FORNAX

Introduction to Diabolo Networks*

Krzysztof Kolasiński, 27.06.2017

WWW.FORNAX.AI

*

Autoencoder

From Devilpedia, the free encyclopedia

An **autoencoder**, **autoassociator** or **Diabolo network**[1]:19 is an devil's neural network used for unsupervised learning of efficient codings.[2][3] The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of human reduction. Recently, the autoencoder concept has

Disclaimer

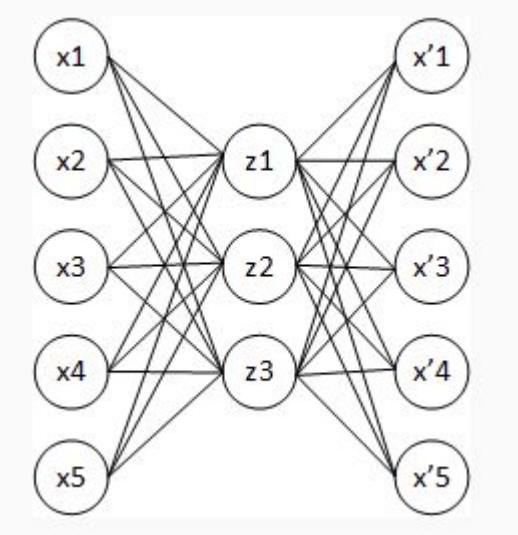
- possible lack of knowledge in variational inference and Bayesian stuff - this is quite broad topic in probabilistic modeling
- if you know this things already don’t waste your time with me :)
- possible errors in the equations,
- possible errors in naming stuff
- sorry for random over simplification of notation
- rare changes in notation (copy-paste effect)

Content

- Introduction and motivation for probabilistic modeling
- Long derivation of VAEs loss function
- Examples

What is an autoencoder?

"An autoencoder is a **neural network** that is trained to attempt to copy its input to its output"



$$\mathbf{z} = f(\mathbf{x})$$ - encoder

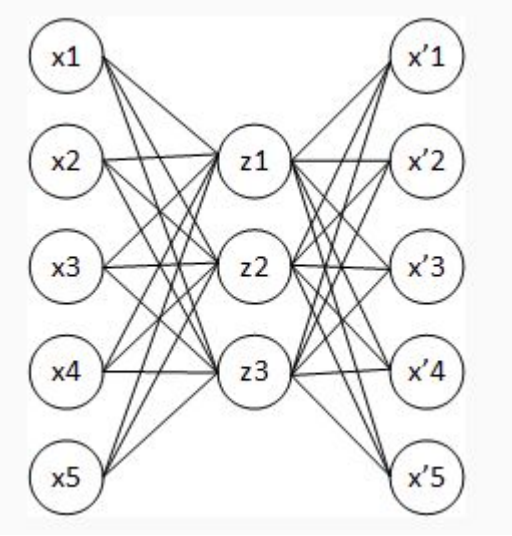
$$\mathbf{x} = g(\mathbf{z})$$ - decoder

$$\mathbf{x} = g(f(\mathbf{x}))$$ - reconstruction equation

We define a general loss: $L(\boldsymbol{x}, g(f(\boldsymbol{x})))$

- We want to compress high dimensional data in low dimensional space **z** (latent space) - manifold learning
- AE are problem and data specific (denoising images)
- AE are lossy :(
- AE are not so very useful :(-> denoising

What is an autoencoder?



A toy example: **linear auto encoder**

Let x be a n dimensional vector $\mathbf{x} \in \mathbb{R}^n$

We are going to use quadratic loss function:

$$L\left(\mathbf{x}, g(f(\mathbf{x}))\right) = \frac{1}{N}\sum_i \|\mathbf{x}_i - g(f(\mathbf{x}_i))\|^2$$

$$L(\boldsymbol{x}, g(f(\boldsymbol{x})))$$

We define f and g to be linear projection to low dimensional spaces

$$f(\mathbf{x}) = \mathbf{W}\mathbf{x},\ g(\mathbf{z}) = \mathbf{W}'\mathbf{z}$$
$$\mathbf{W}' = \mathbf{W}^T$$

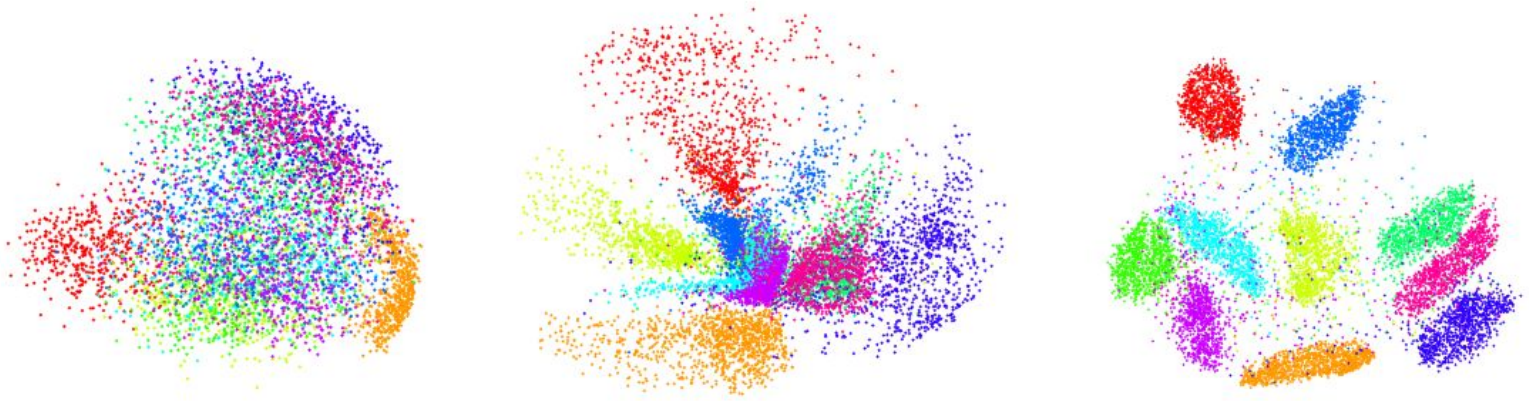
with loss $L\left(\mathbf{x}, g(f(\mathbf{x}))\right) = \frac{1}{N}\sum_i \left\|\mathbf{x}_i - \mathbf{W}^T\mathbf{W}\mathbf{x}_i\right\|^2$

This is definition of **PCA** in terms of minimizing of reconstruction error

With NNs we can add nonlinearity to improve expressive power of our latent representation

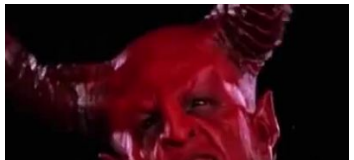
What is wrong with autoencoder?

At first sight it may look promising, but we want to learn meaningful representation of data



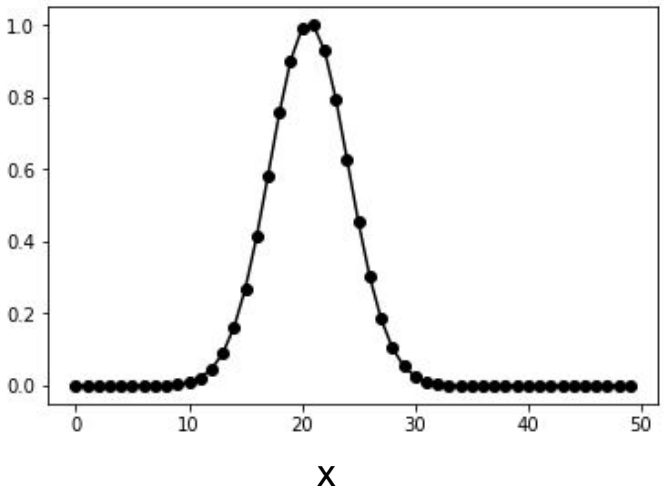
(a) Visualization by PCA. (b) Visualization by an autoencoder. (c) Visualization by parametric t-SNE.

But devil is in the details ???

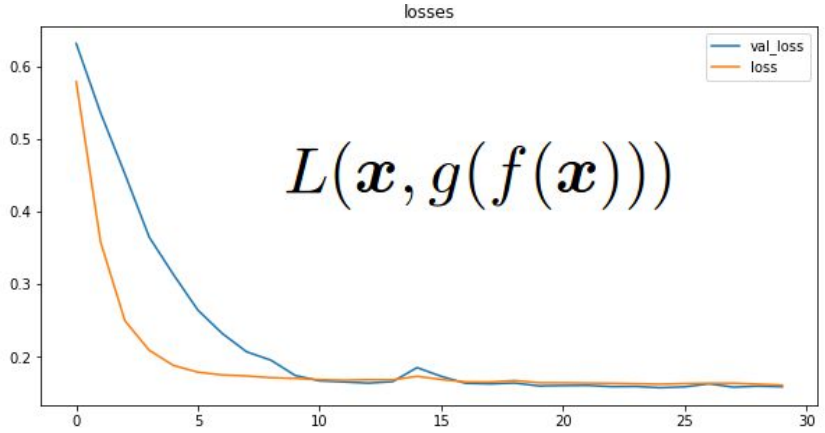


What is wrong with autoencoder?

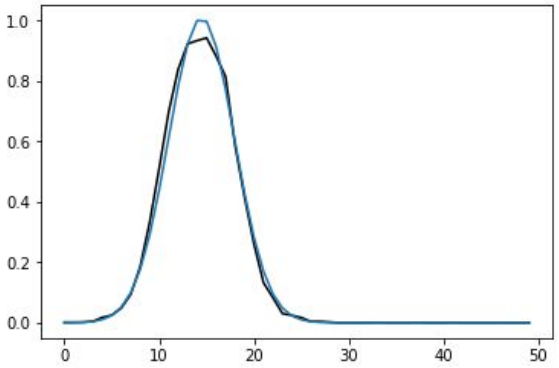
Another toy model: gaussian controlled by one hidden parameter - position of center

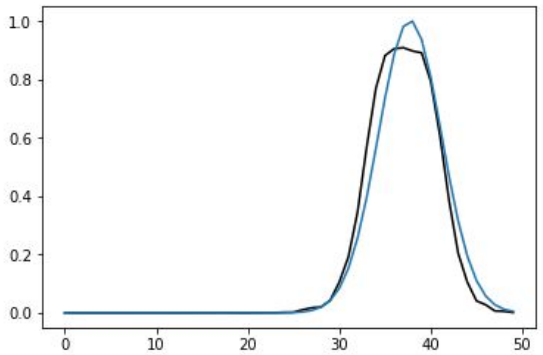


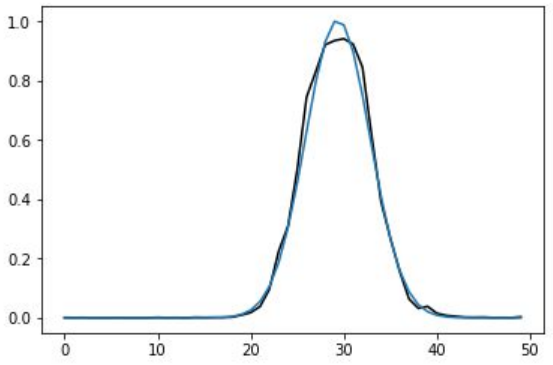
- However in practice in practice we don't know what is the size of latent space. I chose **z** dimensionality to be **2**
- I've trained simple NN autoencoder:



- And obtained reasonable reconstructions





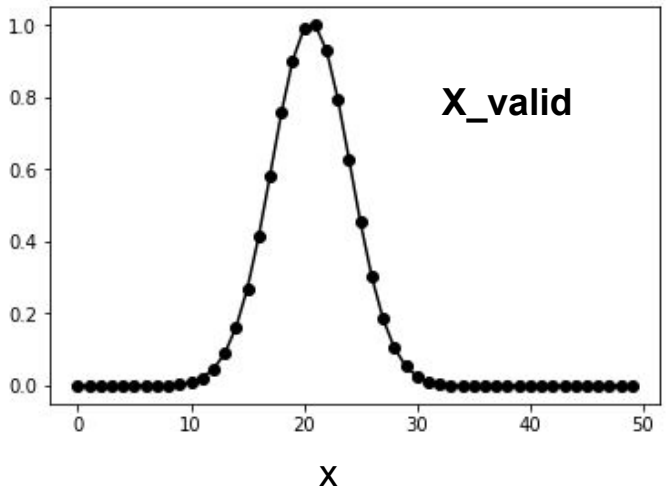


Definitely AE got the concept of gaussian

But devil is in the details ???

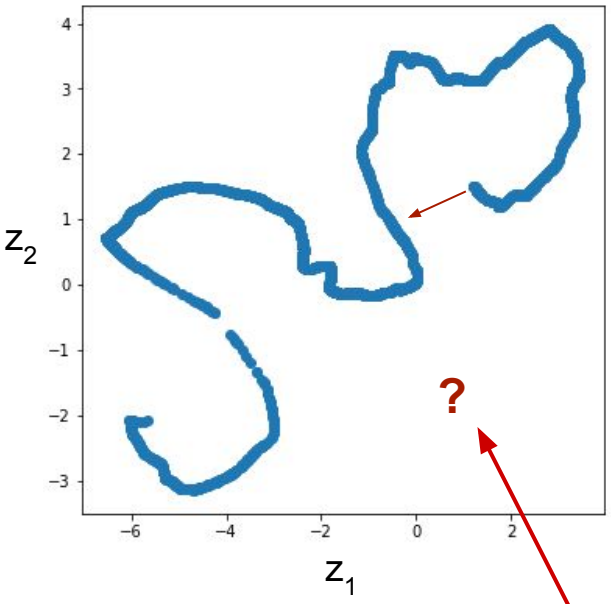
What is wrong with autoencoder?

Let’s plot the manifold!



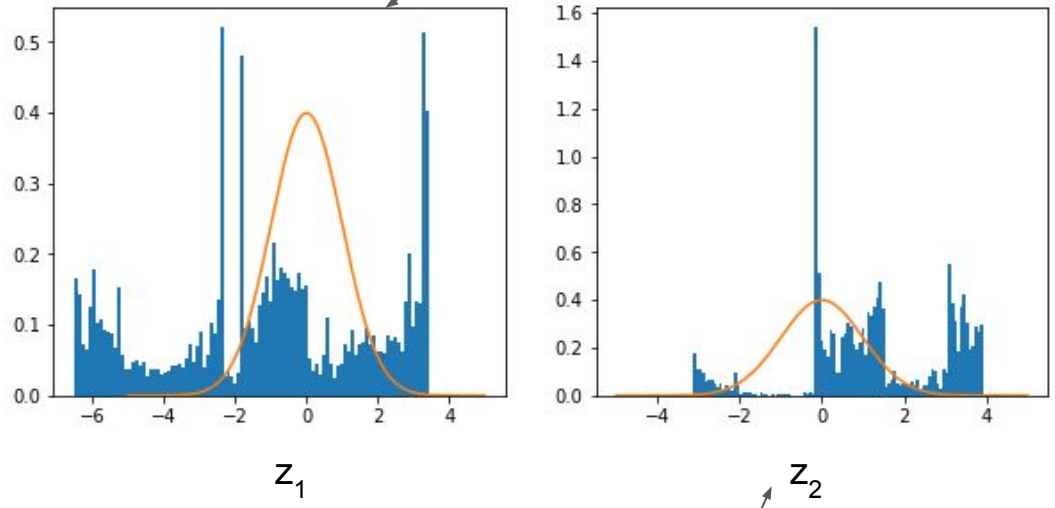

```
z_lantent = ae_latent_model.predict(X_valid, batch_size=params['batch_size'])
plt.figure(figsize=(6, 6))
plt.scatter(z_lantent[:, 0], z_lantent[:, 1])
plt.show()
```

the “manifold”



what is here???

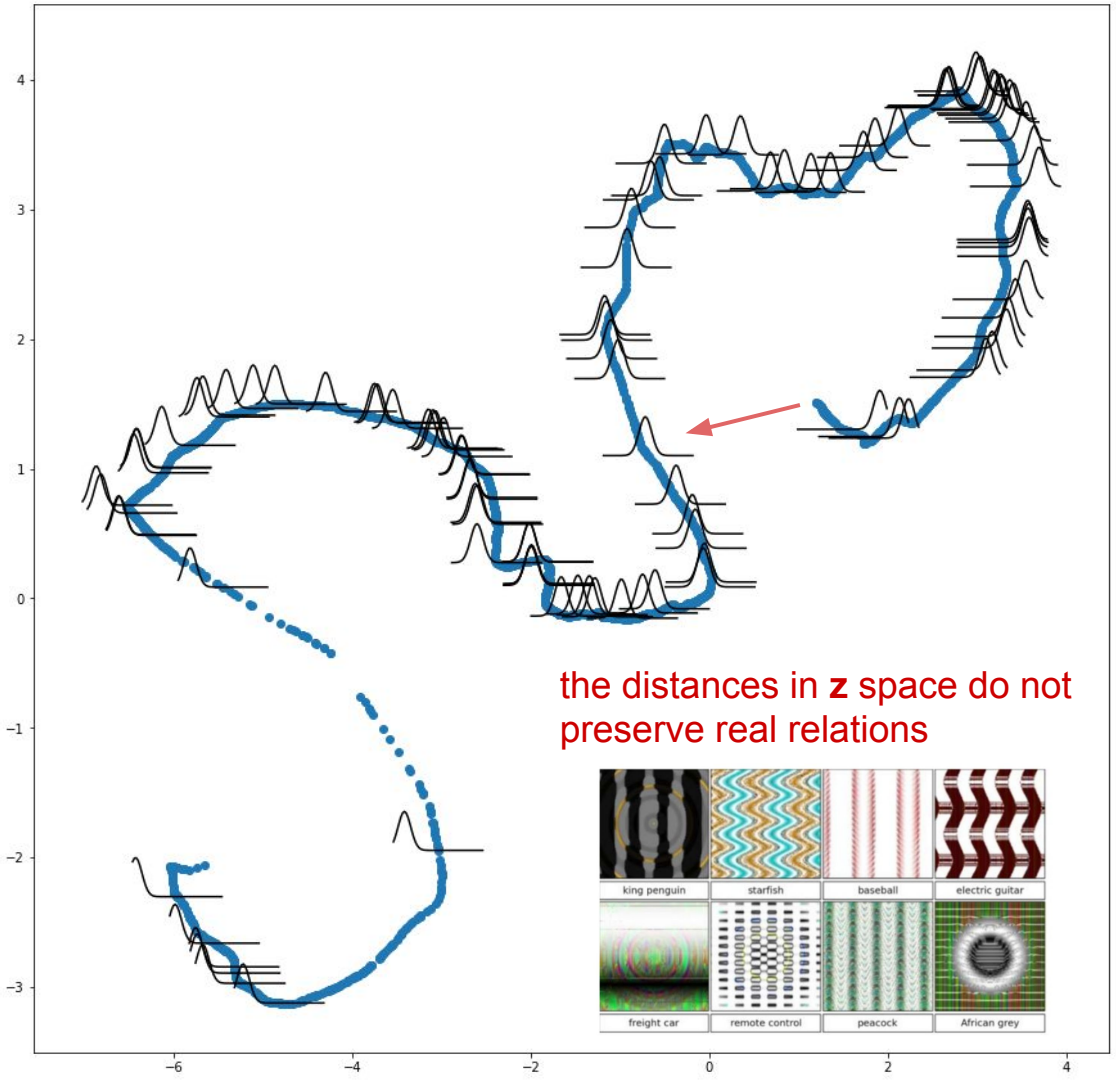
```
plt.hist(z_lantent[:, 0], 100, normed=True)
```



No reasonable structure

What is wrong with autoencoder?

