

# Generative modelling with deep latent variable models

**Marco Fraccaro**



# Outline

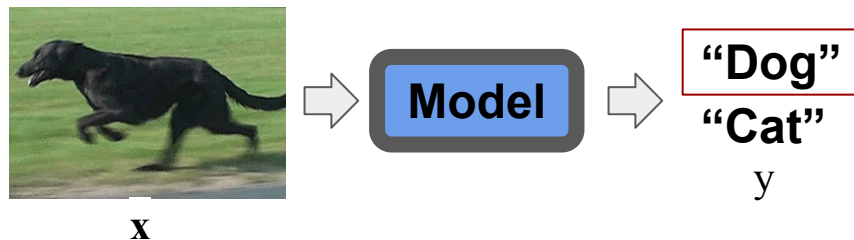
1. Introduction to deep generative models
2. Latent variable models
3. Variational Auto-Encoders
4. ---- *VAE exercise* ---
5. Advance topics in VAE research
6. Extending VAEs to sequential data

# Introduction to deep generative models

# Machine learning

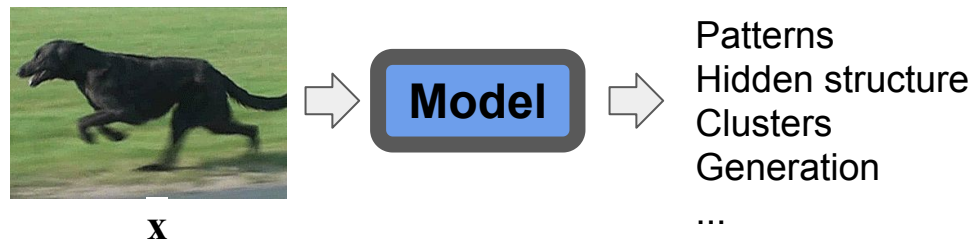
Build computers that can *learn* from data without being explicitly programmed

## Supervised learning



- Focuses on predictive tasks (e.g. classifiers)
- Models  $p(y|x)$
- It requires datasets containing labelled data
  - Labelled data is scarce and expensive to obtain

## Unsupervised learning



- Focuses on descriptive tasks
- Models  $p(x)$
- It does not require labelled data
  - We can exploit large datasets of unlabelled data

# Generative modelling

- A branch of unsupervised learning
- Given a training dataset, the goal is to build a model that is able to *generate* data that comes from the data distribution  $p(\mathbf{x})$ 
  - E.g. given a dataset of faces, we learn how to generate realistic new faces
- It's not all about generating pretty pictures. Generative models help understand the data: *"What I cannot create, I do not understand"*, Richard Feynman.



Ian Goodfellow

@goodfellow\_ian

Follow



4.5 years of GAN progress on face generation. [arxiv.org/abs/1406.2661](https://arxiv.org/abs/1406.2661)  
[arxiv.org/abs/1511.06434](https://arxiv.org/abs/1511.06434)  
[arxiv.org/abs/1606.07536](https://arxiv.org/abs/1606.07536)  
[arxiv.org/abs/1710.10196](https://arxiv.org/abs/1710.10196)  
[arxiv.org/abs/1812.04948](https://arxiv.org/abs/1812.04948)



4:40 PM - 14 Jan 2019

1,415 Retweets 3,761 Likes



# The data distribution $p(\mathbf{x})$

We use the training dataset to learn a probability distribution over the data,  $p(\mathbf{x})$ .

- In other words, we want to learn a complex probability distributions  $p(\mathbf{x})$  given a finite sample of possibly high-dimensional data points  $\mathbf{x}$  drawn from that distribution.

Knowing this probability distribution we are interested in:

1. Sample new data points  $\mathbf{x}^{(s)} \sim p(\mathbf{x})$ , i.e. *generating* from the model
2. Evaluate  $p(\mathbf{x})$  for a test data point
3. Extract a latent representation of the data

Note: for most complex classes of models some of these procedures will be impossible, or will be only possible making approximations

# Generative models considered in this presentation

There are lots of different kinds of generative models, we only focus on the ones that are

1. Flexible, i.e. can model a wide range of data distributions -> deep neural networks
2. Scalable, so that training that can exploit large unlabelled datasets -> SGD on GPUs
3. “Easy” to implement -> deep learning libraries

# Applications: generation of images, speech, music, text



2014



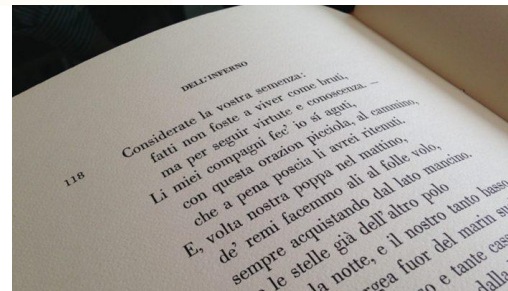
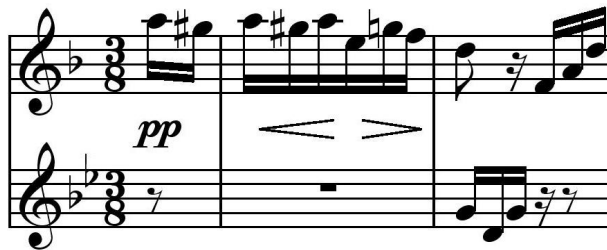
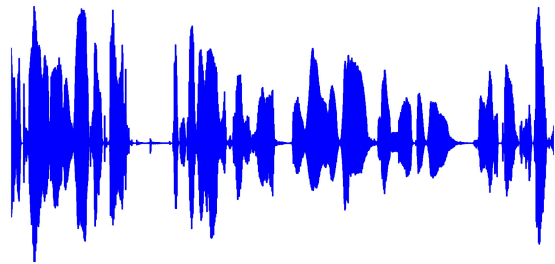
2015



2016



2017



# Applications: neural machine translation

*Conditional* generative model:  $p(\text{target language}|\text{source language})$

Example:

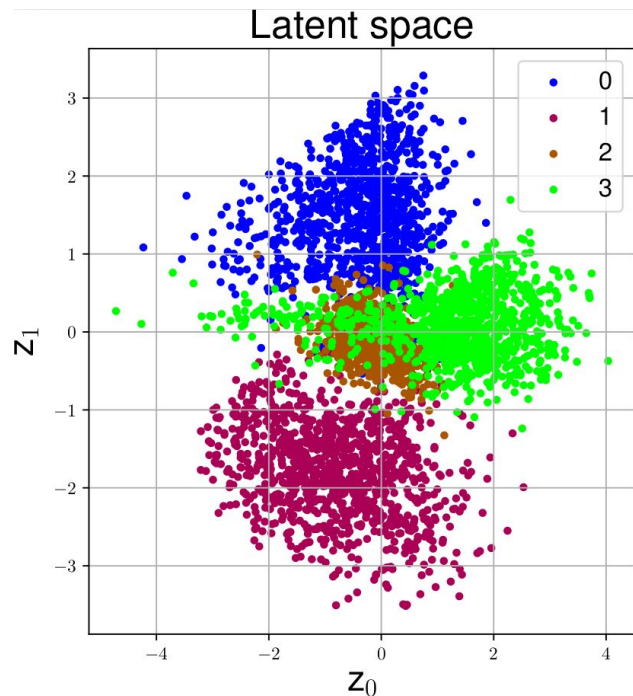


# Applications: semi-supervised learning

In many applications (e.g. healthcare) having unlabelled data is easy, but labelled data is rare or expensive to obtain

- Can we leverage the unlabelled data to improve the performance of supervised methods?
- Idea: use unlabelled data to learn a “good” low-dimensional latent representation of the data, over which we construct a much simpler classifier

Example: latent space of MNIST learned in an unsupervised way shows well-defined class-specific clusters



# Applications: model-based reinforcement learning

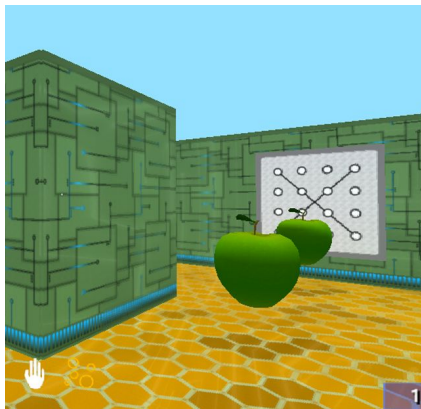


Reinforcement learning: building agents that take actions in an environment to maximize a reward.

Generative models can be used as simulators for *model-based reinforcement learning* (e.g. for planning)

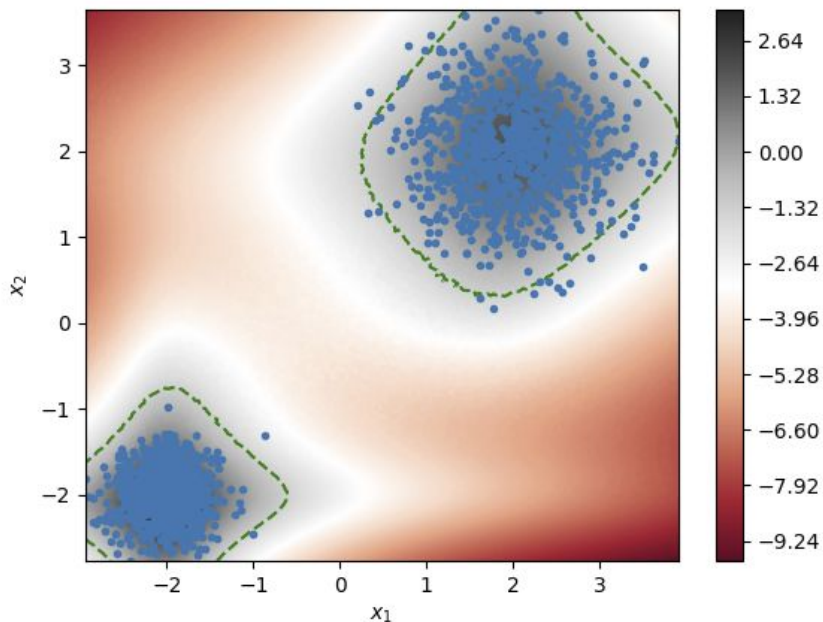
We need **generative models** that can

- build a model of the world in an unsupervised way
- coherently predicting hundreds of time steps in the future
- Imagine the evolution of the world given a sequence of action

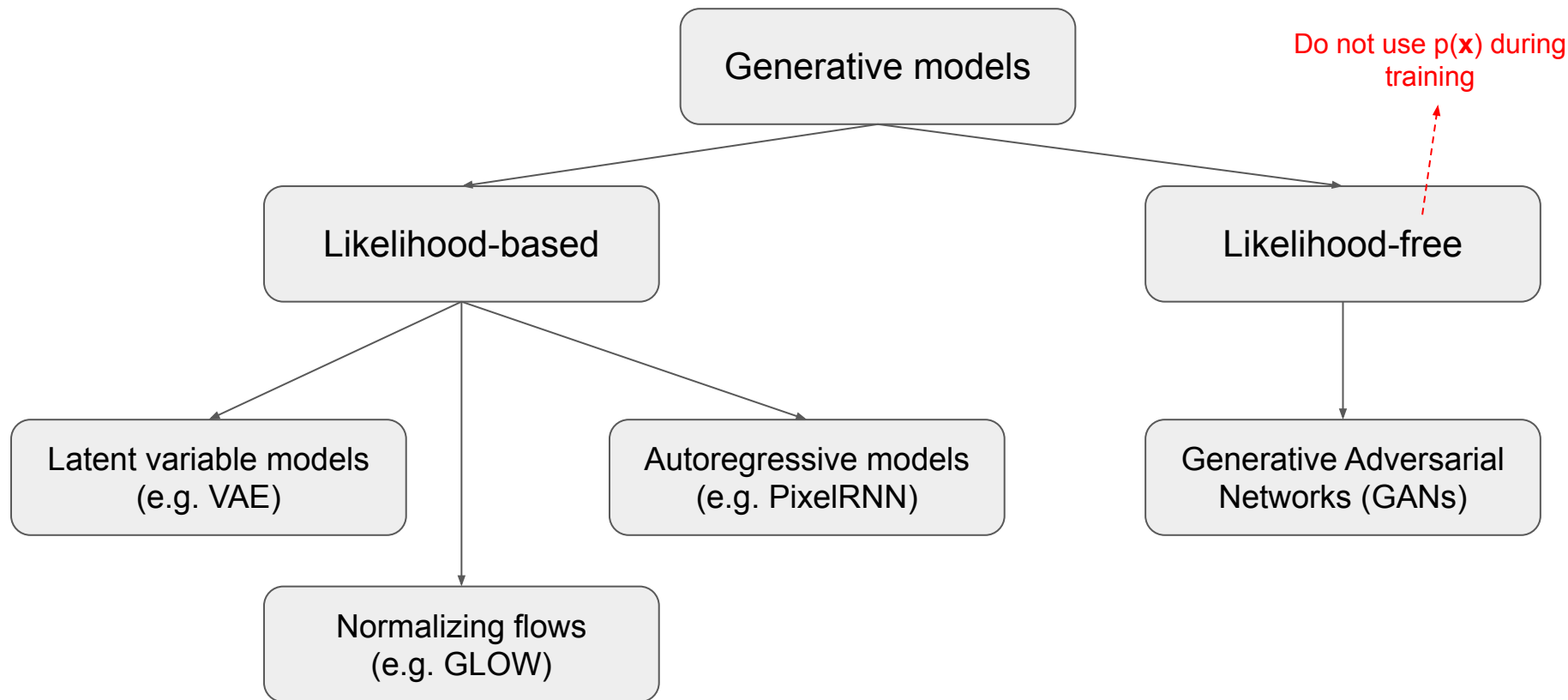


# Applications: anomaly detection

- Real world data contains lot of outliers/anomalies
- Since we know how to evaluate  $p(\mathbf{x})$  for a test data point we can perform unsupervised anomaly detection via density estimation
  - We consider *anomalies* points whose  $p(\mathbf{x})$  is below a certain threshold



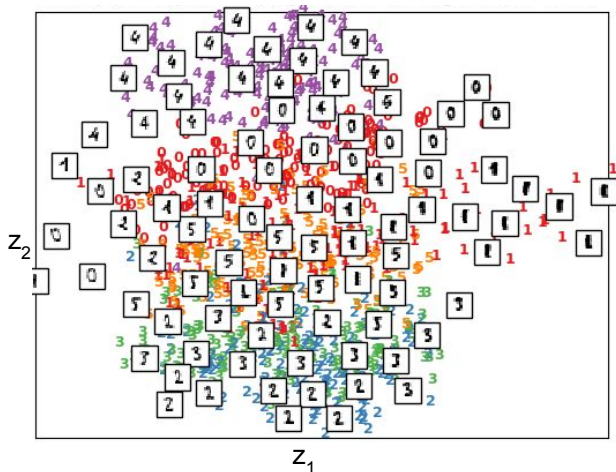
# Taxonomy of generative models



# Likelihood-based generative models

## Latent variable models

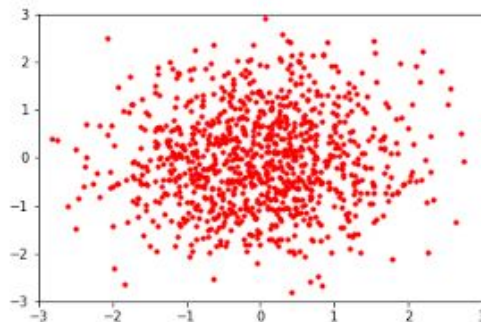
Main idea: introduce a low-dimensional latent variable  $\mathbf{z}$  that models hidden causes in the data-generating process.



[Image source: Scikit learn docs for *Manifold learning with PCA*]

## Normalizing flows

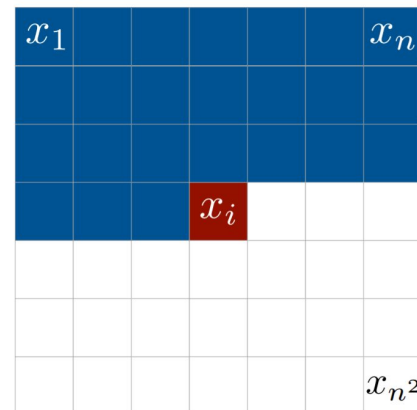
Main idea: we can model complex probability distributions starting from a simple distribution and applying a series of invertible transformations to it.



[Image source: "Normalizing Flows in 100 Lines of JAX" by Eric Jang]

## Autoregressive models

Main idea: build a model which predicts future values from past ones.

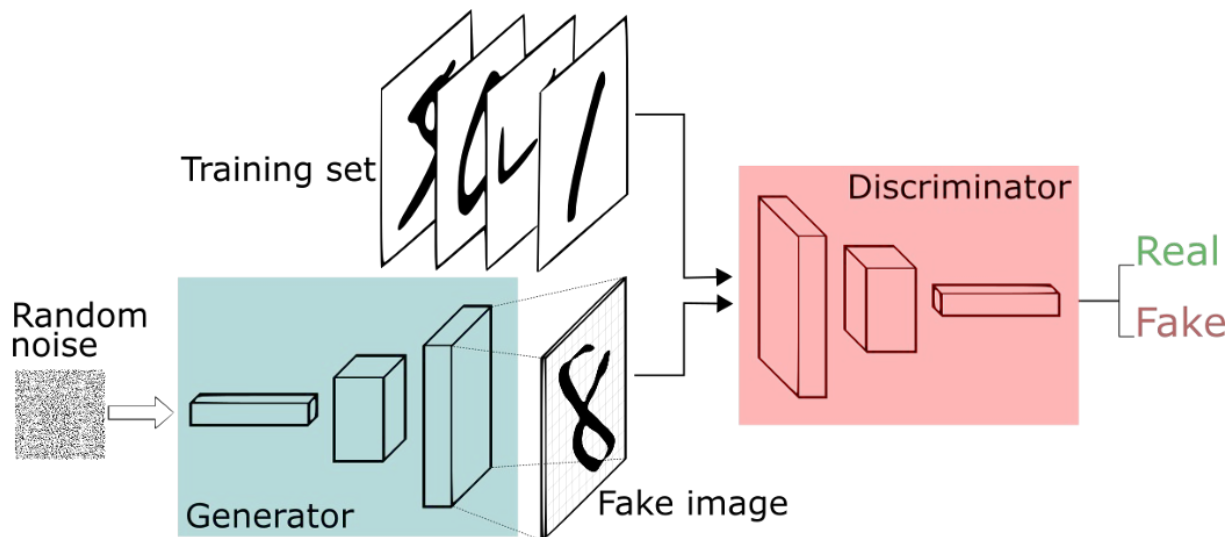


[Image source: "Pixel RNNs" by van den Oord et al., 2016]

# Likelihood-free generative models

## Generative adversarial networks

Main idea: adversarial game in which a generator tries to generate realistic data points that can fool a discriminator.



[Image source: "Generative Adversarial Networks - Explained" by Rohith Gandhi]

# Comparison of generative models

**DISCLAIMER: do not fully trust this slide, take it only as a very high level overview!!!**

- Summing up all the differences and subtleties of these complex models in one slide is impossible
- There is lots of research focusing on building variants of these models that make all these cells “greener”
- There is lots of research focusing on combining the advantages of different classes of models

There is no clear winner, which model to use depends on the application and on personal preferences/religion

	Expressivity	Sampling	Evaluating $p(\mathbf{x})$	Extracting latent representation
LVM (VAE)			Approximated	Approximated
Flow-based	Invertible transformations			
Autoregressive		Slow		No
GAN			No	No

# Latent variable models

# Latent variable models (LVMs)

To model a data point  $\mathbf{x}$  we can introduce a latent variable  $\mathbf{z}$ , that expresses hidden causes in the data-generating process.

- $\mathbf{z}$  is unobserved, but we can learn it from data.
- $\mathbf{z}$  is typically lower dimensional than  $\mathbf{x}$ , i.e. it captures a lower dimensional representation of the data



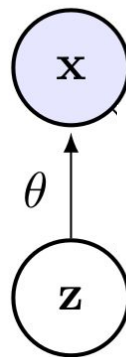
Example 1. **Linear LVMs:** Mixture of Gaussians, principal component analysis (PCA)

Example 2. **Non-linear LVMs:** VAEs

# Latent variable models

1. We define a *prior distribution* over the latent variables,  $p_{\theta}(\mathbf{z})$ .
2. We define a conditional distribution  $p_{\theta}(\mathbf{x}|\mathbf{z})$  for the data, also known as *likelihood*.
3. The generative model is defined in terms of a joint probability density:

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



$\theta$  collects all the model parameters, and will be learned from data.

**Generation:** to generate a data point we first get a sample  $\mathbf{z}^{(s)}$  from  $p_{\theta}(\mathbf{z})$ , and then get a sample  $\mathbf{x}^{(s)}$  from  $p_{\theta}(\mathbf{x}|\mathbf{z}^{(s)})$ .


# Marginal likelihood

$p_{\theta}(\mathbf{x})$  can be obtained from the joint distribution by integrating over the latent variables (assuming that  $\mathbf{z}$  is continuous):

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



The integral in  $p_{\theta}(\mathbf{x})$  is often intractable, and will need to be approximated.



The latent variable in the model allows us to express the complex marginal distribution  $p_{\theta}(\mathbf{x})$  in terms of a more tractable joint distribution, whose components  $p(\mathbf{x}|\mathbf{z})$  and  $p(\mathbf{z})$  are typically much simpler to define (e.g using exponential family distributions).

# Posterior distribution



Given a data point  $\mathbf{x}$ , how do we *infer* its latent variables  $\mathbf{z}$ ? Bayes' rule!

