Variational Auto-Encoders

Diederik P. Kingma

OpenAI

Introduction and Motivation

Motivation and applications

- Versatile framework for unsupervised and semi-supervised deep learning
- Representation Learning. E.g.:
  - 2D visualisation
  - Data-efficient learning. Semi-supervised learning
- Artificial Creativity. E.g.:
  - Image/text resynthesis, Molecule design

Sad Kanye -> Happy Kanye

- “Smile vector”. Tom White, 2016, twitter: @dribnet

Background

Probabilistic Models

- $\mathbf{x}$: Observed random variables
- $p^*(\mathbf{x})$ or: underlying unknown process
- $p_{\boldsymbol{\theta}}(\mathbf{x})$: model distribution
- Goal: $p_{\boldsymbol{\theta}}(\mathbf{x}) \approx p^*(\mathbf{x})$
  - We wish flexible $p_{\boldsymbol{\theta}}(\mathbf{x})$
- Conditional modeling goal: $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{y}) \approx p^*(\mathbf{x}|\mathbf{y})$

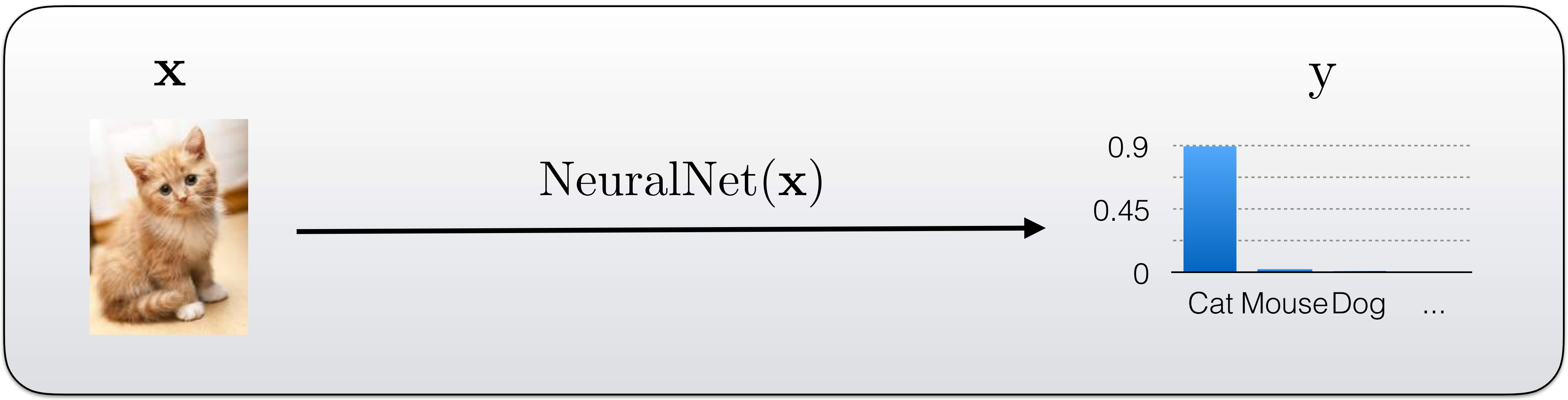
Concept 1:

Parameterization of conditional distributions with Neural Networks

Common example

$$\mathbf{p} = \mathrm{NeuralNet}(\mathbf{x})$$

$$p_{\boldsymbol{\theta}}(y|\mathbf{x}) = \mathrm{Categorical}(y; \mathbf{p})$$



Concept 2:

Generalization into Directed Models parameterized with Bayesian Networks

Directed graphical models / Bayesian networks

- Joint distribution factorizes as:

$$p_{\boldsymbol{\theta}}(\mathbf{x}_1, ..., \mathbf{x}_M) = \prod_{j=1}^{M} p_{\boldsymbol{\theta}}(\mathbf{x}_j | Pa(\mathbf{x}_j))$$

- We parameterize conditionals using neural networks:

$$\begin{aligned} \boldsymbol{\eta} &= \text{NeuralNet}(Pa(\mathbf{x})) \\ p_{\boldsymbol{\theta}}(\mathbf{x}|Pa(\mathbf{x})) &= p_{\boldsymbol{\theta}}(\mathbf{y}|\boldsymbol{\eta}) \end{aligned}$$

- Traditionally: parameterized using probability tables

Maximum Likelihood (ML)

- Log-probability of a datapoint x:

$$L_{\boldsymbol{\theta}}(\mathbf{x}) = \log p_{\boldsymbol{\theta}}(\mathbf{x})$$

- Log-likelihood of i.i.d. dataset:

$$L_{\boldsymbol{\theta}}(\mathcal{D}) = \frac{1}{N_{\mathcal{D}}} \sum_{\mathbf{x} \in \mathcal{D}} L_{\boldsymbol{\theta}}(\mathbf{x})$$

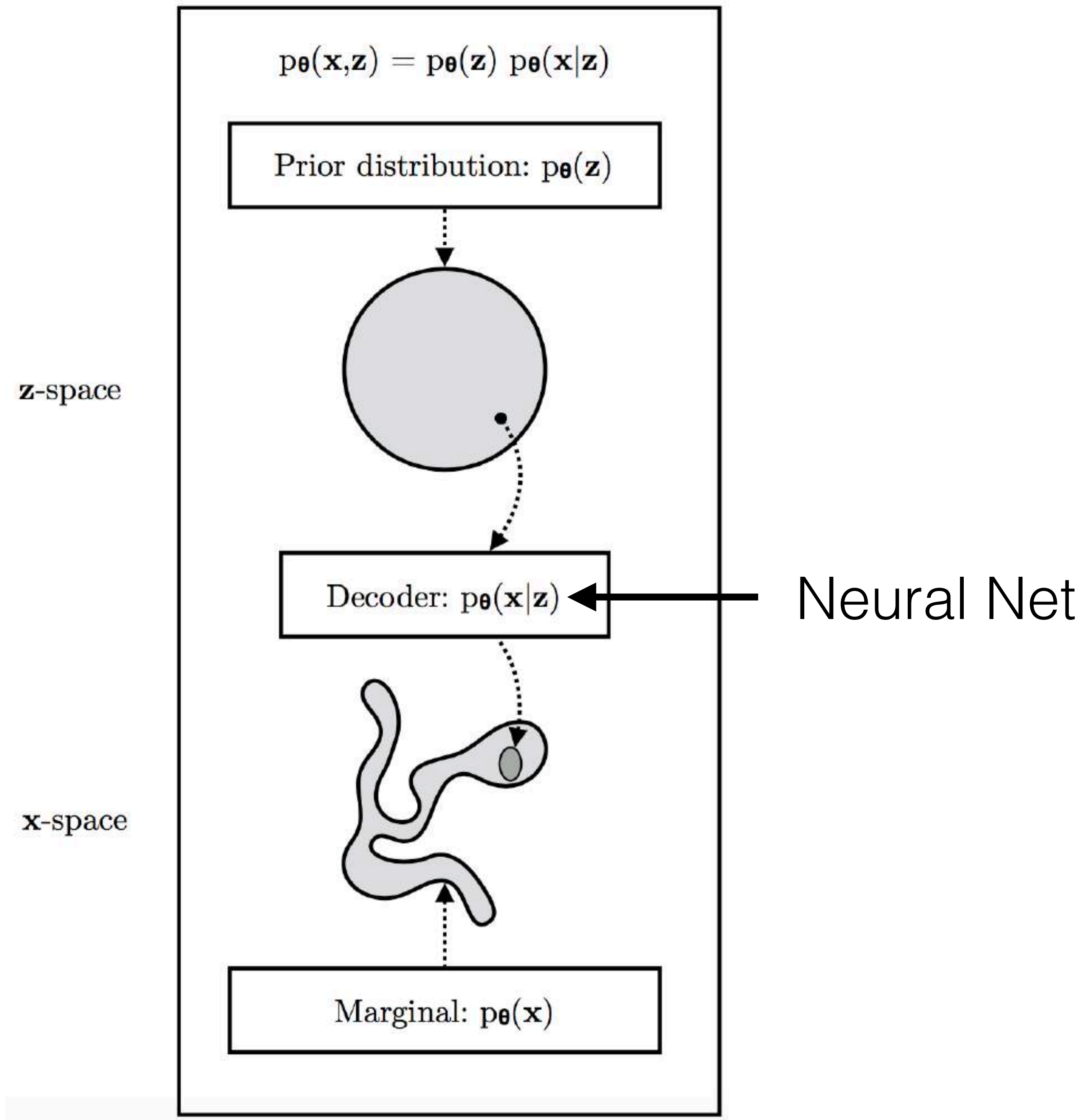
- Optimizable with (minibatch) SGD

Concept 3:

Generalization into Deep Latent-Variable Models

Deep Latent-Variable Model (DLVM)

- Introduction of latent variables in graph
- Latent-variable model $p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})$
  where conditionals are parameterized with neural networks
- Advantages:
  - Extremely flexible: even if each conditional is simple (e.g. conditional Gaussian), the marginal likelihood can be arbitrarily complex
- Disadvantage:
  - $p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\, d\mathbf{z}$ is intractable


$p_\theta(\mathbf{x},\mathbf{z}) = p_\theta(\mathbf{z})\ p_\theta(\mathbf{x}|\mathbf{z})$
Prior distribution: $p_\theta(\mathbf{z})$
$\mathbf{z}$-space
Decoder: $p_\theta(\mathbf{x}|\mathbf{z})$
Neural Net
$\mathbf{x}$-space
Marginal: $p_\theta(\mathbf{x})$

DLVM: Optimization is non-trivial

- By direct optimization of log p(x) ?
  - Intractable marg. likelihood
- With expectation maximization (EM)?
  - Intractable posterior: $p(z|x) = p(x,z)/p(x)$
- With MAP: point estimate of $p(z|x)$?
  - Overfits
- With trad. variational EM and MCMC-EM?
  - Slow
- And none tells us how to do fast posterior inference

Variational Autoencoders (VAEs)

Solution: Variational Autoencoder (VAE)

- Introduce q(z|x): <u>parametric model</u> of true posterior
  - Parameterized by another neural network
- Joint optimization of q(z|x) and p(x,z)
  - Remarkably simple objective: *evidence lower bound* (ELBO) [MacKay, 1992]

Encoder / Approximate Posterior

- $q_{\boldsymbol{\varphi}}(\mathbf{z}|\mathbf{x})$: *parametric model* of the posterior
  $\boldsymbol{\varphi}$: *variational parameters*

- We optimize the variational parameters $\boldsymbol{\varphi}$ such that:

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \approx p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$$

- Like a DLVM, the inference model can be (almost) any directed graphical model:

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = q_{\boldsymbol{\phi}}(\mathbf{x}_1, ..., \mathbf{x}_M|\mathbf{x}) = \prod_{j=1}^{M} q_{\boldsymbol{\phi}}(\mathbf{z}_j|Pa(\mathbf{z}_j), \mathbf{x})$$

- Note that traditionally, variational methods employ *local variational parameters*. We only have global $\boldsymbol{\varphi}$

Evidence Lower Bound / ELBO

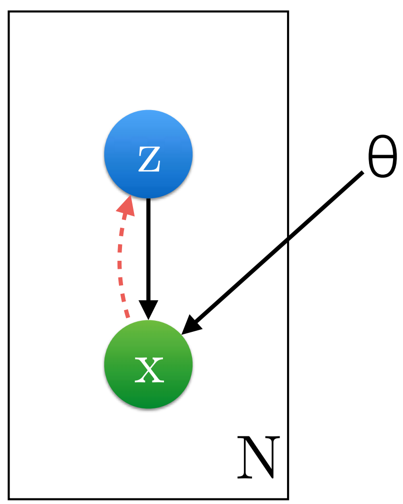
Objective (ELBO):

$$\mathcal{L}(x;\theta) = \mathbb{E}_{q(z|x)}\left[\log p(x,z) - \log q(z|x)\right]$$

Can be rewritten as:

$$\mathcal{L}(x;\theta) = \log p(x) - D_{KL}(q(z|x)||p(z|x))$$

Example



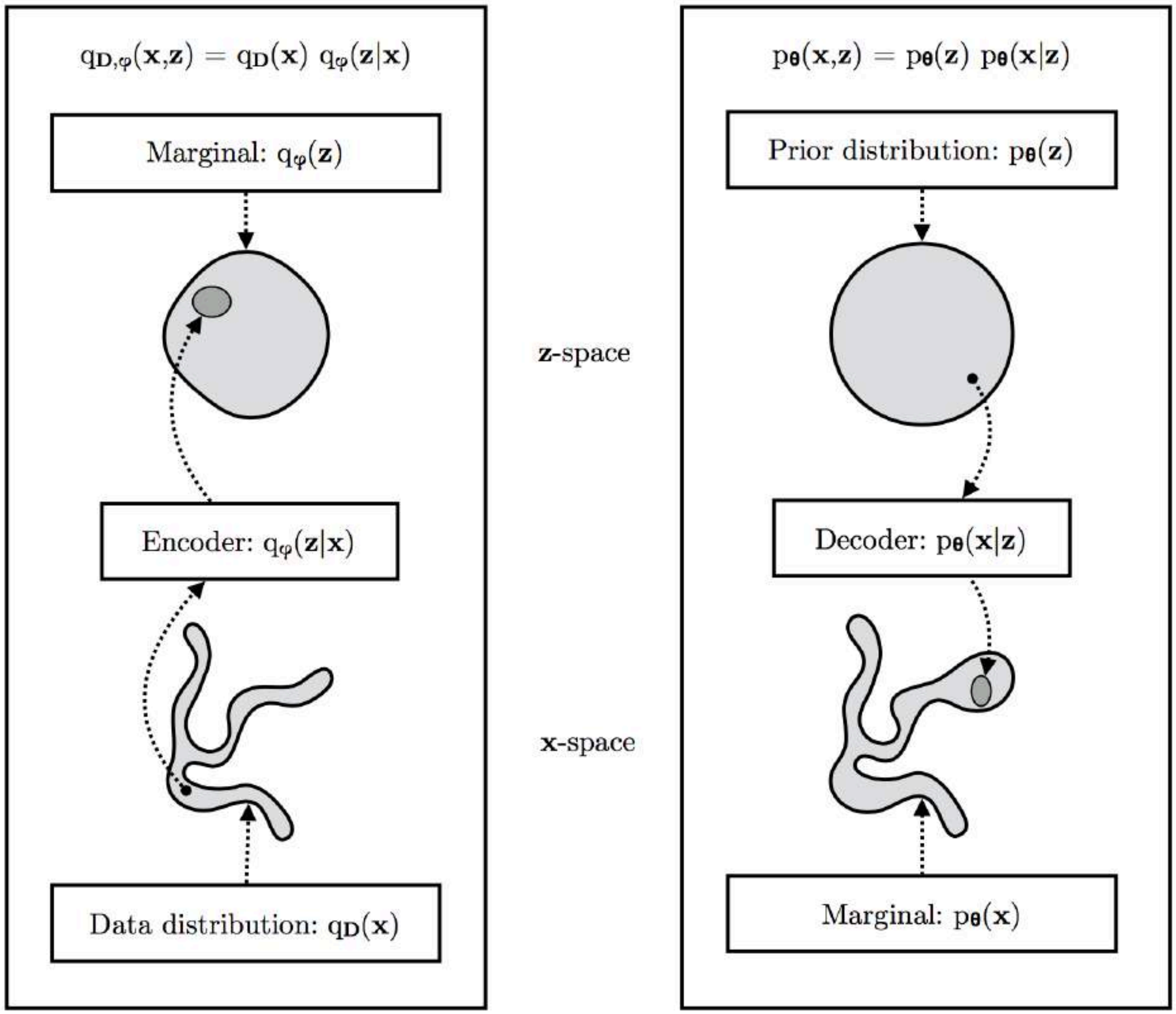
1. Maximization of log p(x)
   => Good marginal likelihood

