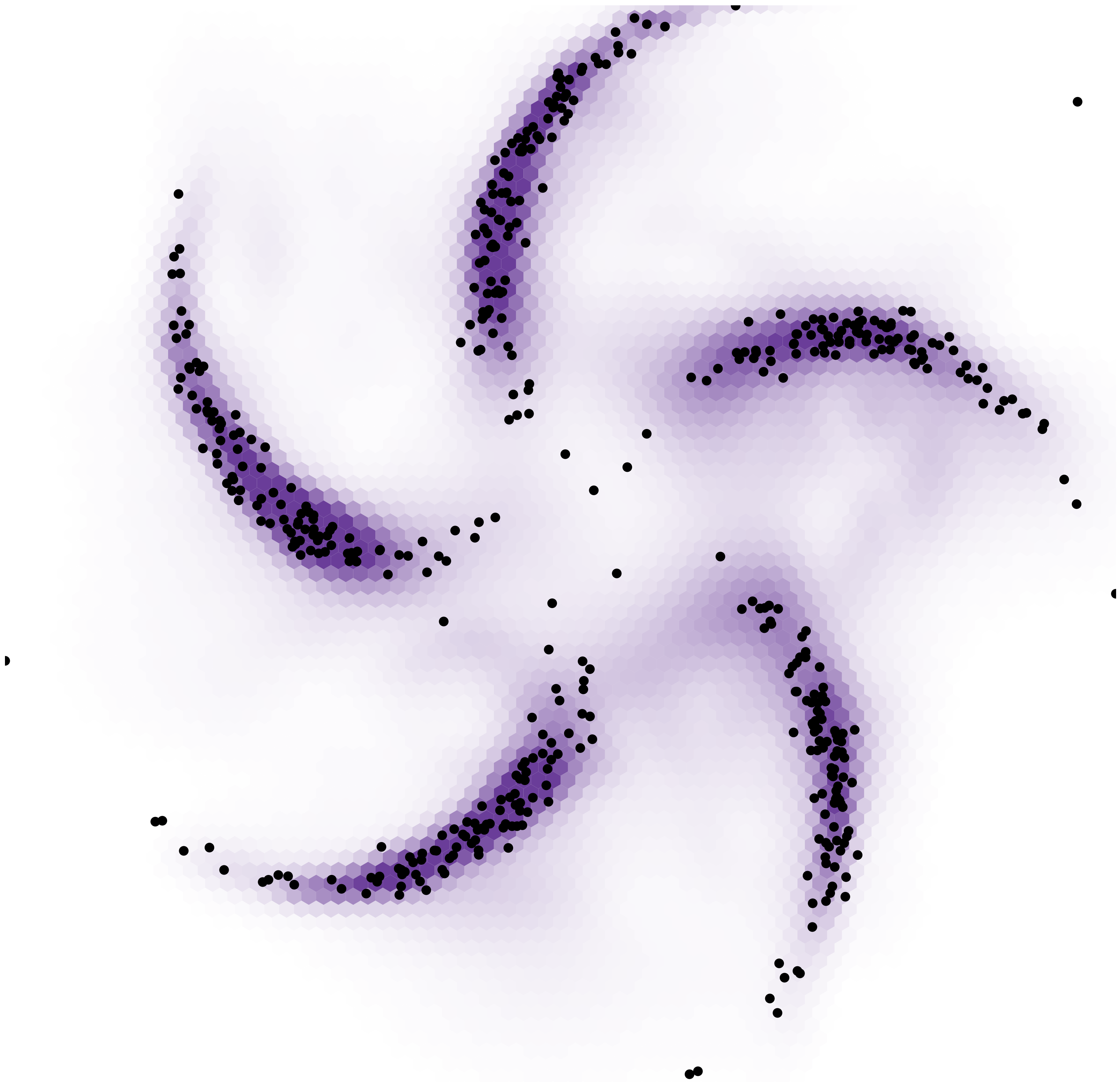
Probabilistic graphical models? [4,5,6]

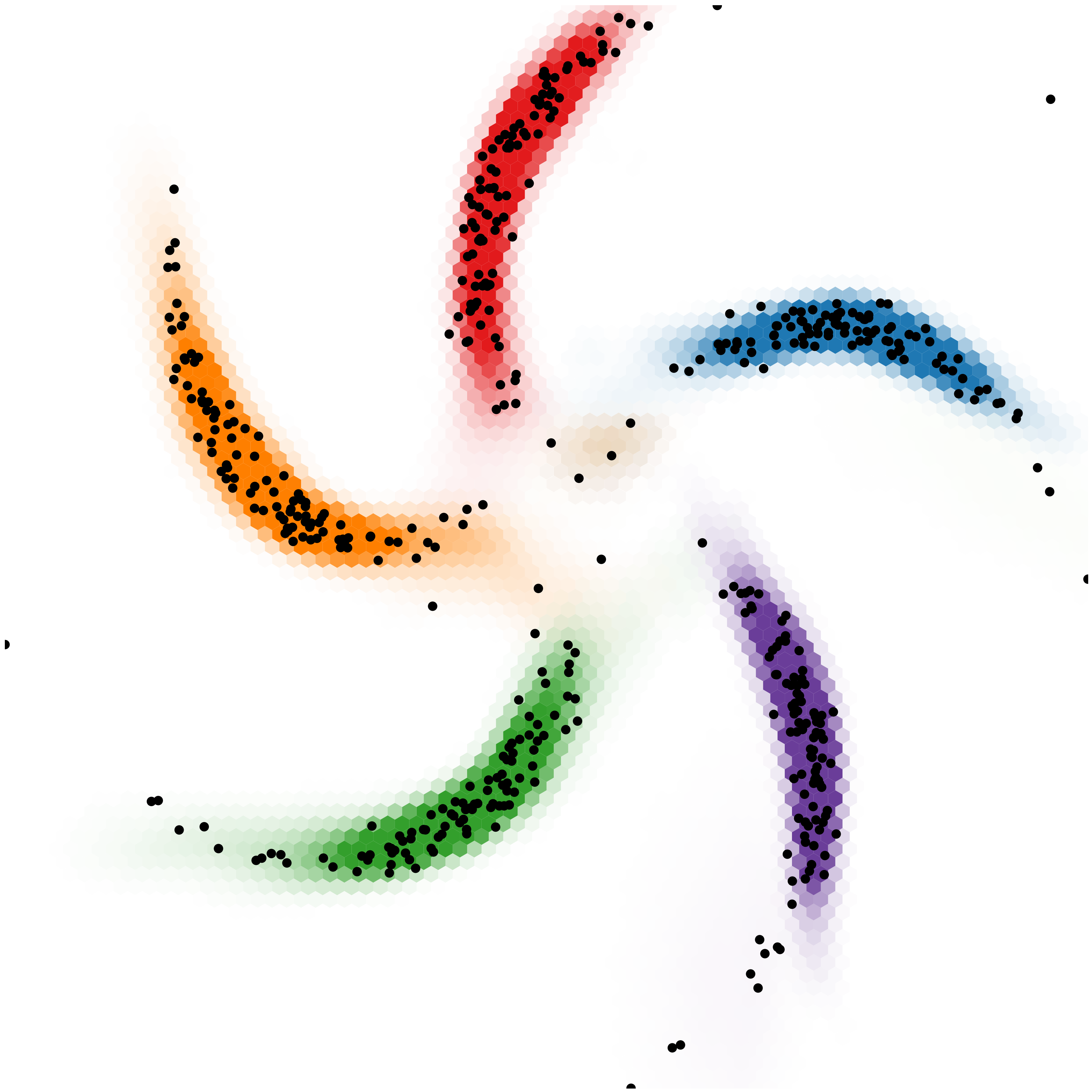


[1] Srivastava, Mansimov, Salakhutdinov. Unsupervised learning of video representations using LSTMs. ICML 2015.
[2] Ranzato, MarcAurelio, et al. Video (language) modeling: a baseline for generative models of natural videos. Preprint 2015.
[3] Sutskever, Hinton, and Taylor. The Recurrent Temporal Restricted Boltzmann Machine. NIPS 2008.
[4] Fox, Sudderth, Jordan, Willsky. Bayesian nonparametric inference of switching dynamic linear models. IEEE TSP 2011.
[5] **Johnson** and Willsky. Bayesian nonparametric hidden semi-Markov models. JMLR 2013.
[6] Murphy. Machine learning: a probabilistic perspective. MIT Press 2012.

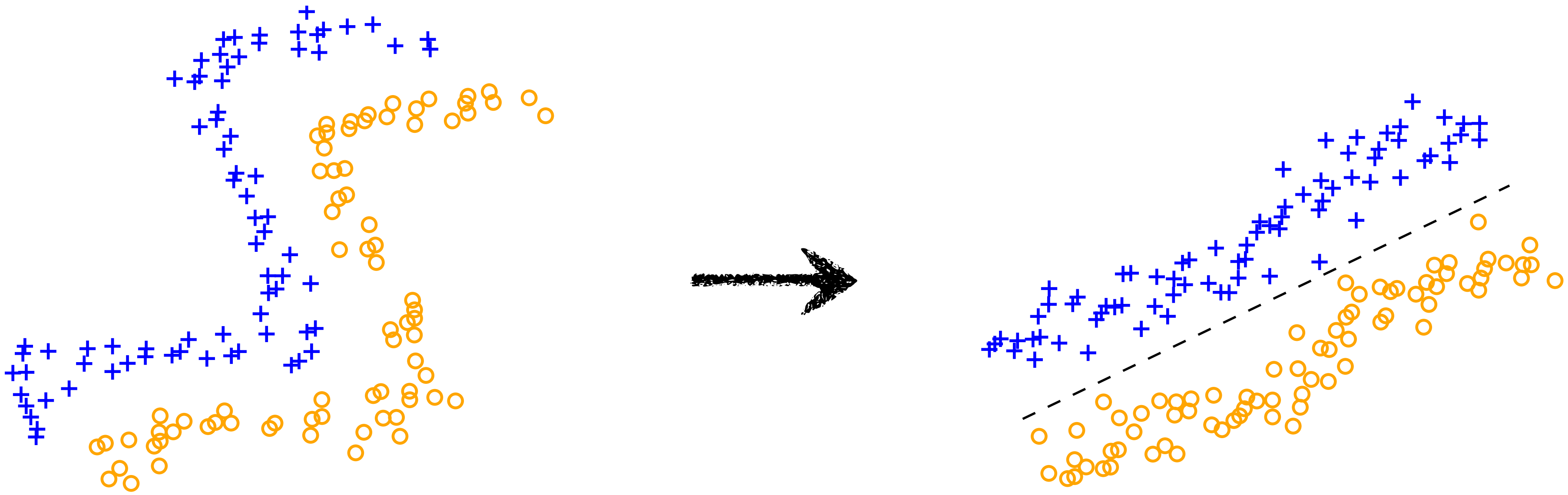




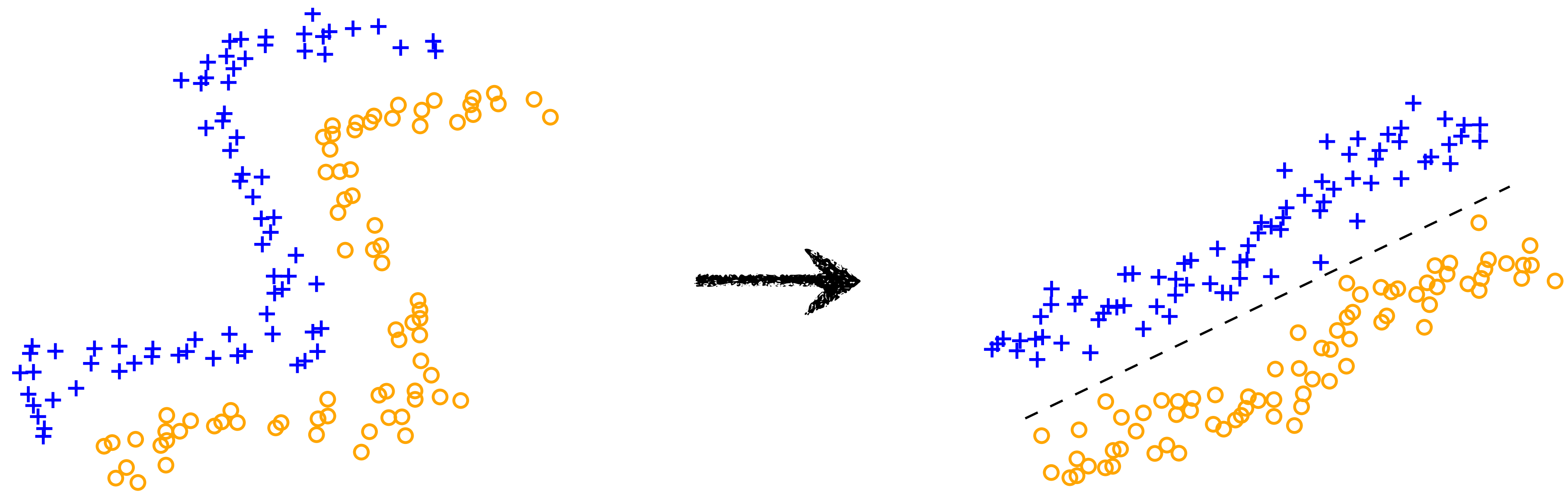




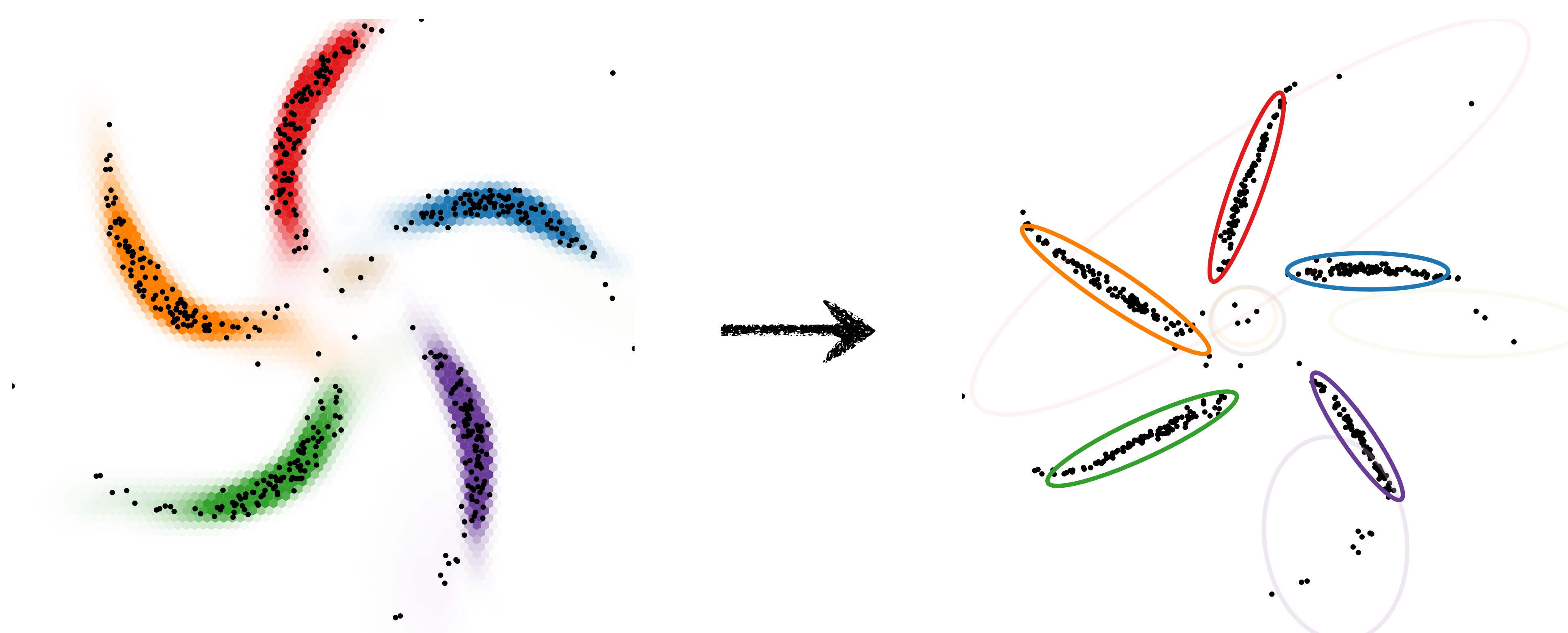
supervised
learning



supervised learning



unsupervised learning



Probabilistic graphical models

Deep learning

Probabilistic graphical models

+ structured representations
+ priors and uncertainty
– rigid assumptions may not fit
– feature engineering

Deep learning

Probabilistic graphical models | Deep learning

+ structured representations

+ priors and uncertainty

– rigid assumptions may not fit

– feature engineering

+ arbitrary inference queries

+ data and computational efficiency

– but only in rigid model classes

Probabilistic graphical models

+ structured representations
+ priors and uncertainty
– rigid assumptions may not fit
– feature engineering

+ arbitrary inference queries
+ data and computational efficiency
– but only in rigid model classes

Deep learning

– neural net “goo”
– difficult parameterization
+ flexible, high capacity
+ feature learning

Probabilistic graphical models

+ structured representations
+ priors and uncertainty
– rigid assumptions may not fit
– feature engineering

+ arbitrary inference queries
+ data and computational efficiency
– but only in rigid model classes

Deep learning

– neural net "goo"
– difficult parameterization
+ flexible, high capacity
+ feature learning

– limited inference queries
– data- and compute-hungry
+ recognition networks learn how to do inference

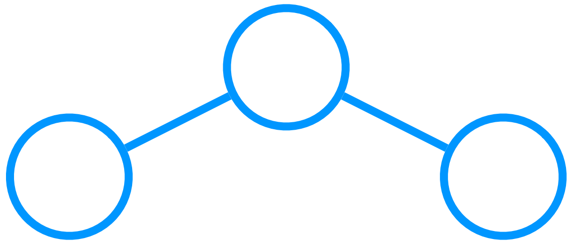
Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen
2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory
3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation
4. Make SVAEs and related PGM + DNN architectures super obvious so that you can invent better ones

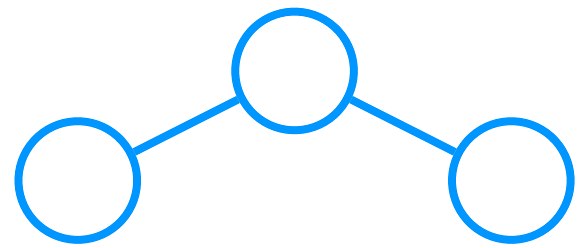
Goals

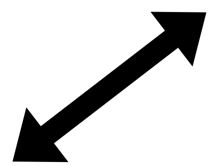
1. Motivate why PGMs + DNNs are a revolution waiting to happen
2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory
3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation
4. Make SVAEs and related PGM + DNN architectures super obvious so that you can invent better ones

Graphs



Graphs

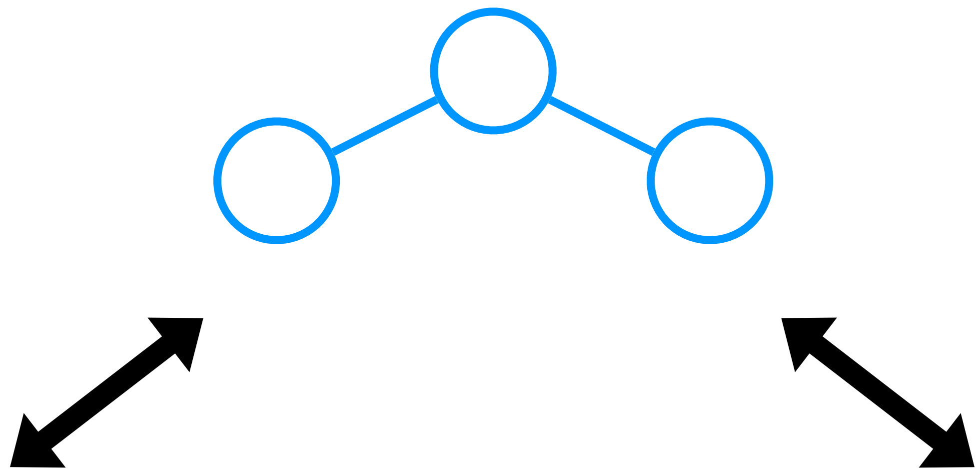




Independence of RVs

$$X_1 \perp\!\!\!\perp X_3 \mid X_2$$

Graphs



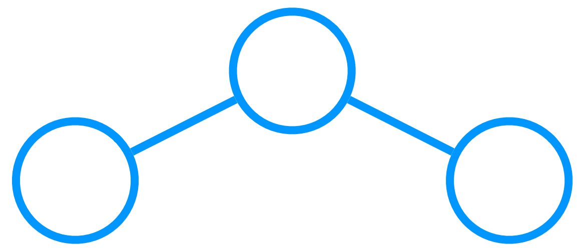
Independence of RVs

$$X_1 \perp\!\!\!\perp X_3 \mid X_2$$

Algebraic structure in density

$$p(x) = \frac{1}{Z} \prod_C \psi_C(x_C)$$

Graphs



Independence of RVs

$$X_1 \perp\!\!\!\perp X_3 \mid X_2$$

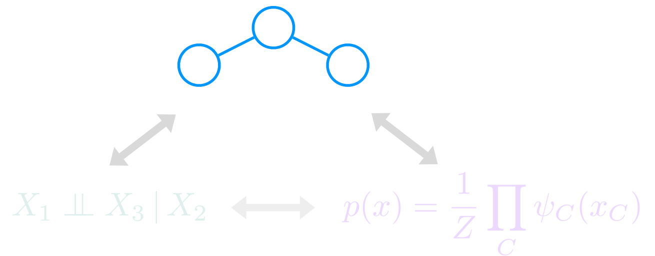
Algebraic structure in density

$$p(x) = \frac{1}{Z} \prod_C \psi_C(x_C)$$

$$G = (V, E)$$

$$V = \{1, 2, \ldots, n\}$$

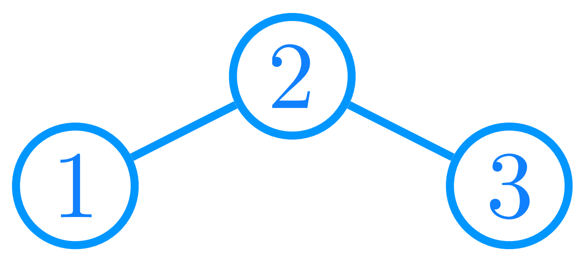
$$E \subseteq \{ \, \{u, v\} : u, v \in V \, \}$$



$$G = (V, E)$$

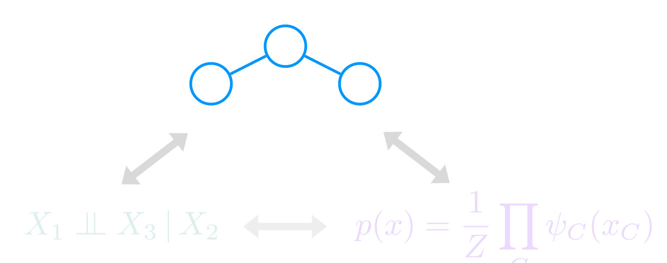
$$V = \{1, 2, \ldots, n\}$$

$$E \subseteq \{\, \{u, v\} : u, v \in V \,\}$$

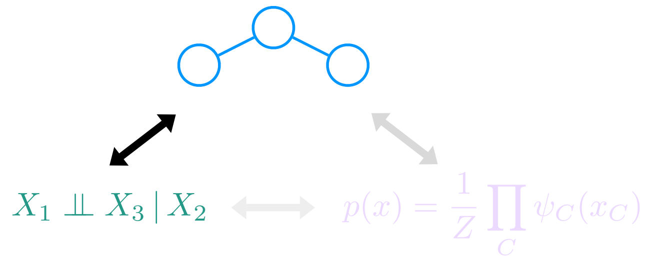


$$V = \{1, 2, 3\}$$

$$E = \{\, \{1, 2\}, \{2, 3\} \,\}$$

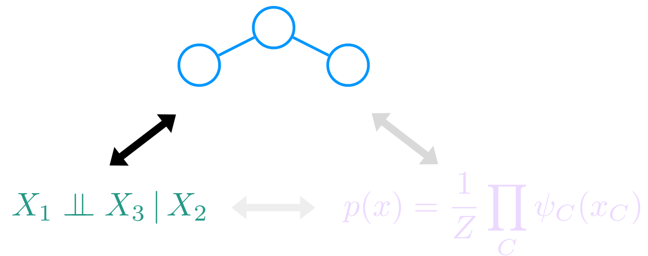


Def $X = \{X_v\}_{v \in V}$ is *Markov on* G iff for disjoint $A, B, C \subseteq V$ when C disconnects A from B we have $X_A \perp\!\!\!\perp X_B \,|\, X_C$.



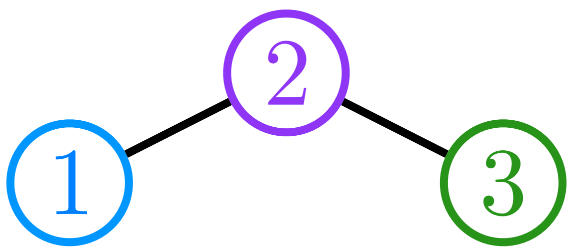
Def $X = \{X_v\}_{v \in V}$ is *Markov on G* iff for disjoint $A, B, C \subseteq V$ when C disconnects A from B we have $X_A \perp\!\!\!\perp X_B \mid X_C$.

<u>disconnects</u> means no path from A to B after removing C.

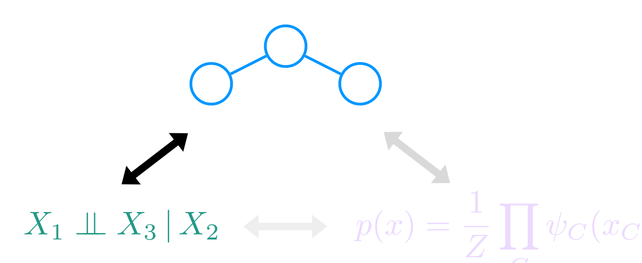


Def $X = \{X_v\}_{v \in V}$ is *Markov on G* iff for disjoint $A, B, C \subseteq V$ when C disconnects A from B we have $X_A \perp\!\!\!\perp X_B \,|\, X_C$.

disconnects means no path from A to B after removing C.

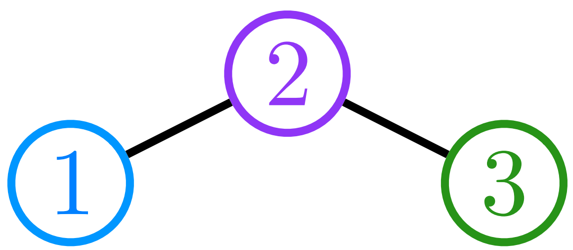


$$\Longrightarrow \qquad X_1 \perp\!\!\!\perp X_3 \,|\, X_2$$

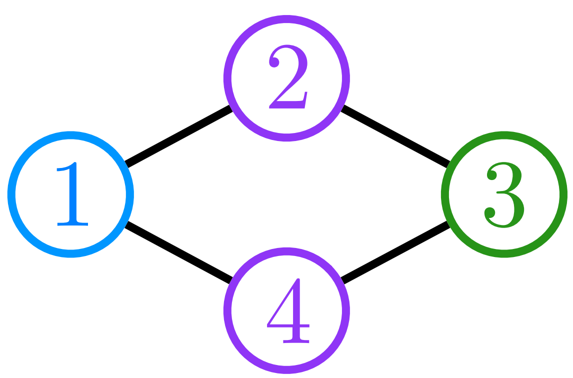


Def $X = \{X_v\}_{v \in V}$ is *Markov on G* iff for disjoint $A, B, C \subseteq V$ when C disconnects A from B we have $X_A \perp\!\!\!\perp X_B \,|\, X_C$.

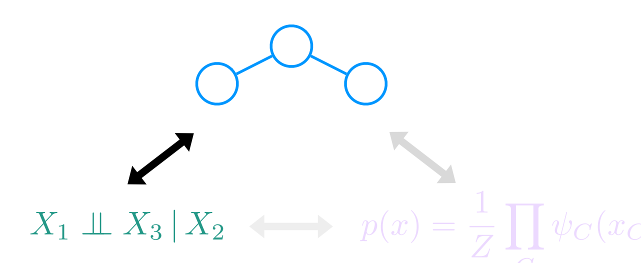
<u>disconnects</u> means no path from A to B after removing C.



$$\implies \qquad X_1 \perp\!\!\!\perp X_3 \,|\, X_2$$

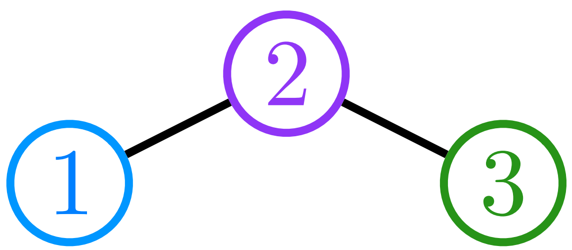


$$\implies \qquad X_1 \perp\!\!\!\perp X_3 \,|\, X_2, X_4$$

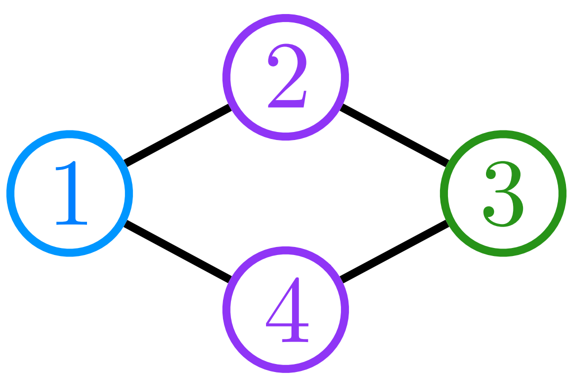


Def $X = \{X_v\}_{v \in V}$ is *Markov on* G iff for disjoint $A, B, C \subseteq V$ when C disconnects A from B we have $X_A \perp\!\!\!\perp X_B \mid X_C$.

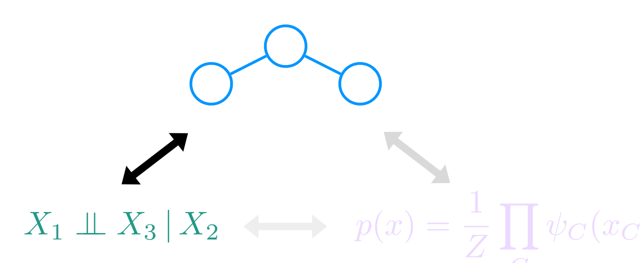
<u>disconnects</u> means no path from A to B after removing C.



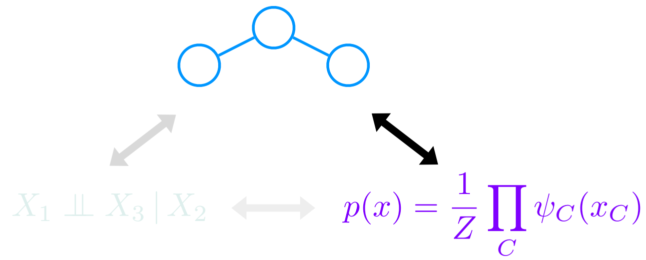
$$\Longrightarrow \qquad X_1 \perp\!\!\!\perp X_3 \mid X_2$$



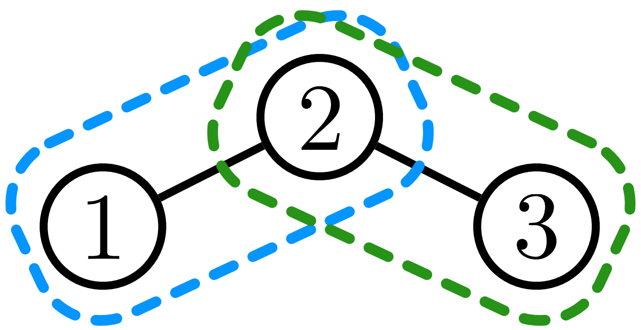
$$\not\Longrightarrow \qquad X_1 \perp\!\!\!\perp X_3 \mid X_2$$



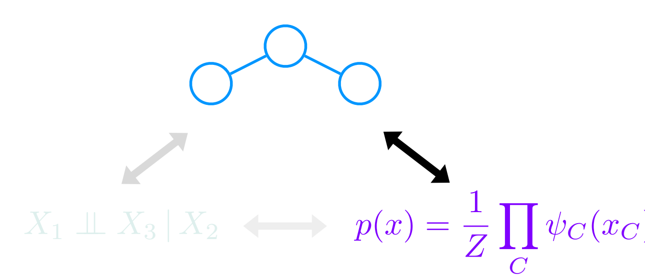
Def $X = \{X_v\}_{v \in V}$ *factorizes on* G iff $p(x) \propto \prod_C \psi_C(x_C)$ where the product is over cliques of G.



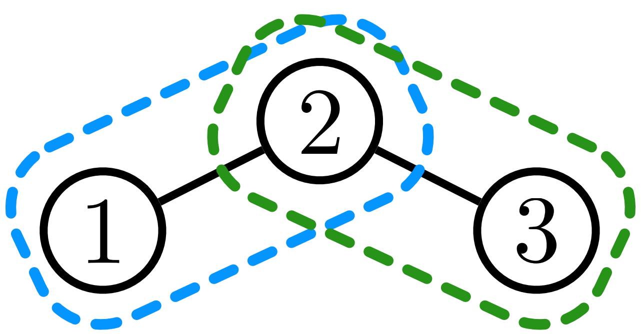
Def $X = \{X_v\}_{v \in V}$ *factorizes on* G iff $p(x) \propto \prod_C \psi_C(x_C)$ where the product is over cliques of G.



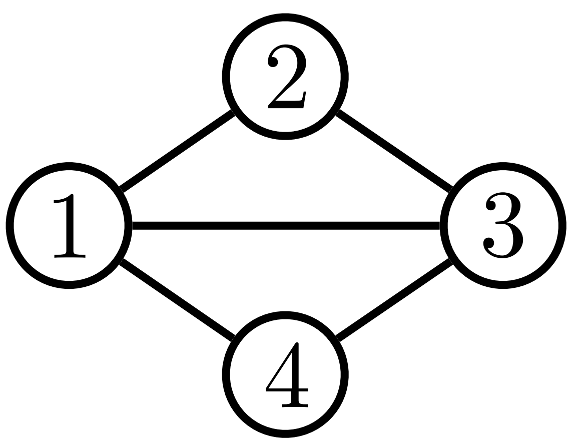
$$\Longrightarrow \quad p(x) \propto \psi_{12}(x_1, x_2)\psi_{23}(x_2, x_3)$$

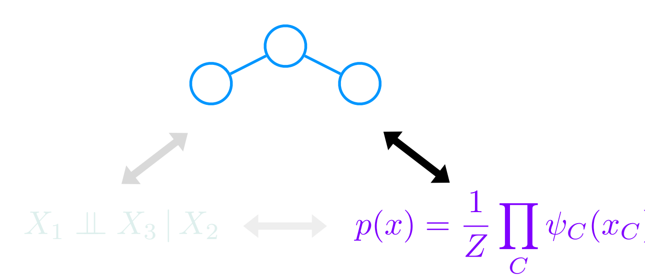


Def $X = \{X_v\}_{v \in V}$ *factorizes on* G iff $p(x) \propto \prod_C \psi_C(x_C)$ where the product is over cliques of G.

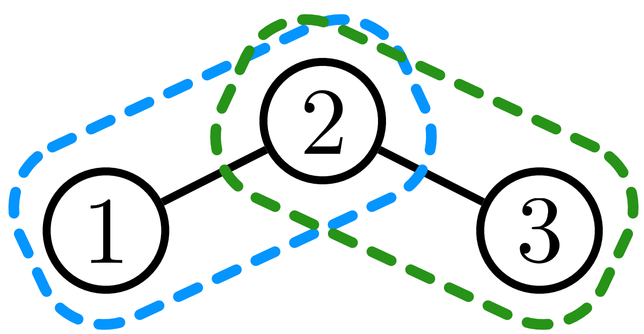


$$\Longrightarrow \quad p(x) \propto \psi_{12}(x_1, x_2)\psi_{23}(x_2, x_3)$$

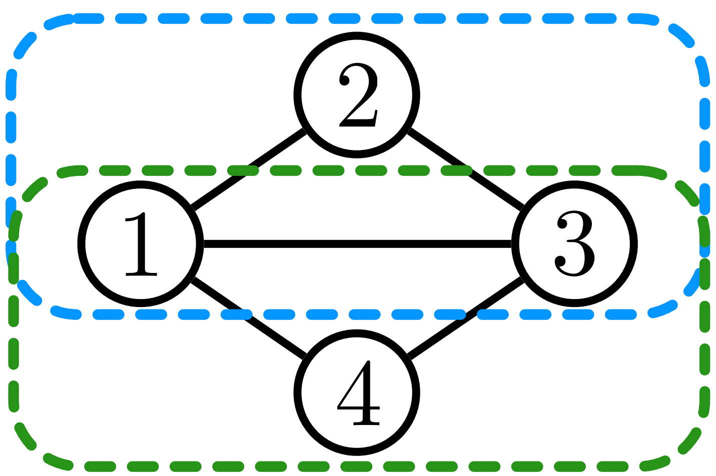




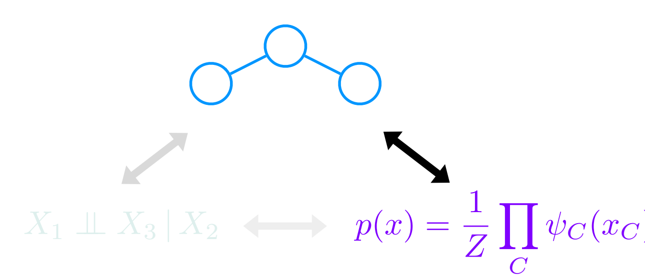
Def $X = \{X_v\}_{v \in V}$ *factorizes on* G iff $p(x) \propto \prod_C \psi_C(x_C)$ where the product is over cliques of G.



$$\Longrightarrow \quad p(x) \propto \psi_{12}(x_1, x_2)\psi_{23}(x_2, x_3)$$

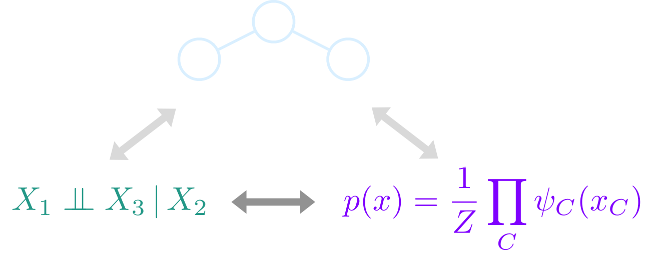


$$\Longrightarrow \quad p(x) \propto \psi_{123}(x_1, x_2, x_3)\psi_{134}(x_1, x_3, x_4)$$



Def $X = \{X_v\}_{v \in V}$ is *Markov on* G iff for disjoint $A, B, C \subseteq V$ when C disconnects A from B we have $X_A \perp\!\!\!\perp X_B \,|\, X_C$.

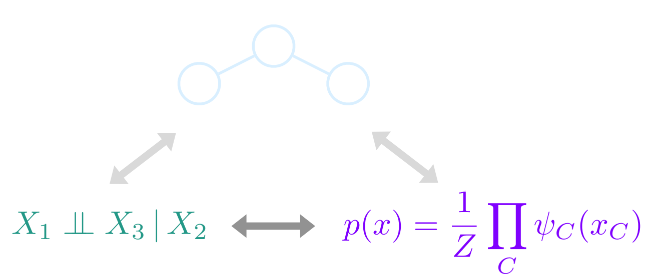
Def $X = \{X_v\}_{v \in V}$ *factorizes on* G iff $p(x) \propto \prod_C \psi_C(x_C)$ where the product is over cliques of G.



Def $X = \{X_v\}_{v \in V}$ is *Markov on G* iff for disjoint $A, B, C \subseteq V$ when C disconnects A from B we have $X_A \perp\!\!\!\perp X_B \,|\, X_C$.

Def $X = \{X_v\}_{v \in V}$ *factorizes on G* iff $p(x) \propto \prod_C \psi_C(x_C)$ where the product is over cliques of G.

$$\text{Markov on } G \quad \not\Longleftrightarrow \quad \text{factorizes on } G$$

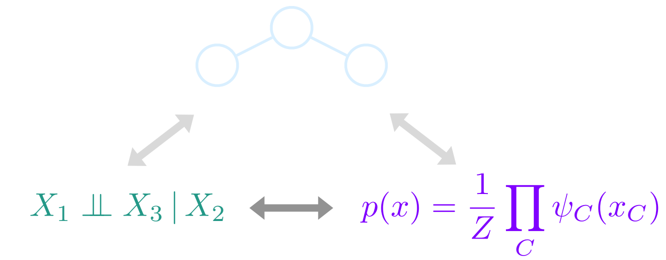


Def $X = \{X_v\}_{v \in V}$ is *Markov on* G iff for disjoint $A, B, C \subseteq V$ when C disconnects A from B we have $X_A \perp\!\!\!\perp X_B \mid X_C$.

Def $X = \{X_v\}_{v \in V}$ *factorizes on* G iff $p(x) \propto \prod_C \psi_C(x_C)$ where the product is over cliques of G.

Markov on G $\quad \not\Longleftrightarrow \quad$ factorizes on G

... but they are the same if $p(x) > 0$.



A good way to ensure $p(x) > 0$ is to have $p(x) = \exp(-E(x))$.

$$p(x) = \frac{1}{Z} \prod_C \psi_C(x_C)$$

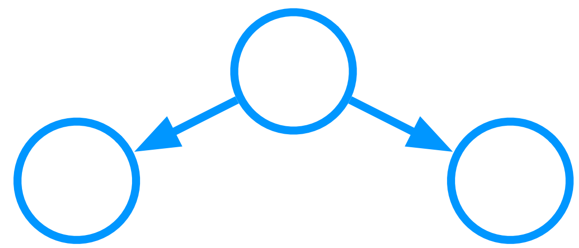
$$E(x) = \log Z + \sum_C \phi_C(x_C)$$

A good way to ensure $p(x) > 0$ is to have $p(x) = \exp(-E(x))$.

$$p(x) = \frac{1}{Z} \prod_{C} \psi_C(x_C)$$

$$E(x; \eta) = \log Z + \sum_{C} \eta_C \cdot \phi_C(x_C)$$

Graphs



Independence of RVs

$$X_1 \perp\!\!\!\perp X_3 \mid X_2$$

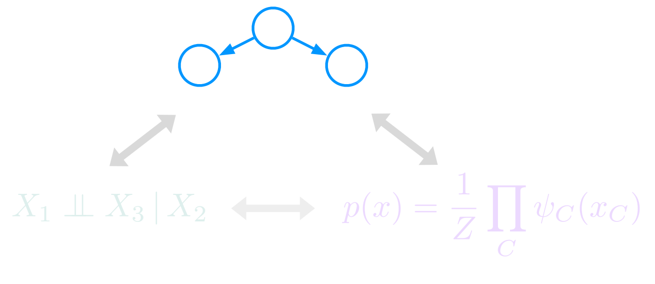
Algebraic structure in density

$$p(x) = \prod_{v \in V} p(x_v \mid x_{\mathrm{pa}(v)})$$

$$G = (V, E)$$

$$V = \{1, 2, \ldots, n\}$$

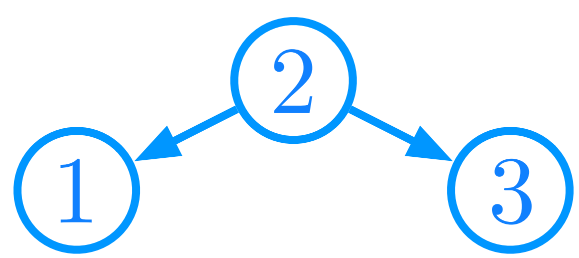
$$E \subseteq V \times V$$



$$G = (V, E)$$

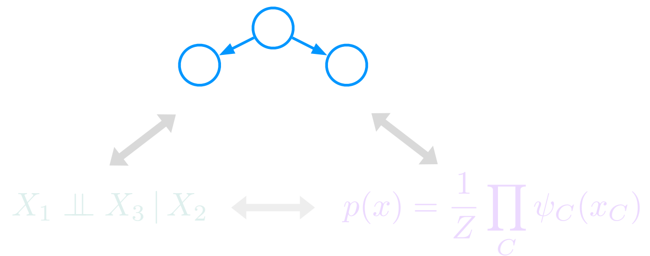
$$V = \{1, 2, \ldots, n\}$$

$$E \subseteq V \times V$$



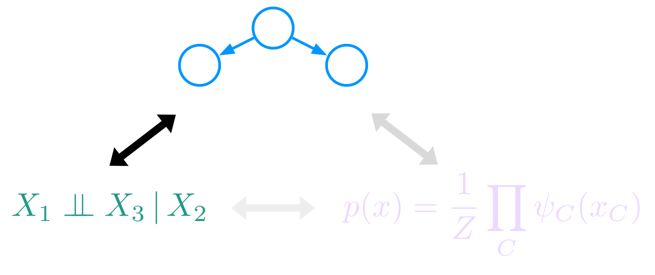
$$V = \{1, 2, 3\}$$

$$E = \{\ (2, 1),\ (2, 3)\ \}$$



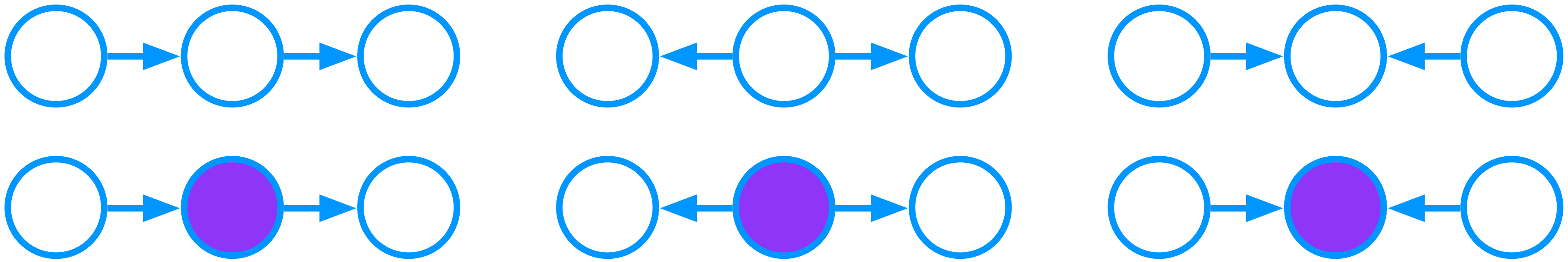
Def $X = \{X_v\}_{v \in V}$ is *Markov on G* iff for disjoint $A, B, C \subseteq V$ when C d-separates A from B we have $X_A \perp\!\!\!\perp X_B \,|\, X_C$.

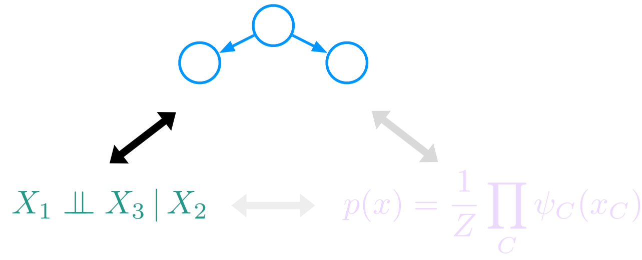
d-separates means no unblocked undirected path from A to B.



Def $X = \{X_v\}_{v \in V}$ is *Markov on G* iff for disjoint $A, B, C \subseteq V$ when C d-separates A from B we have $X_A \perp\!\!\!\perp X_B \,|\, X_C$.

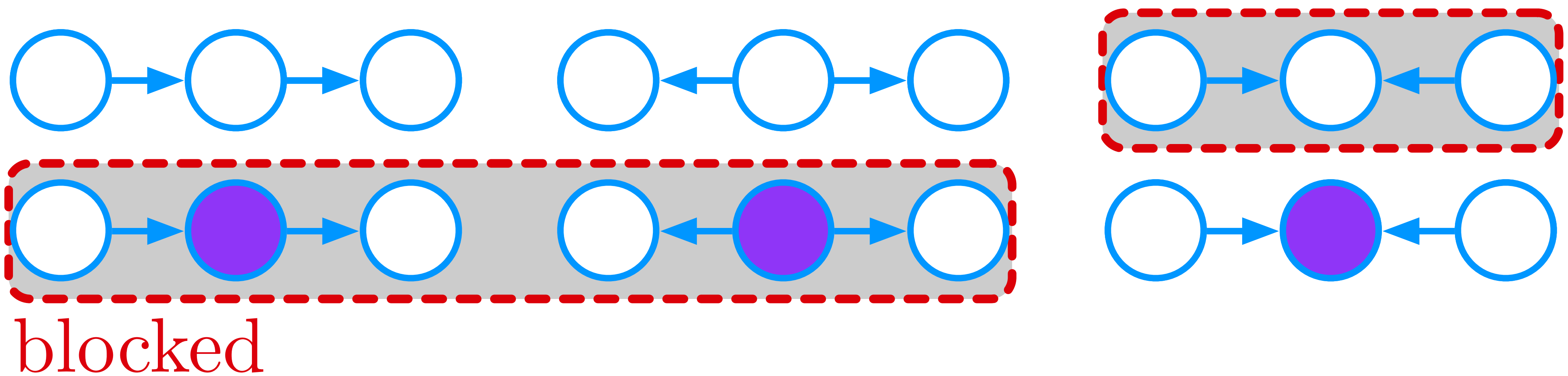
d-separates means no unblocked undirected path from A to B.

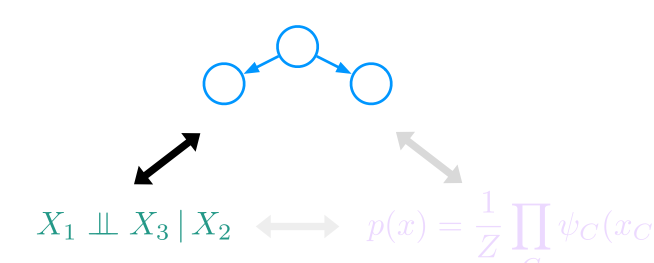




Def $X = \{X_v\}_{v \in V}$ is *Markov on G* iff for disjoint $A, B, C \subseteq V$ when C d-separates A from B we have $X_A \perp\!\!\!\perp X_B \,|\, X_C$.

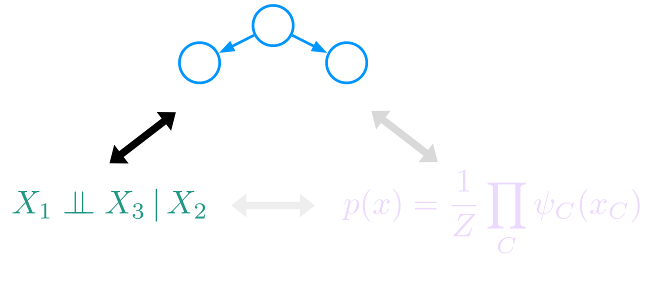
d-separates means no unblocked undirected path from A to B.





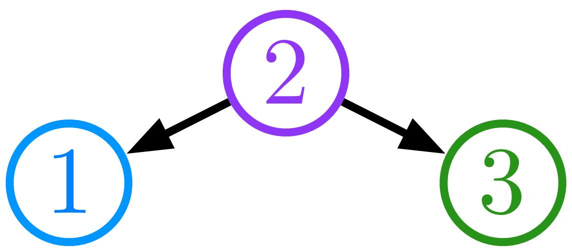
Def $X = \{X_v\}_{v \in V}$ is *Markov on* G iff for disjoint $A, B, C \subseteq V$ when C d-separates A from B we have $X_A \perp\!\!\!\perp X_B \,|\, X_C$.

d-separates means no unblocked undirected path from A to B.

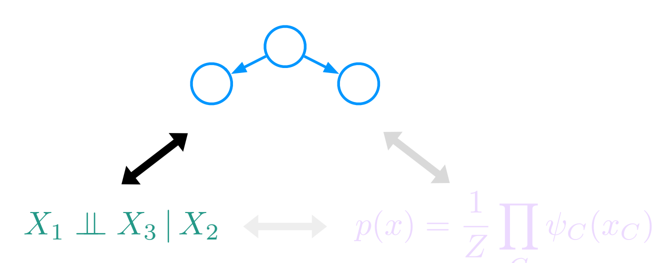


Def $X = \{X_v\}_{v \in V}$ is *Markov on G* iff for disjoint $A, B, C \subseteq V$ when C d-separates A from B we have $X_A \perp\!\!\!\perp X_B \,|\, X_C$.

d-separates means no unblocked undirected path from A to B.

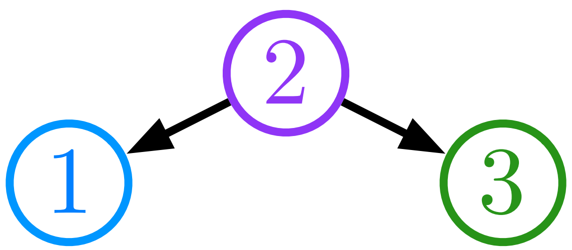


$$\Longrightarrow \qquad X_1 \perp\!\!\!\perp X_3 \,|\, X_2$$

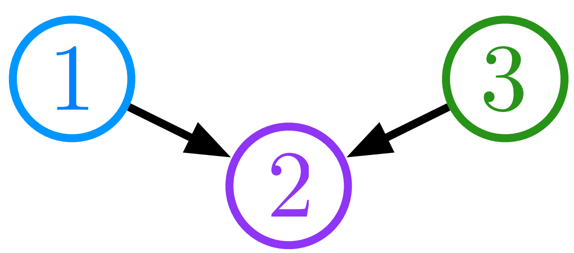


Def $X = \{X_v\}_{v \in V}$ is *Markov on G* iff for disjoint $A, B, C \subseteq V$ when C d-separates A from B we have $X_A \perp\!\!\!\perp X_B \,|\, X_C$.

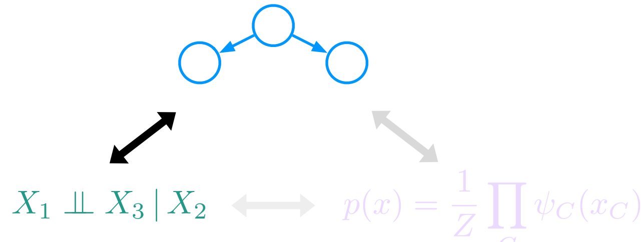
d-separates means no unblocked undirected path from A to B.



$$\Longrightarrow \qquad X_1 \perp\!\!\!\perp X_3 \,|\, X_2$$

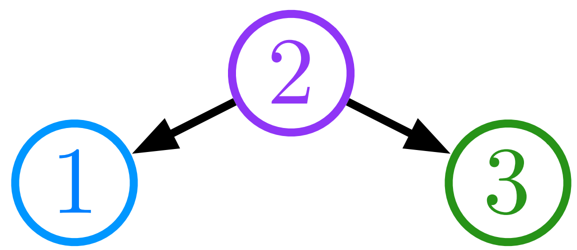


$$\Longrightarrow \qquad X_1 \perp\!\!\!\perp X_3$$

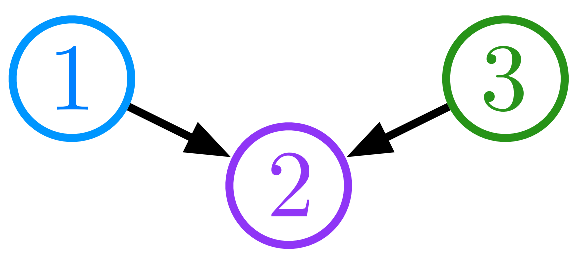


Def $X = \{X_v\}_{v \in V}$ is *Markov on G* iff for disjoint $A, B, C \subseteq V$ when C d-separates A from B we have $X_A \perp\!\!\!\perp X_B \,|\, X_C$.

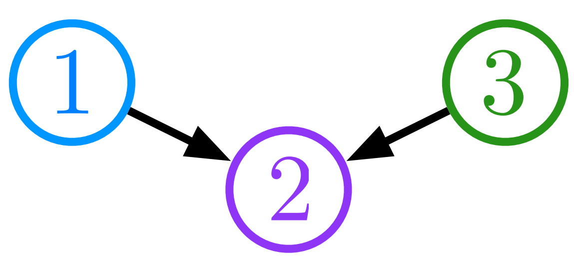
d-separates means no unblocked undirected path from A to B.



