Variational Lossy Autoencoder

Chen et al.

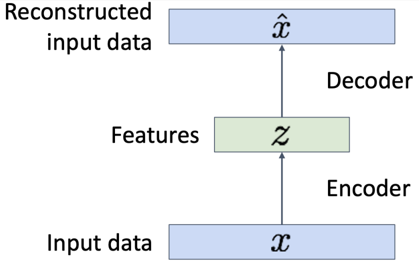
Dylan Green

March 17, 2020

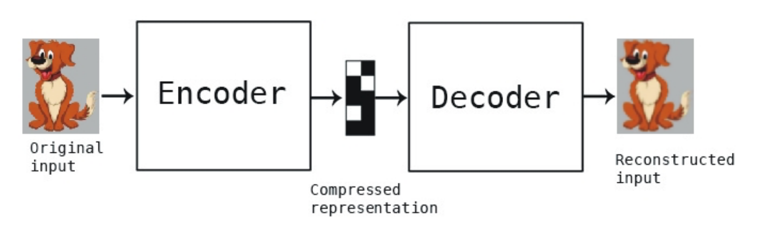
Autoencoders

Autoencoders

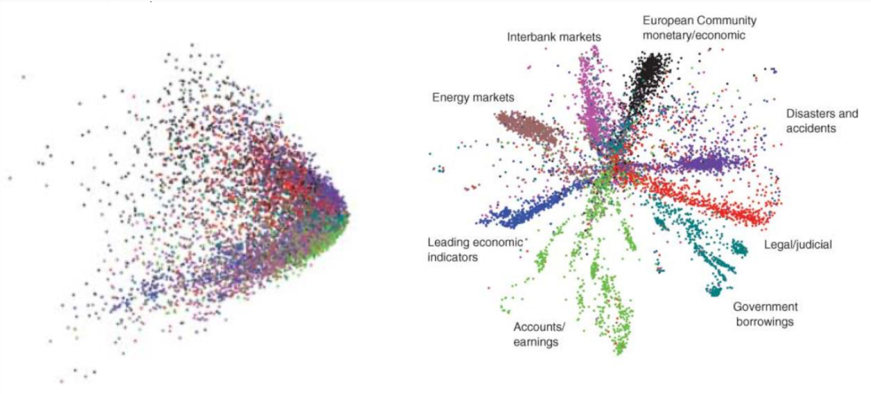
- Unsupervised deep learning model
- Loss: $\|x - \hat{x}\|^2$
- $\dim(z) \ll \dim(x)$
- Features should extract useful, high-level information from the input data