2. Minimization of $D_{KL}(q(z|x)||p(z|x))$
   => Accurate (and fast) posterior inference

Stochastic Gradient Descent (SGD)

- Minibatch SGD: requires unbiased gradients estimates
- Reparameterization trick for continuous latent variables [Kingma and Welling, 2013]
- REINFORCE for discrete latent variables
- Adam optimizer adaptively pre-conditioned SGD [Kingma and Ba, 2014]
- Weight normalisation for faster convergence [Salimans and Kingma, 2015]

ELBO as KL Divergence



ML objective = - $D_{KL}(\ q_{\mathbf{D}}(\mathbf{x})\ ||\ p_{\boldsymbol{\theta}}(\mathbf{x})\ )$

ELBO objective = - $D_{KL}(\ q_{\mathbf{D}}(\mathbf{x},\mathbf{z})\ ||\ p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})\ )$

Gradients

- An unbiased gradient estimator of the ELBO w.r.t. the generative model parameters is straightforwardly obtained:

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta},\phi}(\mathbf{x}) &= \nabla_{\boldsymbol{\theta}} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}) \right] \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\nabla_{\boldsymbol{\theta}} (\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})) \right] \\
&\simeq \nabla_{\boldsymbol{\theta}} (\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}))
\end{aligned}
$$

- A gradient estimator of the ELBO w.r.t. the variational parameters φ is more difficult to obtain:

$$
\begin{aligned}
\nabla_\phi \mathcal{L}_{\boldsymbol{\theta},\phi}(\mathbf{x}) &= \nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}) \right] \\
&\neq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\nabla_\phi (\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})) \right]
\end{aligned}
$$

Reparameterization Trick

- Construct the following Monte Carlo estimator:

$$\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$$
$$\mathbf{z} = g(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})$$
$$\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}; \boldsymbol{\epsilon}) = \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$$

- where p($\boldsymbol{\varepsilon}$) and g() chosen such that $\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$

- Which has a simple Monte Carlo gradient:

$$\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}; \boldsymbol{\epsilon})$$

Reparameterization Trick

- This is an unbiased estimator of the exact single-datapoint ELBO gradient:

$$\begin{aligned}
\mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x};\boldsymbol{\epsilon})\right] &= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}(\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}))\right] \\
&= \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right]) \\
&= \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x})
\end{aligned}$$

Reparameterization Trick

- Under reparameterization, density is given by:

$$\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \log p(\boldsymbol{\epsilon}) - \log |\det(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}})|$$

- Important: choose transformations g() for which the logdet is computationally affordable/simple

Factorized Gaussian Posterior

- A common choice is a simple factorized Gaussian encoder:

$$
\begin{aligned}
(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) &= \mathrm{EncoderNeuralNet}_{\phi}(\mathbf{x}) \\
q_{\phi}(\mathbf{z}|\mathbf{x}) &= \prod_i q_{\phi}(z_i|\mathbf{x}) = \prod_i \mathcal{N}(z_i; \mu_i, \sigma_i^2)
\end{aligned}
$$

- After reparameterization, we can write:

$$
\begin{aligned}
\boldsymbol{\epsilon} &\sim \mathcal{N}(0, \mathbf{I}) \\
(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) &= \mathrm{EncoderNeuralNet}_{\phi}(\mathbf{x}) \\
\mathbf{z} &= \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}
\end{aligned}
$$

Factorized Gaussian Posterior

- The Jacobian of the transformation is:

$$\frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}} = \mathrm{diag}(\boldsymbol{\sigma})$$

- Determinant of diagonal matrix is product of diag. entries.

- So the posterior density is:

$$\begin{aligned}\log q_\phi(\mathbf{z}|\mathbf{x}) &= \log p(\boldsymbol{\epsilon}) - \log|\det(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}})| \\ &= \sum_i \log \mathcal{N}(\epsilon_i; 0, 1) - \log \sigma_i\end{aligned}$$

Full-covariance Gaussian posterior

- The factorized Gaussian posterior can be extended to a Gaussian with full covariance:

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$$

- A reparameterization of this distribution with a surprisingly simple determinant, is:

$$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$$
$$\mathbf{z} = \boldsymbol{\mu} + \mathbf{L}\boldsymbol{\epsilon}$$

- where $\mathbf{L}$ is a lower (or upper) triangular matrix, with non-zero entries on the diagonal. The off-diagonal element define the correlations (covariance) of the elements in $\mathbf{z}$.

Full-covariance Gaussian posterior

- This reason for this parameterization of the full-covariance Gaussian, is that the Jacobian determinant is remarkably simple. The Jacobian is trivial:

$$\frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}} = \mathbf{L}$$

- And the determinant of a triangular matrix is simply the product of its diagonal terms. So:

$$\log |\det(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}})| = \sum_i \log |L_{ii}|$$

Full-covariance Gaussian posterior

- This parameterization corresponds to the Cholesky decomposition of the covariance of $\mathbf{z}$:

$$
\begin{aligned}
\boldsymbol{\Sigma} &= \mathbb{E}\left[(\mathbf{z} - \mathbb{E}\left[\mathbf{z}||\right])(\mathbf{z} - \mathbb{E}\left[\mathbf{z}||\right])^T\right] \\
&= \mathbb{E}\left[\mathbf{L}\boldsymbol{\epsilon}(\mathbf{L}\boldsymbol{\epsilon})^T\right] \\
&= \mathbb{E}\left[\mathbf{L}\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T\mathbf{L}^T\right] \\
&= \mathbf{L}\mathbb{E}\left[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T\right]\mathbf{L}^T \\
&= \mathbf{L}\mathbf{L}^T
\end{aligned}
$$

Full-covariance Gaussian posterior

- One way to construct the matrix $\mathbf{L}$ is as follows:

$$\begin{aligned}(\boldsymbol{\mu}, \log \boldsymbol{\sigma}, \mathbf{L}') &\leftarrow \text{EncoderNeuralNet}_{\phi}(\mathbf{x}) \\ \mathbf{L} &\leftarrow \mathbf{L}_{mask} \odot \mathbf{L}' + \text{diag}(\boldsymbol{\sigma})\end{aligned}$$

- $\mathbf{L}_{mask}$ is a masking matrix.