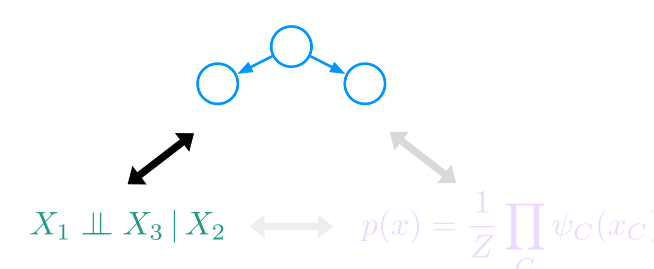
$$\implies \quad X_1 \perp\!\!\!\perp X_3 \,|\, X_2$$



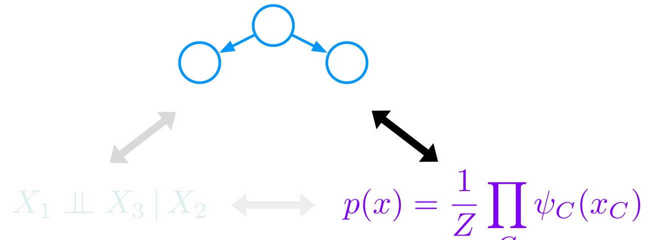
$$\implies \quad X_1 \perp\!\!\!\perp X_3$$



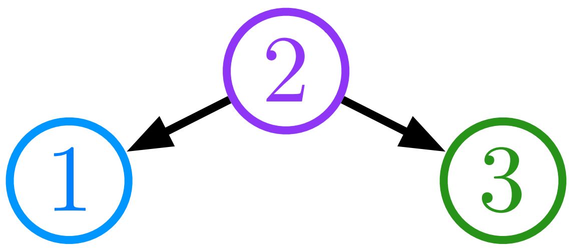
$$\not\Longrightarrow \quad X_1 \perp\!\!\!\perp X_3 \,|\, X_2$$



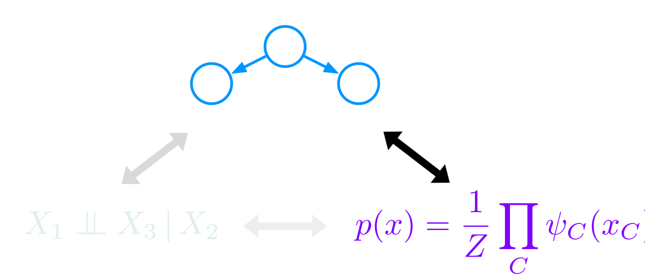
Def $X = \{X_v\}_{v \in V}$ *factorizes on* G iff $p(x) = \prod_{v \in V} p(x_v \mid x_{\mathrm{pa}(v)})$.



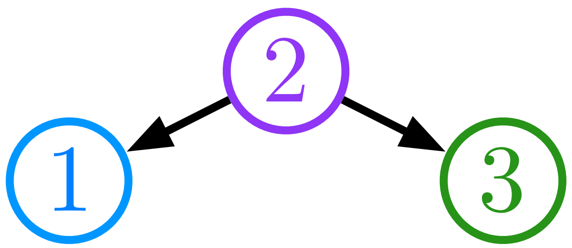
Def $X = \{X_v\}_{v \in V}$ *factorizes on* G iff $p(x) = \prod_{v \in V} p(x_v \mid x_{\mathrm{pa}(v)})$.



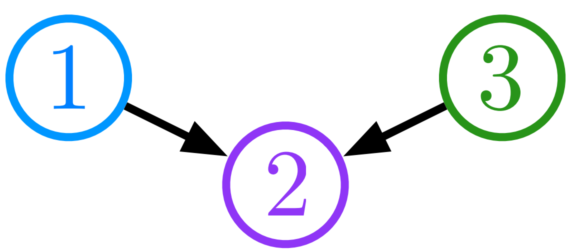
$$\Longrightarrow \quad p(x) = p(x_2)p(x_1 \mid x_2)p(x_3 \mid x_2)$$



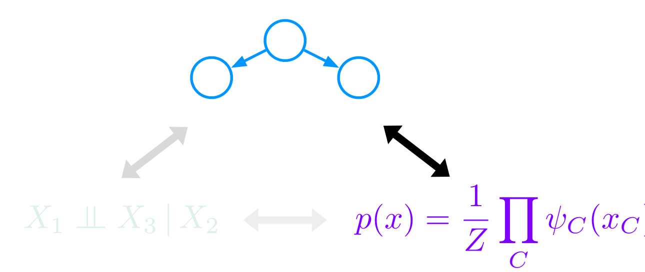
Def $X = \{X_v\}_{v \in V}$ *factorizes on* G iff $p(x) = \prod_{v \in V} p(x_v \,|\, x_{\mathrm{pa}(v)})$.

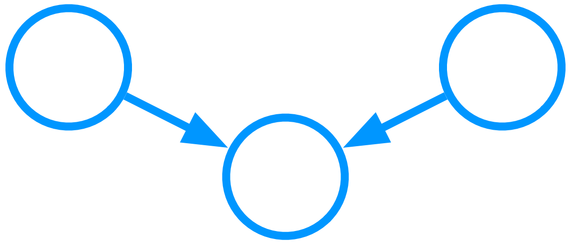


$$\implies \quad p(x) = p(x_2)p(x_1 \,|\, x_2)p(x_3 \,|\, x_2)$$



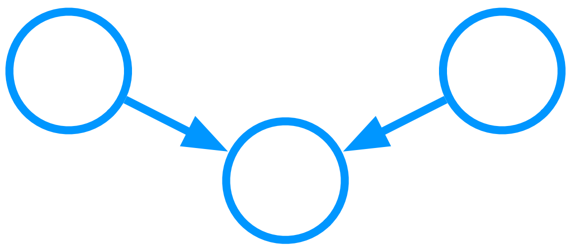
$$\implies \quad p(x) = p(x_1)p(x_3)p(x_2 \,|\, x_1, x_3)$$





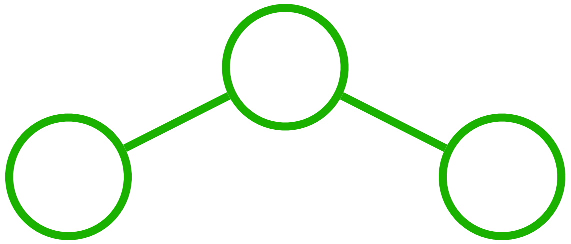
Directed models often appear with generative models

- Ancestral sampling
- For Gaussians, like having a Cholesky factorization



Directed models often appear with generative models

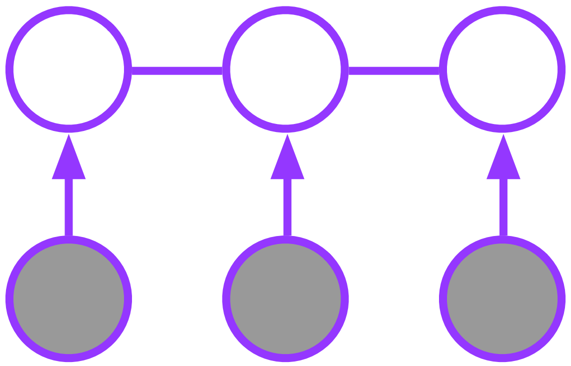
- Ancestral sampling
- For Gaussians, like having a Cholesky factorization



Undirected models often appear in inference

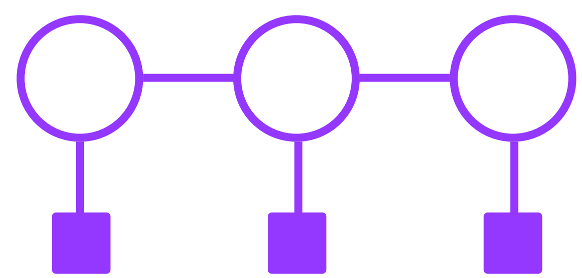
- For Gaussians, like solving the linear system

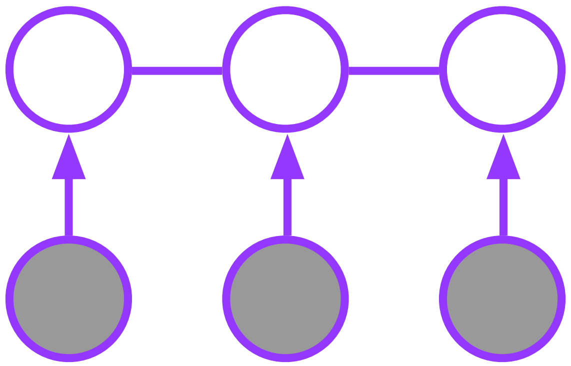
$$J\mu = h$$



Conditional random fields (CRFs) are PGMs where potentials depend on exogenous data

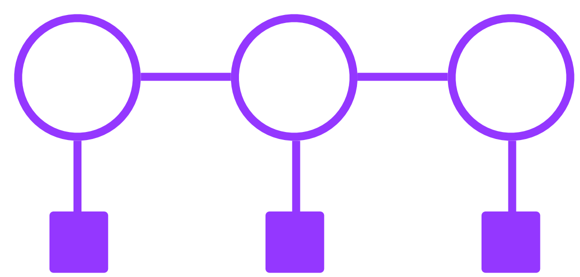
$$p(y\,;\,x) \propto \psi_1(y_1\,;\,x_1)\psi_{12}(y_1, y_2)\psi_2(y_2\,;\,x_2)\psi_{23}(y_2, y_3)\psi_3(y_3\,;\,x_3)$$



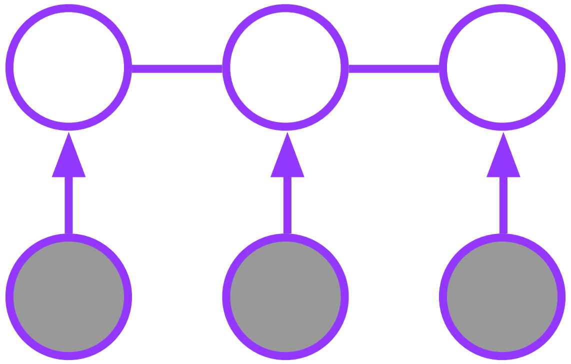


Conditional random fields (CRFs) are PGMs where potentials depend on exogenous data

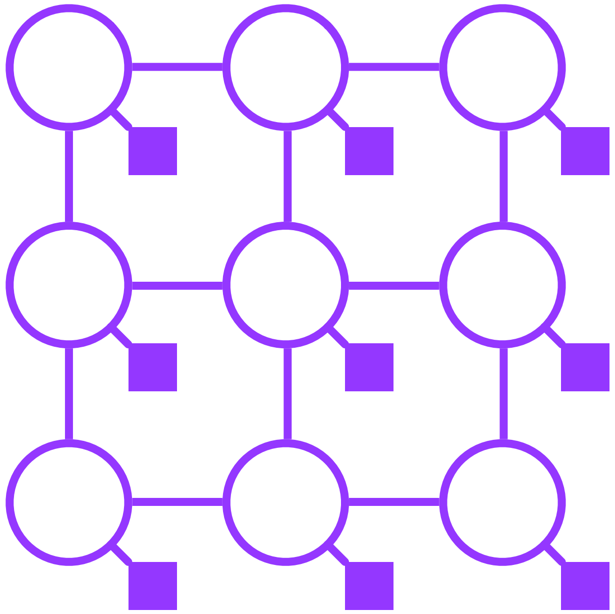
$$p(y\,;\,x) \propto \psi_1(y_1\,;\,x_1)\psi_{12}(y_1,y_2)\psi_2(y_2\,;\,x_2)\psi_{23}(y_2,y_3)\psi_3(y_3\,;\,x_3)$$



$$\psi_n(y_n\,;\,x_n) = \psi(y_n\,;\,f(x_n,\phi)) \;\; \text{for neural network} \;\; f(\,\cdot\,,\phi)$$



Conditional random fields (CRFs) are PGMs where potentials depend on exogenous data



Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen
2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory
3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation
4. Make SVAEs and related PGM + DNN architectures super obvious so that you can invent better ones

Def An *exponential family of densities* is

Def An *exponential family of densities* is

$$t(x)$$

defined by statistic function $t : \mathcal{X} \to W$

Def An *exponential family of densities* is

$$t(x)$$

defined by statistic function $t : \mathcal{X} \to \boxed{W}$ finite-dim real vector space

Def An *exponential family of densities* is

$$\exp(\langle \eta, t(x) \rangle \qquad\qquad)$$

defined by statistic function $t : \mathcal{X} \to W$ finite-dim real vector space

indexed by natural parameter $\eta \in \Theta \subseteq W$

Def An *exponential family of densities* is

$$p(x\,;\,\eta) = \exp(\langle \eta,\, t(x)\rangle - \mathcal{A}(\eta))$$

defined by statistic function $t : \mathcal{X} \to W$ finite-dim real vector space

indexed by natural parameter $\eta \in \Theta \subseteq W$

defines a log normalizer $\mathcal{A} : \Theta \to \mathbb{R}$

$$\mathcal{A}(\eta) \triangleq \log \int_{\mathcal{X}} \exp(\langle \eta,\, t(x)\rangle)\,\mathrm{d}x$$

Def An *exponential family of densities* is

$$p(x\,;\,\eta) = \exp(\langle \eta,\, t(x)\rangle - \mathcal{A}(\eta))$$

defined by statistic function $t : \mathcal{X} \to W$ finite-dim real vector space

indexed by natural parameter $\eta \in \Theta \subseteq W$

defines a log normalizer $\mathcal{A} : \Theta \to \mathbb{R}$

$$\mathcal{A}(\eta) \triangleq \log \int_{\mathcal{X}} \exp(\langle \eta,\, t(x)\rangle)\,\mathrm{d}x$$

$$\Theta \triangleq \{\eta \in W : \mathcal{A}(\eta) < \infty\}$$

Def An *exponential family of densities* is

$$p(x\,;\,\eta) = \exp(\langle \eta,\, t(x)\rangle - \mathcal{A}(\eta))$$

$$\mathcal{A}(\eta) \triangleq \log \int_{\mathcal{X}} \exp(\langle \eta,\, t(x)\rangle)\,\mathrm{d}x$$

Def An *exponential family of densities* is

$$p(x\,;\,\eta) = \exp(\langle \eta,\, t(x)\rangle - \mathcal{A}(\eta))$$

$$\mathcal{A}(\eta) \triangleq \log \int_{\mathcal{X}} \exp(\langle \eta,\, t(x)\rangle)\, \mathrm{d}x$$

Claim $\mathcal{A}$ is convex and $\nabla \mathcal{A} : \Theta \to \mathcal{M} \subseteq W$ is

$$\nabla \mathcal{A}(\eta) = \mathbb{E}[t(X)]$$

Def An *exponential family of densities* is

$$p(x\,;\,\eta) = \exp(\langle \eta,\, t(x)\rangle - \mathcal{A}(\eta))$$

$$\mathcal{A}(\eta) \triangleq \log \int_{\mathcal{X}} \exp(\langle \eta,\, t(x)\rangle)\,\mathrm{d}x$$

Claim $\mathcal{A}$ is convex and $\nabla\mathcal{A} : \Theta \to \mathcal{M} \subseteq W$ is

$$\nabla\mathcal{A}(\eta) = \mathbb{E}[t(X)]$$

Proof

$$\nabla\mathcal{A}(\eta) = \frac{1}{\int_{\mathcal{X}} \exp(\langle \eta,\, t(x)\rangle)\,\mathrm{d}x} \int_{\mathcal{X}} t(x) \exp(\langle \eta,\, t(x)\rangle)\,\mathrm{d}x$$

$$= \int_{\mathcal{X}} t(x) p(x\,;\,\eta)\,\mathrm{d}x$$

Def An *exponential family of densities* is

$$p(x\,;\,\eta) = \exp(\langle \eta,\, t(x)\rangle - \mathcal{A}(\eta))$$

$$\mathcal{A}(\eta) \triangleq \log \int_{\mathcal{X}} \exp(\langle \eta,\, t(x)\rangle)\,\mathrm{d}x$$

Claim $\mathcal{A}$ is convex and $\nabla \mathcal{A} : \Theta \to \mathcal{M} \subseteq W$ is

$$\nabla \mathcal{A}(\eta) = \mathbb{E}[t(X)]$$

$$\nabla^2 \mathcal{A}(\eta) = \mathbb{E}[t(X)t(X)^\mathsf{T}]$$

$$\vdots$$

Def An *exponential family of densities* is

$$p(x\,;\,\eta) = \exp(\langle \eta,\, t(x)\rangle - \mathcal{A}(\eta))$$

$$\mathcal{A}(\eta) \triangleq \log \int_{\mathcal{X}} \exp(\langle \eta,\, t(x)\rangle)\,\mathrm{d}x$$

Claim $\mathcal{A}$ is convex and $\nabla \mathcal{A} : \Theta \to \mathcal{M} \subseteq W$ is

$$\nabla \mathcal{A}(\eta) = \mathbb{E}[t(X)]$$

$$\mathcal{M} \triangleq \{\mu \in W : \exists p\,.\,\mathbb{E}_p[t(X)] = \mu\}$$

Def An *exponential family of densities* is

$$p(x\,;\,\eta) = \exp(\langle \eta,\, t(x)\rangle - \mathcal{A}(\eta))$$

$$\mathcal{A}(\eta) \triangleq \log \int_{\mathcal{X}} \exp(\langle \eta,\, t(x)\rangle)\,\mathrm{d}x$$

Claim $\mathcal{A}$ is convex and $\nabla\mathcal{A} : \Theta \to \mathcal{M} \subseteq W$ is

$$\nabla\mathcal{A}(\eta) = \mathbb{E}[t(X)]$$

Claim The KL divergence between two members is

$$\mathrm{KL}(\eta_1, \eta_2) = \langle \eta_1 - \eta_2,\, \nabla\mathcal{A}(\eta_1)\rangle - (\mathcal{A}(\eta_1) - \mathcal{A}(\eta_2))$$

Def An *exponential family of densities* is

$$p(x\,;\,\eta) = \exp(\langle \eta,\, t(x)\rangle - \mathcal{A}(\eta))$$

$$\mathcal{A}(\eta) \triangleq \log \int_{\mathcal{X}} \exp(\langle \eta,\, t(x)\rangle)\,\mathrm{d}x$$

Claim $\mathcal{A}$ is convex and $\nabla\mathcal{A} : \Theta \to \mathcal{M} \subseteq W$ is

$$\nabla\mathcal{A}(\eta) = \mathbb{E}[t(X)]$$

Claim The KL divergence between two members is

$$\mathrm{KL}(\eta_1, \eta_2) = \langle \eta_1 - \eta_2,\, \nabla\mathcal{A}(\eta_1)\rangle - (\mathcal{A}(\eta_1) - \mathcal{A}(\eta_2))$$

Def Say family is *tractable* if $\mathcal{A}$ is easy to evaluate.

Example: binary hidden Markov model (HMM)

$$x = (x_1, x_2, \ldots, x_N), \qquad x_n \in \{0, 1\}$$

Example: binary hidden Markov model (HMM)

$$x = (x_1, x_2, \ldots, x_N), \qquad x_n \in \{0, 1\}$$

$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots)$$

Example: binary hidden Markov model (HMM)

$$x = (x_1, x_2, \ldots, x_N), \qquad x_n \in \{0, 1\}$$

$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots)$$

$$\psi_n(x_n) = \exp(\eta_n x_n), \quad \psi_{n,n+1}(x_n, x_{n+1}) = \exp(\eta_{n,n+1} x_n x_{n+1})$$

Example: binary hidden Markov model (HMM)

$$x = (x_1, x_2, \ldots, x_N), \qquad x_n \in \{0, 1\}$$

$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots)$$

$$\psi_n(x_n) = \exp(\eta_n x_n), \quad \psi_{n,n+1}(x_n, x_{n+1}) = \exp(\eta_{n,n+1} x_n x_{n+1})$$

$$\phi_n(x_n) = \eta_n x_n, \qquad \phi_{n,n+1}(x_n, x_{n+1}) = \eta_{n,n+1} x_n x_{n+1}$$

Example: binary hidden Markov model (HMM)

$$x = (x_1, x_2, \ldots, x_N), \qquad x_n \in \{0, 1\}$$

$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots)$$

$$\psi_n(x_n) = \exp(\eta_n x_n), \quad \psi_{n,n+1}(x_n, x_{n+1}) = \exp(\eta_{n,n+1} x_n x_{n+1})$$
$$\phi_n(x_n) = \eta_n x_n, \qquad \phi_{n,n+1}(x_n, x_{n+1}) = \eta_{n,n+1} x_n x_{n+1}$$

$$t(x) = (\phi_1(x_1), \ldots, \phi_N(x_N),$$
$$\phi_{1,2}(x_1, x_2), \ldots, \phi_{N-1,N}(x_{N-1}, x_N))$$

Example: binary hidden Markov model (HMM)

$$x = (x_1, x_2, \dots, x_N), \qquad x_n \in \{0, 1\}$$

$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \dots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \dots)$$

$$\psi_n(x_n) = \exp(\eta_n x_n), \quad \psi_{n,n+1}(x_n, x_{n+1}) = \exp(\eta_{n,n+1} x_n x_{n+1})$$
$$\phi_n(x_n) = \eta_n x_n, \qquad \phi_{n,n+1}(x_n, x_{n+1}) = \eta_{n,n+1} x_n x_{n+1}$$

$$t(x) = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \begin{bmatrix} x_1 & x_2 & \cdots & x_N \end{bmatrix}$$

Example: binary hidden Markov model (HMM)

$$x = (x_1, x_2, \ldots, x_N), \qquad x_n \in \{0, 1\}$$

$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots)$$

$$\psi_n(x_n) = \exp(\eta_n x_n), \quad \psi_{n,n+1}(x_n, x_{n+1}) = \exp(\eta_{n,n+1} x_n x_{n+1})$$

$$\phi_n(x_n) = \eta_n x_n, \qquad \phi_{n,n+1}(x_n, x_{n+1}) = \eta_{n,n+1} x_n x_{n+1}$$

$$t(x) = x x^{\mathsf{T}}$$

Example: binary hidden Markov model (HMM)

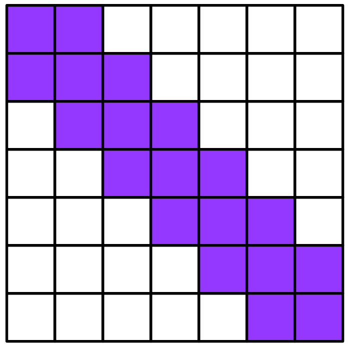
$$x = (x_1, x_2, \ldots, x_N), \qquad x_n \in \{0, 1\}$$

$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots)$$

$$\psi_n(x_n) = \exp(\eta_n x_n), \quad \psi_{n,n+1}(x_n, x_{n+1}) = \exp(\eta_{n,n+1} x_n x_{n+1})$$

$$\phi_n(x_n) = \eta_n x_n, \qquad \phi_{n,n+1}(x_n, x_{n+1}) = \eta_{n,n+1} x_n x_{n+1}$$

$$t(x) = x x^{\mathsf{T}} \qquad \eta =$$



Example: binary hidden Markov model (HMM)

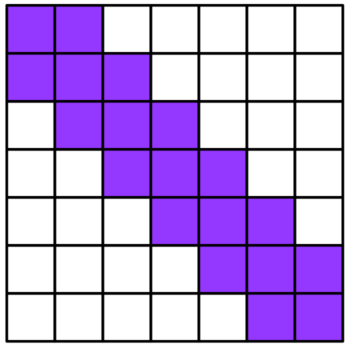
$$x = (x_1, x_2, \ldots, x_N), \qquad x_n \in \{0, 1\}$$

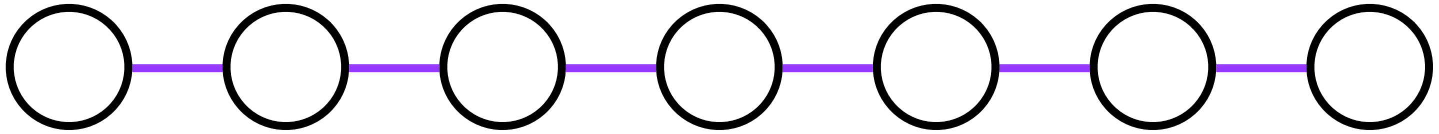
$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots)$$

$$\psi_n(x_n) = \exp(\eta_n x_n), \quad \psi_{n,n+1}(x_n, x_{n+1}) = \exp(\eta_{n,n+1} x_n x_{n+1})$$

$$\phi_n(x_n) = \eta_n x_n, \qquad \phi_{n,n+1}(x_n, x_{n+1}) = \eta_{n,n+1} x_n x_{n+1}$$

$$t(x) = xx^{\mathsf{T}} \qquad \eta =$$





Example: binary hidden Markov model (HMM)

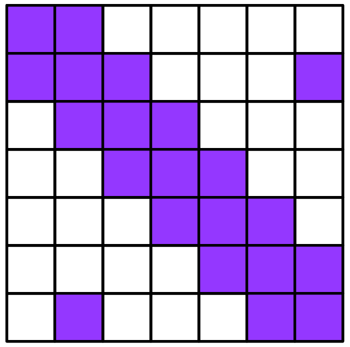
$$x = (x_1, x_2, \ldots, x_N), \qquad x_n \in \{0, 1\}$$

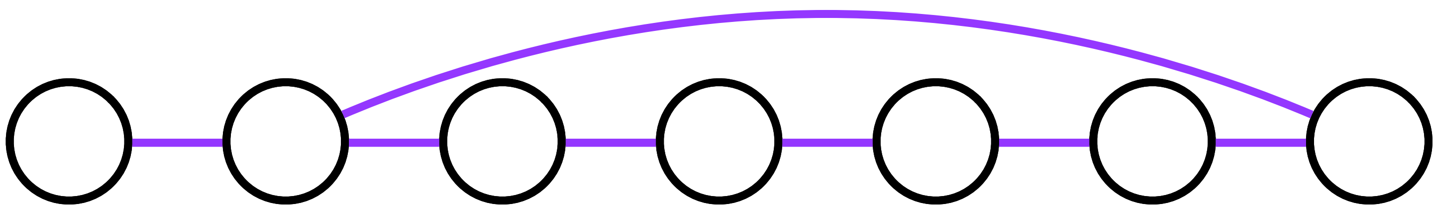
$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots)$$

$$\psi_n(x_n) = \exp(\eta_n x_n), \quad \psi_{n,n+1}(x_n, x_{n+1}) = \exp(\eta_{n,n+1} x_n x_{n+1})$$

$$\phi_n(x_n) = \eta_n x_n, \qquad \phi_{n,n+1}(x_n, x_{n+1}) = \eta_{n,n+1} x_n x_{n+1}$$

$$t(x) = xx^{\mathsf{T}} \qquad \eta =$$



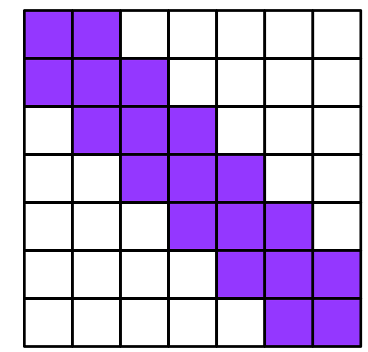


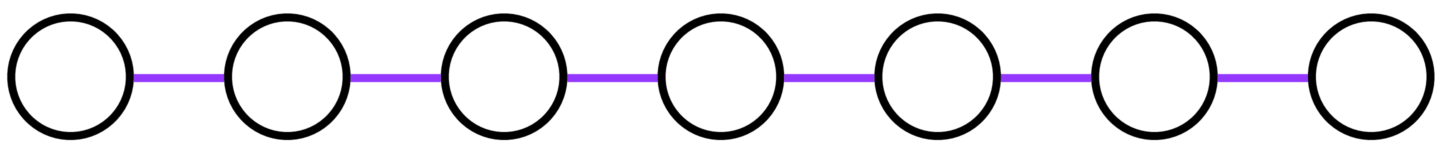
Example: binary hidden Markov model (HMM)

$$x = (x_1, x_2, \ldots, x_N), \qquad x_n \in \{0, 1\}$$

$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots)$$

$$t(x) = x x^{\mathsf{T}}$$

$\eta =$ 

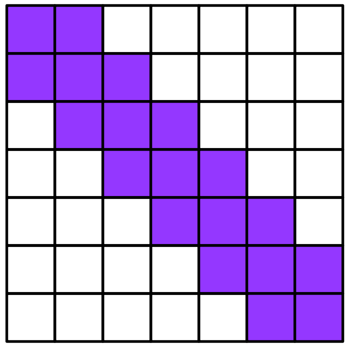


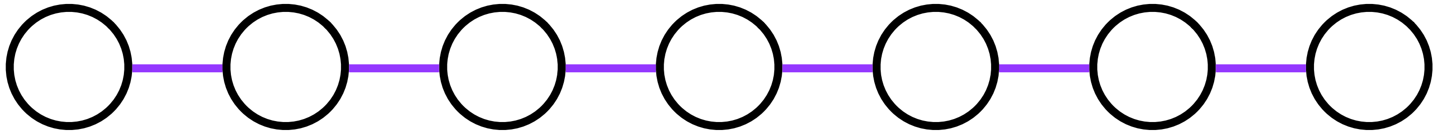
Example: binary hidden Markov model (HMM)

$$x = (x_1, x_2, \ldots, x_N), \qquad x_n \in \{0, 1\}$$

$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots)$$

$$t(x) = x x^{\mathsf{T}}$$

$\eta =$ 



$$\mathcal{A}(\eta) = \log \sum_{x \in \{0,1\}^N} \exp(\langle \eta, t(x) \rangle)$$

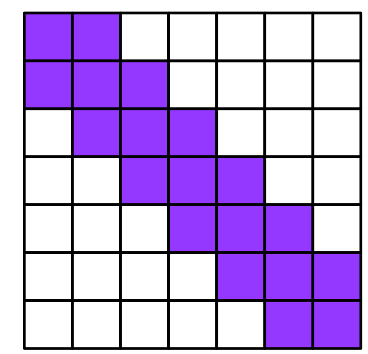
Example: binary hidden Markov model (HMM)

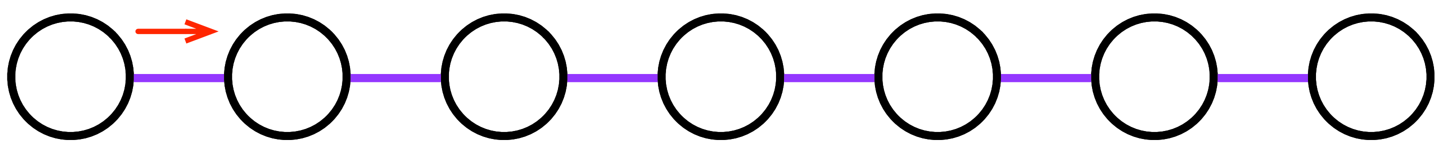
$$x = (x_1, x_2, \dots, x_N), \qquad x_n \in \{0, 1\}$$

$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \dots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \dots)$$

$$t(x) = x x^\mathsf{T}$$

$\eta =$





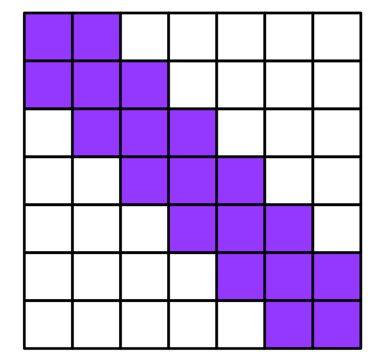
$$\mathcal{A}(\eta) = \log \sum_{x \in \{0,1\}^N} \exp(\langle \eta, t(x) \rangle)$$

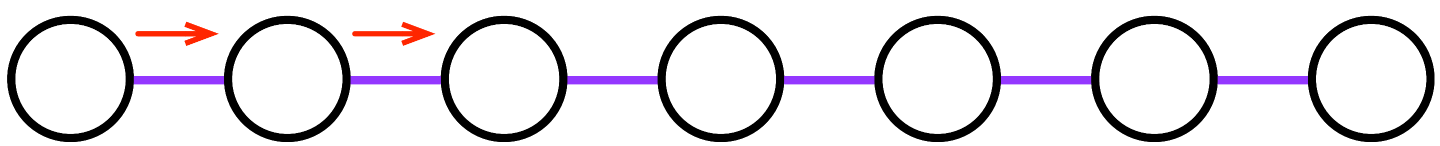
Example: binary hidden Markov model (HMM)

$$x = (x_1, x_2, \ldots, x_N), \qquad x_n \in \{0, 1\}$$

$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots)$$

$$t(x) = x x^{\mathsf{T}} \qquad \eta =$$





$$\mathcal{A}(\eta) = \log \sum_{x \in \{0,1\}^N} \exp(\langle \eta, t(x) \rangle)$$

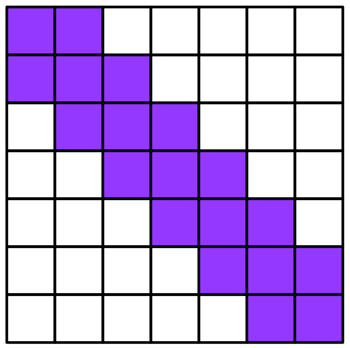
Example: binary hidden Markov model (HMM)

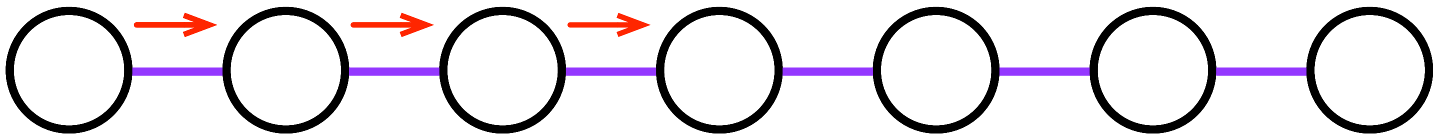
$$x = (x_1, x_2, \ldots, x_N), \qquad x_n \in \{0, 1\}$$

$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots)$$

$$t(x) = x x^\mathsf{T}$$

$$\eta =$$





$$\mathcal{A}(\eta) = \log \sum_{x \in \{0,1\}^N} \exp(\langle \eta, t(x) \rangle)$$

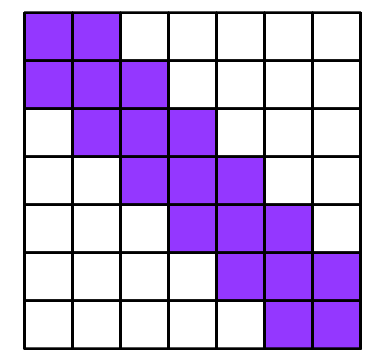
Example: binary hidden Markov model (HMM)

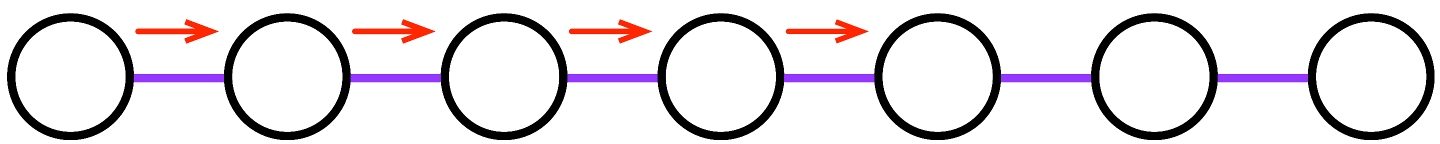
$$x = (x_1, x_2, \dots, x_N), \qquad x_n \in \{0, 1\}$$

$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \dots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \dots)$$

$$t(x) = x x^{\mathsf{T}}$$

$$\eta =$$





$$\mathcal{A}(\eta) = \log \sum_{x \in \{0,1\}^N} \exp(\langle \eta, t(x) \rangle)$$

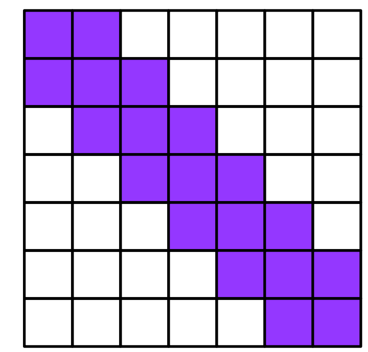
Example: binary hidden Markov model (HMM)

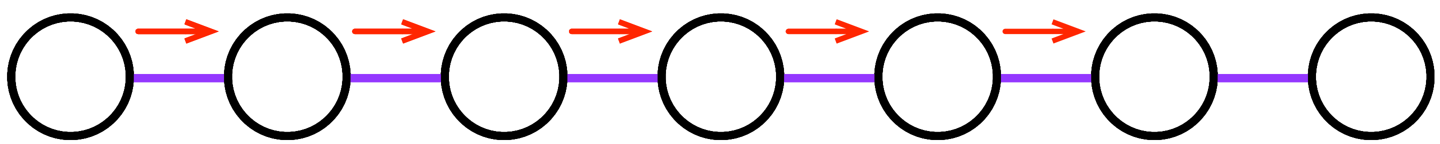
$$x = (x_1, x_2, \ldots, x_N), \qquad x_n \in \{0, 1\}$$

$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots)$$

$$t(x) = x x^\mathsf{T}$$

$$\eta =$$





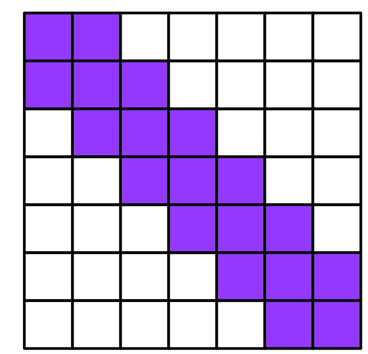
$$\mathcal{A}(\eta) = \log \sum_{x \in \{0,1\}^N} \exp(\langle \eta, t(x) \rangle)$$

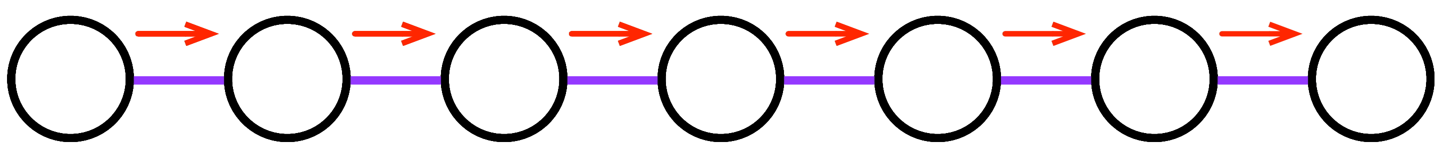
Example: binary hidden Markov model (HMM)

$$x = (x_1, x_2, \ldots, x_N), \qquad x_n \in \{0, 1\}$$

$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots)$$

$$t(x) = x x^{\mathsf{T}}$$

$\eta =$ 



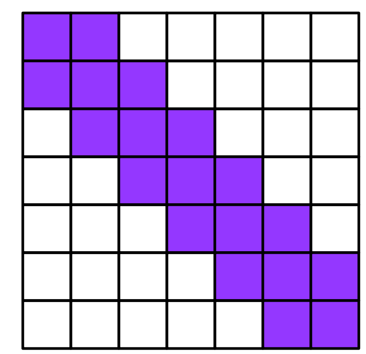
$$\mathcal{A}(\eta) = \log \sum_{x \in \{0,1\}^N} \exp(\langle \eta, t(x) \rangle)$$

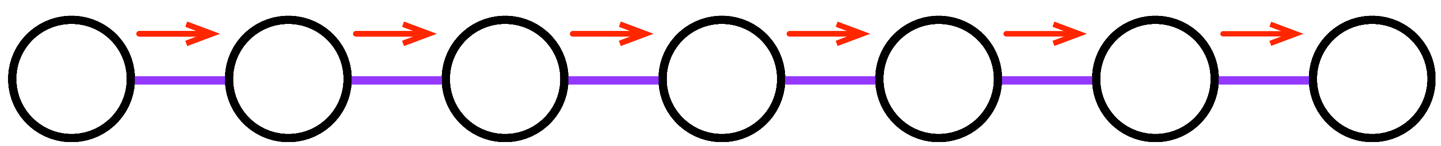
Example: binary hidden Markov model (HMM)

$$x = (x_1, x_2, \ldots, x_N), \qquad x_n \in \{0, 1\}$$

$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots)$$

$$t(x) = x x^{\mathsf{T}} \qquad \qquad \eta =$$





$$\mathcal{A}(\eta) = \log \sum_{x \in \{0,1\}^N} \exp(\langle \eta, t(x) \rangle)$$

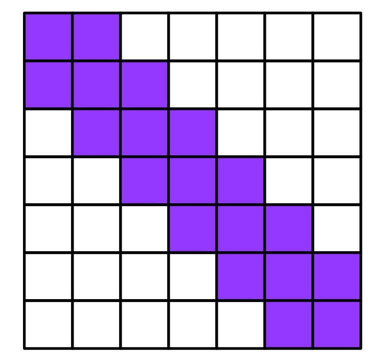
$$\nabla \mathcal{A}(\eta) = (\mathbb{E}[x_1], \ldots, \mathbb{E}[x_N], \mathbb{E}[x_1 x_2], \ldots)$$

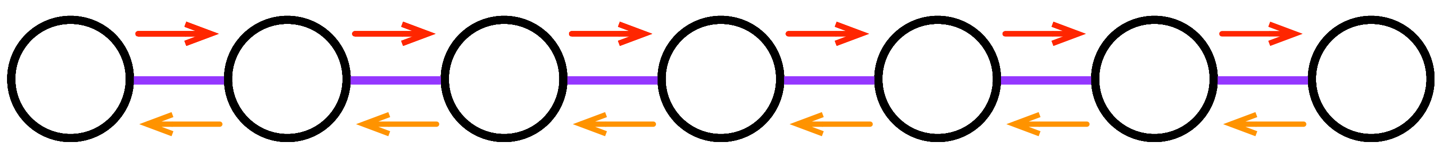
Example: binary hidden Markov model (HMM)

$$x = (x_1, x_2, \ldots, x_N), \qquad x_n \in \{0, 1\}$$

$$p(x) \propto \exp(\eta_1 x_1 + \eta_2 x_2 + \ldots$$
$$\eta_{12} x_1 x_2 + \eta_{23} x_2 x_3 + \ldots)$$

$$t(x) = x x^{\mathsf{T}}$$

$\eta =$ 



$$\mathcal{A}(\eta) = \log \sum_{x \in \{0,1\}^N} \exp(\langle \eta, t(x) \rangle)$$

$$\nabla \mathcal{A}(\eta) = (\mathbb{E}[x_1], \ldots, \mathbb{E}[x_N], \mathbb{E}[x_1 x_2], \ldots)$$

Example: hidden Markov model (HMM)

```
from scipy.stats import logsumexp

def log_normalizer(natparams, data):
  log_pi, log_A, log_B = natparams
  log_alpha = log_pi
  for y in data:
    log_alpha = logsumexp(log_alpha[:, None] + log_A, axis=0) + log_B[:, y]
  return logsumexp(log_alpha)


from autograd import grad
E_stats = grad(log_normalizer)(natparams, data)
```

https://github.com/HIPS/autograd/blob/master/examples/hmm_em.py

What about maximum likelihood?

What about maximum likelihood?

Say $\{x_n\}_{n=1}^N$ are samples, consider

$$\frac{1}{N}\log p(x\,;\,\eta) = \langle \eta,\ \frac{1}{N}\sum_{n=1}^{N} t(x_n)\rangle - \mathcal{A}(\eta)$$

$$= \langle \eta,\ \hat{\mu}\rangle - \mathcal{A}(\eta)$$

What about maximum likelihood?

Say $\{x_n\}_{n=1}^N$ are samples, consider

$$\frac{1}{N}\log p(x\,;\,\eta) = \langle \eta,\, \frac{1}{N}\sum_{n=1}^{N} t(x_n)\rangle - \mathcal{A}(\eta)$$

$$= \langle \eta,\, \hat{\mu}\rangle - \mathcal{A}(\eta)$$

So maximum likelihood is the concave problem

$$\mathcal{A}^*(\mu) \triangleq \sup_{\eta\in\Theta}\langle \eta,\, \mu\rangle - \mathcal{A}(\eta)$$

What about maximum likelihood?

Say $\{x_n\}_{n=1}^N$ are samples, consider

$$\frac{1}{N}\log p(x\,;\,\eta) = \langle \eta,\ \frac{1}{N}\sum_{n=1}^{N} t(x_n)\rangle - \mathcal{A}(\eta)$$

$$= \langle \eta,\ \hat{\mu}\rangle - \mathcal{A}(\eta)$$

So maximum likelihood is the concave problem

$$\mathcal{A}^*(\mu) \triangleq \sup_{\eta\in\Theta} \langle \eta,\ \mu\rangle - \mathcal{A}(\eta)$$

Gradient ascent is

$$\eta_{k+1} = \eta_k + \alpha_k(\mu - \nabla\mathcal{A}(\eta_k))$$

What about maximum likelihood?

Say $\{x_n\}_{n=1}^N$ are samples, consider

$$\frac{1}{N}\log p(x\,;\,\eta) = \langle \eta,\ \frac{1}{N}\sum_{n=1}^{N} t(x_n)\rangle - \mathcal{A}(\eta)$$
$$= \langle \eta,\ \hat{\mu}\rangle - \mathcal{A}(\eta)$$

So maximum likelihood is the concave problem

$$\mathcal{A}^*(\mu) \triangleq \sup_{\eta\in\Theta} \langle \eta,\ \mu\rangle - \mathcal{A}(\eta)$$

Gradient ascent is

$$\eta_{k+1} = \eta_k + \alpha_k(\mu - \nabla\mathcal{A}(\eta_k))$$

Claim

$$\nabla\mathcal{A}^*(\mu) = \arg\sup_{\eta\in\Theta} \langle \eta,\ \mu\rangle - \mathcal{A}(\eta)$$

$$\mathcal{A}^*(\mu) = \sup_{\eta \in \Theta} \langle \eta,\, \mu \rangle - \mathcal{A}(\eta) \qquad \mathcal{A}(\eta) = \sup_{\mu \in \mathcal{M}} \langle \eta,\, \mu \rangle - \mathcal{A}^*(\mu)$$

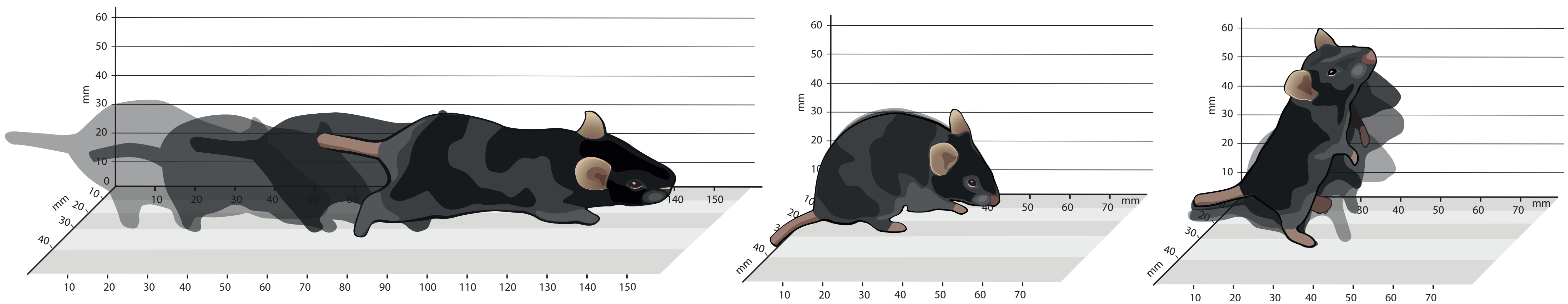
$$\eta(\mu) = \nabla \mathcal{A}^*(\mu) \qquad \mu(\eta) = \nabla \mathcal{A}(\eta)$$

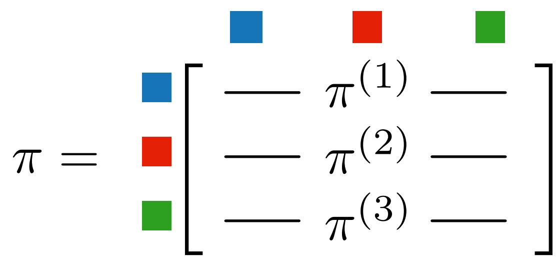
Summary for tractable exponential families

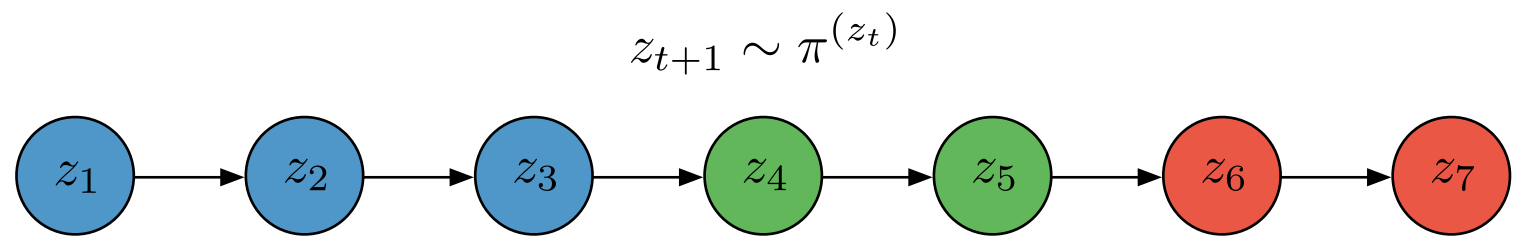
For each tractable exponential family…

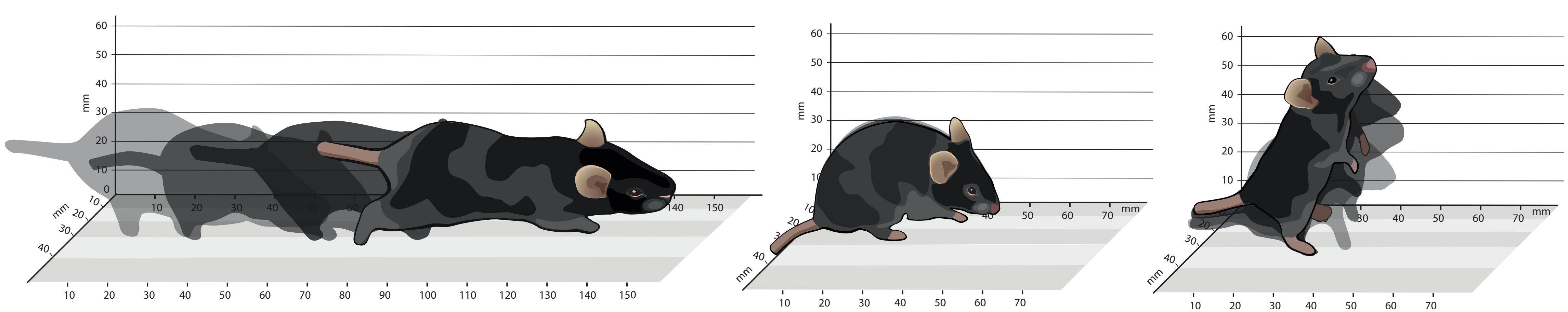
1. implement $t(x)$ and $\mathcal{A}(\eta)$ for exact inference
   and mean field variational inference
2. implement $\mathcal{A}^*(\mu)$ for maximum likelihood
   and (variational) expectation-maximization (EM)
3. implement `sample` for drawing samples
   and Gibbs sampling MCMC

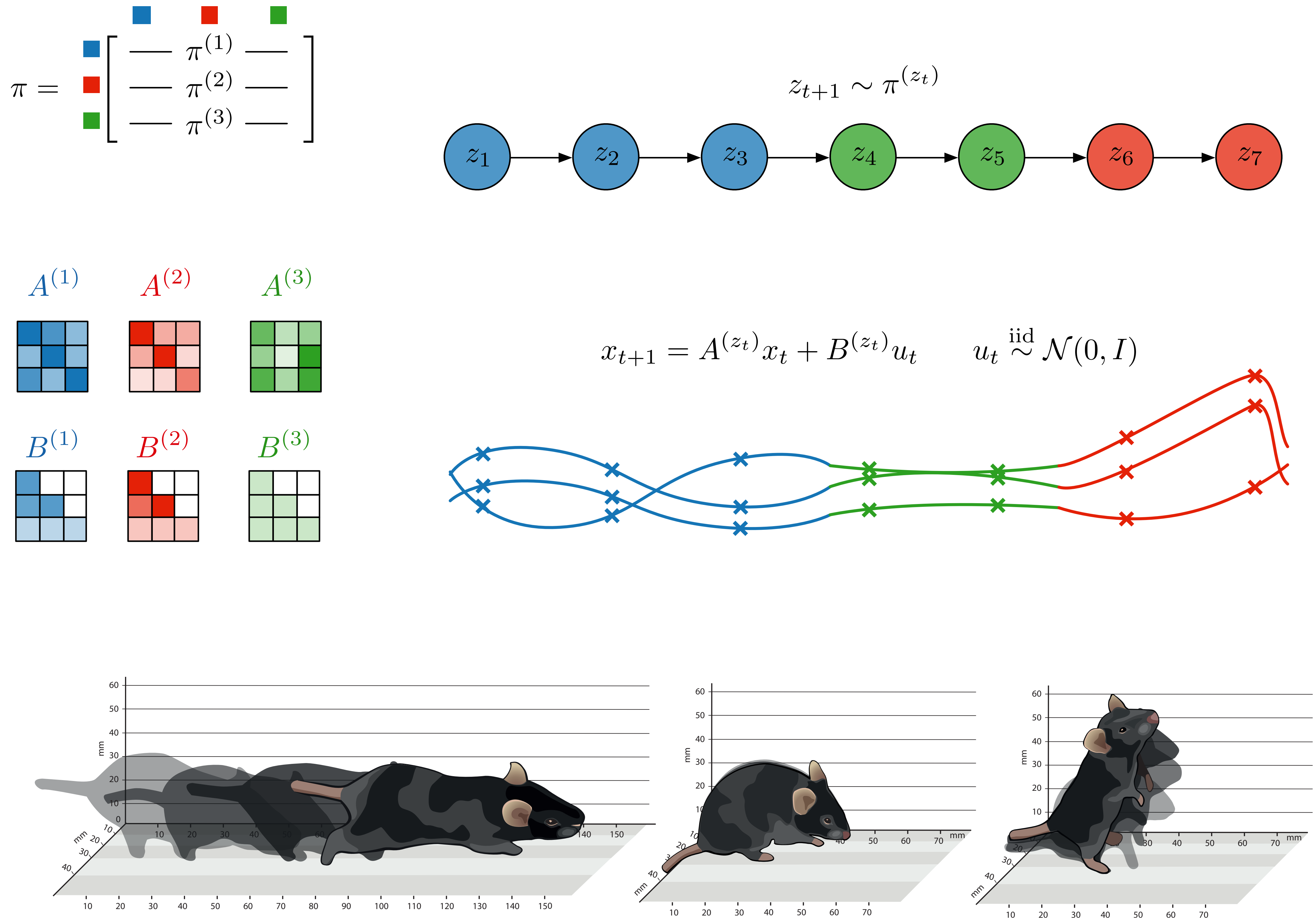
Next up: composing tractable families into intractable ones!