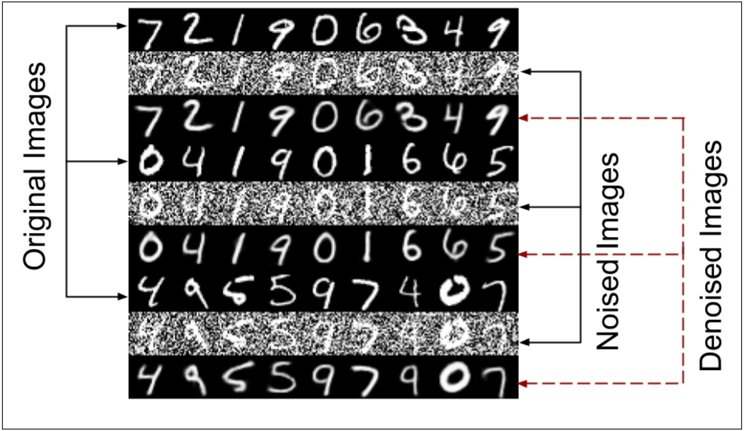
Applications - Data Compression



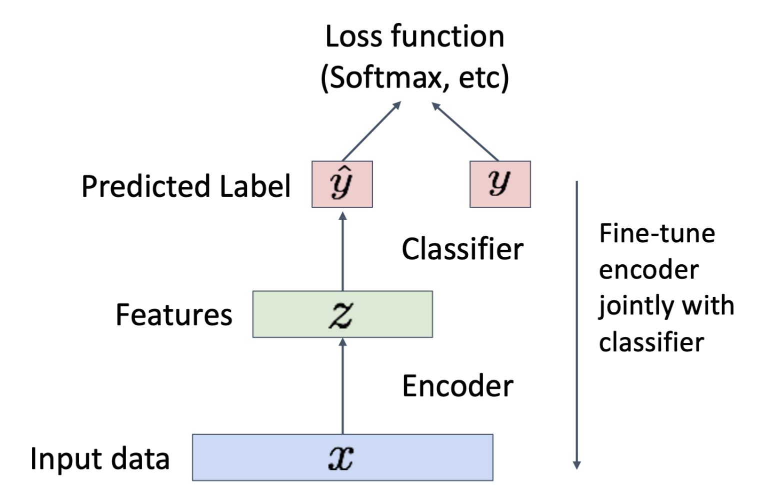
Applications - Visualization



Applications - Denoising Autoencoders



Applications - Unsupervised Feature Learning



Autoencoders as "Generative" Models?

- What if we want to generate new data using this model?
  - Pick a random z, use decoder to generate new image
- **Problem:**
  - Model maps each x to a point in z-space
  - How to pick a "good" z?

Variational Autoencoders (VAEs)

VAEs

- A probabilistic spin on autoencoders
    - Learn latent variables z from input data
    - Sample from the model to generate new data
- Intuition: x is an image, z encodes high-level information about the image (i.e. attributes, orientation, etc.)

VAEs

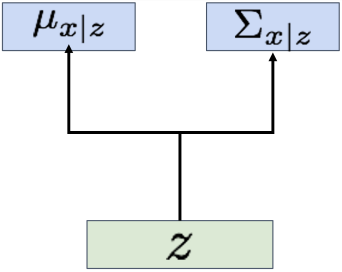
- Assume a generative model with a latent variable z distributed according to some prior distribution $p(z)$
- The observed variable x is then distributed according to a conditional likelihood $p_\theta(x|z)$
- Sample in two steps:
  - $z \sim p(z)$
  - $x \sim p_\theta(x|z)$
- Marginal likelihood of the data under this model is then

$$p_\theta(x) = \int p_\theta(x, z)dz = \int p_\theta(x|z)p(z)dz$$

VAEs

For the standard VAE:

- Choose $p(z) = \mathcal{N}(z|0, I)$
- Represent $p_\theta(x|z)$ with a neural network:
  - Can be thought of as a stochastic decoder network
  - Input: z, Outputs: mean $\mu_{x|z}$ and diagonal covariance $\Sigma_{x|z}$



VAEs - Training

- Objective: maximize marginal likelihood of training data:

$$\max_{\theta} \sum_{i} \log p_{\theta}(x^{(i)})$$

where

$$p_{\theta}(x^{(i)}) = \int p_{\theta}(x^{(i)}|z)p(z)dz$$

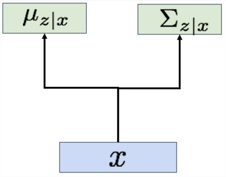
- **Problem:** This integral is intractable

VAEs - Training

- Potential fix: try Bayes' rule:

$$p_\theta(x^{(i)}) = \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}$$

- **Another problem:** $p_\theta(z \mid x^{(i)})$ is also intractable
- **Solution:** Introduce (stochastic) encoder network $q_\phi(z \mid x^{(i)})$
  - $q_\phi(z \mid x^{(i)}) \approx p_\theta(z \mid x^{(i)})$
  - Input: x, Outputs: mean $\mu_{z|x}$ and diagonal covariance $\Sigma_{z|x}$



- Jointly train q_ϕ, p_θ

VAEs - ELBO

Plug this in to marginal likelihood

$$
\begin{aligned}
\log p_\theta(x) &= \log \int p_\theta(x, z) dz \\
&= \log \int q_\phi(z|x) \frac{p_\theta(z, x)}{q_\phi(z|x)} dz \\
&\geq \mathbb{E}_{q_\phi(z|x)} \left[\log \frac{p_\theta(x, z)}{q_\phi(z|x)} \right] \quad \text{(Jensen's Inequality)} \\
&\triangleq \mathcal{L}(\theta, \phi)
\end{aligned}
$$