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

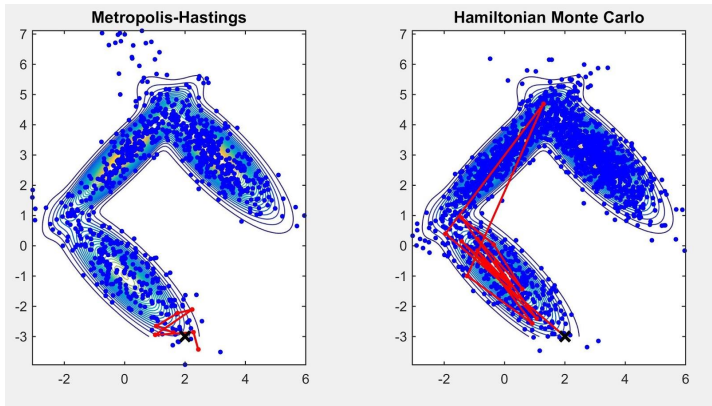
Due to the intractable integral in  $p_{\theta}(\mathbf{x})$ , the posterior  $p_{\theta}(\mathbf{z}|\mathbf{x})$  is also often intractable

- A lot of the research in probabilistic modelling focuses on effective ways to approximate this posterior distribution

# Approximate (posterior) inference

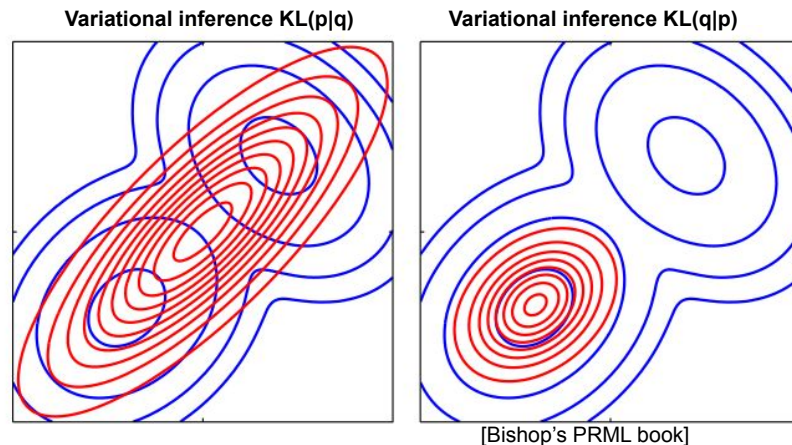
## Sampling techniques

- sample-based approximation to the posterior distribution
- slow but asymptotically exact
- e.g. Markov Chain Monte Carlo (MCMC), SMC, ..



## Deterministic approximation techniques

- we assume the posterior comes from a simpler family of distribution
- faster but not asymptotically exact
- e.g. Variational inference (VI), EP, ..



# Variational inference

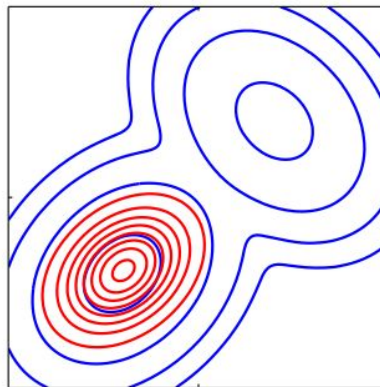
- Variational inference uses the calculus of variations to find the posterior approximation  $q(\mathbf{z})$  that minimizes the *Kullback–Leibler (KL) divergence* between  $q(\mathbf{z})$  and the true posterior  $p(\mathbf{z}|\mathbf{x})$ , i.e. a measure of dissimilarity between the two distributions
- We assume the posterior comes from a simple parametric family of distribution (e.g. Gaussian), or make some factorization assumptions
- We cast a complex inference procedure to a simpler optimization problem.

$$\text{KL} [q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})] = -\mathbb{E}_{q(\mathbf{z})} \left[ \log \frac{p(\mathbf{z}|\mathbf{x})}{q(\mathbf{z})} \right]$$

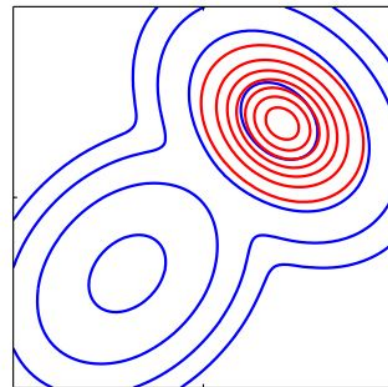
Important properties of the KL divergence:

1.  $\text{KL} \geq 0$
2.  $\text{KL} = 0$  if and only if  $q(\mathbf{z})=p(\mathbf{z}|\mathbf{x})$
3.  $\text{KL}[q||p] \neq \text{KL}[p||q]$ , so it is not a distance in the mathematical sense

KL(q||p): first local minimum



KL(q||p): second local minimum



[Bishop's PRML book]

# Posterior inference

The KL divergence is still not tractable, as the intractable posterior appears at the numerator inside the logarithm. However we can rewrite it as

$$\begin{aligned}\text{KL} [q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})] &= -\mathbb{E}_{q(\mathbf{z})} \left[ \log \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} - \log p(\mathbf{x}) \right] \\ &= -\underbrace{\mathbb{E}_{q(\mathbf{z})} \left[ \log \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right]}_{\mathcal{F}(q)} + \log p(\mathbf{x})\end{aligned}$$

$\mathcal{F}(q)$  is the *Evidence Lower BOund (ELBO)*:

1. Due to the non-negativity of the KL divergence, the ELBO represents a lower bound to the evidence  $\log p(\mathbf{x})$  for any  $q$ , i.e.  $\log p(\mathbf{x}) \geq \mathcal{F}(q)$
2. The closer the ELBO is to the marginal likelihood  $\log p(\mathbf{x})$ , the closer (in KL sense) the variational approximation will be to the posterior distribution
3. To minimize the KL we can just maximize the ELBO with respect to the distribution  $q(\mathbf{z})$ 
  - a. all the terms in the ELBO are tractable/simple to approximate

# Parameter learning (1)

Very often, the parameters  $\theta$  of the generative model are unknown

- Apart from being able to perform posterior inference, we also need to learn the parameters  $\theta$ . Given a training set with  $N$  data points we can use *Maximum Likelihood Estimation*:

$$\begin{aligned}\mathcal{L}(\theta) &= \sum_{i=1}^N \log p_{\theta}(\mathbf{x}^i) \\ &= \sum_{i=1}^N \underbrace{\log \int p_{\theta}(\mathbf{x}^i, \mathbf{z}^i) d\mathbf{z}^i}_{\mathcal{L}_i(\theta)}\end{aligned}$$



$\mathbf{x}_1, \dots, \mathbf{x}_N$


While we have a different latent variable  $\mathbf{z}^i$  for each data point, the parameters  $\theta$  are shared across all data points

# Parameter learning (2)

- As we have seen,  $p_{\theta}(\mathbf{x})$  is intractable but we can approximate it with the ELBO
- To learn the parameters of the model, instead of maximizing the log-likelihood we maximize the total ELBO with respect to  $\theta$  and  $q(\mathbf{z})$

Re-derivation of the ELBO (commonly used in VAE literature):

$$\begin{aligned}\mathcal{L}_i(\theta) &= \log p_{\theta}(\mathbf{x}) = \log \int p_{\theta}(\mathbf{x}, \mathbf{z}) d\mathbf{z} \\ &= \log \int \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} q(\mathbf{z}) d\mathbf{z} \\ &= \log \mathbb{E}_{q(\mathbf{z})} \left[ \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right] \\ &\geq \mathbb{E}_{q(\mathbf{z})} \left[ \log \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right] = \mathcal{F}_i(\theta, q)\end{aligned}$$

 **Jensen's inequality,**  
using concavity of the  
logarithm

# The Expectation-Maximization (EM) algorithm

- A two-stage iterative optimization method for MLE of the parameters of a model with latent variables.
  - Generalization of the EM algorithm, commonly presented for mixture of Gaussians (which is a LVM!)
  - For many models, each of these step will be simpler than updating both  $q(\mathbf{z})$  and  $\theta$  at the same time (for VAEs however we can easily perform joint maximization)

$$\text{E-step:} \quad q_{k+1} = \underset{q}{\operatorname{argmax}} \mathcal{F}_i(\theta_k, q)$$

$$\text{M-step:} \quad \theta_{k+1} = \underset{\theta}{\operatorname{argmax}} \mathcal{F}_i(\theta, q_{k+1})$$

# Variational autoencoders

# Variational autoencoders (VAEs)

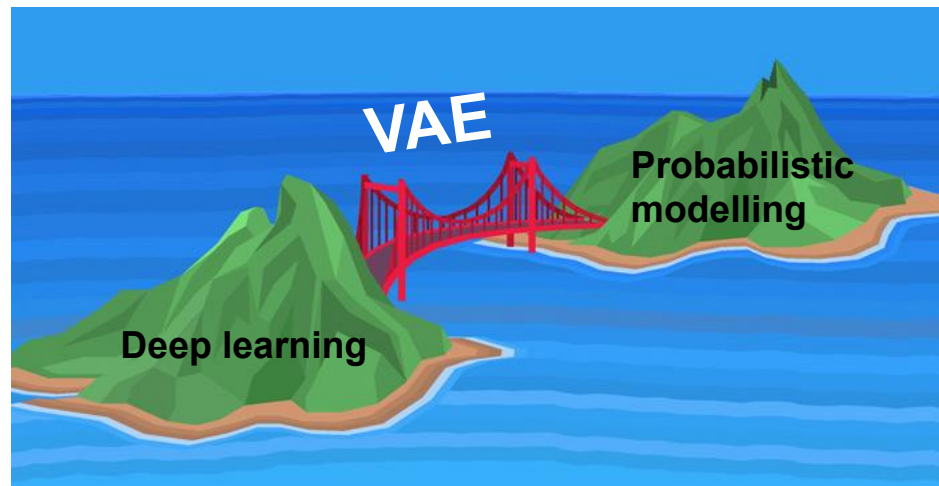
VAEs are latent variable models in which we use deep neural networks to parameterize flexible probability distributions (=“Deep LVMs”)

- Flexible: can be used in many different applications, no manual feature engineering
- Scalable training with Variational Inference
- Similar ideas can be applied in many other domains (e.g. sequential data)

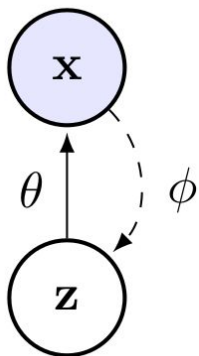
VAEs combine the advantages of 2 popular research areas:

1. **Deep Learning:** expressive function approximators, scalable end-to-end training
2. **Probabilistic modelling:** model uncertainty, more interpretable structure

[Kingma and Welling, 2014; Rezende et al., 2014]



# Generative model of a VAE



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

- $\mathbf{z}$  is a continuous latent variable. Its *prior* is typically an isotropic multivariate Gaussian  $p_{\theta}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$
- $p_{\theta}(\mathbf{x}|\mathbf{z})$  is the *likelihood* or *decoder*
  - For continuous data it is typically a Gaussian distribution parameterized by two deep neural networks

$$p_{\theta}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \mathbf{v}) \quad \text{with} \quad \boldsymbol{\mu} = \text{NN}_1(\mathbf{z}), \quad \log \mathbf{v} = \text{NN}_2(\mathbf{z})$$

- For binary data it is usually a Bernoulli distribution
- $\theta$  are the weights and biases of  $\text{NN}_1$  and  $\text{NN}_2$

# Approximate inference in VAEs

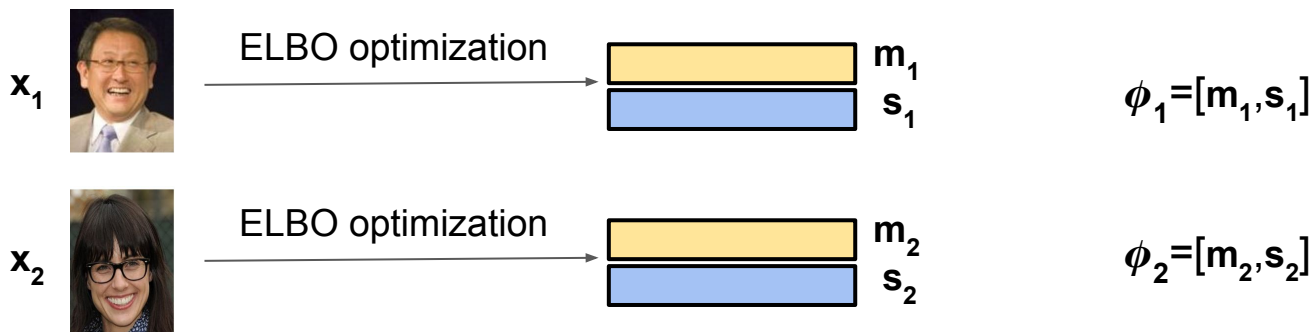
Due to the non-linearities in the neural networks, the marginal likelihood  $p_{\theta}(\mathbf{x})$  and the posterior distribution  $p_{\theta}(\mathbf{z}|\mathbf{x})$  are intractable.

For large data sets of high-dimensional data, variational inference provides a good trade-off between quality of the approximation and scalability of the inference procedure.

- Using amortized variational inference, not the “traditional” way of doing variational inference

# Traditional variational inference

Common assumption: posterior comes from a simple parametric family of distribution, with a different set of parameters for each data point. To make this explicit we write  $q(\mathbf{z})$  as  $q_{\phi_i}(\mathbf{z}^i)$  with  $\phi_i$  the parameters relative to data point  $\mathbf{x}^i$ . For a Gaussian variational approximation  $N(\mathbf{z}^i; \mathbf{m}_i, \mathbf{s}_i)$ :



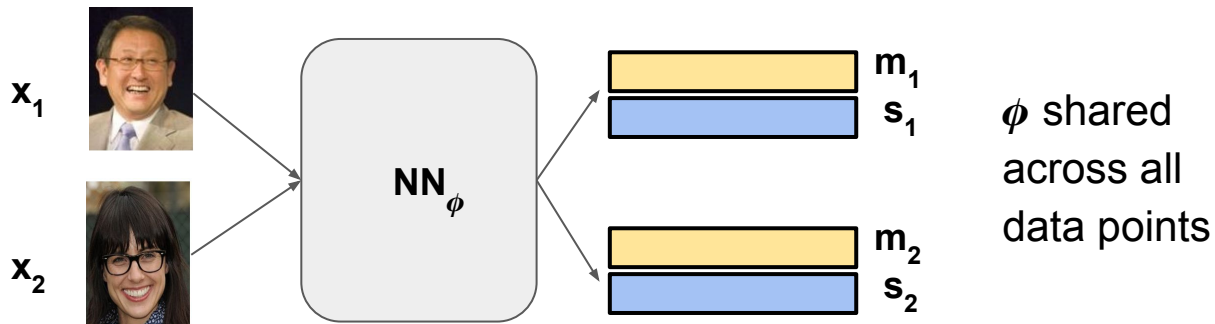
## Problems:

1. **Linear scaling** of the parameters of the variational approximation with the number of data points in the data sets: problem in large data sets that may contain millions of elements :(
2. When a new data point arrives, we need to optimize the ELBO to obtain  $\phi_i$

# Amortized variational inference in VAEs

- Instead of having a different set of parameters  $\phi_i$  to learn for each data point, we share the variational parameters  $\phi$  across all data points (we have therefore dropped the  $i$  subscript in the notation)
- We use deep neural networks that take the data point  $\mathbf{x}^i$  as input, and output the mean and diagonal covariance matrix of the corresponding Gaussian variational approximation  $q_\phi(\mathbf{z}^i|\mathbf{x}^i)$ , which is also known as *inference network* or *encoder*

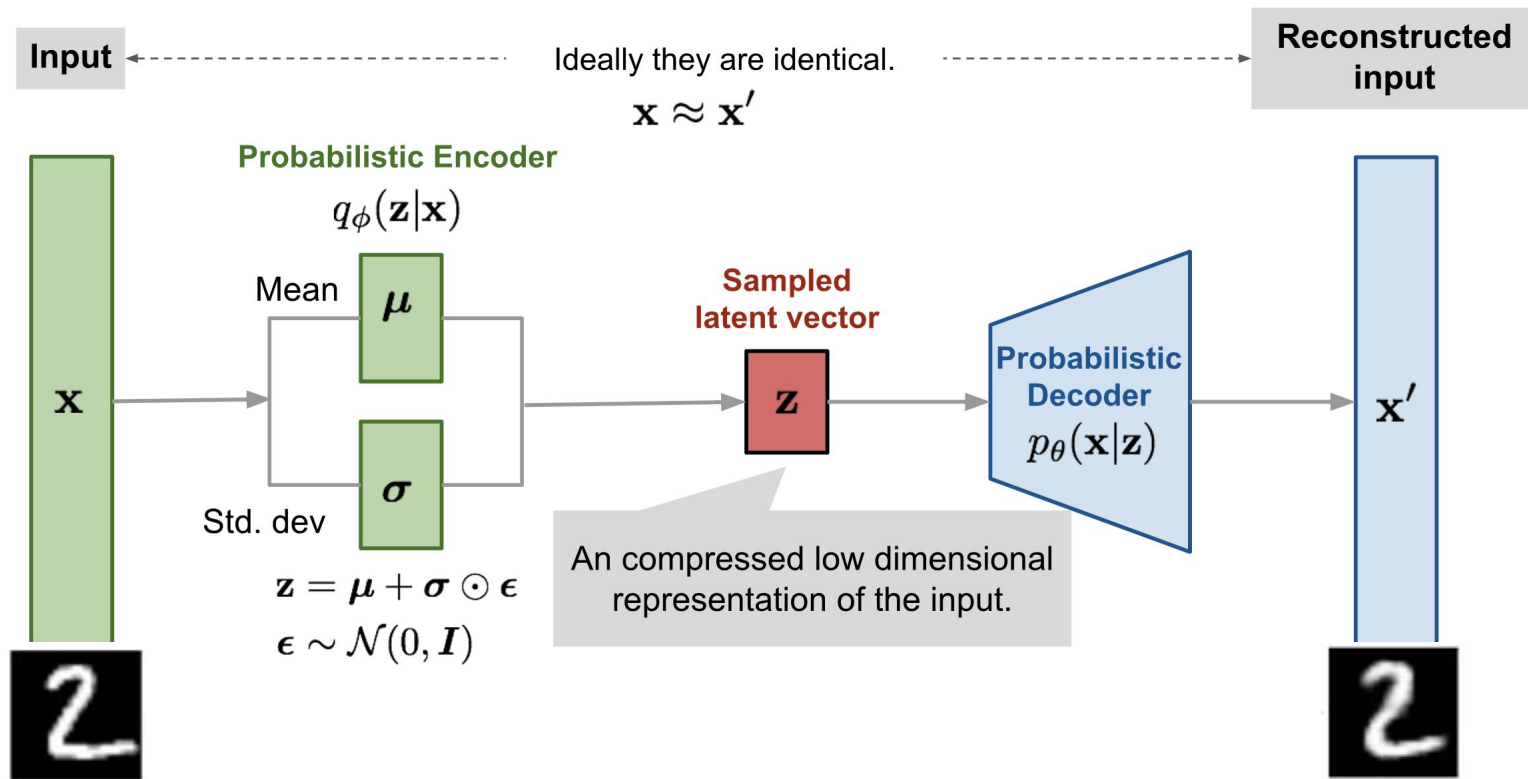
## Solutions:



1. Cost of learning the variational parameters is amortised across all data points
2. When a previously unseen data point arrives we can immediately compute its variational approximation without the need to run an expensive optimization of the ELBO

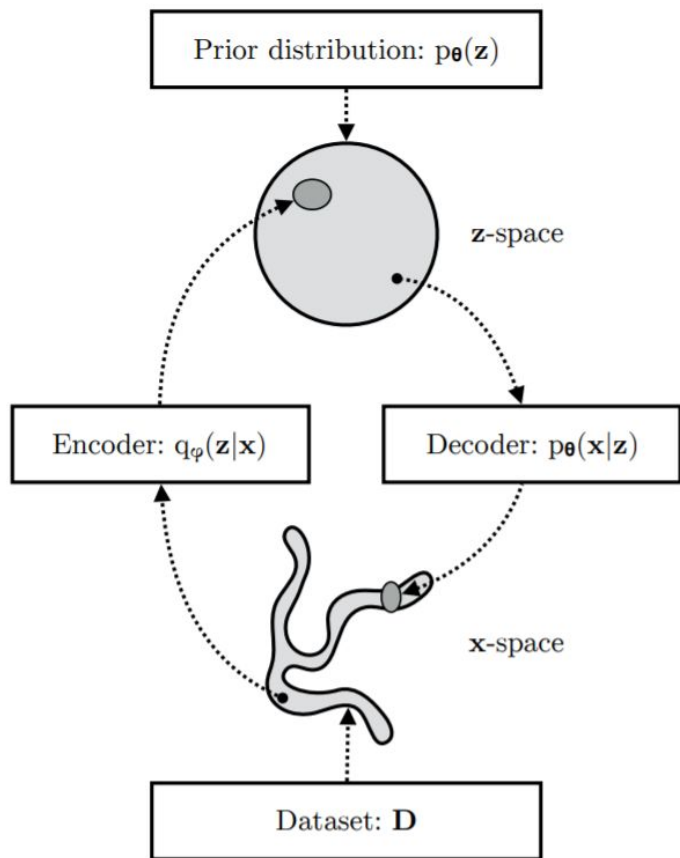
TRADEOFF: the posterior approximation found with amortised inference will always be worse than the one found with the traditional approach

# Auto-encoding in VAEs



[Image source: Blog post *From Autoencoder to Beta-VAE*, Lilian Weng, 2018]

# Role of encoder and decoder in a VAE



A VAE learns stochastic mappings between **z-space** and **x-space**

- A simple distribution in **z-space** becomes a complicated one in **x-space**

[Image source: Kingma and Welling, 2017]

# Parameter learning with the ELBO

- The variational approximation depends on  $\phi$ , therefore  $\mathcal{F}(\theta, q) \rightarrow \mathcal{F}(\theta, \phi)$
- Since both generative model and variational approximation are defined using deep neural networks we can use backpropagation to optimize the ELBO with gradient ascent algorithms (see caveat in next slide)
- $\theta$  and  $\phi$  can be learned jointly (unlike EM)

$$\begin{aligned}\mathcal{F}_i(\theta, \phi) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[ \log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \right] \\ &= \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction term}} - \underbrace{\text{KL} [q_\phi(\mathbf{z}|\mathbf{x}) || \log p_\theta(\mathbf{z})]}_{\text{Regularization term}} .\end{aligned}$$

Reconstruction term: encourages the model to be able to reconstruct the data accurately

Regularization term: penalizes posterior approximations that are too far from the prior

# Reparameterization trick

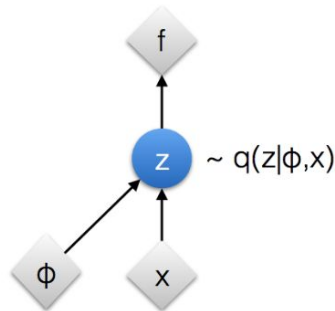
Issue. we need to back-propagate through a stochastic node.