mm
mm
mm

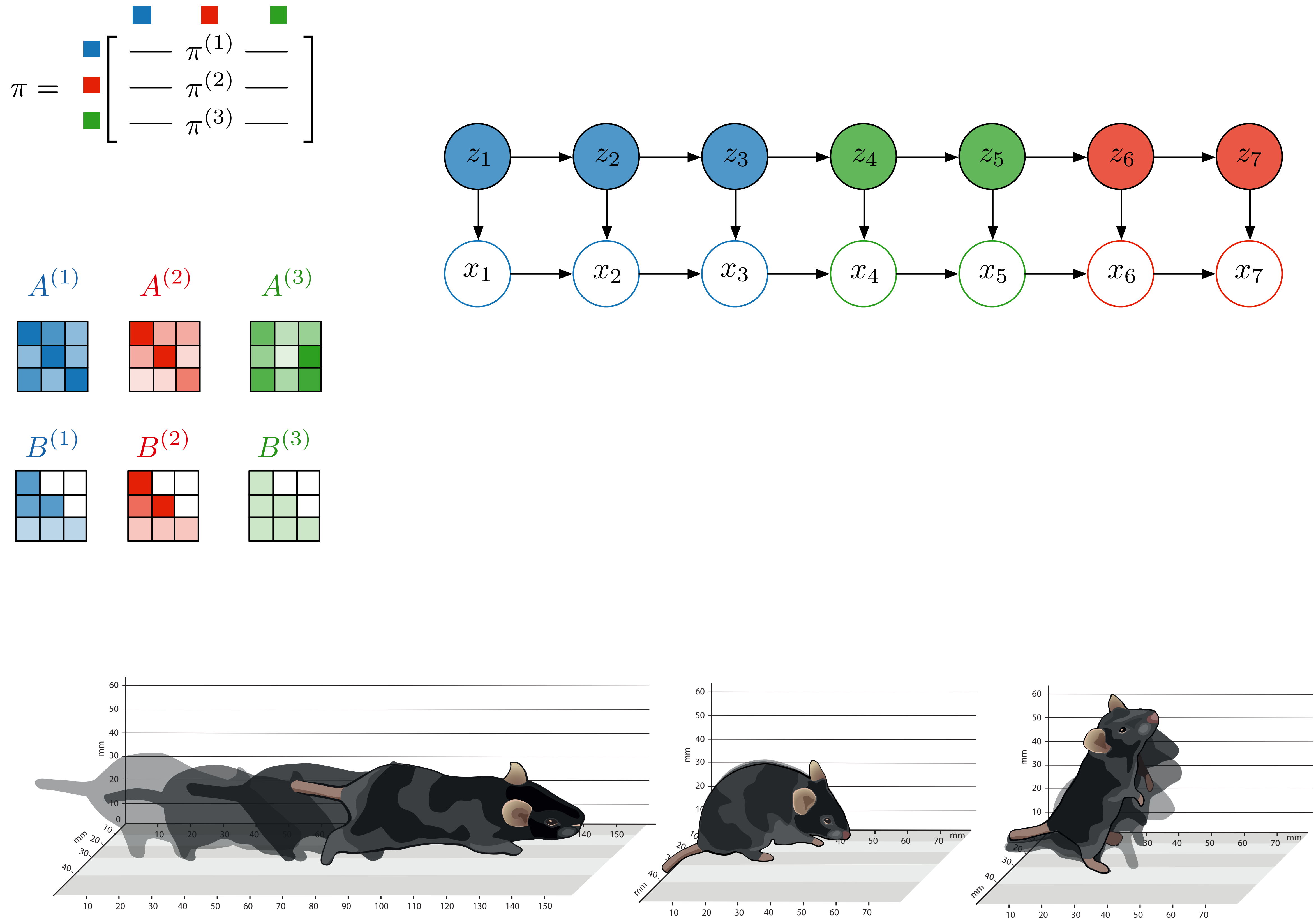

$\pi = \begin{bmatrix} — \pi^{(1)} — \\ — \pi^{(2)} — \\ — \pi^{(3)} — \end{bmatrix}$


$z_{t+1} \sim \pi^{(z_t)}$
z_1 z_2 z_3 z_4 z_5 z_6 z_7

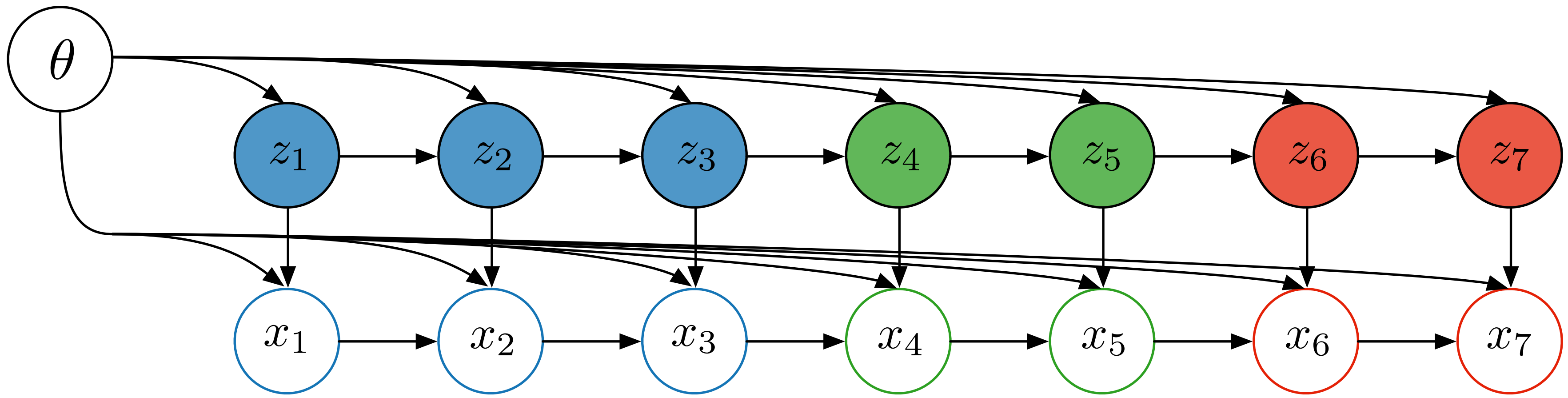




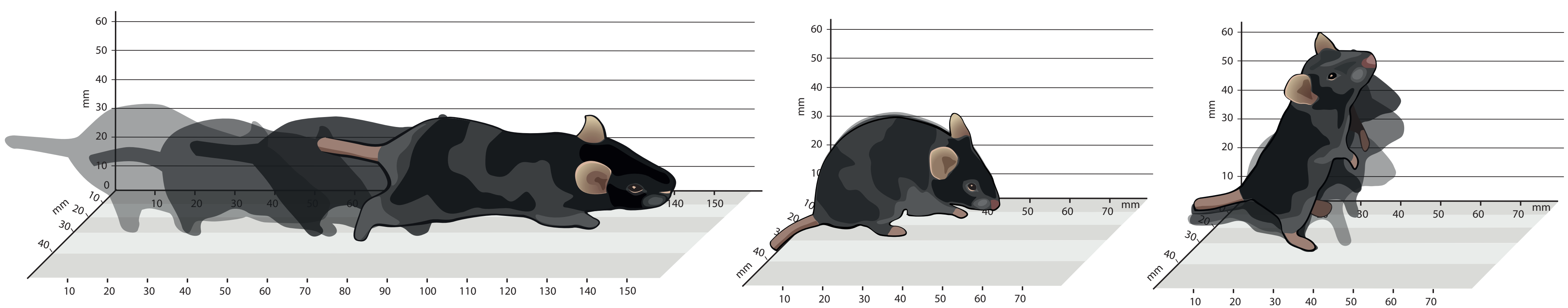
$\pi = \begin{bmatrix} — \pi^{(1)} — \\ — \pi^{(2)} — \\ — \pi^{(3)} — \end{bmatrix}$
$z_{t+1} \sim \pi^{(z_t)}$
z_1 z_2 z_3 z_4 z_5 z_6 z_7
$A^{(1)}$ $A^{(2)}$ $A^{(3)}$
$B^{(1)}$ $B^{(2)}$ $B^{(3)}$
$x_{t+1} = A^{(z_t)} x_t + B^{(z_t)} u_t \qquad u_t \overset{\text{iid}}{\sim} \mathcal{N}(0, I)$
mm

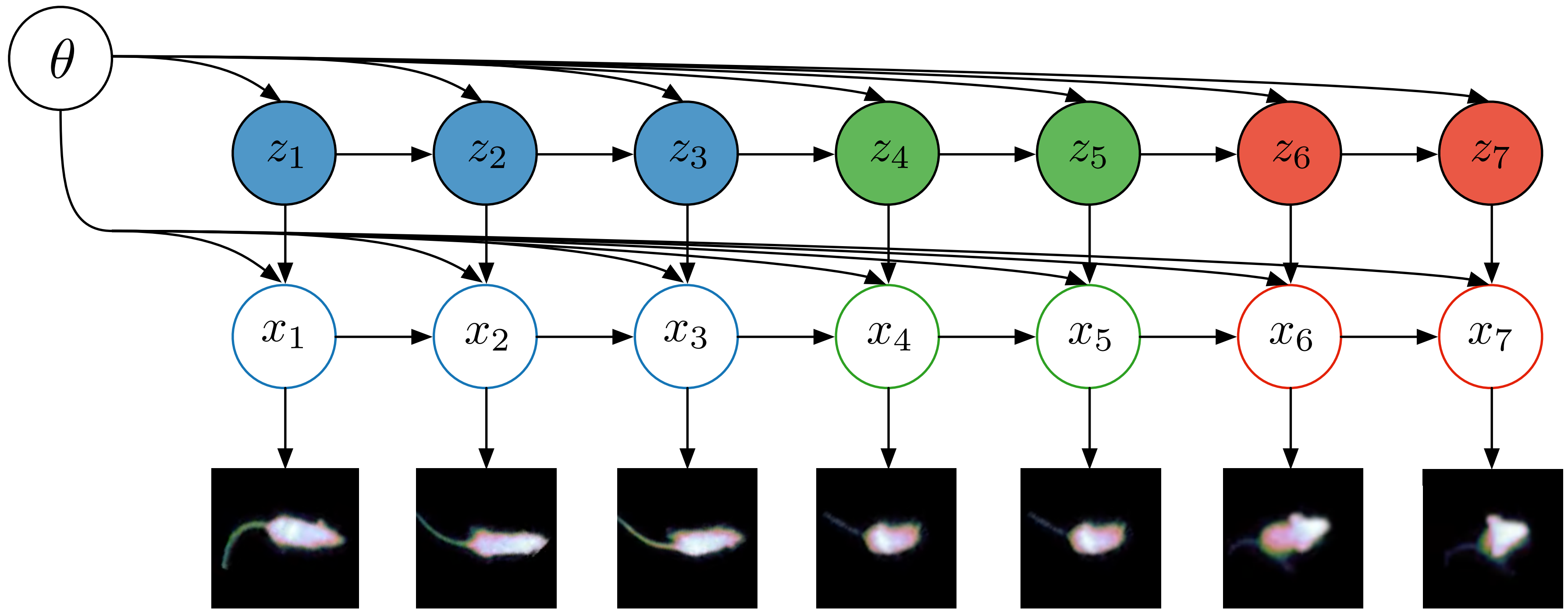


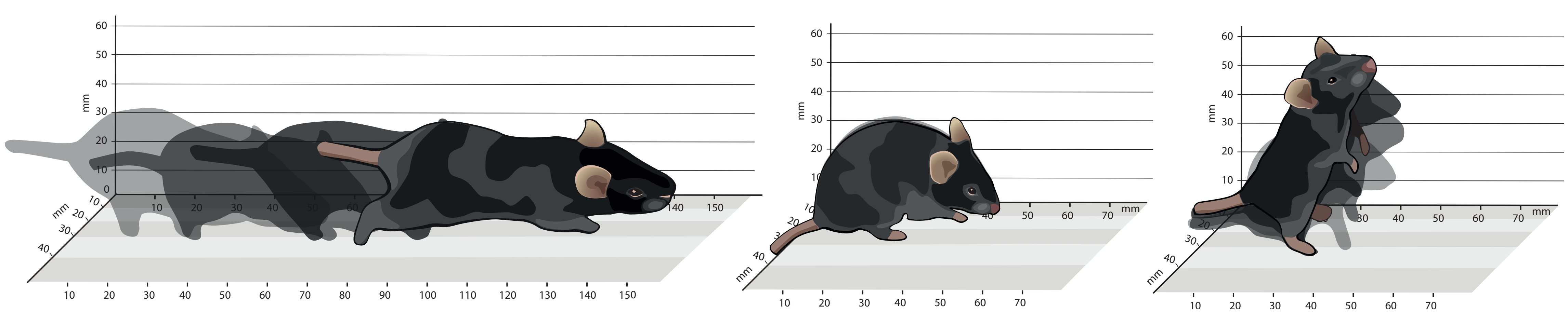
$\pi = \begin{bmatrix} — \pi^{(1)} — \\ — \pi^{(2)} — \\ — \pi^{(3)} — \end{bmatrix}$
z_1 z_2 z_3 z_4 z_5 z_6 z_7
x_1 x_2 x_3 x_4 x_5 x_6 x_7
$A^{(1)}$ $A^{(2)}$ $A^{(3)}$
$B^{(1)}$ $B^{(2)}$ $B^{(3)}$
mm

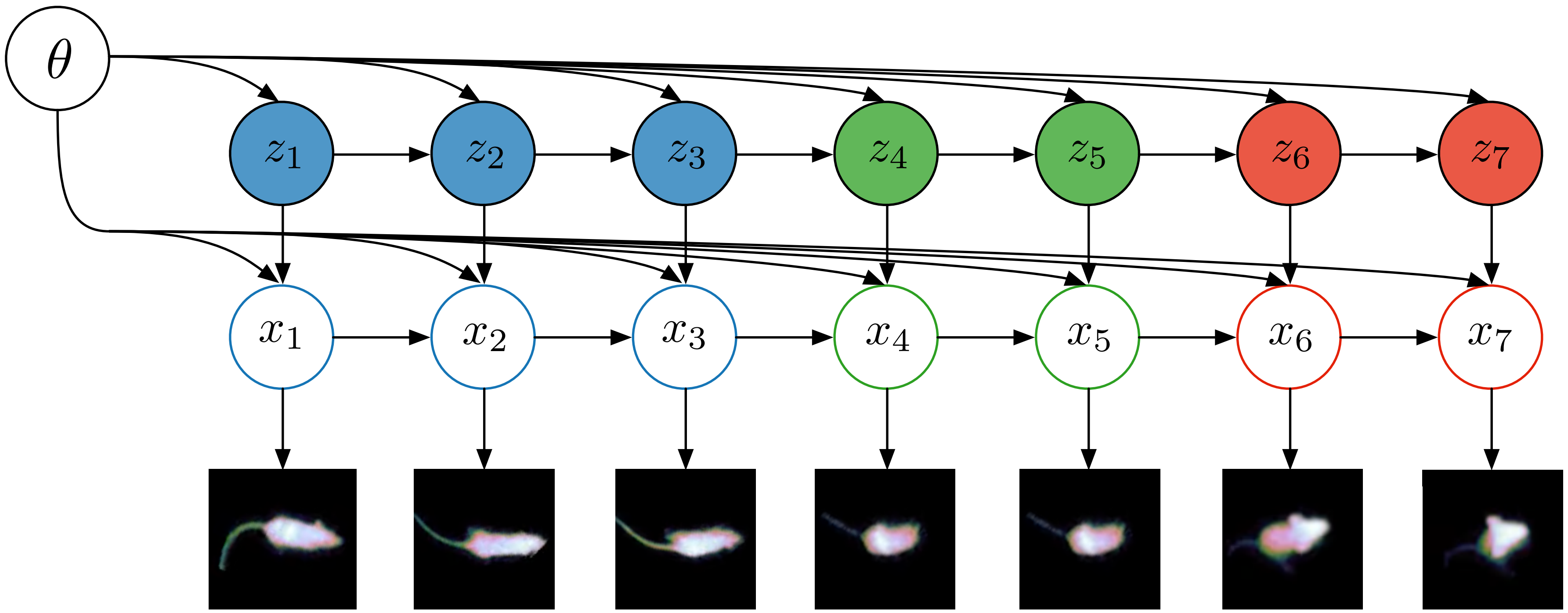


θ
z_1
z_2
z_3
z_4
z_5
z_6
z_7
x_1
x_2
x_3
x_4
x_5
x_6
x_7

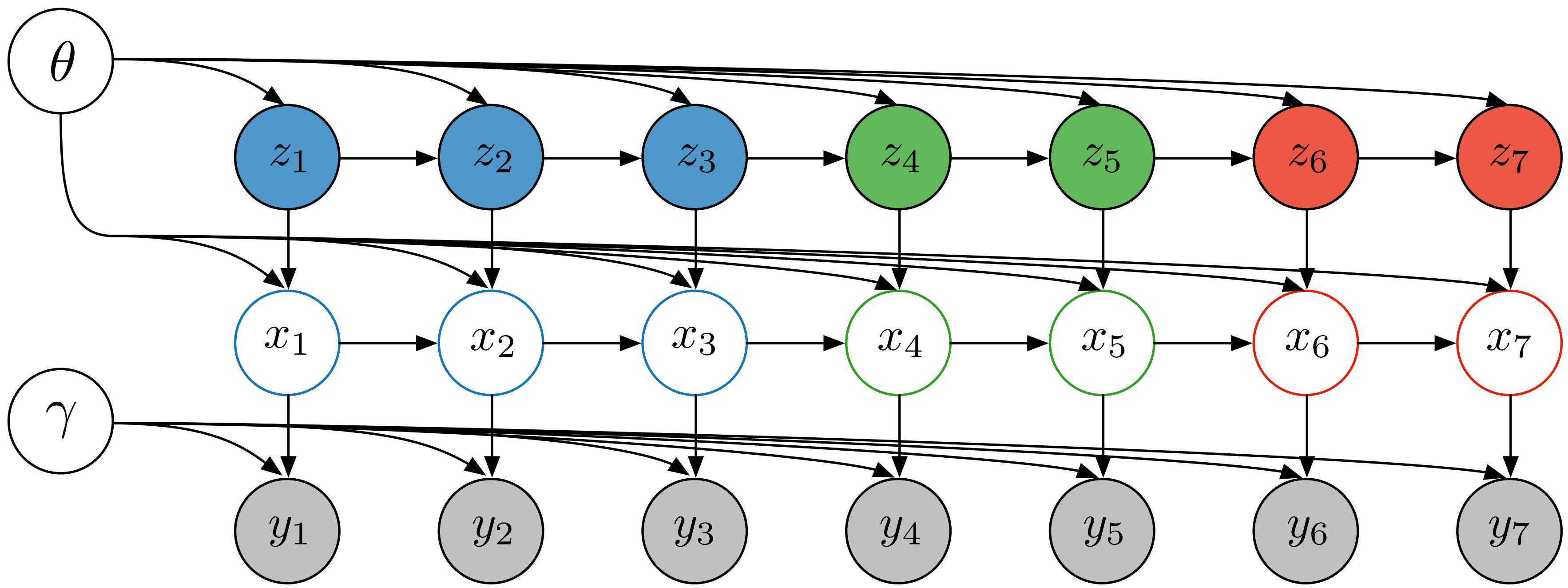



θ
z_1 z_2 z_3 z_4 z_5 z_6 z_7
x_1 x_2 x_3 x_4 x_5 x_6 x_7


mm
60 50 40 30 20 10 0
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150
mm
60 50 40 30 20 10
10 20 30 40 50 60 70
mm
60 50 40 30 20 10
10 20 30 40 50 60 70



θ
z_1
z_2
z_3
z_4
z_5
z_6
z_7
x_1
x_2
x_3
x_4
x_5
x_6
x_7



$$y_t \mid x_t \sim \mathcal{N}(Cx_t, \Sigma), \qquad \gamma = (C, \Sigma)$$

Compose exp. families, get compositional algorithms?

Compose exp. families, get compositional algorithms?

For $x = (x_1, x_2)$ consider a negative energy function

$$\begin{aligned}\log p(x\,;\,\eta) &= \langle \eta,\ t(x)\rangle + \text{const.}\\ &= \langle \eta_{10},\ t_1(x_1)\rangle + \langle \eta_{01},\ t_2(x_2)\rangle\\ &\quad + \langle \eta_{11},\ t_1(x_1) \otimes t_2(x_2)\rangle + \text{const.}\end{aligned}$$

Compose exp. families, get compositional algorithms?

For $x = (x_1, x_2)$ consider a negative energy function

$$\begin{aligned}\log p(x\,;\,\eta) &= \langle \eta,\, t(x) \rangle + \text{const.} \\ &= \langle \eta_{10},\, t_1(x_1) \rangle + \langle \eta_{01},\, t_2(x_2) \rangle \\ &\quad + \langle \eta_{11},\, t_1(x_1) \otimes t_2(x_2) \rangle + \text{const.}\end{aligned}$$

$$\log p(x_1 \mid x_2) = \langle \eta_{10} + \eta_{11} \cdot t_2(x_2),\, t_1(x_1) \rangle + \text{const.}$$

Compose exp. families, get compositional algorithms?

For $x = (x_1, x_2)$ consider a negative energy function

$$\begin{aligned}\log p(x\,;\,\eta) &= \langle \eta,\, t(x) \rangle + \text{const.} \\ &= \langle \eta_{10},\, t_1(x_1) \rangle + \langle \eta_{01},\, t_2(x_2) \rangle \\ &\quad + \langle \eta_{11},\, t_1(x_1) \otimes t_2(x_2) \rangle + \text{const.}\end{aligned}$$

Compose exp. families, get compositional algorithms?

For $x = (x_1, \ldots, x_M)$ consider a negative energy function

$$\begin{aligned}
\log p(x\,;\,\eta) &= \langle \eta,\, t(x) \rangle + \text{const.} \\
&= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta,\, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle \\
&\triangleq g(t_1(x_1), \ldots, t_M(x_M))
\end{aligned}$$

Compose exp. families, get compositional algorithms?

For $x = (x_1, \ldots, x_M)$ consider a negative energy function

$$\begin{aligned} \log p(x\,;\,\eta) &= \langle \eta,\, t(x) \rangle + \text{const.} \\ &= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta,\, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle \\ &\triangleq g(t_1(x_1), \ldots, t_M(x_M)) \end{aligned}$$

where g is a multi-affine polynomial

Compose exp. families, get compositional algorithms?

For $x = (x_1, \ldots, x_M)$ consider a negative energy function

$$\begin{aligned}\log p(x\,;\,\eta) &= \langle \eta, \, t(x) \rangle + \text{const.} \\ &= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta, \, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle \\ &\triangleq g(t_1(x_1), \ldots, t_M(x_M))\end{aligned}$$

where g is a multi-affine polynomial

$\beta \in \boldsymbol{\beta} \subseteq \{0,1\}^M$ indexes monomial terms

Compose exp. families, get compositional algorithms?

For $x = (x_1, \ldots, x_M)$ consider a negative energy function

$$\begin{aligned} \log p(x\,;\,\eta) &= \langle \eta,\, t(x) \rangle + \text{const.} \\ &= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta,\, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle \\ &\triangleq g(t_1(x_1), \ldots, t_M(x_M)) \end{aligned}$$

where g is a multi-affine polynomial

$\beta \in \boldsymbol{\beta} \subseteq \{0,1\}^M$ indexes monomial terms

each $t_m(x_m)$ corresponds to a tractable exponential family

Compose exp. families, get compositional algorithms?

For $x = (x_1, \ldots, x_M)$ consider a negative energy function

$$
\begin{aligned}
\log p(x\,;\,\eta) &= \langle \eta, t(x) \rangle + \text{const.} \\
&= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle \\
&\triangleq g(t_1(x_1), \ldots, t_M(x_M))
\end{aligned}
$$

where g is a multi-affine polynomial

$\beta \in \boldsymbol{\beta} \subseteq \{0,1\}^M$ indexes monomial terms

each $t_m(x_m)$ corresponds to a tractable exponential family

But the normalizer $\mathcal{A}$ is not tractable!

$$\begin{aligned}\log p(x\,;\,\eta) &= \langle \eta,\ t(x)\rangle + \text{const.}\\ &= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta,\ t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle\\ &\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}\end{aligned}$$

$$\begin{aligned}\log p(x\,;\,\eta) &= \langle \eta,\, t(x)\rangle + \text{const.}\\ &= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta,\, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle \\ &\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}\end{aligned}$$

Can we build algorithms for

1. approximate sampling $x \sim p(x\,;\,\eta)$ via MCMC?
2. approximate expectations $\mathbb{E}[t(X)]$ and $\mathcal{A}$?
3. variational Expectation-Maximization to estimate η?

that exploit the tractable parts?

$$
\begin{aligned}
\log p(x\,;\,\eta) &= \langle \eta,\ t(x) \rangle + \text{const.} \\
&= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta,\ t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle \\
&\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}
\end{aligned}
$$

$$\begin{aligned}
\log p(x\,;\,\eta) &= \langle \eta,\ t(x)\rangle + \text{const.} \\
&= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta,\ t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle \\
&\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}
\end{aligned}$$

Claim [Gibbs sampling]

$$p(x_m \,|\, x_{\neg m}) = \exp(\langle \eta_m^*, t_m(x_m)\rangle - \mathcal{A}_m(\eta_m^*))$$

$$\eta_m^* \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))$$

$$
\begin{aligned}
\log p(x\,;\,\eta) &= \langle \eta,\, t(x) \rangle + \text{const.} \\
&= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta,\, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle \\
&\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}
\end{aligned}
$$

Claim [Gibbs sampling]

$$
p(x_m \,|\, x_{\neg m}) = \exp(\langle \eta_m^*, t_m(x_m) \rangle - \mathcal{A}_m(\eta_m^*))
$$

$$
\eta_m^* \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))
$$

```
from autograd import grad

def gibbs(g, samplers, niter, x):
  for _ in range(niter):
    for m in range(M):
      x[m] = samplers[m](grad(g)(*x))
  return x
```

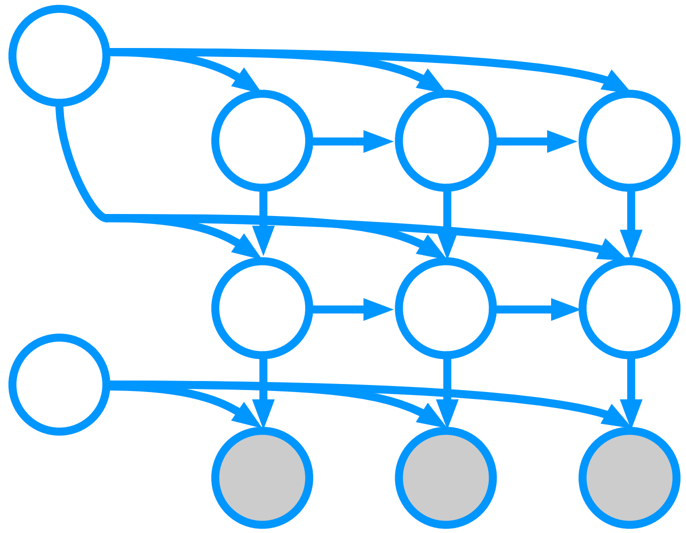
$$
\begin{aligned}
\log p(x\,;\,\eta) &= \langle \eta,\, t(x)\rangle + \text{const.}\\
&= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta,\, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle\\
&\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}
\end{aligned}
$$

Claim [Gibbs sampling]

$$
p(x_m \mid x_{\neg m}) = \exp(\langle \eta_m^*, t_m(x_m)\rangle - \mathcal{A}_m(\eta_m^*))
$$

$$
\eta_m^* \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))
$$

```
from autograd import grad

def gibbs(g, samplers, niter, x):
  for _ in range(niter):
    for m in range(M):
      x[m] = samplers[m](grad(g)(*x))
  return x
```



$$\log p(x\,;\,\eta) = \langle \eta,\, t(x) \rangle + \text{const.}$$

$$= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta,\, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle$$

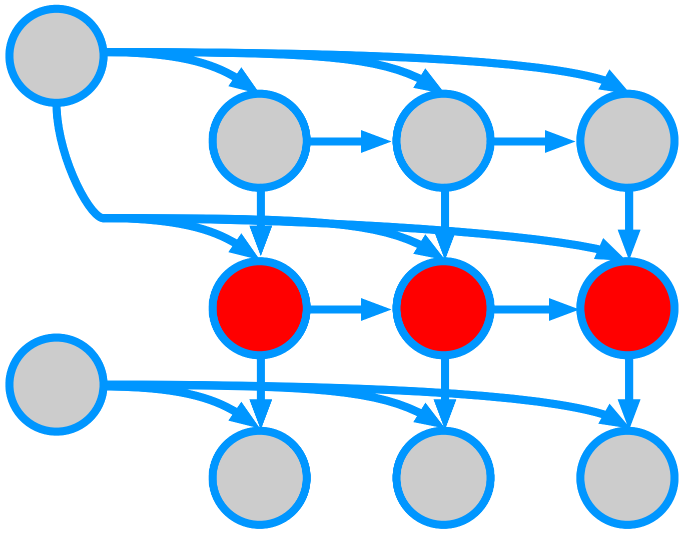
$$\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}$$

Claim [Gibbs sampling]

$$p(x_m \,|\, x_{\neg m}) = \exp(\langle \eta_m^*, t_m(x_m) \rangle - \mathcal{A}_m(\eta_m^*))$$

$$\eta_m^* \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))$$

```
from autograd import grad

def gibbs(g, samplers, niter, x):
  for _ in range(niter):
    for m in range(M):
      x[m] = samplers[m](grad(g)(*x))
  return x
```



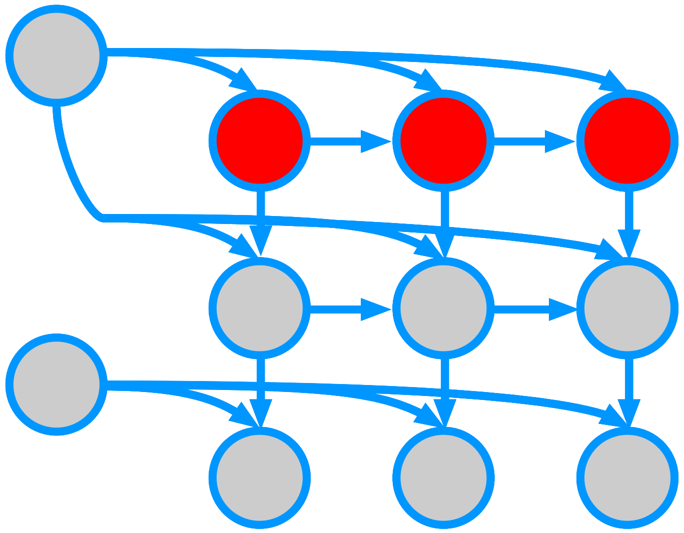
$$\begin{aligned}\log p(x\,;\,\eta) &= \langle \eta,\, t(x)\rangle + \text{const.}\\ &= \sum_{\beta\in\boldsymbol{\beta}} \langle \eta_\beta,\, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M}\rangle\\ &\triangleq g(t_1(x_1),\ldots,t_M(x_M)) + \text{const.}\end{aligned}$$

Claim [Gibbs sampling]

$$p(x_m \,|\, x_{\neg m}) = \exp(\langle \eta_m^*, t_m(x_m)\rangle - \mathcal{A}_m(\eta_m^*))$$

$$\eta_m^* \triangleq \nabla_m g(t_1(x_1),\ldots,t_M(x_M))$$

```
from autograd import grad

def gibbs(g, samplers, niter, x):
  for _ in range(niter):
    for m in range(M):
      x[m] = samplers[m](grad(g)(*x))
  return x
```



$$\log p(x\,;\,\eta) = \langle \eta,\, t(x) \rangle + \text{const.}$$

$$= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta,\, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle$$

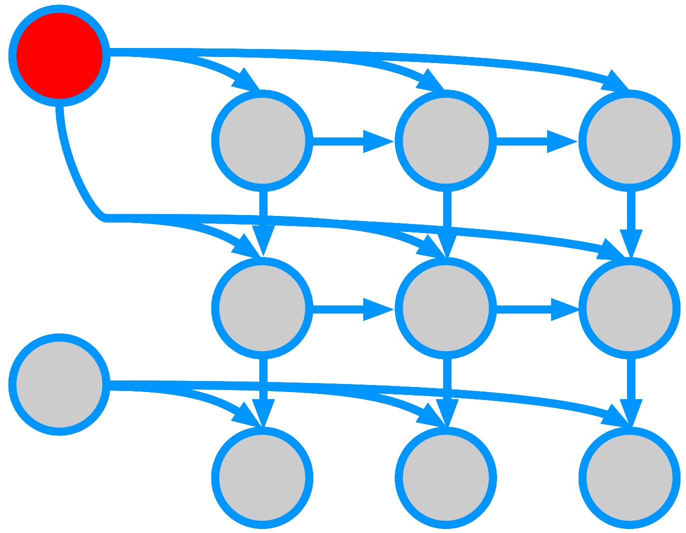
$$\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}$$

Claim [Gibbs sampling]

$$p(x_m \,|\, x_{\neg m}) = \exp(\langle \eta_m^*, t_m(x_m) \rangle - \mathcal{A}_m(\eta_m^*))$$

$$\eta_m^* \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))$$

```
from autograd import grad

def gibbs(g, samplers, niter, x):
  for _ in range(niter):
    for m in range(M):
      x[m] = samplers[m](grad(g)(*x))
  return x
```



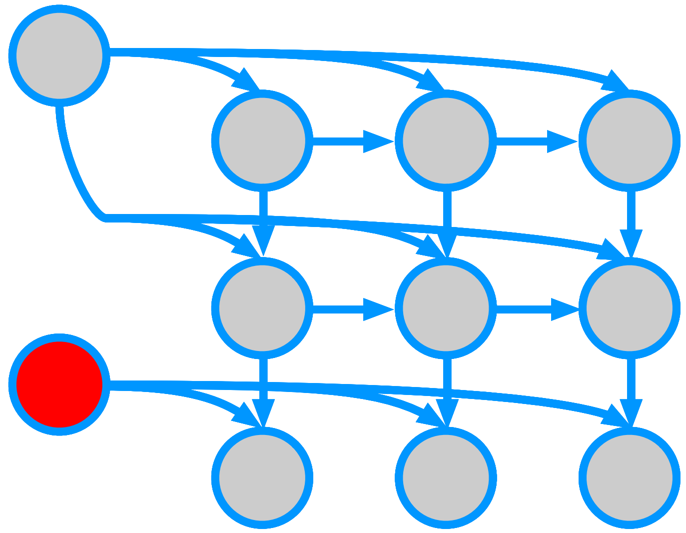
$$\log p(x\,;\,\eta) = \langle \eta,\, t(x)\rangle + \text{const.}$$
$$= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta,\, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle$$
$$\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}$$

Claim [Gibbs sampling]

$$p(x_m \,|\, x_{\neg m}) = \exp(\langle \eta_m^*, t_m(x_m)\rangle - \mathcal{A}_m(\eta_m^*))$$

$$\eta_m^* \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))$$

```
from autograd import grad

def gibbs(g, samplers, niter, x):
  for _ in range(niter):
    for m in range(M):
      x[m] = samplers[m](grad(g)(*x))
  return x
```



$$\log p(x\,;\,\eta) = \langle \eta,\, t(x)\rangle + \text{const.}$$

$$= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta,\, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle$$

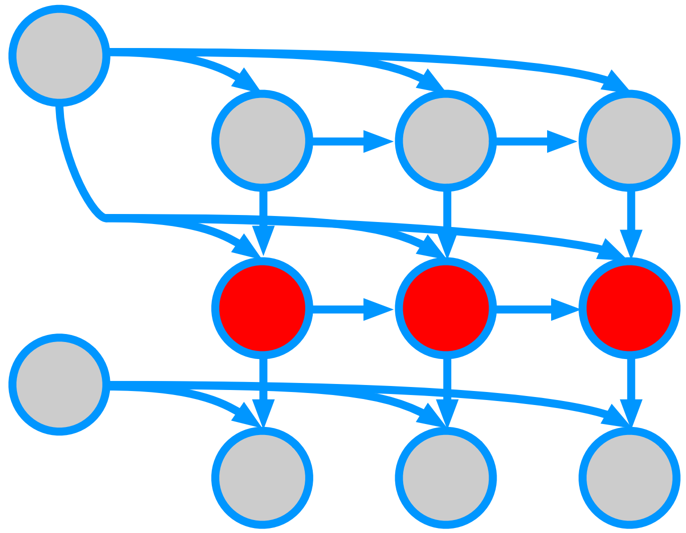
$$\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}$$

Claim [Gibbs sampling]

$$p(x_m \mid x_{\neg m}) = \exp(\langle \eta_m^*, t_m(x_m)\rangle - \mathcal{A}_m(\eta_m^*))$$

$$\eta_m^* \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))$$

```
from autograd import grad

def gibbs(g, samplers, niter, x):
  for _ in range(niter):
    for m in range(M):
      x[m] = samplers[m](grad(g)(*x))
  return x
```



$$\log p(x\,;\,\eta) = \langle \eta,\, t(x) \rangle + \text{const.}$$

$$= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta,\, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle$$

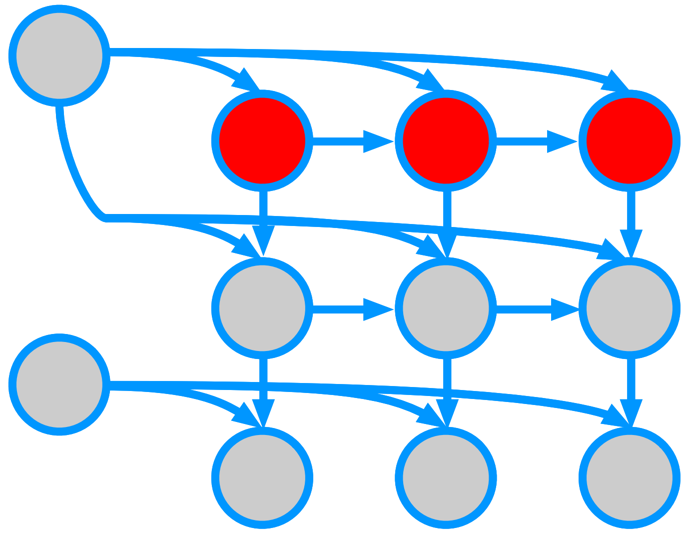
$$\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}$$

Claim [Gibbs sampling]

$$p(x_m \,|\, x_{\neg m}) = \exp(\langle \eta_m^*, t_m(x_m) \rangle - \mathcal{A}_m(\eta_m^*))$$

$$\eta_m^* \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))$$

```
from autograd import grad

def gibbs(g, samplers, niter, x):
  for _ in range(niter):
    for m in range(M):
      x[m] = samplers[m](grad(g)(*x))
  return x
```



$$\log p(x\,;\,\eta) = \langle \eta,\, t(x) \rangle + \text{const.}$$

$$= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta,\, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle$$

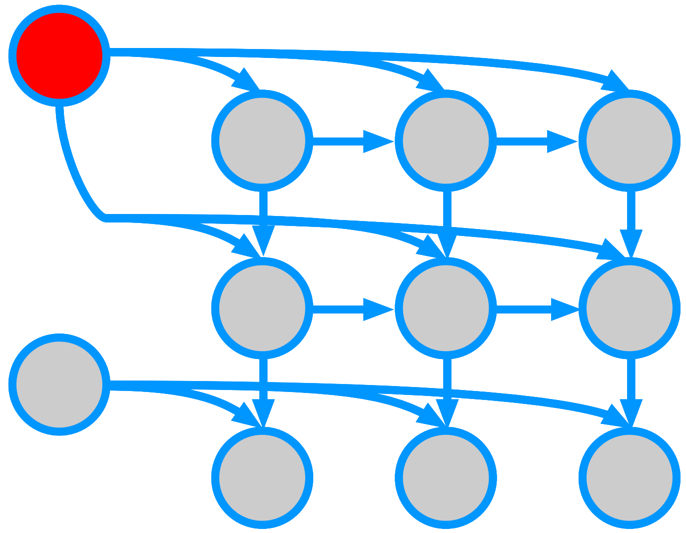
$$\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}$$

Claim [Gibbs sampling]

$$p(x_m \,|\, x_{\neg m}) = \exp(\langle \eta_m^*, t_m(x_m) \rangle - \mathcal{A}_m(\eta_m^*))$$

$$\eta_m^* \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))$$

```
from autograd import grad

def gibbs(g, samplers, niter, x):
  for _ in range(niter):
    for m in range(M):
      x[m] = samplers[m](grad(g)(*x))
  return x
```



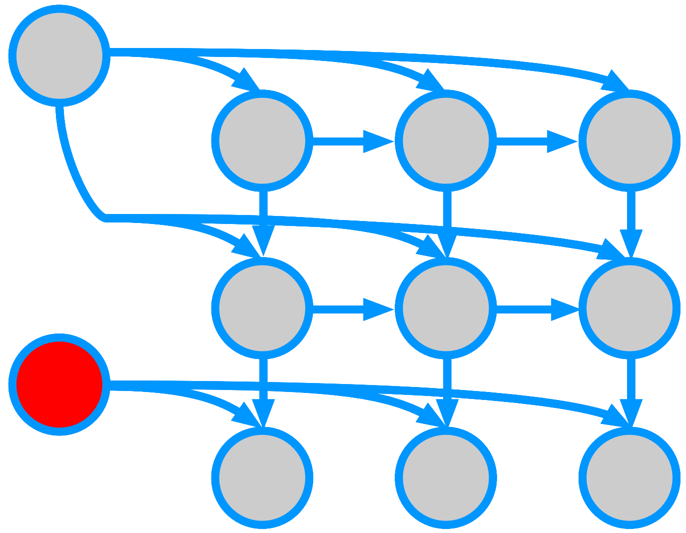
$$\log p(x\,;\,\eta) = \langle \eta,\, t(x) \rangle + \text{const.}$$
$$= \sum_{\beta \in \boldsymbol{\beta}} \langle \eta_\beta,\, t_1(x_1)^{\beta_1} \otimes t_2(x_2)^{\beta_2} \otimes \cdots \otimes t_M(x_M)^{\beta_M} \rangle$$
$$\triangleq g(t_1(x_1), \ldots, t_M(x_M)) + \text{const.}$$

Claim [Gibbs sampling]

$$p(x_m \mid x_{\neg m}) = \exp(\langle \eta_m^*, t_m(x_m) \rangle - \mathcal{A}_m(\eta_m^*))$$

$$\eta_m^* \triangleq \nabla_m g(t_1(x_1), \ldots, t_M(x_M))$$

```
from autograd import grad

def gibbs(g, samplers, niter, x):
  for _ in range(niter):
    for m in range(M):
      x[m] = samplers[m](grad(g)(*x))
  return x
```



Consider the variational family

$$q(x) = \prod_{m=1}^{M} q_m(x_m\,;\,\eta_m)$$

$$q(x_m\,;\,\eta_m) = \exp(\langle \eta_m, t_m(x_m)\rangle - \mathcal{A}_m(\eta_m))$$

Consider the variational family

$$q(x) = \prod_{m=1}^{M} q_m(x_m\,;\,\eta_m)$$

$$q(x_m\,;\,\eta_m) = \exp(\langle \eta_m, t_m(x_m) \rangle - \mathcal{A}_m(\eta_m))$$

That's a sub-family of $p(x\,;\,\eta)$ with linear constraints on η.

Consider the variational family

$$q(x) = \prod_{m=1}^{M} q_m(x_m\,;\,\eta_m)$$

$$q(x_m\,;\,\eta_m) = \exp(\langle \eta_m,\, t_m(x_m)\rangle - \mathcal{A}_m(\eta_m))$$

That's a sub-family of $p(x\,;\,\eta)$ with linear constraints on η.

Claim [Variational lower bound]

$$\mathcal{A}(\eta) \geq \langle \eta,\, \mathbb{E}_q[t(X)]\rangle - \sum_m \mathcal{A}_m^*(\mu_m)$$

$$= g(\mu_1, \ldots, \mu_M) - \sum_m \mathcal{A}_m^*(\mu_m) \triangleq \mathcal{L}(\eta)$$

Consider the variational family

$$q(x) = \prod_{m=1}^{M} q_m(x_m\,;\,\eta_m)$$

$$q(x_m\,;\,\eta_m) = \exp(\langle \eta_m,\, t_m(x_m) \rangle - \mathcal{A}_m(\eta_m))$$

That's a sub-family of $p(x\,;\,\eta)$ with linear constraints on η.

Claim [Variational lower bound]

$$\begin{aligned}\mathcal{A}(\eta) &\geq \langle \eta,\, \mathbb{E}_q[t(X)] \rangle - \textstyle\sum_m \mathcal{A}_m^*(\mu_m) \\ &= g(\mu_1, \ldots, \mu_M) - \textstyle\sum_m \mathcal{A}_m^*(\mu_m) \triangleq \mathcal{L}(\eta)\end{aligned}$$

Proof Use the variational definition of $\mathcal{A}$:

$$\mathcal{A}(\eta) = \sup_{\mu \in \mathcal{M}} \langle \eta,\, \mu \rangle - \mathcal{A}^*(\mu)$$

Consider the variational family

$$q(x) = \prod_{m=1}^{M} q_m(x_m\,;\,\eta_m)$$

$$q(x_m\,;\,\eta_m) = \exp(\langle \eta_m,\, t_m(x_m) \rangle - \mathcal{A}_m(\eta_m))$$

That's a sub-family of $p(x\,;\,\eta)$ with linear constraints on η.

$$g(\nabla \mathcal{A}_1(\eta_1), \ldots, \nabla \mathcal{A}_M(\eta_M)) - (\textstyle\sum_m \langle \eta_m,\, \nabla \mathcal{A}_m(\eta_m) \rangle - \mathcal{A}_m(\eta_m))$$

Consider the variational family

$$q(x) = \prod_{m=1}^{M} q_m(x_m\,;\,\eta_m)$$

$$q(x_m\,;\,\eta_m) = \exp(\langle \eta_m,\, t_m(x_m) \rangle - \mathcal{A}_m(\eta_m))$$

That's a sub-family of $p(x\,;\,\eta)$ with linear constraints on η.

Claim [Variational lower bound]

$$\begin{aligned} \mathcal{A}(\eta) &\geq \langle \eta,\, \mathbb{E}_q[t(X)] \rangle - \textstyle\sum_m \mathcal{A}_m^*(\mu_m) \\ &= g(\mu_1, \ldots, \mu_M) - \textstyle\sum_m \mathcal{A}_m^*(\mu_m) \triangleq \mathcal{L}(\eta) \end{aligned}$$

$$g(\nabla \mathcal{A}_1(\eta_1), \ldots, \nabla \mathcal{A}_M(\eta_M)) - (\textstyle\sum_m \langle \eta_m,\, \nabla \mathcal{A}_m(\eta_m) \rangle - \mathcal{A}_m(\eta_m))$$

Consider the variational family

$$q(x) = \prod_{m=1}^{M} q_m(x_m\,;\,\eta_m)$$

$$q(x_m\,;\,\eta_m) = \exp(\langle \eta_m, t_m(x_m)\rangle - \mathcal{A}_m(\eta_m))$$

Claim [Structured mean field]

$$\arg\max_{\eta_m} \mathcal{L}(\eta_1, \ldots, \eta_M) = \nabla_m g(\mu_1, \ldots, \mu_M))$$

$$\text{where} \;\; \mu_{m'} = \nabla \mathcal{A}_{m'}(\eta_{m'}) \;\; \text{for} \;\; m' = 1, \ldots, M$$

Consider the variational family

$$q(x) = \prod_{m=1}^{M} q_m(x_m\,;\,\eta_m)$$

$$q(x_m\,;\,\eta_m) = \exp(\langle \eta_m, t_m(x_m) \rangle - \mathcal{A}_m(\eta_m))$$

Claim [Structured mean field]

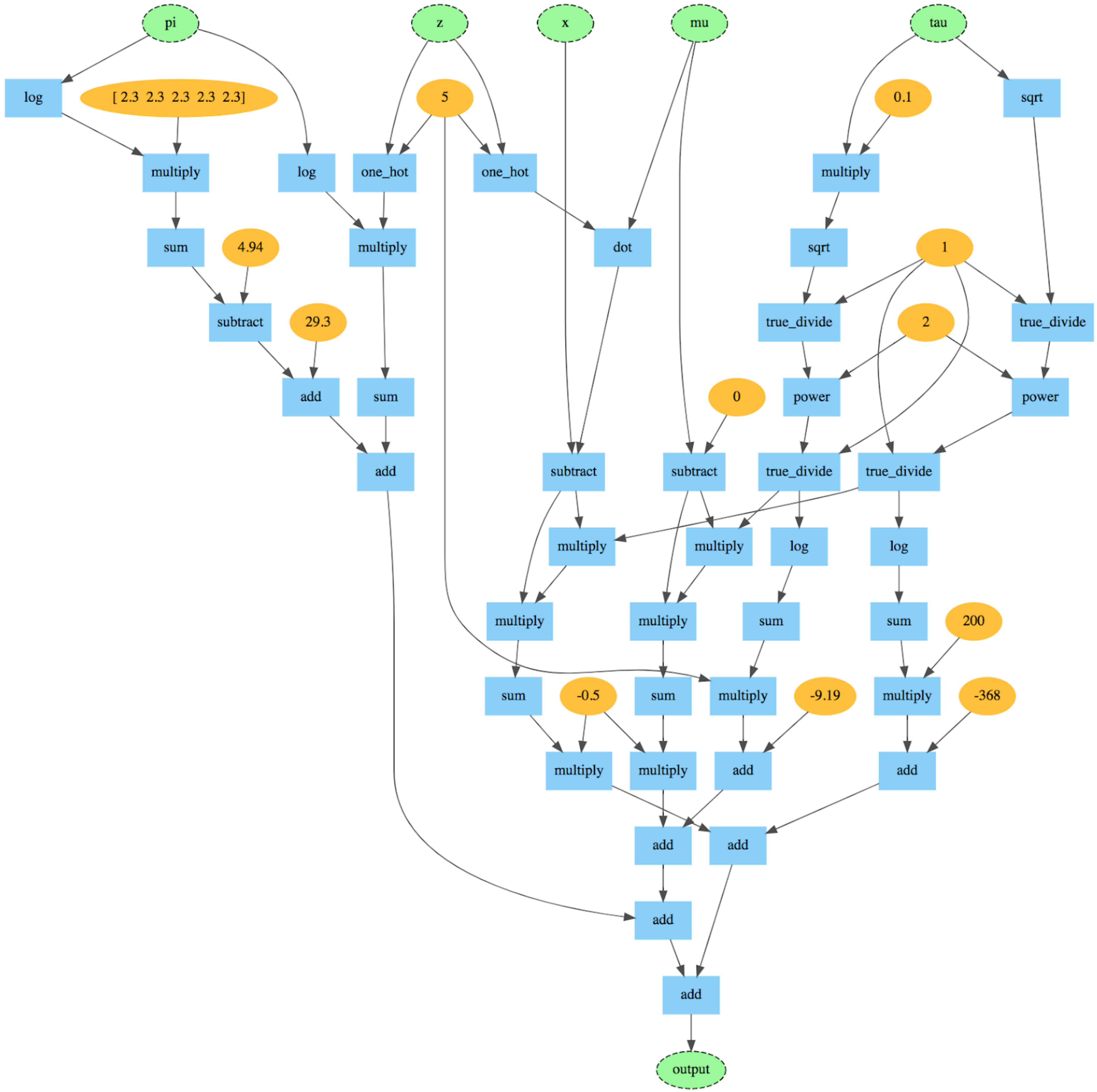
$$\arg\max_{\eta_m} \mathcal{L}(\eta_1, \ldots, \eta_M) = \nabla_m g(\mu_1, \ldots, \mu_M))$$

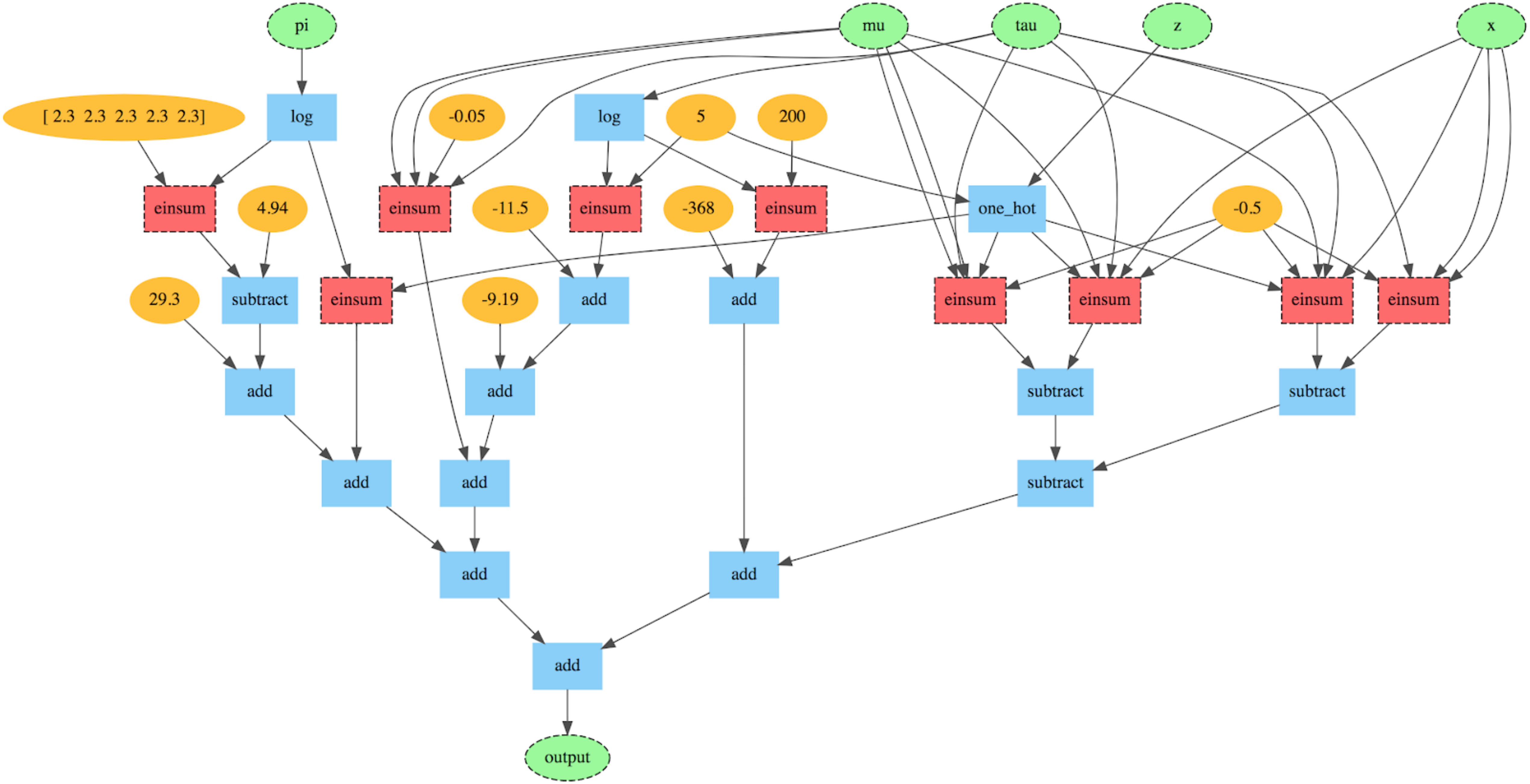
$$\text{where} \quad \mu_{m'} = \nabla \mathcal{A}_{m'}(\eta_{m'}) \quad \text{for} \quad m' = 1, \ldots, M$$

```
def meanfield(g, As, etas):
  def meanfield_sweep(mus):
    for m in range(M):
      mus[m] = grad(As[m])(grad(g, m)(*mus))
    return mus

  mus = [grad(As[m])(etas[m]) for m in range(M)]
  mu_stars = fixed_point(meanfield_sweep, mus)
  return [grad(g, m)(*mu_stars) for m in range(M)]
```

```python
def neg_energy(eta_prior, t_y, t_theta, t_z, t_x):
  t_z_node = markovchain.pair_to_node(t_z)
  t_z_trans = t_z[..., :-1, :, :]
  t_x_init = lds.pair_to_node(t_x[..., 0, :, :])
  t_x_trans = t_x[..., :-1, :, :]
  t_xy = gaussian.stats_product(lds.pair_to_node(t_x), t_y)
  return dot(eta_prior, t_theta) - logZ_theta(eta_prior)                  \
      + np.einsum('i,...i->',        t_theta[0], t_z_node[..., 0, :])       \
      + np.einsum('ij,...tij->',     t_theta[1], t_z_trans)                 \
      + np.einsum('kij,k,...ij->',   t_theta[2], t_z_node[..., 0, :], t_x_init)  \
      + np.einsum('kij,tk,...tij->', t_theta[3], t_z_node, t_x)             \
      + np.einsum('ij,...tij->',     t_theta[4], t_xy)
```

```python
def normal_logpdf(x, loc,  scale):
  prec = 1. / scale**2
  return -(np.sum(prec * mu**2) - np.sum(np.log(prec))
           + np.log(2. * np.pi)) * N / 2.

def normal_logpdf(pi, z, mu, tau, x):
  logp = (np.sum((alpha-1) * np.log(x)) - np.sum(gammaln(alpha))
          + np.sum(gammaln(np.sum(alpha, -1))))
  logp += normal_logpdf(mu, 0., 1./np.sqrt(kappa * tau))
  logp += np.sum(one_hot(z, K) * np.log(pi))
  logp += (a-1)*np.log(tau) - b*tau + a*np.log(b) - gammaln(a)
  mu_z = np.dot(one_hot(z, K), mu)
  loglike = normal_logpdf(x, mu_z, 1./np.sqrt(tau))
  return logp + loglike
```


pi
z
x
mu
tau
log
[2.3 2.3 2.3 2.3 2.3]
5
sqrt
0.1
multiply
log
one_hot
one_hot
multiply
sum
4.94
multiply
dot
sqrt
1
subtract
29.3
true_divide
2
true_divide
add
sum
0
power
power
add
subtract
subtract
true_divide
true_divide
multiply
multiply
log
log
multiply
multiply
sum
sum
200
sum
-0.5
sum
multiply
-9.19
multiply
-368
multiply
multiply
add
add
add
add
add
add
output



pi
mu
tau
z
x
[2.3 2.3 2.3 2.3 2.3]
log
-0.05
log
5
200
einsum
4.94
einsum
-11.5
einsum
-368
einsum
one_hot
-0.5
29.3
subtract
einsum
-9.19
add
add
einsum
einsum
einsum
einsum
add
add
subtract
subtract
add
add
subtract
add
add
add
output