$\mathcal{L}(\theta, \phi)$ is the log **E**vidence **L**ower **BO**und, or ELBO

VAEs - ELBO

Rearranging:

$$\log p_\theta(x) \geq \underbrace{\underbrace{\left(\mathbb{E}_{z\sim q_\phi(z|x)} \log p_\theta(x|z)\right)}_{\text{Reconstruction Loss}} - \underbrace{KL\left(q_\phi(z|x)||p(z)\right)}_{\text{Regularization}}}_{\mathcal{L}(\theta,\phi)-\text{VAE Objective}}$$

VAEs - ELBO

Another derivation:

$$
\begin{aligned}
D_{\mathrm{KL}}\left[q_x(z) \parallel p(z|x)\right] &= \mathbb{E}_{z\sim q_x(z)}\left[\log q_x(z) - \log p(z|x)\right] \\
&= \mathbb{E}_{z\sim q_x(z)}\left[\log q_x(z) - \log \frac{p(z,x)}{p(x)}\right] \\
&= \mathbb{E}_{z\sim q_x(z)}\left[\log q_x(z) - \log p(z) - \log p(x|z) + \log p(x)\right] \\
&= \underbrace{\mathbb{E}_{z\sim q_x(z)}\left[\log q_x(z) - \log p(z) - \log p(x|z)\right]}_{\text{Only this part depends on } z} + \log p(x)
\end{aligned}
$$

Rearranging gives us:

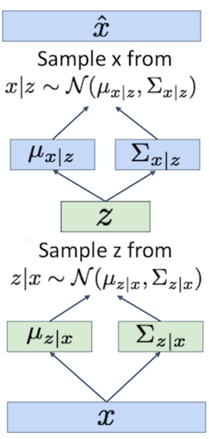
$$
\begin{aligned}
\log p(x) &= -\mathbb{E}_{z\sim q_x(z)}\left[\log q_x(z) - \log p(z) - \log p(x|z)\right] + D_{\mathrm{KL}}\left[q_x(z) \parallel p(z|x)\right] \\
&= \underbrace{\mathbb{E}_{z\sim q_x(z)}\left[\log p(z) + \log p(x|z) - \log q_x(z)\right]}_{\text{Variational Lower Bound}} + \underbrace{D_{\mathrm{KL}}\left[q_x(z) \parallel p(z|x)\right]}_{\geq 0}
\end{aligned}
$$

Takeaway: $\mathcal{L}(\theta, \phi)$ becomes exact if $q_\phi(z|x) = p_\theta(z|x)$

VAEs - Training

Train by maximizing

$$\mathcal{L}(\theta, \phi) = \left(\mathbb{E}_{z \sim q_\phi(z|x)} \log p_\theta(x|z)\right) - KL\left(q_\phi(z|x) || p(z)\right)$$

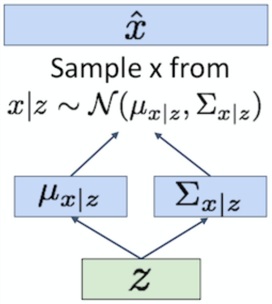


1. Run input through encoder to get $q_\phi(z|x)$
2. Sample z from $q_\phi(z|x)$ using "reparameterization" trick:
   - $\epsilon \sim \mathcal{N}(0, I)$
   - $z = \mu_{z|x} + \epsilon \odot \Sigma_{z|x}$
3. Run sampled z through decoder to get $p_\theta(x|z)$
4. Loss can be computed in closed form

VAEs - Sampling

To sample from the model:

1. Sample $z \sim p(z)$
2. Run sampled z through decoder to get $p_\theta(x|z)$
3. Sample $x \sim p_\theta(x|z)$ to generate new data