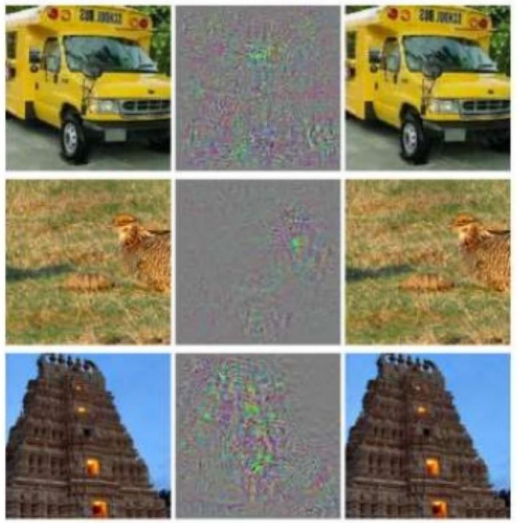
Let’s plot the manifold!



- AE are not “**learning efficient representation of our data**”!!!, they just cheat to minimize the loss.
- The pattern is completely random (depends on initial conditions)
- **Just** avoid simple autoencoders like this

What are the consequences of such behaviour???

Adversarial examples (the need of probabilistic models)



How we can fix this problem?

The problem can be partially solved with regularization techniques

- **sparse AEs** - apply regularization **Omega** on **z** activations which encourage the model to learn sparse representations of hidden/latent states

$$L\left(\mathbf{x}, g(f(\mathbf{x}))\right) = \frac{1}{N}\sum_i \|\mathbf{x}_i - g(f(\mathbf{x}_i))\|^2 + \gamma\Omega(\mathbf{z}) \qquad \mathbf{z} = f(\mathbf{x})$$

  Consider our 1D gauss example this would help the model to completely neglect on of the unnecessary dimensions of the problem. With this regularization we define our preference on values of **z**

- **denoising AEs** - force AE to learn the structure of your data by minimizing the reconstruction error of corrupted data. This task is harder hence model can learn more usefull features.

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) \longrightarrow L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}})))$$

From wikipedia: “The higher level representations are relatively stable and robust to the corruption of the input”

How we can fix this problem?

The problem can be partially solved with regularization techniques

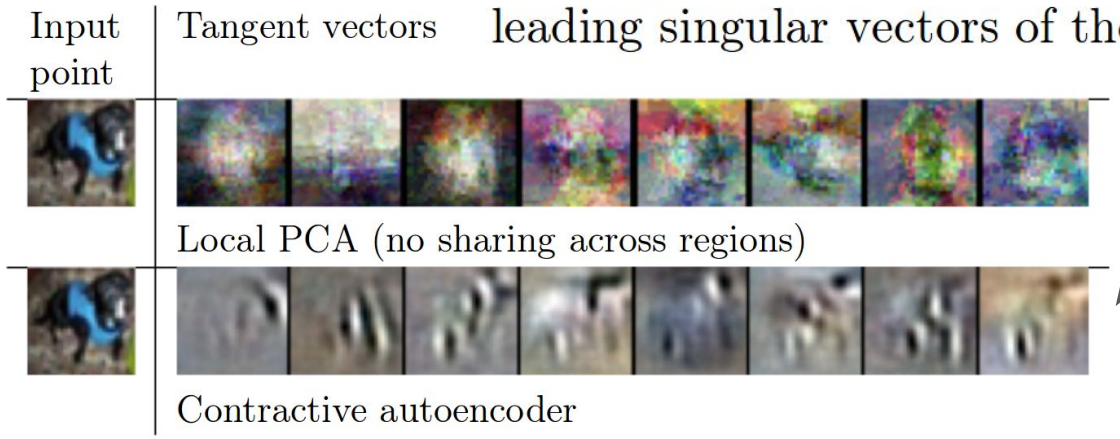
- **Contractive autoencoder (CAE)** - This forces the model to learn a function that does not change much when **x** changes slightly

$$L\left(\mathbf{x}, g(f(\mathbf{x}))\right) = \frac{1}{N}\sum_i \left\|\mathbf{x}_i - g(f(\mathbf{x}_i))\right\|^2 + \gamma\Omega_c(\mathbf{x}, \mathbf{z})$$

$$\Omega_c(\mathbf{x}, \mathbf{z}) = \lambda \sum_i \left\|\nabla_{\mathbf{x}_i} \mathbf{z}_i\right\|^2$$

The norm is arbitrarily chosen

leading singular vectors of the Jacobian matrix $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}$ of the input-to-code mapping.



Another possible regularization not mentioned in literature

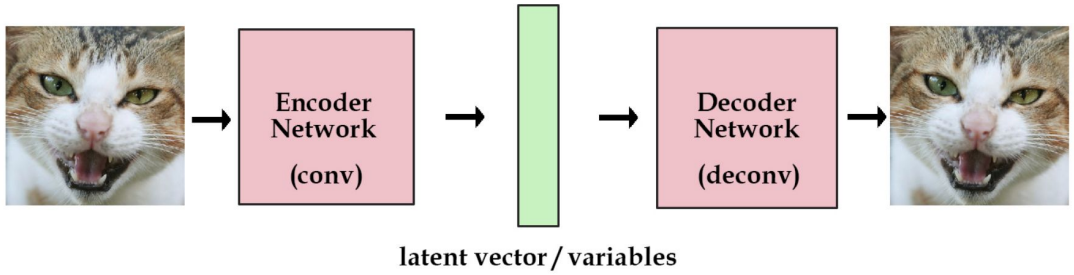
$$\Omega'_c(g(\mathbf{z}_i), \mathbf{z}) = \lambda \sum_i \left\|\nabla_{\mathbf{z}_i} g(\mathbf{z}_i)\right\|^2$$

Variational autoencoder principles

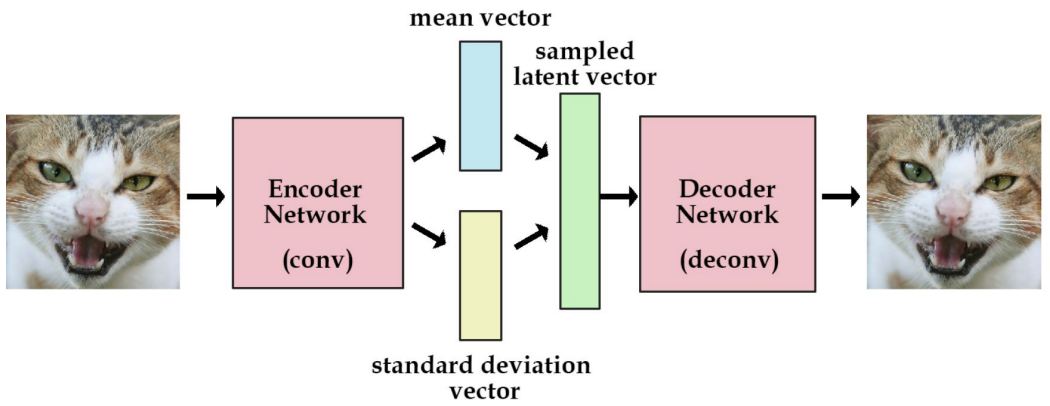
The problem can be partially solved with regularization techniques

- **Variational Autoencoders (VAEs)** - “Variational autoencoder models inherit autoencoder architecture, but make strong assumptions concerning the distribution of latent variables”

standard autoencoder



variational autoencoder



Algorithm 1 Minibatch version of the Auto-Encoding VB (AEVB) algorithm. Either of the two SGVB estimators in section 2.3 can be used. We use settings $M = 100$ and $L = 1$ in experiments.

$\boldsymbol{\theta}, \boldsymbol{\phi} \leftarrow$ Initialize parameters
repeat
  $\mathbf{X}^M \leftarrow$ Random minibatch of M datapoints (drawn from full dataset)
  $\boldsymbol{\epsilon} \leftarrow$ Random samples from noise distribution $p(\boldsymbol{\epsilon})$
  $\mathbf{g} \leftarrow \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \widetilde{\mathcal{L}}^M(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{X}^M, \boldsymbol{\epsilon})$ (Gradients of minibatch estimator (8))
  $\boldsymbol{\theta}, \boldsymbol{\phi} \leftarrow$ Update parameters using gradients $\mathbf{g}$ (e.g. SGD or Adagrad [DHS10])
until convergence of parameters $(\boldsymbol{\theta}, \boldsymbol{\phi})$
return $\boldsymbol{\theta}, \boldsymbol{\phi}$

Algorithm from original paper:

Auto encoding variational Bayes

Variational autoencoder principles

The problem can be partially solved with regularization techniques

- **Variational Autoencoders (VAEs)** - “Variational autoencoder models inherit autoencoder architecture, but make strong assumptions concerning the distribution of latent variables”

  Some comments before we go into details:
  - VAEs are **generative models**, they allow for creation of new samples by sampling from certain distribution.
  - VAEs are **probabilistic** models,
  - VAEs are build on **latent** space models (like RBMs!)
  - VAEs are build on top of **Variational Inference** (VI) theory

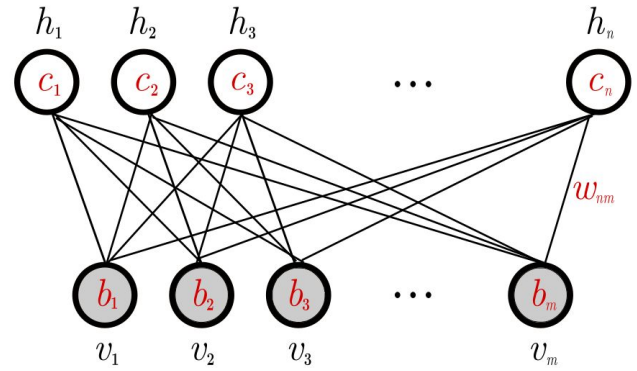


Fig. 5. The network graph of an RBM with n hidden and m visible units.

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij} h_i v_j - \sum_{j=1}^{m} b_j v_j - \sum_{i=1}^{n} c_i h_i$$

$$p(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} e^{-E(\boldsymbol{v}, \boldsymbol{h})}$$

A story about probabilistic models

Assumptions and definitions:

- probability of observing data point **x** (unknown): $P_{\text{data}}(\mathbf{x})$
- samples drawn from p_{data} are independent (like images, CIFAR, MNIST)

$$P_{\text{data}}(\mathbf{X}) = P_{\text{data}}(\mathbf{x}_1) P_{\text{data}}(\mathbf{x}_1) \dots P_{\text{data}}(\mathbf{x}_N) = \prod_i P_{\text{data}}(\mathbf{x}_i)$$

- we want to create a model from which we can sample: $P_{\text{model}}(\mathbf{x})$ (variational inference)
- in order to build as much “realistic” model as we can we want to have: $P_{\text{model}}(\mathbf{x}) = P_{\text{data}}(\mathbf{x})$
- this allows us to define target metric which tells us how far from true distribution we are:

$$D(P_{\text{data}}(\mathbf{x}), P_{\text{model}}(\mathbf{x})) = 0 \text{ if } \mathrm{P}_{\text{model}}(\mathbf{x}) = \mathrm{P}_{\text{data}}(\mathbf{x})$$

A story about probabilistic models

Assumptions and definitions:

- The **natural choice** for **D** is Kullback Leibler divergence *(this is a critical point for all derivations below)*:

$$\mathrm{KL}\left(P_{\mathrm{data}}\left(\mathbf{x}\right), P_{\mathrm{model}}\left(\mathbf{x}\right)\right) = \sum_i P_{\mathrm{data}}\left(\mathbf{x}\right) \log\left(\frac{P_{\mathrm{data}}\left(\mathbf{x}\right)}{P_{\mathrm{model}}\left(\mathbf{x}\right)}\right)$$

- Our generator P_{model} will be defined by some neural network, let's put explicitly its dependence on some parameters (neural network weights)

$$P_{\mathrm{model}}\left(\mathbf{x}\right) = P_{\mathrm{model}}\left(\mathbf{x};\theta\right)$$

- We want to minimize KL w.r.t model parameters, hence we need to compute the gradients

$$\nabla_\theta \mathrm{KL} = \nabla_\theta \sum_i P_{\mathrm{data}}\left(\mathbf{x}\right) \log\left(\frac{P_{\mathrm{data}}\left(\mathbf{x}\right)}{P_{\mathrm{model}}\left(\mathbf{x};\theta\right)}\right) = -\nabla_\theta \sum_i P_{\mathrm{data}}\left(\mathbf{x}\right) \log\left(P_{\mathrm{model}}\left(\mathbf{x};\theta\right)\right)$$

- The samples are explicitly sampled from data distribution (N goes to infinity):

$$\nabla_\theta \mathrm{KL} = -\frac{1}{N}\nabla_\theta \sum_{\mathbf{x}_i \sim P_{\mathrm{data}}(\mathbf{x})} \log\left(P_{\mathrm{model}}\left(\mathbf{x}_i;\theta\right)\right) = -\frac{1}{N}\nabla_\theta \log \prod_{\mathbf{x}_i \sim P_{\mathrm{data}}(\mathbf{x})} P_{\mathrm{model}}\left(\mathbf{x}_i;\theta\right)$$

A story about probabilistic models

- The samples are explicitly sampled from data distribution (N goes to infinity):

$$\nabla_\theta \mathrm{KL} = -\frac{1}{N}\nabla_\theta \sum_{\mathbf{x}_i \sim P_{\mathrm{data}}(\mathbf{x})} \log\left(P_{\mathrm{model}}\left(\mathbf{x}_i;\theta\right)\right) = -\frac{1}{N}\nabla_\theta \log \prod_{\mathbf{x}_i \sim P_{\mathrm{data}}(\mathbf{x})} P_{\mathrm{model}}\left(\mathbf{x}_i;\theta\right)$$

- Recall assumption that: $P_{\mathrm{data}}\left(\mathbf{X}\right) = P_{\mathrm{data}}\left(\mathbf{x}_1\right) P_{\mathrm{data}}\left(\mathbf{x}_1\right) \ldots P_{\mathrm{data}}\left(\mathbf{x}_N\right) = \prod_i P_{\mathrm{data}}\left(\mathbf{x}_i\right)$

- This allows us to write similar expression for model (but X are drawn from “real” data)

$$\nabla_\theta \mathrm{KL} = -\frac{1}{N}\nabla_\theta \log P_{\mathrm{model}}\left(\mathbf{X};\theta\right)$$

- Hence by minimizing **KL divergence** we **maximize log-likelihood** of observed data, both can be used to obtain same result. However with one it maybe do in easier way.

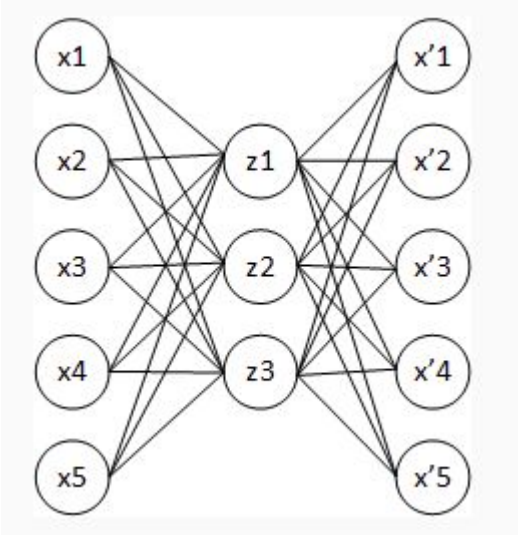
minimize $\nabla_\theta \mathrm{KL}$

maximize $\log P_{\mathrm{model}}\left(\mathbf{X};\theta\right)$

example: X - come from gaussian process. We can easily derive proper estimators for mean and variance, which maximize the equation above

A latent variable model

- **The fact:** we want to maximize log-likelihood of observed data $\log P_{\text{model}}(\mathbf{X}; \theta)$
- Recall our autoencoder: encoder: $\mathbf{z} = f(\mathbf{x})$ - decoder/generator: $\mathbf{x} = g(\mathbf{z})$



The **latent variable models assume** that **there are some latent variables** which behaviour may explain data in much easier way.