- The log-determinant is identical to the factorized Gaussian case:

$$\log |\det(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}})| = \sum_i \log \sigma_i$$

Full-covariance Gaussian posterior

- Therefore, density equal to diagonal Gaussian case!

$$
\begin{aligned}
\log q_\phi(\mathbf{z}|\mathbf{x}) &= \log p(\boldsymbol{\epsilon}) - \log|\det(\frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}})| \\
&= \sum_i \log \mathcal{N}(\epsilon_i; 0, 1) - \log \sigma_i
\end{aligned}
$$

Beyond Gaussian posteriors

Normalizing Flows

- Full-covariance Gaussian:
  - One transformation operation: $\mathbf{f}_t(\boldsymbol{\varepsilon}, \mathbf{x}) = \mathrm{L}\boldsymbol{\varepsilon}$
- Normalizing flows:
  - Multiple transformation steps

Normalizing Flows

- Define $\mathbf{z} \sim q_{\phi}(\mathbf{z}|\mathbf{x})$ as:

$$\begin{aligned}
&\boldsymbol{\epsilon}_0 \sim p(\boldsymbol{\epsilon}) \\
&\text{for t} = 1 \ldots \text{T:} \\
&\quad \boldsymbol{\epsilon}_t = \mathbf{f}_t(\boldsymbol{\epsilon}_{t-1}, \mathbf{x}) \\
&\mathbf{z} = \boldsymbol{\epsilon}_T
\end{aligned}$$

- The Jacobian of the transformation factorizes:

$$\frac{d\mathbf{z}}{d\boldsymbol{\epsilon}_0} = \prod_{t=1}^{T} \frac{d\boldsymbol{\epsilon}_t}{d\boldsymbol{\epsilon}_{t-1}}$$

- And the density

$$\log q_{\phi}(\mathbf{z}|\mathbf{x}) = \log p(\boldsymbol{\epsilon}_0) - \sum_{t=1}^{T} \log \det \left| \frac{d\boldsymbol{\epsilon}_t}{d\boldsymbol{\epsilon}_{t-1}} \right|$$

[Rezende and Mohamed, 2015]

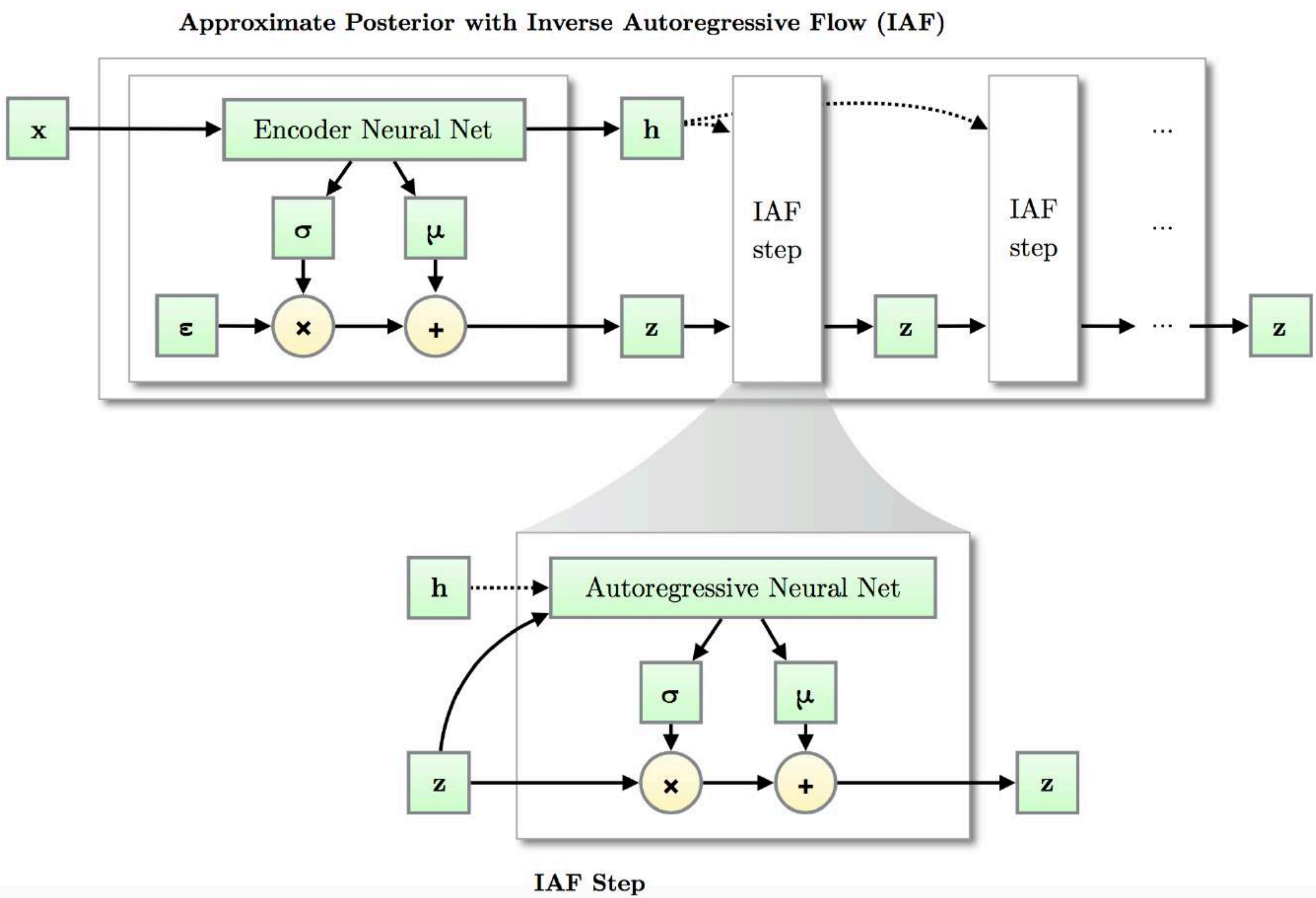
Inverse Autoregressive Flows

- Probably the most flexible type of transformation, with simple determinant, that can be chained.

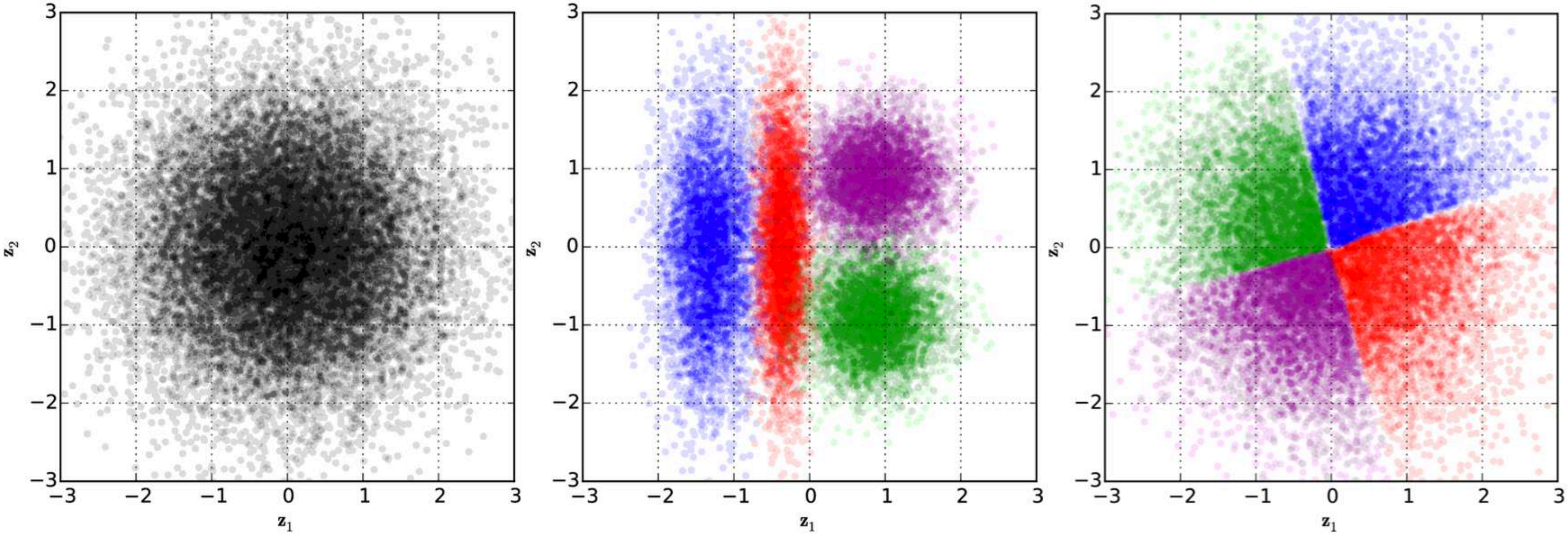
- Each transformation given by a autoregressive neural net, with triangular Jacobian