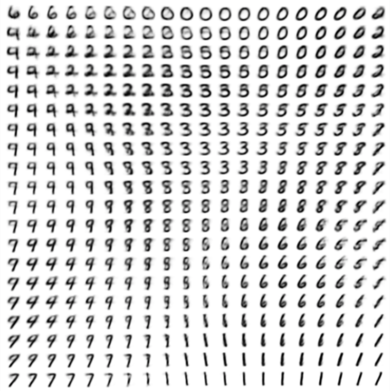


VAEs - Samples

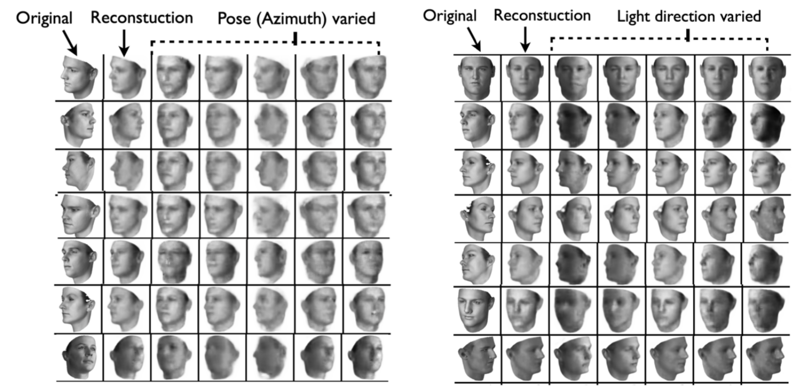
32x32 CIFAR-10

Labeled Faces in the Wild

VAEs - Moving Through Latent Space



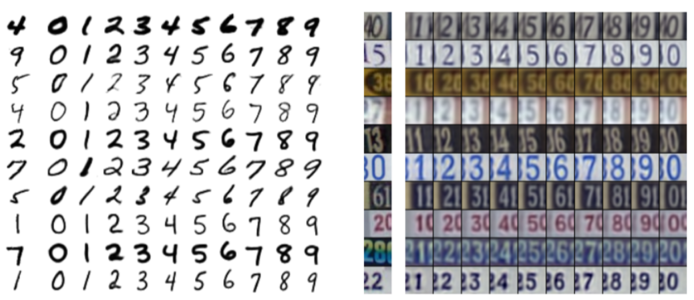
VAEs - "Image Editing"



VAEs - Class Conditional VAEs

Hold y fixed, vary z

VAEs - Class Conditional VAEs



Hold z fixed, vary y

Variational Lossy Autoencoder (VLAE)

How to improve on VAEs?

- Reconstructed images are often blurry
- Simple decoder distribution $p_\theta(x|z)$ lacks expressivity
    - Due to diagonal covariance $\Sigma_{x|z}$, all pixels are generated independently from one another
    - All entropy in the data must be explained by z
    - Not just content and style, but local features like texture
- Idea: use a decoder capable of modelling local correlations

Autoregressive Models

- Define some ordering over pixels
- Chain rule of probability

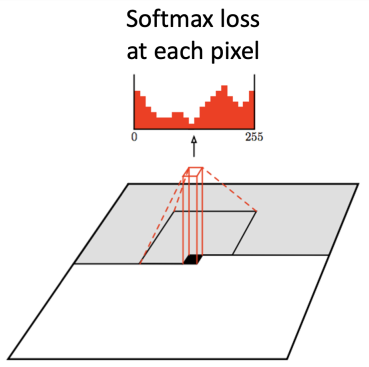
$$\begin{aligned} p(x) &= p(x_1, x_2, \ldots, x_d) \\ &= p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)\ldots \\ &= \prod_{i=1}^{d} p\left(x_i|\mathrm{x}_{1:i-1}\right) \end{aligned}$$

- Model $p\left(x_i|\mathrm{x}_{1:i-1}\right)$ with a neural network p_θ and maximize log likelihood

$$\log p_\theta(\mathrm{x}) = \sum_{i=1}^{d} \log p_\theta\left(x_i|\mathrm{x}_{1:i-1}\right)$$

PixelCNN

- Dependency on previous pixels modelled by a (masked) CNN
- Training for each location can be done in parallel
- Sampling must be done sequentially
- Powerful generative models in their own right



Powerful Decoders

What happens if we use a powerful decoder like this?