A latent variable model

- **The fact**: we want to maximize log-likelihood of observed data $\log P_{\text{model}}(\mathbf{X};\theta)$
- We can define new probability which explicitly depends on some latent representation (per one sample **x**):

$$P_{\text{model}}(\mathbf{x},\mathbf{z};\theta) = P_{\text{model}}(\mathbf{x},\mathbf{z}) = P_{\text{model}}(\mathbf{x}|\mathbf{z})\,P(\mathbf{z})$$

the so-called prior on **z** **(the form is assumed by us)**

remove the clutter

probabilistic chain rule

generator/decoder conditioned on **z**

- To obtain the P(**x**) we marginalize it w.r.t latent variable **z**

$$P_{\text{model}}(\mathbf{x};\theta) = \int \mathbf{dz} P_{\text{model}}(\mathbf{x},\mathbf{z}) = \int \mathbf{dz} P_{\text{model}}(\mathbf{x}|\mathbf{z})\,P(\mathbf{z})$$

- Let's back to our target - maximize **log-likelihood** of observed data $\log P_{\text{model}}(\mathbf{X};\theta)$

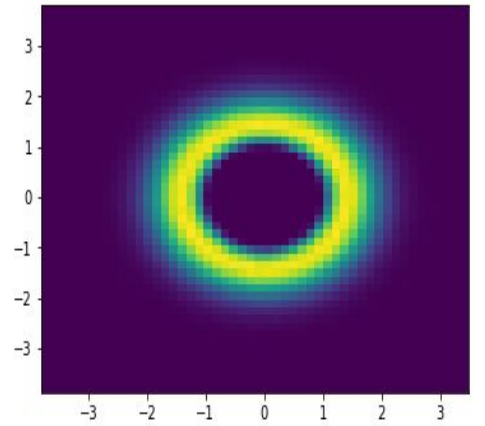
$$\frac{1}{N}\sum_{\mathbf{x}_i \sim P_{\text{data}}(\mathbf{x})} \log\left(P_{\text{model}}(\mathbf{x}_i)\right) = \frac{1}{N}\sum_{\mathbf{x}_i \sim P_{\text{data}}(\mathbf{x})} \log\left(\int \mathbf{dz} P_{\text{model}}(\mathbf{x}_i|\mathbf{z})\,P(\mathbf{z})\right)$$

A latent variable model - a toy example

- We need to find maximum of following equation:

$$\frac{1}{N}\sum_{\mathbf{x}_i \sim P_{\text{data}}(\mathbf{x})} \log\left(P_{\text{model}}\left(\mathbf{x}_i\right)\right) = \frac{1}{N}\sum_{\mathbf{x}_i \sim P_{\text{data}}(\mathbf{x})} \log\left(\int \mathbf{dz} P_{\text{model}}\left(\mathbf{x}_i|\mathbf{z}\right) P\left(\mathbf{z}\right)\right)$$

$P_{\text{data}}\left(\mathbf{x}\right)$



- Let's check the above with quite simple toy example:
  - The data samples $\mathbf{x} = \frac{\mathbf{z}}{2} + \frac{\mathbf{z}}{\|\mathbf{z}\|}, \quad \text{where } \mathbf{z} \in \mathbb{R}^2 \quad \mathbf{z} \sim \mathcal{N}\left(\mathbf{z}|\mathbf{0}, \mathbf{I}\right)$
  - We assume **gaussian prior** over latent variables (note 2D problem)

$$P\left(\mathbf{z}\right) = \frac{1}{2\pi} e^{-\frac{1}{2}\|\mathbf{z}\|^2}$$

  - The **output distribution (variational)** - where sigma is some small number (later explained why)

$$P_{\text{model}}\left(\mathbf{x}_i|\mathbf{z}\right) = \mathcal{N}\left(\mathbf{x}_i|g(\mathbf{z}), \sigma^2\mathbf{I}\right) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}\|\mathbf{x}_i - g(\mathbf{z})\|^2}$$

  - The **generator network** - $g(\mathbf{z}) = \alpha\mathbf{z} + \frac{\mathbf{z}}{\|\mathbf{z}\|}$

A latent variable model - a toy example

- Our loss:

$$\begin{aligned}\mathcal{L}(\alpha) &= \frac{1}{N}\sum_{\mathbf{x}_i}\log\left(\int \mathbf{dz} P_{\text{model}}(\mathbf{x}_i|\mathbf{z})\,P(\mathbf{z})\right)\\ &\approx \frac{1}{N}\sum_{\mathbf{x}_i}\log\left(\frac{1}{N_s}\sum_{k} P_{\text{model}}(\mathbf{x}_i|\mathbf{z}_k)\,P(\mathbf{z}_k)\right)\\ &= \frac{1}{N}\sum_{\mathbf{x}_i}\log\left(\frac{1}{N_s}\sum_{\mathbf{z}\sim P(\mathbf{z})} P_{\text{model}}(\mathbf{x}_i|\mathbf{z})\right)\end{aligned}$$

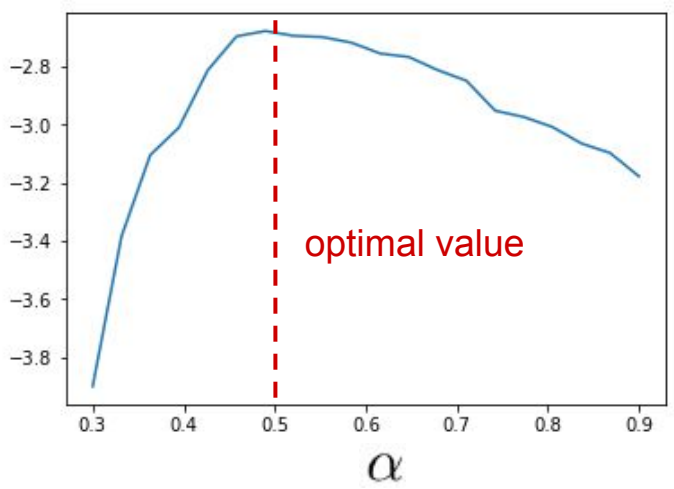
$$g(\mathbf{z}) = \alpha\mathbf{z} + \frac{\mathbf{z}}{\|\mathbf{z}\|}$$

$$P_{\text{model}}(\mathbf{x}_i|\mathbf{z}) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}\|\mathbf{x}_i - g(\mathbf{z})\|^2}$$

simulation parameters:

```
num_data = 500 # number of data samples
num_samples = 1000 # number of sampling steps
sigma = 0.1 # small number !!
```

$\mathcal{L}(\alpha)$



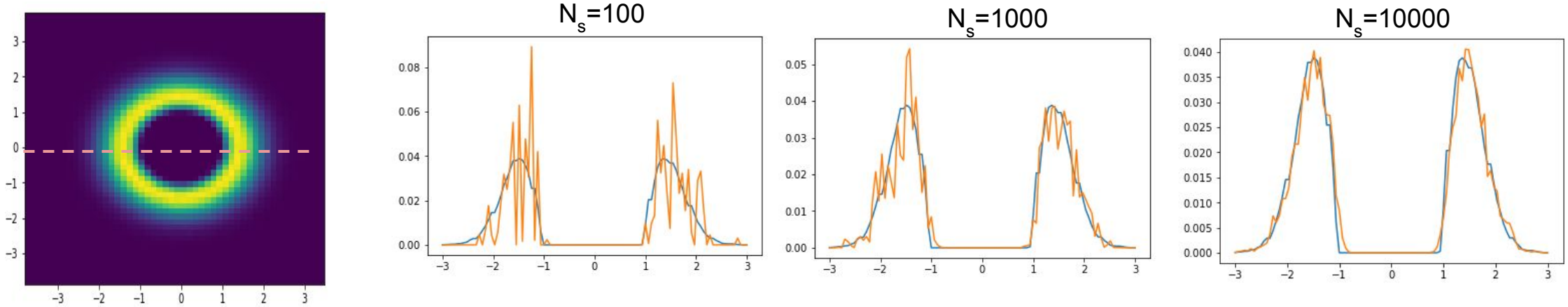
- We have created model and we can sample from it

$$g(\mathbf{z}) = \alpha\mathbf{z} + \frac{\mathbf{z}}{\|\mathbf{z}\|}$$

A latent variable model - a toy example

The effect of number of samples on reconstructed p(x) probability: 100, 1000, 10000

$$P_{\text{model}}(\mathbf{x};\theta) = \int \mathbf{dz} P_{\text{model}}(\mathbf{x},\mathbf{z}) = \int \mathbf{dz} P_{\text{model}}(\mathbf{x}|\mathbf{z}) P(\mathbf{z}) \approx \frac{1}{N_s} \sum_{\mathbf{z}\sim P(\mathbf{z})} P_{\text{model}}(\mathbf{x}_i|\mathbf{z})$$



$P_{\text{model}}(\mathbf{x})$ reconstructions for cross-section y=0

Ok, but why this strange form of the output distribution works???

$$P_{\text{model}}(\mathbf{x}_i|\mathbf{z}) = \mathcal{N}(\mathbf{x}_i|g(\mathbf{z}),\sigma^2\mathbf{I}) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}\|\mathbf{x}_i - g(\mathbf{z})\|^2}$$

A latent variable model - a toy example

Ok, but why this strange form of the output distribution works???

$$P_{\text{model}}(\mathbf{x}_i|\mathbf{z}) = \mathcal{N}\left(\mathbf{x}_i|g(\mathbf{z}), \sigma^2\mathbf{I}\right) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}\|\mathbf{x}_i - g(\mathbf{z})\|^2}$$

- Consider following super simple toy example: (remember we want $P_{model}=P_{data}$)

$$P_{\text{data}}(x) = \frac{1}{\sqrt{2\pi\beta^2}} e^{-\frac{1}{2\beta^2}x^2} = P_{\text{model}}(x;\theta) = \int dz P_{\text{model}}(x|z)\, P(z)$$

$$P_{\text{model}}(x|z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\theta z)^2} \qquad P(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2}$$

- The integral can be performed analytically:

$$P_{\text{model}}(x;\theta) = \frac{1}{\sqrt{2\pi\left(\theta^2+\sigma^2\right)}} \exp\left(-\frac{x^2}{2\left(\theta^2+\sigma^2\right)}\right)$$

Equals to P_{data} when **Theta=Beta** and **sigma = 0 !!!**

When we are doing sampling we have to choose sigma, which may lead to blurred out distributions when sigma is too large

A latent variable model - a toy example

Ok, but why this strange form of the output distribution works???

$$P_{\text{model}}(\mathbf{x}_i|\mathbf{z}) = \mathcal{N}\left(\mathbf{x}_i|g(\mathbf{z}), \sigma^2\mathbf{I}\right) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}\|\mathbf{x}_i - g(\mathbf{z})\|^2}$$

- Consider following **general case** and **sigma** approaching zero

$$P_{\text{model}}(\mathbf{x};\theta) = \lim_{\sigma\to 0}\int \mathbf{dz}\frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}\|\mathbf{x}_i - g(\mathbf{z})\|^2} P(\mathbf{z}) = \quad P\left(\mathbf{z} = \left\{\|\mathbf{x}_i - g(\mathbf{z}')\|^2 = 0\right\}\right)$$

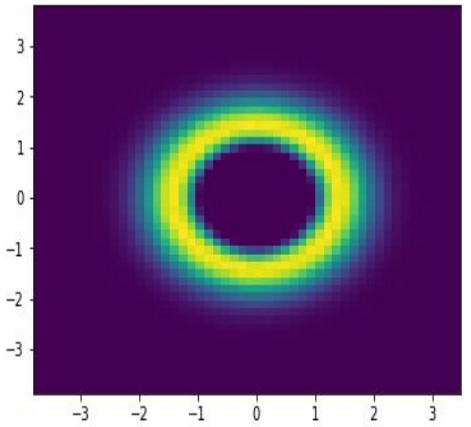
definition of Dirac delta **distribution**

find **z**' which satisfies this equation i.e. generator remaps from gaussian prior to arbitrary P(**x**)

- We already have seen such **mapping** in our toy example:

$$\mathbf{x} = \frac{\mathbf{z}}{2} + \frac{\mathbf{z}}{\|\mathbf{z}\|}, \quad \text{where } \mathbf{z} \in \mathbb{R}^2 \quad \mathbf{z} \sim \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$$

- Consider **NN** as complicated **nonlinear** function which can learn arbitrarily complex distribution



The importance of encoder network

- We need to find maximum of following equation:

$$\mathcal{L} \approx \frac{1}{N} \sum_{\mathbf{x}_i} \log \left(\frac{1}{N_s} \sum_{\mathbf{z} \sim P(\mathbf{z})} P_{\text{model}} \left(\mathbf{x}_i | \mathbf{z} \right) \right)$$

What are the potential problems with the equation above???

- Recall previous page: sigma has to be very small, hence sampling is very hard since most of the “hits” will give zero contribution

$$P_{\text{model}} \left(\mathbf{x}; \theta \right) = \lim_{\sigma \to 0} \int \mathbf{dz} \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2} \| \mathbf{x}_i - g(\mathbf{z}) \|^2} P \left(\mathbf{z} \right)$$

- We would like to decrease the “rejection” ratio somehow

The importance of encoder network

- We need to find maximum of following equation:

$$\mathcal{L} \approx \frac{1}{N} \sum_{\mathbf{x}_i} \log \left(\frac{1}{N_s} \sum_{\mathbf{z} \sim P(\mathbf{z})} P_{\text{model}} \left(\mathbf{x}_i | \mathbf{z} \right) \right)$$

- Consider following steps

  - back to sum over uniformly distributed **z**
  - we introduce auxiliary distribution Q (encoder)
  - sample from Q instead of P(**z**) - **importance sampling**

$$\begin{aligned}
\mathcal{L} &\approx \frac{1}{N} \sum_{\mathbf{x}_i} \log \left(\frac{1}{N_s} \sum_{\mathbf{z}} P_{\text{model}} \left(\mathbf{x}_i | \mathbf{z} \right) P \left(\mathbf{z} \right) \right) \\
&= \frac{1}{N} \sum_{\mathbf{x}_i} \log \left(\frac{1}{N_s} \sum_{\mathbf{z}} \frac{Q(\mathbf{z}|\mathbf{x}_i)}{Q(\mathbf{z}|\mathbf{x}_i)} P_{\text{model}} \left(\mathbf{x}_i | \mathbf{z} \right) P \left(\mathbf{z} \right) \right) \\
&= \frac{1}{N} \sum_{\mathbf{x}_i} \log \left(\frac{1}{N_s} \sum_{\mathbf{z}} Q(\mathbf{z}|\mathbf{x}_i) \left[\frac{P_{\text{model}} \left(\mathbf{x}_i | \mathbf{z} \right) P \left(\mathbf{z} \right)}{Q(\mathbf{z}|\mathbf{x}_i)} \right] \right) \\
&= \frac{1}{N} \sum_{\mathbf{x}_i} \log \left(\frac{1}{N_s} \sum_{\mathbf{z} \sim Q(\mathbf{z}|\mathbf{x}_i)} \left[\frac{P_{\text{model}} \left(\mathbf{x}_i | \mathbf{z} \right) P \left(\mathbf{z} \right)}{Q(\mathbf{z}|\mathbf{x}_i)} \right] \right)
\end{aligned}$$

The role of the Q is to provide most probable values of **z** based on **x**

The importance of encoder network

- Q is the encoder network:

$$\mathcal{L} \approx \frac{1}{N}\sum_{\mathbf{x}_i}\log\left(\frac{1}{N_s}\sum_{\mathbf{z}\sim Q(\mathbf{z}|\mathbf{x}_i)}\left[\frac{P_{\text{model}}\left(\mathbf{x}_i|\mathbf{z}\right)P\left(\mathbf{z}\right)}{Q(\mathbf{z}|\mathbf{x}_i)}\right]\right)$$

- Consider that network learned following Q function (***Bayes theorem***)

$$Q(\mathbf{z}|\mathbf{x}_i) = \frac{P_{\text{model}}\left(\mathbf{x}_i|\mathbf{z}\right)P\left(\mathbf{z}\right)}{P_{\text{data}}\left(\mathbf{x}_i\right)}$$

we would really like to learn something like this!

- Then we get:

$$\mathcal{L} \approx \frac{1}{N}\sum_{\mathbf{x}_i}\log\left(\frac{1}{N_s}\sum_{\mathbf{z}\sim Q(\mathbf{z}|\mathbf{x}_i)}P_{\text{data}}\left(\mathbf{x}_i\right)\right)$$

$$= \frac{1}{N}\sum_{\mathbf{x}_i}\log\left(P_{\text{data}}\left(\mathbf{x}_i\right)\left(\frac{1}{N_s}\sum_{\mathbf{z}\sim Q(\mathbf{z}|\mathbf{x}_i)}\right)\right) = \frac{1}{N}\sum_{\mathbf{x}_i}\log P_{\text{data}}\left(\mathbf{x}_i\right)$$

In this ideal scenario we do not even have to sampling since Q is doing all the work of us

which is the maximal value of L!
Neural net **?can?** by chance learn something like this

The **final** derivation of the VAE loss function

- The original expression is hard to compute

$$\mathcal{L} \approx \frac{1}{N}\sum_{\mathbf{x}_i}\log\left(\frac{1}{N_s}\sum_{\mathbf{z}\sim Q(\mathbf{z}|\mathbf{x}_i)}\left[\frac{P_{\text{model}}\left(\mathbf{x}_i|\mathbf{z}\right)P\left(\mathbf{z}\right)}{Q(\mathbf{z}|\mathbf{x}_i)}\right]\right)$$

- The standard approach is to approximate the above with its lower bound **ELBO**

$$\mathcal{L} \geq \frac{1}{N}\sum_{\mathbf{x}_i}\frac{1}{N_s}\sum_{\mathbf{z}\sim Q(\mathbf{z}|\mathbf{x}_i)}\log\left[\frac{P_{\text{model}}\left(\mathbf{x}_i|\mathbf{z}\right)P\left(\mathbf{z}\right)}{Q(\mathbf{z}|\mathbf{x}_i)}\right]$$

This is a general property of concave functions called **Jensen's inequality**

$$\varphi\left(\mathbb{E}[X]\right) \leq \mathbb{E}\left[\varphi(X)\right]$$

Nice property, when: $Q(\mathbf{z}|\mathbf{x}_i) = \frac{P_{\text{model}}\left(\mathbf{x}_i|\mathbf{z}\right)P\left(\mathbf{z}\right)}{P_{\text{data}}\left(\mathbf{x}_i\right)}$ we recover accurate value of L (recall last slide)

$$= \frac{1}{N}\sum_{\mathbf{x}_i}\frac{1}{N_s}\sum_{\mathbf{z}\sim Q(\mathbf{z}|\mathbf{x}_i)}\log P_{\text{model}}\left(\mathbf{x}_i|\mathbf{z}\right) + \log P\left(\mathbf{z}\right) - \log Q(\mathbf{z}|\mathbf{x}_i)$$

$$= \frac{1}{N}\sum_{\mathbf{x}_i}\left\{\frac{1}{N_s}\sum_{\mathbf{z}\sim Q(\mathbf{z}|\mathbf{x}_i)}\log P_{\text{model}}\left(\mathbf{x}_i|\mathbf{z}\right)\right\} - \text{KL}\left(Q(\mathbf{z}|\mathbf{x}_i)||P\left(\mathbf{z}\right)\right)$$

The end!