Domain-specific term graph rewriting implementation

- **Tracer** using Autograd's API to map Python to term graphs
- **Pattern matcher** to do pattern-directed invocation
    - Python-embedded pattern language
    - Compiled into **continuation-passing matcher combinators** (~300 loc)

```
pat = (Einsum, Str('formula'), Segment('args1'),
       (Choice(Subtract('op'), Add('op')), Val('x'), Val('y')), Segment('args2'))
```

- **Rewriters** are syntactic graph macros using tracing to get **quasi-quasiquotes**

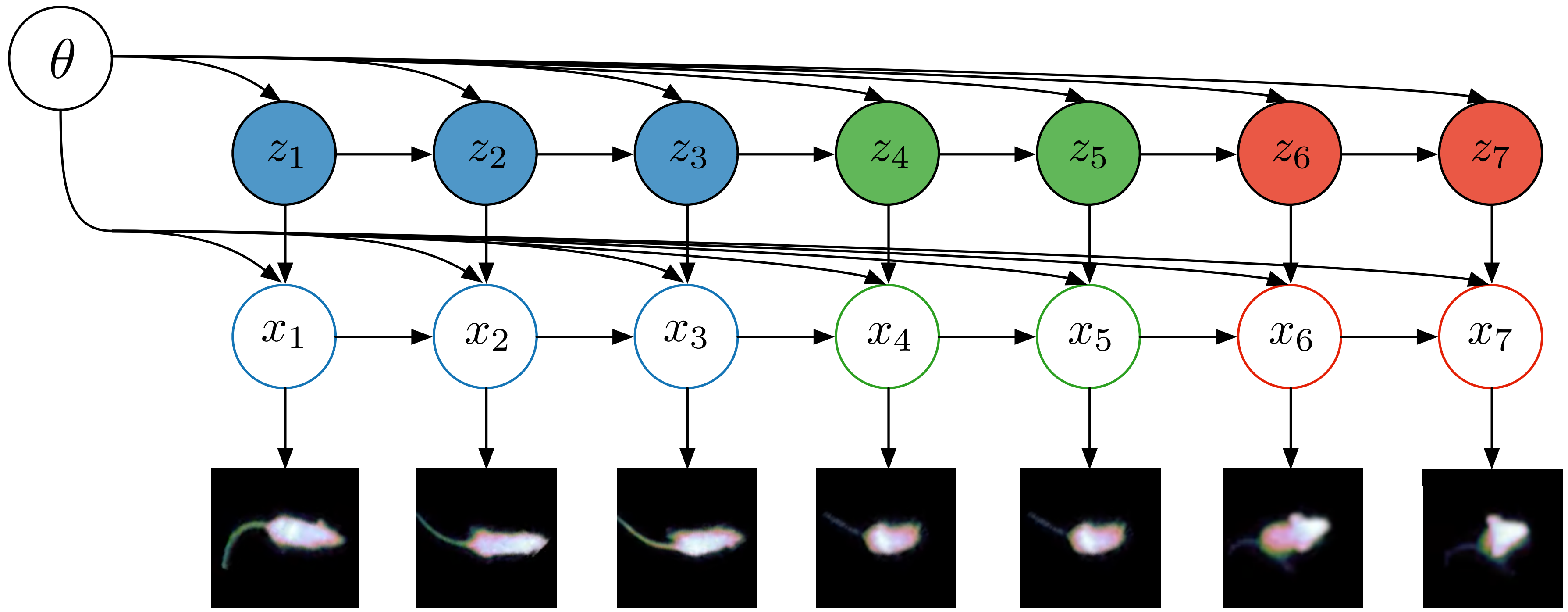
```
def rewriter(formula, op, x, y, args1, args2):
  return op(np.einsum(formula, *(args1 + (x,) + args2)),
            np.einsum(formula, *(args1 + (y,) + args2)))

distribute_einsum = Rule(pat, rewriter) # Rule is a namedtuple
```

Goals

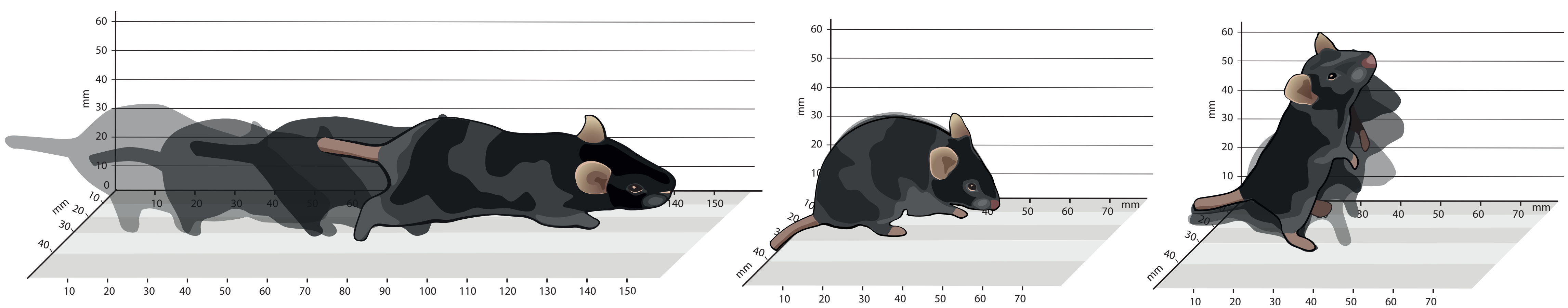
1. Motivate why PGMs + DNNs are a revolution waiting to happen
2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory
3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation
4. Make SVAEs and related PGM + DNN architectures super obvious so that you can invent better ones

Goals

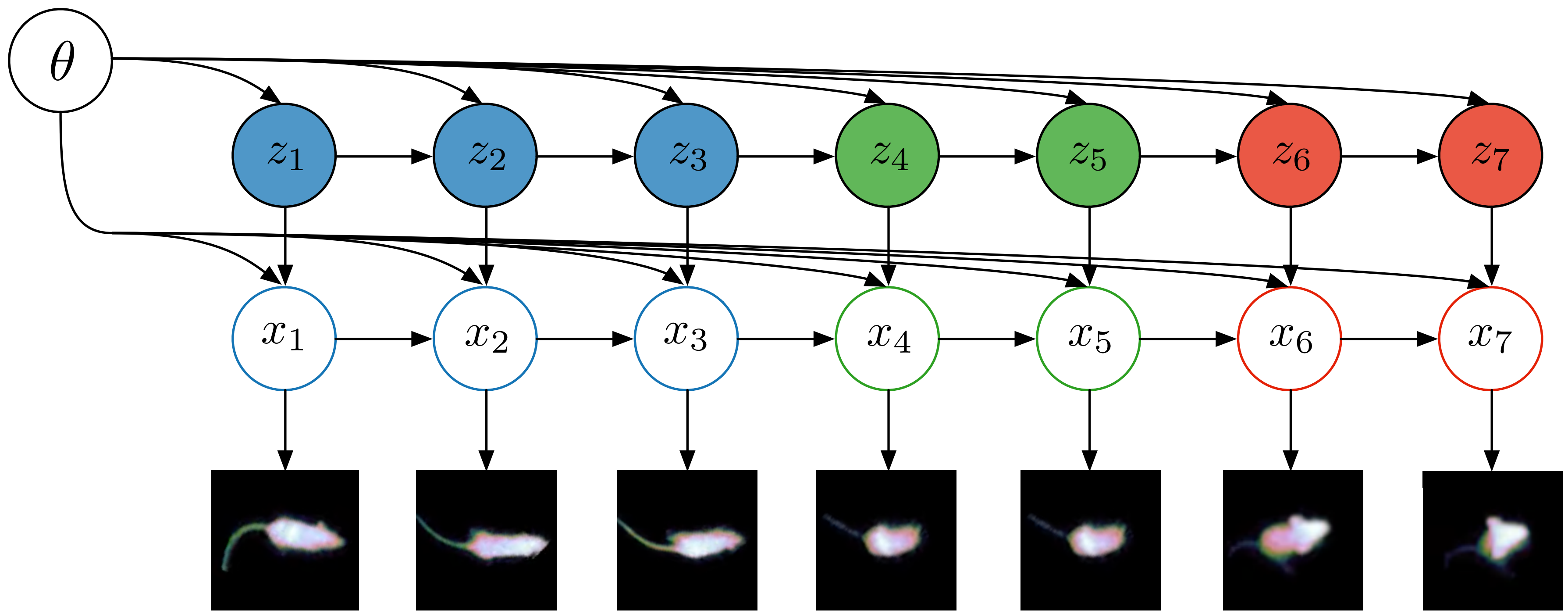
1. Motivate why PGMs + DNNs are a revolution waiting to happen
2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory
3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation
4. Make SVAEs and related PGM + DNN architectures super obvious so that you can invent better ones



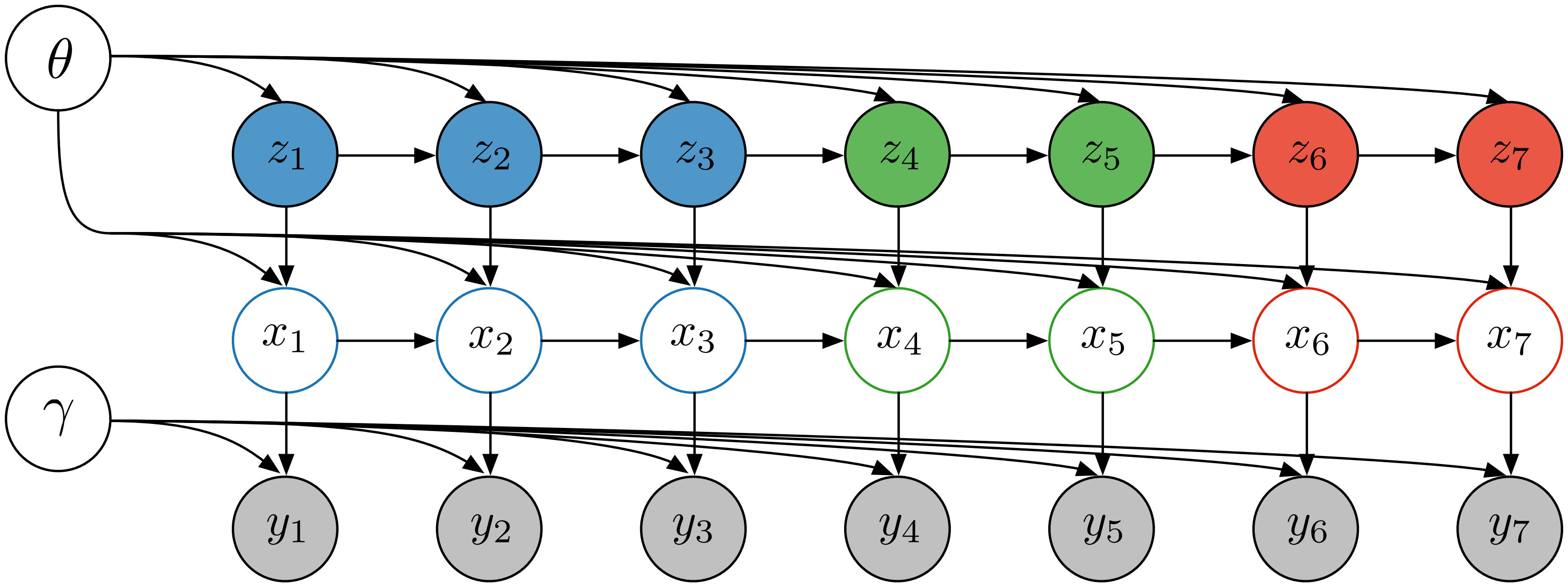
θ
z_1
z_2
z_3
z_4
z_5
z_6
z_7
x_1
x_2
x_3
x_4
x_5
x_6
x_7



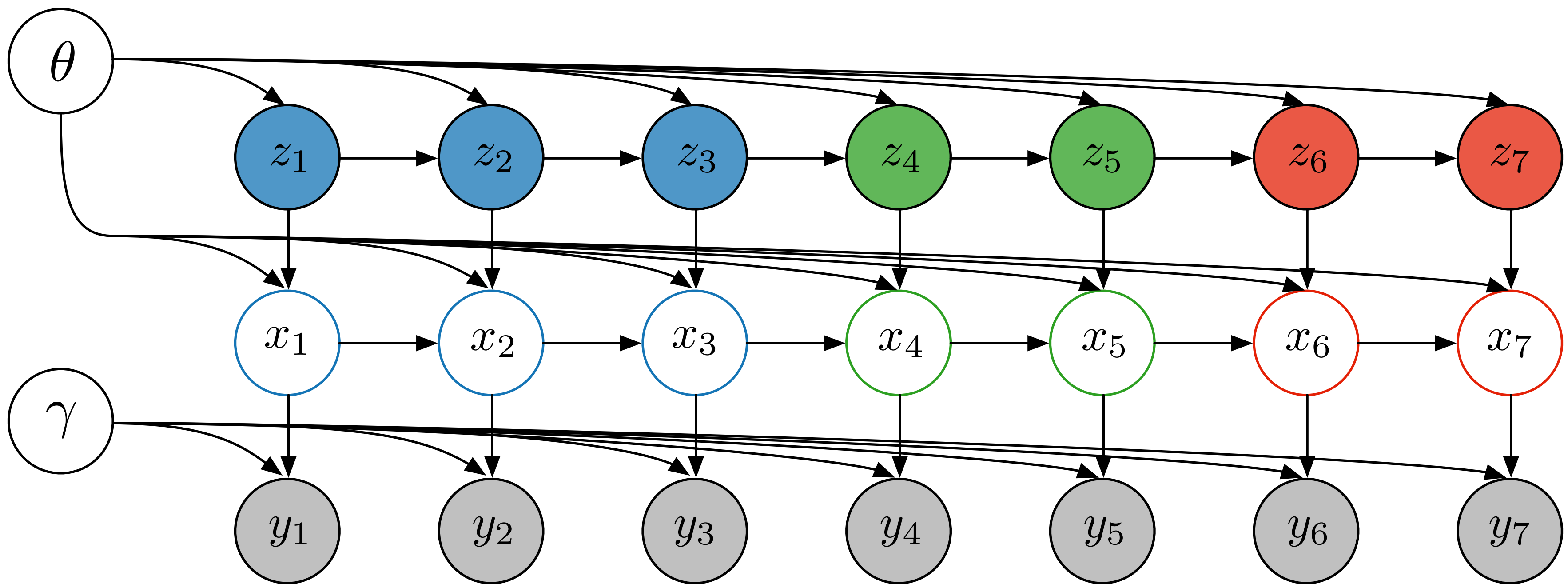
mm



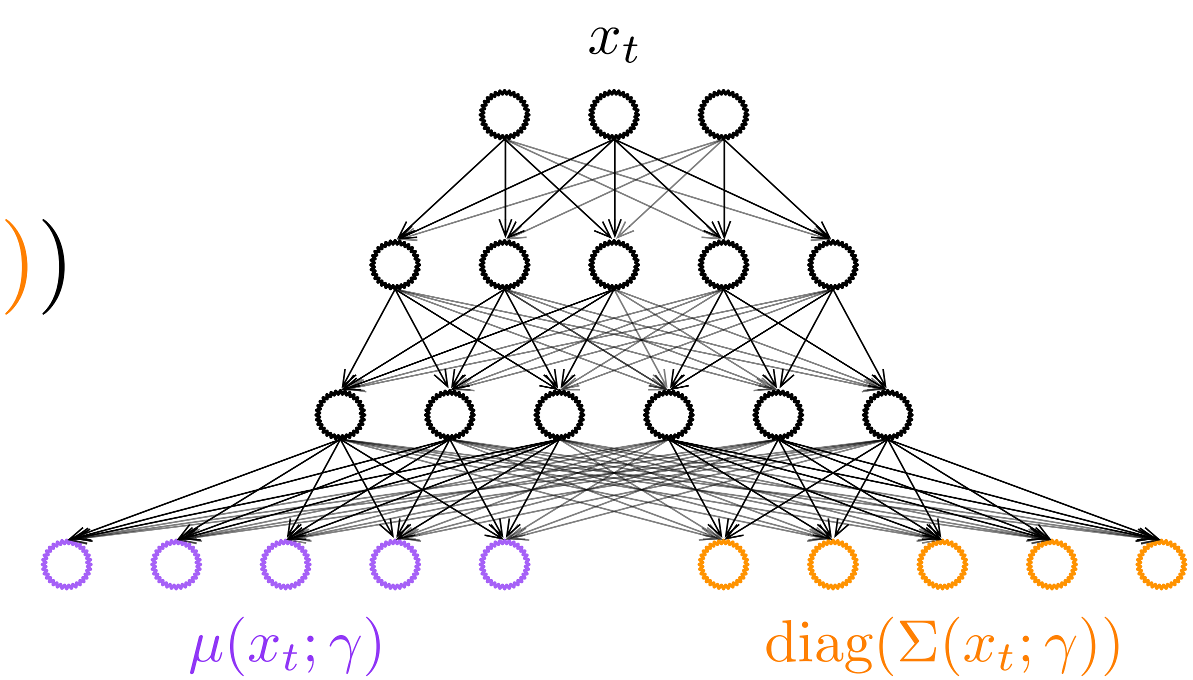
θ
z_1
z_2
z_3
z_4
z_5
z_6
z_7
x_1
x_2
x_3
x_4
x_5
x_6
x_7

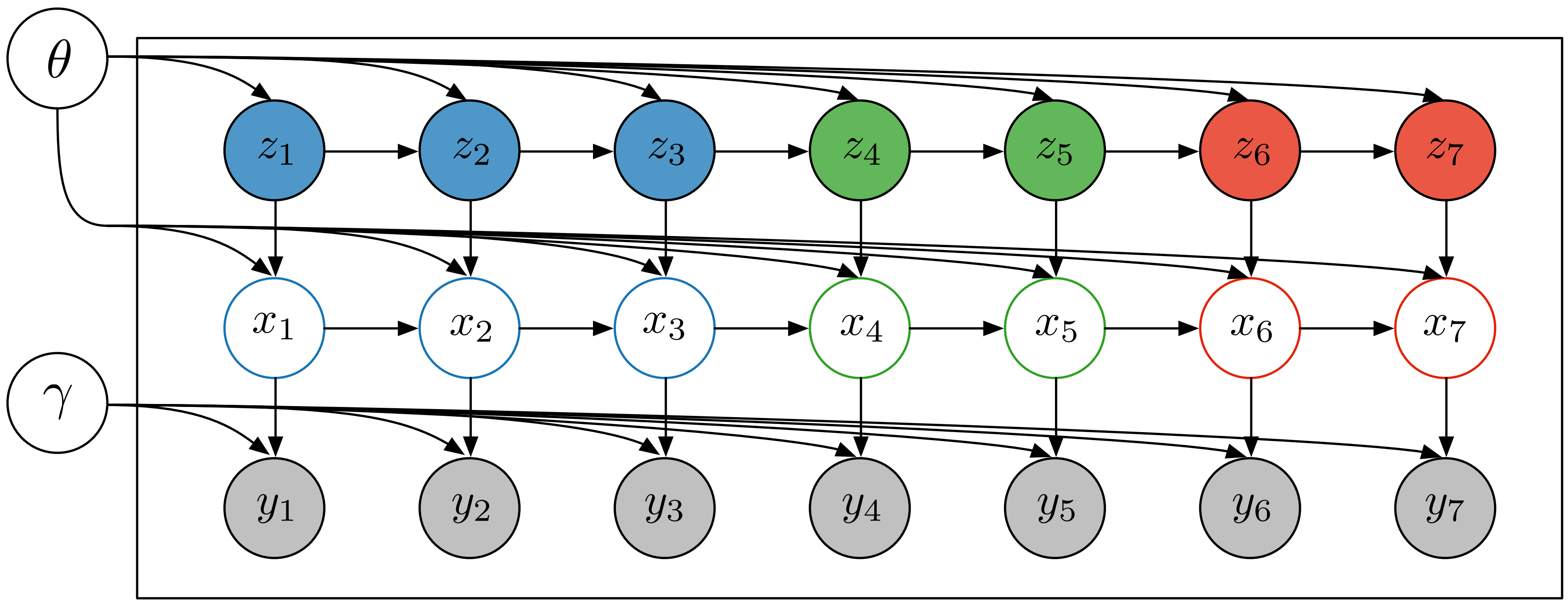


θ
z_1
z_2
z_3
z_4
z_5
z_6
z_7
x_1
x_2
x_3
x_4
x_5
x_6
x_7
γ
y_1
y_2
y_3
y_4
y_5
y_6
y_7

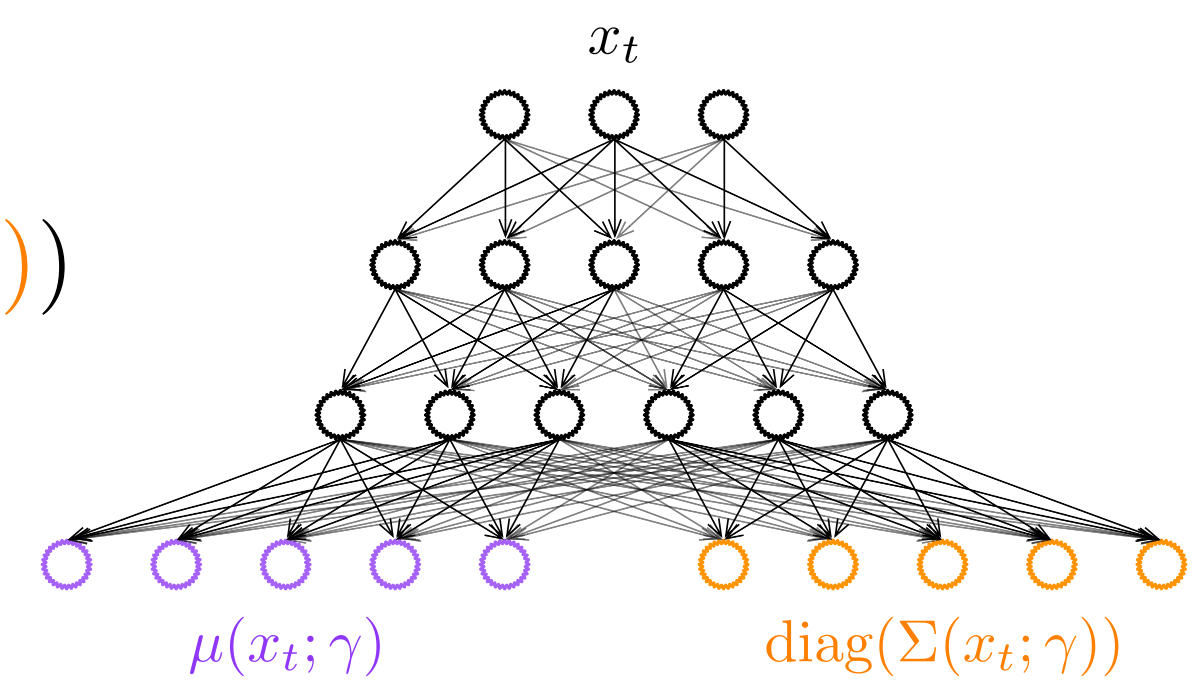


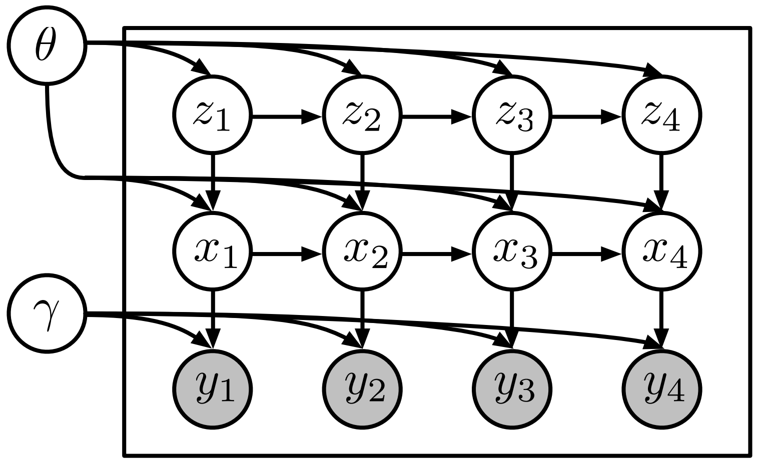
$$y_t \mid x_t, \gamma \;\sim\; \mathcal{N}(\mu(x_t;\gamma), \Sigma(x_t;\gamma))$$



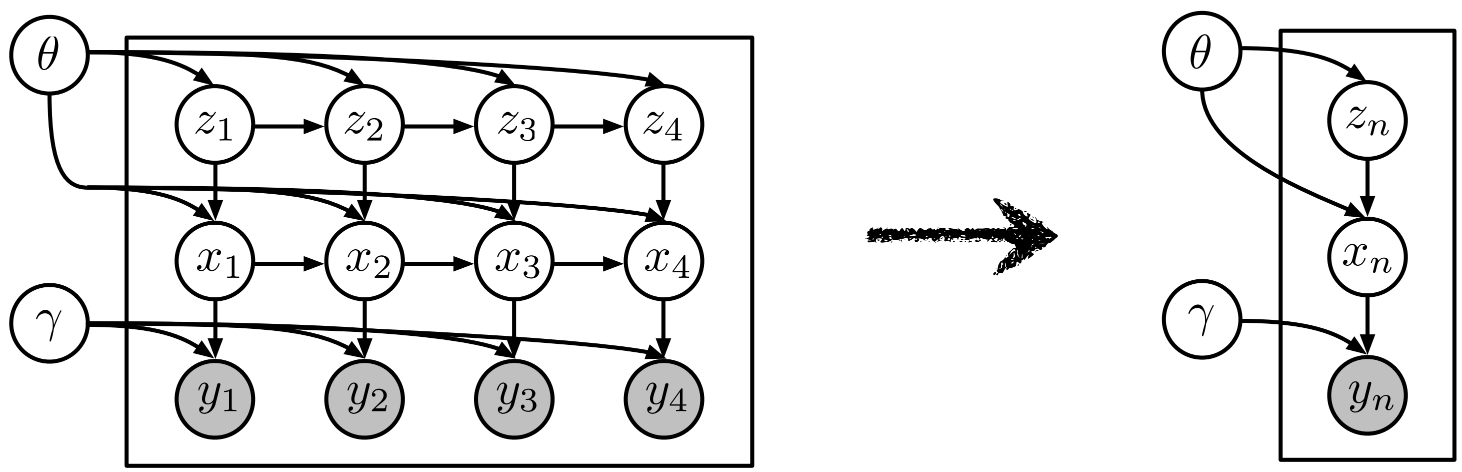


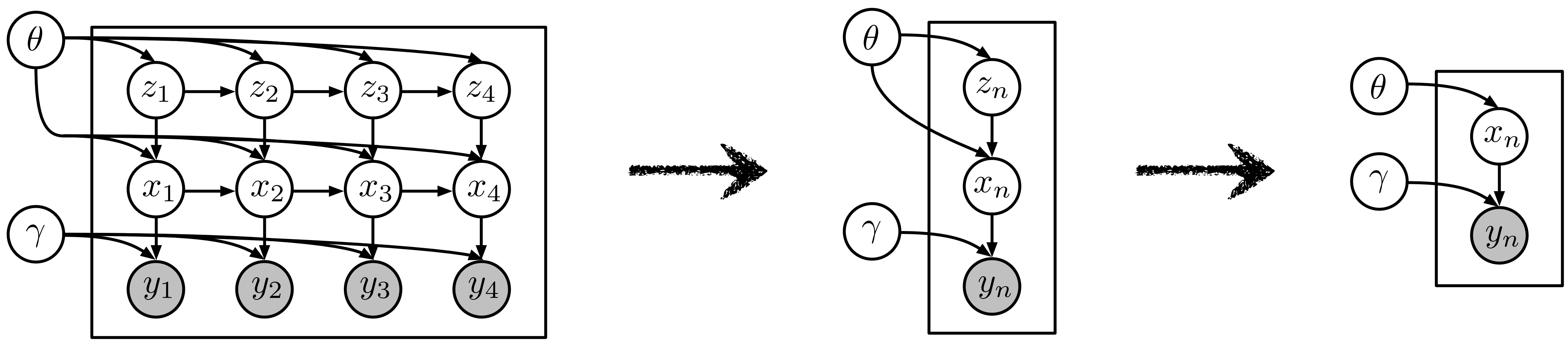
$$y_t \mid x_t, \gamma \ \sim \ \mathcal{N}(\mu(x_t;\gamma), \Sigma(x_t;\gamma))$$



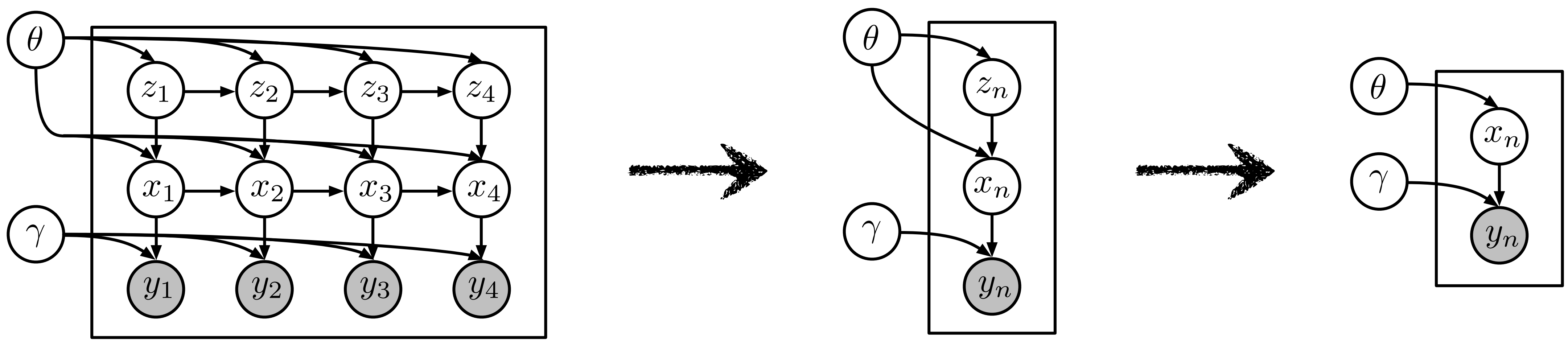


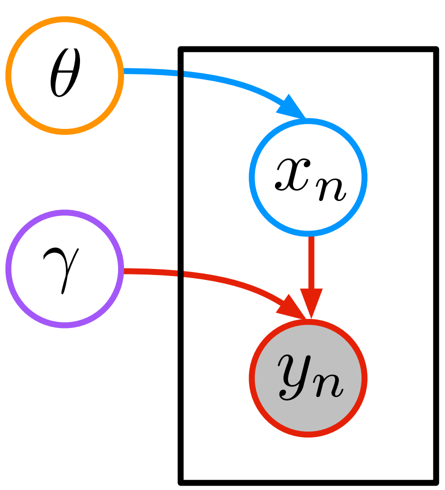
θ
z_1
z_2
z_3
z_4
x_1
x_2
x_3
x_4
γ
y_1
y_2
y_3
y_4

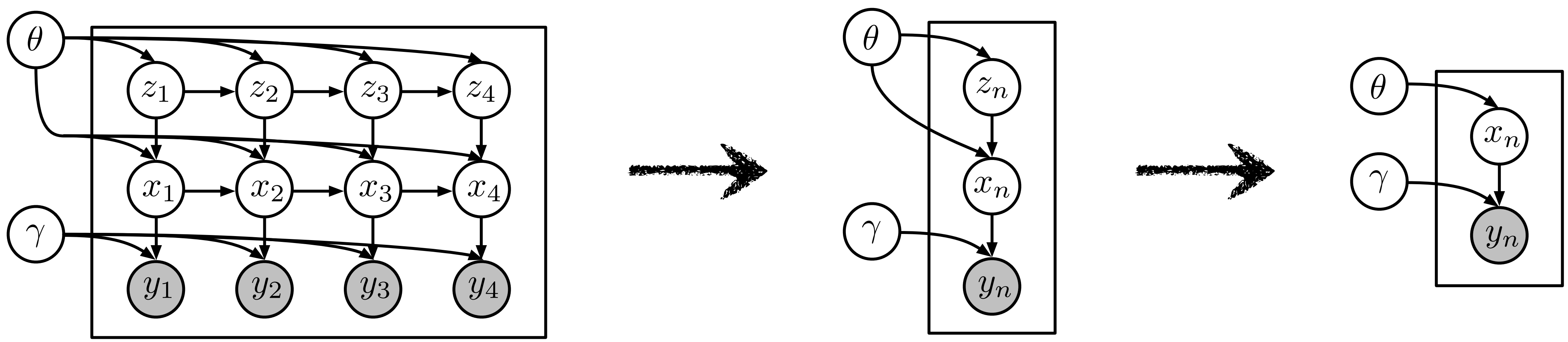

θ
z_1
z_2
z_3
z_4
x_1
x_2
x_3
x_4
γ
y_1
y_2
y_3
y_4
θ
z_n
x_n
γ
y_n

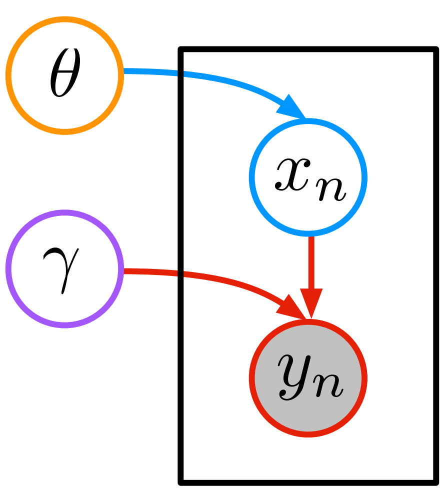


θ
γ
z_1 z_2 z_3 z_4
x_1 x_2 x_3 x_4
y_1 y_2 y_3 y_4
θ
γ
z_n
x_n
y_n
θ
γ
x_n
y_n


θ
γ
z1 z2 z3 z4
x1 x2 x3 x4
y1 y2 y3 y4
zn
xn
yn

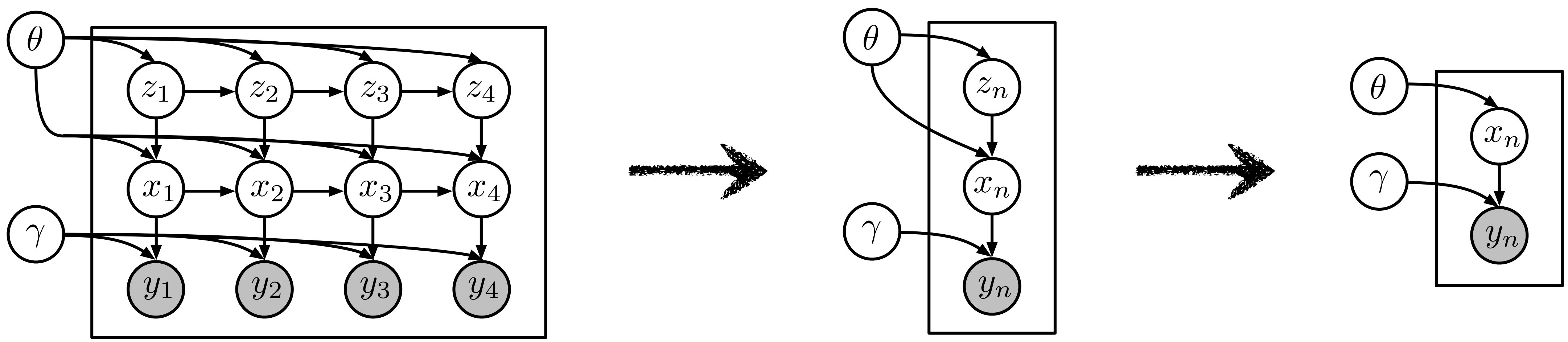

θ
γ
xn
yn

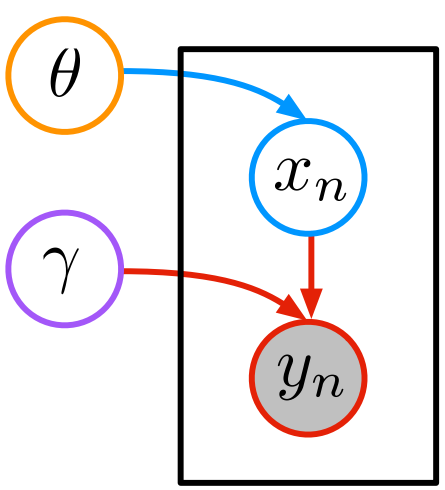




$p(\theta)$ conjugate prior on global variables

$p(x \mid \theta)$ exponential family on local variables

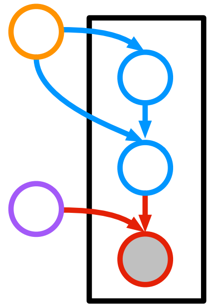




$p(\theta)$ conjugate prior on global variables

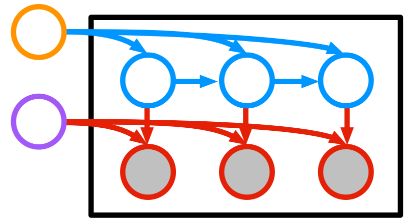
$p(x \mid \theta)$ exponential family on local variables

$p(y \mid x, \gamma)$ neural network observation model



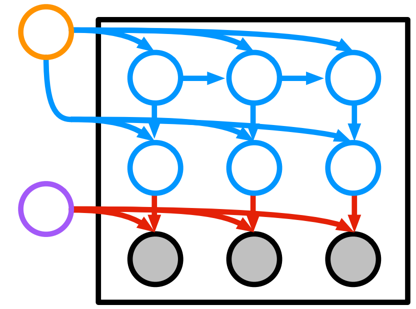
[1]

Gaussian mixture model



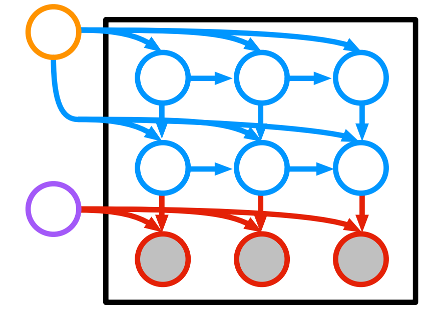
[2]

Linear dynamical system



[3]

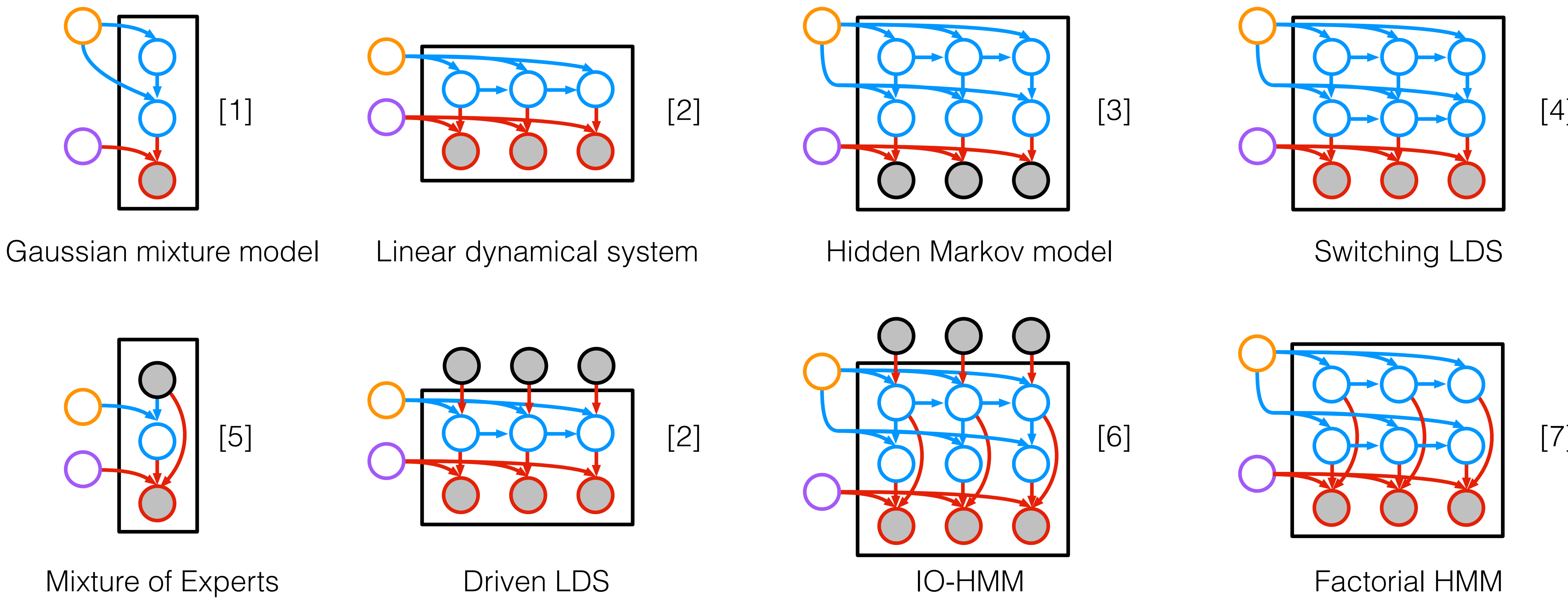
Hidden Markov model



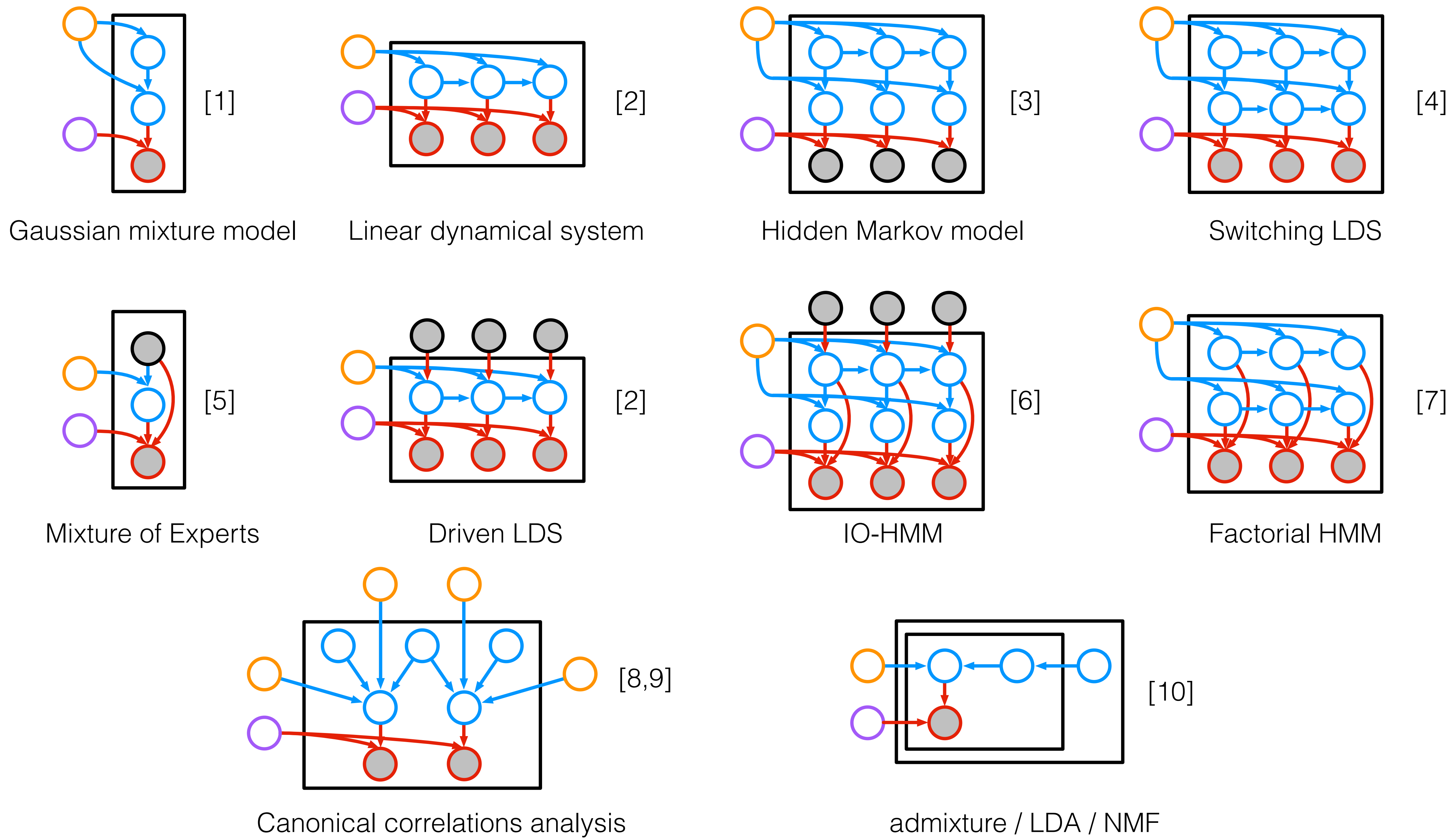
[4]

Switching LDS

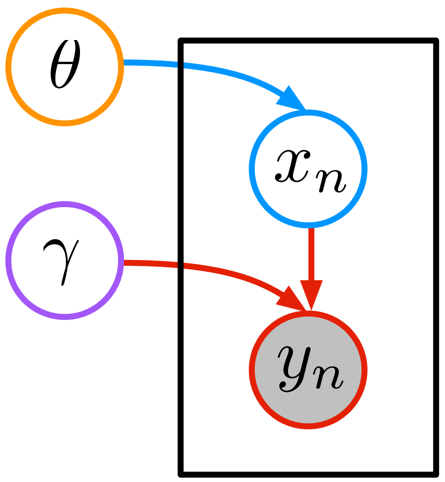
[1] Palmer, Wipf, Kreutz-Delgado, and Rao. Variational EM algorithms for non-Gaussian latent variable models. NIPS 2005.
[2] Ghahramani and Beal. Propagation algorithms for variational Bayesian learning. NIPS 2001.
[3] Beal. Variational algorithms for approximate Bayesian inference, Ch. 3. U of London Ph.D. Thesis 2003.
[4] Ghahramani and Hinton. Variational learning for switching state-space models. Neural Computation 2000.



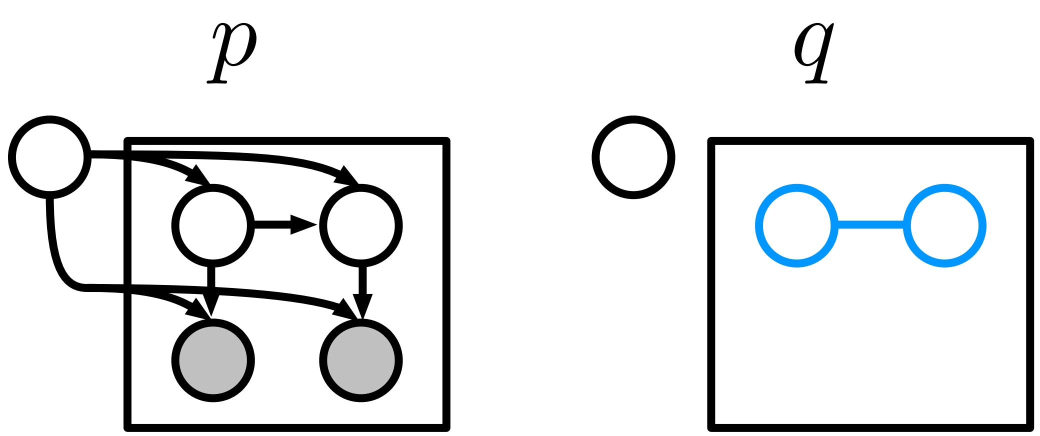
[1] Palmer, Wipf, Kreutz-Delgado, and Rao. Variational EM algorithms for non-Gaussian latent variable models. NIPS 2005.
[2] Ghahramani and Beal. Propagation algorithms for variational Bayesian learning. NIPS 2001.
[3] Beal. Variational algorithms for approximate Bayesian inference, Ch. 3. U of London Ph.D. Thesis 2003.
[4] Ghahramani and Hinton. Variational learning for switching state-space models. Neural Computation 2000.
[5] Jordan and Jacobs. Hierarchical Mixtures of Experts and the EM algorithm. Neural Computation 1994.
[6] Bengio and Frasconi. An Input Output HMM Architecture. NIPS 1995.
[7] Ghahramani and Jordan. Factorial Hidden Markov Models. Machine Learning 1997.



[1] Palmer, Wipf, Kreutz-Delgado, and Rao. Variational EM algorithms for non-Gaussian latent variable models. NIPS 2005.
[2] Ghahramani and Beal. Propagation algorithms for variational Bayesian learning. NIPS 2001.
[3] Beal. Variational algorithms for approximate Bayesian inference, Ch. 3. U of London Ph.D. Thesis 2003.
[4] Ghahramani and Hinton. Variational learning for switching state-space models. Neural Computation 2000.
[5] Jordan and Jacobs. Hierarchical Mixtures of Experts and the EM algorithm. Neural Computation 1994.
[6] Bengio and Frasconi. An Input Output HMM Architecture. NIPS 1995.
[7] Ghahramani and Jordan. Factorial Hidden Markov Models. Machine Learning 1997.
[8] Bach and Jordan. A probabilistic interpretation of Canonical Correlation Analysis. Tech. Report 2005.
[9] Archambeau and Bach. Sparse probabilistic projections. NIPS 2008.
[10] Hoffman, Bach, Blei. Online learning for Latent Dirichlet Allocation. NIPS 2010.



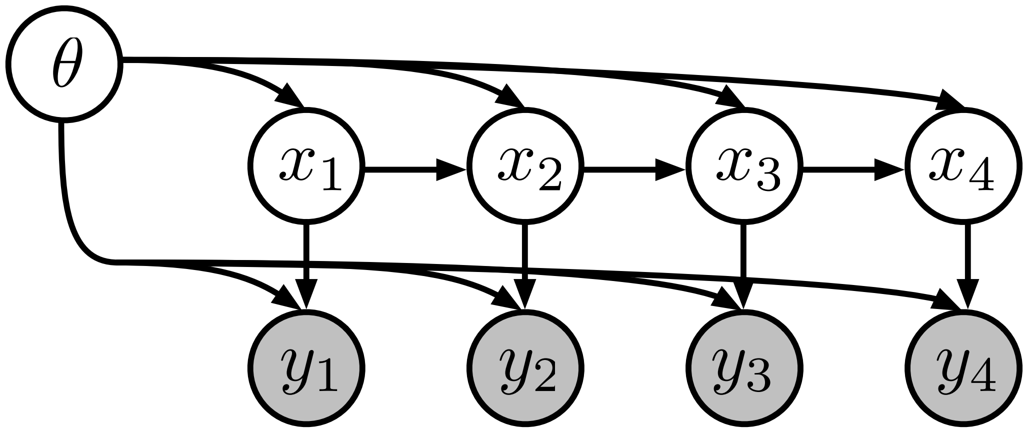
Inference?



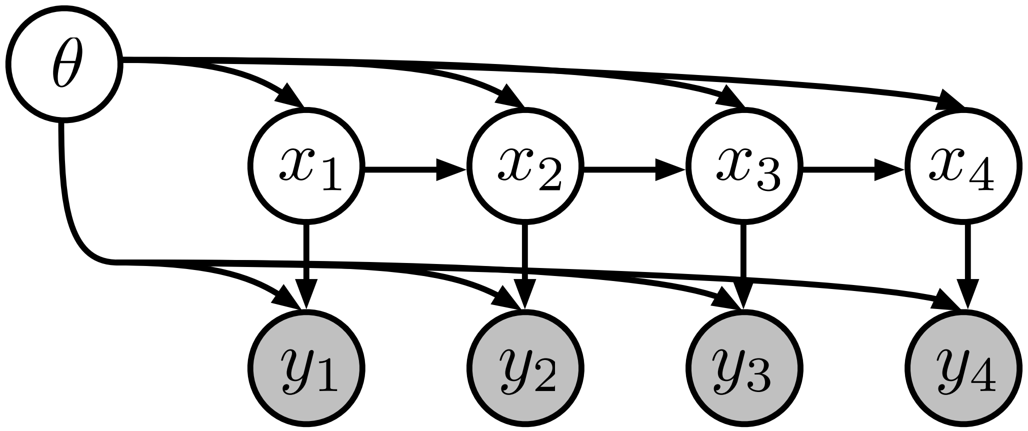
$$q^*(x) \triangleq \underset{q(x)}{\arg\max}\, \mathcal{L}[\, q(\theta)q(x)\,]$$

Natural gradient SVI
for nice exp. fam. PGMs [1,2]

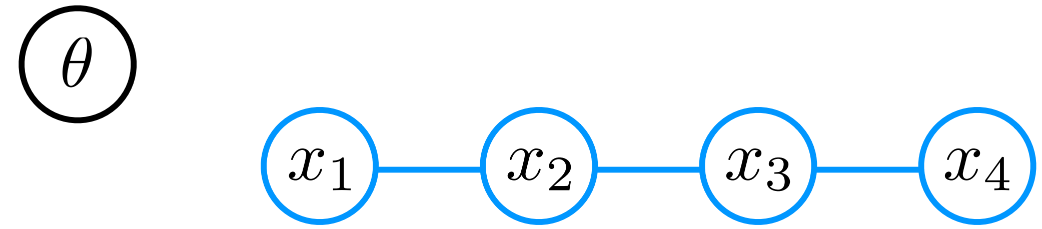
[1] Hoffman, Bach, Blei. Online learning for Latent Dirichlet Allocation. NIPS 2010.
[2] Hoffman, Blei, Wang, and Paisley. Stochastic variational inference. JMLR 2013.