- Good news: great for generative modelling
- Bad news: the model completely ignores the latent code

Powerful Decoders

First recall that the goal of designing an efficient coding protocol is to minimize the expected code length of communicating $\mathbf{x}$. To explain Bits-Back Coding, let's first consider a more naive coding scheme. VAE can be seen as a way to encode data in a two-part code: $p(\mathbf{z})$ and $p(\mathbf{x}|\mathbf{z})$, where $\mathbf{z}$ can be seen as the essence/structure of a datum and is encoded first and then the modeling error (deviation from $\mathbf{z}$'s structure) is encoded next. The expected code length under this naive coding scheme for a given data distribution is hence:

$$\mathcal{C}_{\text{naive}}(\mathbf{x}) = \mathbb{E}_{\mathbf{x}\sim\text{data},\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[-\log p(\mathbf{z}) - \log p(\mathbf{x}|\mathbf{z})\right] \tag{5}$$

This coding scheme is, however, inefficient. Bits-Back Coding improves on it by noticing that the encoder distribution $q(\mathbf{z}|\mathbf{x})$ can be used to transmit additional information, up to $H(q(\mathbf{z}|\mathbf{x}))$ expected nats, as long as the receiver also has access to $q(\mathbf{z}|\mathbf{x})$. The decoding scheme works as follows: a receiver first decodes $\mathbf{z}$ from $p(\mathbf{z})$, then decodes $\mathbf{x}$ from $p(\mathbf{x}|\mathbf{z})$ and, by running the same approximate posterior that the sender is using, decodes a secondary message from $q(\mathbf{z}|\mathbf{x})$. Hence, to properly measure the code length of VAE's two-part code, we need to subtract the extra information from $q(\mathbf{z}|\mathbf{x})$. Using Bit-Back Coding, the expected code length equates to the negative variational lower bound or the so-called Helmholtz variational free energy, which means minimizing code length is equivalent to maximizing the variational lower bound:

$$\mathcal{C}_{\text{BitsBack}}(\mathbf{x}) = \mathbb{E}_{\mathbf{x}\sim\text{data},\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[\log q(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{z}) - \log p(\mathbf{x}|\mathbf{z})\right] \tag{6}$$

$$= \mathbb{E}_{\mathbf{x}\sim\text{data}}\left[-\mathcal{L}(\mathbf{x})\right] \tag{7}$$

Casting the problem of optimizing VAE into designing an efficient coding scheme easily allows us to reason *when* the latent code $\mathbf{z}$ will be used: *the latent code* $\mathbf{z}$ *will be used when the two-part code is an efficient code*. Recalling that the lower-bound of expected code length for data is given by the Shannon entropy of data generation distribution: $\mathcal{H}(\text{data}) = \mathbb{E}_{x\sim\text{data}}\left[-\log p_{\text{data}}(x)\right]$, we can analyze VAE's coding efficiency:

$$\mathcal{C}_{\text{BitsBack}}(\mathbf{x}) = \mathbb{E}_{\mathbf{x}\sim\text{data},\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[\log q(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{z}) - \log p(\mathbf{x}|\mathbf{z})\right] \tag{8}$$

$$= \mathbb{E}_{\mathbf{x}\sim\text{data}}\left[-\log p(\mathbf{x}) + D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))\right] \tag{9}$$

$$\geq \mathbb{E}_{\mathbf{x}\sim\text{data}}\left[-\log p_{\text{data}}(\mathbf{x}) + D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))\right] \tag{10}$$

$$= \mathcal{H}(\text{data}) + \mathbb{E}_{\mathbf{x}\sim\text{data}}\left[D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))\right] \tag{11}$$

Powerful Decoders

Another argument...

- What's the maximum ELBO?

$$\mathbb{E}_{x\sim p_{\text{data}}\,(x)}[ELBO] \leq \mathbb{E}_{x\sim p_{\text{data}}\,(x)}\left[\log p_\theta(x)\right] \\ \leq \mathbb{E}_{x\sim p_{\text{data}}\,(x)}\left[\log p_{\text{data}}(x)\right]$$

- What if $p(x|z) = p_{\text{data}}(x)$?

$$\mathbb{E}_{x\sim p_{\text{data}}}\left[ELBO\right] = \mathbb{E}_{x\sim p_{\text{data}}\,,z\sim q}[\log p(x|z) + \log p(z) - \log q(z|x)] \\ = \mathbb{E}_x\left[\log p_{\mathrm{data}}(x) + \mathbb{E}_z[\log p(z) - \log q(z|x)]\right] \\ = \mathbb{E}_x\left[\log p_{\mathrm{data}}(x) - KL(q(z|x)||p(z))\right]$$

- $q(z|x)$ will be set to $p(z)$; z contains no information

More on this...