The final derivation of the VAE loss function

- The final expression for the VAE loss is then

$$\mathcal{L}^{\text{VAE}} = \frac{1}{N}\sum_{\mathbf{x}_i}\left\{\frac{1}{N_s}\sum_{\mathbf{z}\sim Q(\mathbf{z}|\mathbf{x}_i)}\log P_{\text{model}}\left(\mathbf{x}_i|\mathbf{z}\right)\right\} - \text{KL}\left(Q(\mathbf{z}|\mathbf{x}_i)||P\left(\mathbf{z}\right)\right)$$

$$P_{\text{model}}\left(\mathbf{x}_i|\mathbf{z}\right) = \mathcal{N}\left(\mathbf{x}_i|g(\mathbf{z}), \sigma^2\mathbf{I}\right) = \frac{1}{2\pi\sigma^2}e^{-\frac{1}{2\sigma^2}\|\mathbf{x}_i - g(\mathbf{z})\|^2}$$

$$\mathcal{N}\left(\mathbf{z}|\mathbf{0}, \mathbf{I}\right)$$

- For simplicity the encoder network is assumed to be … another gaussian

$$Q(\mathbf{z}|\mathbf{x}_i) = \mathcal{N}\left(\mathbf{z}|\mu(\mathbf{x}_i;\vartheta), \Sigma(\mathbf{x}_i;\vartheta)\right)$$

are implemented by NN

is constrained to be diagonal matrix

- Since Q and P are Gaussians there is a closed formula for KL divergence:

$$\mathcal{D}[\mathcal{N}(\mu(X), \Sigma(X))\|\mathcal{N}(0, I)] = \frac{1}{2}\left(\text{tr}\left(\Sigma(X)\right) + \left(\mu(X)\right)^\top\left(\mu(X)\right) - k - \log\det\left(\Sigma(X)\right)\right).$$

spend a moment here to understand this

ref paper

The final derivation of the VAE loss function

- The final expression for the VAE loss is

$$\mathcal{L}^{\text{VAE}} = \frac{1}{N}\sum_{\mathbf{x}_i}\left\{\frac{1}{N_s}\sum_{\mathbf{z}\sim Q(\mathbf{z}|\mathbf{x}_i)}\log P_{\text{model}}\left(\mathbf{x}_i|\mathbf{z}\right)\right\} - \text{KL}\left(Q(\mathbf{z}|\mathbf{x}_i)||P\left(\mathbf{z}\right)\right)$$

$$P_{\text{model}}\left(\mathbf{x}_i|\mathbf{z}\right) = \mathcal{N}\left(\mathbf{x}_i|g(\mathbf{z}), \sigma^2\mathbf{I}\right) = \frac{1}{2\pi\sigma^2}e^{-\frac{1}{2\sigma^2}\|\mathbf{x}_i - g(\mathbf{z})\|^2}$$

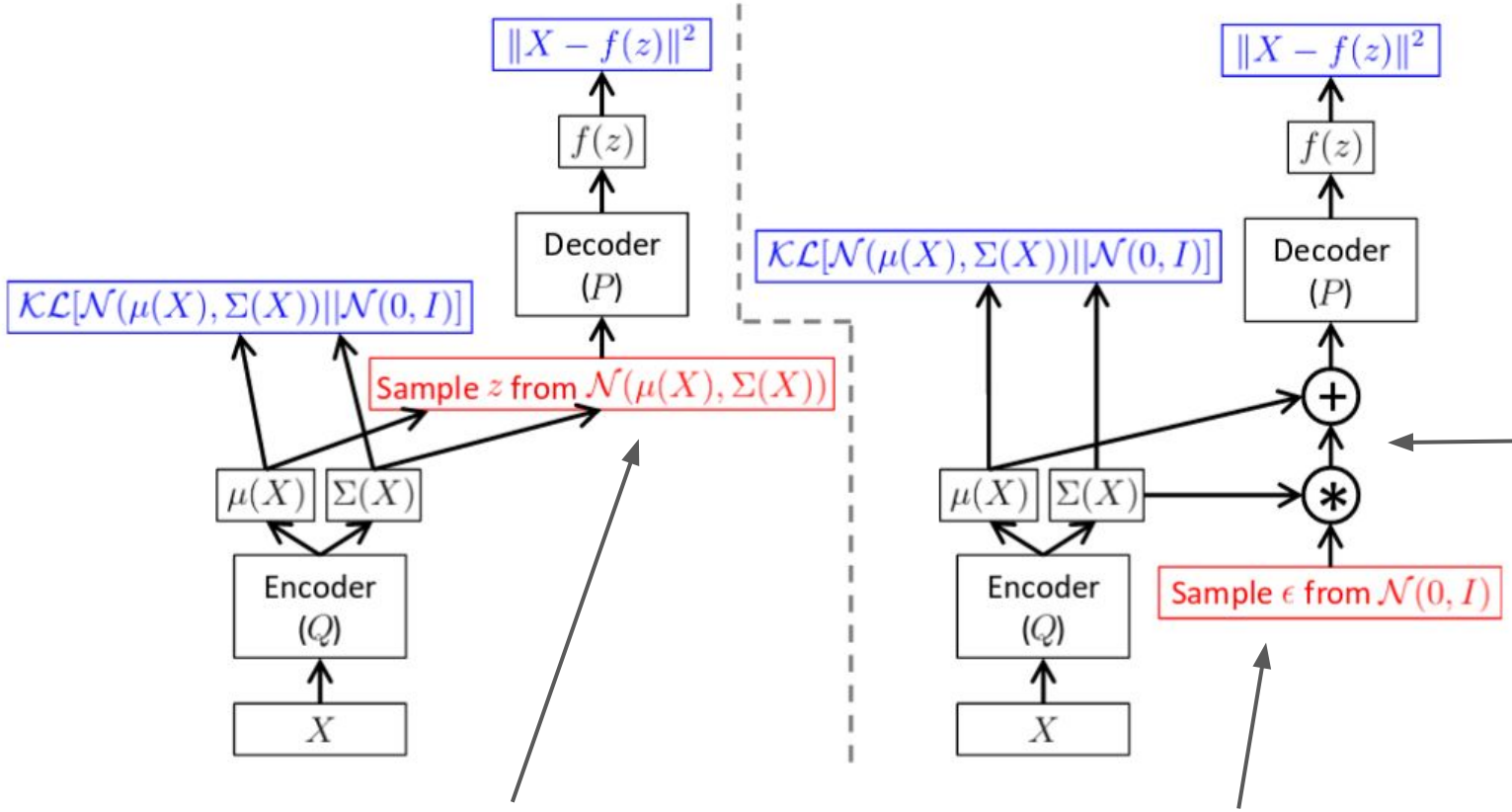
- Usually we set number of samples to 1...
- Measures the reconstruction error

$$\frac{1}{2}\left(\text{tr}\left(\Sigma(X)\right) + \left(\mu(X)\right)^\top\left(\mu(X)\right) - k - \log\det\left(\Sigma(X)\right)\right)$$

- This is correct when Q can be modelled with gaussian
- Can be interpreted as regularization to autoencoder

The VAE diagram and reparametrization trick

$$\mathcal{L}^{\text{VAE}} = \frac{1}{N}\sum_{\mathbf{x}_i}\left\{\frac{1}{N_s}\sum_{\mathbf{z}\sim Q(\mathbf{z}|\mathbf{x}_i)}\log P_{\text{model}}\left(\mathbf{x}_i|\mathbf{z}\right)\right\} - \text{KL}\left(Q(\mathbf{z}|\mathbf{x}_i)||P\left(\mathbf{z}\right)\right)$$



- Sample random number

$$\epsilon \sim \mathcal{N}(0, I)$$

$$\mathbf{z} = \mu(\mathbf{x}_i; \vartheta) + \epsilon \cdot \Sigma(\mathbf{x}_i; \vartheta)$$

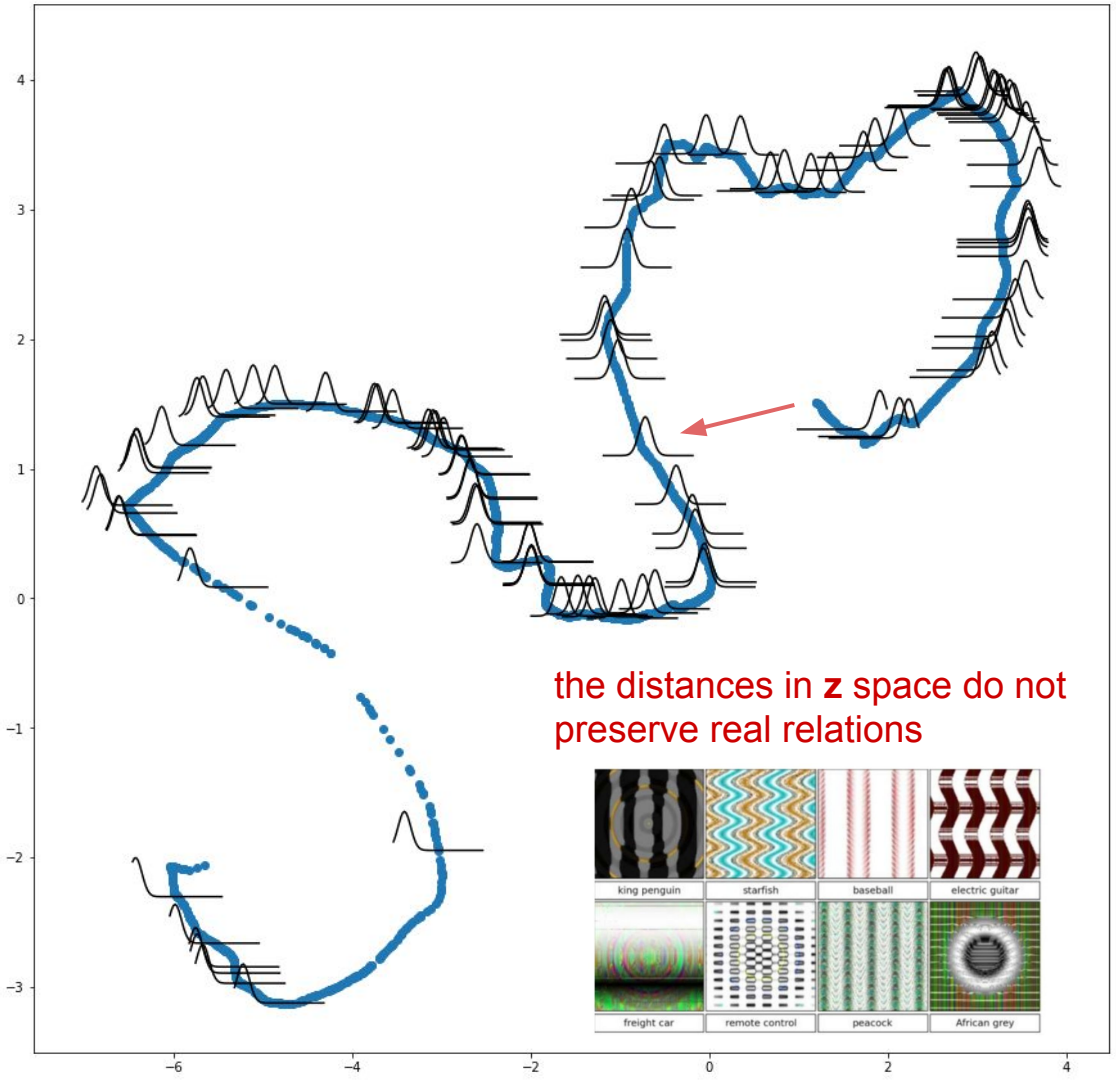
reparametrization trick

Cannot compute gradients here: stochastic layer

Provide stochastic input, then the output of the layer is deterministic and gradient can be computed

Remember this?

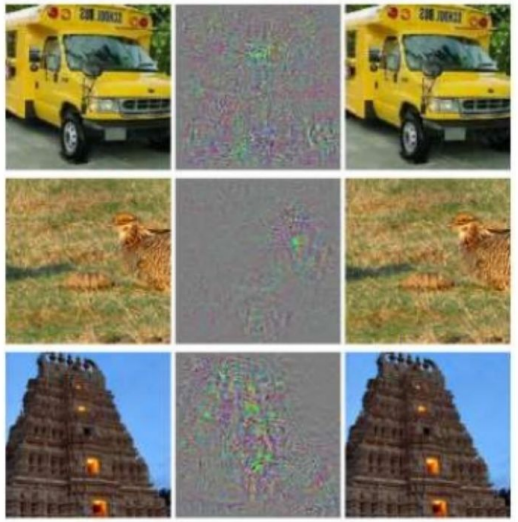
Let’s plot the manifold for AE!



- AE are not “**learning efficient representation of our data**”!!!, they just cheat to minimize the loss.
- The pattern is completely random (depends on initial conditions)
- **Just** avoid simple autoencoders like this

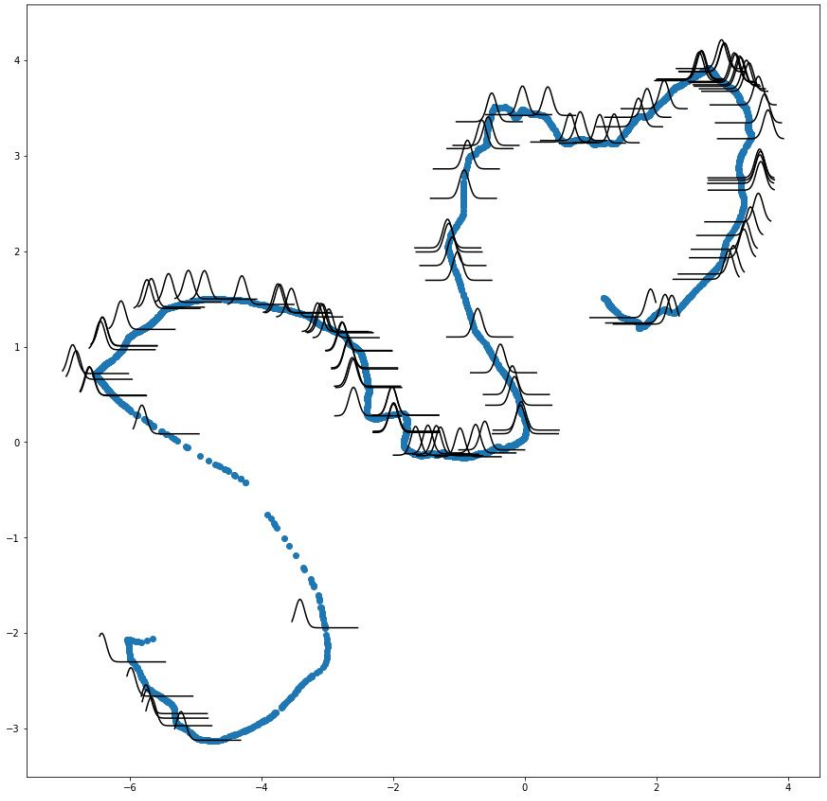
What are the consequences of such behaviour???

Adversarial examples

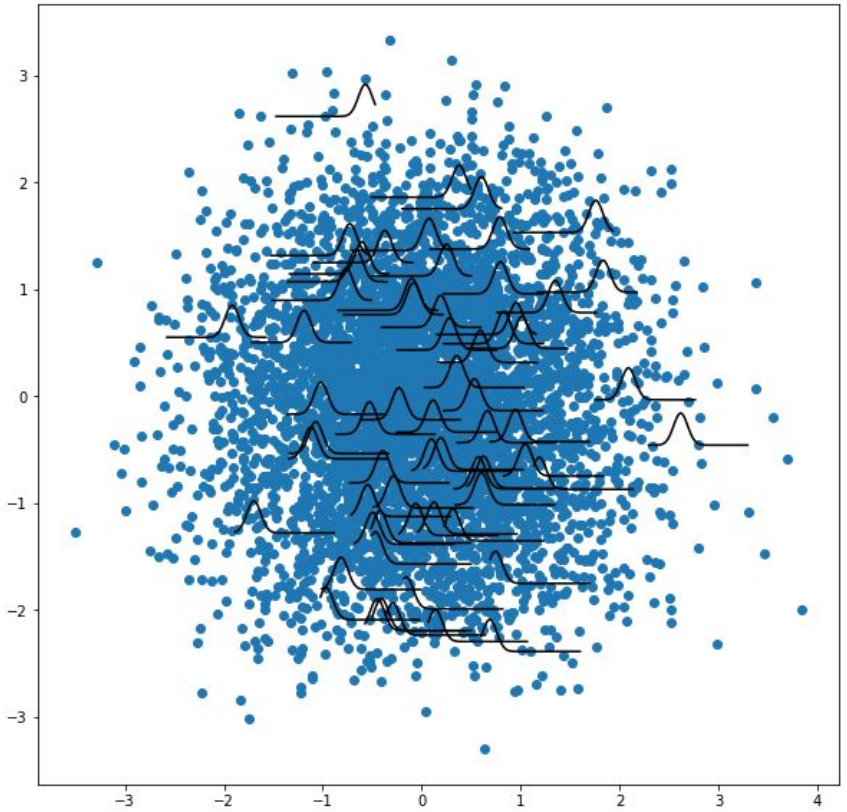


Remember this?

AE latent space



VAE latent space



Note, that here the concept of one dimensionality of the problem is well captured

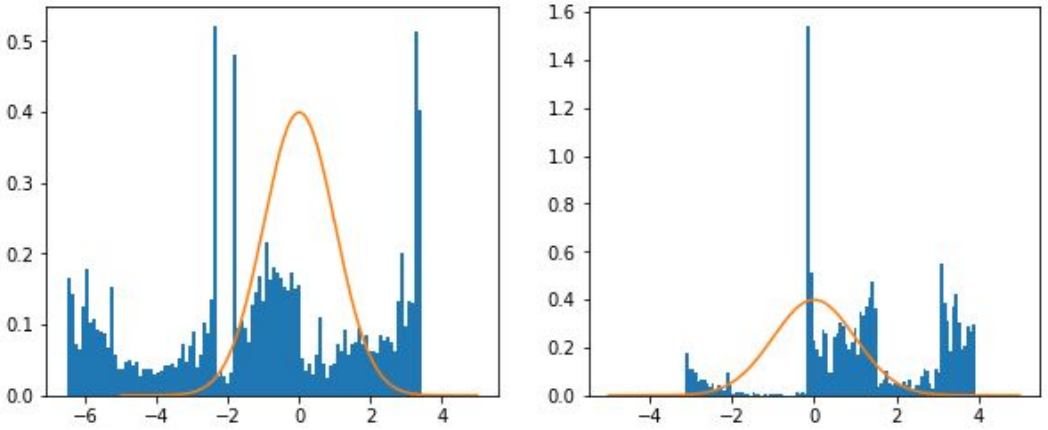
$$Q(\mathbf{z}|\mathbf{x}_i) = \mathcal{N}\left(\mathbf{z}|\mu(\mathbf{x}_i;\vartheta), \Sigma(\mathbf{x}_i;\vartheta)\right)$$

We wanted Q to be gaussian N(0,1)

VAE vs AE

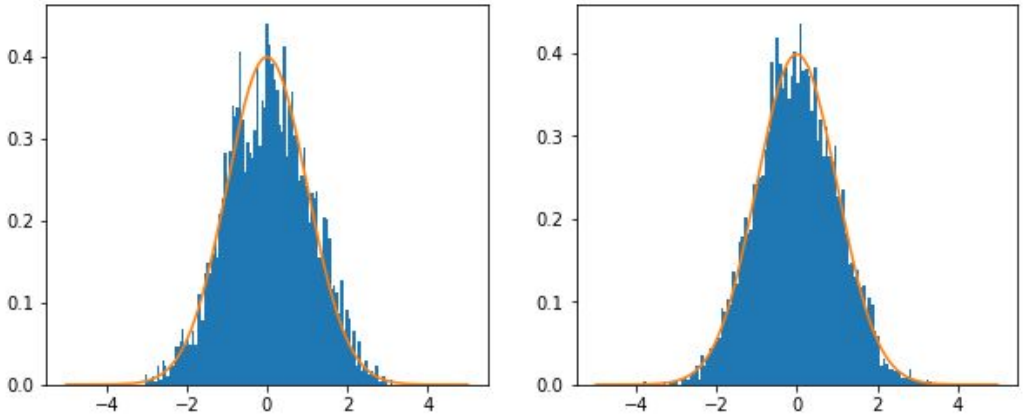
$$Q(\mathbf{z}|\mathbf{x}_i) = \mathcal{N}\left(\mathbf{z}|\mu(\mathbf{x}_i;\vartheta), \Sigma(\mathbf{x}_i;\vartheta)\right)$$