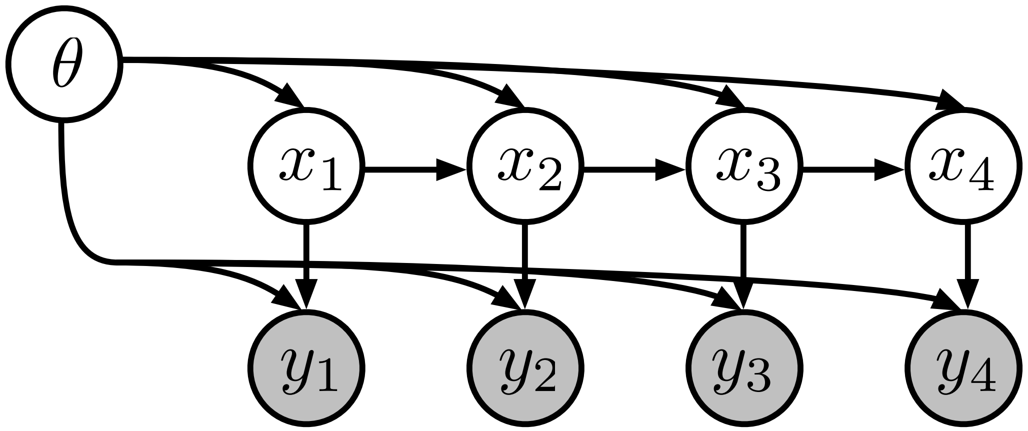
$p(x \mid \theta)$ is a linear dynamical system
$p(y \mid x, \theta)$ is a linear-Gaussian observation
$p(\theta)$ is a conjugate prior

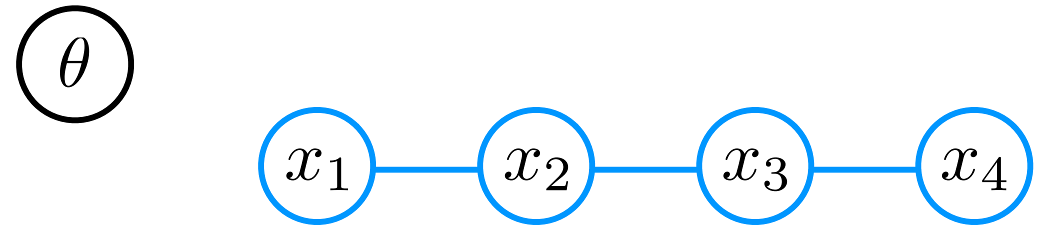


$p(x \mid \theta)$ is a linear dynamical system
$p(y \mid x, \theta)$ is a linear-Gaussian observation
$p(\theta)$ is a conjugate prior



$$q(\theta)q(x) \approx p(\theta, x \mid y)$$

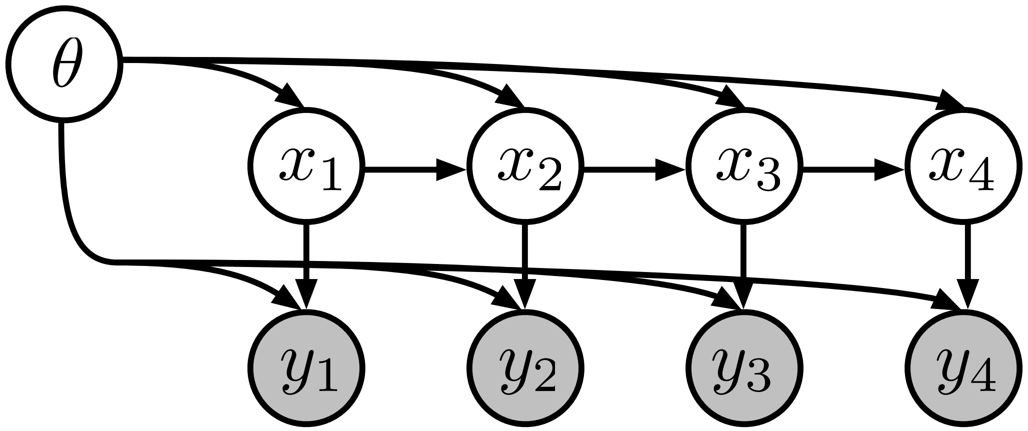


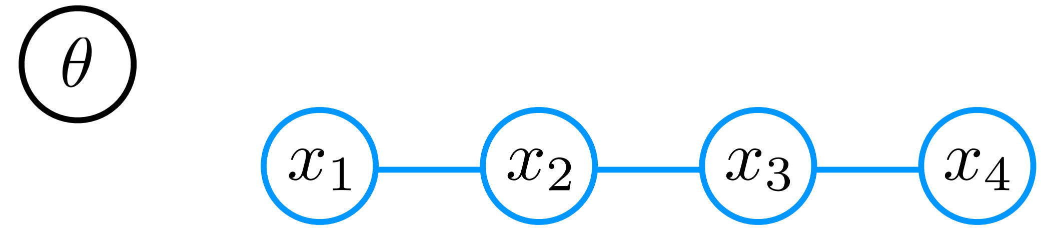


$p(x \mid \theta)$ is a linear dynamical system
$p(y \mid x, \theta)$ is a linear-Gaussian observation
$p(\theta)$ is a conjugate prior

$$q(\theta)q(x) \approx p(\theta, x \mid y)$$

$$\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)}\left[\log \frac{p(\theta, x, y)}{q(\theta)q(x)}\right]$$



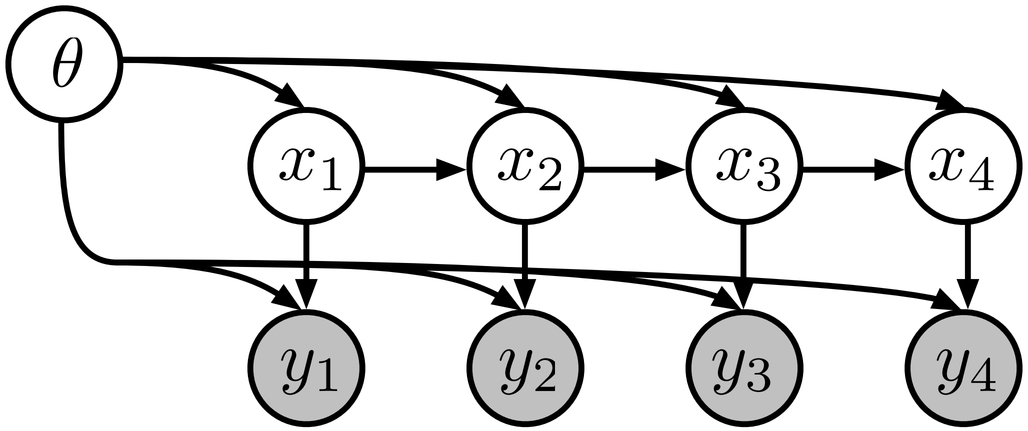


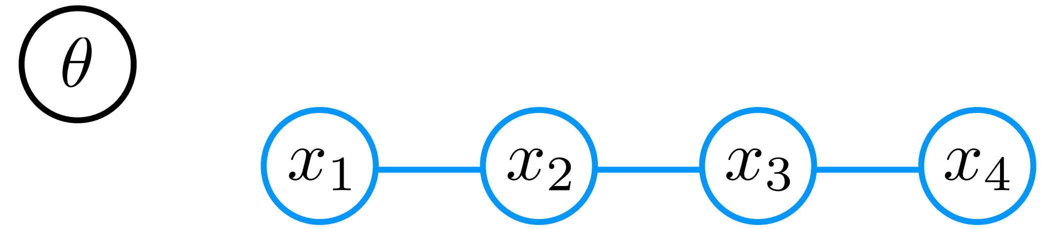
$p(x \mid \theta)$ is a linear dynamical system
$p(y \mid x, \theta)$ is a linear-Gaussian observation
$p(\theta)$ is a conjugate prior

$$q(\theta) q(x) \approx p(\theta, x \mid y)$$

$$\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta) q(x)} \left[\log \frac{p(\theta, x, y)}{q(\theta) q(x)} \right]$$

$$\eta_x^*(\eta_\theta) \triangleq \arg\max_{\eta_x} \mathcal{L}(\eta_\theta, \eta_x) \qquad \mathcal{L}_{\mathrm{SVI}}(\eta_\theta) \triangleq \mathcal{L}(\eta_\theta, \eta_x^*(\eta_\theta))$$





$p(x \mid \theta)$ is a linear dynamical system
$p(y \mid x, \theta)$ is a linear-Gaussian observation
$p(\theta)$ is a conjugate prior

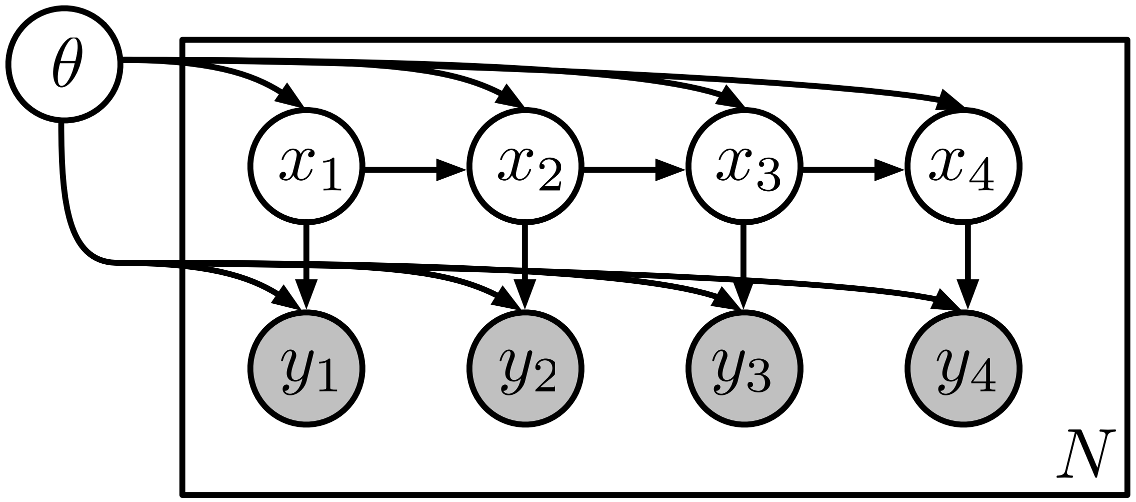
$$q(\theta)q(x) \approx p(\theta, x \mid y)$$

$$\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)}\left[\log \frac{p(\theta,x,y)}{q(\theta)q(x)}\right]$$

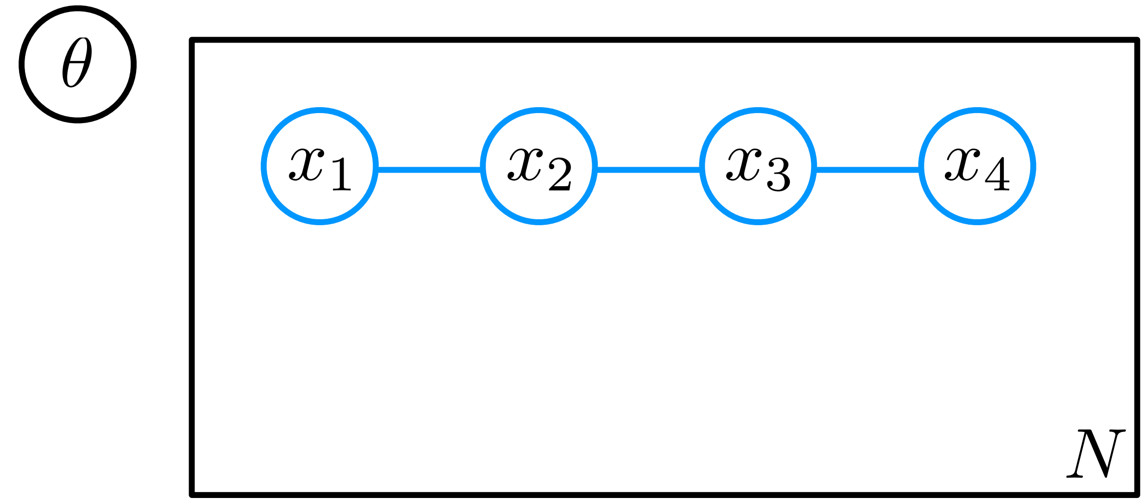
$$\eta_x^*(\eta_\theta) \triangleq \arg\max_{\eta_x} \mathcal{L}(\eta_\theta, \eta_x) \qquad \mathcal{L}_{\text{SVI}}(\eta_\theta) \triangleq \mathcal{L}(\eta_\theta, \eta_x^*(\eta_\theta))$$

Proposition (natural gradient SVI of Hoffman et al. 2013)

$$\widetilde{\nabla}\mathcal{L}_{\text{SVI}}(\eta_\theta) = \eta_\theta^0 + \mathbb{E}_{q^*(x)}(t_{xy}(x,y), 1) - \eta_\theta$$



$p(x \mid \theta)$ is a linear dynamical system
$p(y \mid x, \theta)$ is a linear-Gaussian observation
$p(\theta)$ is a conjugate prior



$$q(\theta)q(x) \approx p(\theta, x \mid y)$$

$$\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)}\left[\log \frac{p(\theta,x,y)}{q(\theta)q(x)}\right]$$

$$\eta_x^*(\eta_\theta) \triangleq \arg\max_{\eta_x} \mathcal{L}(\eta_\theta, \eta_x) \qquad \mathcal{L}_{\text{SVI}}(\eta_\theta) \triangleq \mathcal{L}(\eta_\theta, \eta_x^*(\eta_\theta))$$

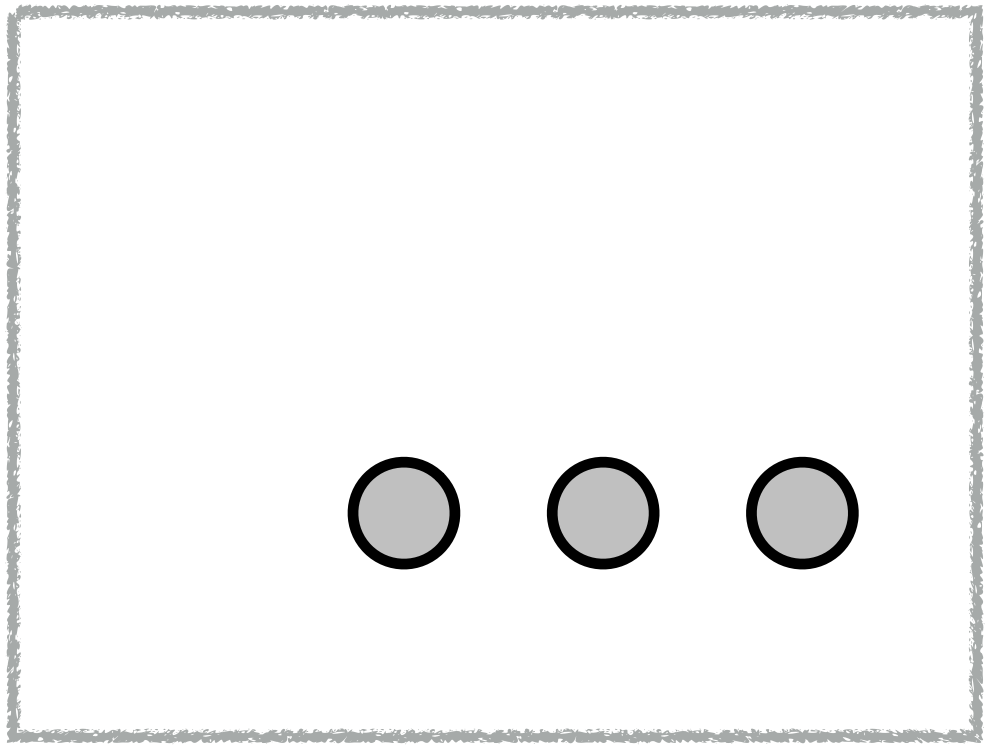
Proposition (natural gradient SVI of Hoffman et al. 2013)

$$\widetilde{\nabla}\mathcal{L}_{\text{SVI}}(\eta_\theta) = \eta_\theta^0 + \sum_{n=1}^{N} \mathbb{E}_{q^*(x_n)}(t_{xy}(x_n, y_n), 1) - \eta_\theta$$

Step 1: compute evidence potentials

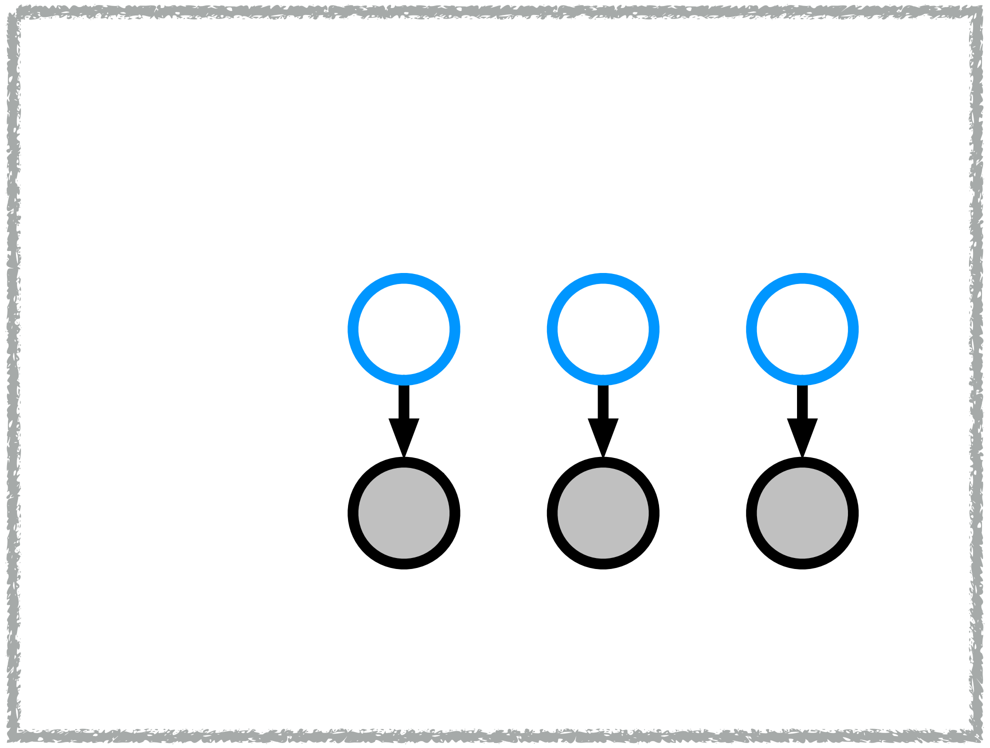
[1] **Johnson** and Willsky. Stochastic variational inference for Bayesian time series models. ICML 2014.
[2] Foti, Xu, Laird, and Fox. Stochastic variational inference for hidden Markov models. NIPS 2014.

Step 1: compute evidence potentials



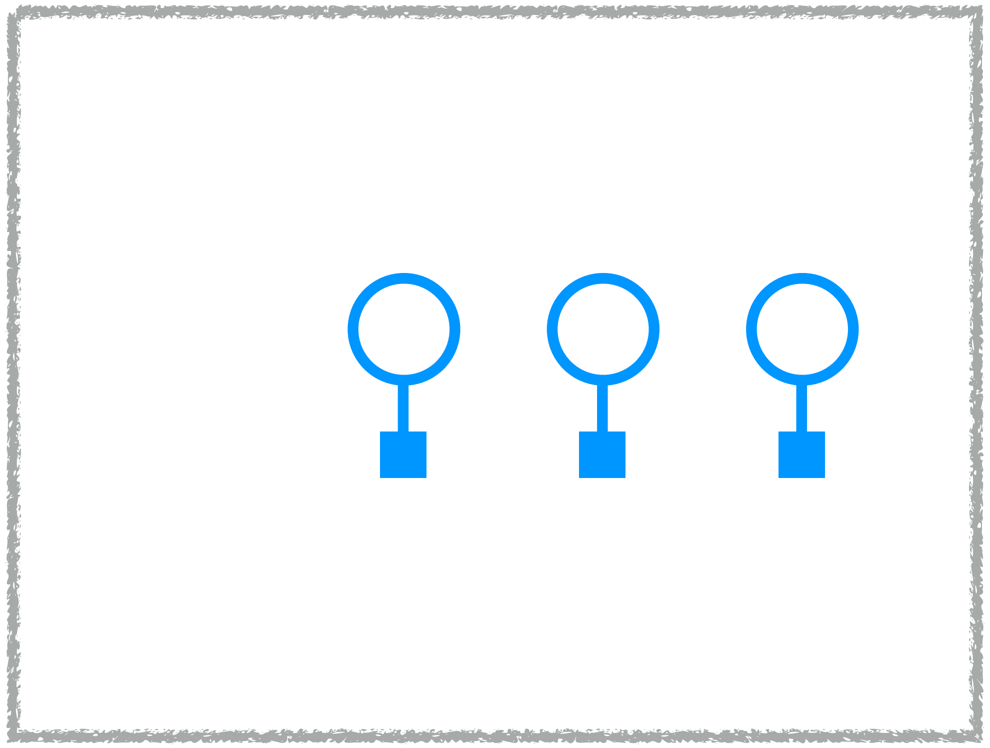
[1] **Johnson** and Willsky. Stochastic variational inference for Bayesian time series models. ICML 2014.
[2] Foti, Xu, Laird, and Fox. Stochastic variational inference for hidden Markov models. NIPS 2014.

Step 1: compute evidence potentials



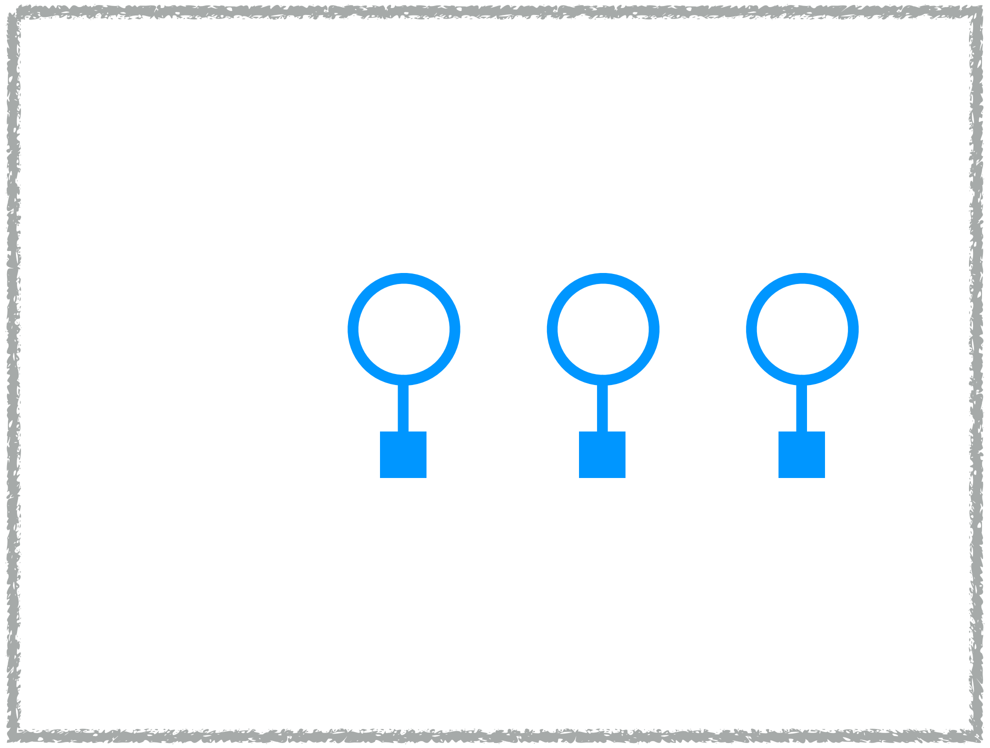
[1] **Johnson** and Willsky. Stochastic variational inference for Bayesian time series models. ICML 2014.
[2] Foti, Xu, Laird, and Fox. Stochastic variational inference for hidden Markov models. NIPS 2014.

Step 1: compute evidence potentials



[1] **Johnson** and Willsky. Stochastic variational inference for Bayesian time series models. ICML 2014.
[2] Foti, Xu, Laird, and Fox. Stochastic variational inference for hidden Markov models. NIPS 2014.

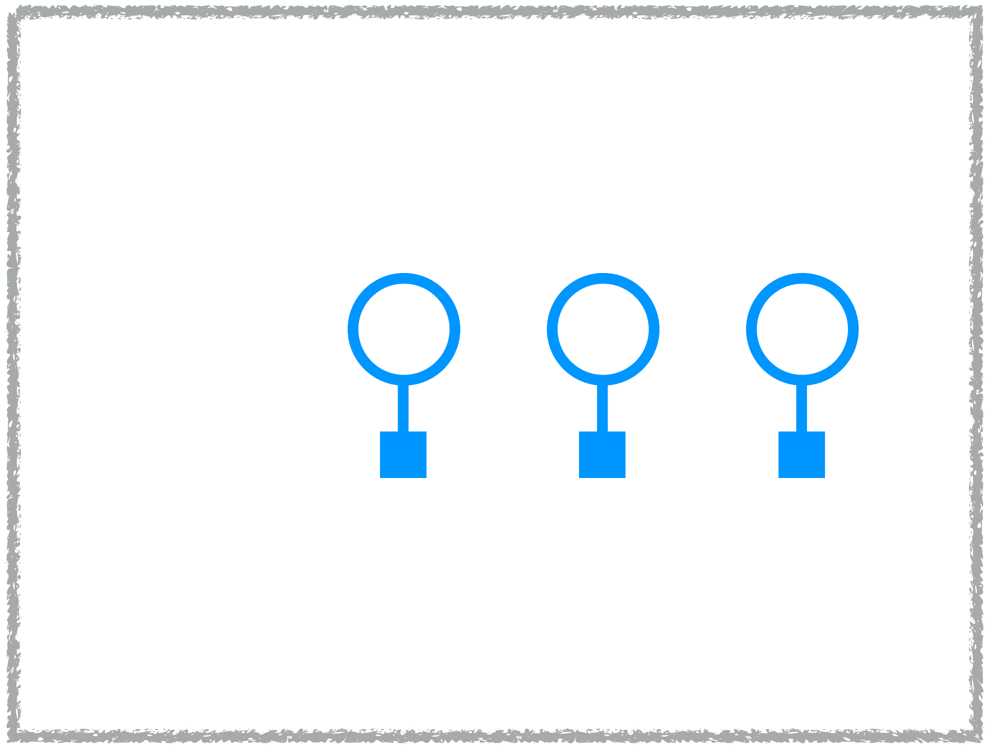
Step 1: compute evidence potentials



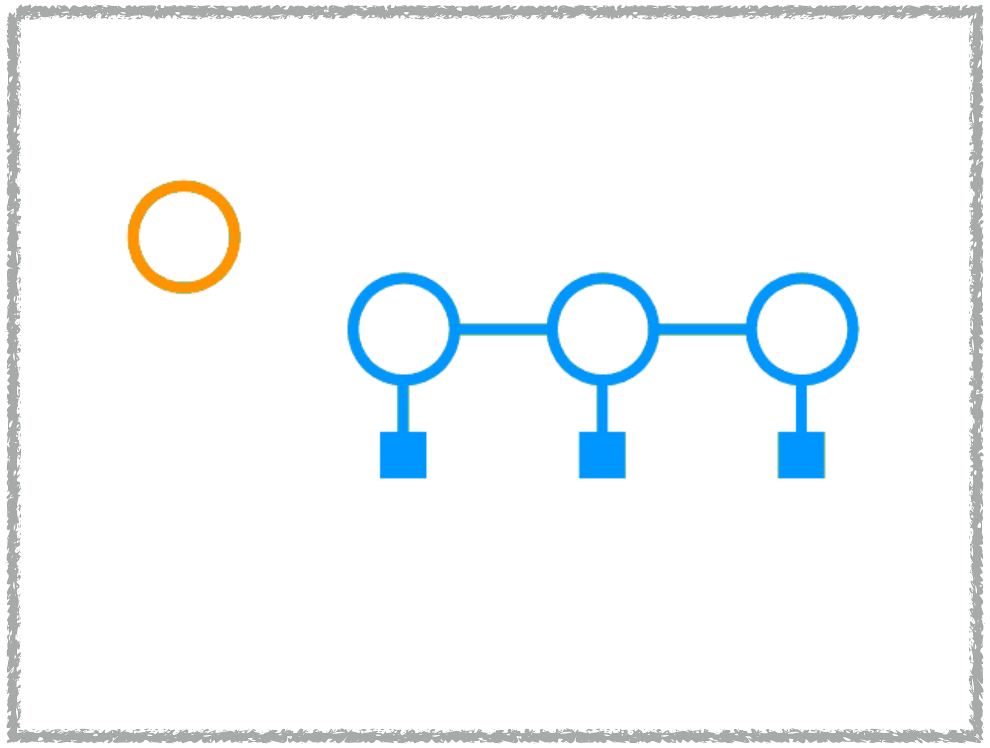
Step 2: run fast message passing

[1] **Johnson** and Willsky. Stochastic variational inference for Bayesian time series models. ICML 2014.
[2] Foti, Xu, Laird, and Fox. Stochastic variational inference for hidden Markov models. NIPS 2014.

Step 1: compute evidence potentials

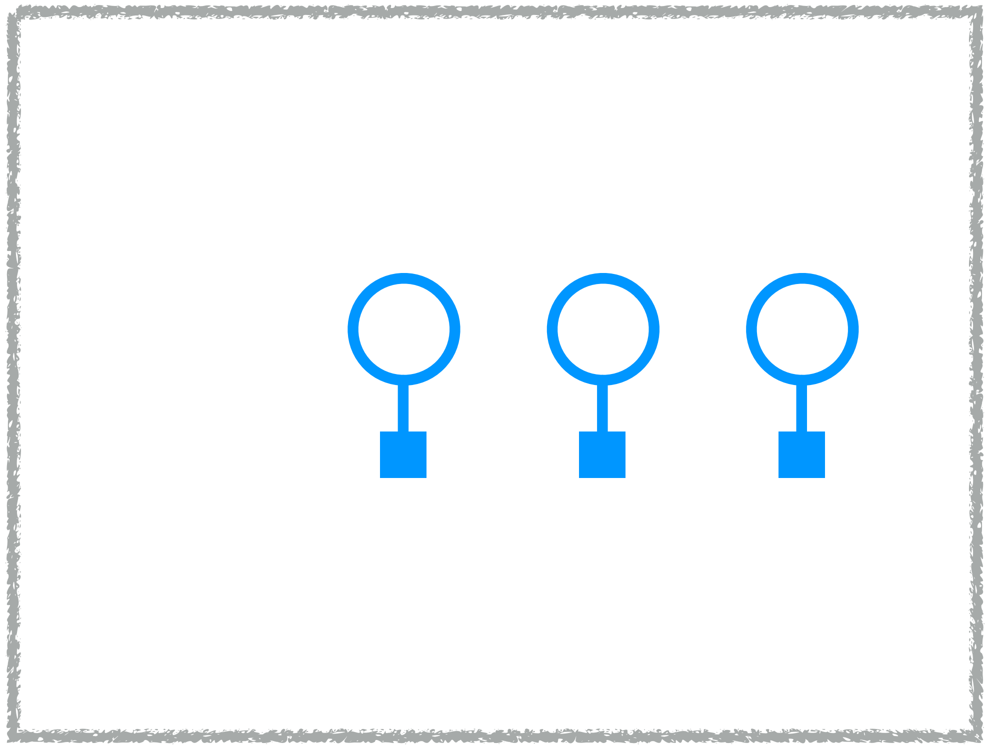


Step 2: run fast message passing

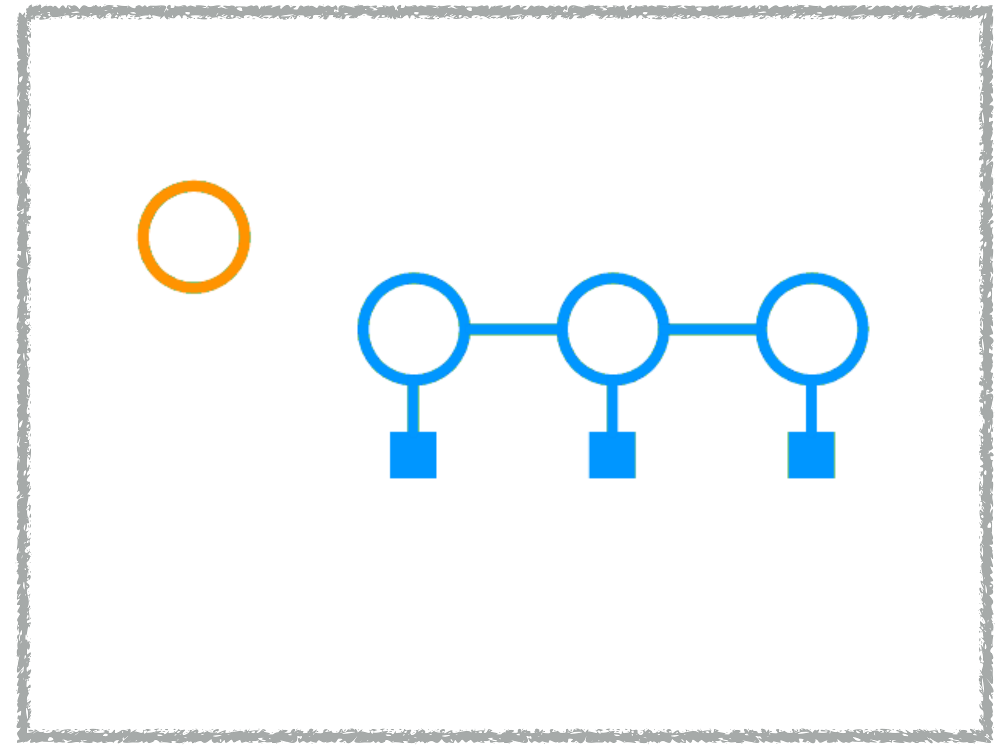


[1] **Johnson** and Willsky. Stochastic variational inference for Bayesian time series models. ICML 2014.
[2] Foti, Xu, Laird, and Fox. Stochastic variational inference for hidden Markov models. NIPS 2014.

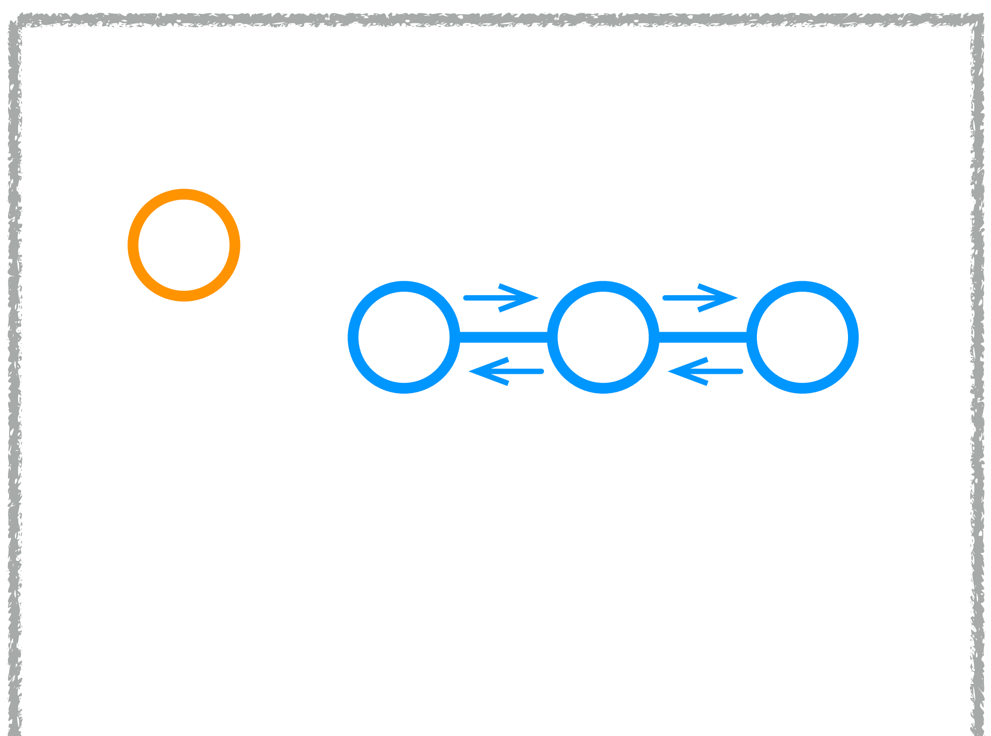
Step 1: compute evidence potentials



Step 2: run fast message passing

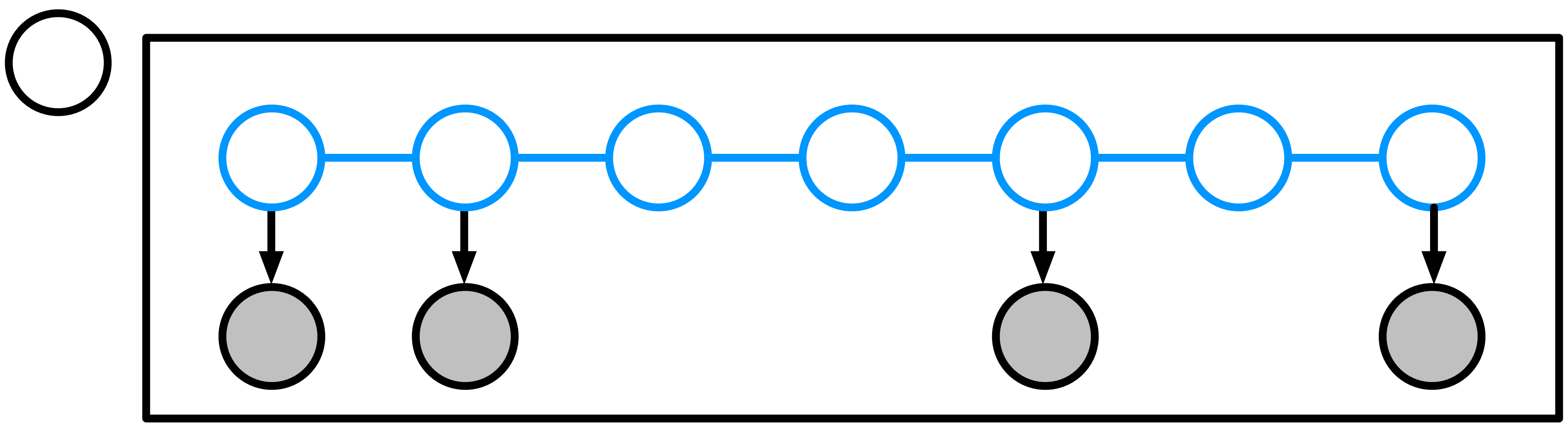


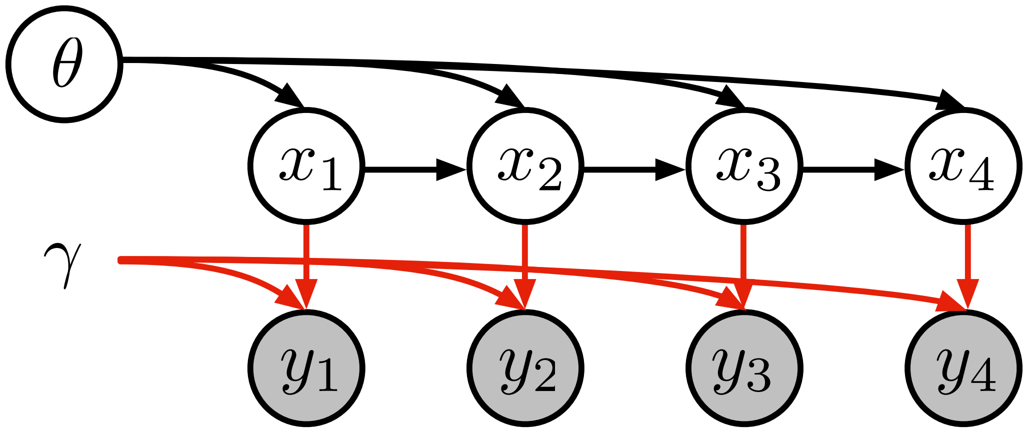
Step 3: compute natural gradient



[1] **Johnson** and Willsky. Stochastic variational inference for Bayesian time series models. ICML 2014.
[2] Foti, Xu, Laird, and Fox. Stochastic variational inference for hidden Markov models. NIPS 2014.

arbitrary inference queries

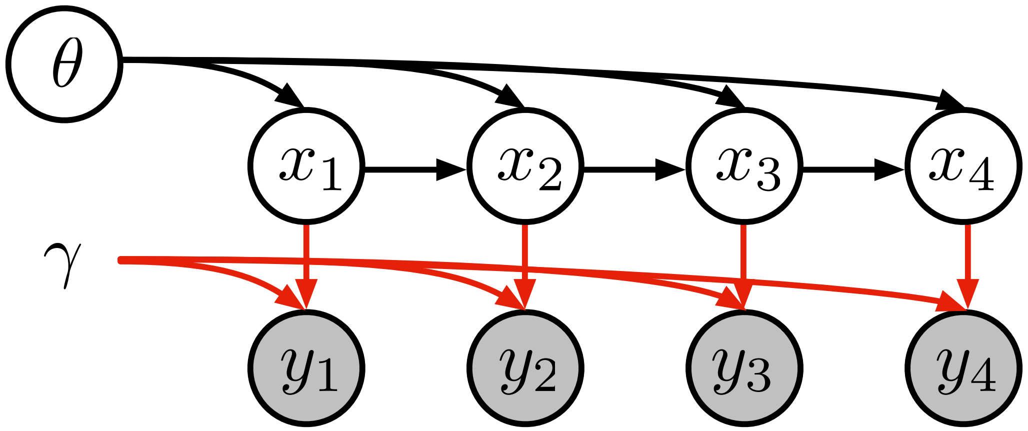


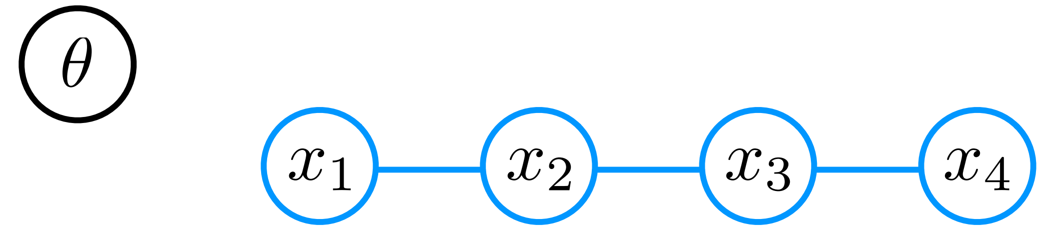


$p(x \mid \theta)$ is a linear dynamical system

$p(y \mid x, \gamma)$ is a neural network decoder

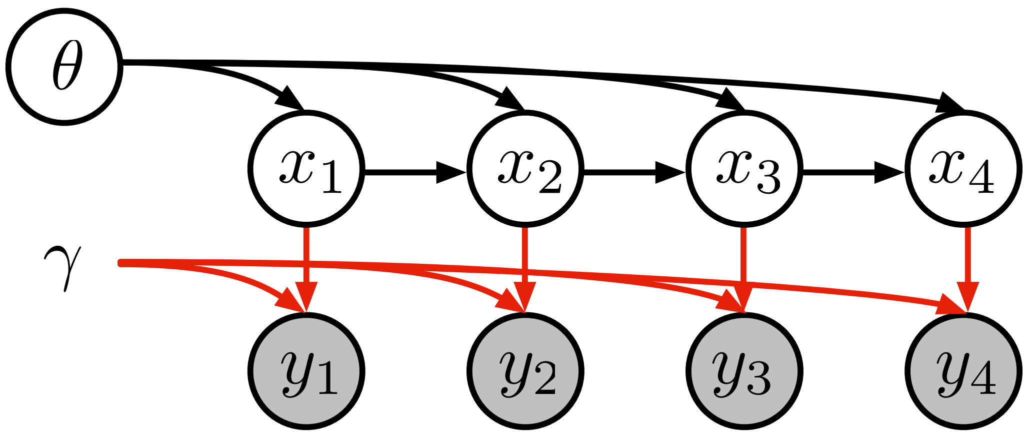
$p(\theta)$ is a conjugate prior

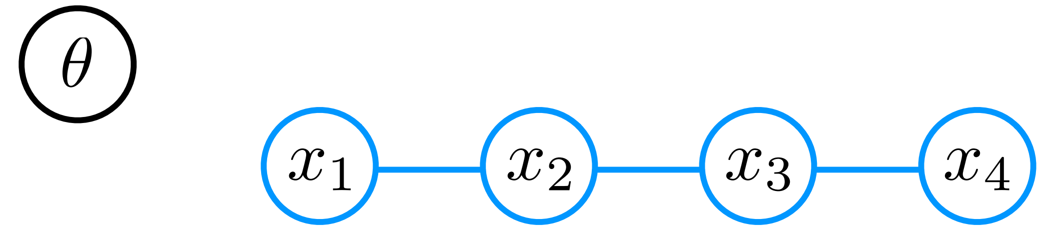




$p(x \mid \theta)$ is a linear dynamical system
$p(y \mid x, \gamma)$ is a neural network decoder
$p(\theta)$ is a conjugate prior

$$q(\theta)q(x) \approx p(\theta, x \mid y, \gamma)$$





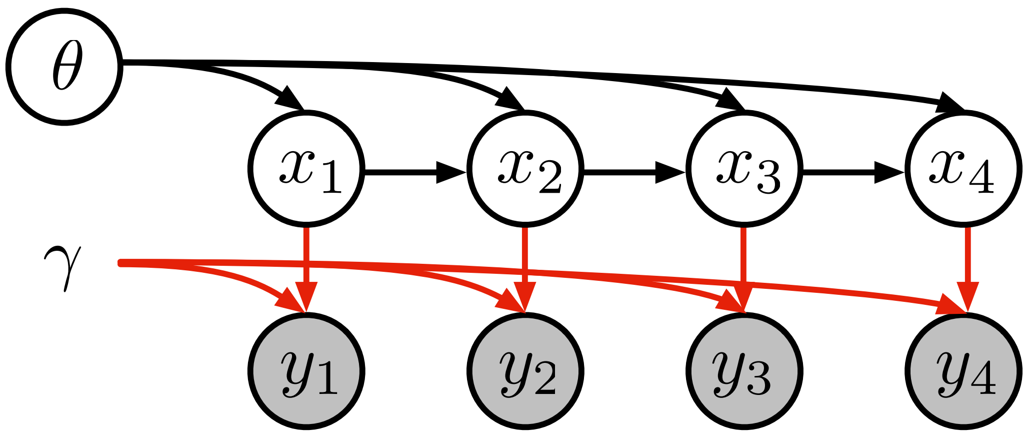
$p(x \,|\, \theta)$ is a linear dynamical system

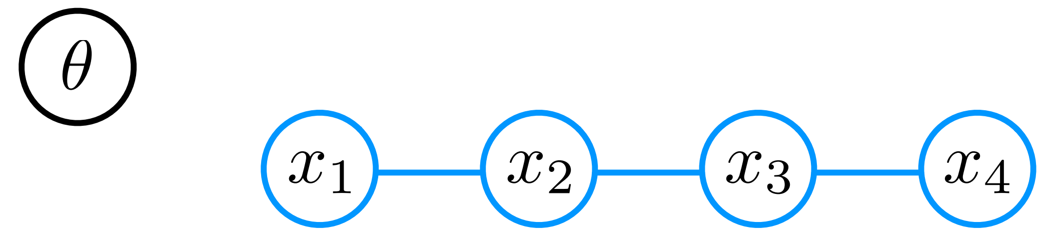
$p(y \,|\, x, \gamma)$ is a neural network decoder

$p(\theta)$ is a conjugate prior

$$q(\theta)q(x) \approx p(\theta, x \,|\, y, \gamma)$$

$$\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)}\left[\log \frac{p(\theta, x)p(y \,|\, x, \gamma)}{q(\theta)q(x)}\right]$$



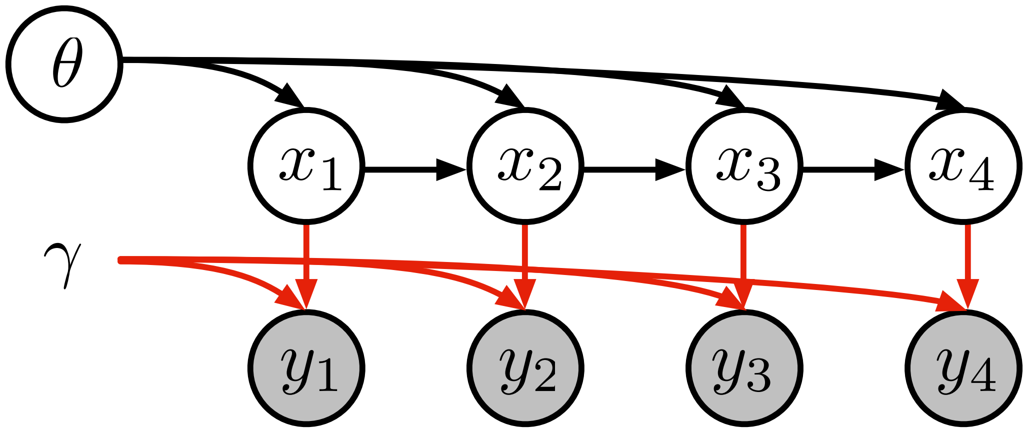


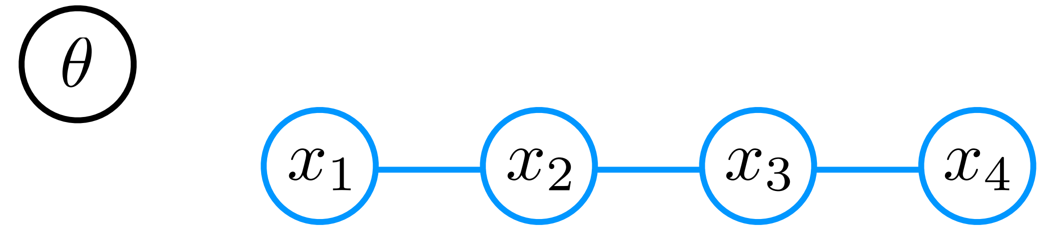
$p(x \mid \theta)$ is a linear dynamical system
$p(y \mid x, \gamma)$ is a neural network decoder
$p(\theta)$ is a conjugate prior

$$q(\theta)q(x) \approx p(\theta, x \mid y, \gamma)$$

$$\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)}\left[\log \frac{p(\theta, x)p(y \mid x, \gamma)}{q(\theta)q(x)}\right]$$

$$\eta_x^*(\eta_\theta) \triangleq \arg\max_{\eta_x} \mathcal{L}(\eta_\theta, \eta_x)$$





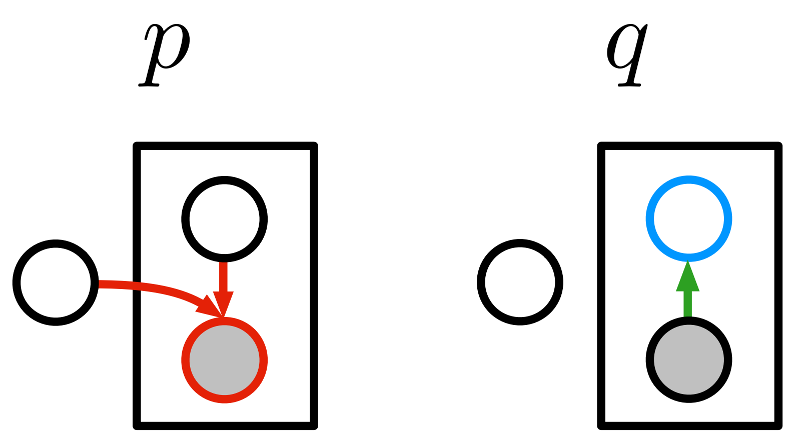
$p(x \mid \theta)$ is a linear dynamical system
$p(y \mid x, \gamma)$ is a neural network decoder
$p(\theta)$ is a conjugate prior

$$q(\theta)q(x) \approx p(\theta, x \mid y, \gamma)$$

$$\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)}\left[\log \frac{p(\theta, x)p(y \mid x, \gamma)}{q(\theta)q(x)}\right]$$

$$\eta_x^*(\eta_\theta) \triangleq \arg\max_{\eta_x} \mathcal{L}(\eta_\theta, \eta_x)$$

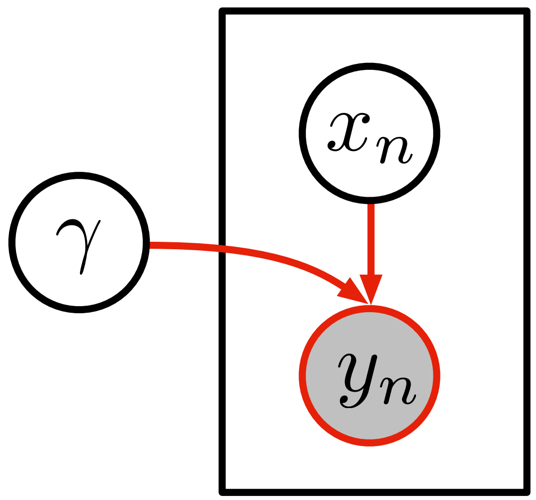
$$\mathcal{L}_{\mathrm{SVI}}(\eta_\theta) \triangleq \mathcal{L}(\eta_\theta, \eta_x^*(\eta_\theta))$$

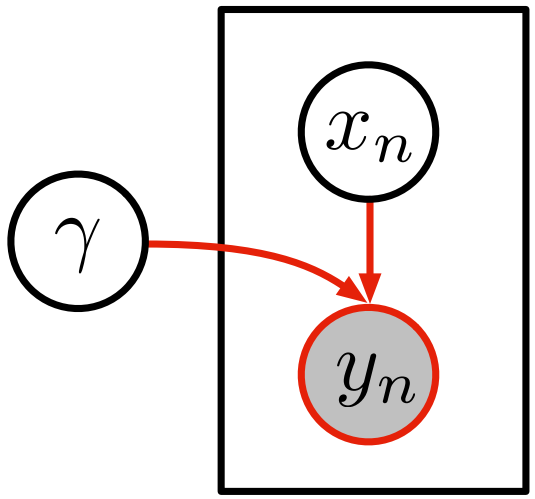


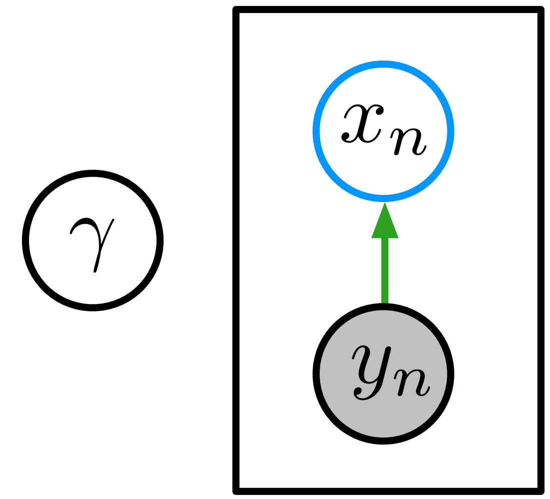
$$q^*(x) \triangleq \mathcal{N}(x \mid \mu(y;\phi), \Sigma(y;\phi))$$

Variational autoencoders and amortized inference [1,2]

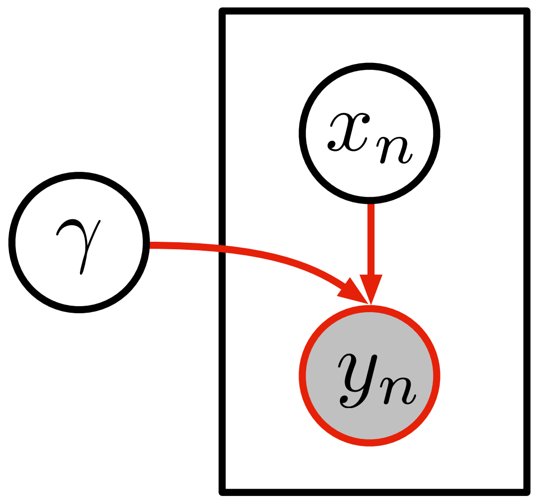
[1] Kingma and Welling. Auto-encoding variational Bayes. ICLR 2014.
[2] Rezende, Mohamed, and Wierstra. Stochastic backpropagation and approximate inference in deep generative models. ICML 2014