- Best known way to construct arbitrarily flexible posteriors

Inverse Autoregressive Flow



Posteriors in 2D space



(a) Prior distribution (b) Posteriors in standard VAE (c) Posteriors in VAE with IAF

Deep IAF helps towards better likelihoods

Table 1: Generative modeling results on the dynamically sampled binarized MNIST version used in previous publications (Burda et al., 2015). Shown are averages; the number between brackets are standard deviations across 5 optimization runs. The right column shows an importance sampled estimate of the marginal likelihood for each model with 128 samples. Best previous results are reproduced in the first segment: [1]: (Salimans et al., 2014) [2]: (Burda et al., 2015) [3]: (Kaae Sønderby et al., 2016) [4]: (Tran et al., 2015)

Model	VLB	$\log p(\mathbf{x}) \approx$
Convolutional VAE + HVI [1]	-83.49	-81.94
DLGM 2hl + IWAE [2]		-82.90
LVAE [3]		-81.74
DRAW + VGP [4]	-79.88	
Diagonal covariance	-84.08 ($\pm$ 0.10)	-81.08 ($\pm$ 0.08)
IAF (Depth $= 2$, Width $= 320$)	-82.02 ($\pm$ 0.08)	-79.77 ($\pm$ 0.06)
IAF (Depth $= 2$, Width $= 1920$)	-81.17 ($\pm$ 0.08)	-79.30 ($\pm$ 0.08)
IAF (Depth $= 4$, Width $= 1920$)	-80.93 ($\pm$ 0.09)	-79.17 ($\pm$ 0.08)
IAF (Depth $= 8$, Width $= 1920$)	-80.80 ($\pm$ 0.07)	**-79.10** ($\pm$ 0.07)

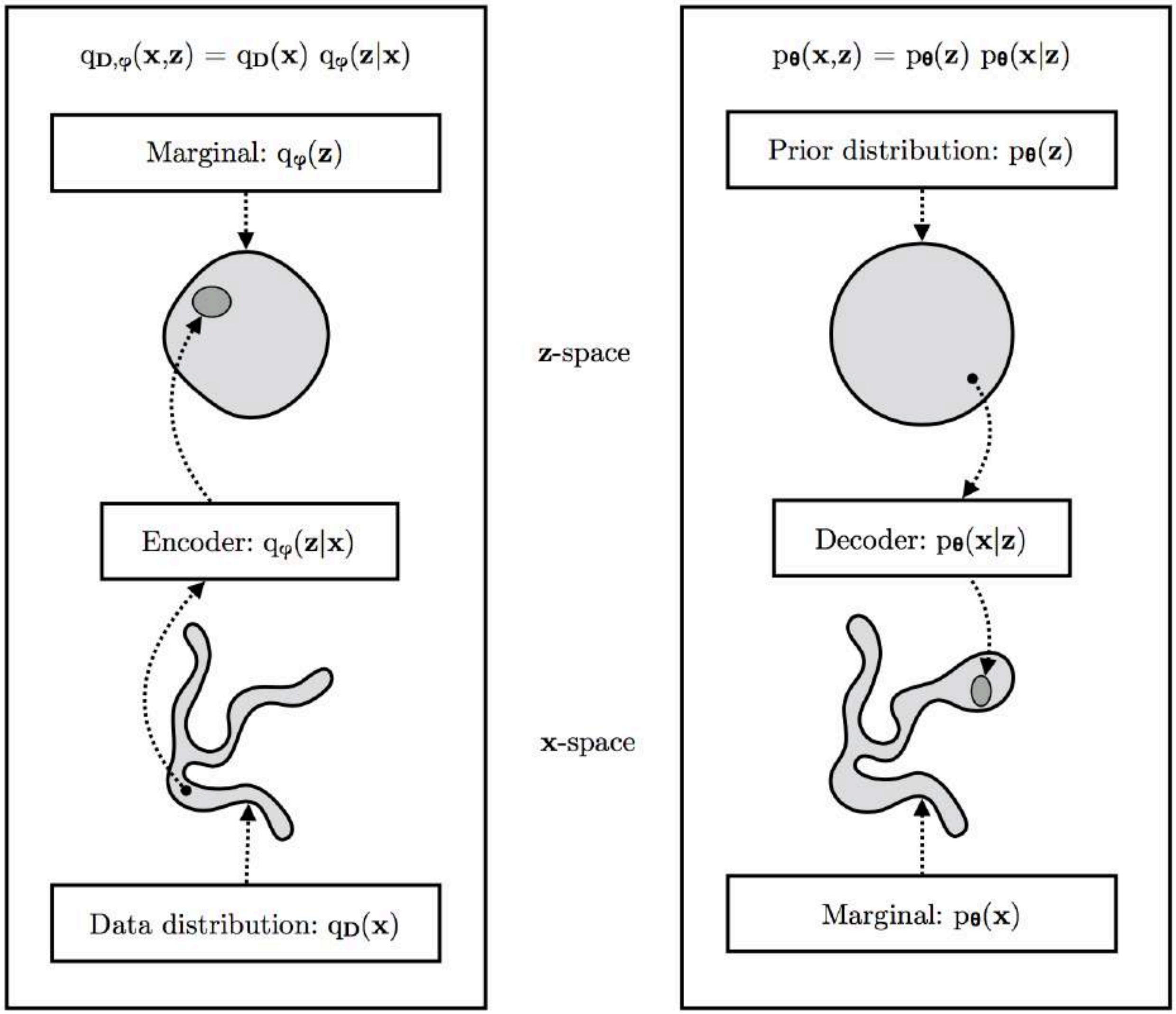
[Kingma, Salimans and Welling, 2014]

Optimization Issues

- Overpruning:
  - Solution 1: KL annealing
  - Solution 2: Free bits (see IAF paper)
- ‘Blurriness’ of samples
  - Solution: better Q or P models

Better generative models

Improving Q versus improving P



ML objective = - $D_{KL}(\ q_{\mathbf{D}}(\mathbf{x}) \ ||\ p_{\boldsymbol\theta}(\mathbf{x})\ )$