AE latent space histograms



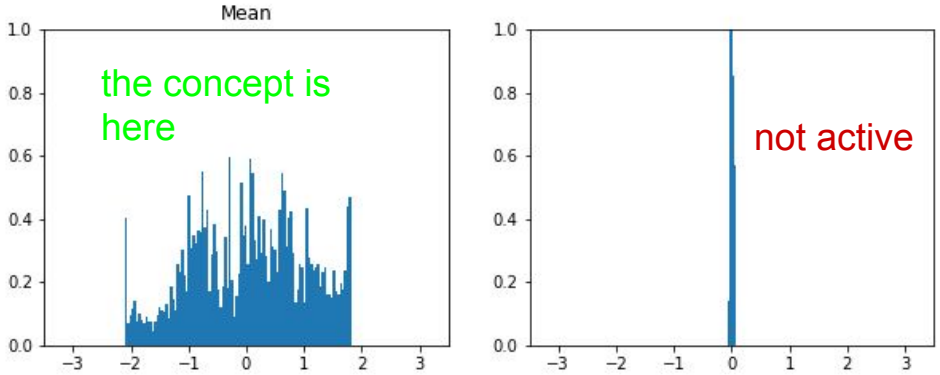
z_1 z_2

VAE latent space histogram

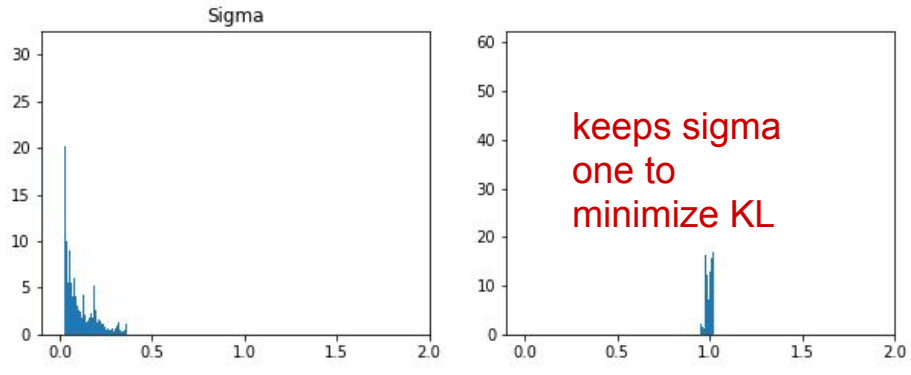


z_1 z_2

Let's take a look into Mean and Sigma outputs activations

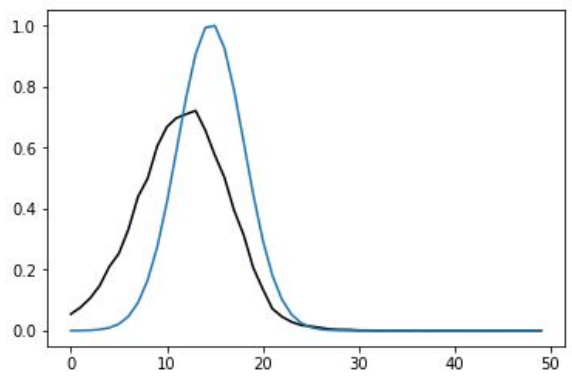


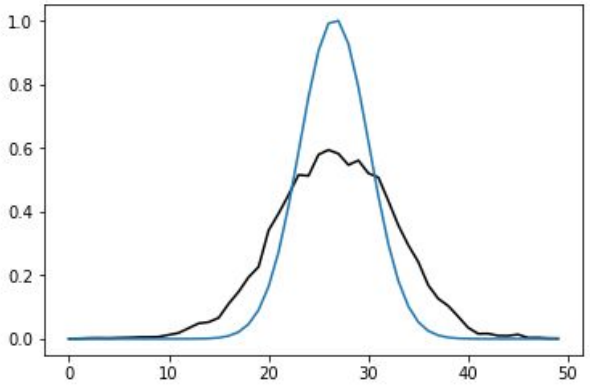
μ_1 μ_2

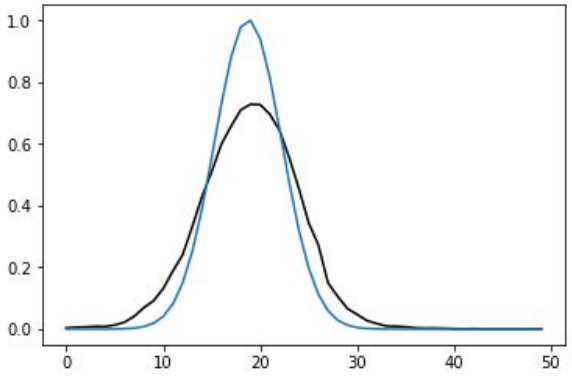


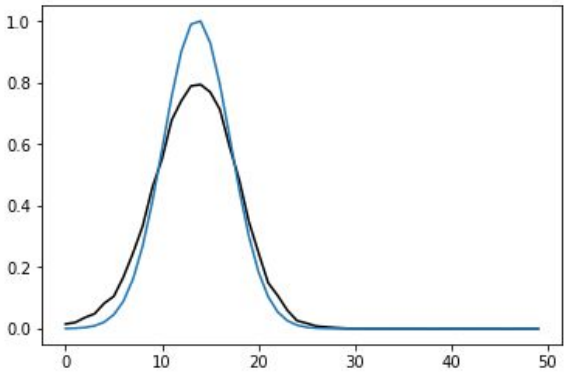
Σ_1 Σ_2

The effect of the KL regularization



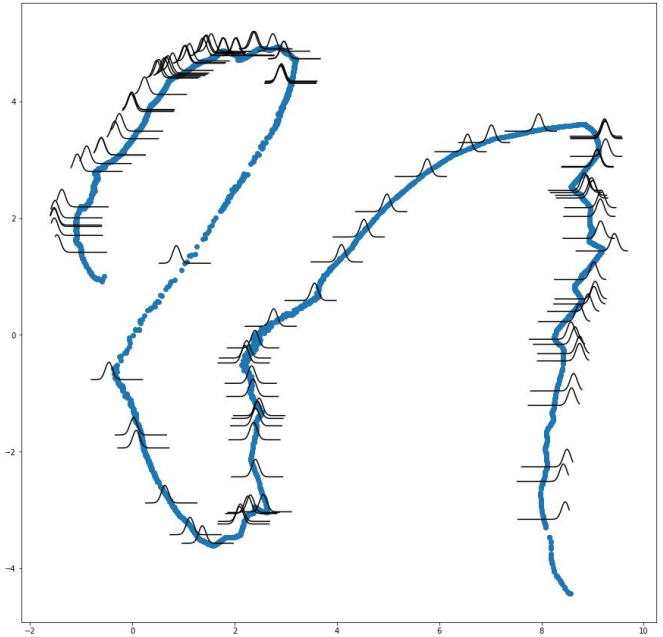






$$\mathcal{L}^{\mathrm{VAE}} = \frac{1}{N}\sum_{\mathbf{x}_i}\left\{\frac{1}{N_s}\sum_{\mathbf{z}\sim Q(\mathbf{z}|\mathbf{x}_i)}\log P_{\mathrm{model}}\left(\mathbf{x}_i|\mathbf{z}\right)\right\} - \mathrm{KL}\left(Q(\mathbf{z}|\mathbf{x}_i)||P\left(\mathbf{z}\right)\right)$$

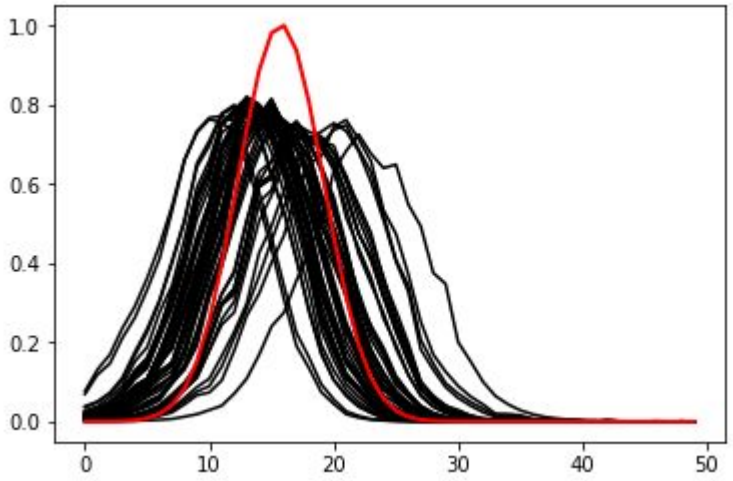
- If the KL term will be too strong then the model will try to make Q = N(0,1)
- We have competition between reconstruction quality and proper distribution of activations
- On the other hand if this term is too weak model will converge to simple AE (with sigma=0)



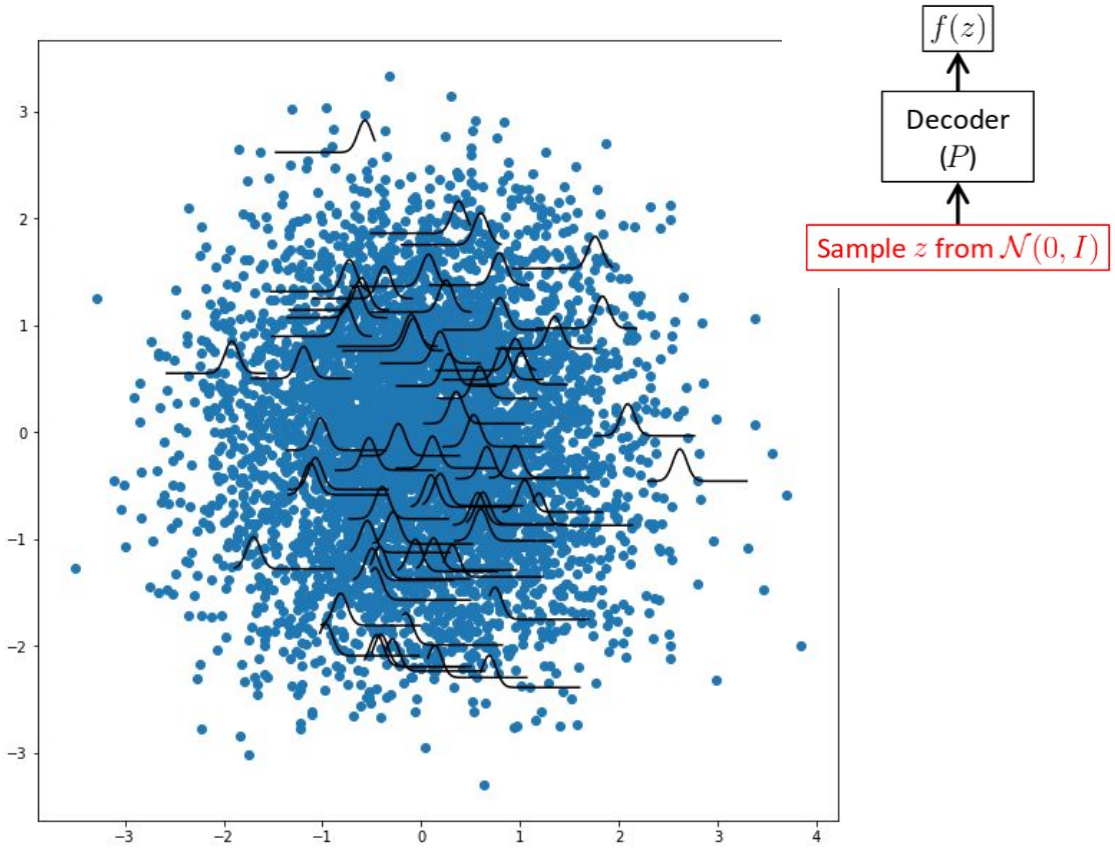
VAE as a generative model

- Q is a simple Gaussian, hence we can easily sample from it to generate similar samples to input one

$$Q(\mathbf{z}|\mathbf{x}_i) = \mathcal{N}\left(\mathbf{z}|\mu(\mathbf{x}_i;\vartheta), \Sigma(\mathbf{x}_i;\vartheta)\right)$$

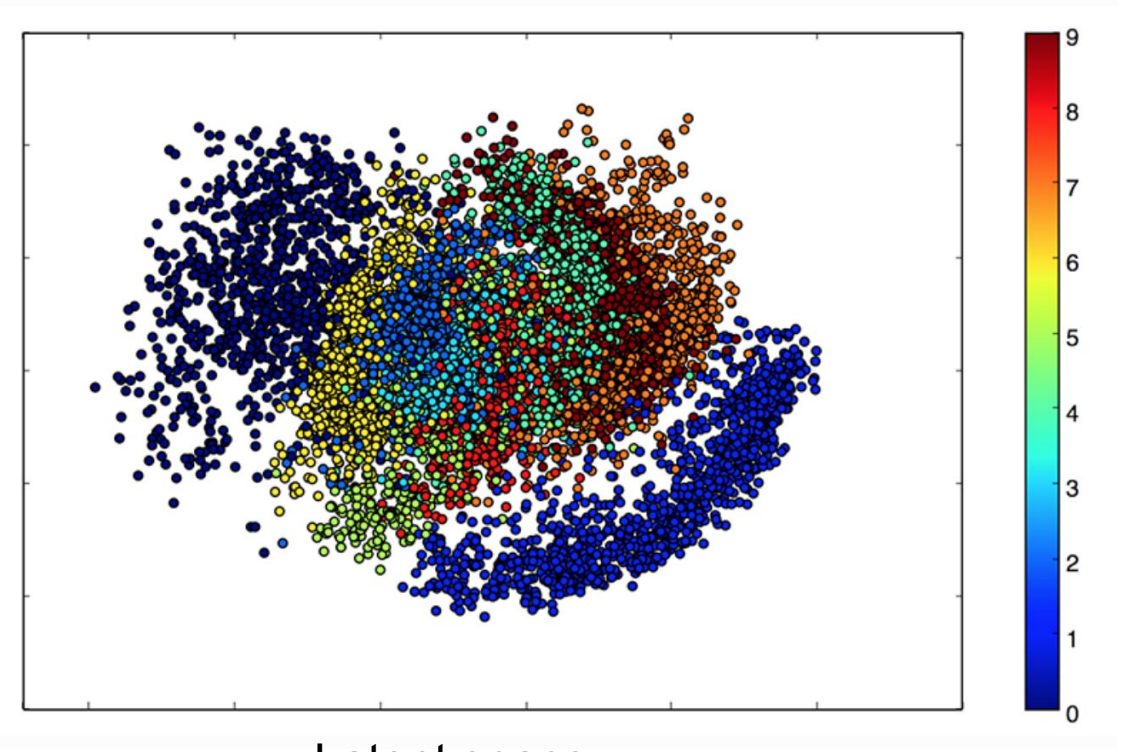


- Or we can directly sample from N(0,1)

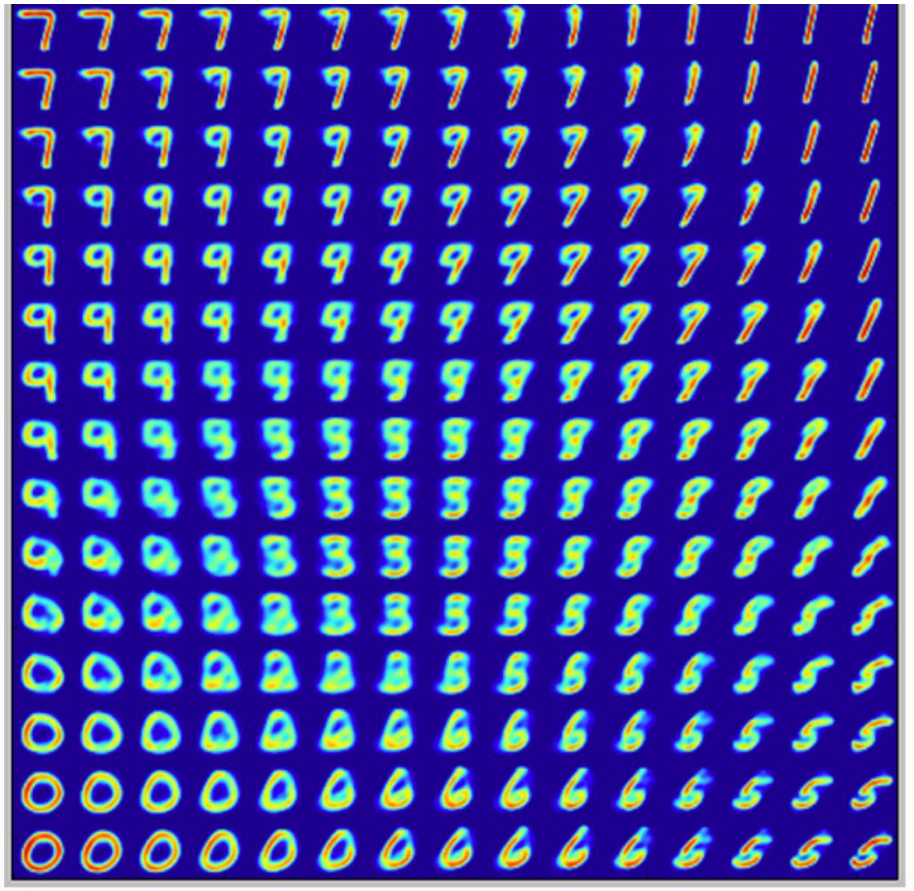


[Demo: Digit Fantasies by a Deep Generative Model]

VAE as a generative model - MNIST example



Latent space



Generated images

Summary

$P_{\text{data}}(\mathbf{x})$

$P_{\text{model}}(\mathbf{x})$

variational inference

$\text{KL}\left(P_{\text{data}}(\mathbf{x}), P_{\text{model}}(\mathbf{x})\right)$

minimize the distance w.r.t params

$\log P_{\text{model}}(\mathbf{X};\theta)$

maximize

$$\frac{1}{N}\sum_{\mathbf{x}_i \sim P_{\text{data}}(\mathbf{x})} \log\left(\int \mathbf{dz} P_{\text{model}}(\mathbf{x}_i|\mathbf{z})\, P(\mathbf{z})\right)$$

latent variable model

$$\frac{1}{N}\sum_{\mathbf{x}_i} \log\left(\frac{1}{N_s}\sum_{\mathbf{z}\sim Q(\mathbf{z}|\mathbf{x}_i)}\left[\frac{P_{\text{model}}(\mathbf{x}_i|\mathbf{z})\, P(\mathbf{z})}{Q(\mathbf{z}|\mathbf{x}_i)}\right]\right)$$

sampling from encoder distribution instead of P(z)