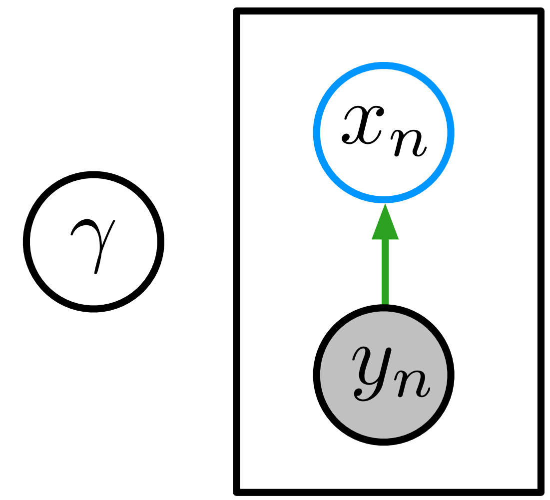

γ
x_n
y_n



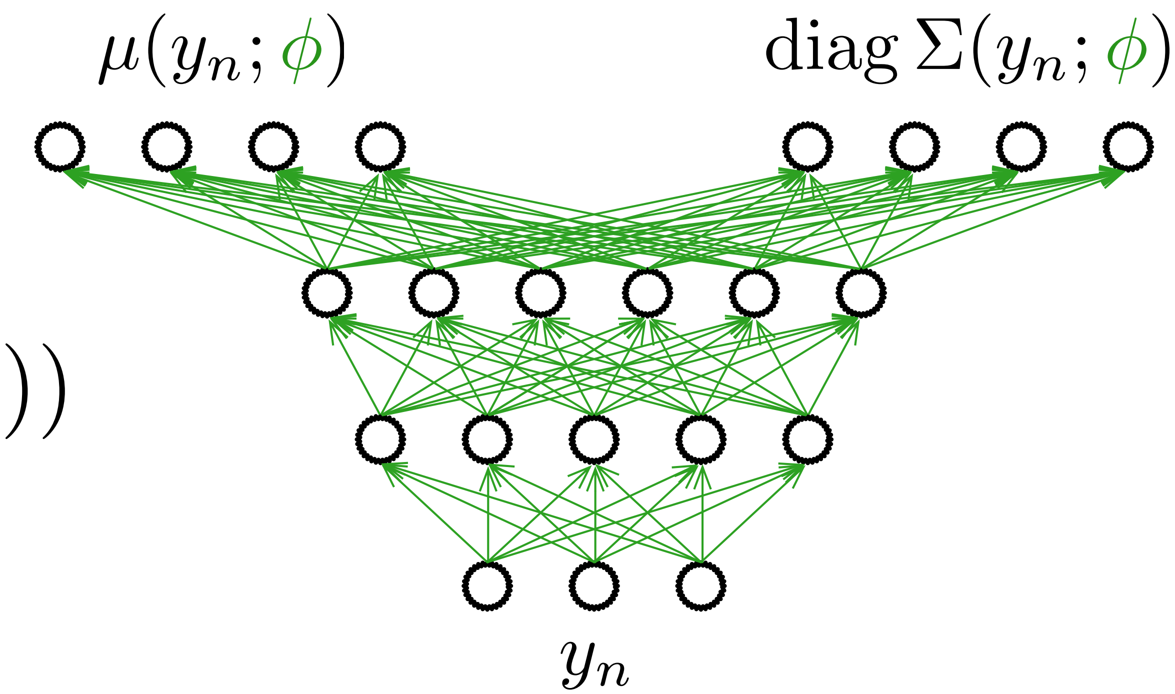


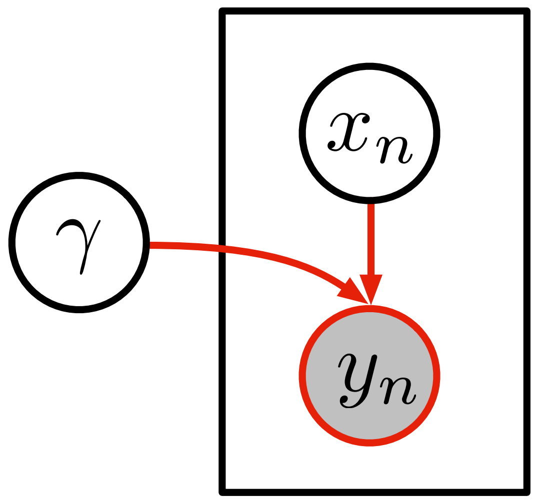
$$q^{\star}(x_n) \triangleq \mathcal{N}(x_n \mid \mu(y_n;\phi),\ \Sigma(y_n;\phi))$$

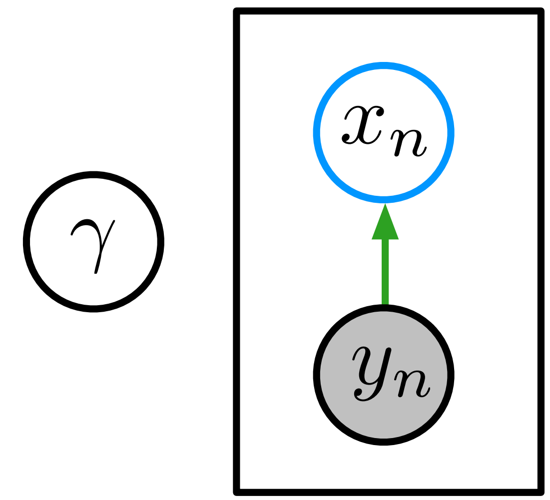


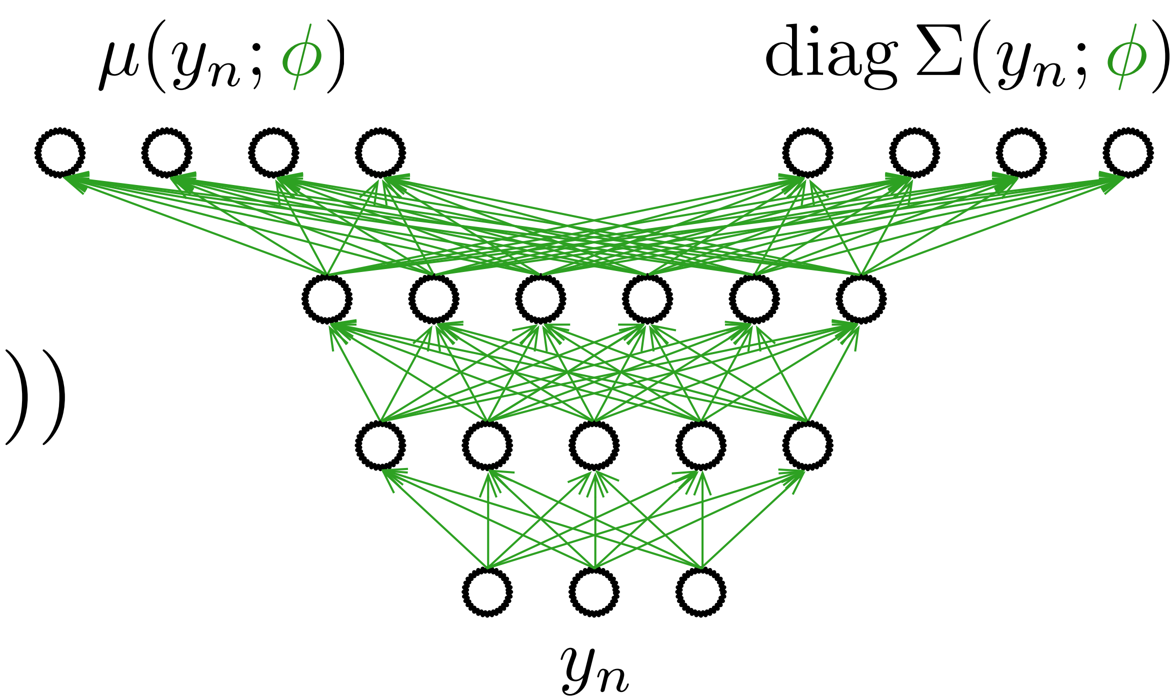


$$q^\star(x_n) \triangleq \mathcal{N}(x_n \mid \mu(y_n; \phi),\ \Sigma(y_n; \phi))$$









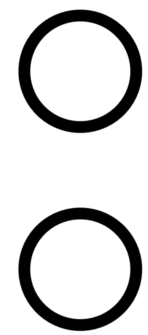
$$q^\star(x_n) \triangleq \mathcal{N}(x_n \mid \mu(y_n; \phi),\ \Sigma(y_n; \phi))$$

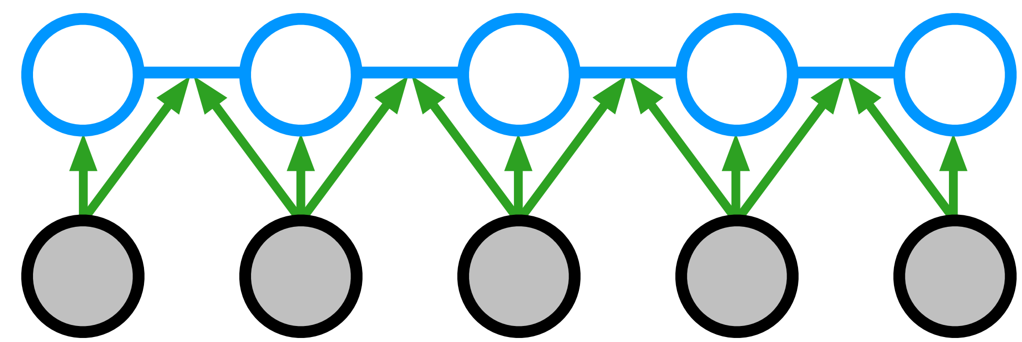
$$\mathcal{L}_{\mathrm{VAE}}(\eta_\gamma, \phi) \triangleq \mathcal{L}(\eta_\gamma, \eta_x^\star(\phi))$$

$$\mu_t(y_t; \phi_\mu)$$

[1,2] $$J_{t,t}(y_t; \phi_D)$$

$$J_{t,t+1}(y_t, y_{t+1}; \phi_B)$$





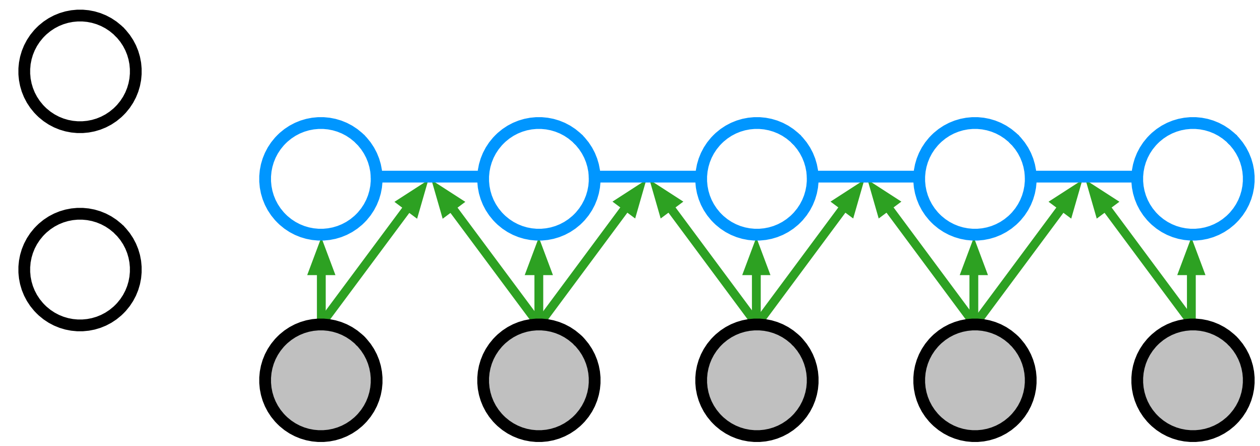
[1] Archer, Park, Buesing, Cunningham, Paninski. Black box variational inference for state space models. ICLR 2016 Workshops.
[2] Gao*, Archer*, Paninski, Cunningham. Linear dynamical neural population models through nonlinear embeddings. NIPS 2016.

[1,2]

$$\mu_t(y_t;\phi_\mu)$$

$$J_{t,t}(y_t;\phi_D)$$

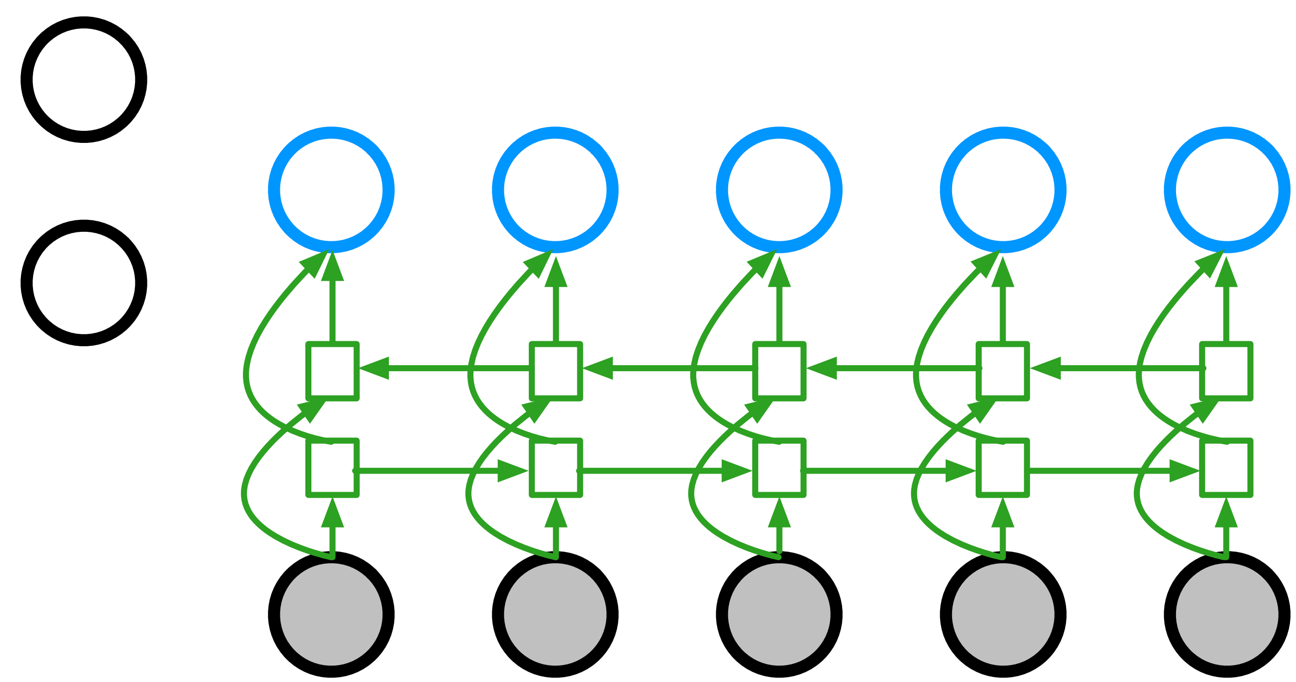
$$J_{t,t+1}(y_t,y_{t+1};\phi_B)$$



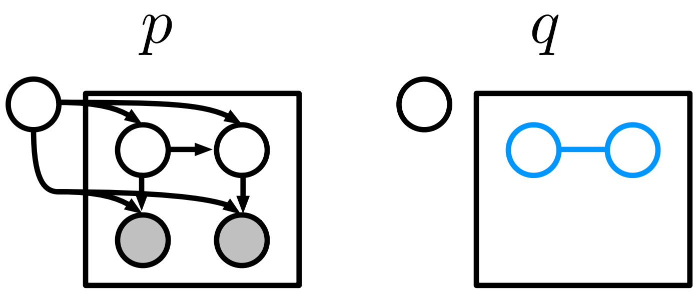
[3]

$$\mu_t(y_{1:T},\hat{x}_{t-1};\phi)$$

$$\Sigma_t(y_{1:T},\hat{x}_{t-1};\phi)$$

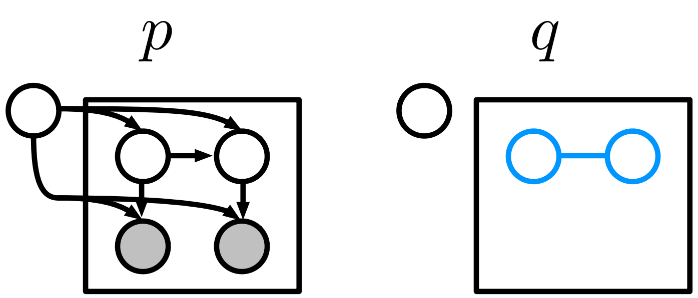


[1] Archer, Park, Buesing, Cunningham, Paninski. Black box variational inference for state space models. ICLR 2016 Workshops.
[2] Gao*, Archer*, Paninski, Cunningham. Linear dynamical neural population models through nonlinear embeddings. NIPS 2016.
[3] Krishnan, Shalit, Sontag. Structured inference networks for nonlinear state space models. AISTATS 2017.



$$q^*(x) \triangleq \arg\max_{q(x)} \mathcal{L}[\, q(\theta)q(x) \,]$$

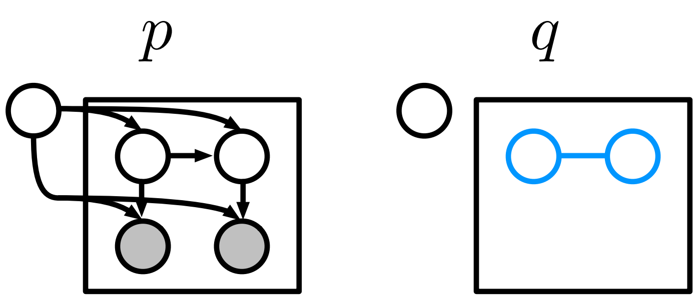
Natural gradient SVI



$$q^*(x) \triangleq \arg\max_{q(x)} \mathcal{L}[\, q(\theta)q(x) \,]$$

Natural gradient SVI

– expensive for general obs.

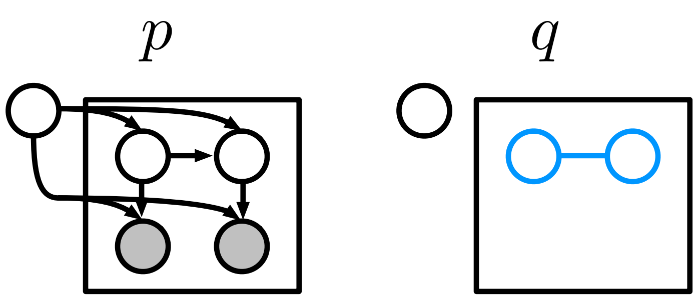


$$q^*(x) \triangleq \arg\max_{q(x)} \mathcal{L}[\, q(\theta) q(x) \,]$$

Natural gradient SVI

− expensive for general obs.

+ optimal local factor



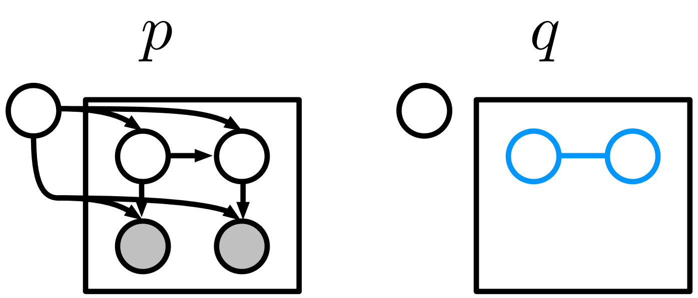
$$q^*(x) \triangleq \arg\max_{q(x)} \mathcal{L}[\, q(\theta)q(x)\,]$$

Natural gradient SVI

– expensive for general obs.

\+ optimal local factor

\+ exploits conj. graph structure



$$q^*(x) \triangleq \arg\max_{q(x)} \mathcal{L}[\, q(\theta) q(x) \,]$$

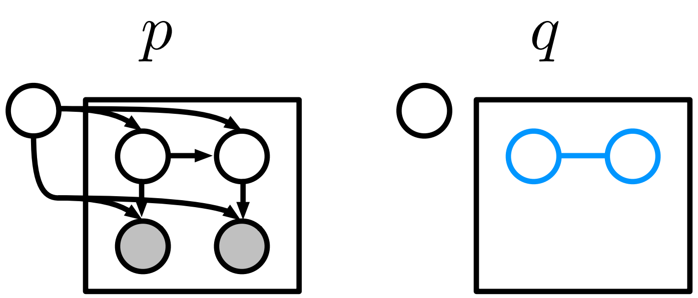
Natural gradient SVI

– expensive for general obs.

+ optimal local factor

+ exploits conj. graph structure

+ arbitrary inference queries



$$q^*(x) \triangleq \arg\max_{q(x)} \mathcal{L}[\, q(\theta) q(x) \,]$$

Natural gradient SVI

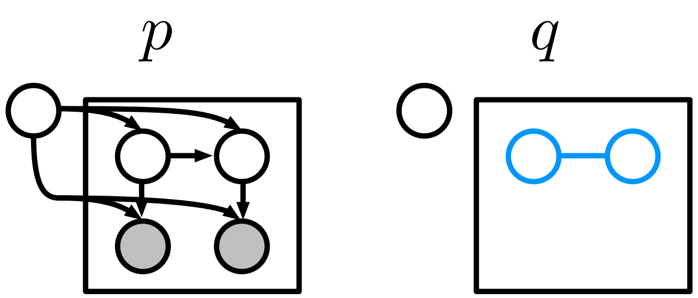
– expensive for general obs.

+ optimal local factor

+ exploits conj. graph structure

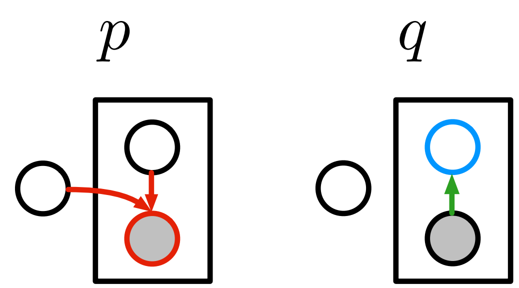
+ arbitrary inference queries

+ natural gradients



$$q^*(x) \triangleq \arg\max_{q(x)} \mathcal{L}[\, q(\theta)q(x)\,]$$

Natural gradient SVI



$$q^*(x) \triangleq \mathcal{N}(x \mid \mu(y;\phi), \Sigma(y;\phi))$$

Variational autoencoders

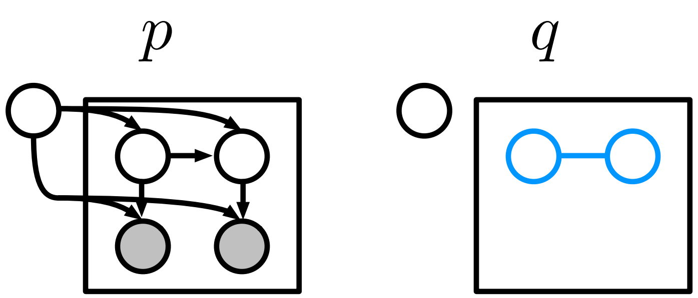
– expensive for general obs.

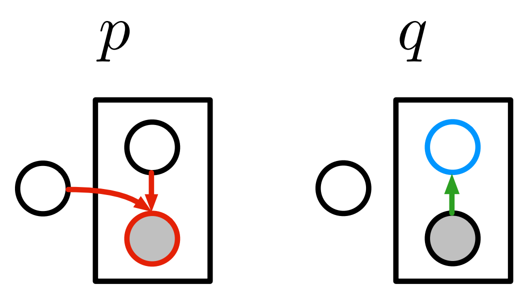
\+ optimal local factor

\+ exploits conj. graph structure

\+ arbitrary inference queries

\+ natural gradients





$$q^*(x) \triangleq \arg\max_{q(x)} \mathcal{L}[\, q(\theta) q(x) \,]$$

$$q^*(x) \triangleq \mathcal{N}(x \,|\, \mu(y;\phi), \Sigma(y;\phi))$$

Natural gradient SVI

– expensive for general obs.

+ optimal local factor

+ exploits conj. graph structure

+ arbitrary inference queries

+ natural gradients

Variational autoencoders

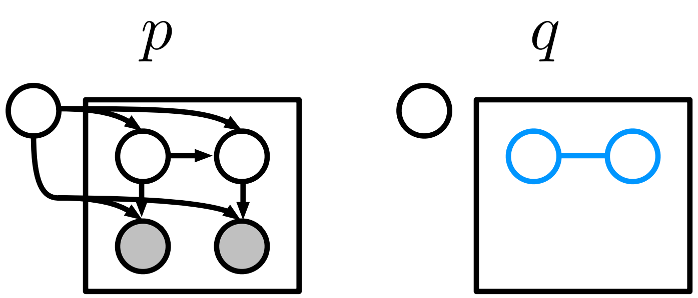
+ fast for general obs.

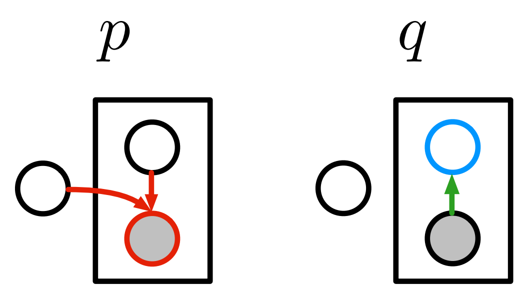
– suboptimal local inference

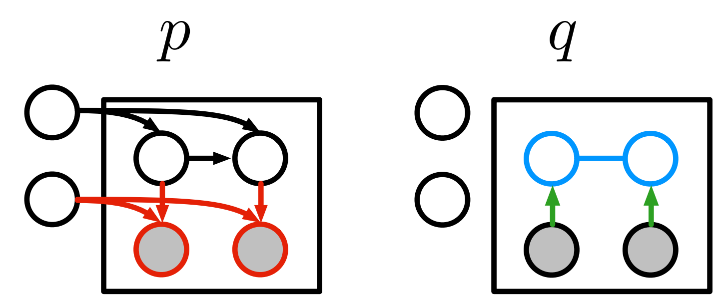
– ϕ does all local inference

– limited inference queries

– no cheap natural gradients







$$q^*(x) \triangleq \underset{q(x)}{\arg\max}\, \mathcal{L}[\, q(\theta)q(x)\,]$$

$$q^*(x) \triangleq \mathcal{N}(x \mid \mu(y;\phi), \Sigma(y;\phi))$$

$$q^*(x) \triangleq \ ?$$

Natural gradient SVI

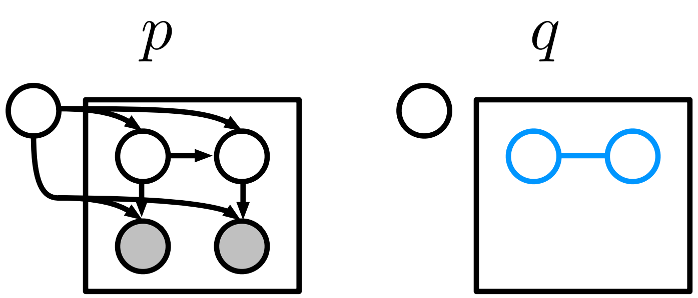
- – expensive for general obs.
- + optimal local factor
- + exploits conj. graph structure
- + arbitrary inference queries
- + natural gradients

Variational autoencoders

- + fast for general obs.
- – suboptimal local inference
- – ϕ does all local inference
- – limited inference queries
- – no cheap natural gradients

Structured VAEs [1]

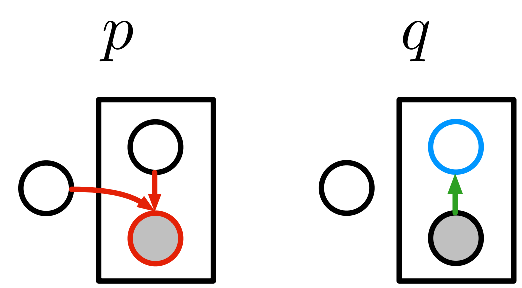
[1] **Johnson**, Duvenaud, Wiltschko, Datta, and Adams. Composing graphical models and neural networks. NIPS 2016.



$$q^*(x) \triangleq \arg\max_{q(x)} \mathcal{L}[\, q(\theta)q(x)\,]$$

Natural gradient SVI

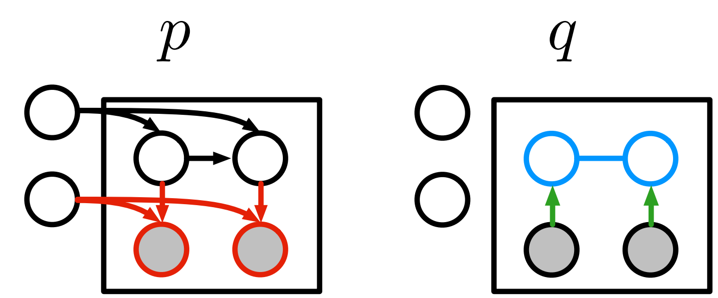
- – expensive for general obs.
- + optimal local factor
- + exploits conj. graph structure
- + arbitrary inference queries
- + natural gradients



$$q^*(x) \triangleq \mathcal{N}(x \,|\, \mu(y;\phi), \Sigma(y;\phi))$$

Variational autoencoders

- + fast for general obs.
- – suboptimal local inference
- – ϕ does all local inference
- – limited inference queries
- – no cheap natural gradients



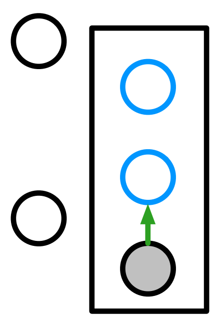
$$q^*(x) \triangleq \; ?$$

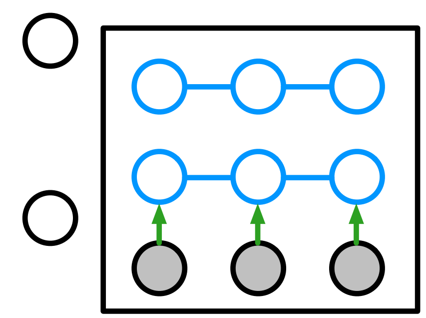
Structured VAEs [1]

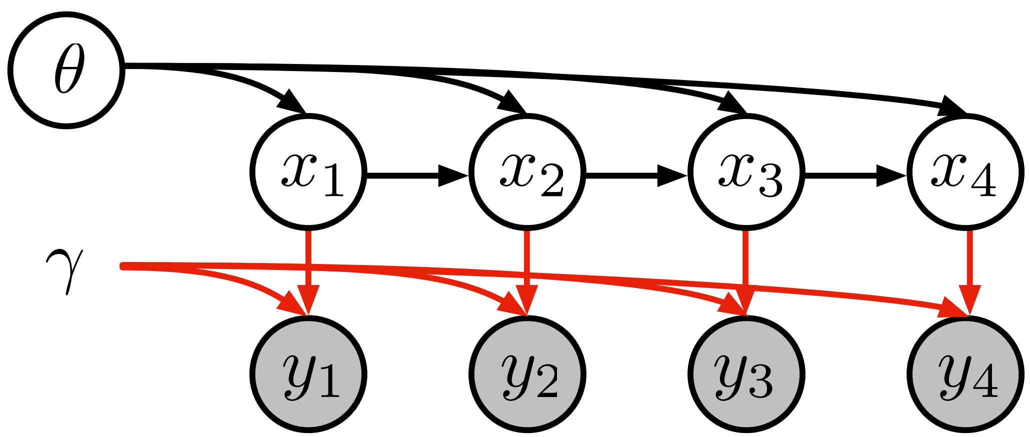
- + fast for general obs.
- ± optimal given conj. evidence
- + exploits conj. graph structure
- + arbitrary inference queries
- + natural gradients on η_θ

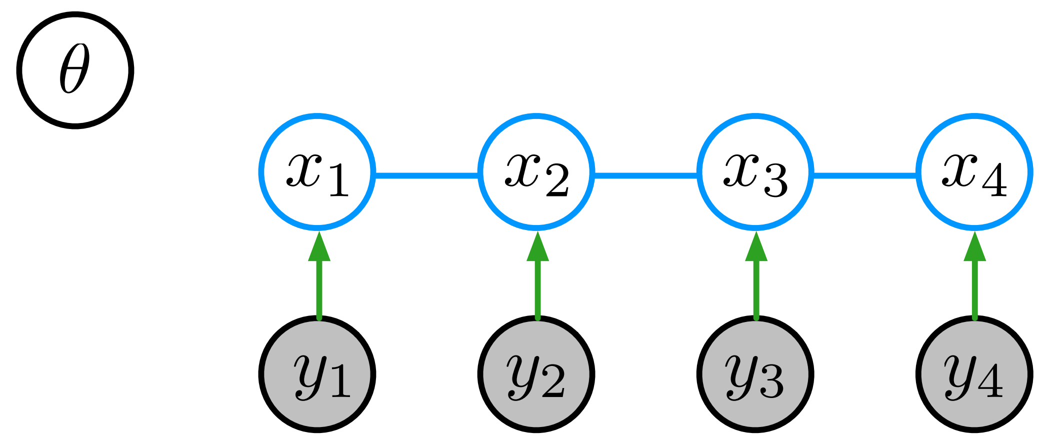
[1] **Johnson**, Duvenaud, Wiltschko, Datta, and Adams. Composing graphical models and neural networks. NIPS 2016.

Inference: recognition networks output conjugate potentials, then apply fast graphical model inference

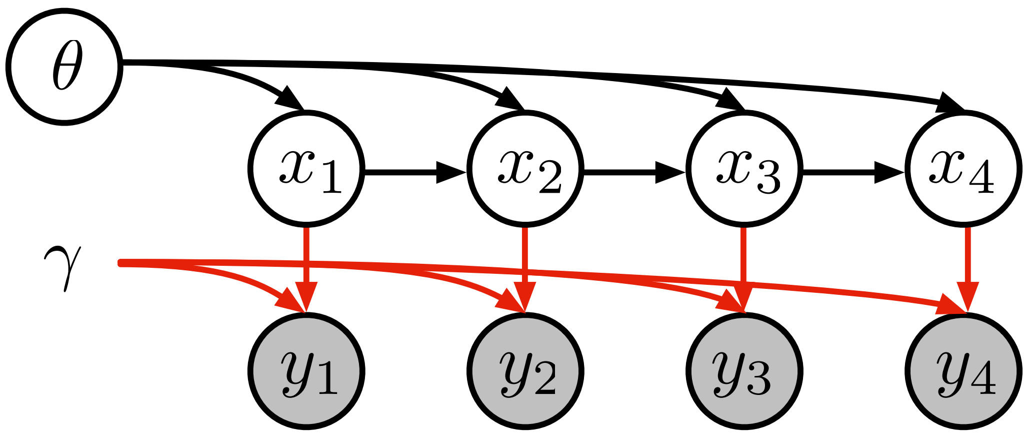


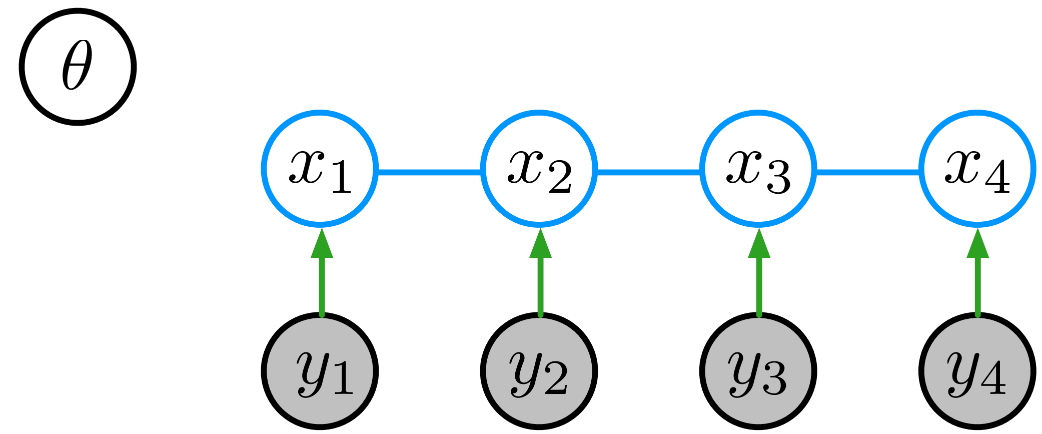






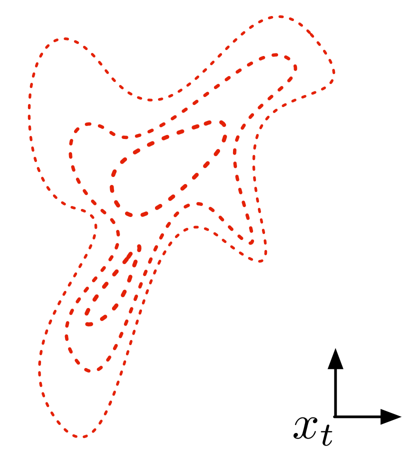
$$\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)}\left[\log \frac{p(\theta, x)p(y \mid x, \gamma)}{q(\theta)q(x)}\right]$$

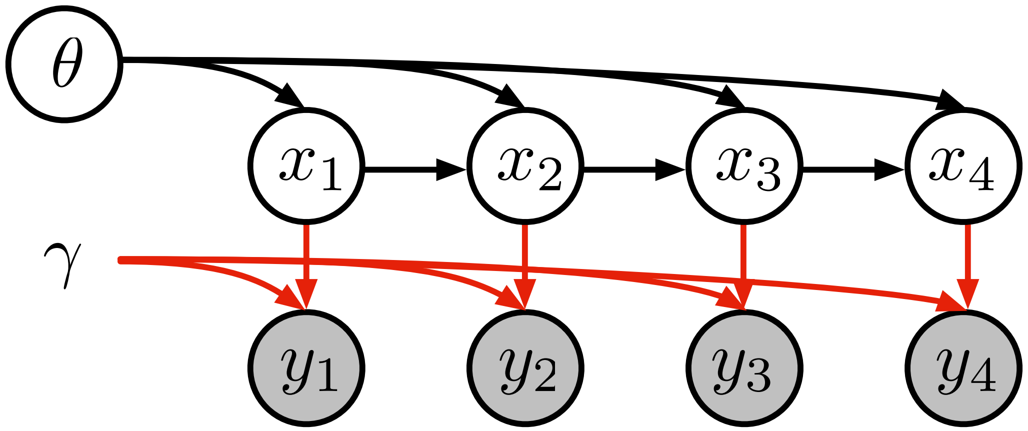


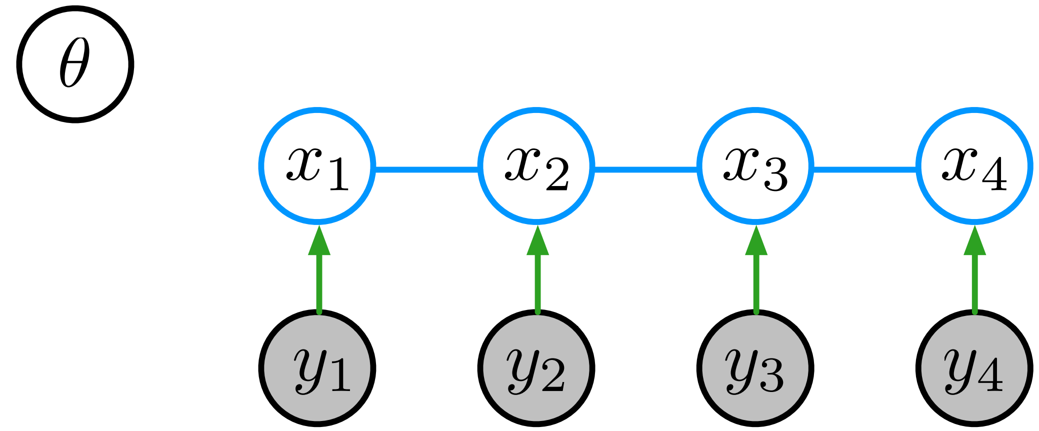


$$\mathbb{E}_{q(\gamma)} \log p(y_t \mid x_t, \gamma)$$

$$\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)}\left[\log \frac{p(\theta, x)p(y \mid x, \gamma)}{q(\theta)q(x)}\right]$$

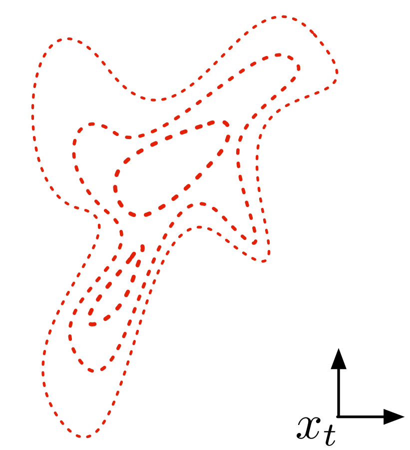






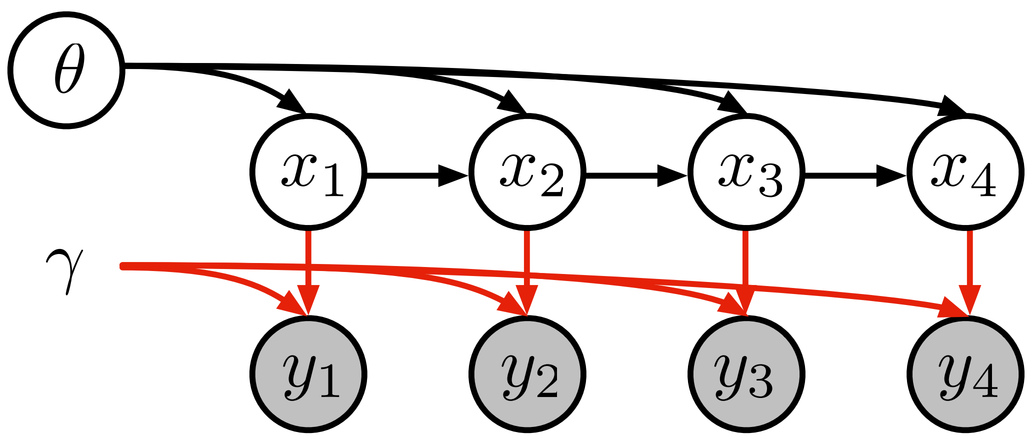
$$\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)}\left[\log \frac{p(\theta, x)p(y \,|\, x, \gamma)}{q(\theta)q(x)}\right]$$

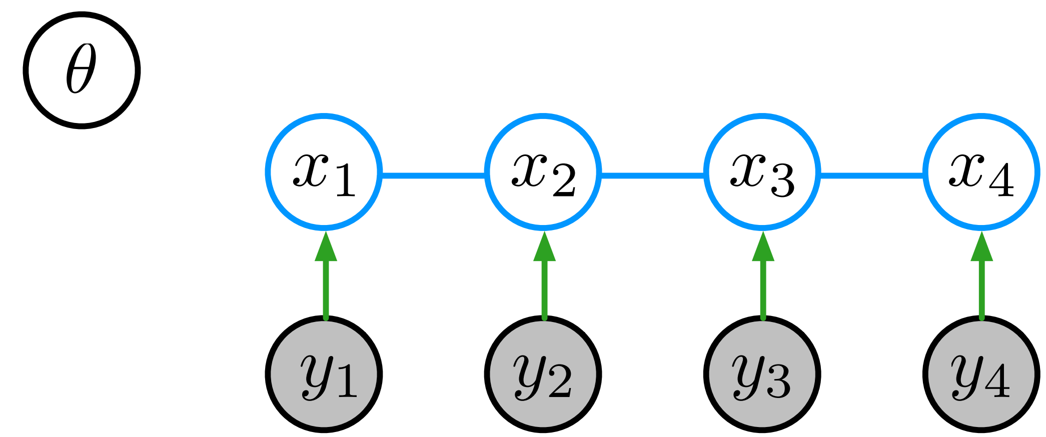
$$\mathbb{E}_{q(\gamma)} \log p(y_t \,|\, x_t, \gamma)$$



$$\widehat{\mathcal{L}}(\eta_\theta, \eta_x, \phi) \triangleq \mathbb{E}_{q(\theta)q(x)}\left[\log \frac{p(\theta, x)\exp(\psi(x\,;\, y, \phi))}{q(\theta)q(x)}\right]$$

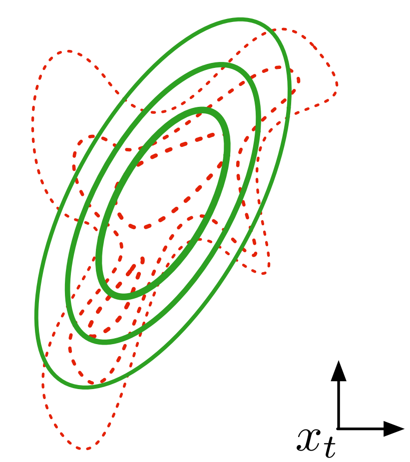
where $\psi(x\,;\, y, \phi)$ is a conjugate potential for $p(x \,|\, \theta)$.





$$\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)}\left[\log \frac{p(\theta, x)p(y \mid x, \gamma)}{q(\theta)q(x)}\right]$$

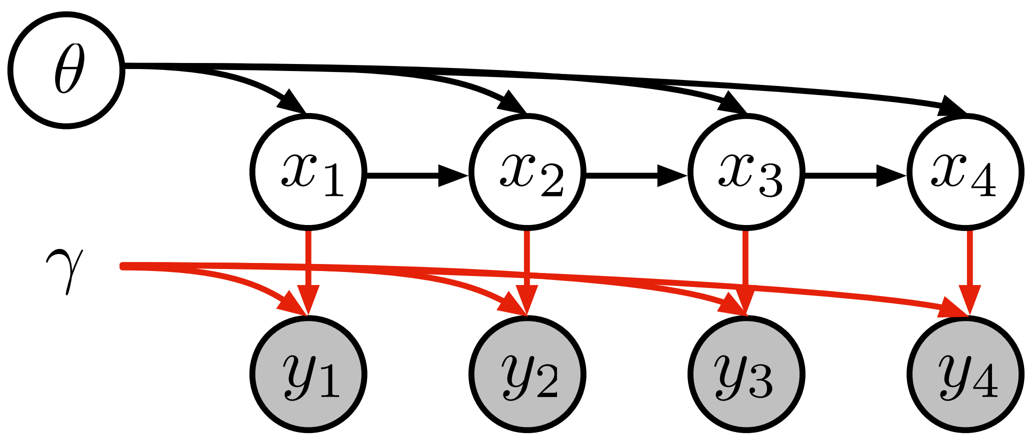
$\mathbb{E}_{q(\gamma)} \log p(y_t \mid x_t, \gamma)$

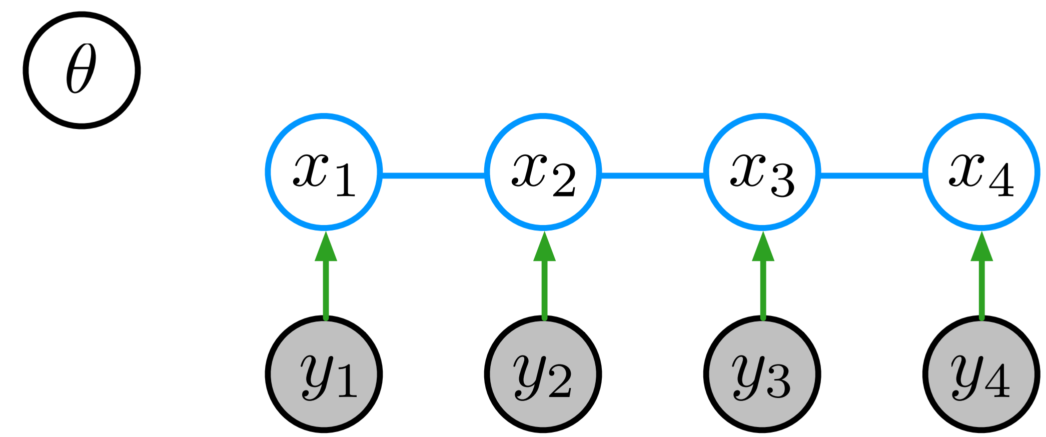


$\psi(x_t; y_t, \phi)$

$$\widehat{\mathcal{L}}(\eta_\theta, \eta_x, \phi) \triangleq \mathbb{E}_{q(\theta)q(x)}\left[\log \frac{p(\theta, x)\exp(\psi(x\,;\,y, \phi))}{q(\theta)q(x)}\right]$$

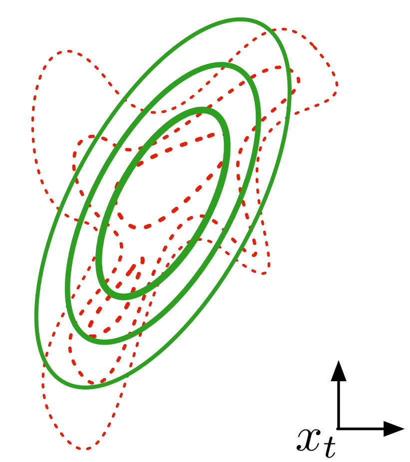
where $\psi(x\,;\,y, \phi)$ is a conjugate potential for $p(x \mid \theta)$.





$$\mathcal{L}(\eta_\theta, \eta_x) \triangleq \mathbb{E}_{q(\theta)q(x)}\left[\log \frac{p(\theta, x)p(y \,|\, x, \gamma)}{q(\theta)q(x)}\right]$$

$\mathbb{E}_{q(\gamma)} \log p(y_t \,|\, x_t, \gamma)$



$\psi(x_t; y_t, \phi)$

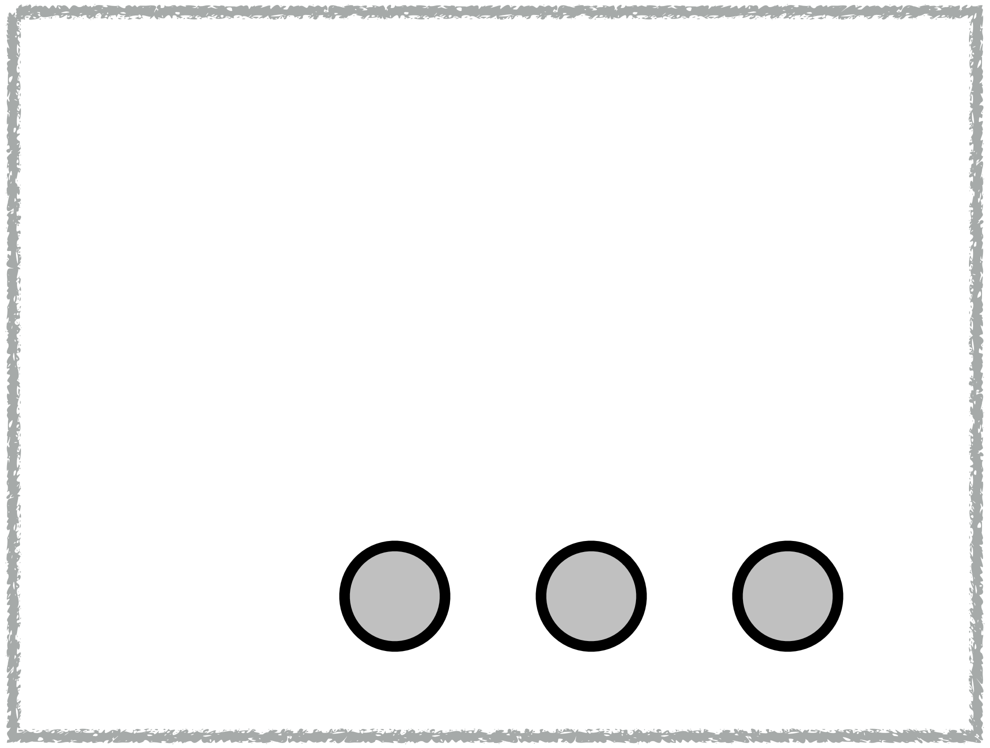
$$\widehat{\mathcal{L}}(\eta_\theta, \eta_x, \phi) \triangleq \mathbb{E}_{q(\theta)q(x)}\left[\log \frac{p(\theta, x)\exp(\psi(x\,;\,y, \phi))}{q(\theta)q(x)}\right]$$

where $\psi(x\,;\,y, \phi)$ is a conjugate potential for $p(x \,|\, \theta)$.