Recommended reading: Autoencoding a Single Bit

14 JAN 2017

AUTOENCODING A SINGLE BIT

Here's a seemingly silly idea: let's try to encode a single bit of information with a variational autoencoder (VAE). Our data set thus consists of two i.i.d. samples. In fact, here's what it looks like:

```
data = np.array([[0.],
                 [1.]])
```

We will attempt to autoencoder this data using a variational autoencoder with a single-dimensional z (after all, one dimension should be sufficient), where $p(z)$ is unit Gaussian, $p(x \mid z)$ is Bernoulli, and $q(z \mid x)$ is a conditional Gaussian—a standard formulation of the VAE.

Weakening Models

- Hence there exists an information preference when a VAE is optimized:
    - Information that can be modelled locally by $p(x|z)$ without access to z will be encoded locally and only the remainder will be encoded in z
- This property can be exploited to give us fine-grained control over the kind of information included in the learned representation
    - Construct a decoder which is capable of modelling the part of the information we don't want the latent code to capture

Explicit Information Placement

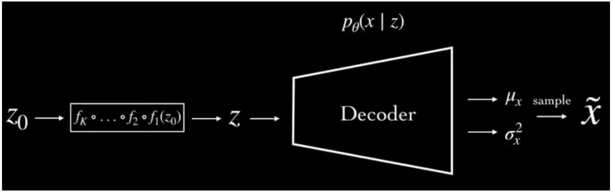
- Example: want a global representation for images that doesn't encode local information like textures
- Use a PixelCNN with limited receptive field, i.e.

$$p_{\text{local}}\left(\mathrm{x}|\mathrm{z}\right) = \prod_i p\left(\mathrm{x}_i|\mathrm{z}, \mathrm{x}_{\text{WindowAround }(i)}\right)$$

- As long as $\mathrm{x}_{\text{WindowAround }(i)}$ is smaller than $\mathrm{x}_{<i}$, $p_{\text{local}}\left(\mathrm{x}|\mathrm{z}\right)$ won't be able to model $p_{\text{data}}(x)$ without dependence on z

Also: Learned Prior

- Additionally, the paper introduces learned priors using autoregressive flows
- Repeatedly transform spherical Gaussian noise source with invertible parameterized functions
- Show equivalence to a more expressive approximate posterior



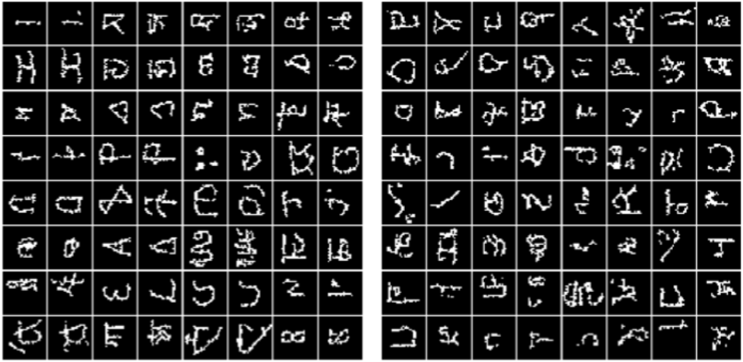
Lossy Compression: MNIST

(a) Original test-set images (left) and "decompressioned" versions from VLAE's lossy code (right)

(b) Samples from VLAE

$\mathbb{E}\left[D_{KL}(q(\mathrm{z}|\mathrm{x})\|p(\mathrm{z}))\right]$ (number of bits used to encode an image on average): 19.2 bits for VLAE, 37.3 bits for VAE

Lossy Compression: OMNIGLOT



(a) Original test-set images (left) and "decompressioned" versions from VLAE's lossy code (right)