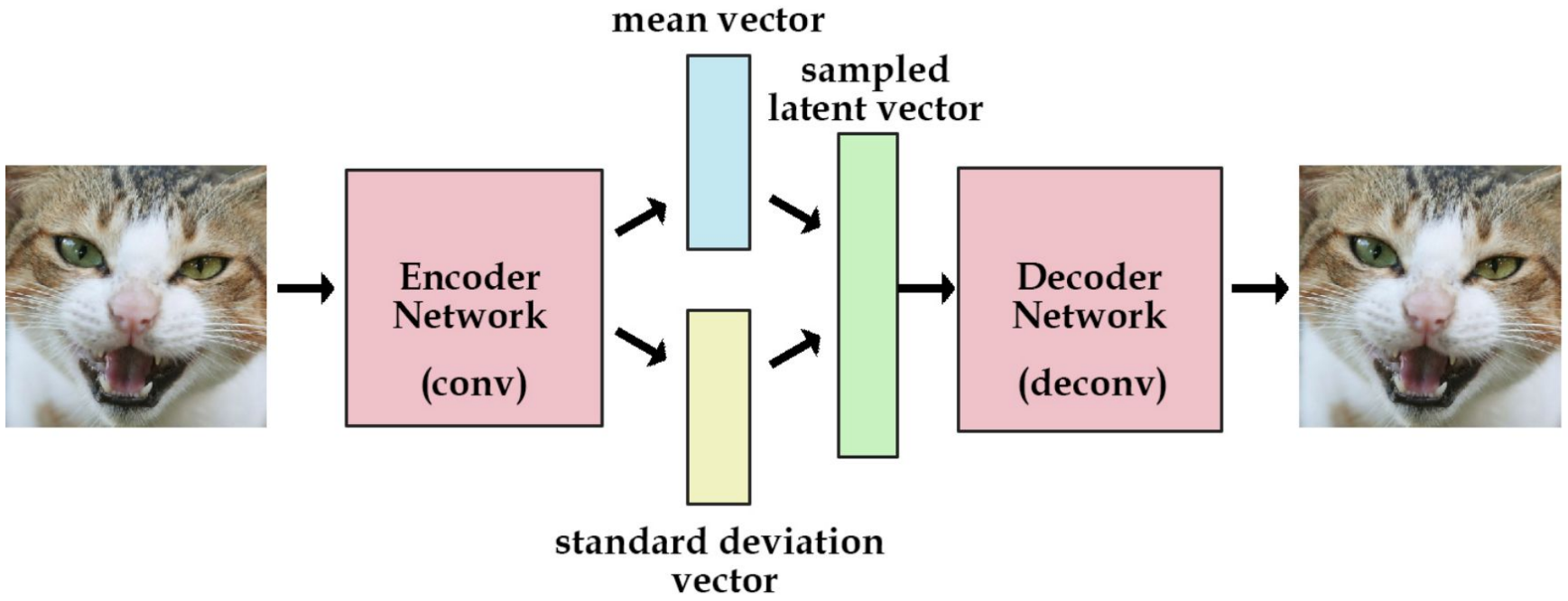
$$\mathcal{L} \geq \frac{1}{N}\sum_{\mathbf{x}_i}\frac{1}{N_s}\sum_{\mathbf{z}\sim Q(\mathbf{z}|\mathbf{x}_i)} \log\left[\frac{P_{\text{model}}(\mathbf{x}_i|\mathbf{z})\, P(\mathbf{z})}{Q(\mathbf{z}|\mathbf{x}_i)}\right]$$

Jensen's inequality

VAE loss

$$\mathcal{L}^{\text{VAE}} = \frac{1}{N}\sum_{\mathbf{x}_i}\left\{\frac{1}{N_s}\sum_{\mathbf{z}\sim Q(\mathbf{z}|\mathbf{x}_i)} \log P_{\text{model}}(\mathbf{x}_i|\mathbf{z})\right\} - \text{KL}\left(Q(\mathbf{z}|\mathbf{x}_i)||P(\mathbf{z})\right)$$

Discussion and overview of recent results



Are there some questions?

Conditional VAEs

Add additional information/labels to your model to force conditional behaviour of the model

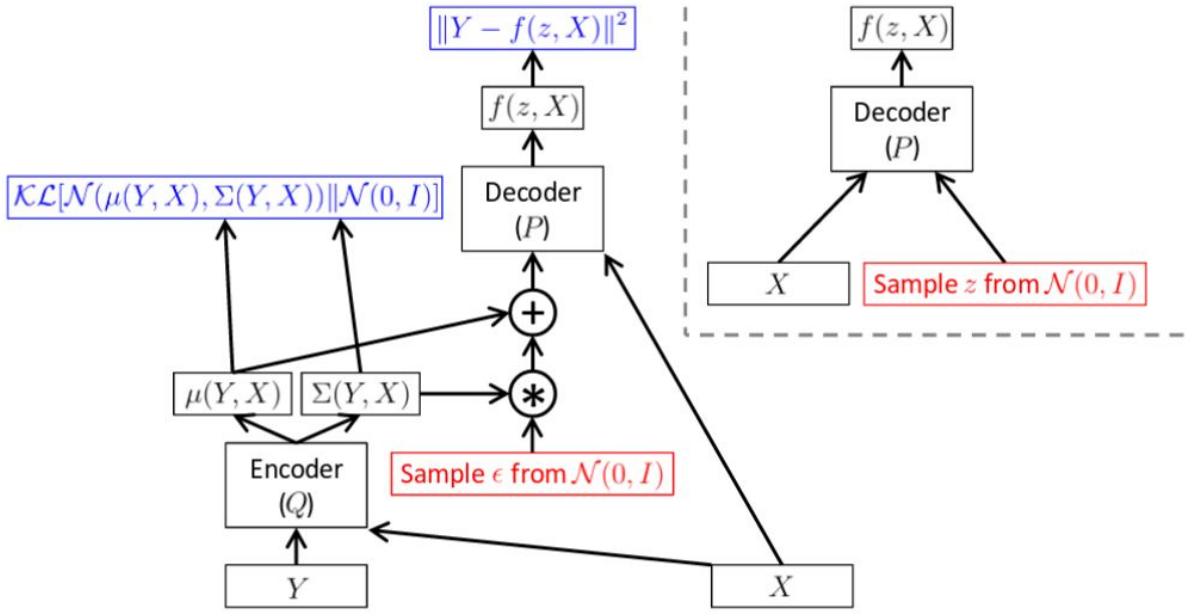
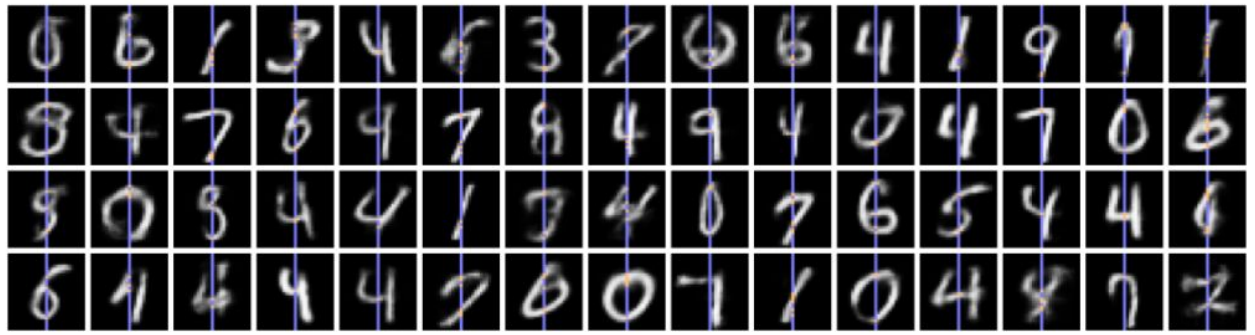


Diagram of CVAE

Some potential applications:

- hole filling
- class dependent object generation



(a) CVAE

[reference]

$$\log P(Y|X) - \mathcal{D}\left[Q(z|Y,X) \| P(z|Y,X)\right] = E_{z\sim Q(\cdot|Y,X)}\left[\log P(Y|z,X)\right] - \mathcal{D}\left[Q(z|Y,X) \| P(z|X)\right]$$

Importance weighted autoencoders (Nov 2016)

Sorry for the change in notation

- The VAE loss for one example **x**

$$log(p(x)) \geq E_{z_1 \ldots z_k \sim q(z|x)} \left[\frac{1}{k} \sum_{i=1}^{k} log \left(\frac{p(x, z_i)}{q(z_i|x)} \right) \right] = L_{VAE}[q].$$

arxiv:1704.02916

in our notation

$$\left\{ \frac{1}{N_s} \sum_{\mathbf{z} \sim Q(\mathbf{z}|\mathbf{x}_i)} \log P_{\text{model}} \left(\mathbf{x}_i | \mathbf{z} \right) \right\} - \text{KL} \left(Q(\mathbf{z}|\mathbf{x}_i) || P(\mathbf{z}) \right)$$

- Instead approximate the loss from direct expression:

$$log(p(x)) \geq E_{z_1 \ldots z_k \sim q(z|x)} \left[log \left(\frac{1}{k} \sum_{i=1}^{k} \frac{p(x, z_i)}{q(z_i|x)} \right) \right] = L_{IWAE}[q]$$

we obtained VAE by simplifying it using Jensen's inequality

- The difference is visible once we compute gradients

$$\nabla_\Theta \mathcal{L}_{VAE}[q] = E_{z_1 \ldots z_k \sim q(z|x)} \left[\sum_{i=1}^{k} \frac{1}{k} \nabla_\Theta log \left(\frac{p(x, z_i)}{q(z_i|x)} \right) \right]$$

$$\nabla_\Theta \mathcal{L}_{IWAE}[q] = E_{z_1 \ldots z_k \sim q(z|x)} \left[\sum_{i=1}^{k} \tilde{w}_i \nabla_\Theta log \left(\frac{p(x, z_i)}{q(z_i|x)} \right) \right]$$

VAE is evenly weighted but IWAE uses importance of each sample

$$\tilde{w}_i = \frac{\frac{p(x,z_i)}{q(z_i|x)}}{\sum_{j=1}^{k} \frac{p(x,z_j)}{q(z_j|x)}}$$

Importance weighted autoencoders (Nov 2016)

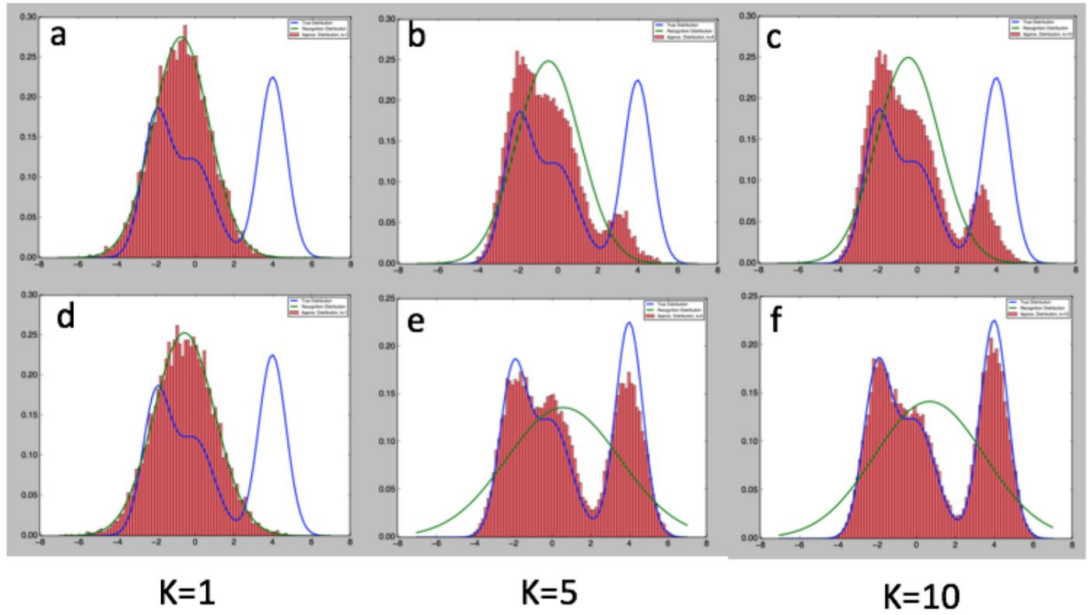
Comparison between VAE and IWAE

$$\nabla_\Theta \mathcal{L}_{VAE}[q] = E_{z_1 \dots z_k \sim q(z|x)} \left[\sum_{i=1}^{k} \frac{1}{k} \nabla_\Theta log \left(\frac{p(x, z_i)}{q(z_i|x)} \right) \right]$$

SVI

$$\nabla_\Theta \mathcal{L}_{IWAE}[q] = E_{z_1 \dots z_k \sim q(z|x)} \left[\sum_{i=1}^{k} \tilde{w}_i \nabla_\Theta log \left(\frac{p(x, z_i)}{q(z_i|x)} \right) \right]$$

W-SVI



blue true posterior, red reconstructions

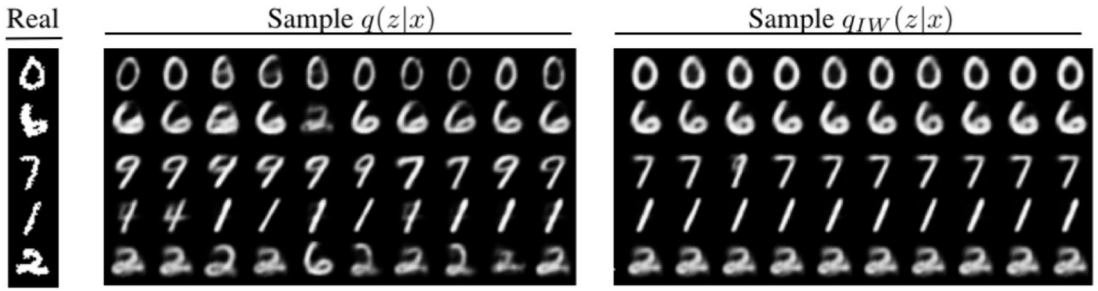


Figure 3: Reconstructions of MNIST samples from $q(z|x)$ and q_{IW}. The model was trained by maximizing the IWAE ELBO with K=50 and 2 latent dimensions. The reconstructions from $q(z|x)$ are greatly improved with the sampling-resampling step of q_{IW}.

A nice thing to have:

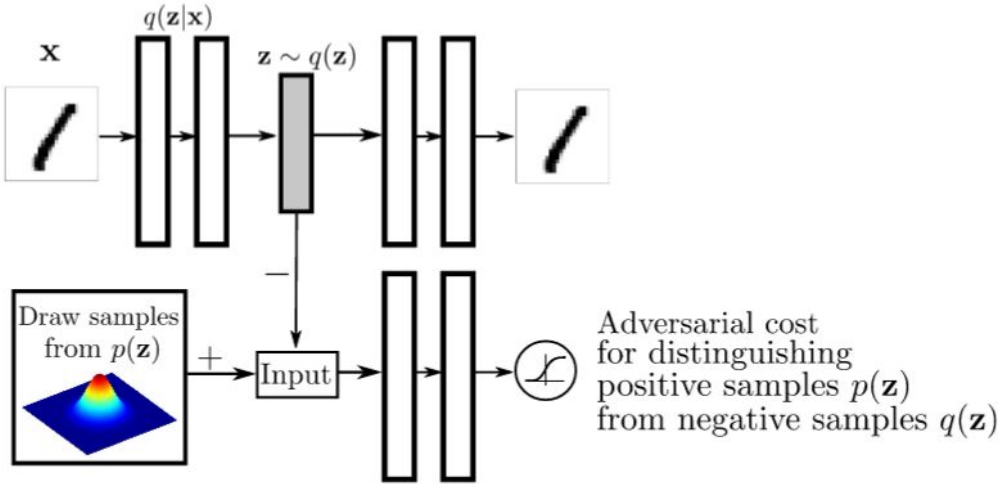
Theorem 1. *For all k, the lower bounds satisfy*

$$\log p(\mathbf{x}) \geq \mathcal{L}_{k+1} \geq \mathcal{L}_k.$$

Moreover, if $p(\mathbf{h}, \mathbf{x})/q(\mathbf{h}|\mathbf{x})$ is bounded, then $\mathcal{L}_k$ approaches $\log p(\mathbf{x})$ as k goes to infinity.

Adversarial Autoencoders (May 2016)

Combine VAE and GANs



"The bottom row diagrams a second network trained to discriminatively predict whether a sample arises from the hidden code of the autoencoder or from a sampled distribution specified by the user"

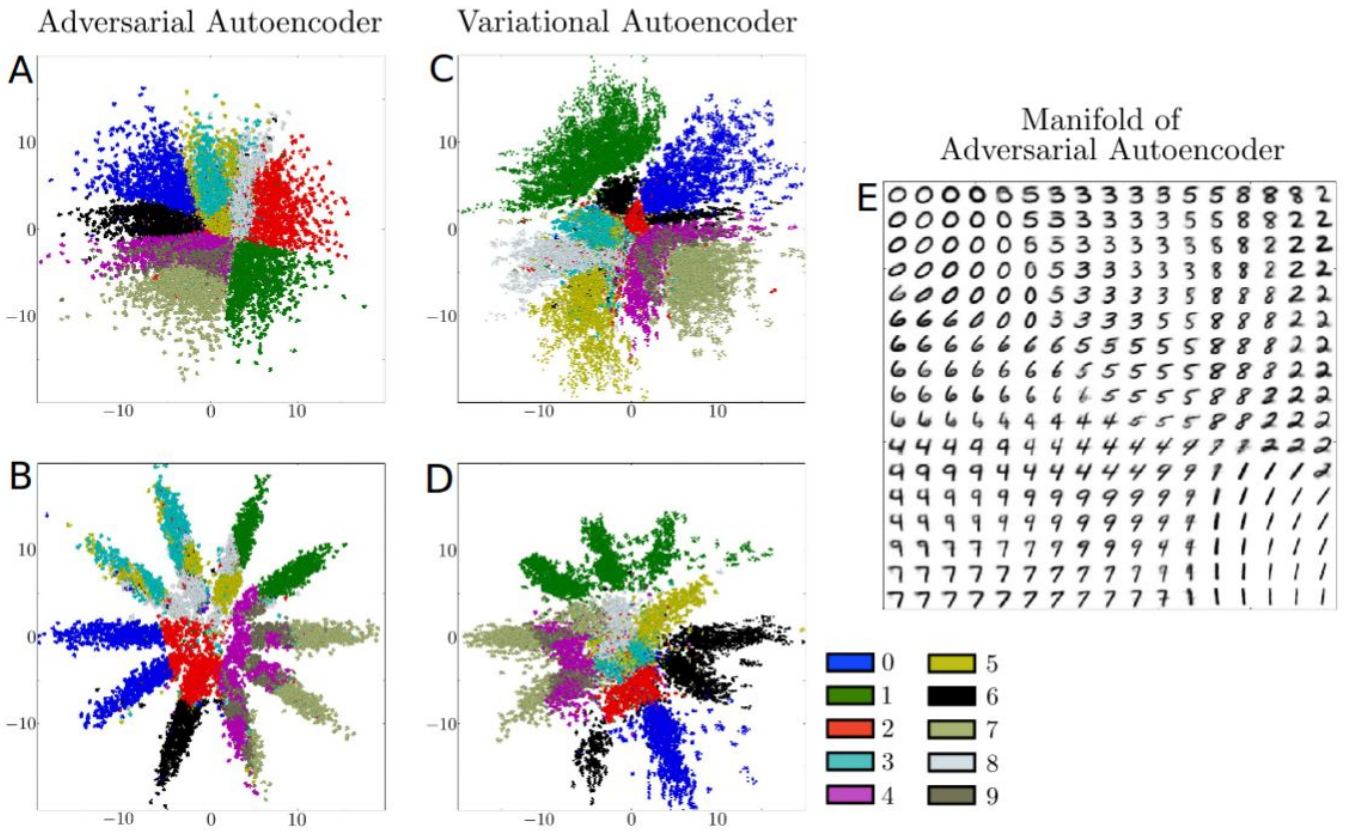
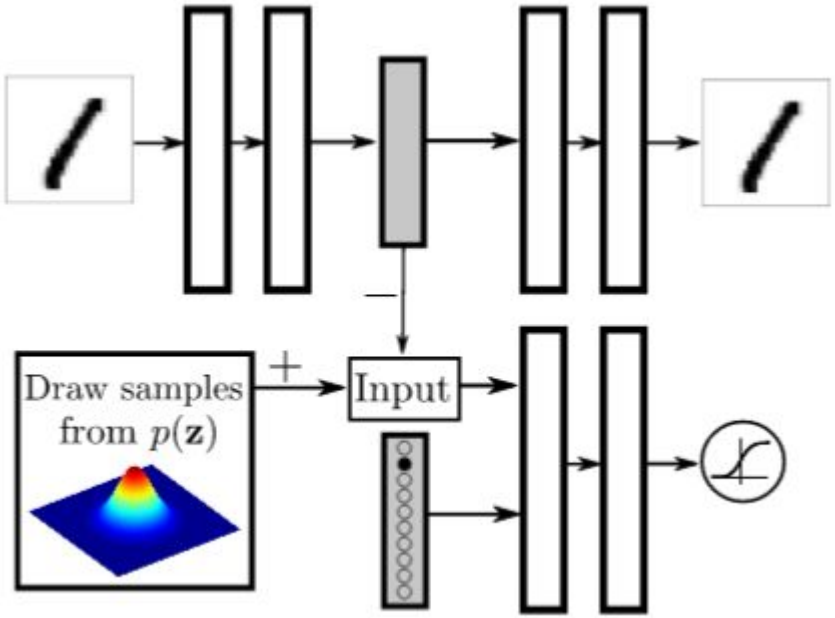


Figure 2: Comparison of adversarial and variational autoencoder on MNIST. The hidden code $\mathbf{z}$ of the *hold-out* images for an adversarial autoencoder fit to (a) a 2-D Gaussian and (b) a mixture of 10 2-D Gaussians. Each color represents the associated label. Same for variational autoencoder with (c) a 2-D gaussian and (d) a mixture of 10 2-D Gaussians. (e) Images generated by uniformly sampling the Gaussian percentiles along each hidden code dimension $\mathbf{z}$ in the 2-D Gaussian adversarial autoencoder.