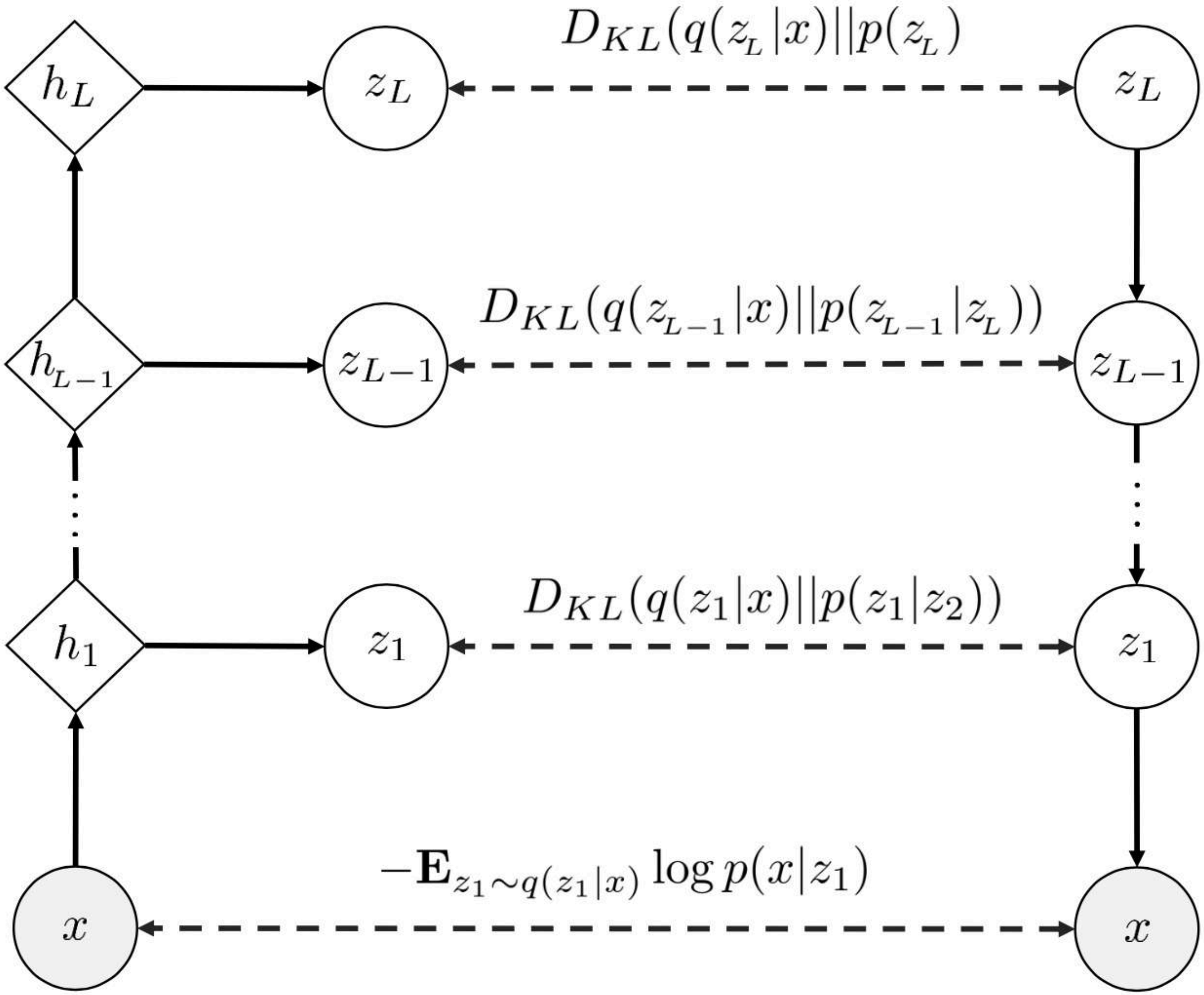
ELBO objective = - $D_{KL}(\ q_{\mathbf{D}}(\mathbf{x},\mathbf{z}) \ ||\ p_{\boldsymbol\theta}(\mathbf{x},\mathbf{z})\ )$

PixelVAE

- Use PixelCNN models as $p(x|z)$ and $p(z)$ models
- No need for complicated $q(z|x)$: just factorized Gaussian

[Gulrajani et al, 2016]

PixelVAE



[Gulrajani et al, 2016]

PixelVAE

PixelVAE

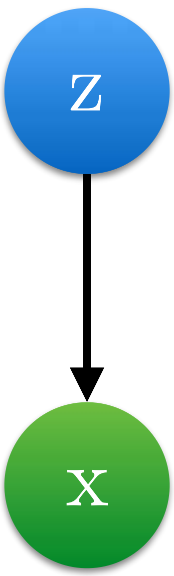
Figure 6: Samples from hierarchical PixelVAE on the 64x64 ImageNet dataset.