Solution. *Reparameterization trick*: write  $\mathbf{z}$  as a deterministic transformation of a simpler random variable, so that all the parameters are in the “deterministic part” of the graph.

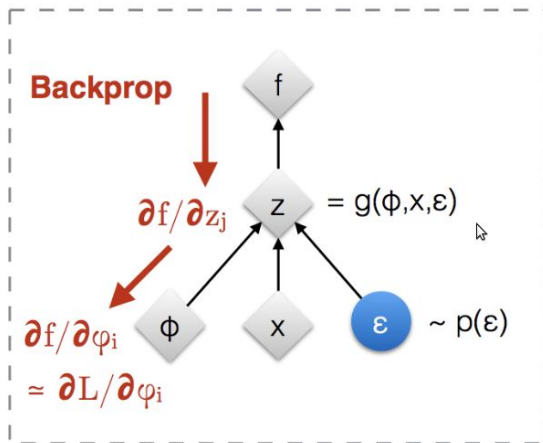
In a VAE  $q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mathbf{m}, \mathbf{s})$  can be rewritten as:

$\mathbf{z} = g(\phi, \mathbf{x}, \boldsymbol{\epsilon}) = \mathbf{m} + \mathbf{s} \odot \boldsymbol{\epsilon}$  with  $p(\boldsymbol{\epsilon}) = \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{I})$

Original form



Reparameterised form



[Kingma, 2013]

[Bengio, 2013]

[Kingma and Welling 2014]

[Rezende et al 2014]

[Source: Kingma's NIPS 2015 workshop slides]

# MNIST implementation in PyTorch

```
class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()

        self.fc1 = nn.Linear(784, 400)
        self.fc21 = nn.Linear(400, 20)
        self.fc22 = nn.Linear(400, 20)
        self.fc3 = nn.Linear(20, 400)
        self.fc4 = nn.Linear(400, 784)

    def encode(self, x):
        h1 = F.relu(self.fc1(x))
        return self.fc21(h1), self.fc22(h1)

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5*logvar)
        eps = torch.randn_like(std)
        return mu + eps*std
```

```
    def decode(self, z):
        h3 = F.relu(self.fc3(z))
        return torch.sigmoid(self.fc4(h3))
```

```
    def forward(self, x):
        mu, logvar = self.encode(x.view(-1, 784))
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar
```

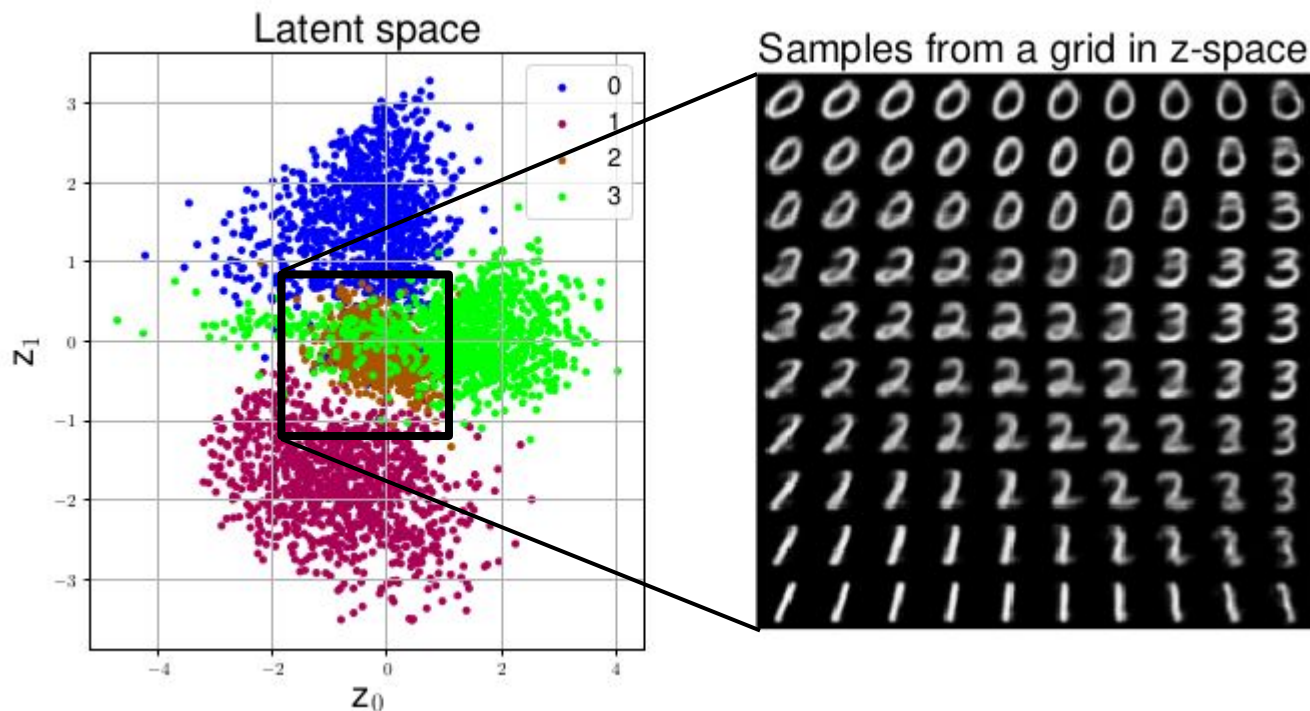
# Reconstruction + KL divergence losses summed over all elements and batch

```
def loss_function(recon_x, x, mu, logvar):
    BCE = F.binary_cross_entropy(recon_x, x.view(-1, 784), reduction='sum')
    # see Appendix B from VAE paper:
    # Kingma and Welling. Auto-Encoding Variational Bayes. ICLR, 2014
    # https://arxiv.org/abs/1312.6114
    # 0.5 * sum(1 + log(sigma^2) - mu^2 - sigma^2)
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return BCE + KLD
```

[Source: <https://github.com/pytorch/examples/blob/master/vae/main.py>]

# MNIST results

Simple model with 2-dimensional latent variables first four digits MNIST digits (for visualization purposes)



# VAE exercises

VAE exercises from DTU's course: *02456 Deep learning*

Tensorflow: <https://bit.ly/2MPatyh>

PyTorch: <https://bit.ly/31pdEAI>

# Advanced topics in VAE research

# Main VAE research directions

1. Better models: build more powerful models e.g.
  - a. adding a hierarchy of latent variables (Rezende et al., 2014; Sønderby et al., 2016; Maaløe et al., 2019)
  - b. more powerful decoders based on autoregressive/flow-based models (Gulrajani et al., 2016; Chen et al., 2017)
  - c. using discrete latent variables (Jang et al., 2017, Maddison et al., 2017)
2. Better posterior approximations: define more expressive posterior distributions e.g.
  - a. using normalizing flows (Rezende and Mohamed, 2015; Kingma et al., 2016)
  - b. using auxiliary variables (Ranganath et al., 2015; Maaløe et al., 2016)
3. Different objective function: e.g.
  - a. tighter ELBO approximations (Burda et al., 2015)
  - b. use other divergence measures (Li and Turner, 2016)
4. Applications: e.g.
  - a. semi-supervised learning (Kingma et al., 2014; Maaløe et al., 2016)
  - b. anomaly detection (Maaløe et al., 2019)
  - c. learning disentangled representations (Higgins et al., 2017)

# Ladder VAE (LVAE)

Adding a hierarchy of latent variables in the generative model (Figure 2a) allows us to model very complex data distributions.

Inference is however harder:

- A bottom-up inference model as in Figure 2b has issues in activating the higher stochastic units
- The LVAE adds a bottom-up deterministic path followed by a top-down stochastic inference path
  - This forces the model to learn to exploit all variables in the hierarchy
- Training tricks: batch-norm, KL warm up

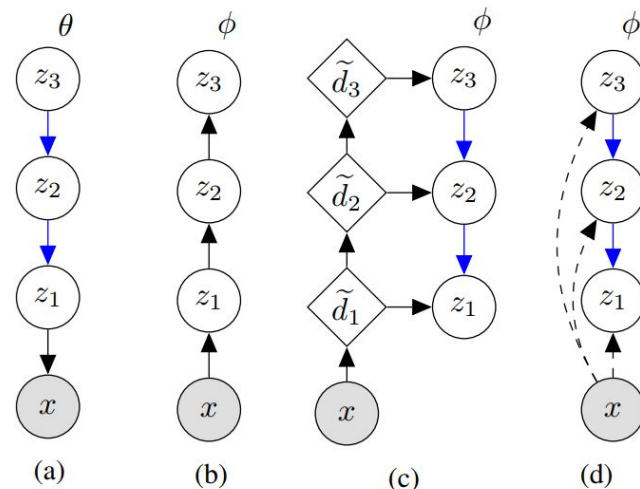


Figure 2. (a) Generative model of a VAE/LVAE with  $L = 3$  stochastic variables, (b) VAE inference model, (c) LVAE inference model, and (d) skip connections among stochastic variables in the LVAE where dashed lines denote a skip-connection. Blue arrows indicate that shared parameters between the inference and generative model in LVAE.

[Sønderby et al., 2016]

[Image source: Maaløe et al., 2019]

# LVAE results

## Inactive units

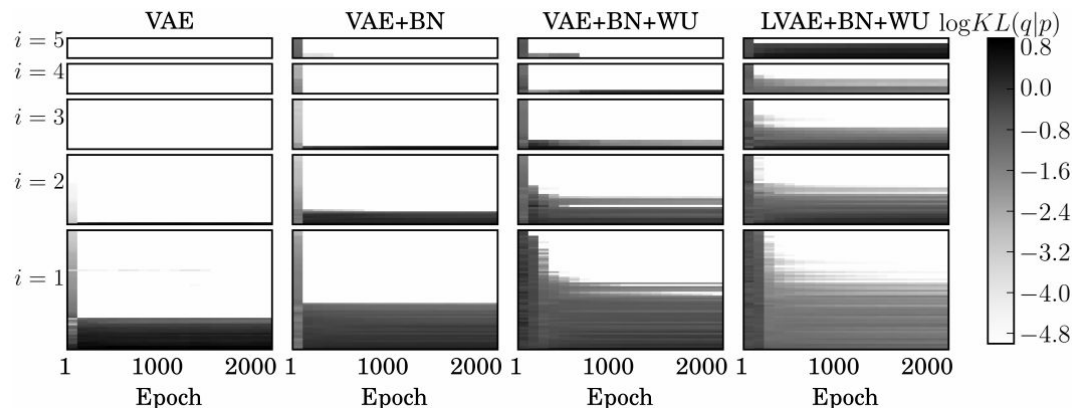


Figure 4:  $\log KL(q|p)$  for each latent unit is shown at different training epochs. Low  $KL$  (white) corresponds to an inactive unit. The units are sorted for visualization. It is clear that vanilla VAE cannot train the higher latent layers, while introducing batch normalization helps. Warm-up creates more active units early in training, some of which are then gradually pruned away during training, resulting in a more distributed final representation. Lastly, we see that the LVAE activates the highest number of units in each layer.

## MNIST latent space

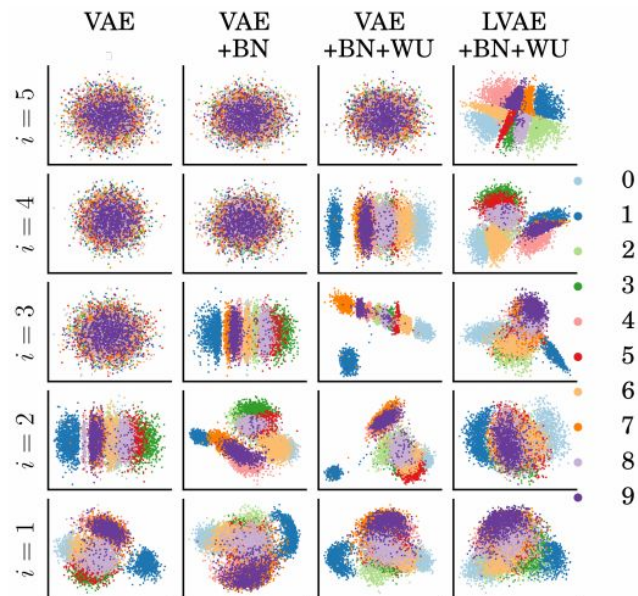


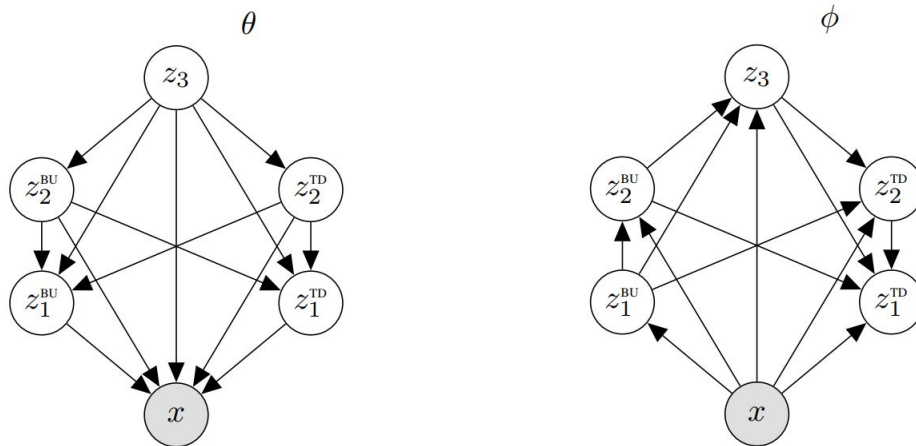
Figure 6: PCA-plots of samples from  $q(z_i|z_{i-1})$  for 5-layer VAE and LVAE models trained on MNIST. Color-coded according to true class label

# Bidirectional-Inference Variational Autoencoder (BIVA)

In practice, LVAE cannot handle very deep LVMs. To solve this, BIVA extends the LVAE in 2 ways:

1. Skip-connected generative model using a deterministic top-down path (Figure a), which allows a better flow of information in the model and to avoid the collapse of latent variables in very deep models
2. Bidirectional inference using a *stochastic* bottom-up and top-down paths that form a very flexible variational approximation (without introducing auxiliary variables)

Doing this, BIVA can use a very deep hierarchy of latent variables (up to 20 layers of stochastic variables)



(a) Graphical model of the generative model (b) Graphical model of the variational approximation

[Maaløe L., Fraccaro M., Liévin V., Winther O. "BIVA: A Very Deep Hierarchy of Latent Variables for Generative Modeling".]

# BIVA results

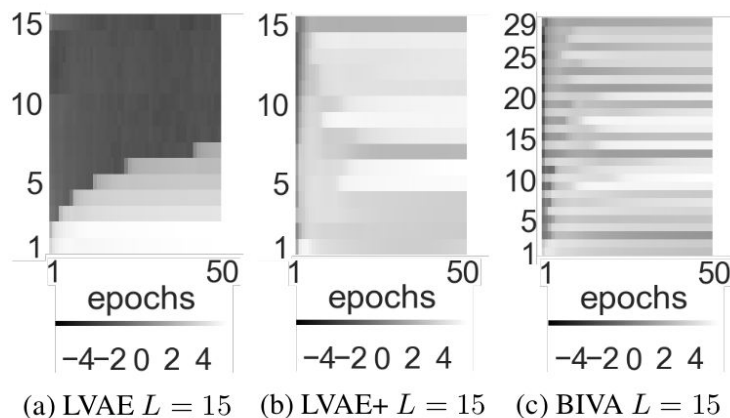


Figure 2: The  $\log KL(q||p)$  for each stochastic latent variable as a function of the training epochs on CIFAR-10. (a) is a  $L = N = 15$  stochastic latent layer LVAE with no skip-connections and no bottom-up inference. (b) is a  $L = N = 15$  LVAE+ with skip-connections and no bottom-up inference. (c) is a  $L = 15$  stochastic latent layer ( $N = 29$  latent variables) BIVA for which  $1, 2, \dots, N$  denotes the stochastic latent variables following the order  $z_1^{\text{BU}}, z_1^{\text{TD}}, z_2^{\text{BU}}, z_2^{\text{TD}}, \dots, z_L$ .



Figure 3: (left) images from the CelebA dataset preprocessed to  $64 \times 64$  following [27]. (right)  $\mathcal{N}(0, I)$  generations of BIVA with  $L = 20$  layers that achieves a  $\mathcal{L}_1 = 2.48$  bits/dim on the test set.

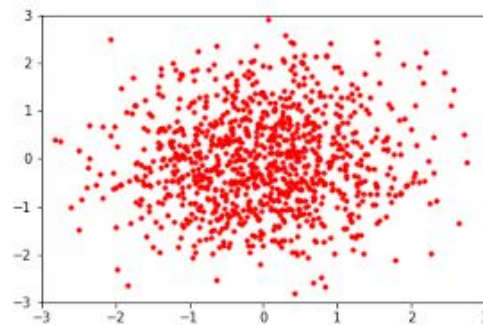
# Variational inference with normalizing flows

We can form a complex variational approximation by starting from a simple density and constructing a chain of  $R$  learnable parametric transformations  $f_r$ , known as *normalizing flows*, that expand and contract the initial density:

$$\mathbf{z}_R = f_R(f_{R-1}(\dots f_1(\mathbf{z}_0)))$$

The resulting probability density over the variable  $\mathbf{z}_R$  can be computed by repeatedly applying the rule for change of variables, giving

$$\log q_R(\mathbf{z}_R) = \log q_0(\mathbf{z}_0) - \sum_{r=1}^R \log \left| \det \frac{\partial f_r}{\partial \mathbf{z}_{r-1}} \right|$$



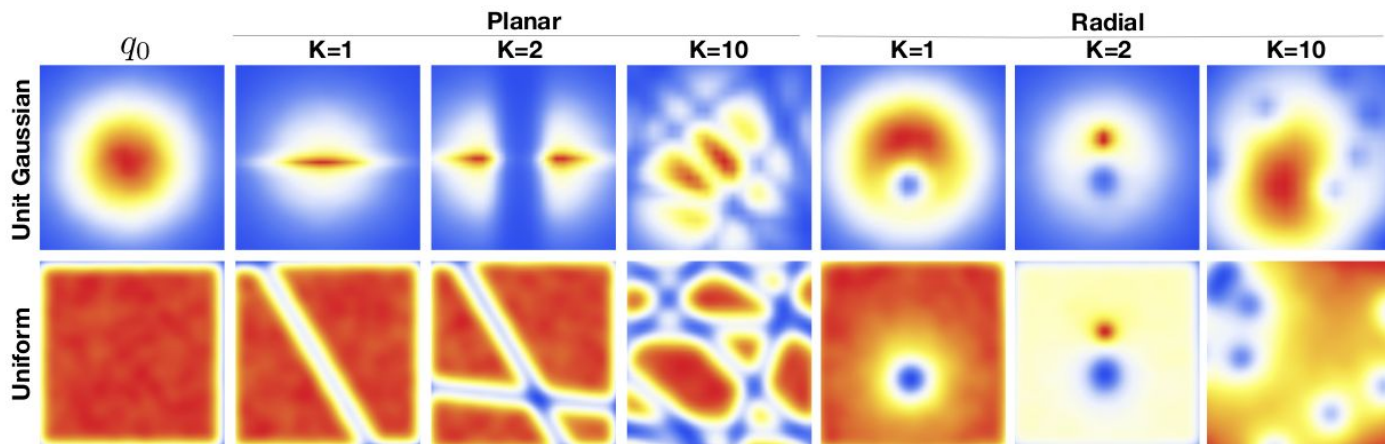
Requirements for the flow:

- The input and output dimensions must be the same.
- The transformation  $f_r$  must be invertible.
- Efficient computation of the determinant of the Jacobian and its gradient.

[Rezende and Mohamed, 2015]

# Complex posteriors with normalizing flows

We define an inference network using a parametric flow family (e.g. planar or radial flows as below), and learn the parameters of the flow.



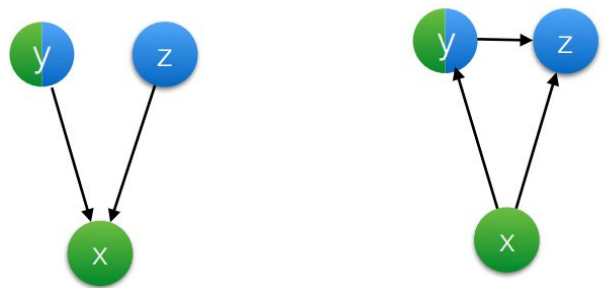
*Figure 1.* Effect of normalizing flow on two distributions.

[Rezende and Mohamed, 2015]

# Semi-supervised learning with VAEs

- Define a generative model that describes the data as being generated by a latent class variable  $y$  in addition to a continuous latent variable  $z$ .
- The class labels  $y$  are treated as latent variables for unlabelled data
- The inference network over  $y$  can be used as a classifier

Generative model    Inference model



[Kingma et al., 2014]  
[Image source: Kingma's NIPS 2015 workshop slides]

	ERROR %
M1+M2 (KINGMA ET AL., 2014)	3.33% ( $\pm 0.14$ )
VAT (MIYATO ET AL., 2015)	2.12%
CATGAN (SPRINGENBERG, 2015)	1.91% ( $\pm 0.10$ )
SDGM (MAALØE ET AL., 2016)	1.32% ( $\pm 0.07$ )
LADDERNET (RASMUS ET AL., 2015)	1.06% ( $\pm 0.37$ )
ADGM (MAALØE ET AL., 2016)	0.96% ( $\pm 0.02$ )
IMPGAN (SALIMANS ET AL., 2016)	0.93% ( $\pm 0.07$ )
TRIPLEGAN (LI ET AL., 2017)	0.91% ( $\pm 0.58$ )
SSLGAN (DAI ET AL., 2017)	0.80% ( $\pm 0.10$ )
<b>BIVA</b>	0.83% ( $\pm 0.02$ )

Table 3. Semi-supervised test error for BIVA on MNIST for 100 randomly chosen and evenly distributed labelled samples.

# References

Most of this discussion is derived from chapters in my PhD thesis:

Fraccaro M. (2018). “Deep Latent Variable Models for Sequential Data.” PhD thesis, Technical University of Denmark, 2018.