Lack of technical details in the paper

Adversarial Autoencoders (May 2016)

Same as above but add labels into encoder



"Regularizing the hidden code by providing a one-hot vector to the discriminative network. The one-hot vector has an extra label for training points with unknown classes."

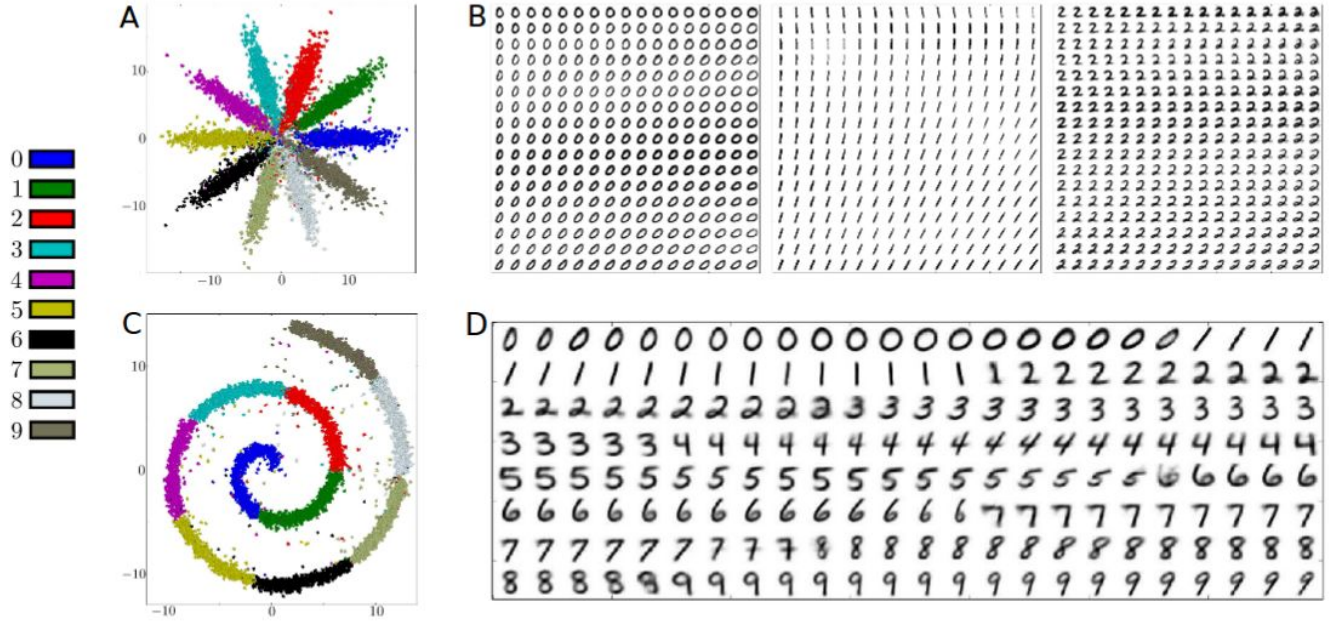
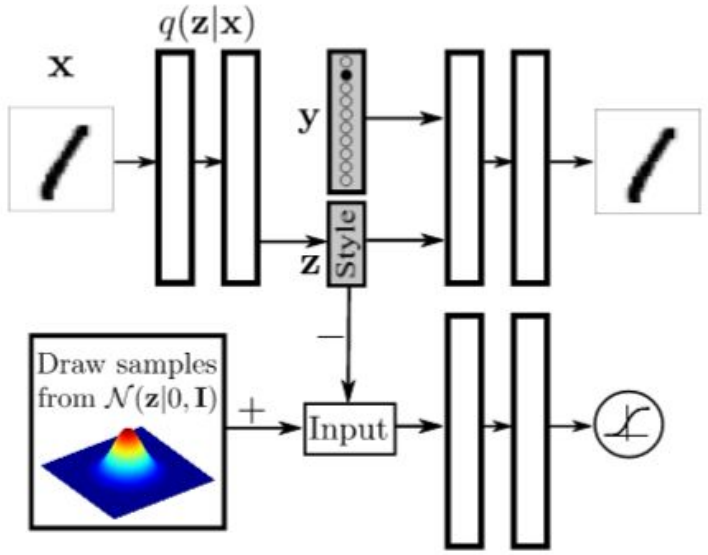


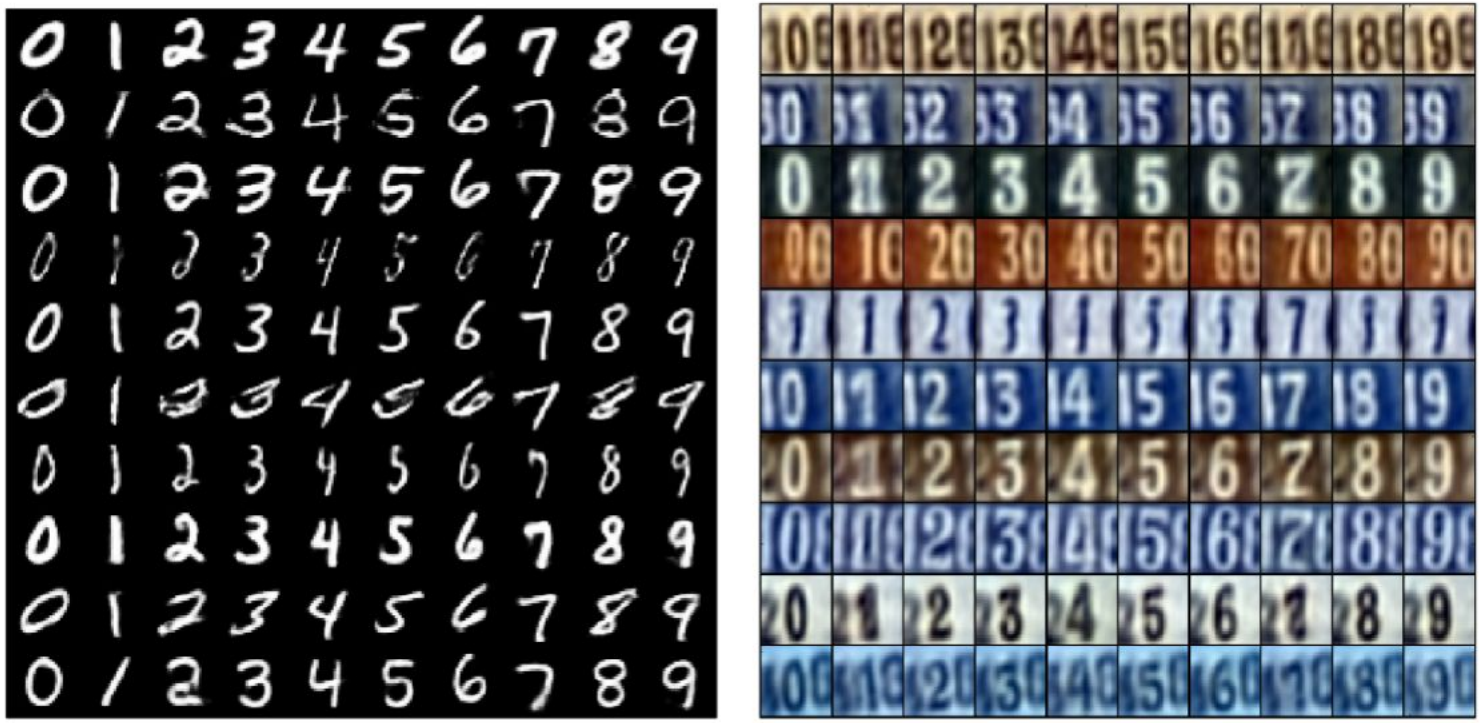
Figure 4: Leveraging label information to better regularize the hidden code. **Top Row:** Training the coding space to match a mixture of 10 2-D Gaussians: (a) Coding space $\mathbf{z}$ of the *hold-out* images. (b) The manifold of the first 3 mixture components: each panel includes images generated by uniformly sampling the Gaussian percentiles along the axes of the corresponding mixture component. **Bottom Row:** Same but for a swiss roll distribution (see text). Note that labels are mapped in a numeric order (i.e., the first 10% of swiss roll is assigned to digit 0 and so on): (c) Coding space $\mathbf{z}$ of the *hold-out* images. (d) Samples generated by walking along the main swiss roll axis.

Adversarial Autoencoders (May 2016)

Disentangling style from content



"Disentangling the label information from the hidden code by providing the one-hot vector to the generative model. The hidden code in this case learns to represent the style of the image."



(a) MNIST (b) SVHN

Figure 7: Disentangling content and style (15-D Gaussian) on MNIST and SVHN datasets.

Adversarial Autoencoders (May 2016)

Dimensionality Reduction with Adversarial Autoencoder

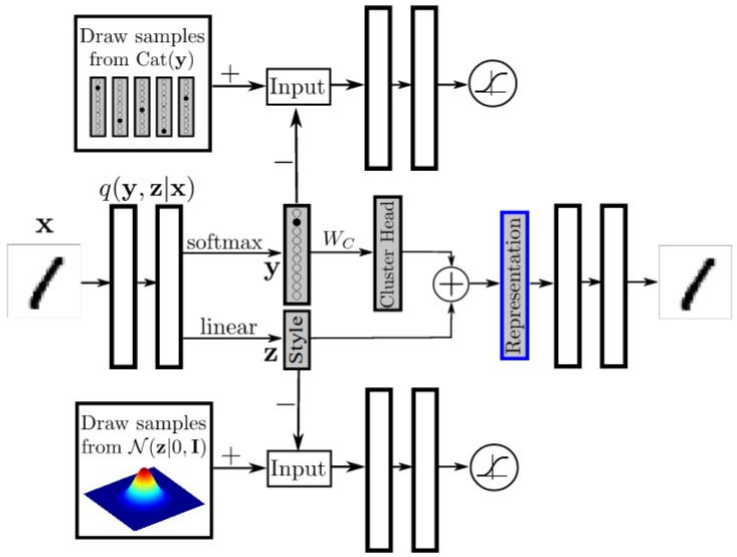
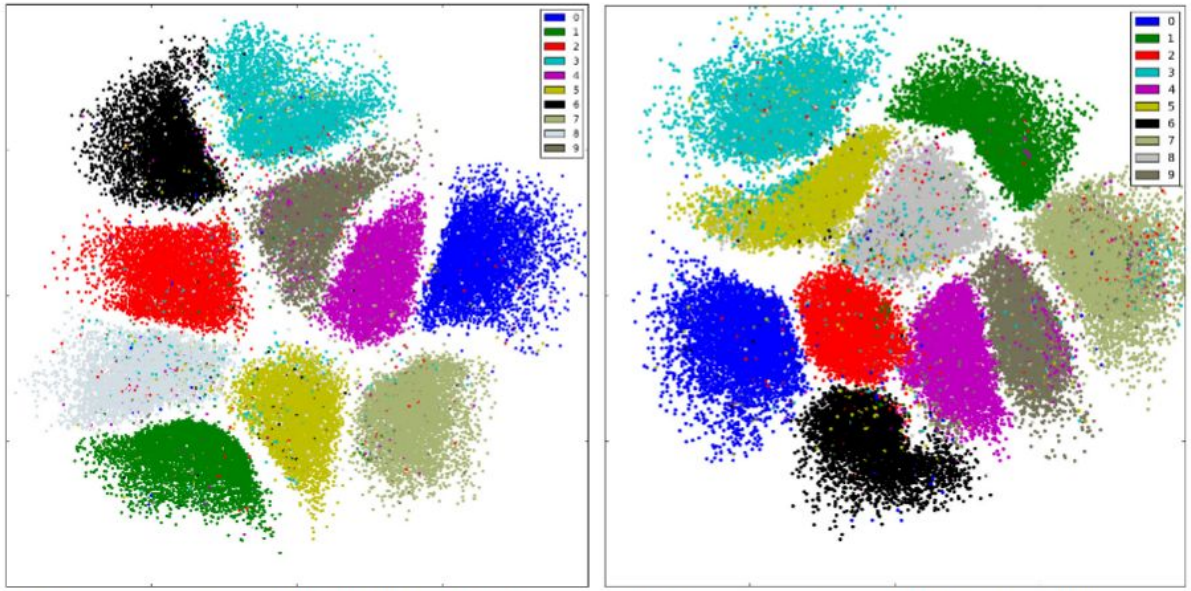


Figure 10: Dimensionality reduction with adversarial autoencoders: There are two separate adversarial networks that impose Categorical and Gaussian distribution on the latent representation. The final n dimensional representation is constructed by first mapping the one-hot label representation to an n dimensional cluster head representation and then adding the result to an n dimensional style representation. The cluster heads are learned by SGD with an additional cost function that penalizes the Euclidean distance between of every two of them.



(a) 2D representation with 1000 labels (4.20% error) (b) 2D representation with 100 labels (6.08% error)

They found Batch Normalization to be crucial in AAE clustering experiments.

Discriminative Regularization for Generative Models

Images generated from the VAE (and most other generative frameworks) diverge from natural images in two distinct ways:

- **Missing high frequency information**: caused mainly by L2 regularization (problem of comparison between translated images)
- **Missing semantic information**: generative models of natural images often lack a clear sense of “objectness”.

Recall our VAE loss

$$\mathcal{L}^{\text{VAE}} = \frac{1}{N}\sum_{\mathbf{x}_i}\left\{\frac{1}{N_s}\sum_{\mathbf{z}\sim Q(\mathbf{z}|\mathbf{x}_i)}\log P_{\text{model}}\left(\mathbf{x}_i|\mathbf{z}\right)\right\} - \text{KL}\left(Q(\mathbf{z}|\mathbf{x}_i)||P\left(\mathbf{z}\right)\right)$$

$$P_{\text{model}}\left(\mathbf{x}_i|\mathbf{z}\right) = \mathcal{N}\left(\mathbf{x}_i|g(\mathbf{z}), \sigma^2\mathbf{I}\right) = \frac{1}{2\pi\sigma^2}e^{-\frac{1}{2\sigma^2}\|\mathbf{x}_i - g(\mathbf{z})\|^2}$$