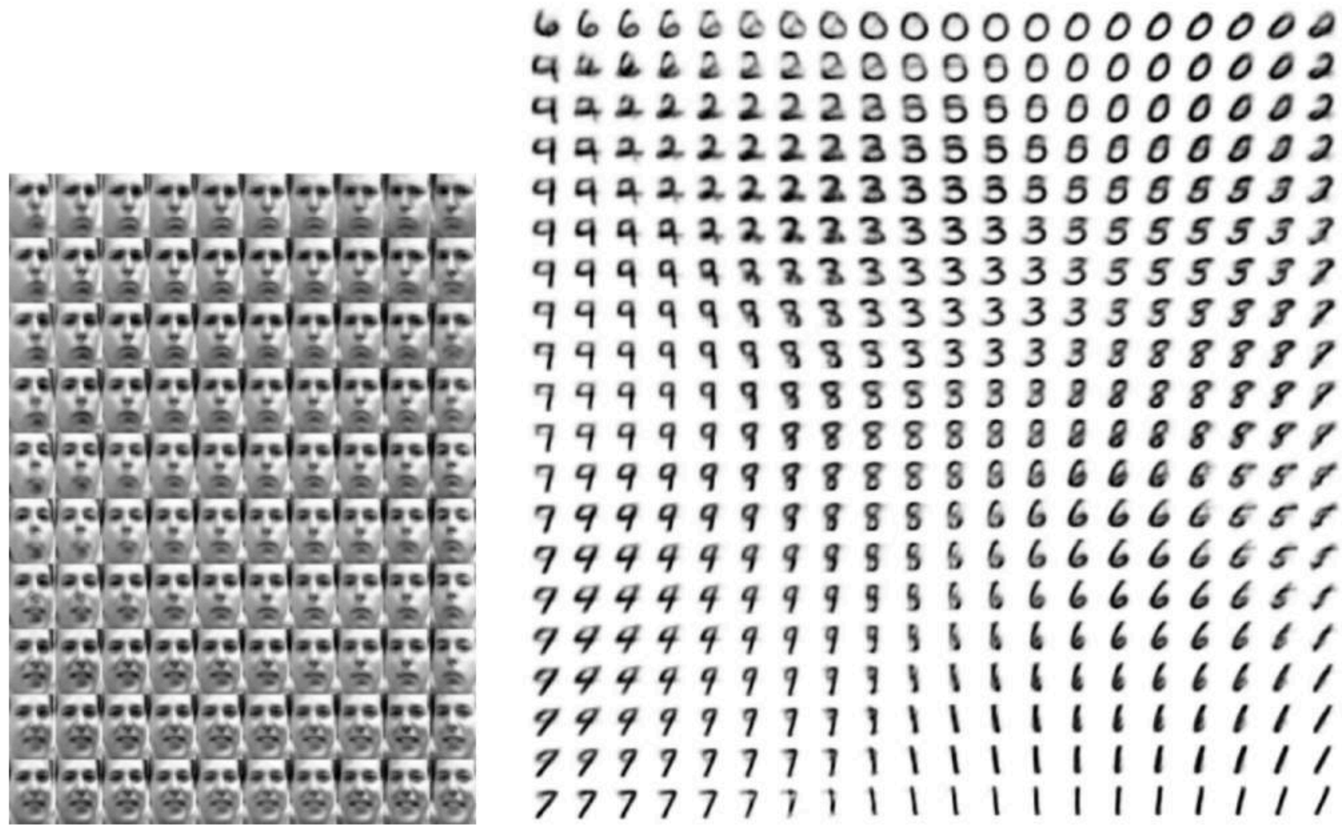
Applications

Visualisation of Data in 2D

Representation learning

2D





(a) Learned Frey Face manifold (b) Learned MNIST manifold

Semi-supervised learning

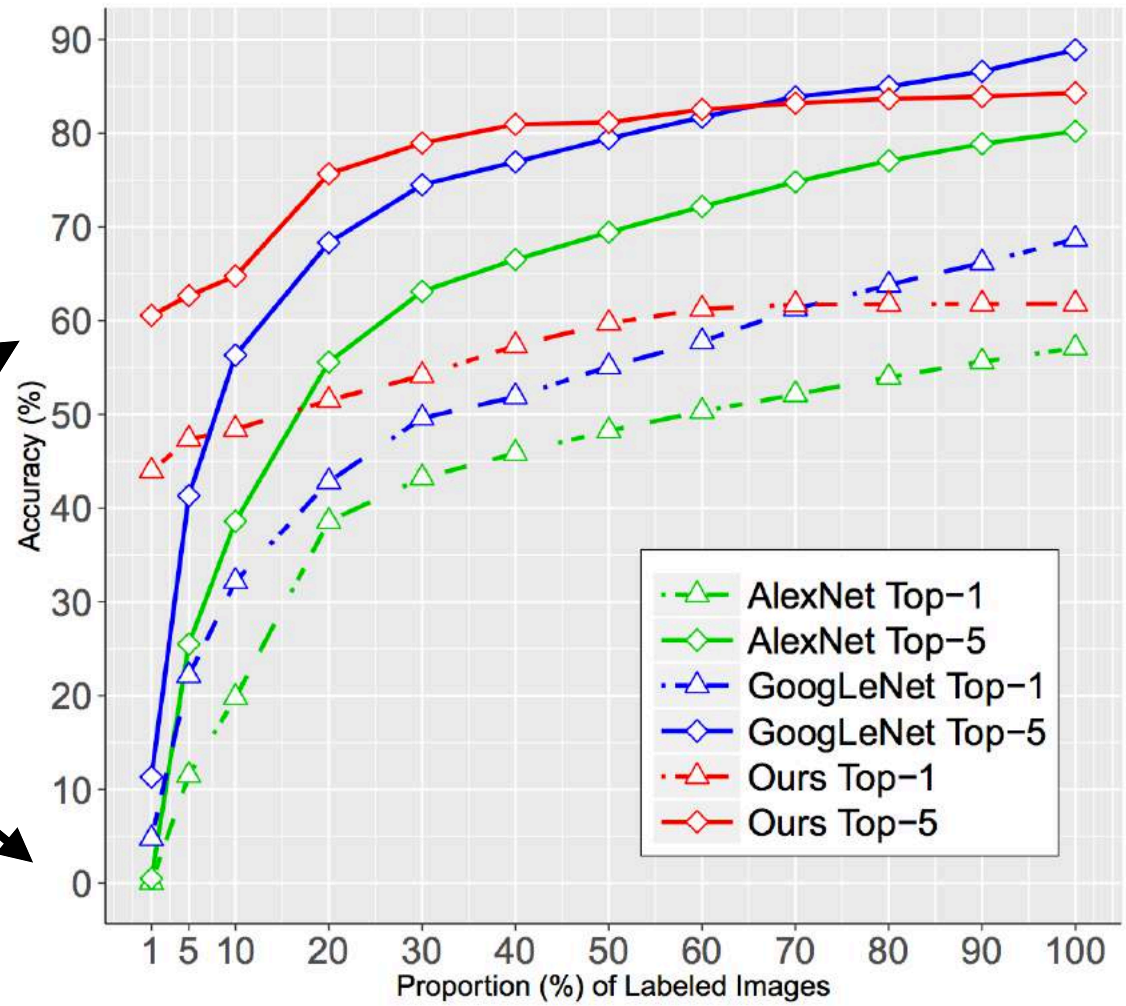
SSL With Auxiliary VAE

	MNIST 100 LABELS	SVHN 1000 LABELS	NORB 1000 LABELS
M1+TSVM (KINGMA ET AL., 2014)	11.82% (±0.25)	55.33% (±0.11)	18.79% (±0.05)
M1+M2 (KINGMA ET AL., 2014)	3.33% (±0.14)	36.02%(±0.10)	-
VAT (MIYATO ET AL., 2015)	2.12%	24.63%	9.88%
LADDER NETWORK (RASMUS ET AL., 2015)	1.06% (±0.37)	-	-
AUXILIARY DEEP GENERATIVE MODEL (ADGM)	**0.96**% (±0.02)	22.86%	10.06% (±0.05)
SKIP DEEP GENERATIVE MODEL (SDGM)	1.32% (±0.07)	**16.61**% (±0.24)	**9.40**% (±0.04)

[Maaløe et al, 2016]

Data-efficient learning on ImageNet

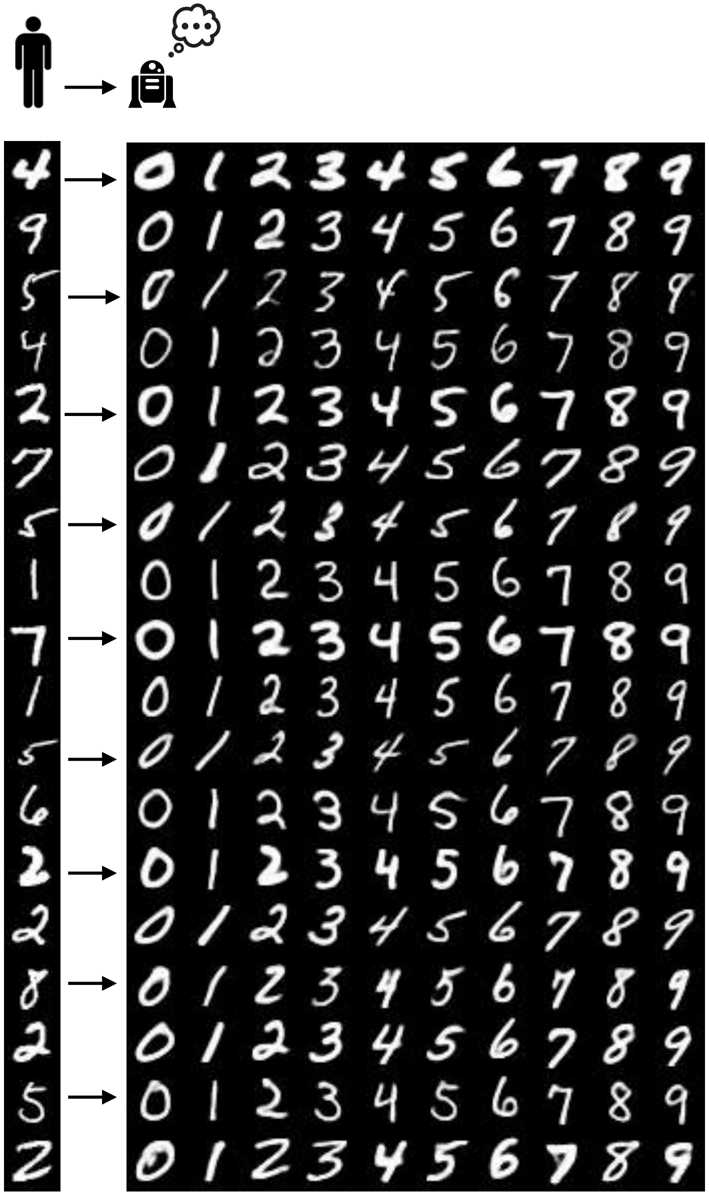
from 10% to 60% accuracy, for 1% labeled