## LVMs and VAEs

Roweis S. and Z. Ghahramani (1999). “A unifying review of linear Gaussian models”. In: Neural Computation

Kingma D. and M. Welling (2014). “Auto-encoding variational Bayes”. In: ICLR.

Kingma D. and M. Welling (2017). “An Introduction to Variational Autoencoders”.

Rezende D. J., S. Mohamed, and D. Wierstra (2014). “Stochastic backpropagation and approximate inference in deep generative models”

## VAE RESEARCH

Burda Y., R. Grosse, and R. Salakhutdinov (2015). “Importance Weighted Autoencoders”.

Chen X., D. P. Kingma, T. Salimans, Y. Duan, P. Dhariwal, J. Schulman, I. Sutskever, and P. Abbeel (2017). “Variational Lossy Autoencoder”. In: ICLR

Gulrajani I., K. Kumar, F. Ahmed, A. A. Taiga, F. Visin, D. Vazquez, and A. Courville (2016). “PixelVAE: A Latent Variable Model for Natural Images”.

Higgins I., L. Matthey, A. Pal et al. (2017). “beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework”. In: ICLR

Jang E., S. Gu, and B. Poole (2016). “Categorical Reparameterization with Gumbel-Softmax”.

Kingma D. P., D. J. Rezende, S. Mohamed, and M. Welling (2014). “Semi-Supervised Learning with Deep Generative Models”. In: ICML

Kingma D. P., T. Salimans, R. Jozefowicz, X. Chen, et al. (2016). “Improved Variational Inference with Inverse Autoregressive Flow”. In: NIPS.

Li Y. and R. E. Turner (2016). “Rényi Divergence Variational Inference”. In: NIPS

Maaløe L., C. K. Sønderby, S. K. Sønderby, and O. Winther (2016). “Auxiliary Deep Generative Models”. In: ICML

Maaløe L., Fraccaro M., Liévin V., Winther O. “BIVA: A Very Deep Hierarchy of Latent Variables for Generative Modeling”.

Maddison C. J., A. Mnih, and Y. W. Teh (2017b). “The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables”. In: ICLR.

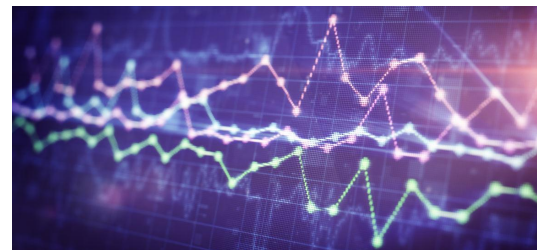
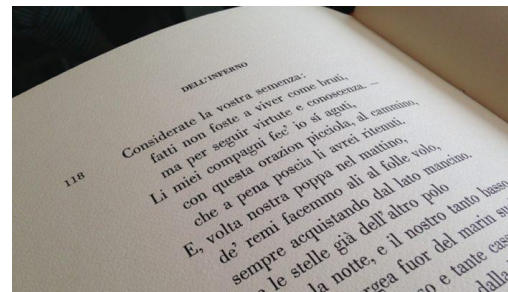
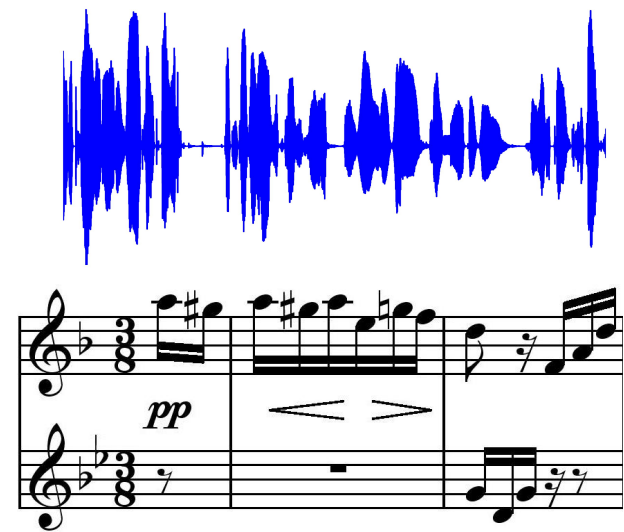
Ranganath R., D. Tran, and D. M. Blei (2015). “Hierarchical Variational Models”.

Rezende D. J. and S. Mohamed (2015). “Variational Inference with Normalizing Flows”. In: ICML

Sønderby C. K., T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther (2016b). “Ladder Variational Autoencoders”. In: NIPS

# Extending VAEs to sequential data

# Sequential data



# Goal

Introduce a general class of models for unsupervised learning of sequential data:

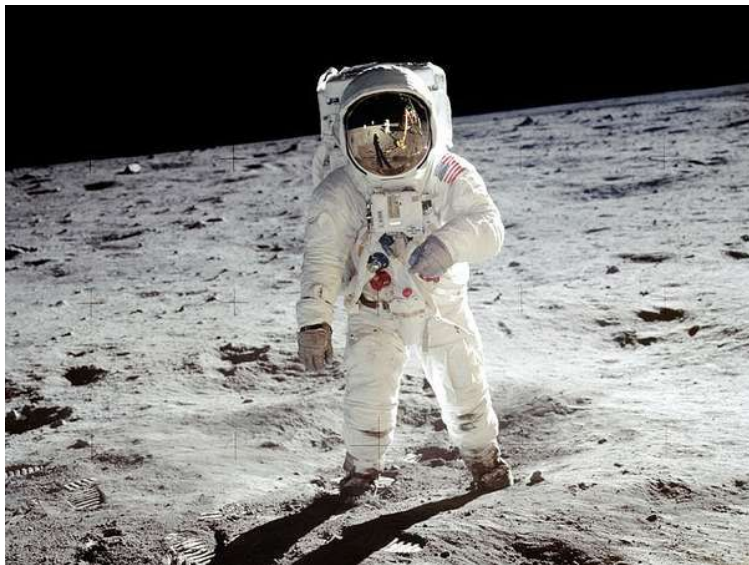
- Can model a wide range of complex temporal data
- Can be trained in a scalable way using large unlabelled datasets

We do it by combining ideas from deep learning and probabilistic modelling

- Flexible models inspired by Variational autoencoders

# State-space models

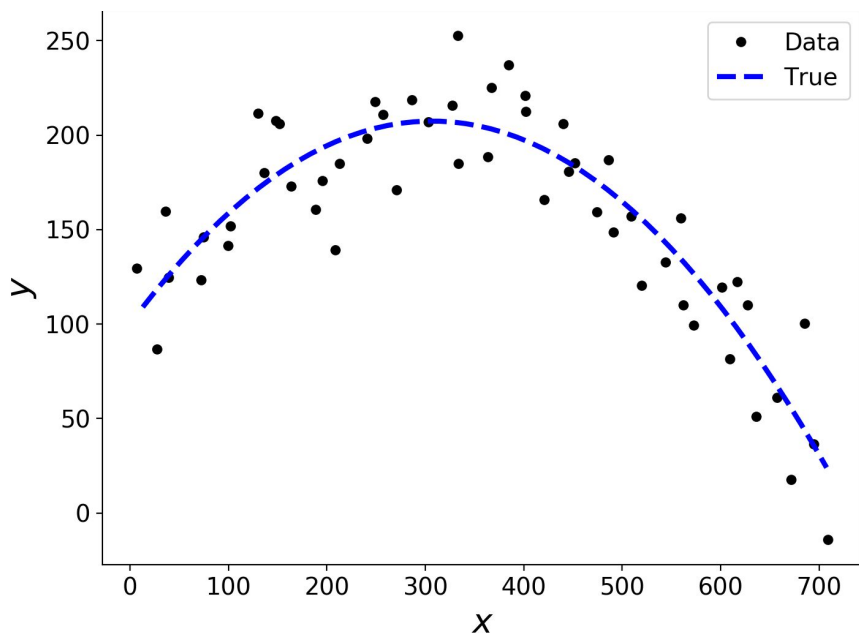
- The main building block of the more advanced models discussed later
- We can see it as the sequential extension of latent variable models
  - The prior changes over time
- Introduced at NASA in the early 1960s to estimate the trajectory of the spacecrafts



“Apollo 11, this is Houston.. we have  
a **state vector** update for you.”

*Transcription from the Apollo 11 mission*

# Trajectory estimation from noisy measurements

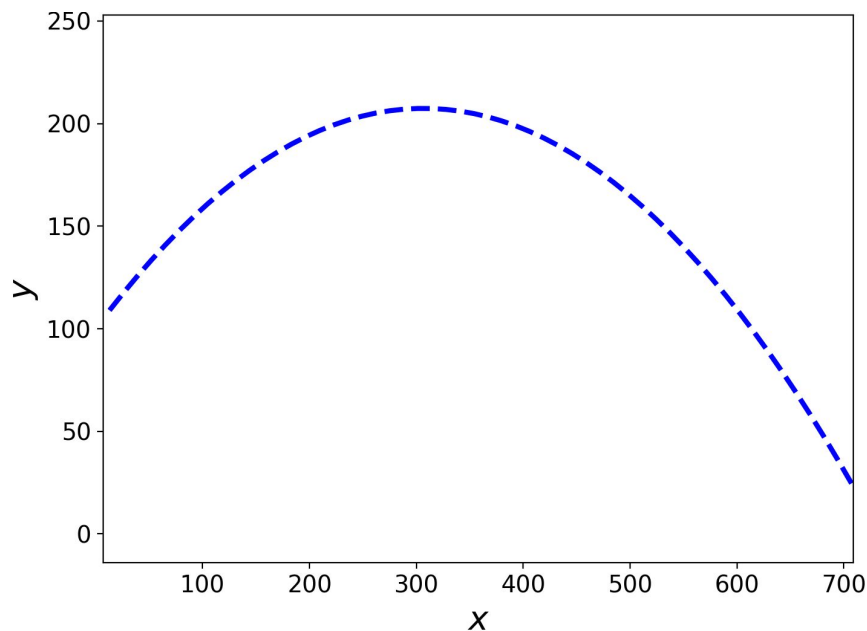


We define state-space models using an example:

- We want to track the movement of a flying ball
  - We want to estimate the true trajectory (blue dashed line)
  - We can only observe the noisy  $(x, y)$  positions (black dots)
- Simplified version of what is done in global positioning systems (GPS)

# Parabolic motion of a ball thrown in vacuum

We can exploit our knowledge of the physics of the moving ball



Newton's equations for parabolic motion:

$$\begin{cases} x_t = x_{t-1} + \dot{x}_{t-1}\Delta \\ \dot{x}_t = \dot{x}_{t-1} \end{cases} \quad \text{uniform motion}$$
$$\begin{cases} y_t = y_{t-1} + \dot{y}_{t-1}\Delta - \frac{1}{2}g\Delta^2 \\ \dot{y}_t = \dot{y}_{t-1} - g\Delta \end{cases} \quad \text{uniformly accelerated motion}$$

$$\mathbf{z}_t = \begin{bmatrix} x_t \\ \dot{x}_t \\ y_t \\ \dot{y}_t \end{bmatrix} \quad \text{summarizes the state of the system at each time step (positions and velocities)}$$

# Transition equation of a state-space model

$$\begin{cases} x_t \\ \dot{x}_t \\ y_t \\ \dot{y}_t \end{cases} = \begin{cases} x_{t-1} + \dot{x}_{t-1}\Delta \\ \dot{x}_{t-1} \\ y_{t-1} + \dot{y}_{t-1}\Delta - \frac{1}{2}g\Delta^2 \\ \dot{y}_{t-1} - g\Delta \end{cases}$$
$$\mathbf{z}_t = \mathbf{A}\mathbf{z}_{t-1} + \mathbf{B}\mathbf{u}_t$$

$$\mathbf{A} = \begin{bmatrix} 1 & \Delta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

Transition matrix

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{2}\Delta^2 & 0 \\ 0 & 0 & 0 & -\Delta \end{bmatrix},$$

Control matrix

$$\mathbf{u}_t = \begin{bmatrix} 0 \\ 0 \\ g \\ g \end{bmatrix}$$

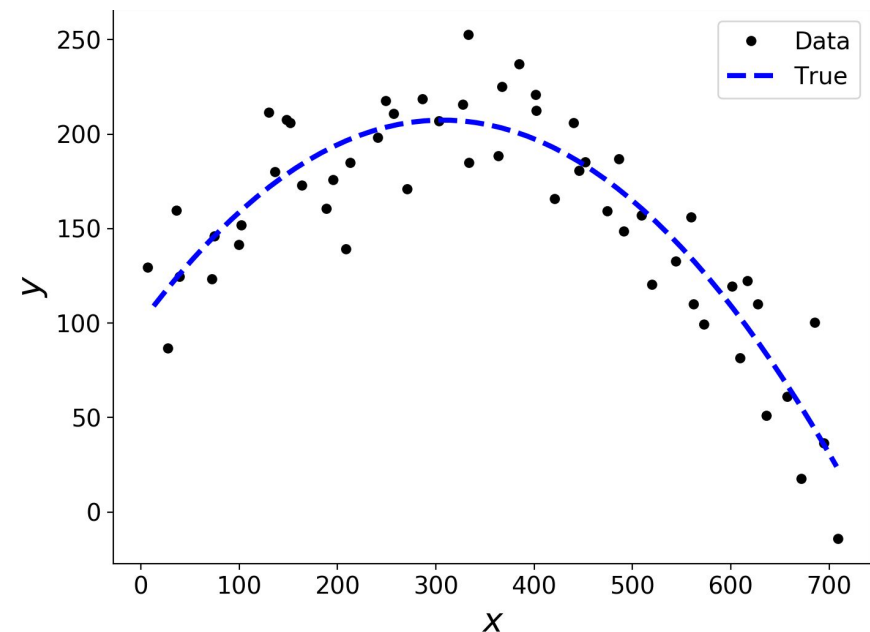
Action/control

**Transition equation**: specifies the evolution of the system in time.

$$\mathbf{z}_t = \mathbf{A}\mathbf{z}_{t-1} + \mathbf{B}\mathbf{u}_t$$

# Emission equation of a state-space model

We do not observe the state  $\mathbf{z}_t$ , but the **noisy position**  $\mathbf{x}_t$ .



$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \mathbf{z}_t = \begin{bmatrix} x_t \\ \dot{x}_t \\ y_t \\ \dot{y}_t \end{bmatrix}$$

Emission matrix

latent position

**Emission equation:** from state to noisy position

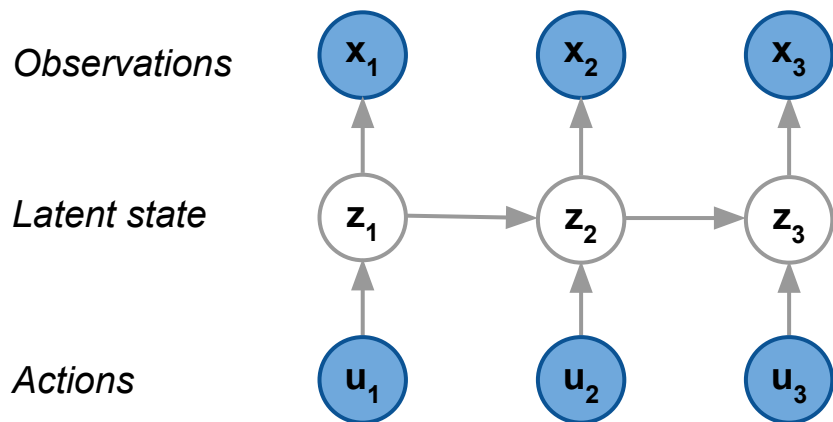
$$\mathbf{x}_t = \mathbf{C}\mathbf{z}_t + \delta_t$$

Noisy observation

State

Measurement noise (Gaussian)

# Linear Gaussian State-Space Models (LGSSM)



- Transition equation:
$$\mathbf{z}_t = \mathbf{A}_t \mathbf{z}_{t-1} + \mathbf{B}_t \mathbf{u}_t + \epsilon_t$$
- Emission equation:
$$\mathbf{x}_t = \mathbf{C}_t \mathbf{z}_t + \delta_t$$

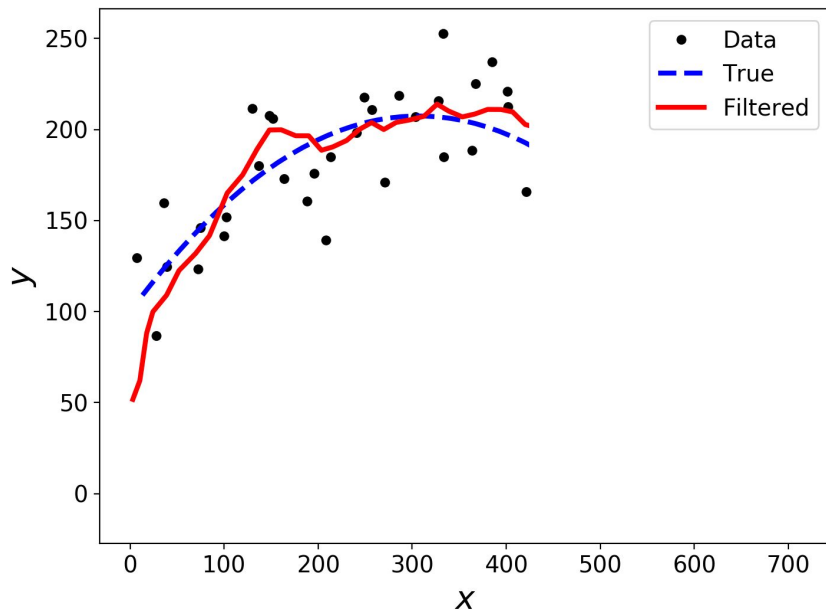
From a probabilistic point of view:

$$p_{\theta_t}(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{u}_t) = \mathcal{N}(\mathbf{z}_t; \mathbf{A}_t \mathbf{z}_{t-1} + \mathbf{B}_t \mathbf{u}_t, \mathbf{Q}_t)$$

$$p_{\theta_t}(\mathbf{x}_t | \mathbf{z}_t) = \mathcal{N}(\mathbf{x}_t; \mathbf{C}_t \mathbf{z}_t, \mathbf{R}_t) ,$$

$$\theta_t = [\mathbf{A}_t, \mathbf{B}_t, \mathbf{C}_t, \mathbf{Q}_t, \mathbf{R}_t]$$

# Posterior inference (filtering)

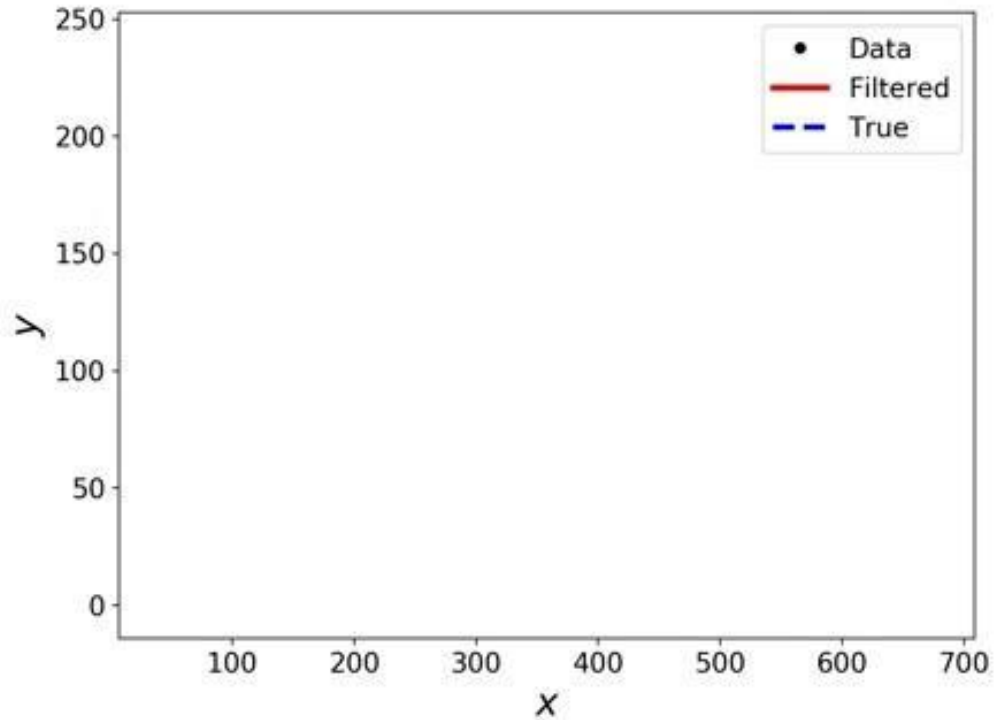


- Our original goal was to estimate the trajectory from the noisy measurements
- We need to estimate the state given the data we have observed so far:

$$p_{\theta}(\mathbf{z}_t | \mathbf{x}_{1:t})$$

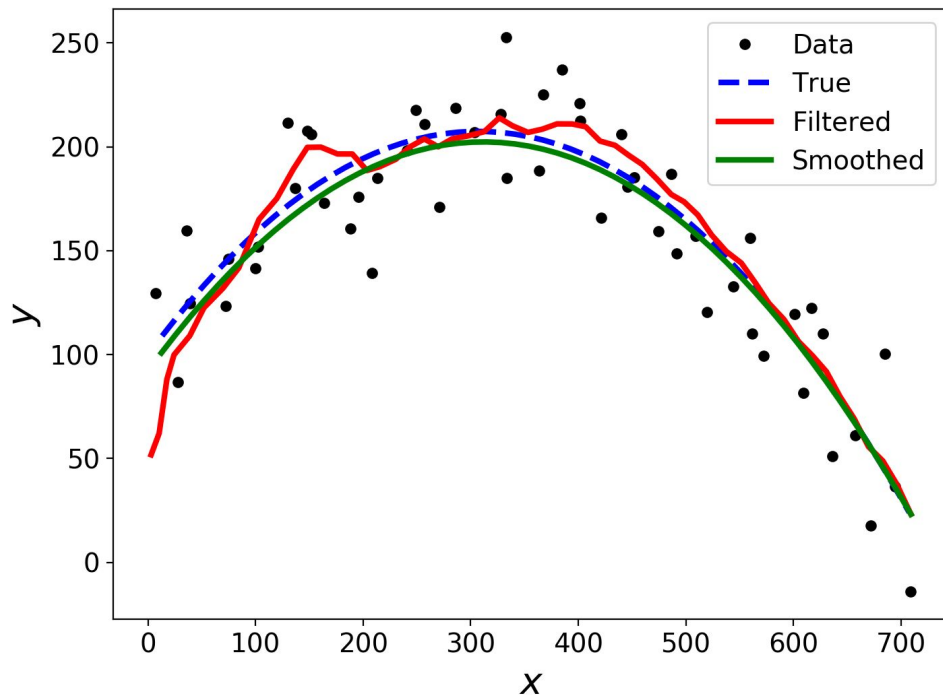
- In a LGSSM we can compute this filtered posterior distribution in an exact way (!!!) with the *Kalman filtering* algorithm

# Kalman filtering



# Posterior inference (smoothing)

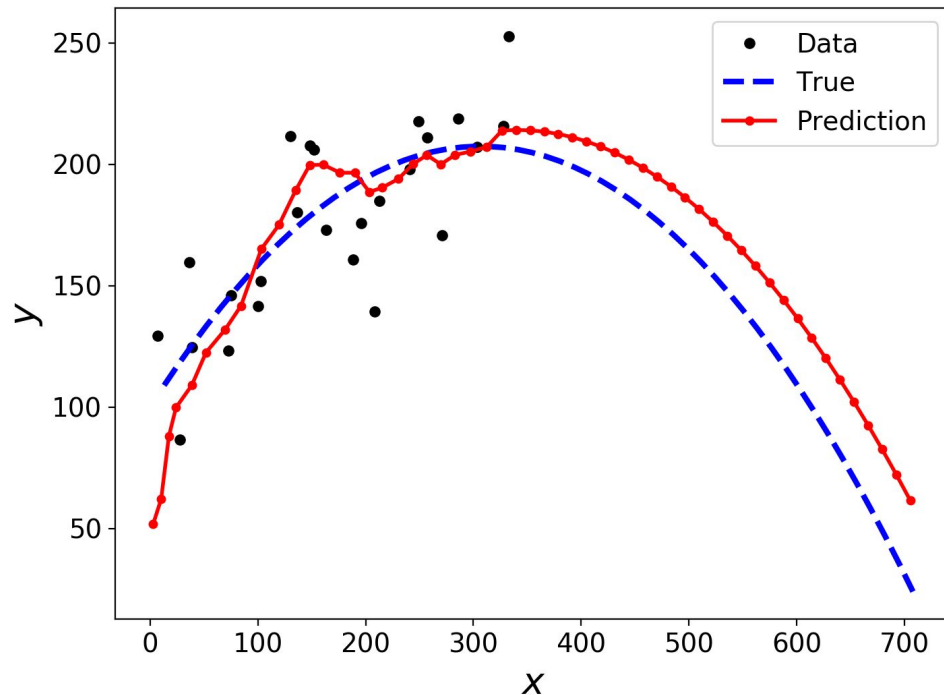
Once we have observed the whole trajectory we can further refine the estimated trajectory at each time step using information from the future observations as well.