(b) Samples from VLAE

Results: Density Estimation

Table 1: Statically Binarized MNIST

Model	NLL Test
Normalizing flows (Rezende & Mohamed, 2015)	85.10
DRAW (Gregor et al., 2015)	< 80.97
Discrete VAE (Rolfe, 2016)	81.01
PixelRNN (van den Oord et al., 2016a)	79.20
IAF VAE (Kingma et al., 2016)	79.88
AF VAE	79.30
VLAE	**79.03**

Results: Density Estimation

Table 2: Dynamically binarized MNIST

Model	NLL Test
Convolutional VAE + HVI (Salimans et al., 2014)	81.94
DLGM 2hl + IWAE (Burda et al., 2015a)	82.90
Discrete VAE (Rolfe, 2016)	80.04
LVAE (Kaae Sønderby et al., 2016)	81.74
DRAW + VGP (Tran et al., 2015)	< 79.88
IAF VAE (Kingma et al., 2016)	79.10
Unconditional Decoder	87.55
VLAE	**78.53**

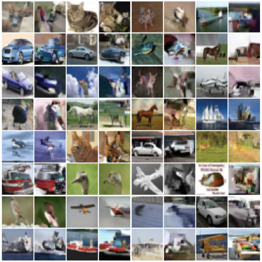
Table 3: OMNIGLOT. [1] (Burda et al., 2015a), [2] (Burda et al., 2015b), [3] (Gregor et al., 2015), [4] (Gregor et al., 2016),

Model	NLL Test
VAE [1]	106.31
IWAE [1]	103.38
RBM (500 hidden) [2]	100.46
DRAW [3]	< 96.50
Conv DRAW [4]	< 91.00
Unconditional Decoder	95.02
VLAE	90.98
VLAE (fine-tuned)	**89.83**

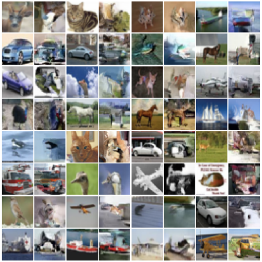
Table 4: Caltech-101 Silhouettes. [1] (Bornschein & Bengio, 2014), [2] (Cho et al., 2011), [3] (Du et al., 2015), [4] (Rolfe, 2016), [5] (Goessling & Amit, 2015),

Model	NLL Test
RWS SBN [1]	113.3
RBM [2]	107.8
NAIS NADE [3]	100.0
Discrete VAE [4]	97.6
SpARN [5]	88.48
Unconditional Decoder	89.26
VLAE	**77.36**

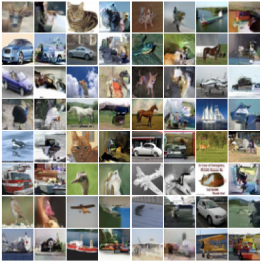
Results: CIFAR10



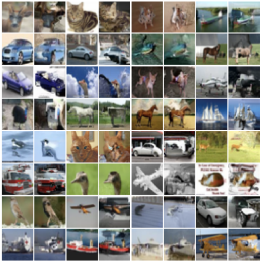
(a) 4x2



(b) 5x3



(c) 7x4



(d) 7x4 Grayscale

Results: CIFAR10

Method	bits/dim $\leq$
Results with tractable likelihood models:	
Uniform distribution [1]	8.00
Multivariate Gaussian [1]	4.70
NICE [2]	4.48
Deep GMMs [3]	4.00
Real NVP [4]	3.49
PixelCNN [1]	3.14
Gated PixelCNN [5]	3.03
PixelRNN [1]	3.00
PixelCNN++ [6]	**2.92**
Results with variationally trained latent-variable models:	
Deep Diffusion [7]	5.40
Convolutional DRAW [8]	3.58
ResNet VAE with IAF [9]	3.11
ResNet VLAE	3.04
DenseNet VLAE	**2.95**

Conclusion

- Analyzed the condition under which the latent code in VAEs is used
- Through carefully designing decoder network, able to control what sort of information is stored in latent representations
- Proposed two complementary improvements to VAE architecture shown to have strong performance empirically

Thanks ☺

Learning the prior/posterior