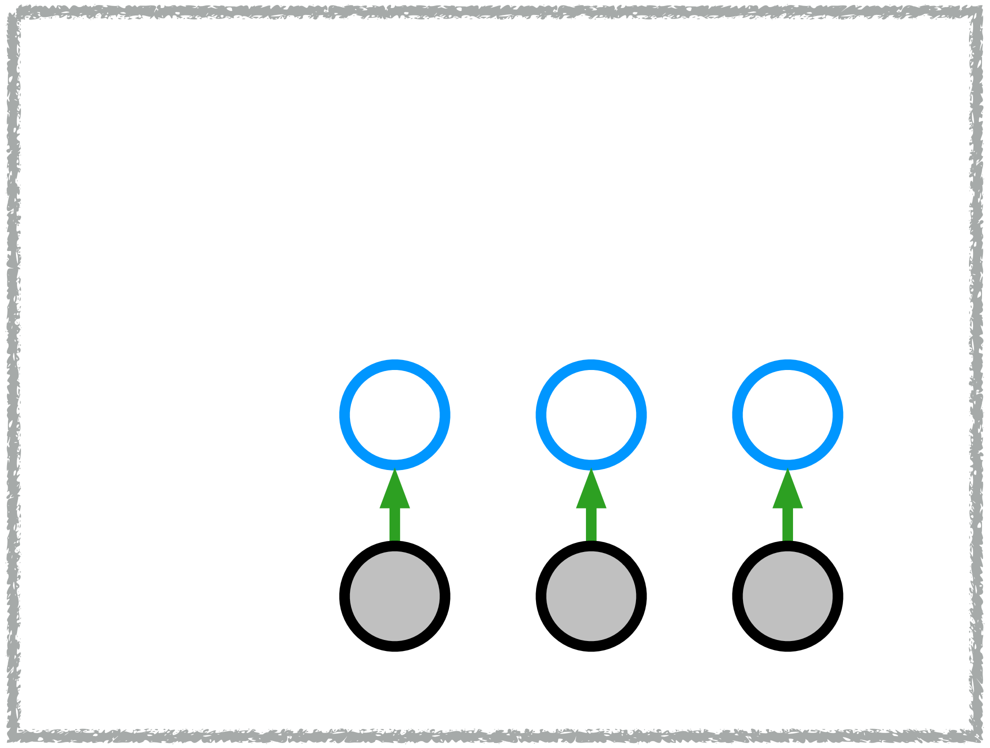
$$\eta_x^*(\eta_\theta, \phi) \triangleq \arg\max_{\eta_x} \widehat{\mathcal{L}}(\eta_\theta, \eta_x, \phi) \qquad \mathcal{L}_{\text{SVAE}}(\eta_\theta, \phi) \triangleq \mathcal{L}(\eta_\theta, \eta_x^*(\eta_\theta, \phi))$$