- In a LGSSM we can compute the smoothed posterior distribution in an exact way:

$$p_{\theta}(\mathbf{z}_t | \mathbf{x}_{1:T})$$

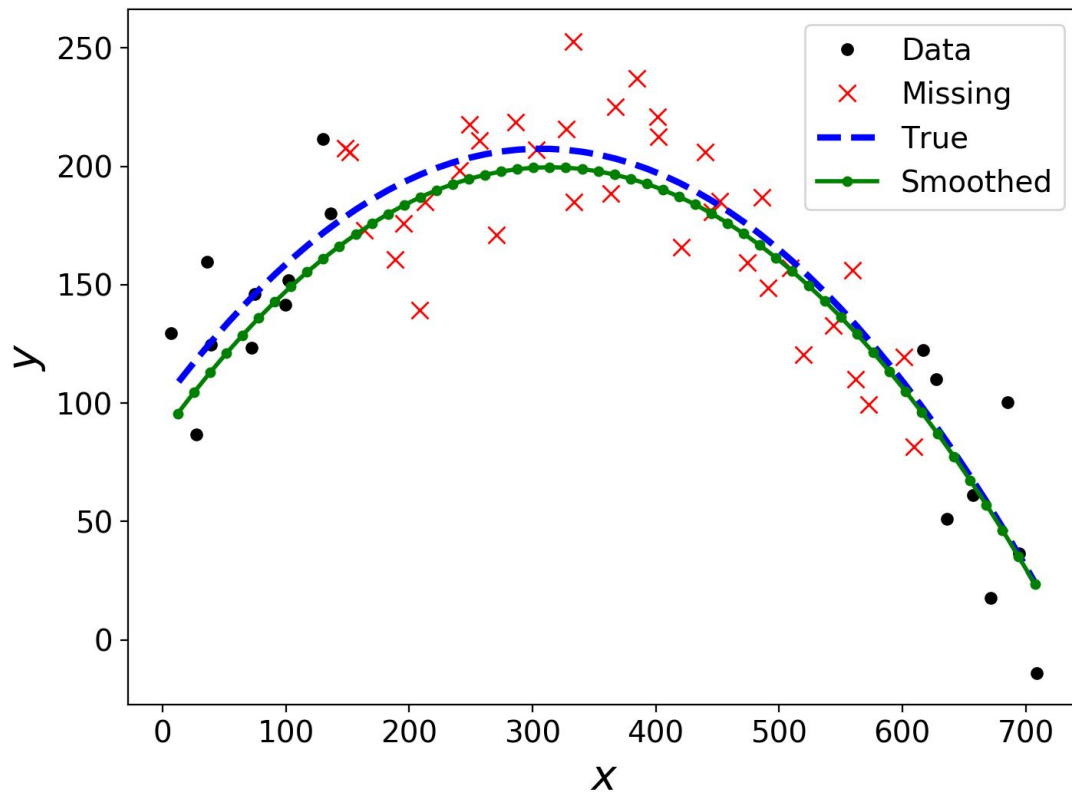
# Predictions



**k-steps ahead prediction:**

$$p_{\theta}(\mathbf{z}_{t+k} | \mathbf{x}_{1:t})$$

# Missing data imputation



# Summary - Linear Gaussian state-space models



- Exact filtering, smoothing and missing data imputation
- Generalizes many common time-series models (e.g. ARIMA)

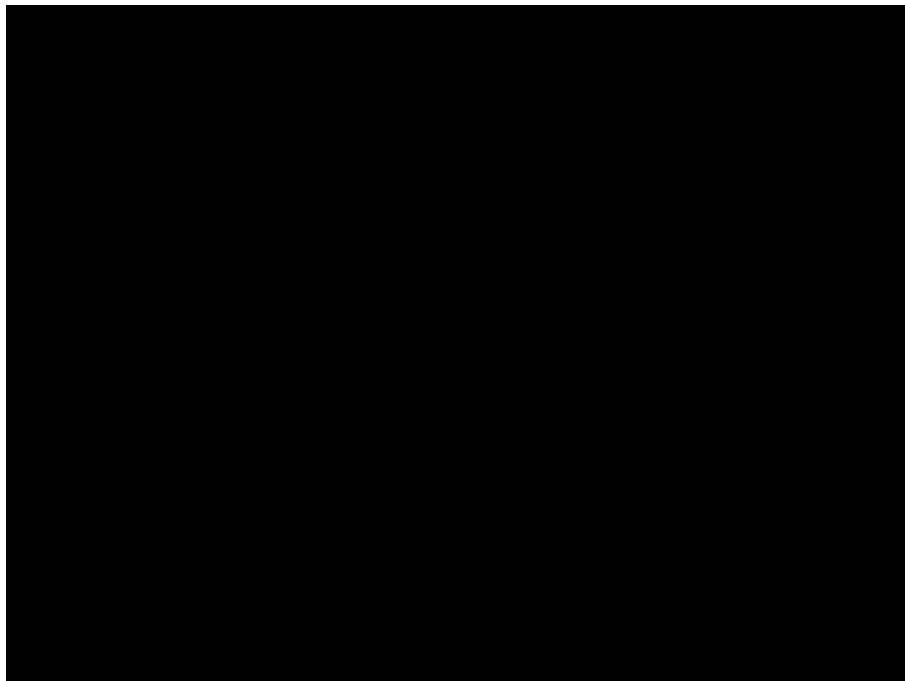


- Strong modelling assumptions:

- Linear transitions and emissions
- Gaussian transitions and measurement noise

We need to  
relax them!

# Unsupervised learning from videos



- We want to learn a generative model for these videos in an unsupervised way
- Much harder task than before:
  - Observations are 1024-dimensional vectors (32x32 images)
  - Non-linear transitions due to the walls
  - We want to learn ALL parameters from data

# Predicting the ball's path from sensory input

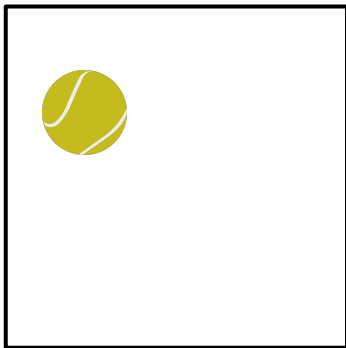
## How do humans do it?

1. We know what a ball looks like, and can easily identify its position in the frames of the video
2. We have a great intuition of Newtonian dynamics
  - a. We can easily predict where the ball will go
3. When the ball bounces, we know exactly how the trajectory will change

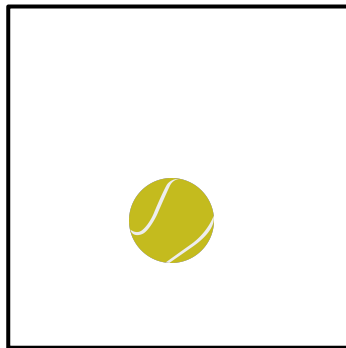
## How can a mathematical model do it?

1. From pixel space to a 2-dimensional latent variable representing the noisy positions
2. Given the *latent* noisy positions we can use a LGSSM
  - a. LGSSM can be used to make predictions
3. The model needs to learn when the ball will bounce and how this will change its *state* vector

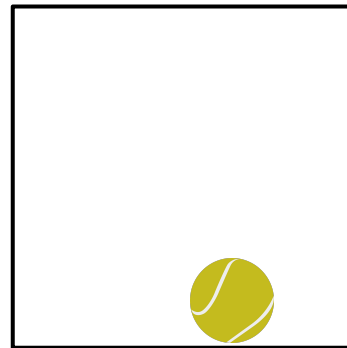
$t = 1$

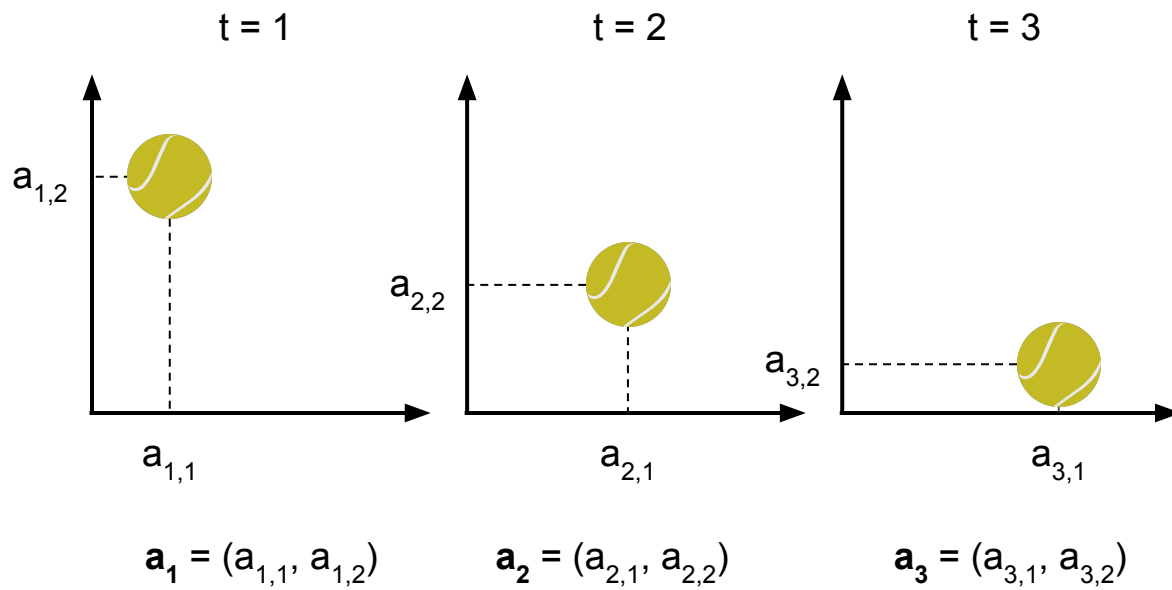


$t = 2$



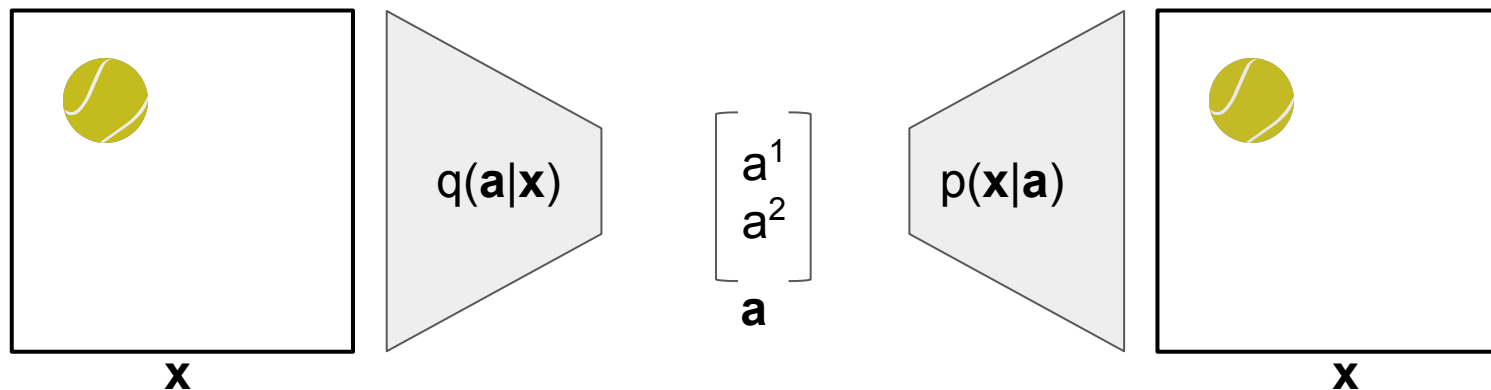
$t = 3$



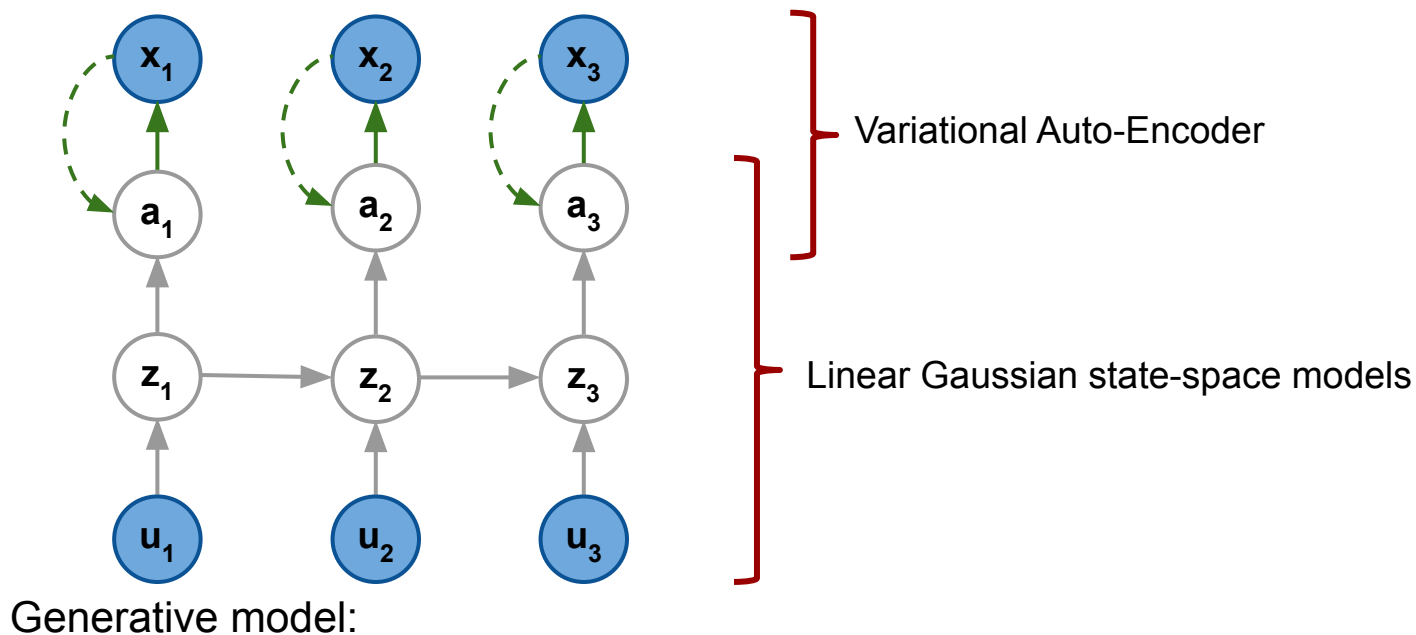


# Variational Auto-Encoder (VAE)

We use a VAE to go from pixel space to a 2-dimensional latent variable representing the noisy positions.



# Kalman Variational Auto-Encoders



$$p(\mathbf{x}, \mathbf{a}, \mathbf{z} | \mathbf{u}) = p_{\theta}(\mathbf{x} | \mathbf{a}) p_{\gamma}(\mathbf{a} | \mathbf{z}) p_{\gamma}(\mathbf{z} | \mathbf{u})$$

[Fraccaro\* M., Kamronn\* S., Paquet U., Winther O. *A Disentangled Recognition and Nonlinear Dynamics Model for Unsupervised Learning*. NIPS 2017.]

# Inference and parameter learning for the KVAE

1. We approximate the intractable posterior  $p(\mathbf{a}, \mathbf{z}|\mathbf{x}, \mathbf{u})$  introducing a variational distribution that factorizes as follows:

$$q(\mathbf{a}, \mathbf{z}|\mathbf{x}, \mathbf{u}) = \prod_{t=1}^T q_{\phi}(\mathbf{a}_t|\mathbf{x}_t) p_{\gamma}(\mathbf{z}|\mathbf{a}, \mathbf{u})$$

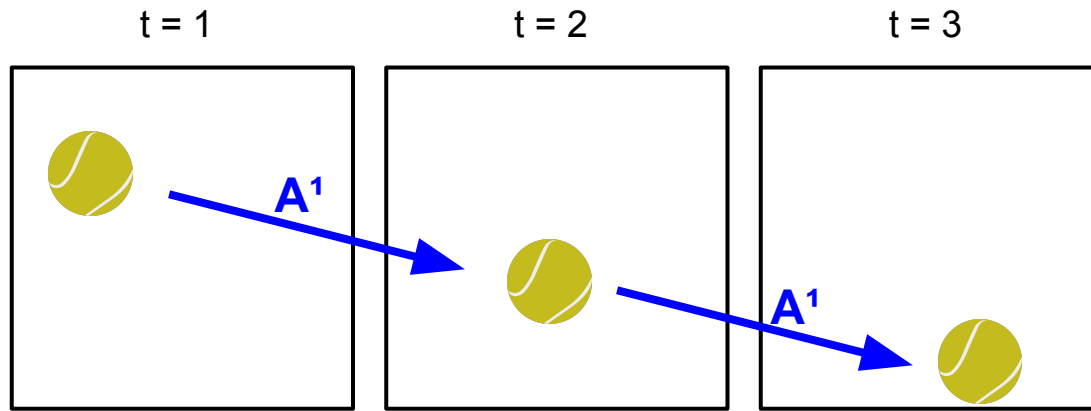
Here we leverage our knowledge of the exact smoothed posterior for the LGSSM.