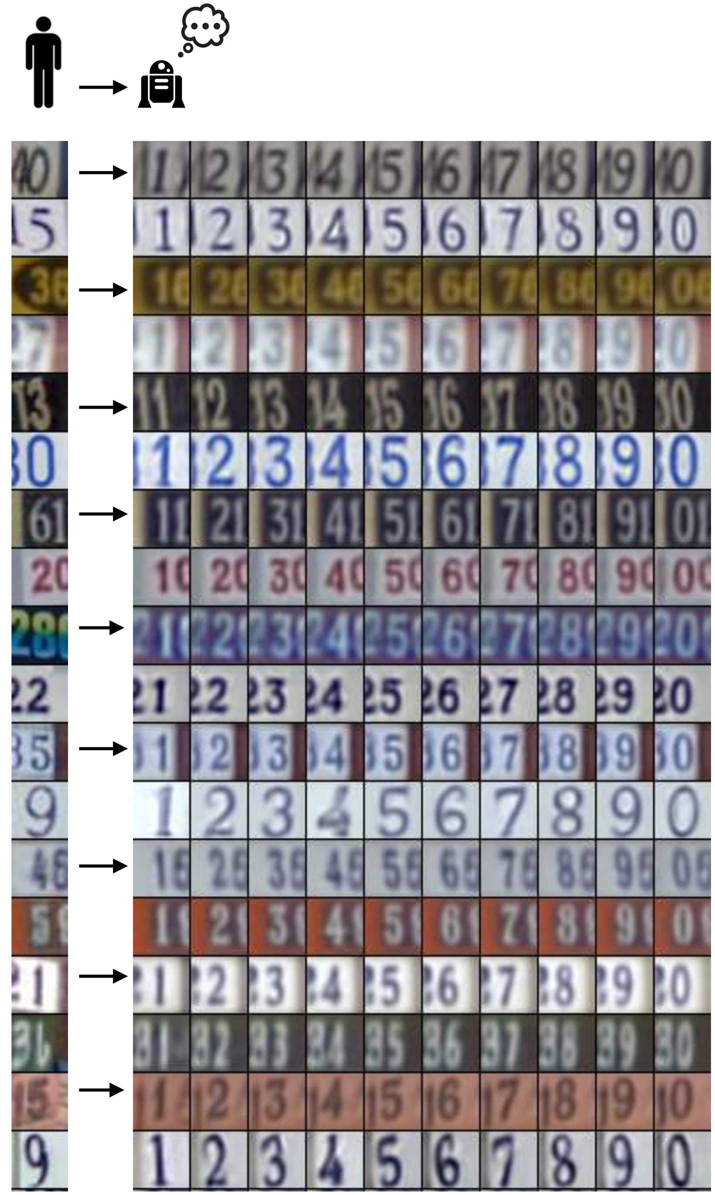


[Pu et al, "Variational Autoencoder for Deep Learning of Images, Labels and Captions", 2016]

(Re)Synthesis

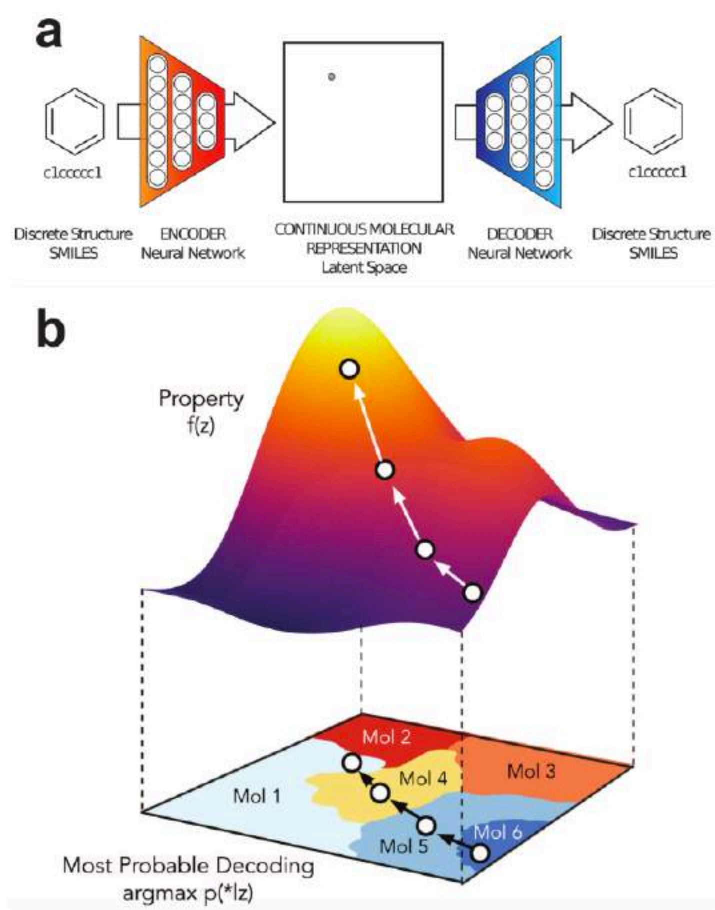
Analogy-making





Automatic chemical design

- VAE trained on text representation of 250K molecules
- Uses latent space to design new drugs and organic LEDs



[Gómez-Bombarelli et al, 2016]

Semantic Editing

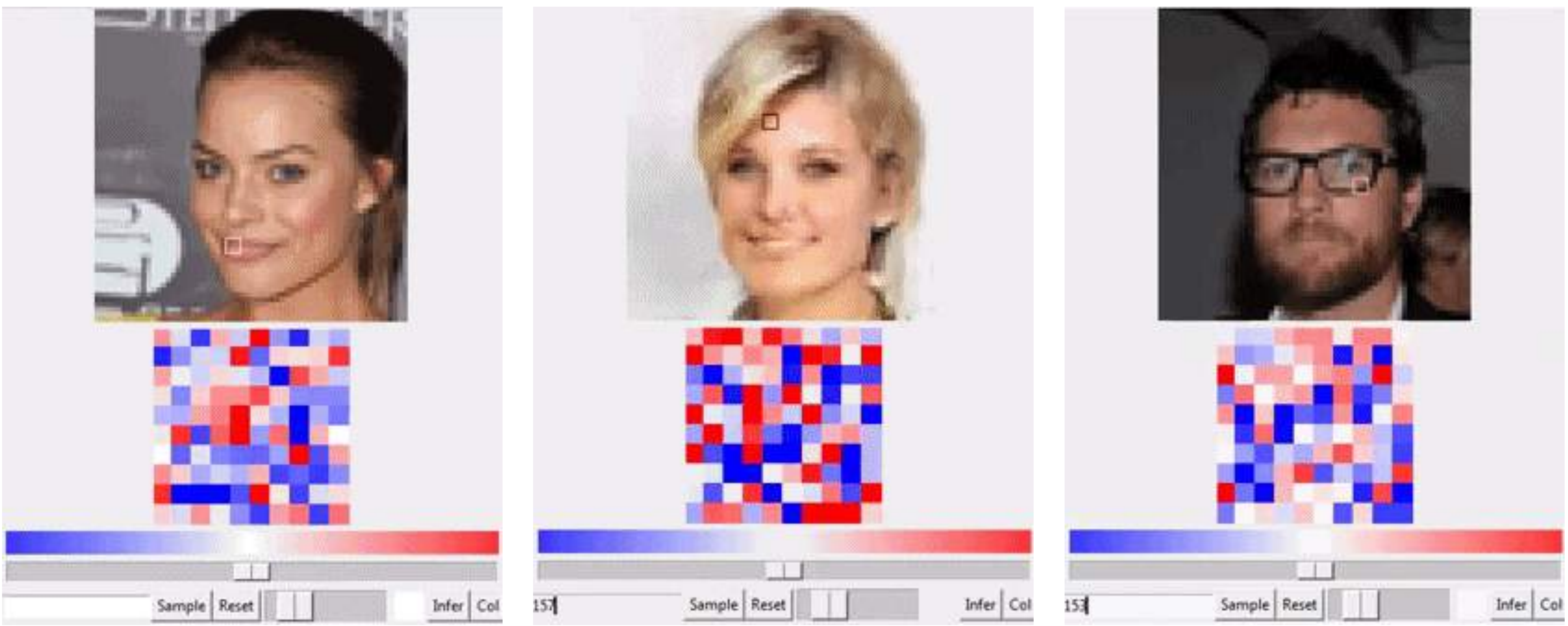
- "Smile vector". Tom White, 2016, twitter: @dribnet

Semantic Editing

- "Smile vector". Tom White, 2016, twitter: @dribnet

Semantic Editing

- "Neural Photo Editing". Andrew Brock et al, 2016



Questions?