Step 1: apply recognition network

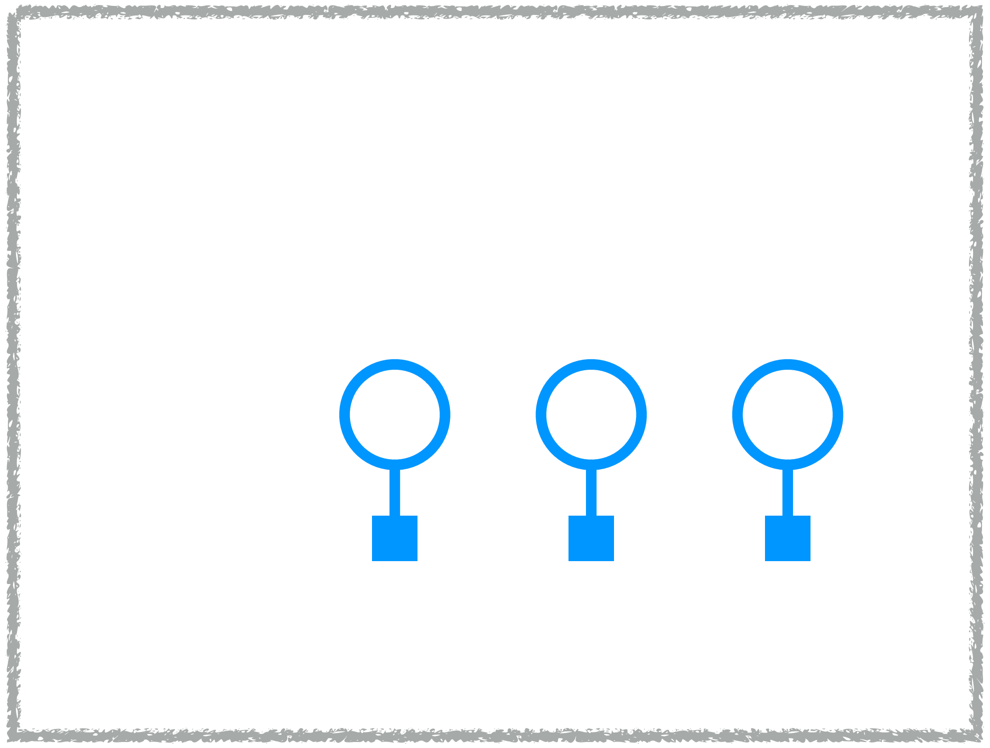
Step 1: apply recognition network



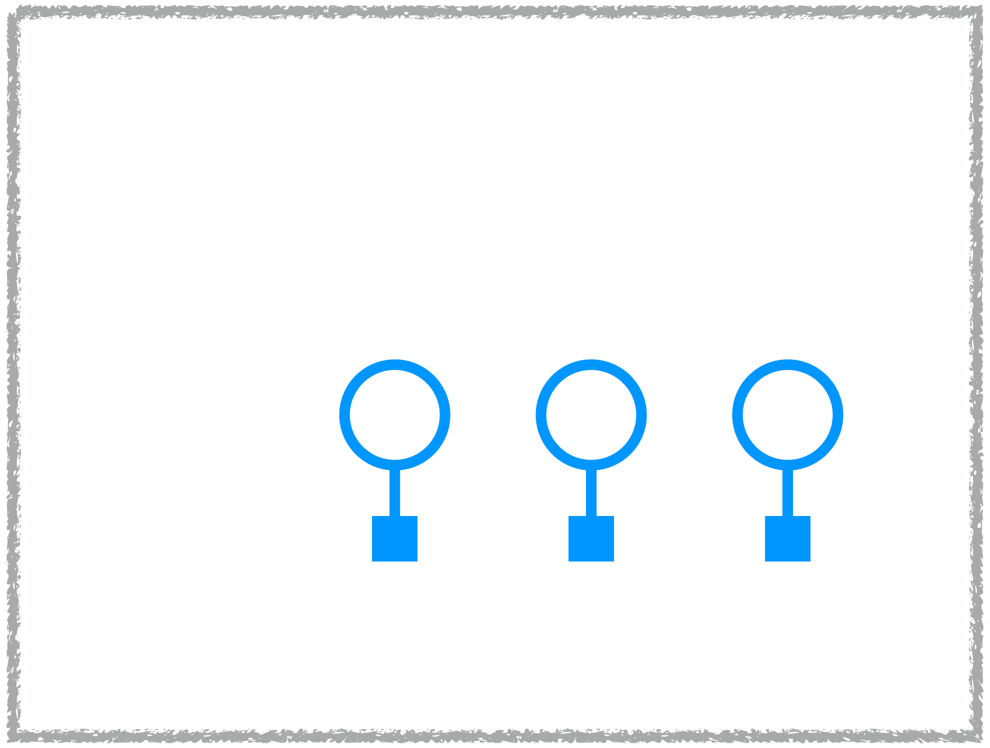
Step 1: apply recognition network



Step 1: apply recognition network

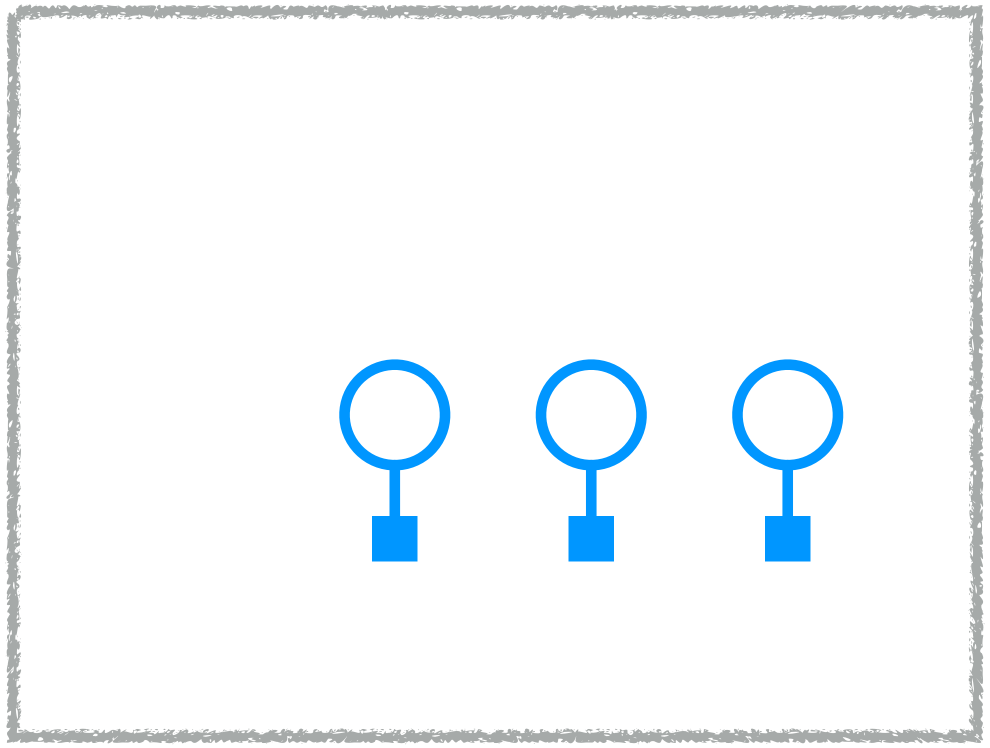


Step 1: apply recognition network

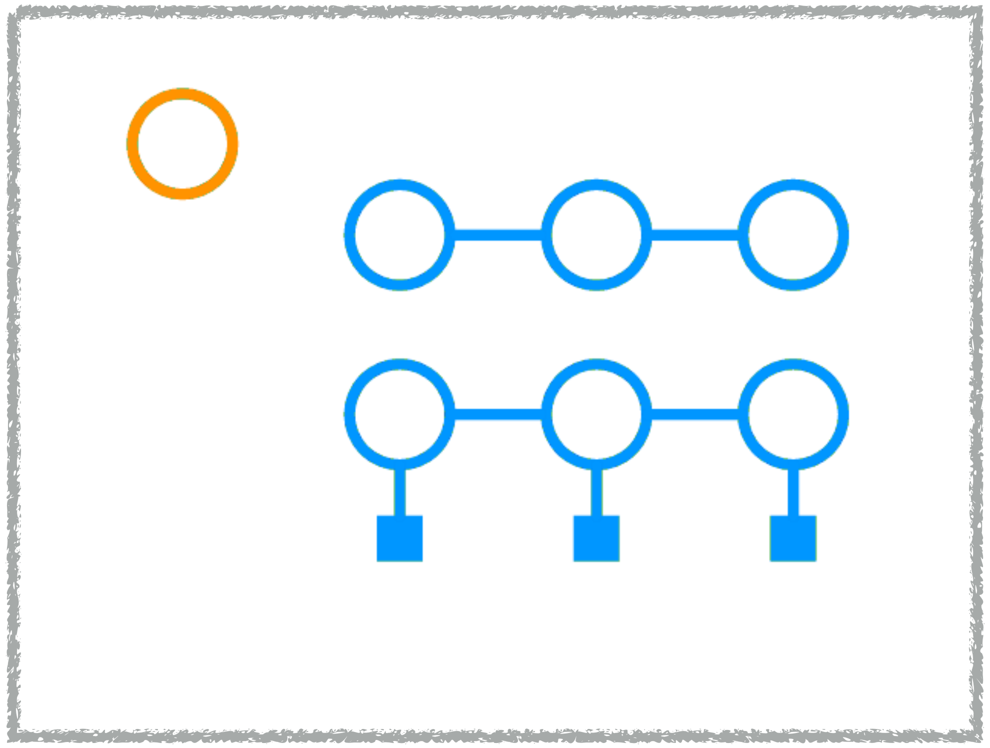


Step 2: run fast PGM algorithms

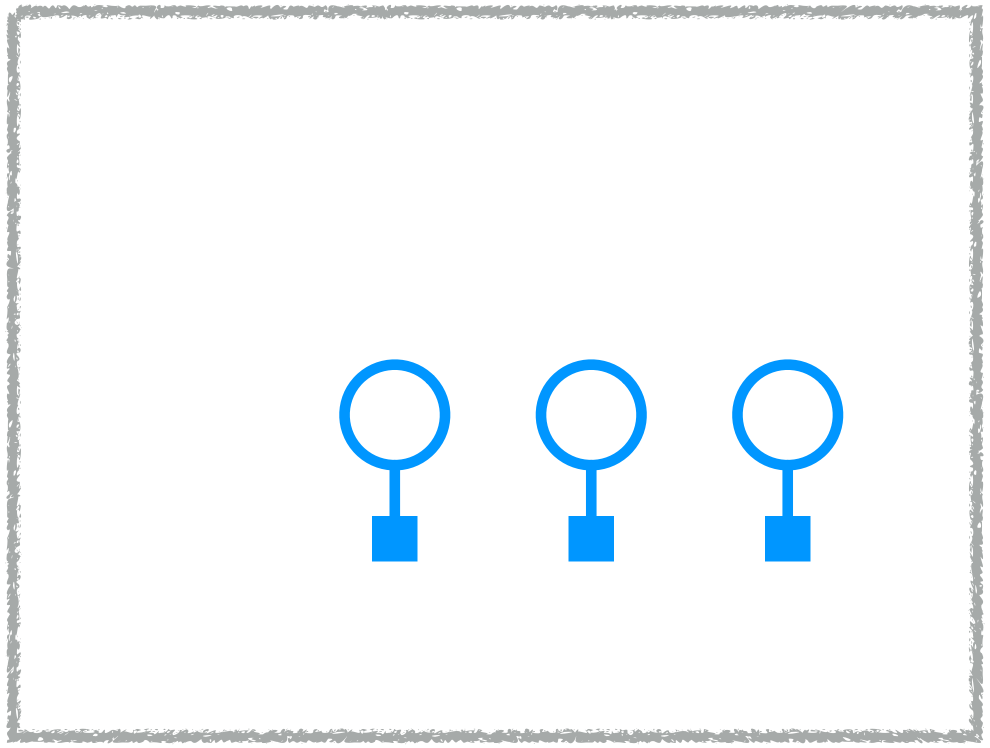
Step 1: apply recognition network



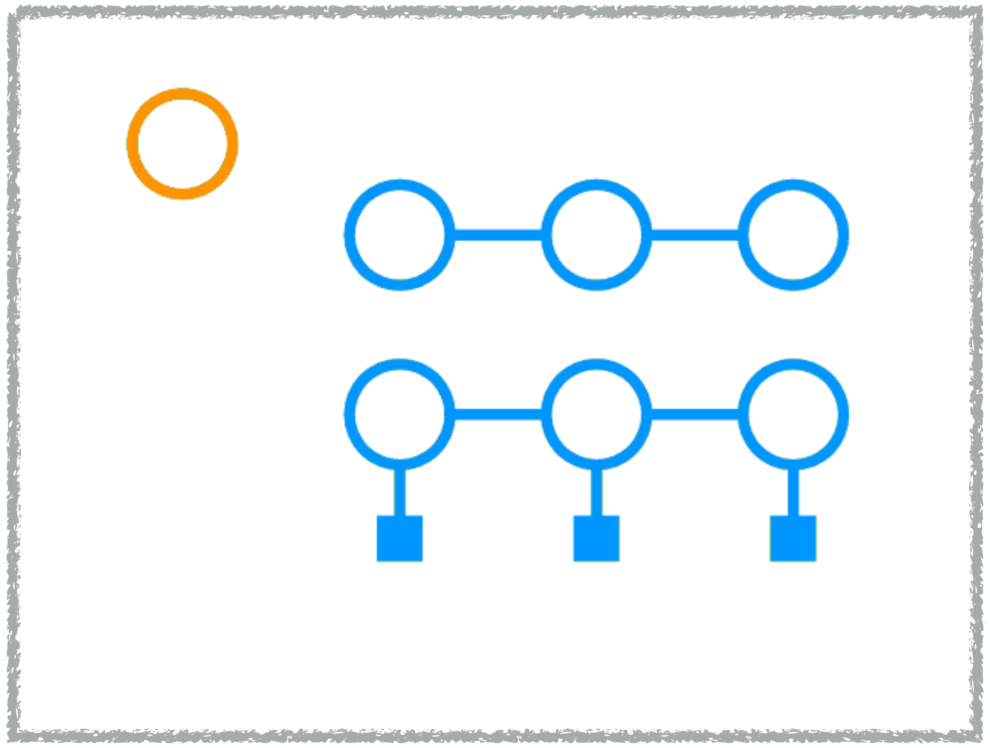
Step 2: run fast PGM algorithms



Step 1: apply recognition network

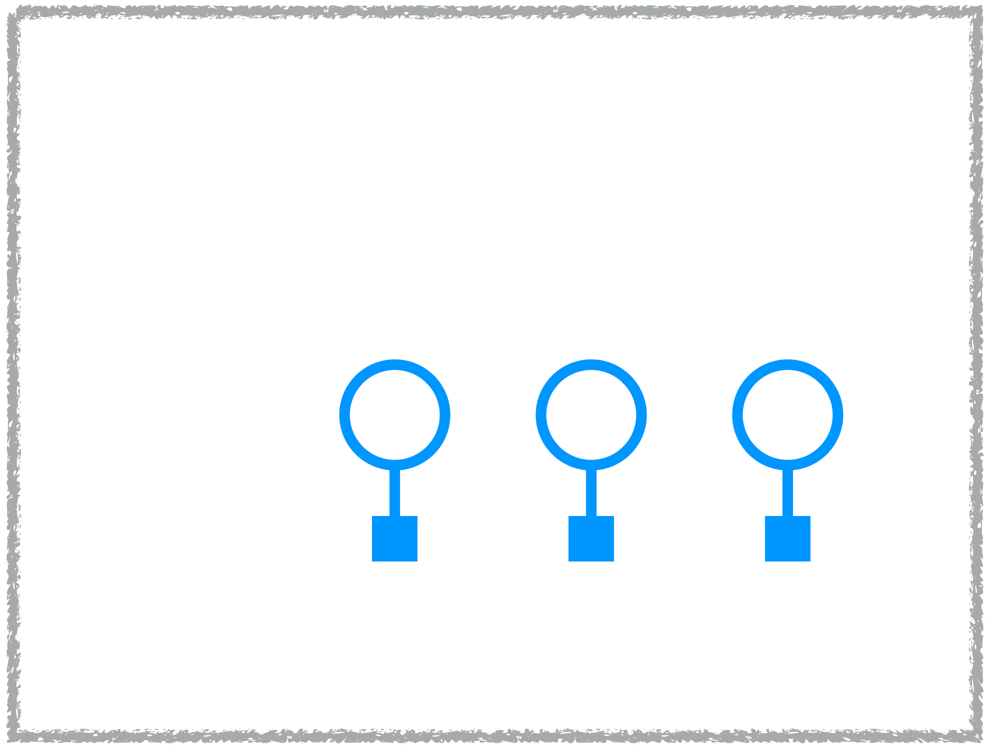


Step 2: run fast PGM algorithms

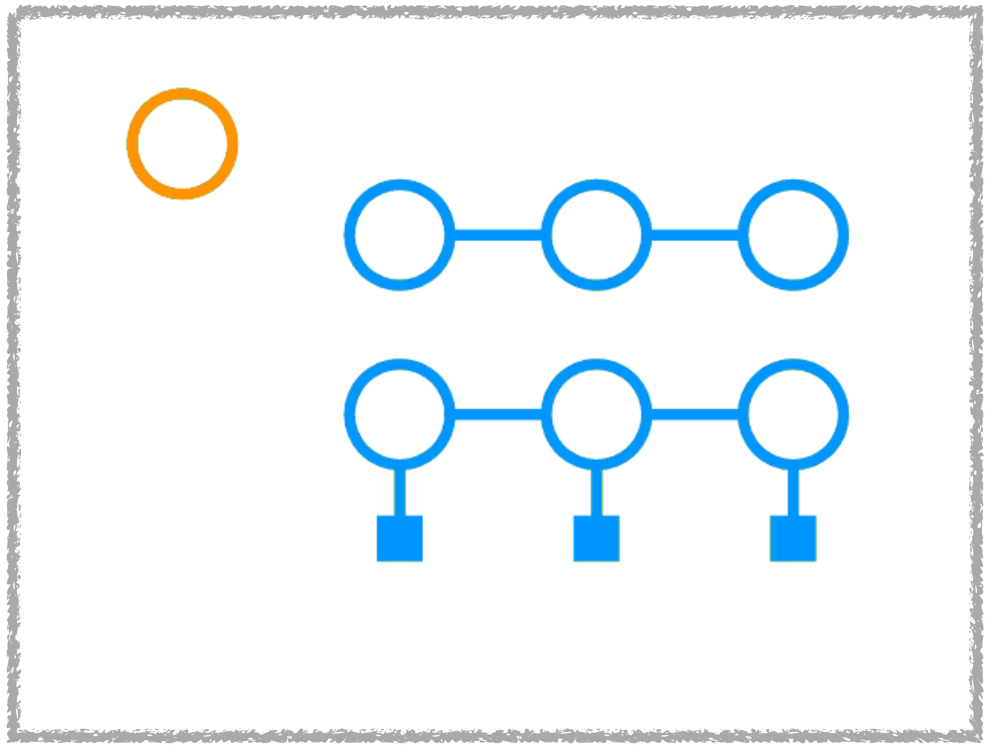


Step 3: sample, compute flat grads

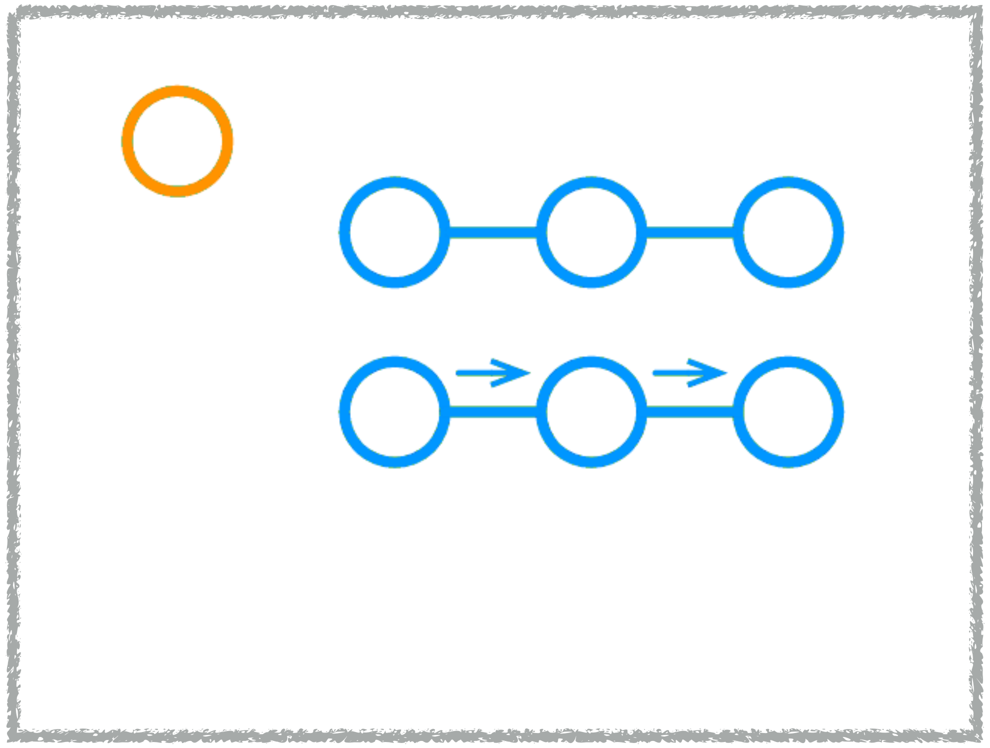
Step 1: apply recognition network



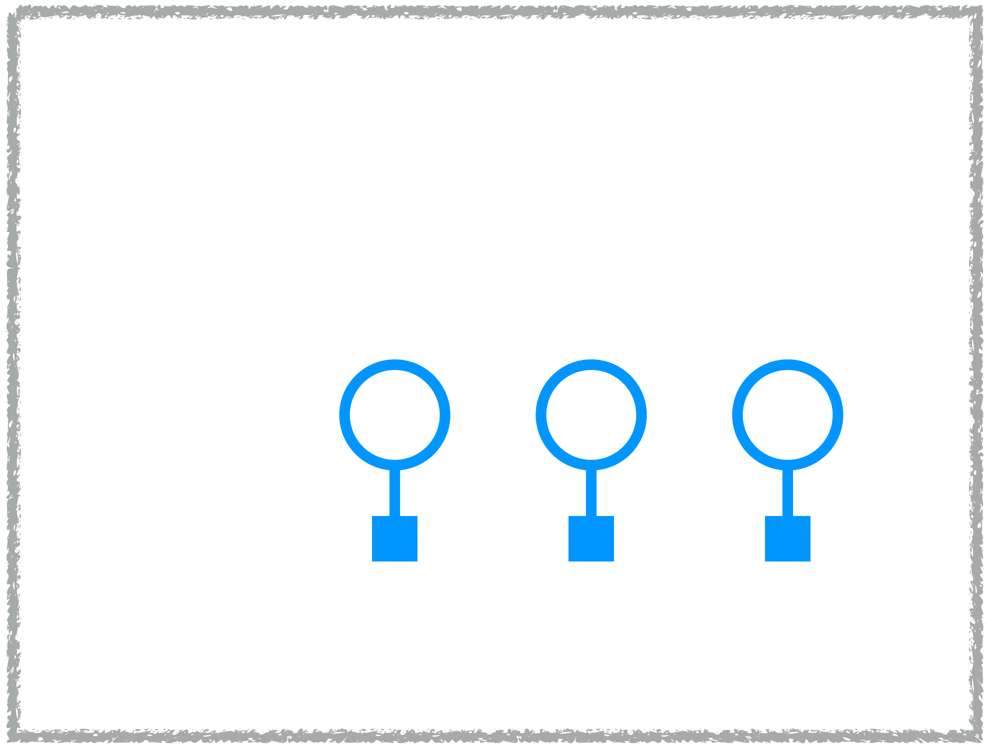
Step 2: run fast PGM algorithms



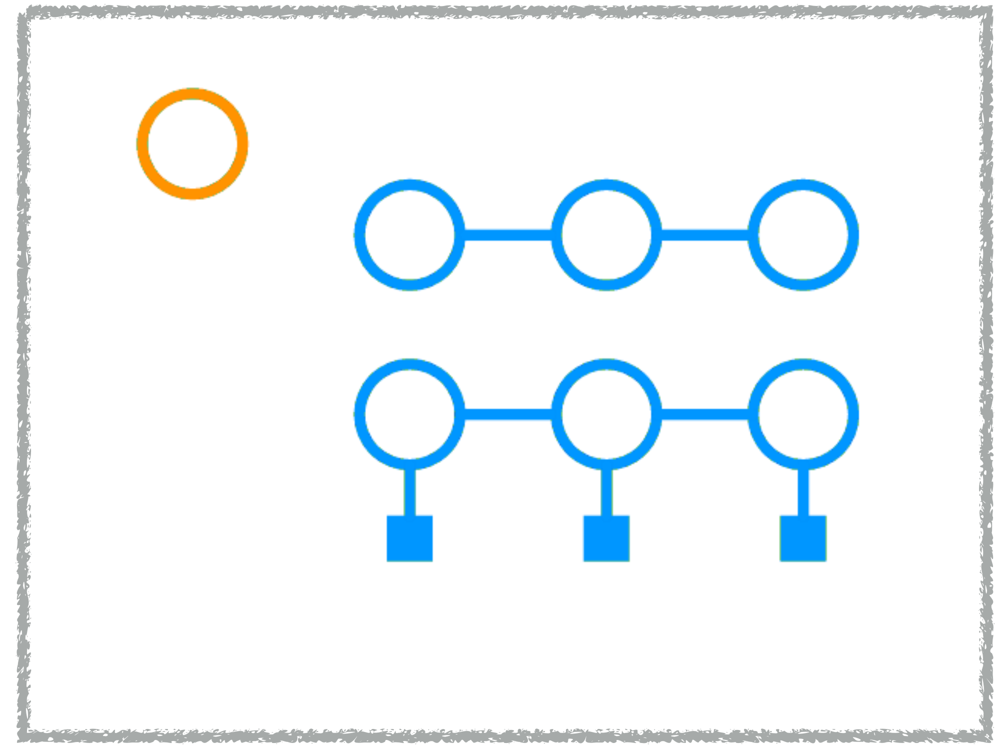
Step 3: sample, compute flat grads



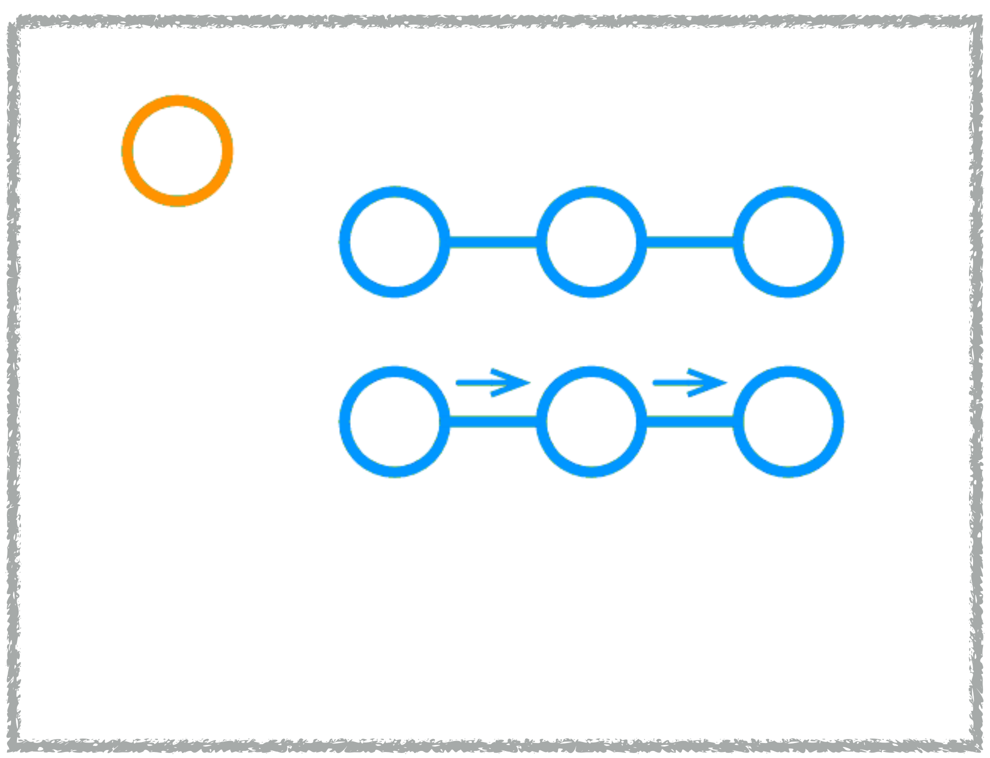
Step 1: apply recognition network



Step 2: run fast PGM algorithms

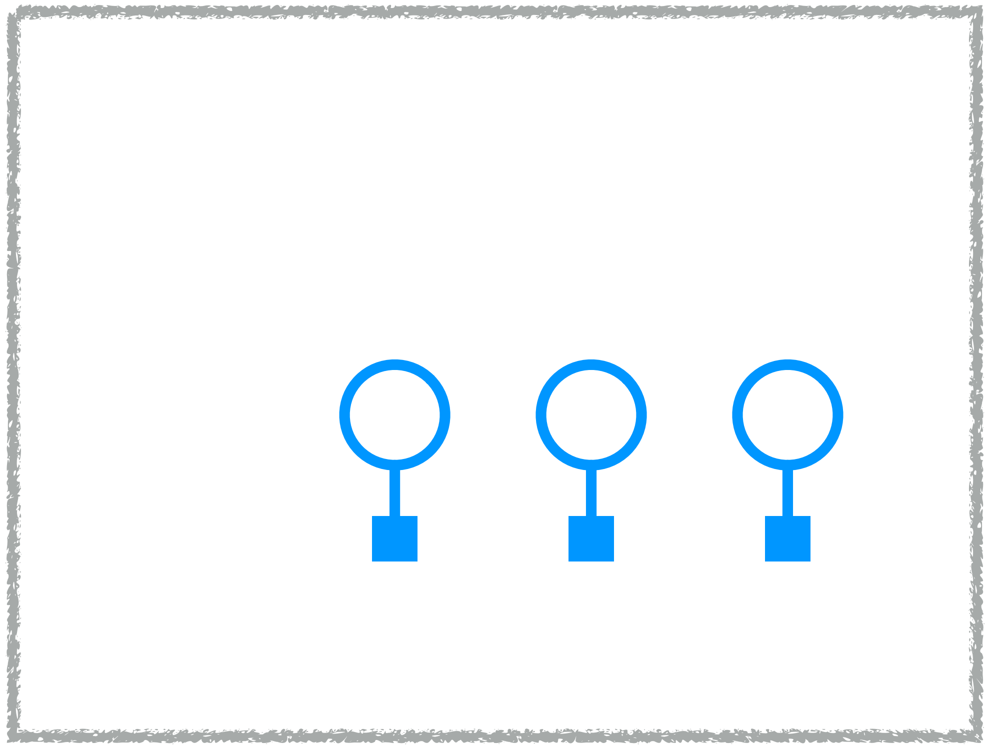


Step 3: sample, compute flat grads

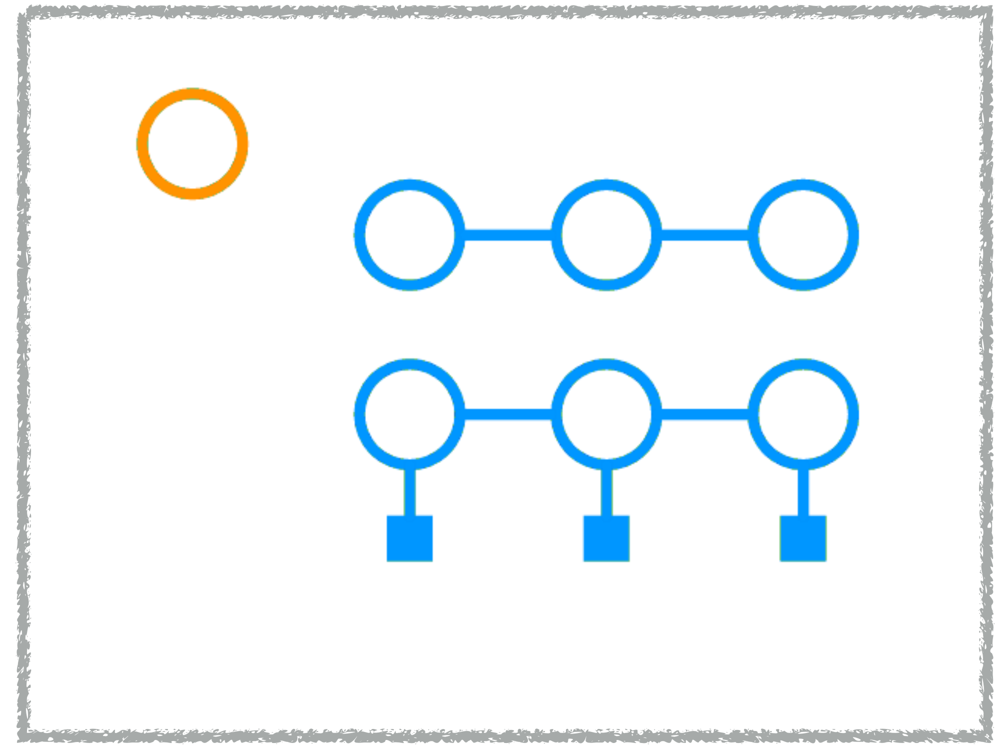


Step 4: compute natural gradient

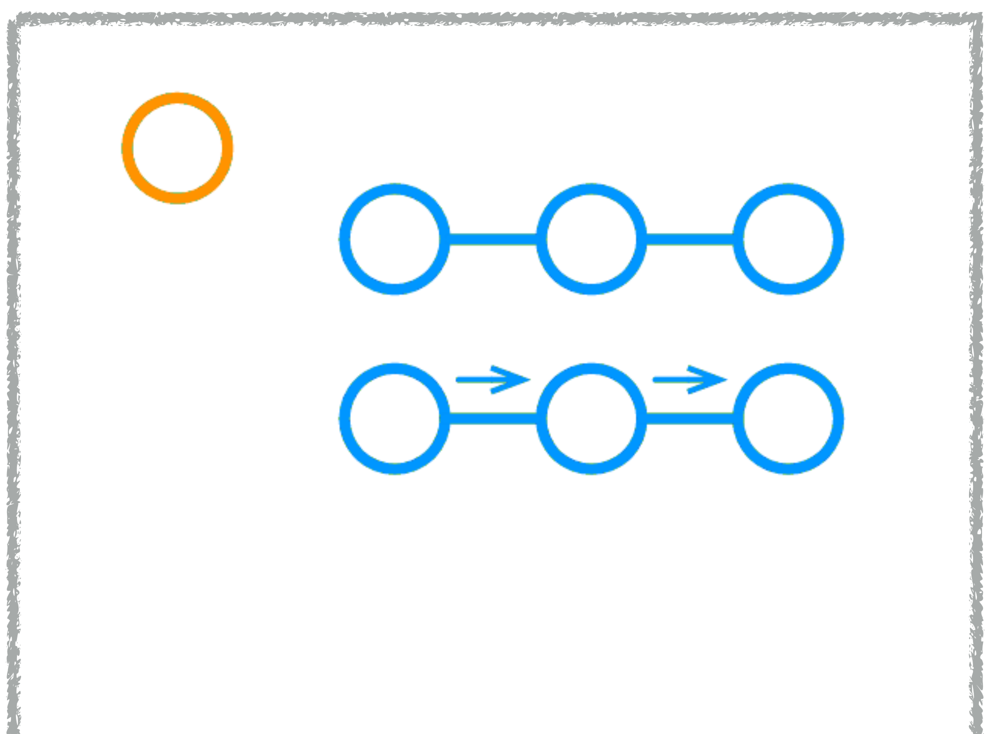
Step 1: apply recognition network



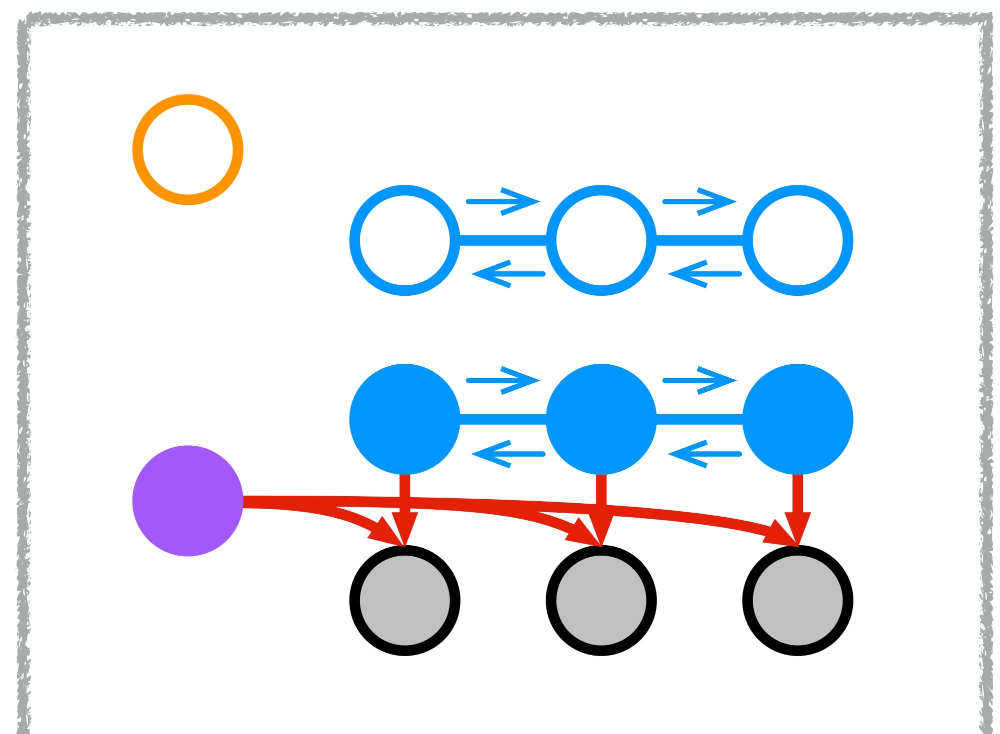
Step 2: run fast PGM algorithms

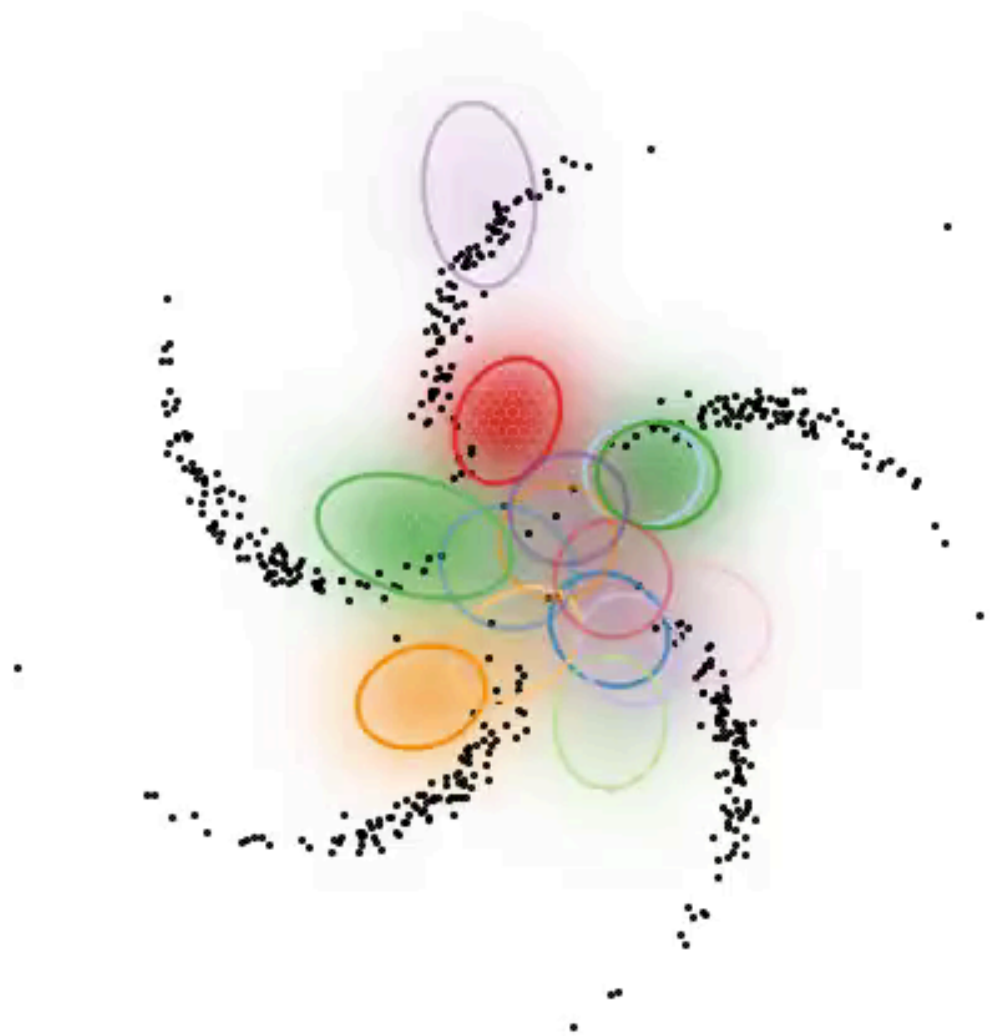


Step 3: sample, compute flat grads

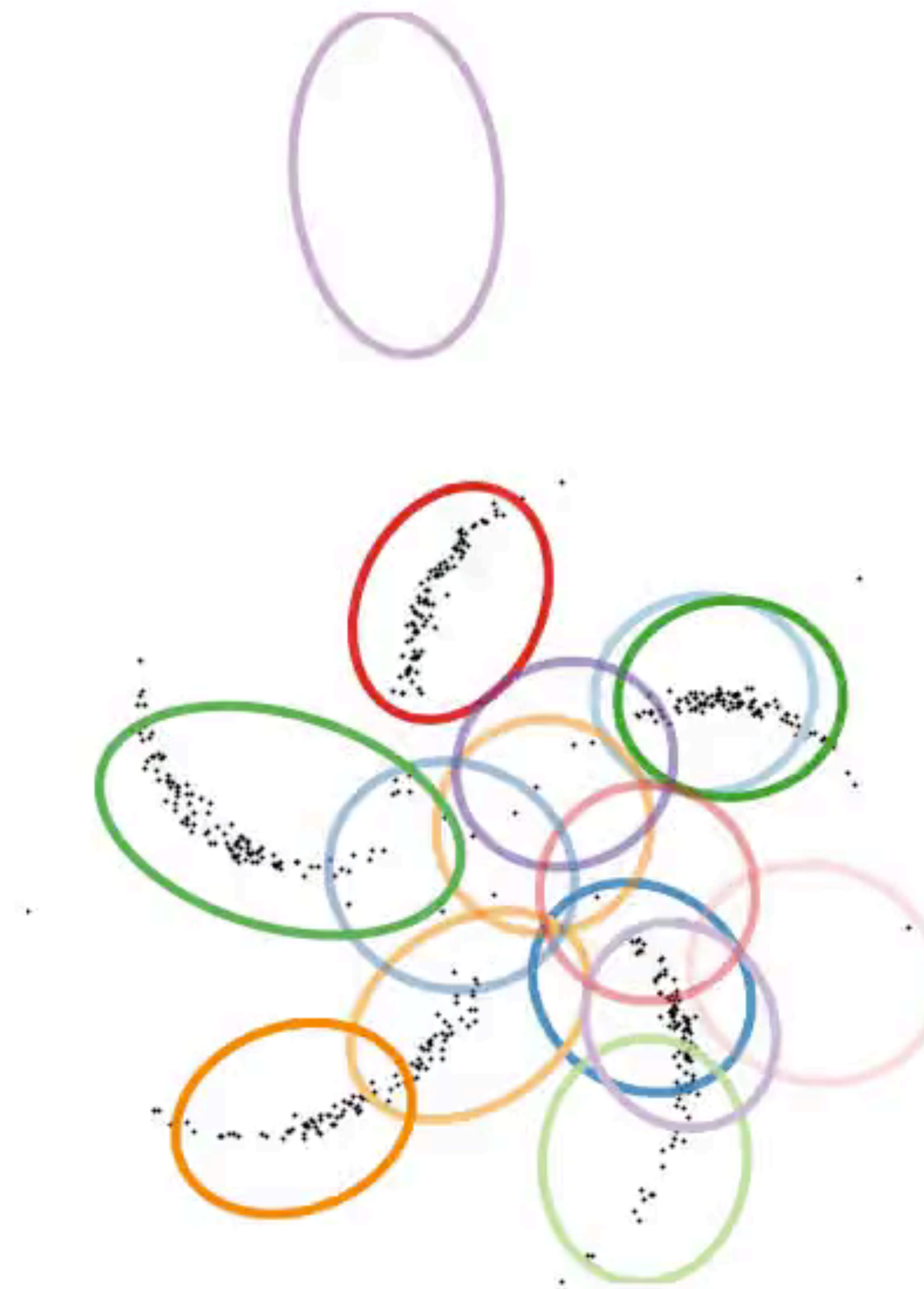


Step 4: compute natural gradient

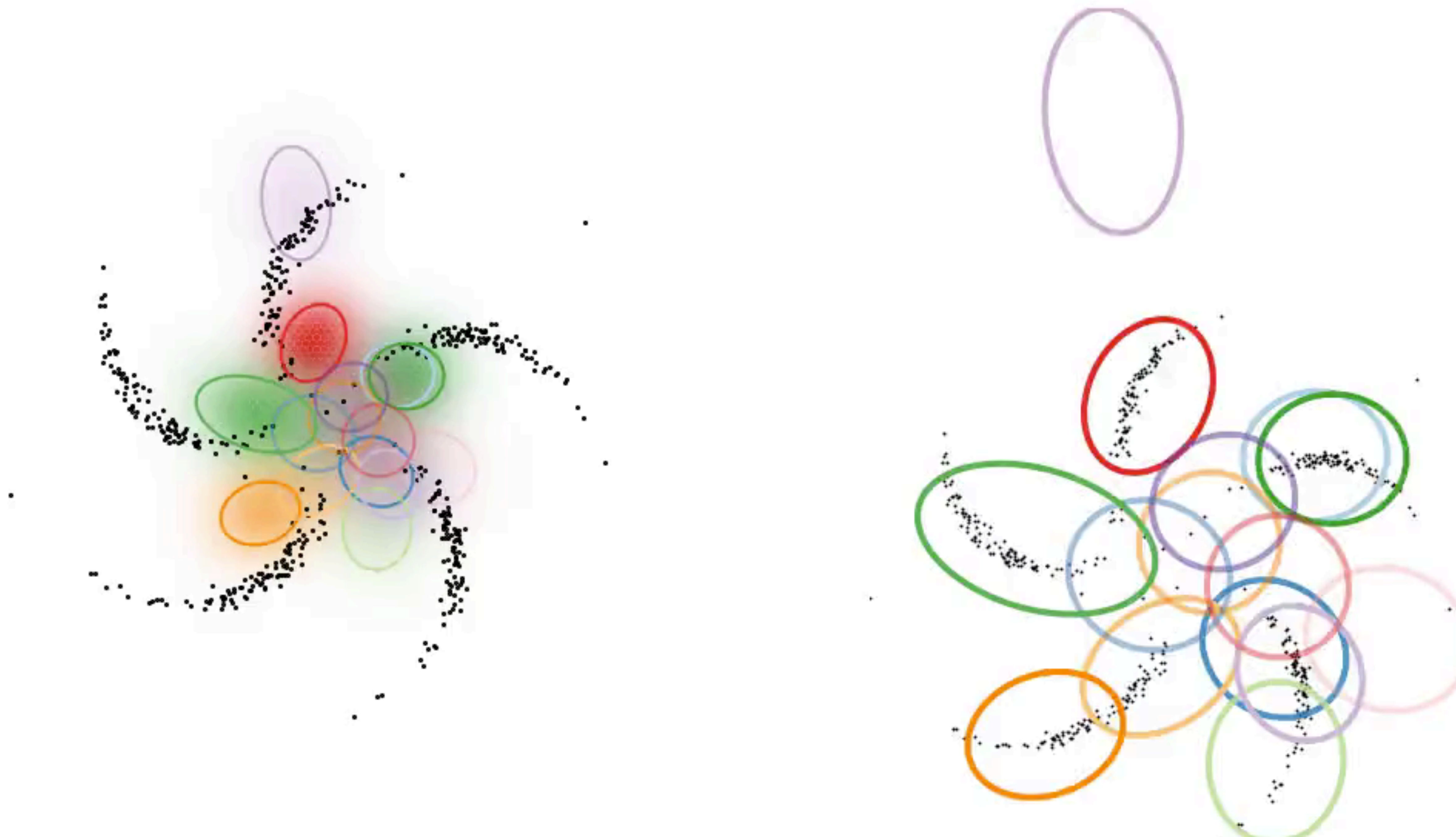




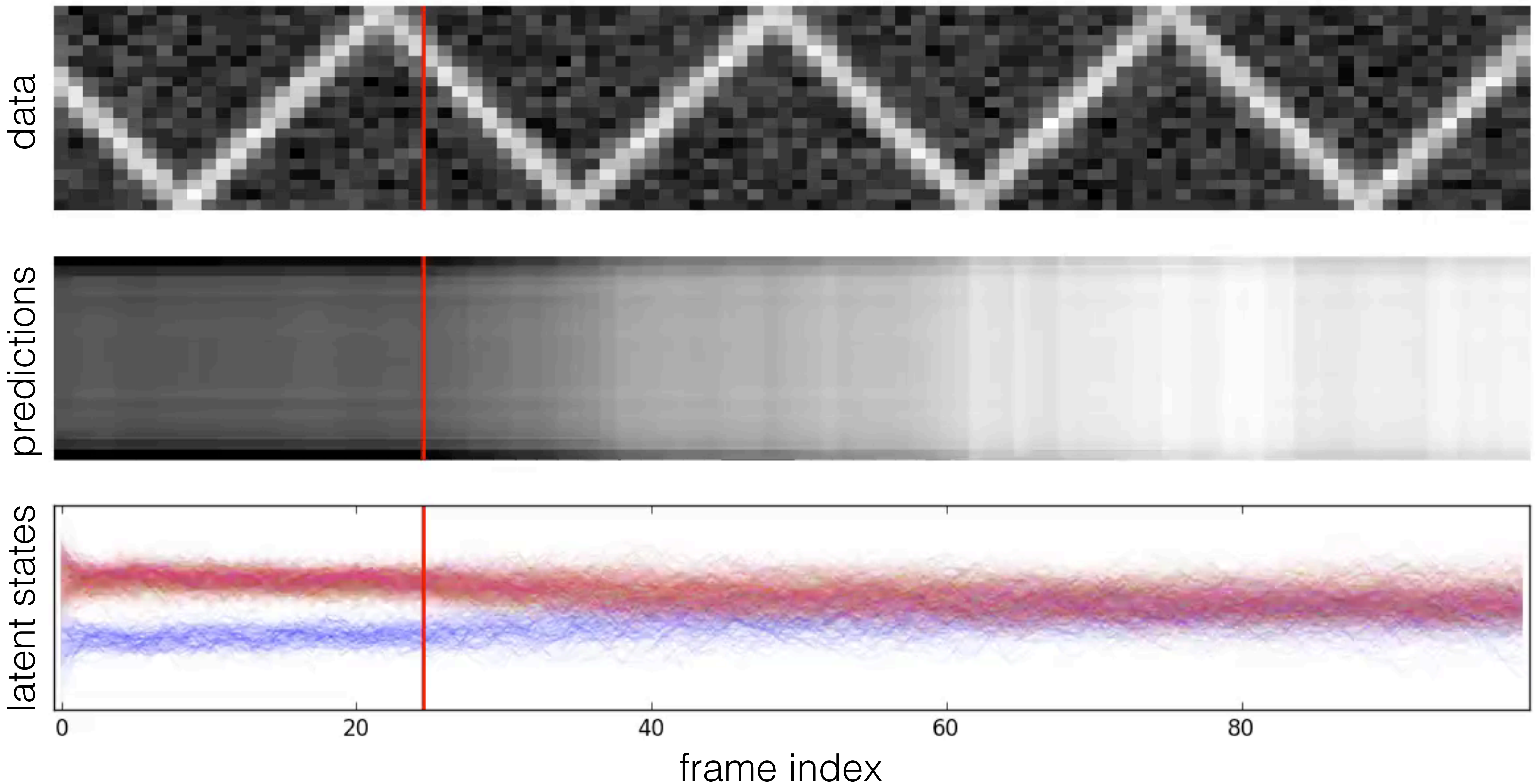
data space



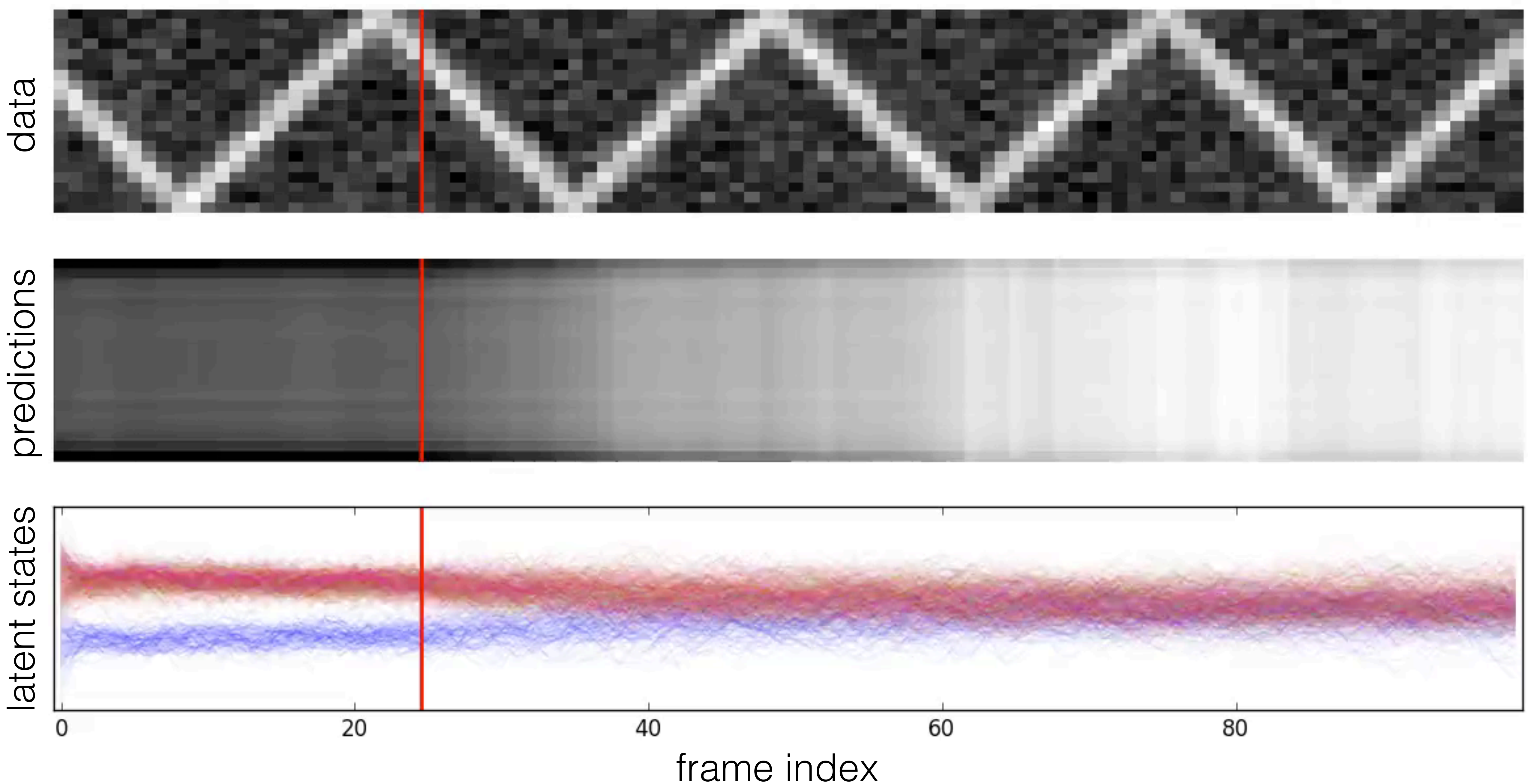
latent space



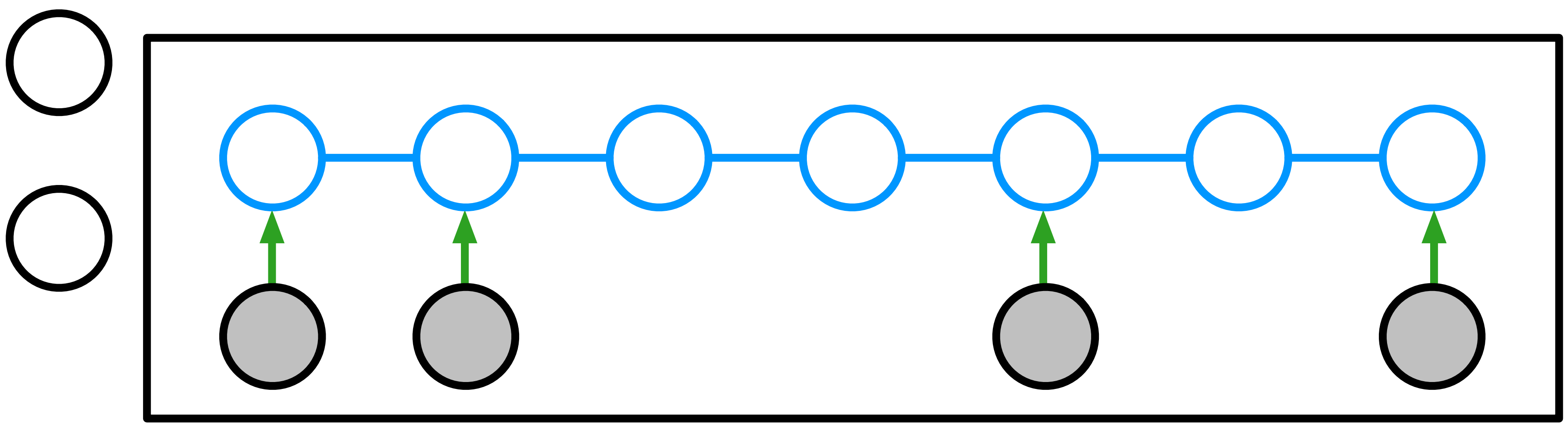
data space latent space



data
predictions
latent states
0
20
40
60
80
frame index

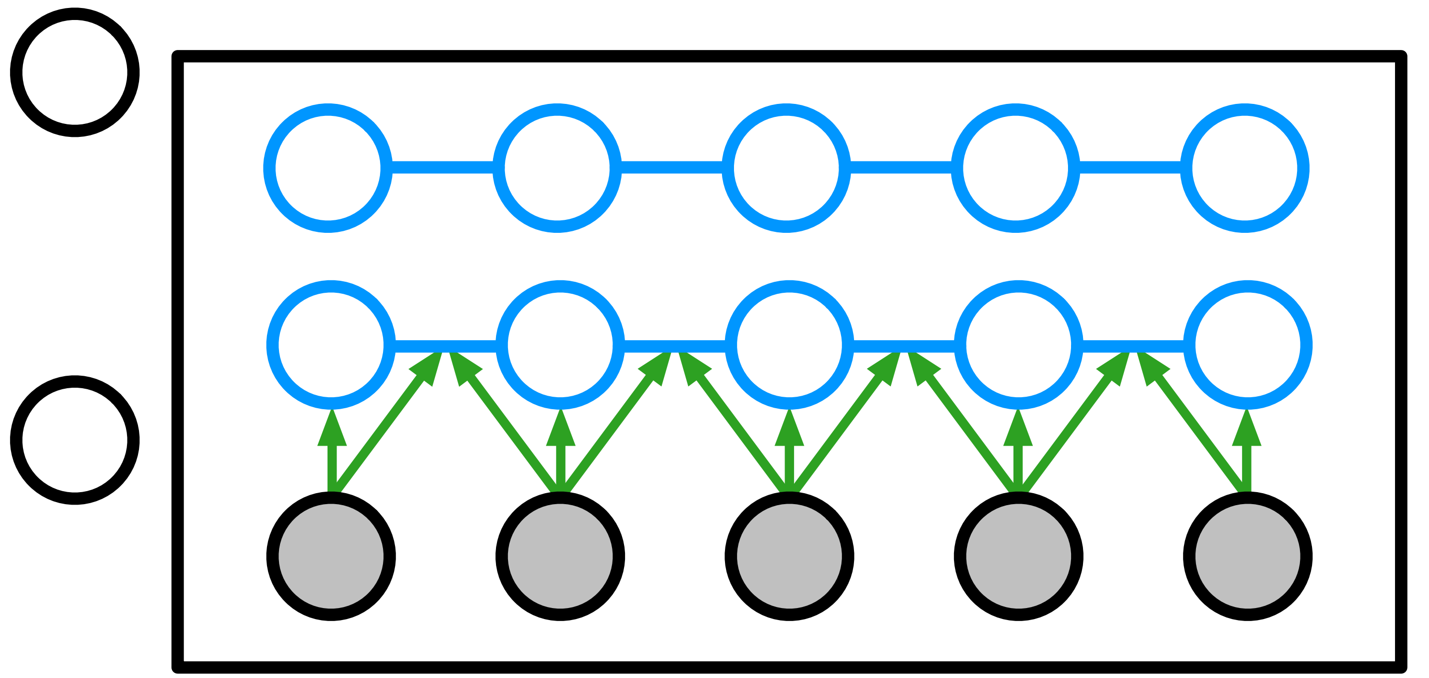

data
predictions
latent states
0
20
40
60
80
frame index

arbitrary inference queries*

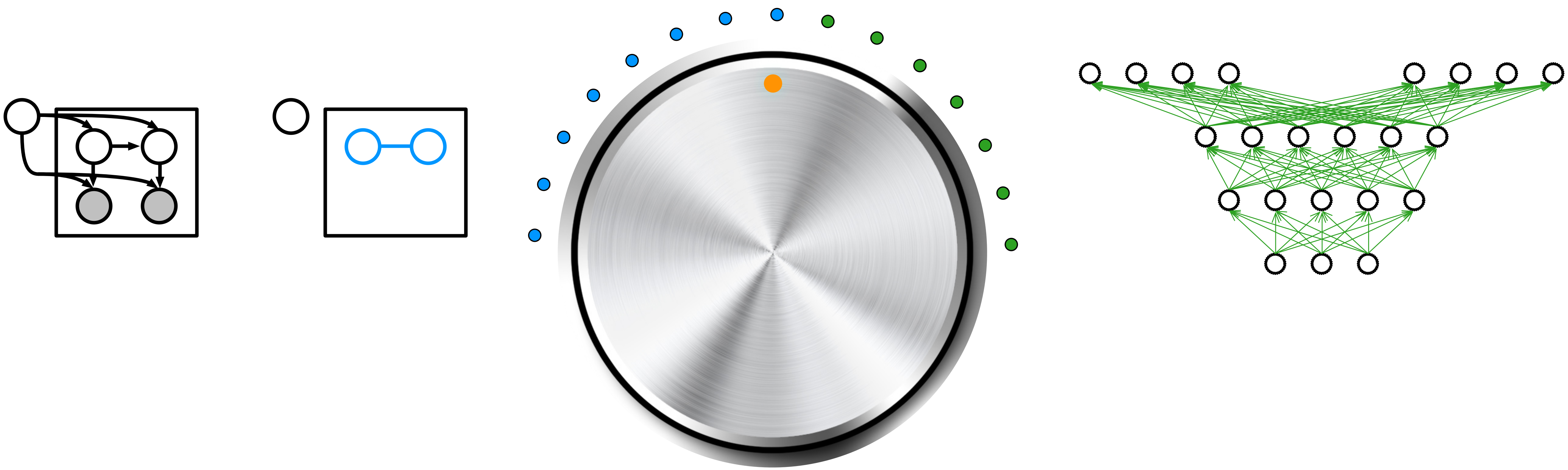


*see next slide

SVAEs can use any inference network architectures

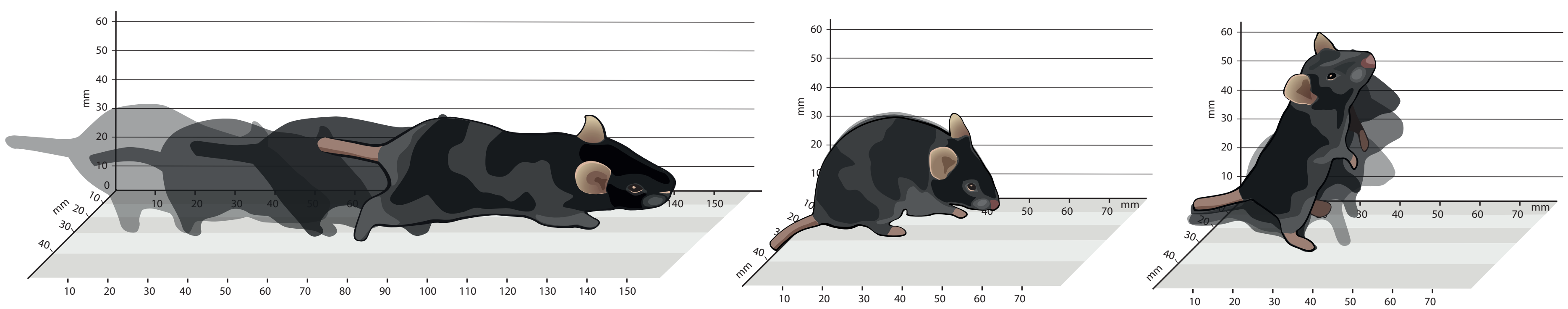


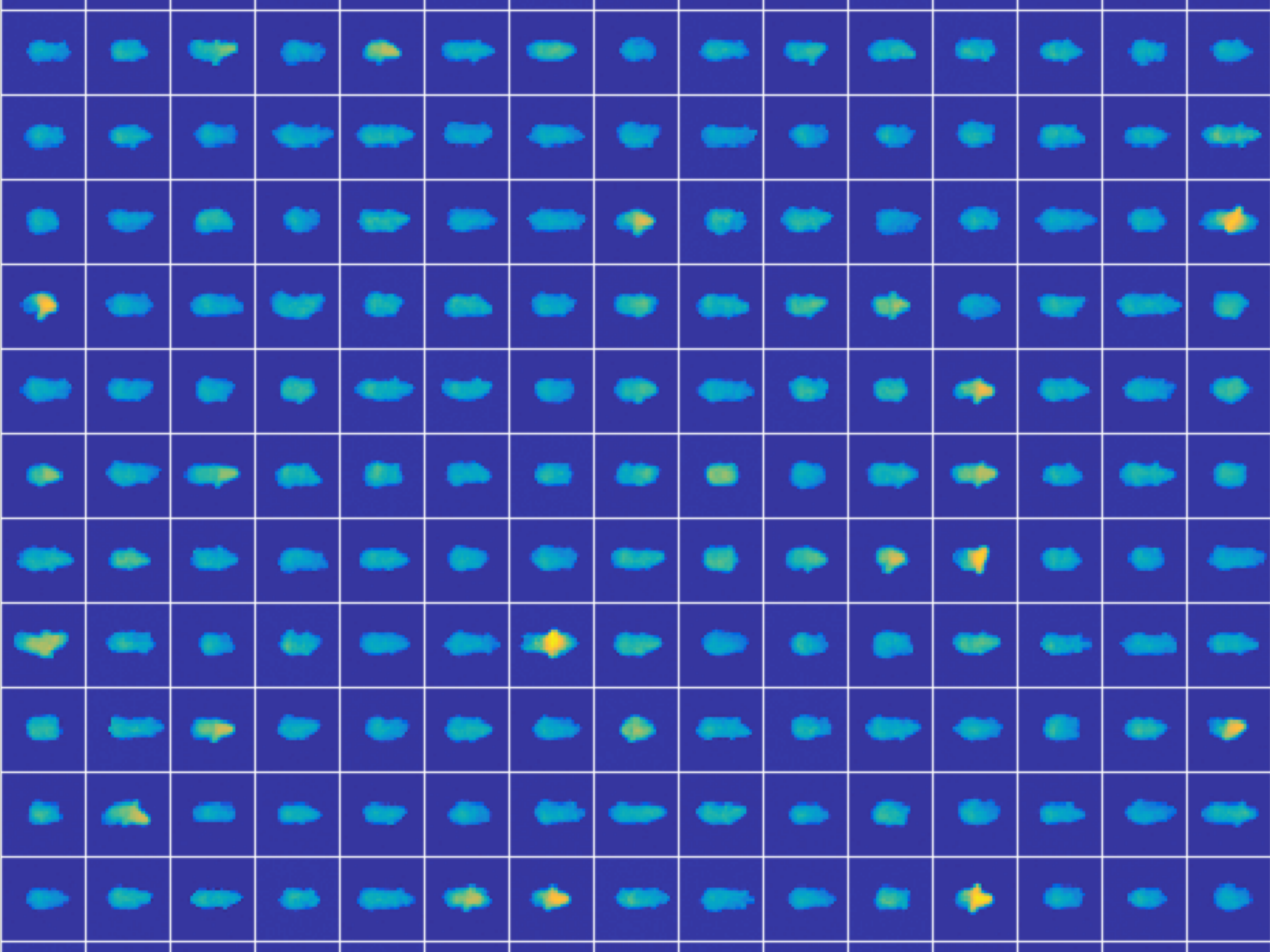
[1] Archer, Park, Buesing, Cunningham, Paninski. Black box variational inference for state space models. ICLR 2016 Workshops.
[2] Gao*, Archer*, Paninski, Cunningham. Linear dynamical neural population models through nonlinear embeddings. NIPS 2016.

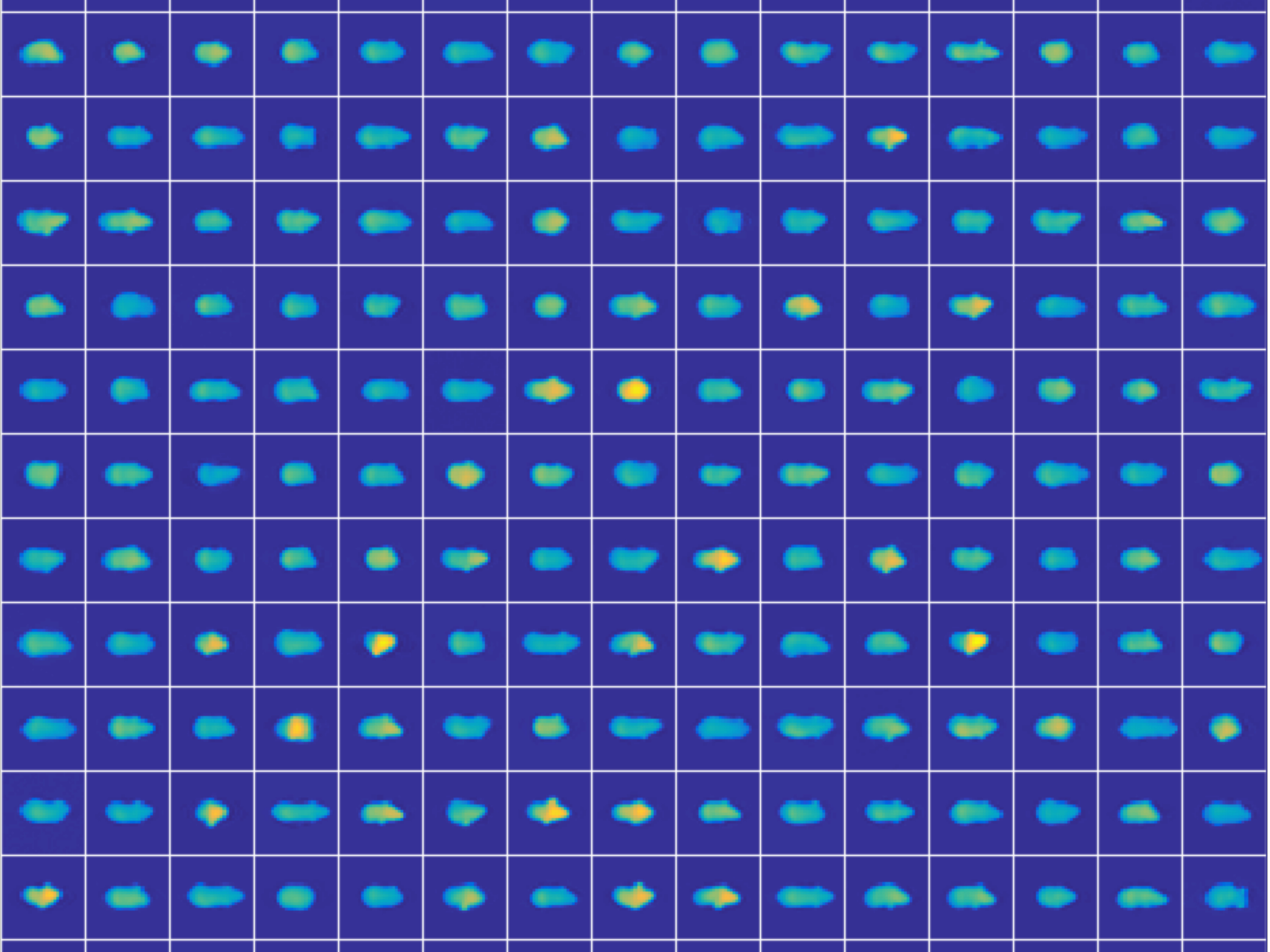


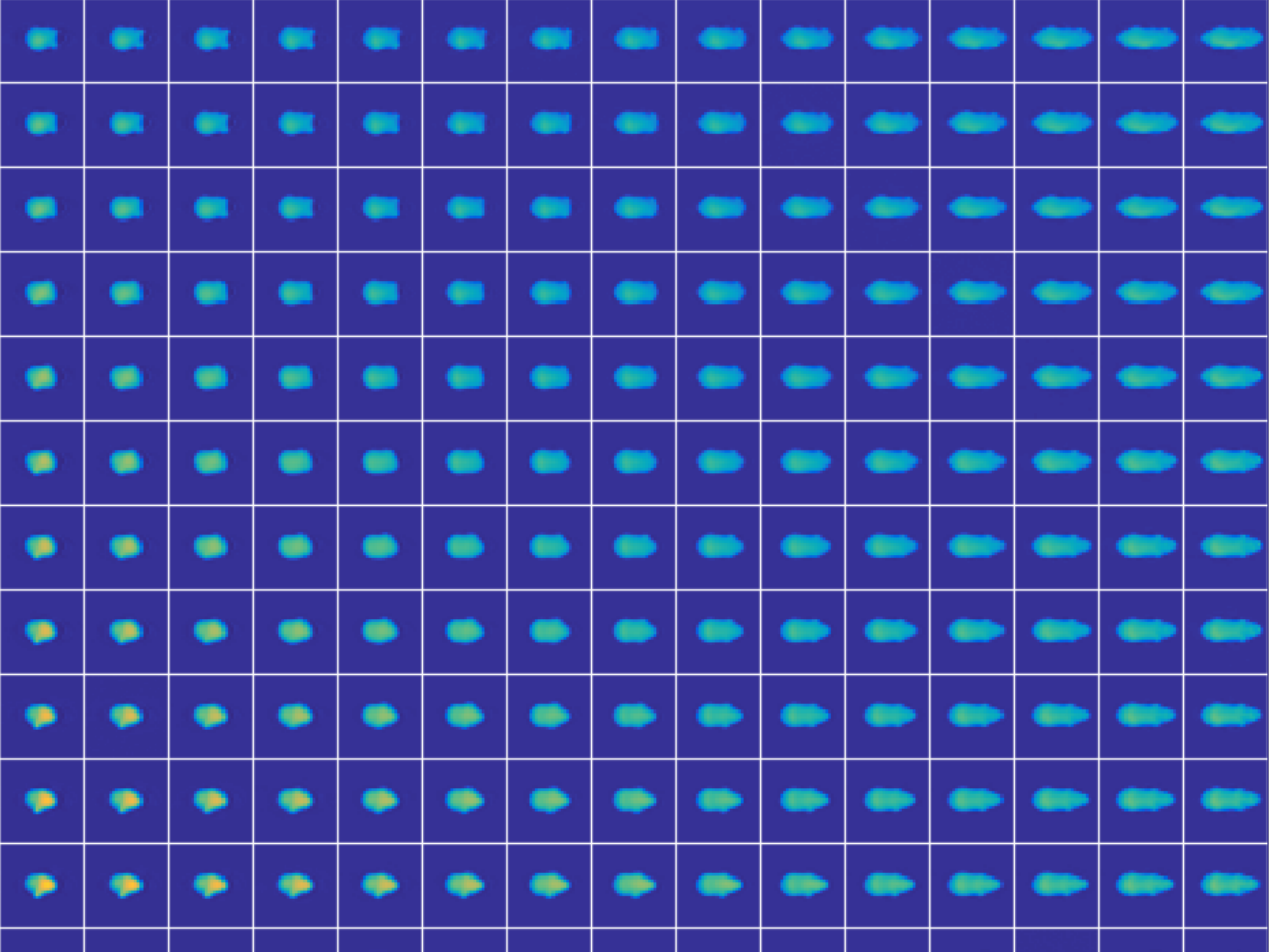
SVAEs

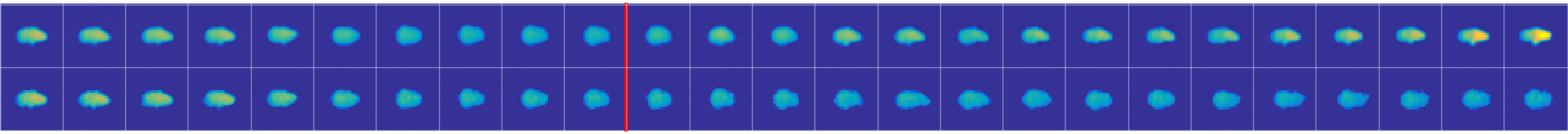
Application: learn syllable representation of behavior from video

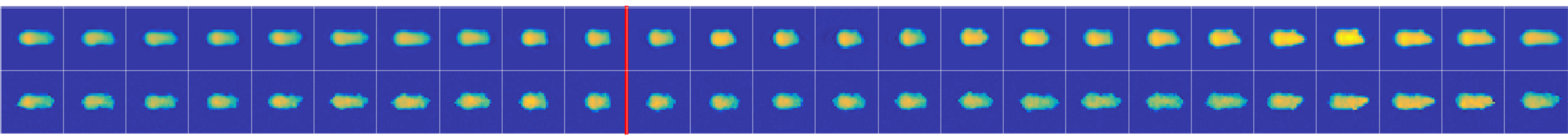


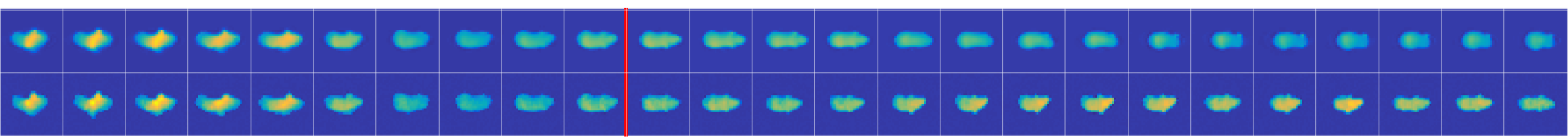


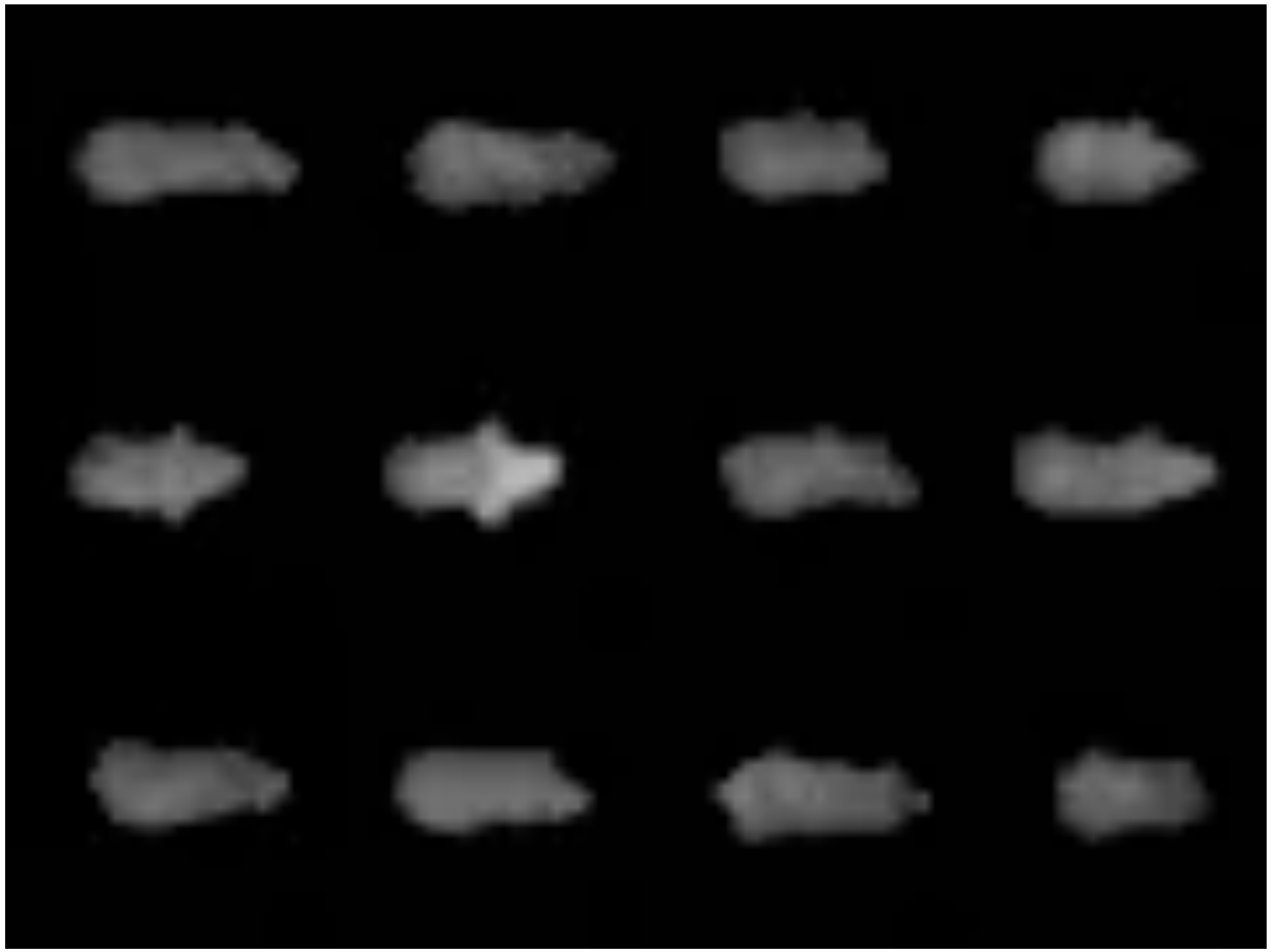




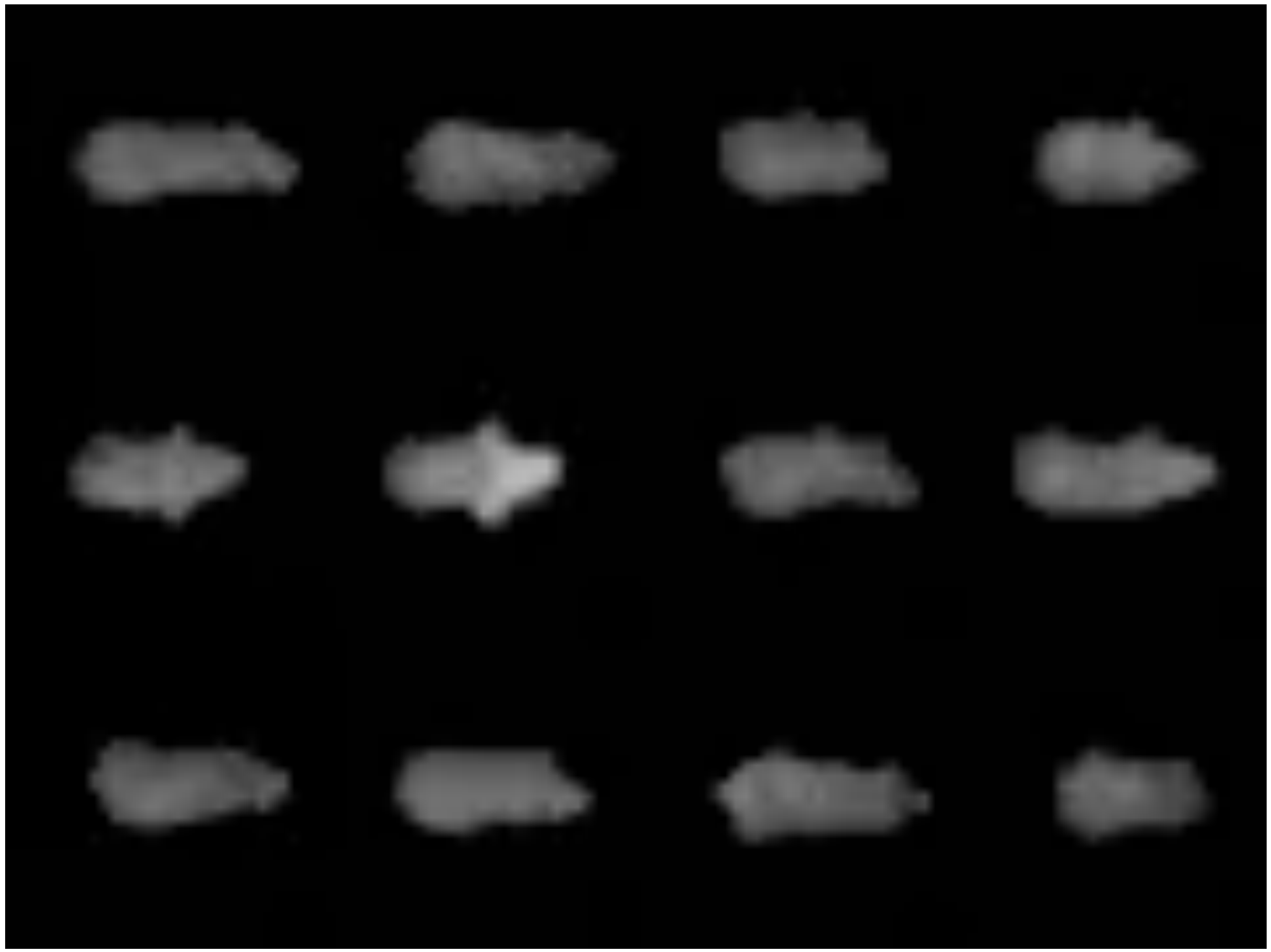




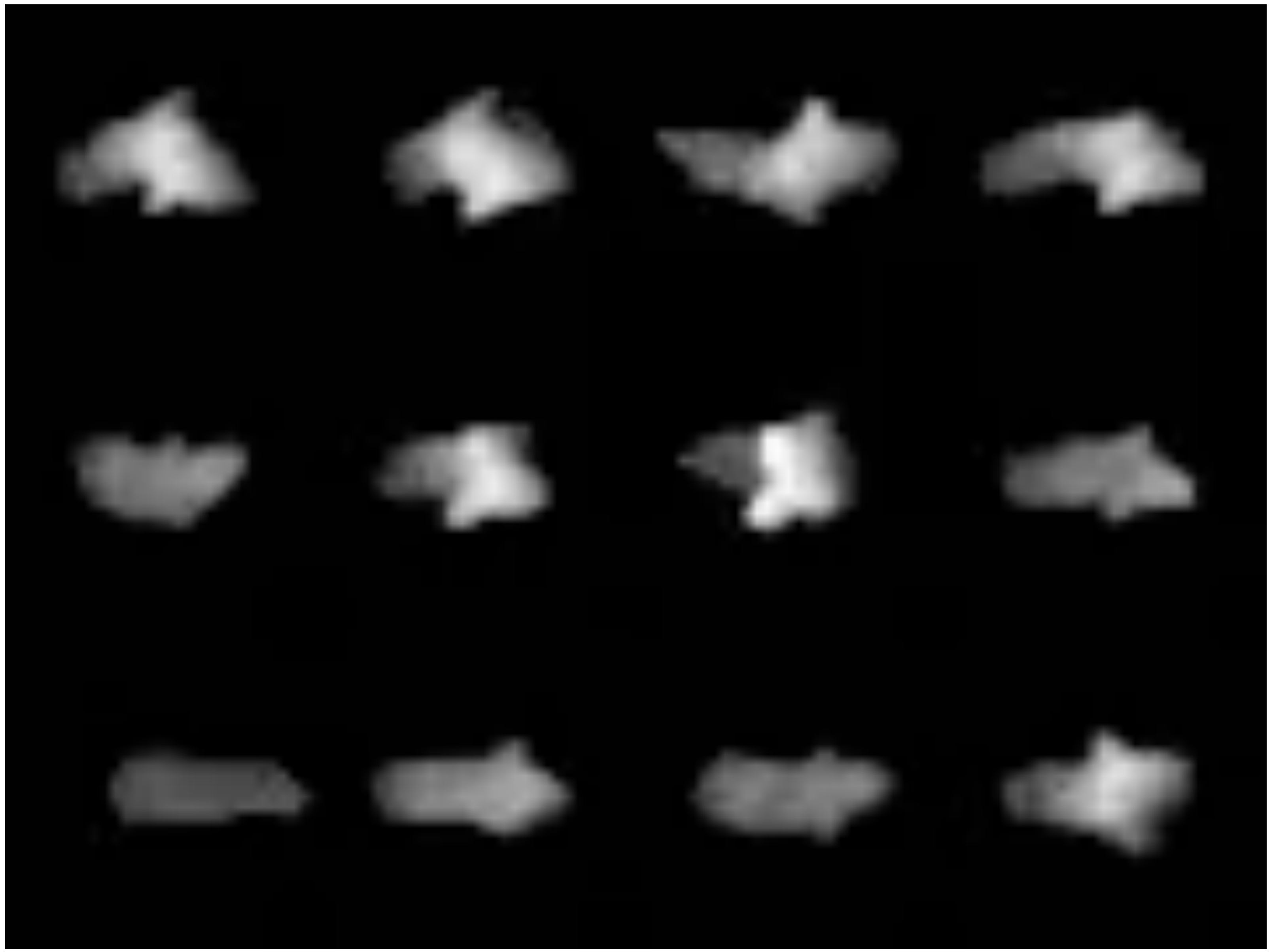




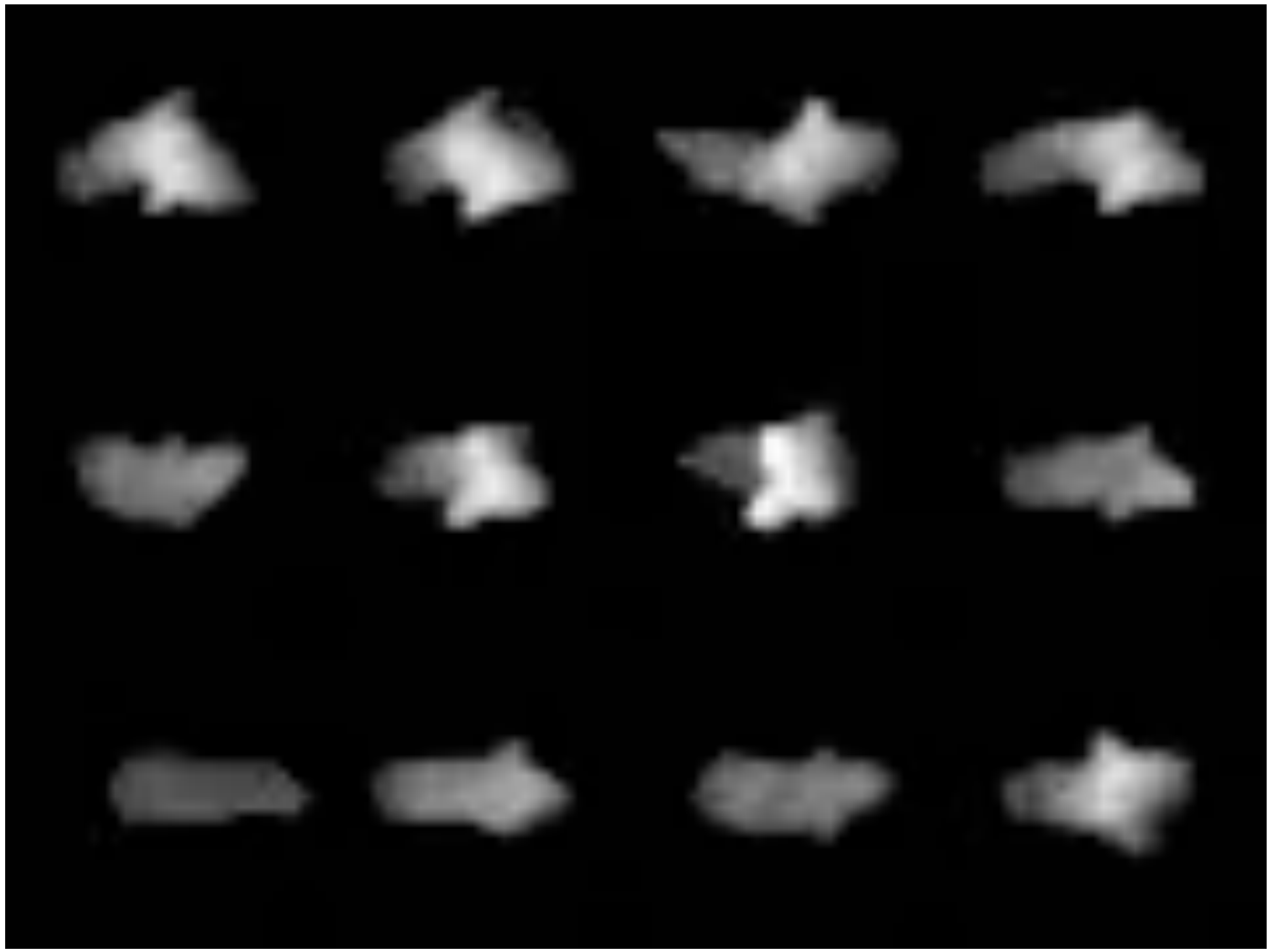
start rear



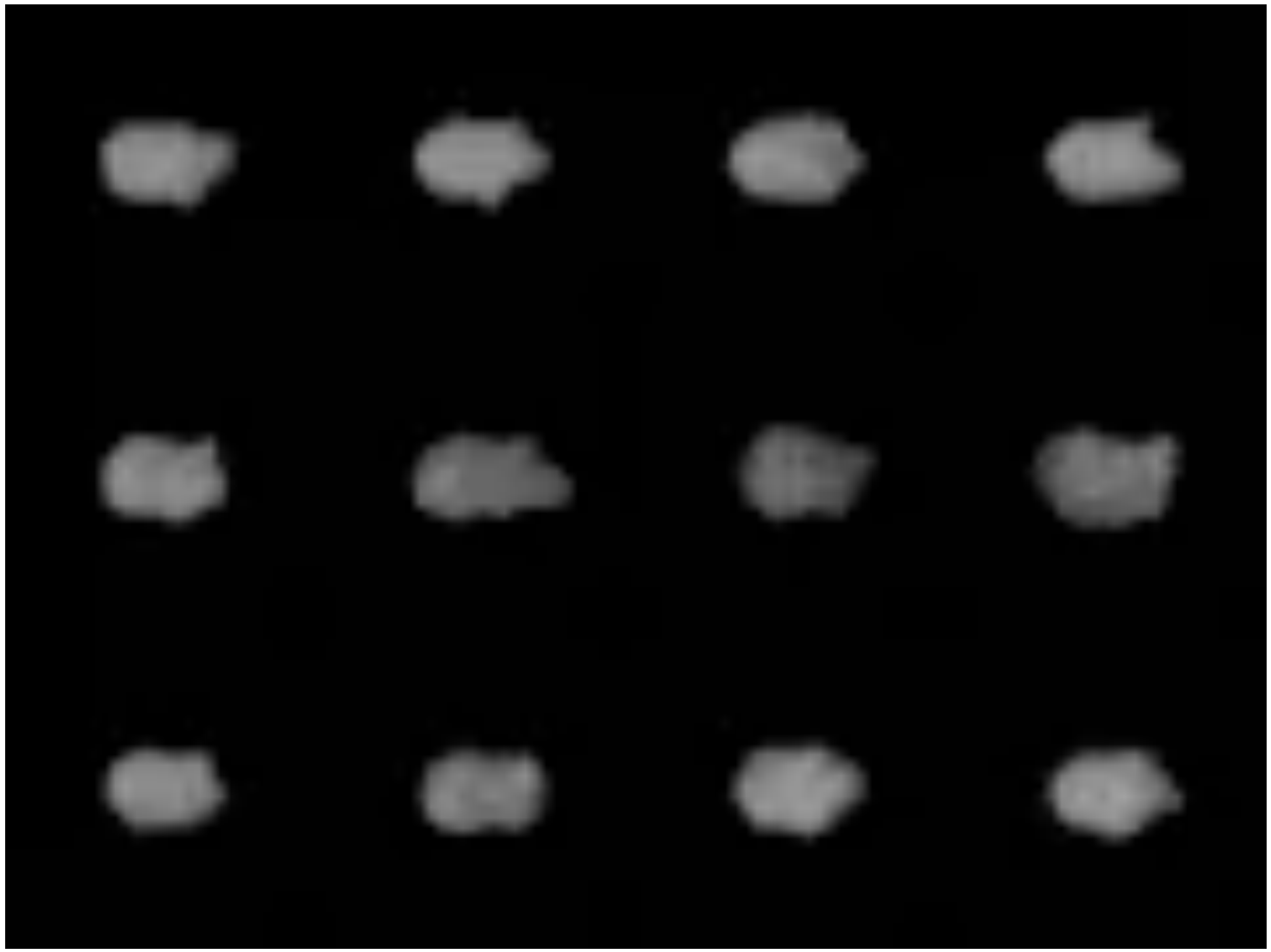
start rear



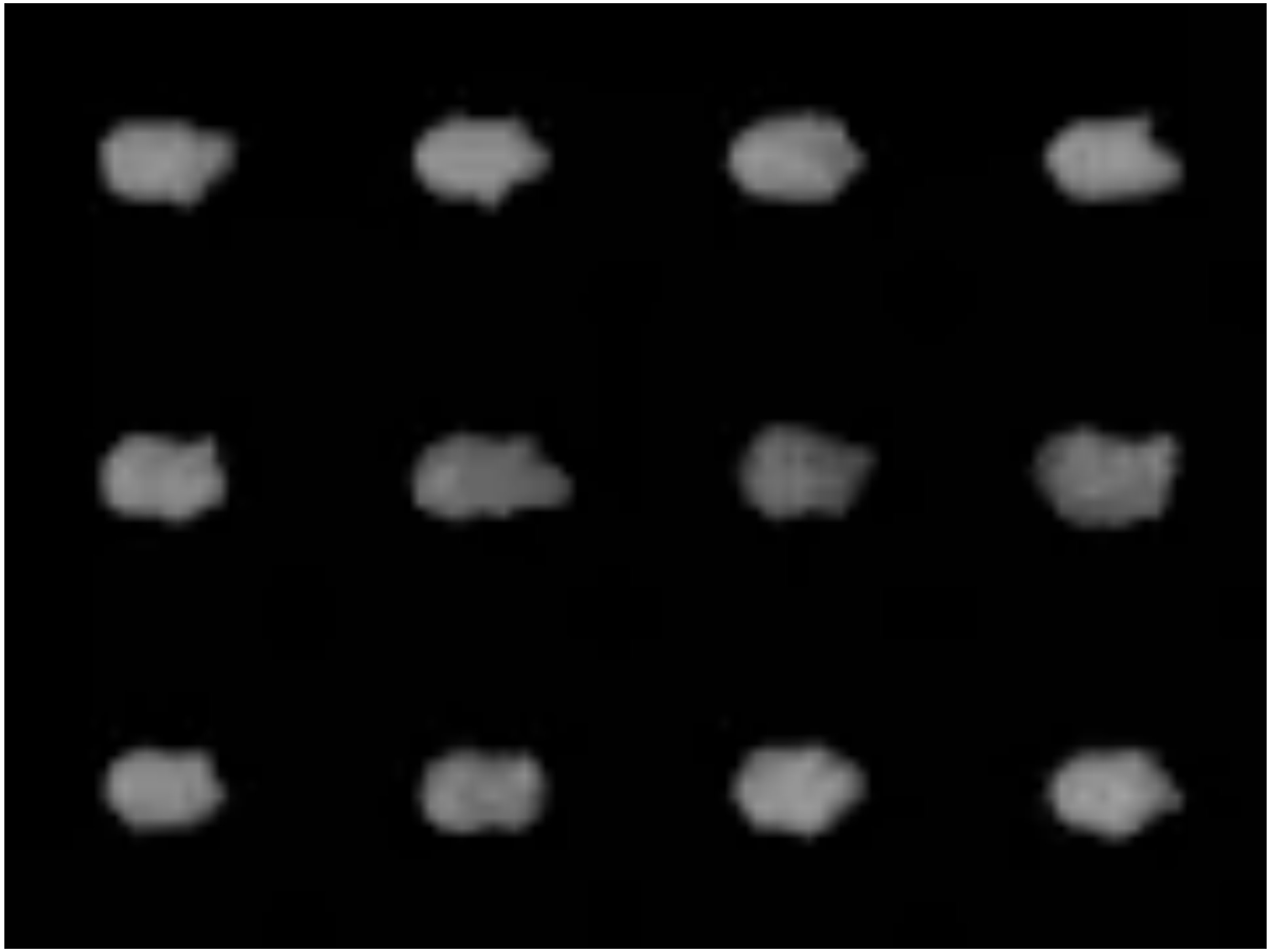
fall from rear



fall from rear

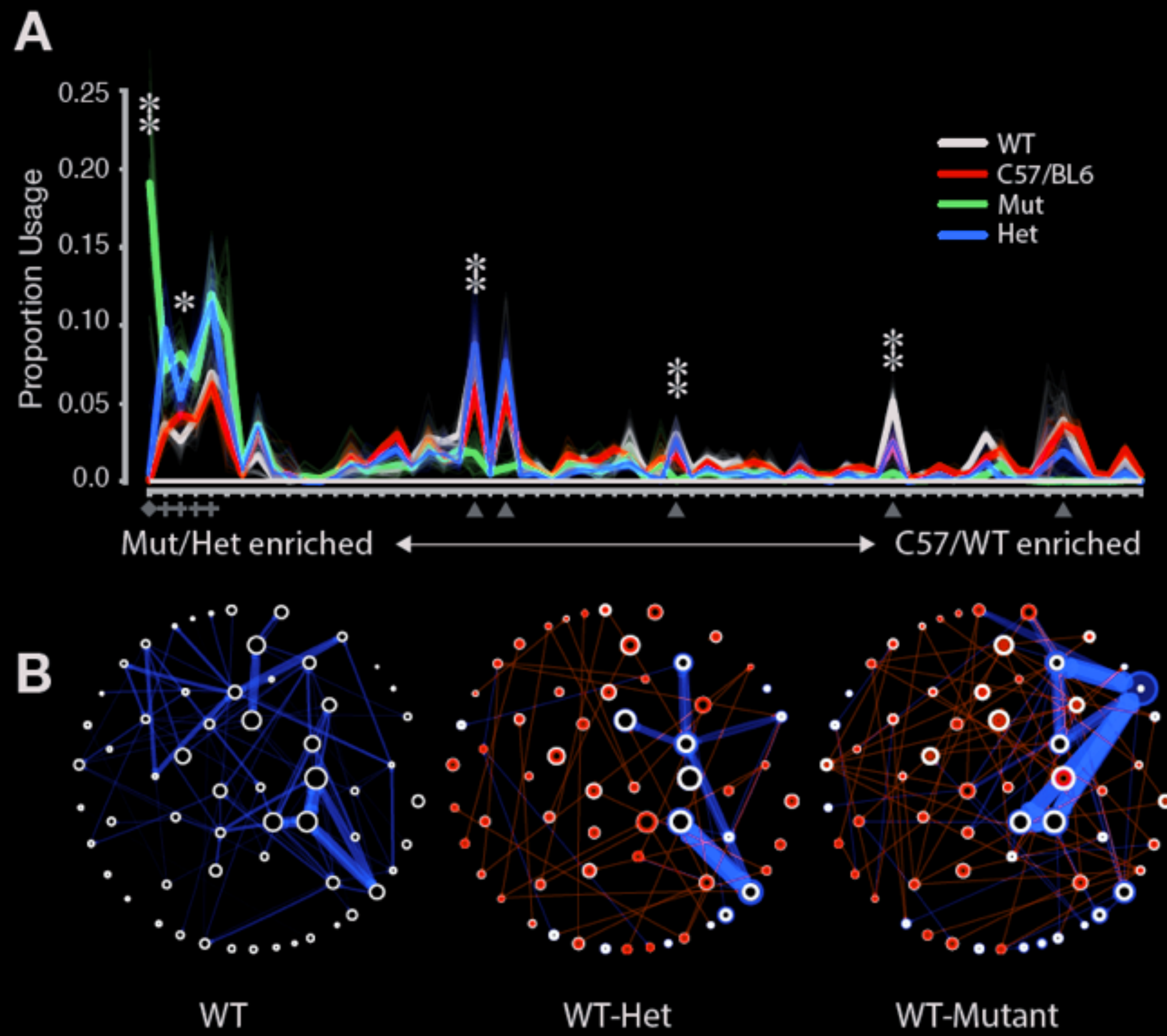


grooming



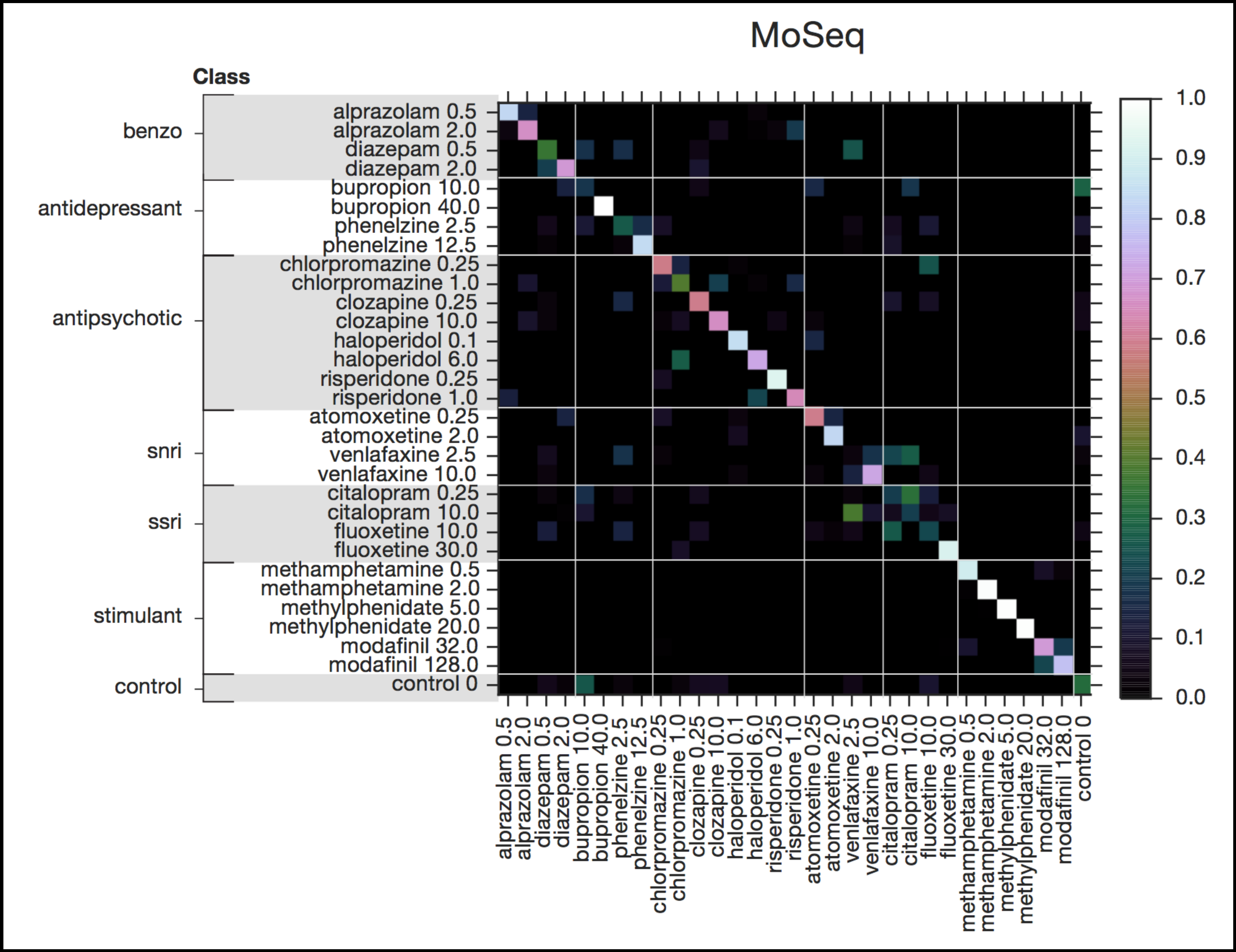
grooming

Discovery of Heterozygous Phenotypes in Ror1b Mice



Alexander Wiltschko, **Matthew Johnson**, et al., Neuron 2015.

… and high and low doses of each drug



from Alex Wiltschko preprint

Goals

Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen

Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen
2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory

Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen
2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory
3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation

Goals

1. Motivate why PGMs + DNNs are a revolution waiting to happen
2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory
3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation
4. Make SVAEs and related PGM + DNN architectures super obvious so that you can invent better ones

Goals

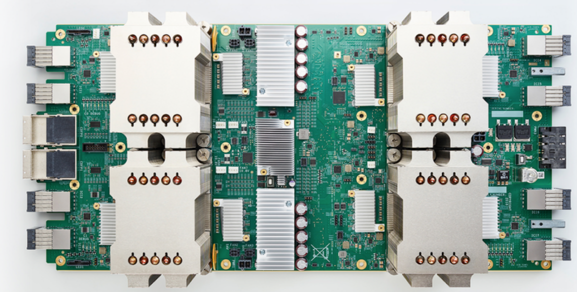
1. Motivate why PGMs + DNNs are a revolution waiting to happen
2. Survey the fundamentals of PGMs and exponential families so that you have a broad view of the territory
3. Show how to unify many models and algorithms in a framework that lets you leverage automatic differentiation
4. Make SVAEs and related PGM + DNN architectures super obvious so that you can invent better ones

Non-goals

1. Cover the recent literature on PGMs + DNNs
2. Unpack all the technical details

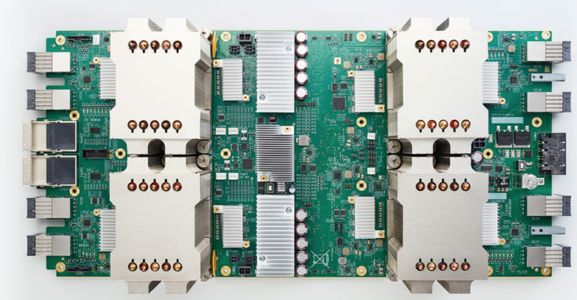
What is JAX?

```
import jax.numpy as np
from jax import jit, grad, vmap

def predict(params, inputs):
  for W, b in params:
    outputs = np.dot(inputs, W) + b
    inputs = np.tanh(outputs)
  return outputs

def loss(params, batch):
  inputs, targets = batch
  preds = predict(params, inputs)
  return np.sum((preds - targets) ** 2)


gradient_fun = jit(grad(loss))
perexample_grads = jit(vmap(grad(loss), (None, 0)))
```



What is JAX?

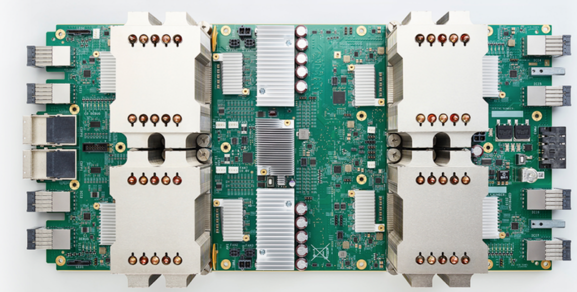
```python
import jax.numpy as np
from jax import jit, grad, vmap

def predict(params, inputs):
  for W, b in params:
    outputs = np.dot(inputs, W) + b
    inputs = np.tanh(outputs)
  return outputs

def loss(params, batch):
  inputs, targets = batch
  preds = predict(params, inputs)
  return np.sum((preds - targets) ** 2)
```



```python
gradient_fun = jit(grad(loss))
perexample_grads = jit(vmap(grad(loss), (None, 0)))
```

What is JAX?

```python
import jax.numpy as np
from jax import jit, grad, vmap

def predict(params, inputs):
  for W, b in params:
    outputs = np.dot(inputs, W) + b
    inputs = np.tanh(outputs)
  return outputs

def loss(params, batch):
  inputs, targets = batch
  preds = predict(params, inputs)
  return np.sum((preds - targets) ** 2)
```



```python
gradient_fun = jit(grad(loss))
perexample_grads = jit(vmap(grad(loss), (None, 0)))
```

JAX is an extensible system for
composable function transformations
of Python + NumPy code.

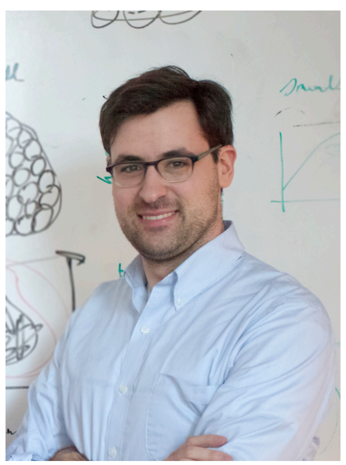
Composing graphical models with neural networks like chocolate and peanut butter

https://youtu.be/O7oD_oX-Gio

or

Graphical models and exponential families in the age of differentiable programming

David Duvenaud



Alex Wiltchko

Matthew D. Hoffman

Dustin Tran

Scott Linderman

Sandeep Robert Datta

Ryan P. Adams

Matthew J Johnson (mattjj@google.com)
July 22 2019 @ UAI 2019