2. We use Jensen's inequality and the variational approximation to define a lower bound to the intractable log-likelihood of a training set,  $\log p(\mathbf{x}|\mathbf{u})$

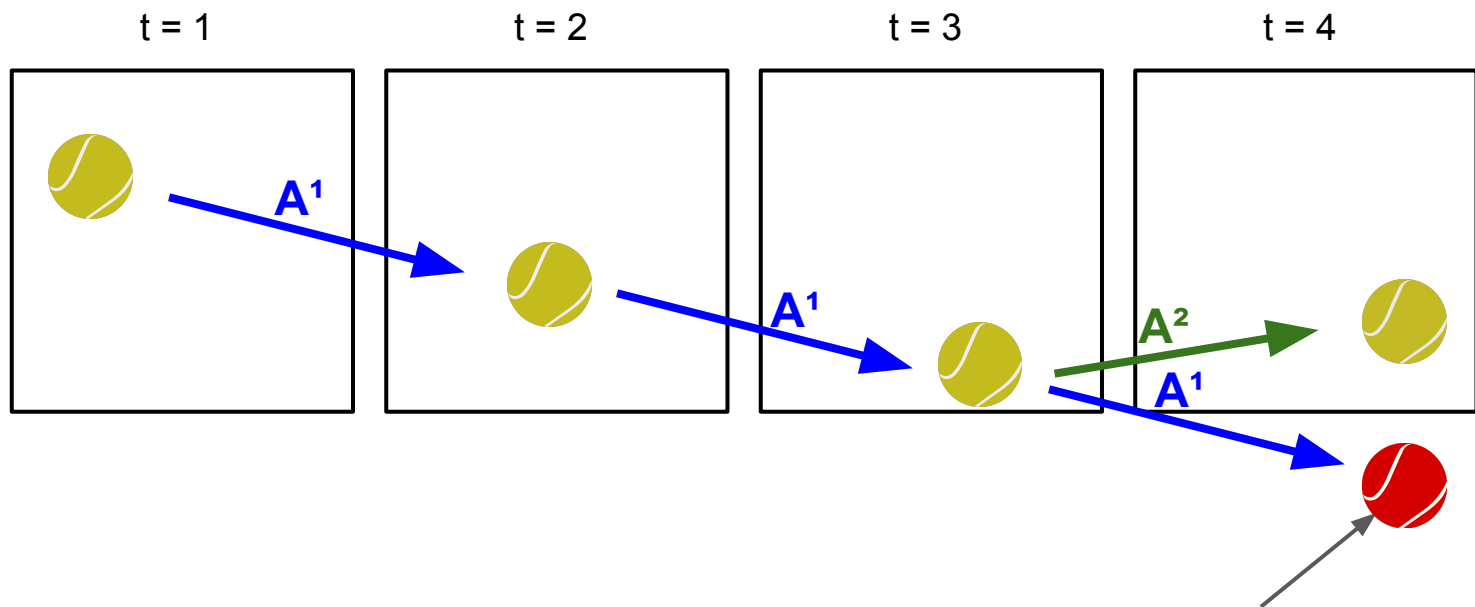
$$\begin{aligned} \log p(\mathbf{x}|\mathbf{u}) &\geq \mathbb{E}_{q(\mathbf{a}, \mathbf{z}|\mathbf{x}, \mathbf{u})} \left[ \log \frac{p_{\theta}(\mathbf{x}|\mathbf{a}) p_{\gamma}(\mathbf{a}|\mathbf{z}) p_{\gamma}(\mathbf{z}|\mathbf{u})}{q(\mathbf{a}, \mathbf{z}|\mathbf{x}, \mathbf{u})} \right] \\ &= \mathcal{F}(\theta, \gamma, \phi) , \end{aligned}$$

3. The parameters of the model and the variational approximation can be found by jointly maximizing the lower bound with stochastic gradient ascent

# Non-linear transitions



# Non-linear transitions



Wrong prediction  
by linear model

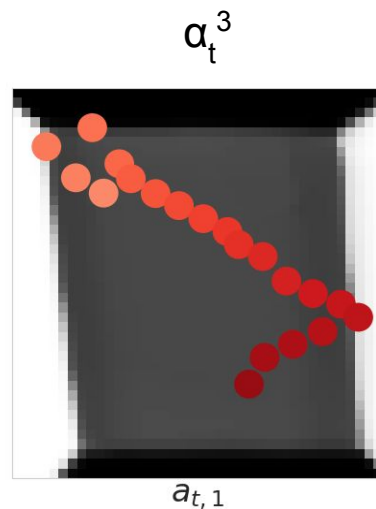
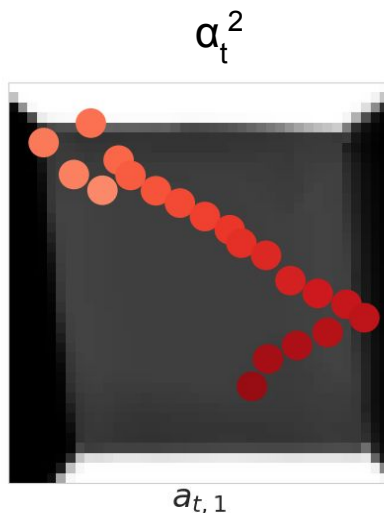
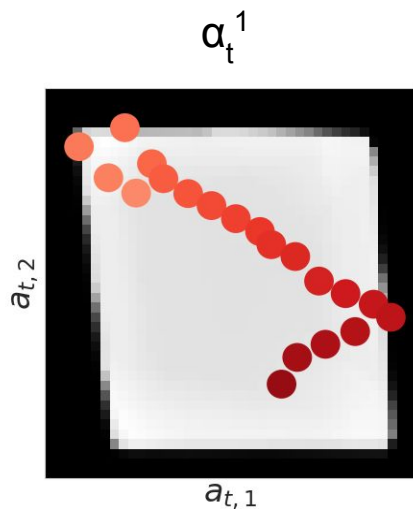
# Dynamically parameterized LGSSM

Transition matrix is a weighted sum:

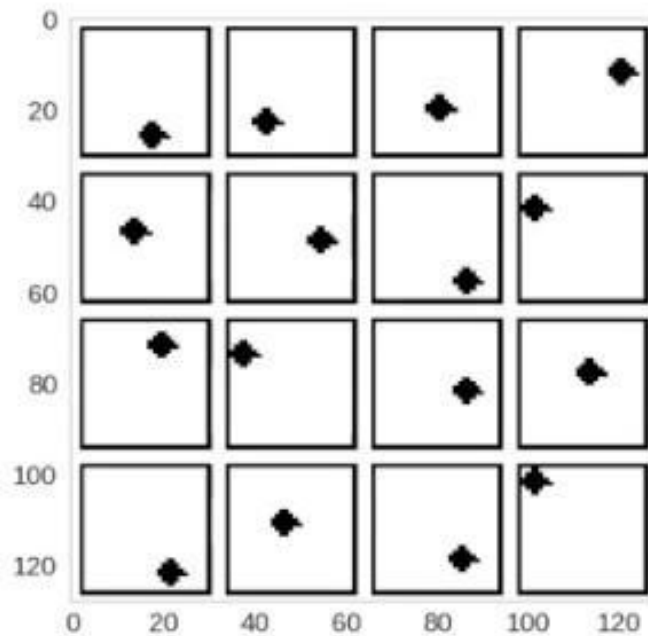
$$\mathbf{A}_t = \alpha_t^1 \mathbf{A}^1 + \alpha_t^2 \mathbf{A}^2 + \alpha_t^3 \mathbf{A}^3$$

Mixture weights as a function of  $\mathbf{a}$

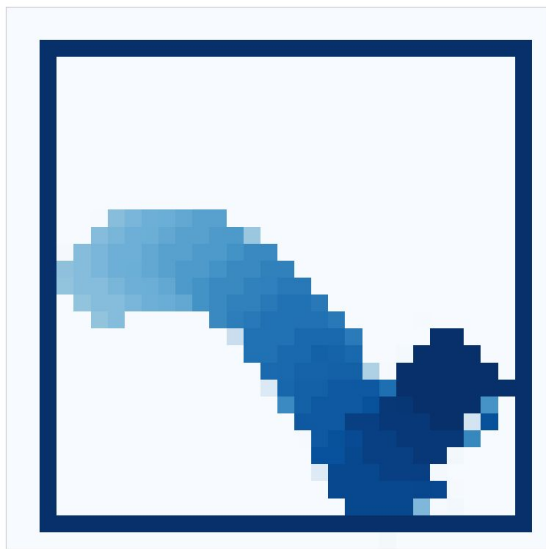
$$\alpha_t = \text{RNN}(\mathbf{a}_{0:t-1})$$



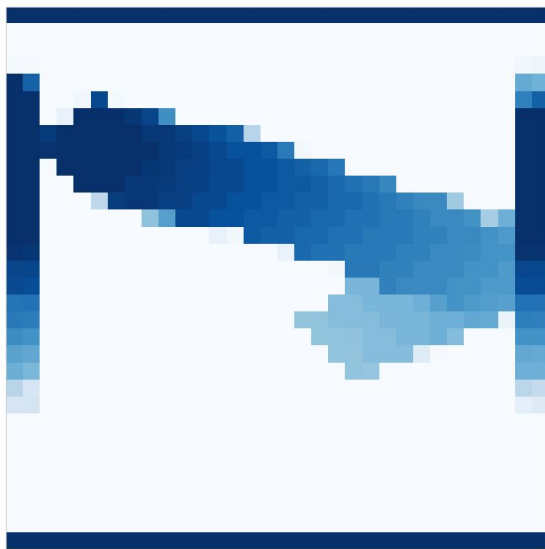
# Long-term generation



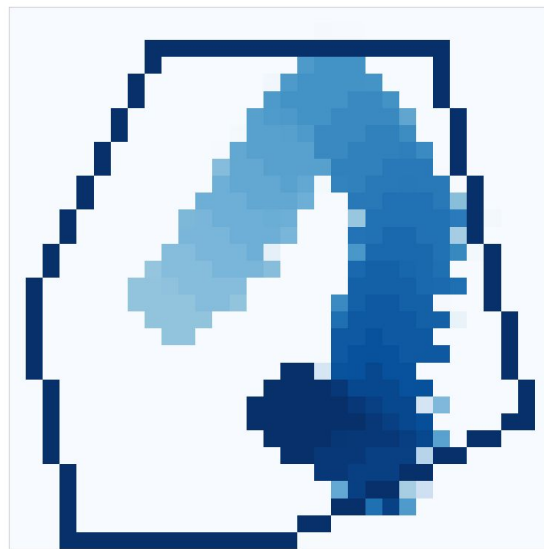
# Results in different environments



Gravity



Pong



Polygon

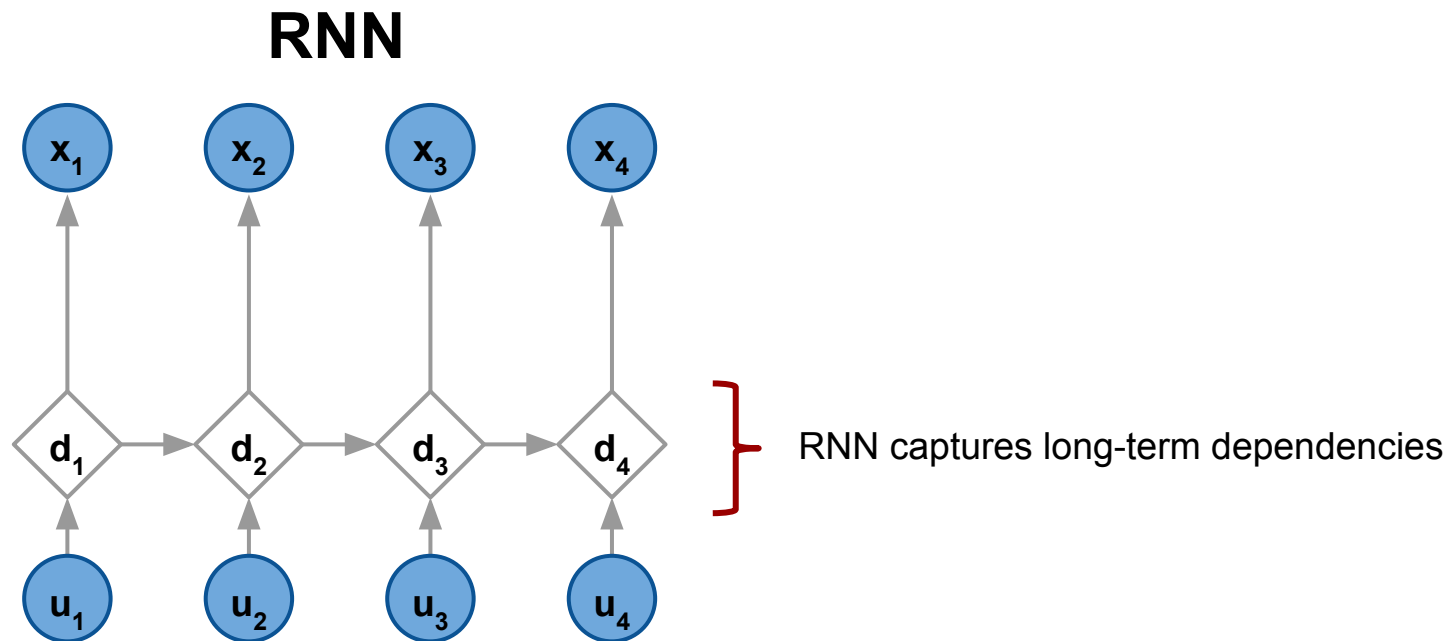
# Summary - Kalman Variational Auto-Encoder



- We can model high-dimensional videos in an unsupervised way
  - Learning everything from data
- Interpretability (disentangled visual and dynamic representations)
- Does not require a lot of data

- Model is not very flexible (e.g. it cannot model speech)
  - Transition model cannot handle complex non-linear dynamics => Non-linear SSM
  - Difficult to capture long-term dependencies in the data => RNN

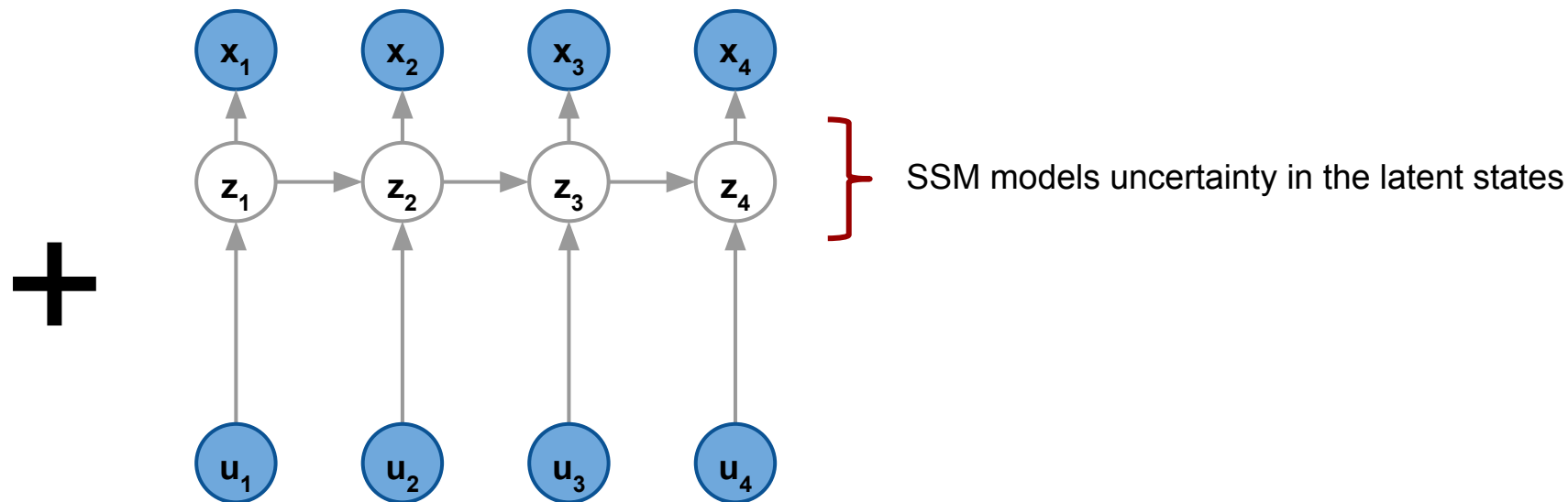
# Stochastic recurrent neural networks (SRNNs)



Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet and Ole Winther. *Sequential Neural Models with Stochastic Layers*. NIPS 2016.

# Stochastic recurrent neural networks (SRNNs)

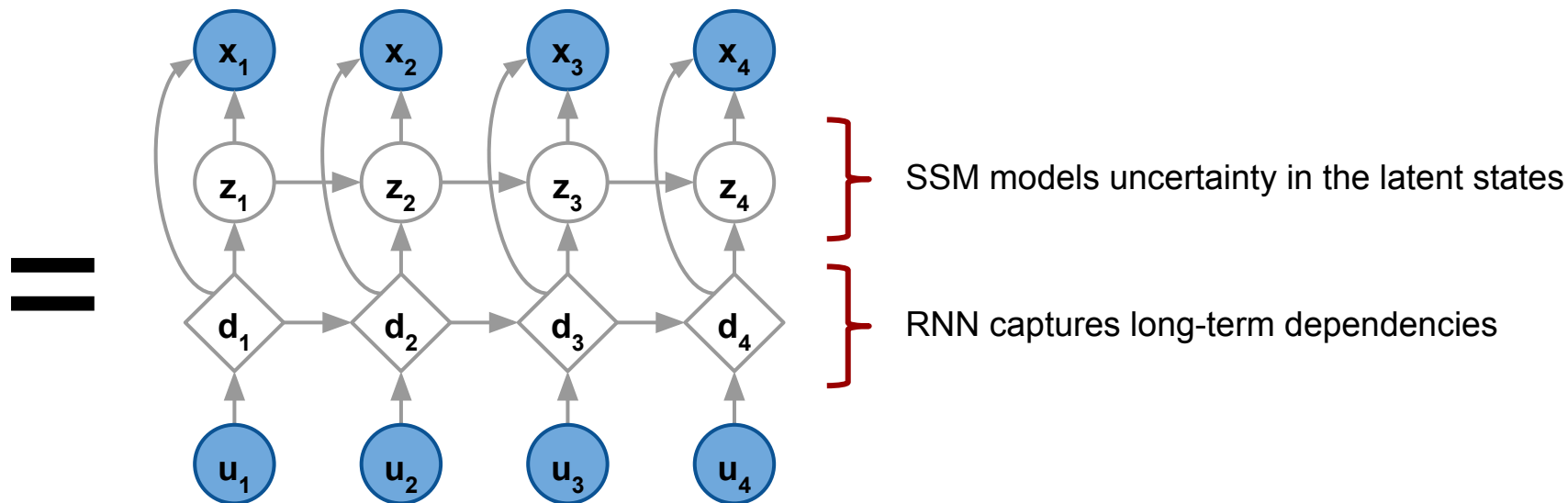
## Non-linear SSM



Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet and Ole Winther. *Sequential Neural Models with Stochastic Layers*. NIPS 2016.

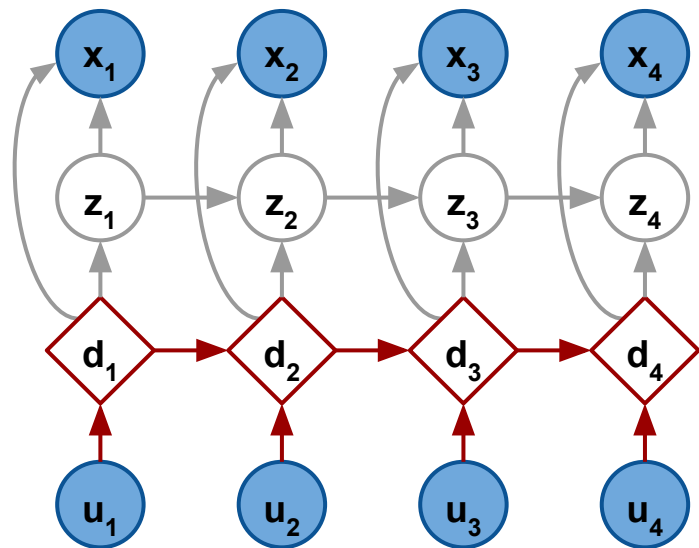
# Stochastic recurrent neural networks (SRNNs)

## SRNN



[Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet and Ole Winther. *Sequential Neural Models with Stochastic Layers*. NIPS 2016.]

# Stochastic recurrent neural networks (SRNNs)



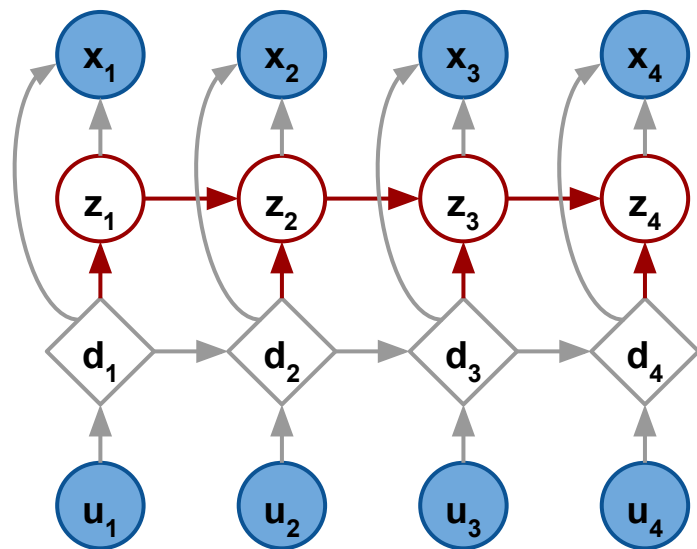
Deterministic transitions (GRU, LSTM, ...)

$$\mathbf{d}_t = f_{\theta}(\mathbf{d}_{t-1}, \mathbf{u}_t)$$

$\Downarrow$

$$p_{\theta}(\mathbf{d}_t | \mathbf{d}_{t-1}, \mathbf{u}_t) = \delta(\mathbf{d}_t - f_{\theta}(\mathbf{d}_{t-1}, \mathbf{u}_t))$$

# Stochastic recurrent neural networks (SRNNs)



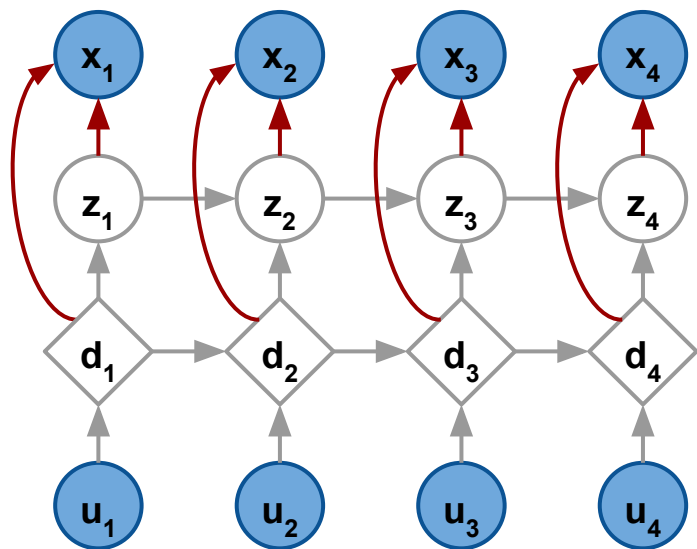
## Stochastic transitions

$$p_{\theta}(z_t | z_{t-1}, d_t) = \mathcal{N}(\mu_t, \mathbf{v}_t)$$

$$\mu_t = \text{NN}_1(z_{t-1}, d_t)$$

$$\log \mathbf{v}_t = \text{NN}_2(z_{t-1}, d_t)$$

# Stochastic recurrent neural networks (SRNNs)



Output probabilities

$$p_{\theta}(\mathbf{x}_t | \mathbf{z}_t, \mathbf{d}_t)$$

# Inference and parameter learning for the SRNN

1. We approximate the intractable posterior  $p_{\theta}(\mathbf{z}, \mathbf{d}|\mathbf{x}, \mathbf{u})$  introducing a variational distribution  $q_{\phi}(\mathbf{z}, \mathbf{d}|\mathbf{x}, \mathbf{u})$  (an inference network)
2. We use Jensen's inequality and the variational approximation to define a lower bound to the intractable log-likelihood of a training set,

$$\log p_{\theta}(\mathbf{x}|\mathbf{u}) \geq \mathcal{F}(\theta, \phi)$$

$$\mathcal{F}(\theta, \phi) = \mathbb{E}_{q_{\phi}(\mathbf{z}, \mathbf{d}|\mathbf{x}, \mathbf{u})} [\log p_{\theta}(\mathbf{x}|\mathbf{z}, \mathbf{d})] - KL [q_{\phi}(\mathbf{z}, \mathbf{d}|\mathbf{x}, \mathbf{u}) || p_{\theta}(\mathbf{z}, \mathbf{d}|\mathbf{u})]$$

3. The optimal  $\theta$  and  $\phi$  can be found by jointly maximizing the lower bound

# Posterior distribution

Due to the deterministic nature of the RNN layer the posterior factorizes as

$$p_{\theta}(\mathbf{z}, \mathbf{d} | \mathbf{x}, \mathbf{u}) = p_{\theta}(\mathbf{d} | \mathbf{u}) p_{\theta}(\mathbf{z} | \mathbf{d}, \mathbf{x})$$


Prior RNN transition probabilities:

$$p_{\theta}(\mathbf{d} | \mathbf{u}) = \prod_t \delta(\mathbf{d}_t - f_{\theta}(\mathbf{d}_{t-1}, \mathbf{u}_t))$$

Smoothed posterior over the states of the SSM:

$p_{\theta}(\mathbf{z} | \mathbf{d}, \mathbf{x})$  is *intractable*, but it can be simplified exploiting some independence properties given by the temporal structure of the model.

# Variational approximation to the posterior

Using the conditional independence properties of the model we can factorize

$$p_{\theta}(\mathbf{z}, \mathbf{d} | \mathbf{x}, \mathbf{u}) = p_{\theta}(\mathbf{d} | \mathbf{u}) \prod_t p_{\theta}(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{d}_{t:T}, \mathbf{x}_{t:T})$$

# Variational approximation to the posterior

Using the conditional independence properties of the model we can factorize

$$p_{\theta}(\mathbf{z}, \mathbf{d} | \mathbf{x}, \mathbf{u}) = \underbrace{p_{\theta}(\mathbf{d} | \mathbf{u})}_{\text{red}} \prod_t p_{\theta}(\mathbf{z}_t | \underbrace{\mathbf{z}_{t-1}}_{\text{blue}}, \underbrace{\mathbf{d}_{t:T}, \mathbf{x}_{t:T}}_{\text{green}})$$

Variational approximation:

$$q_{\phi}(\mathbf{z}, \mathbf{d} | \mathbf{x}, \mathbf{u}) = \underbrace{p_{\theta}(\mathbf{d} | \mathbf{u})}_{\text{red}} \prod_t q_{\phi}(\mathbf{z}_t | \underbrace{\mathbf{z}_{t-1}}_{\text{blue}}, \underbrace{\mathbf{a}_t}_{\text{green}})$$

# Variational approximation to the posterior

Using the conditional independence properties of the model we can factorize

$$p_{\theta}(\mathbf{z}, \mathbf{d} | \mathbf{x}, \mathbf{u}) = \underbrace{p_{\theta}(\mathbf{d} | \mathbf{u})}_{\text{red}} \prod_t p_{\theta}(\mathbf{z}_t | \underbrace{\mathbf{z}_{t-1}}_{\text{blue}}, \underbrace{\mathbf{d}_{t:T}, \mathbf{x}_{t:T}}_{\text{green}})$$

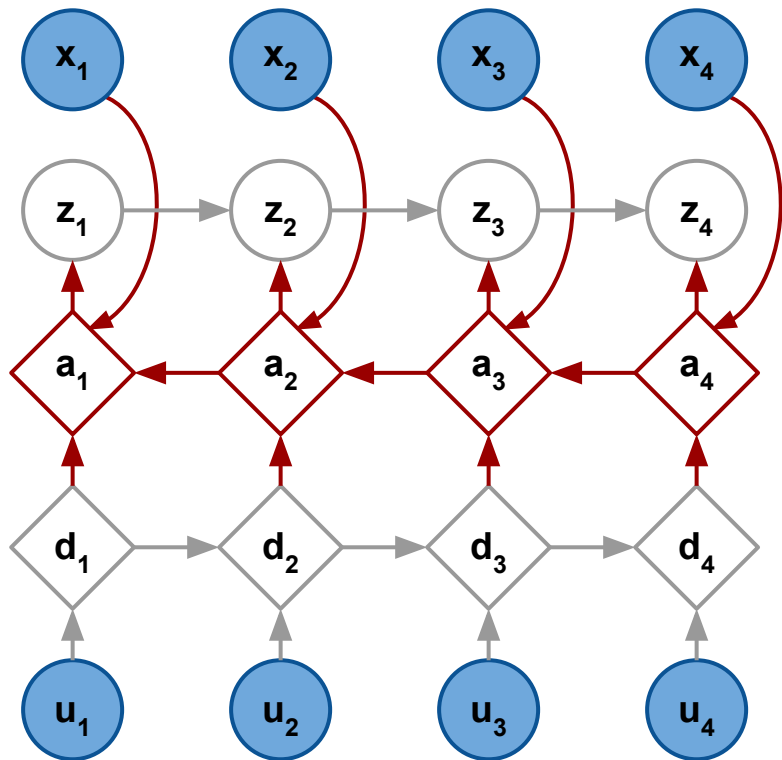
Variational approximation:

$$q_{\phi}(\mathbf{z}, \mathbf{d} | \mathbf{x}, \mathbf{u}) = \underbrace{p_{\theta}(\mathbf{d} | \mathbf{u})}_{\text{red}} \prod_t q_{\phi}(\mathbf{z}_t | \mathbf{z}_{t-1}, \underbrace{\mathbf{a}_t}_{\text{green}})$$

The inference network  $q_{\phi}$  approximates the dependence of  $\mathbf{z}_t$  on  $\mathbf{d}_{t:T}$  and  $\mathbf{x}_{t:T}$  by introducing **auxiliary deterministic states**  $\mathbf{a}_t$  that belong to an RNN running backwards in time:

$$\mathbf{a}_t = g_{\phi}(\mathbf{a}_{t+1}, \begin{bmatrix} \mathbf{d}_t \\ \mathbf{x}_t \end{bmatrix})$$

# Inference network



$$q_{\phi}(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{a}_t) = \mathcal{N}(\boldsymbol{\mu}_t^{(q)}, \mathbf{v}_t^{(q)})$$

$$\boldsymbol{\mu}_t^{(q)} = \text{NN}_1^{(q)}(\mathbf{z}_{t-1}, \mathbf{a}_t)$$

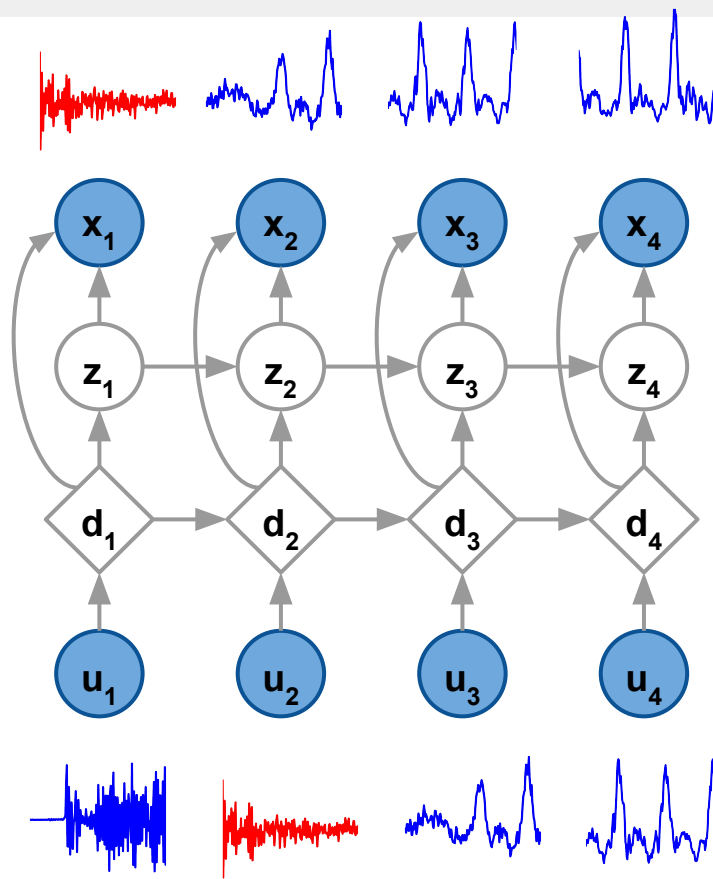
$$\log \mathbf{v}_t^{(q)} = \text{NN}_2^{(q)}(\mathbf{z}_{t-1}, \mathbf{a}_t)$$

# Speech modeling results

Two data sets of raw waveforms:

- **Blizzard**: 300 hours of English, spoken by a single female speaker.
- **TIMIT**: 6300 English sentences read by 630 speakers.

In these experiments we set  $\mathbf{u}_t = \mathbf{x}_{t-1}$ , but  $\mathbf{u}_t$  could also be used to represent additional input information to the model.

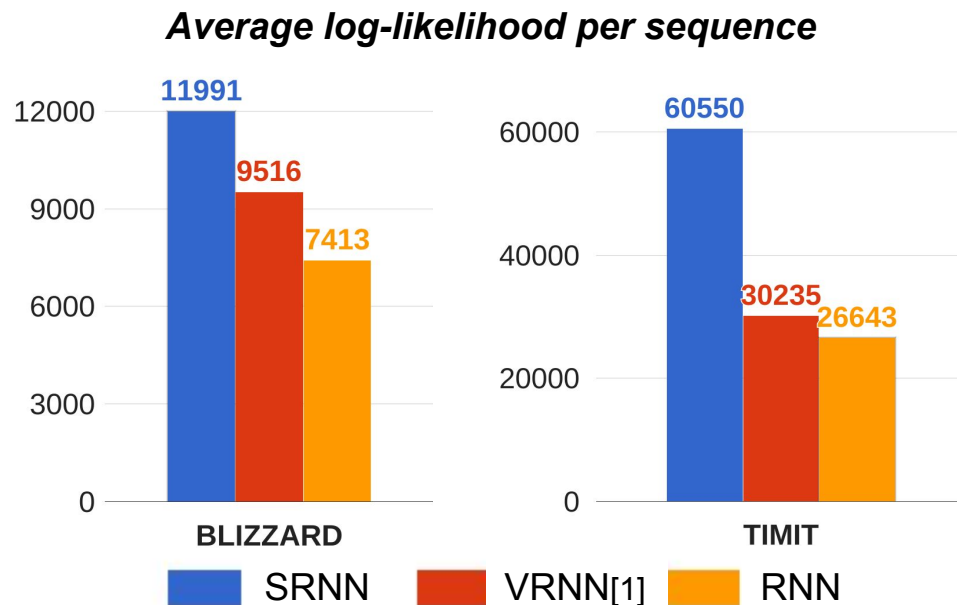


# Speech modeling results

Two data sets of raw waveforms:

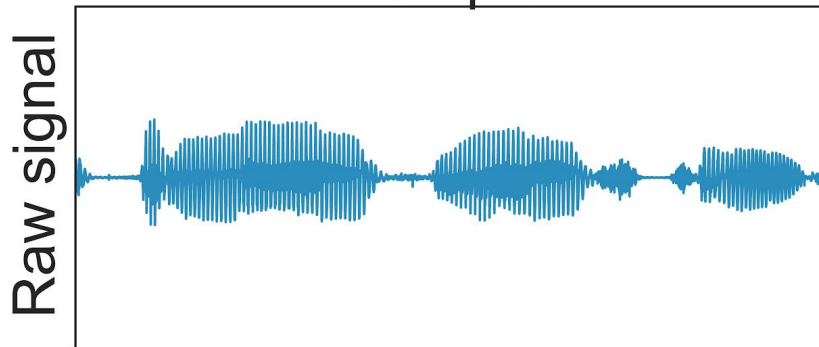
- **Blizzard**: 300 hours of English, spoken by a single female speaker.
- **TIMIT**: 6300 English sentences read by 630 speakers.

In these experiments we set  $\mathbf{u}_t = \mathbf{x}_{t-1}$ , but  $\mathbf{u}_t$  could also be used to represent additional input information to the model.

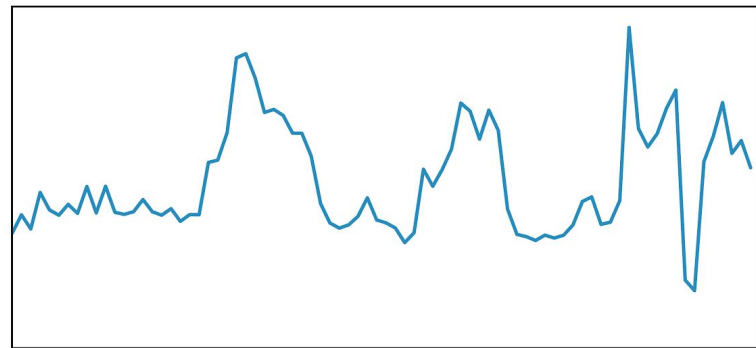
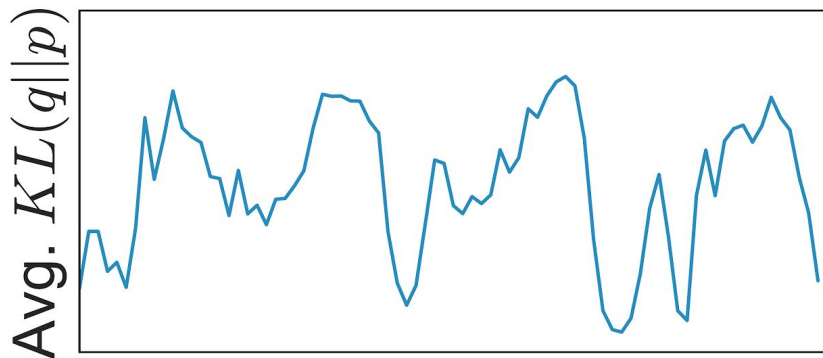
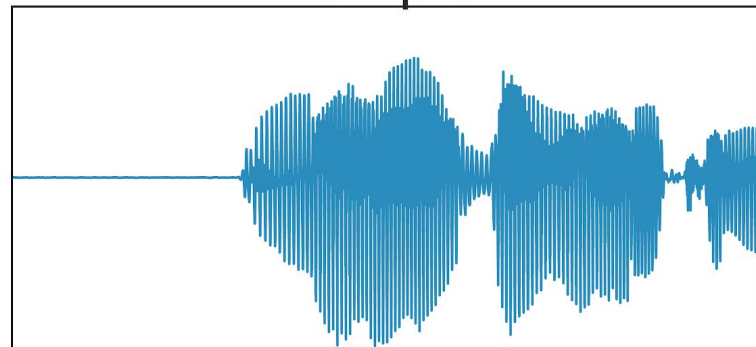


# Importance of the inference network

Example 1

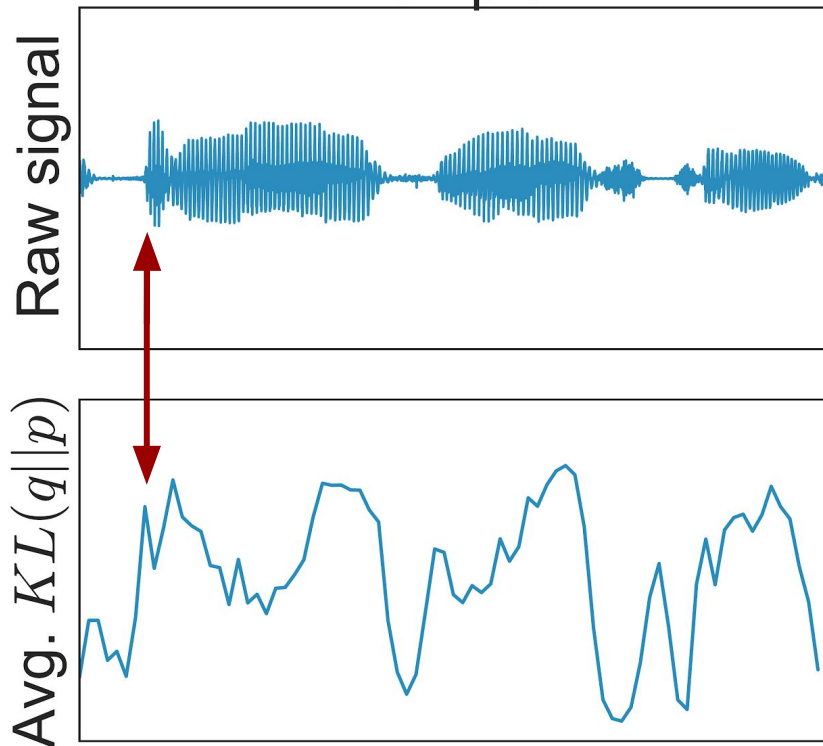


Example 2

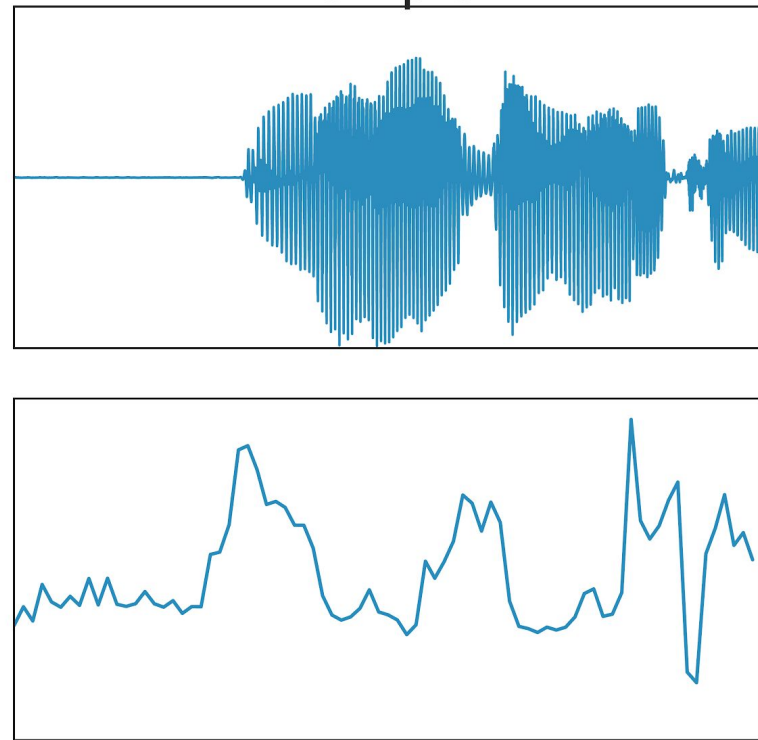


# Importance of the inference network

Example 1



Example 2



# Summary - Stochastic recurrent neural networks



- Flexible model, suitable to a wide range of applications
- Scalable inference thanks to the inference network

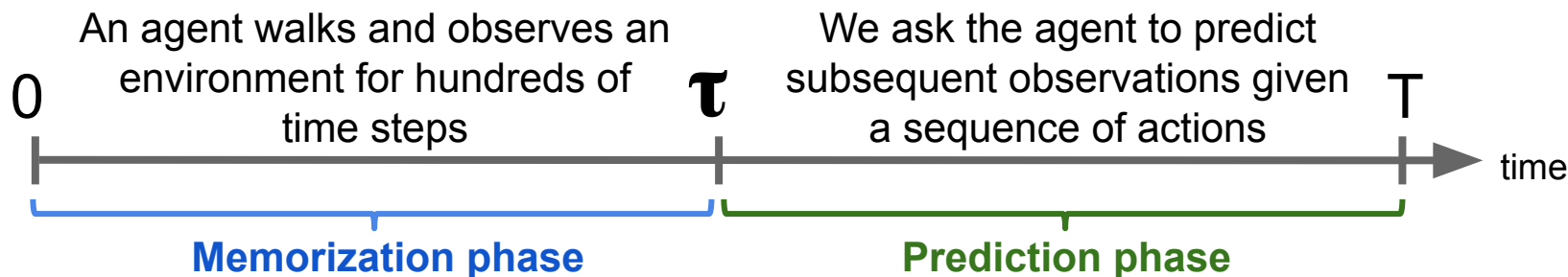


- Lots of parameters to learn, therefore we need lots of data
- Several strong assumptions are needed to ensure scalability during inference
- For some applications that require a *high memory capacity* (e.g. model-based reinforcement learning) the RNN is not scalable enough

↘  
We need external memory architectures

# Generative modelling task

We are given  $T$ -step videos with corresponding action sequences generated by an RL agent acting in an environment. Each video is split in two parts:



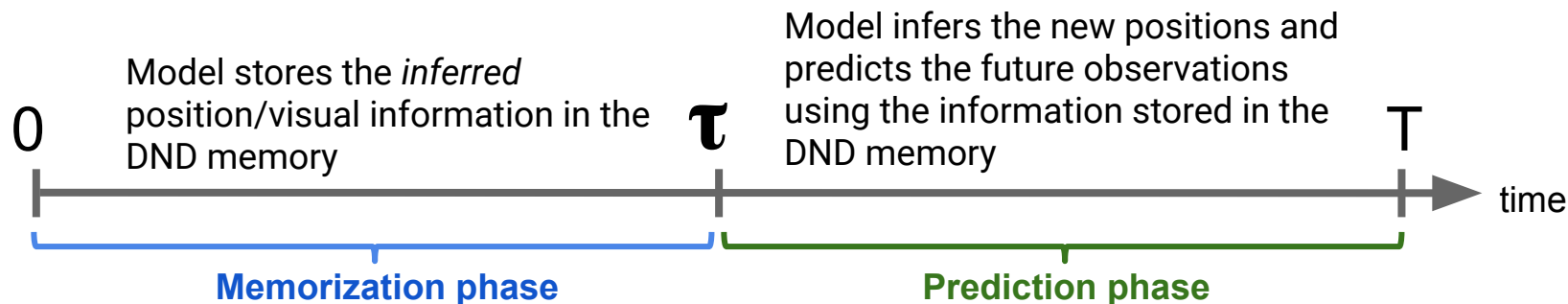
Model needs to be able to

- Remember the past over hundreds of time steps:
  - What the agent has seen
  - Where the agent has seen it
- Predict how a possibly long sequence of actions changes the position of the agent

# Generative Temporal Model with Spatial Memory (GTM-SM)

We introduce an action-conditioned generative models that uses:

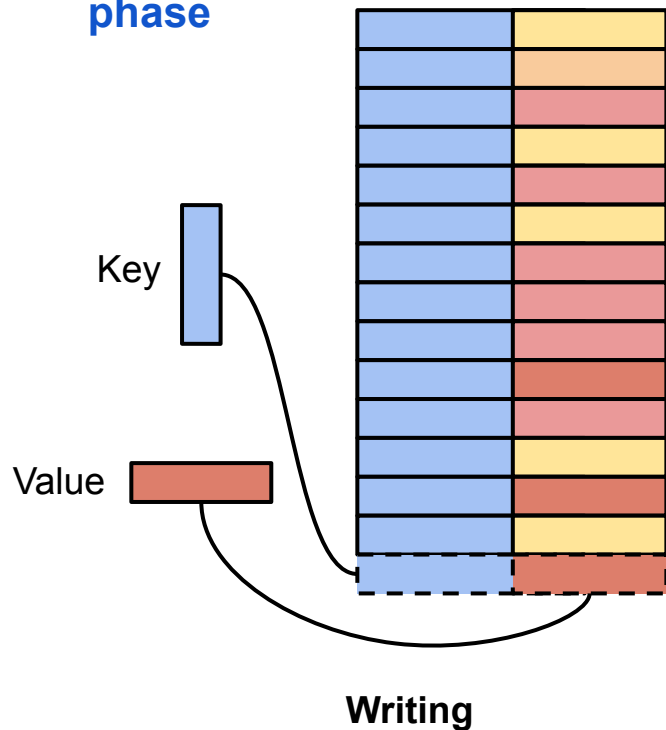
- **State-space model:** learns how to infer the position (state) of the agent given a sequence of actions, using knowledge on the prior dynamics of the moving agent
- **VAE:** learns how to encode the frames into a low-dimensional vector
- **Differentiable Neural Dictionary (DND):** memory that stores the information collected while exploring the environment



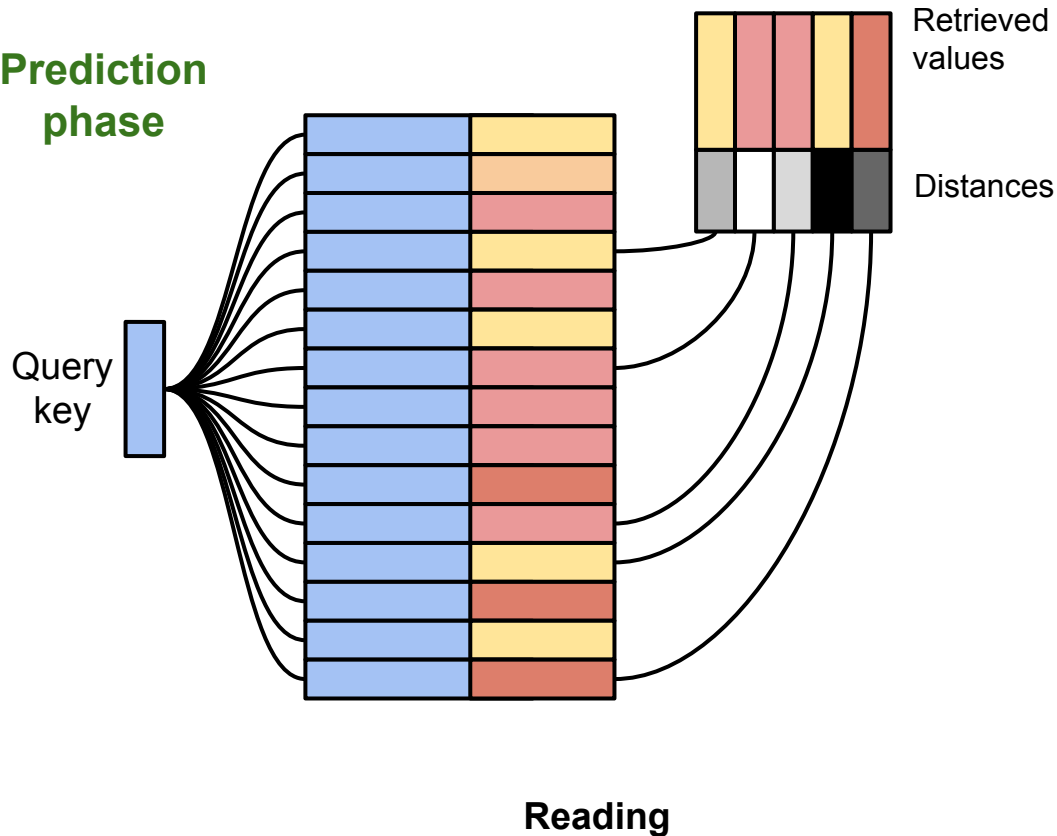
[Marco Fraccaro, Danilo Rezende, Yori Zwols, Alexander Pritzel, S. M. Ali Eslami and Fabio Viola. *Generative Temporal Models with Spatial Memory for Partially Observed Environments*, ICML 2018.]

# Differentiable Neural Dictionaries (DND)

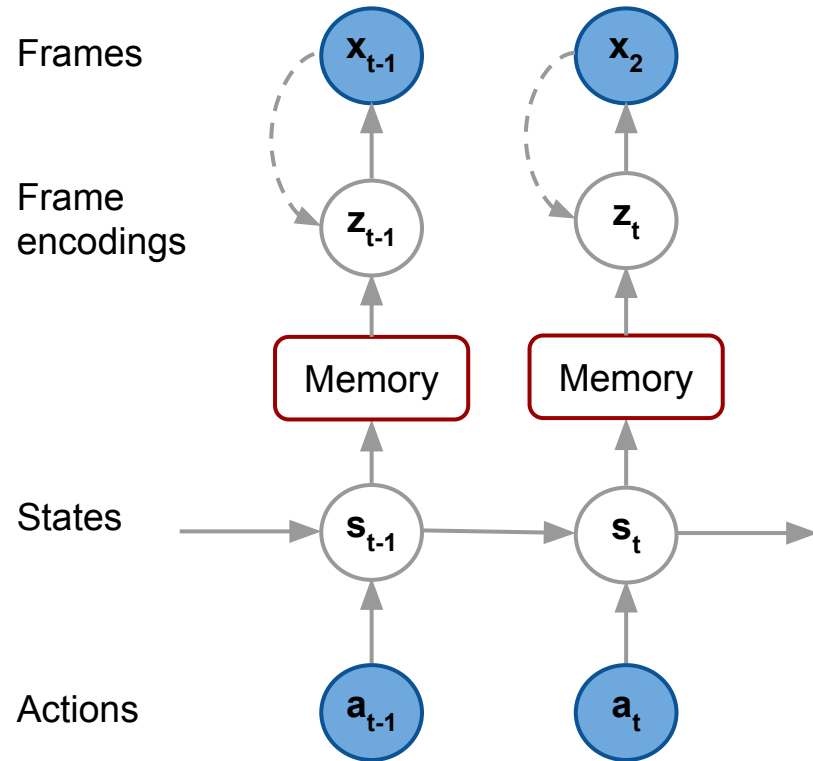
## Memorization phase



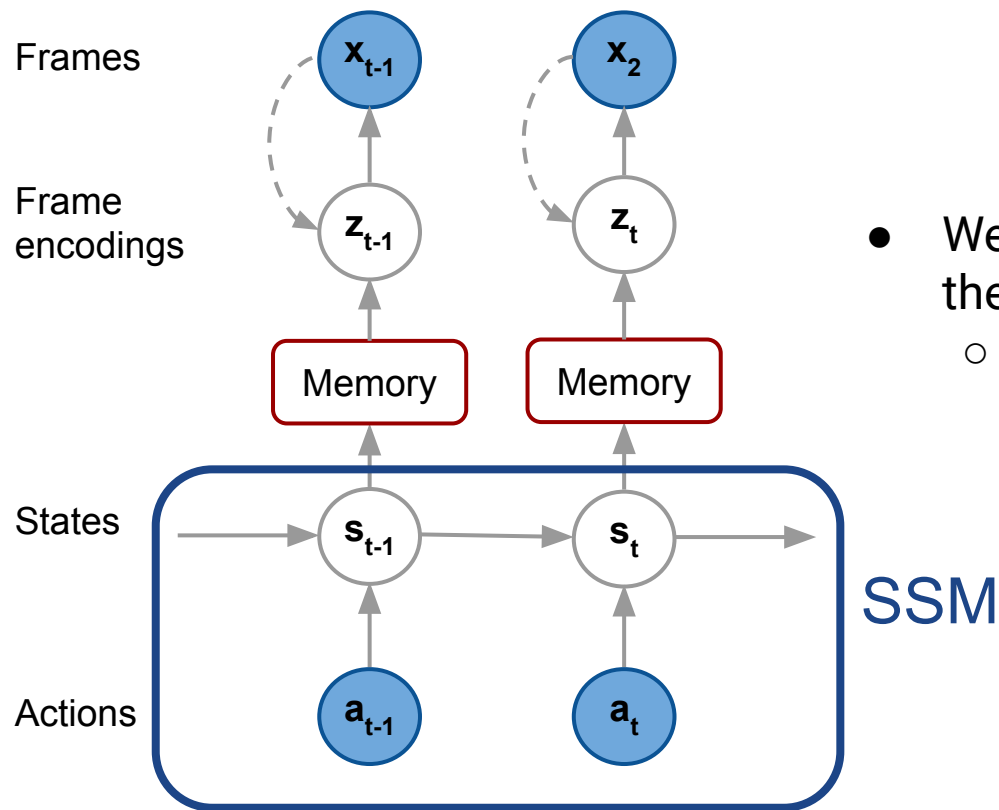
## Prediction phase



# Generative model

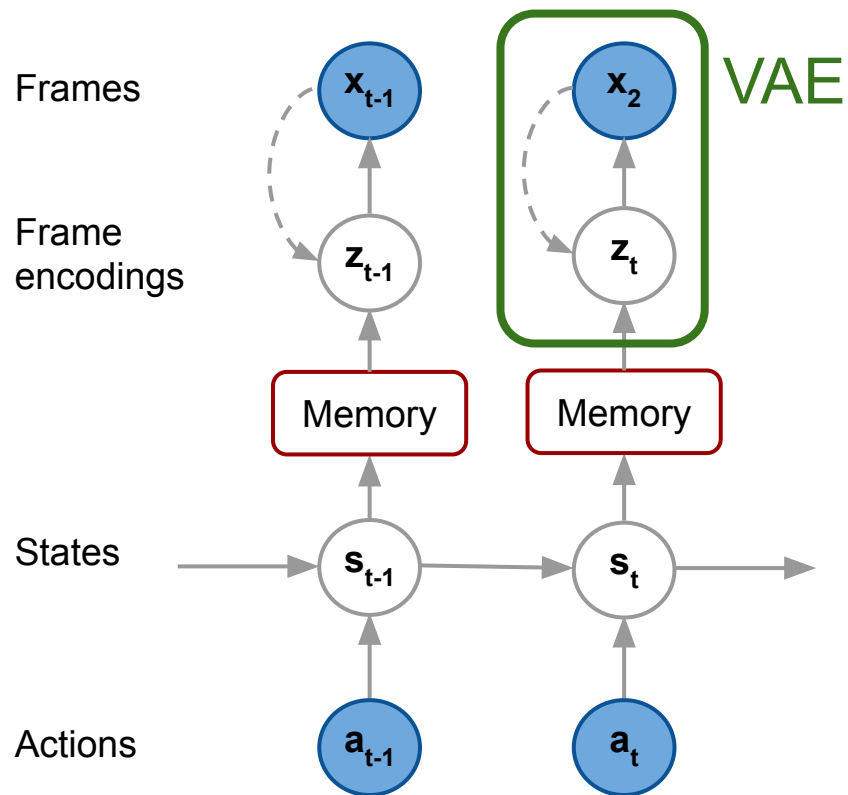


# Generative model



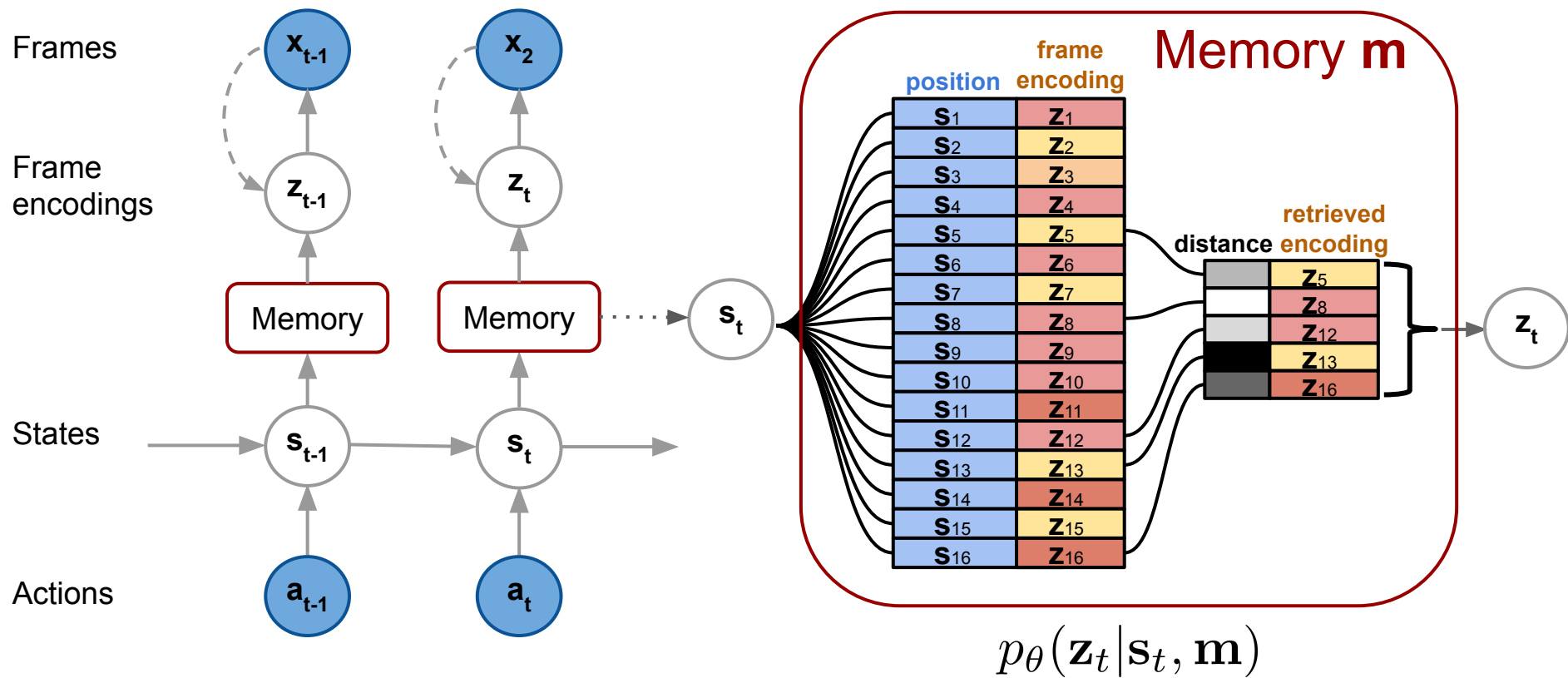
- We use a *state-space model* (SSM) to model the dynamics of the agent:
  - Transition density:  $p_{\theta}(\mathbf{s}_t | \mathbf{s}_{t-1}, \mathbf{a}_t)$

# Generative model



- At each time step we have a VAE that models the observed frames
  - Likelihood:  $p_{\theta}(\mathbf{x}_t | \mathbf{z}_t)$
  - Inference network:  $q_{\phi}(\mathbf{z}_t | \mathbf{x}_t)$
  - Conditional prior:  $p_{\theta}(\mathbf{z}_t | \mathbf{s}_t, \mathbf{m})$

# Generative model



# Inference and parameter learning for the GTM-SM

1. We approximate the intractable posterior introducing a variational distribution (an inference network):

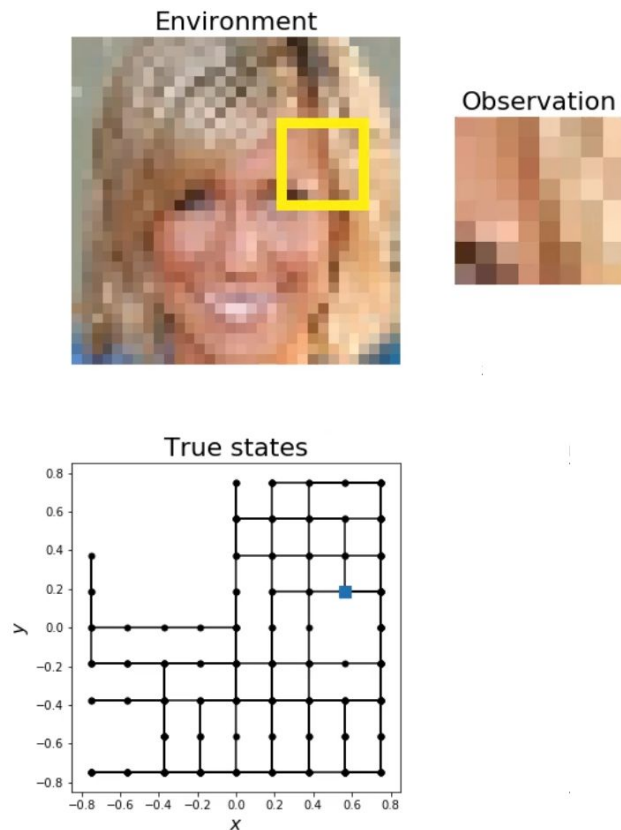
$$q_{\phi}(\mathbf{z}, \mathbf{s} | \mathbf{x}, \mathbf{a}) = q_{\phi}(\mathbf{z} | \mathbf{x}) p_{\theta}(\mathbf{s} | \mathbf{a})$$

2. We use Jensen's inequality and the variational approximation to define a lower bound to the intractable log-likelihood of a training set,

$$\begin{aligned} \log p_{\theta}(\mathbf{x} | \mathbf{a}, \mathbf{v}) &= \log \int p_{\theta}(\mathbf{x}, \mathbf{z}, \mathbf{s} | \mathbf{a}, \mathbf{v}) \, d\mathbf{z} \, d\mathbf{s} & \mathbf{v} &= \{\mathbf{x}_{1:\tau}, \mathbf{a}_{1:\tau}\} \\ &\geq \mathbb{E}_{q_{\phi}(\mathbf{z}, \mathbf{s} | \mathbf{x}, \mathbf{a}, \mathbf{v})} \left[ \log \frac{p_{\theta}(\mathbf{x}, \mathbf{z}, \mathbf{s} | \mathbf{a}, \mathbf{v})}{q_{\phi}(\mathbf{z}, \mathbf{s} | \mathbf{x}, \mathbf{a}, \mathbf{v})} \right] = \mathcal{F}(\theta, \phi) \end{aligned}$$

3. The optimal  $\theta$  and  $\phi$  can be found by jointly maximizing the lower bound

# Image navigation experiment



- Agent walks on top of an image
  - It can only observe an 8x8 crop of the image (yellow square)
- Long sequences:
  - We walk in the image while adding information in the DND for 256 time steps
  - We then generate for 256 time steps
- The environment has walls
  - We need non-linear transitions
- We use a 2-dimensional state space
  - Model learns to represent the position of the agent

# Long-term generation (Image Navigation experiment)

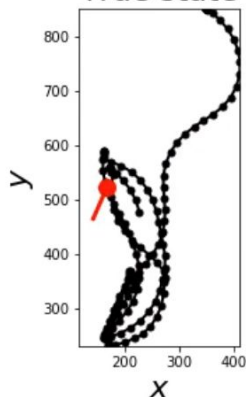


# Walking agent in Labyrinth

Observation

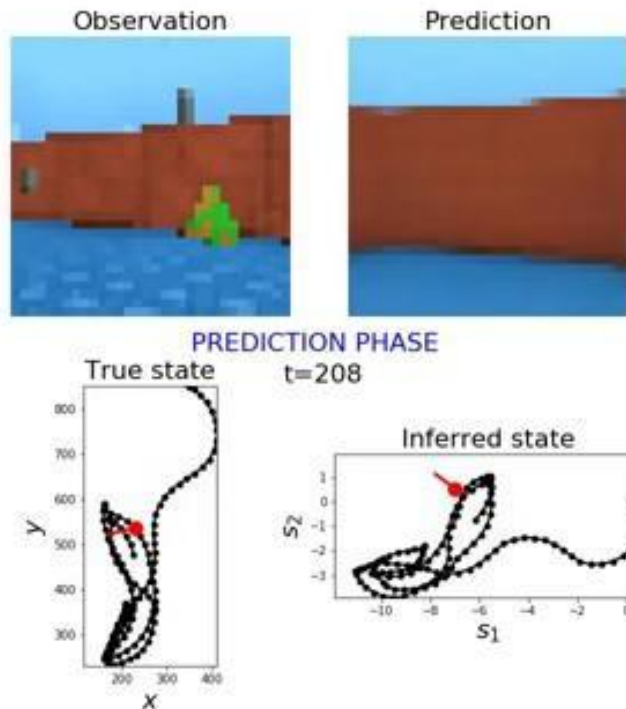


True state



- Action-conditioned videos of an agent walking in an 3D environment with a single room
- We only store in memory the views from the first 150 time steps, then ask the agent to generate the following 150 time steps
- Observation model needs to combine information from different views
  - We use a *Generative Query Network*
- We use a 3-dimensional state-space
  - Model learns to represent the position and orientation of the agent

# Long-term generation (Walking agent in Labyrinth)



# Summary - Generative Temporal Models with Spatial Memory



- The state-space assumption and the DND memory allow us to make coherent predictions over longer time scales
  - This can be used for planning in model-based reinforcement learning
  - In a scalable way



- The physics of the environment need to be known relatively well

# Conclusions

## *Deep Latent Variable Models:*

- **Flexible**
  - Can fit complex data distributions in a wide range of applications
  - Can easily incorporate advances in deep learning architectures
- **Scalable**: suitable for large-scale learning
  - Amortized Variational inference (inference networks)
  - GPU acceleration
- **“Easy” to implement**
  - Using existing deep learning libraries
  - Inference can be seen as a black-box operation (stochastic gradient descent)