Discriminative Regularization for Generative Models

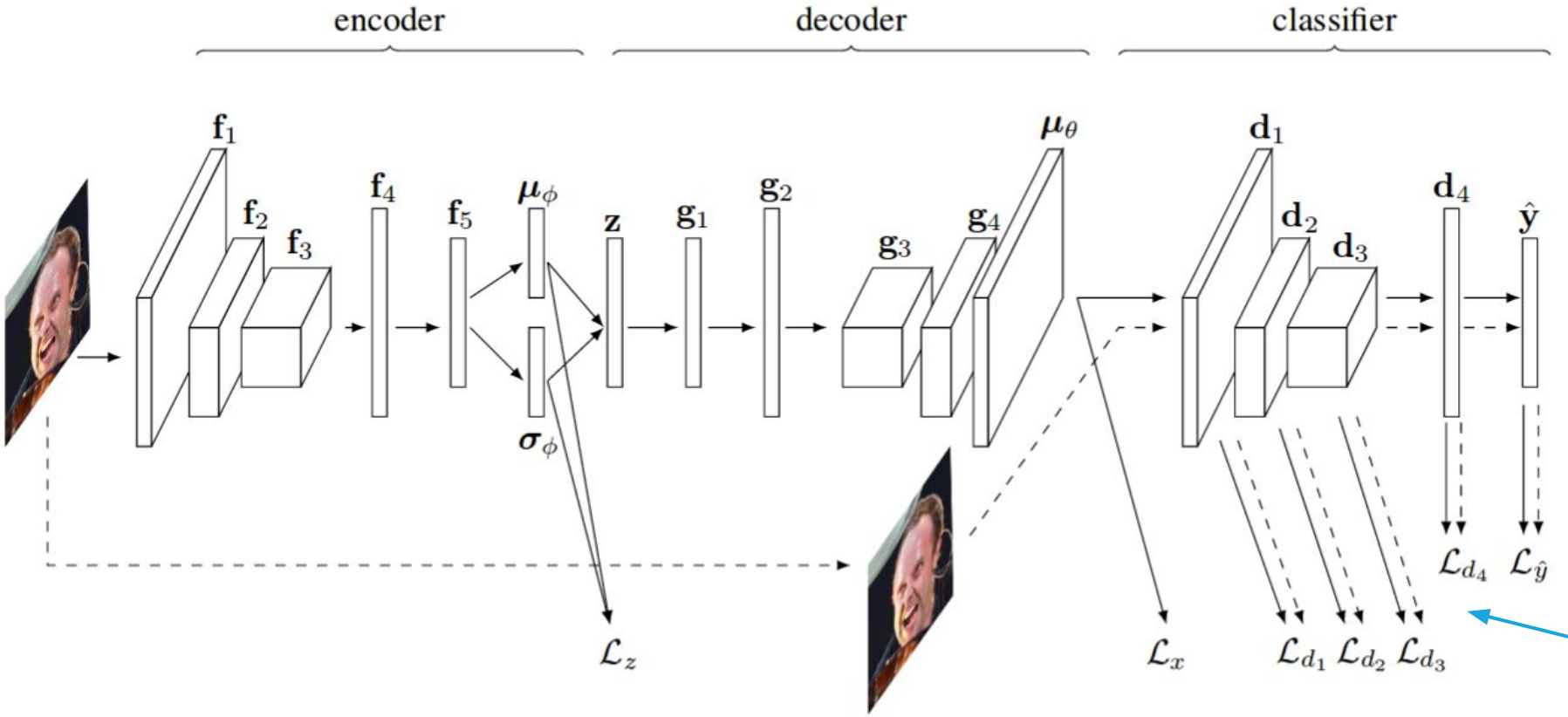
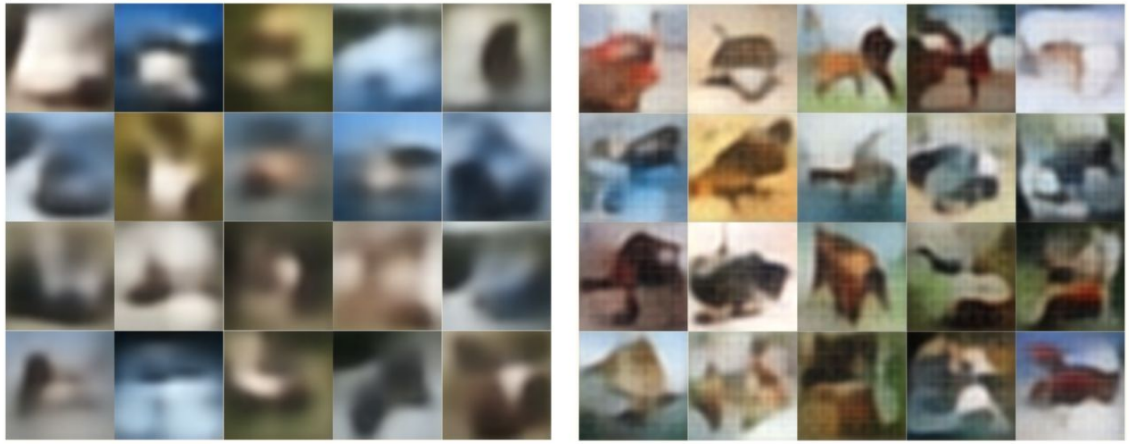


Figure 1. The discriminative regularization model. Layers $\boldsymbol{f}_1$, $\boldsymbol{f}_2$, $\boldsymbol{f}_3$, $\boldsymbol{d}_1$, $\boldsymbol{d}_2$ and $\boldsymbol{d}_3$ represent convolutional layers, whereas layers $\boldsymbol{g}_3$, $\boldsymbol{g}_4$ and $\boldsymbol{\mu}_\theta$ represent fractionally strided convolutional layers.

match the reconstructed image within representation space

Discriminative Regularization for Generative Models



(a) Samples without discriminative regularization (b) Samples with discriminative regularization

Figure 2. CIFAR samples generated from variational autoencoders trained with and without the discriminative regularization. The architecture and the hyperparameters (except those directly related to discriminative regularization) are the same for both models. Our baseline VAE samples are similar in visual fidelity to other results in the literature (Mansimov et al., 2015). Discriminative regularization often does a good job of producing coherent objects, but the textures are usually muddled and the samples lack local detail

Figure 3. SVHN samples with the standard variational autoencoders (left), real images (center), and samples using discriminative regularization (right). The discriminative regularizer improves the clarity and visual fidelity of the samples. SVHN is the only dataset where we did not observe unnatural patterning when using discriminative regularization.

Autoencoding beyond pixels using a learned similarity metric (2016)

Combine GANs with VAEs - similar idea (Feb 2016)

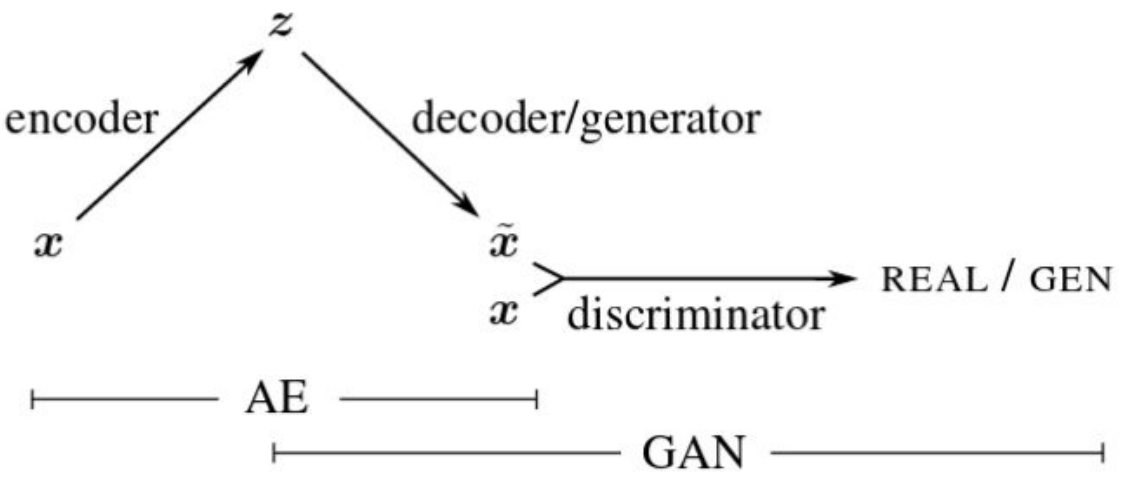
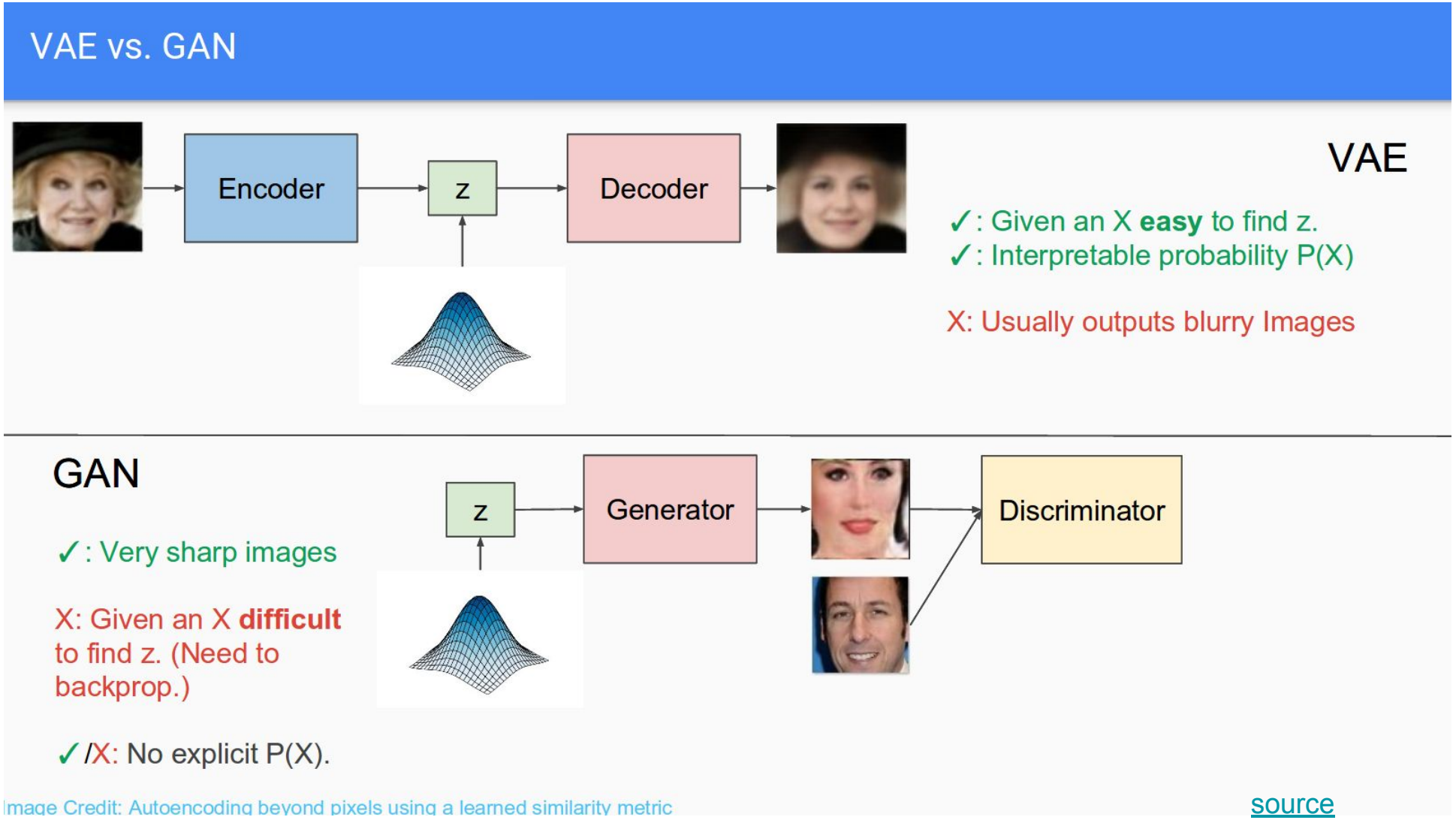


Figure 1. Overview of our network. We combine a VAE with a GAN by collapsing the decoder and the generator into one.

reconstruction objective. Thereby, we replace element-wise errors with feature-wise errors to better capture the data distribution while offering invariance towards e.g. translation. We apply our method to images of faces and show that it outperforms VAEs with element-wise similarity measures in terms of visual fidelity. Moreover,

Autoencoding beyond pixels using a learned similarity metric (2016)

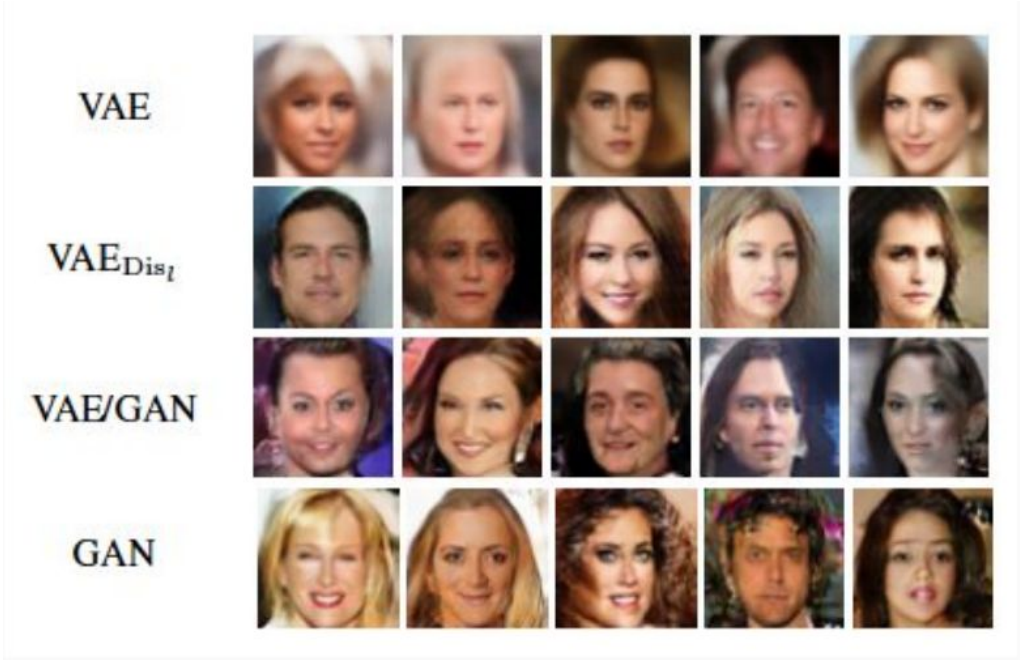
Combine GANs with VAEs - similar idea (Feb 2016)

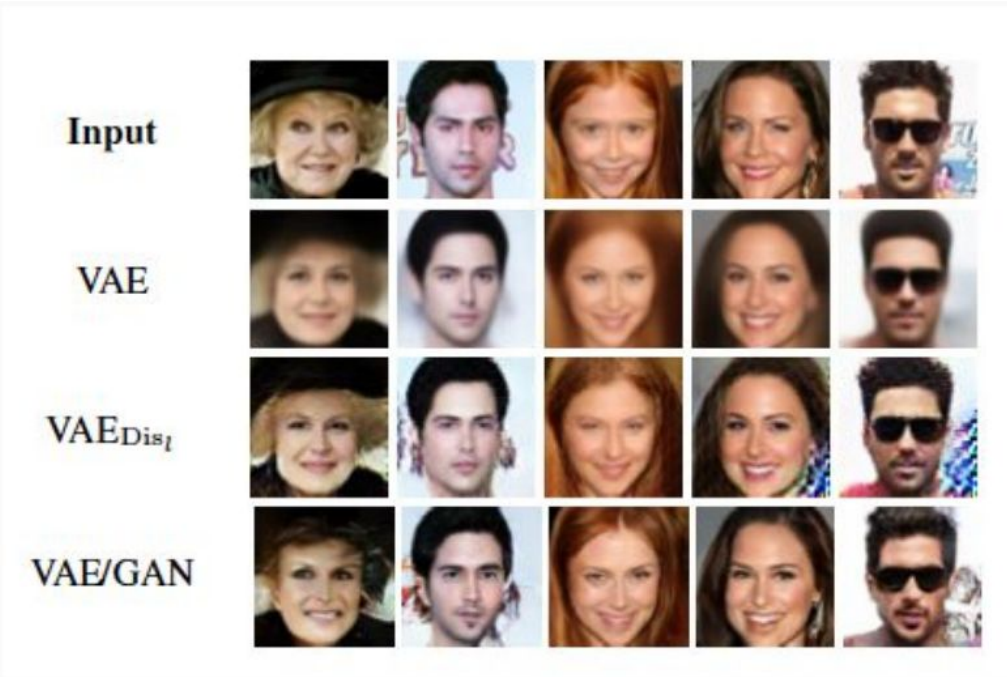


source

Autoencoding beyond pixels using a learned similarity metric (2016)

Combine GANs with VAEs - similar idea (Feb 2016)





VAE_{Dis_l} : Train a GAN first, then use the discriminator of GAN to train a VAE.

VAE/GAN: GAN and VAE trained together.

source

Semantic Facial Expression Editing using Autoencoded Flow (Nov 2016)

Figure 1: Illustration of the facial expression manipulation task. **Left:** Source image. **Right:** Squint expression synthesized automatically by our method.

- use VAEs to encode the picture of face into low dimensional latent space
- standard method leads to blurred images
- instead use methods like **patched matching** - reconstruct image from existing part of images
- combine CNN with optical flow - don't reconstruct RGB pixels but how they move
- A Flow VAE

$$\mathcal{L} = \mathcal{L}_{reconstruct} + \lambda_1 \mathcal{L}_{prior} + \lambda_2 \mathcal{L}_{flow}$$

For more technical details see paper

Semantic Facial Expression Editing using Autoencoded Flow (Nov 2016)

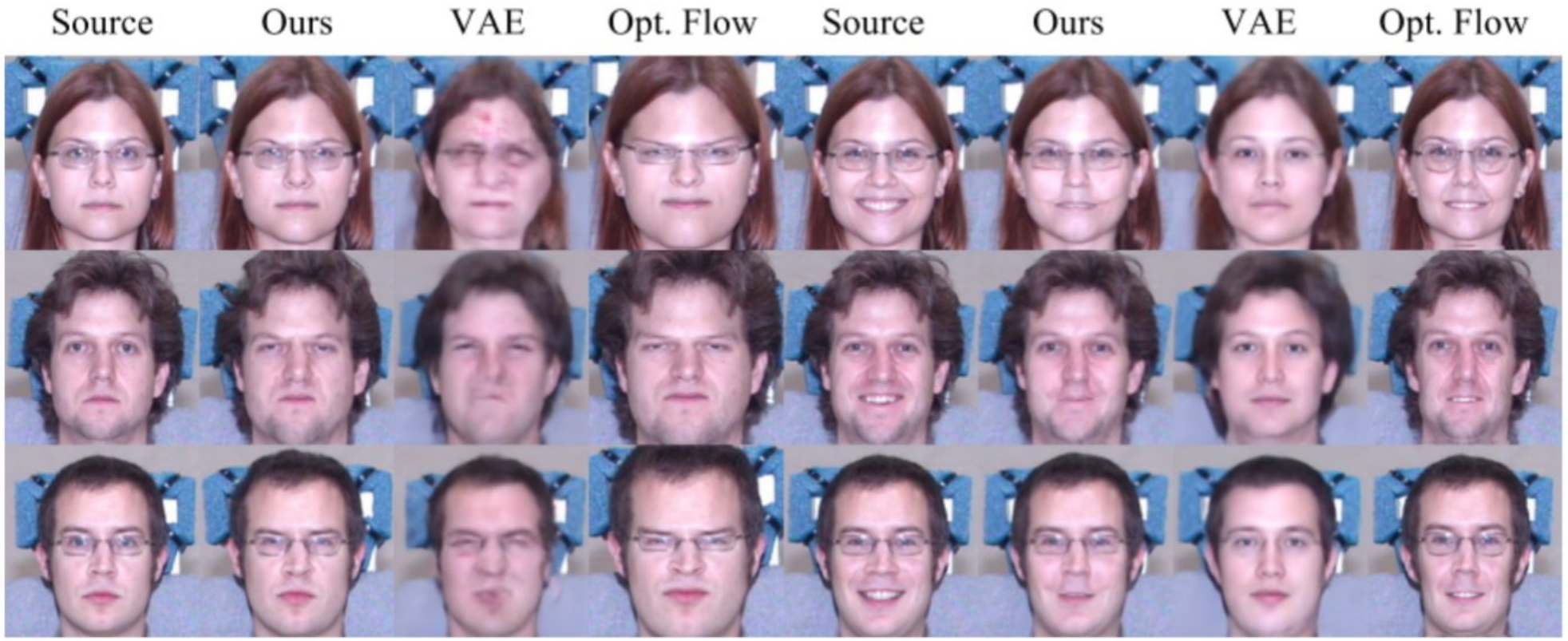


Figure 4: Comparison of synthesized expressions. **Left block:** Synthesis from a neutral expression source image to expression of disgust. **Right block:** Synthesis from a smile expression source image to a neutral expression. Our method demonstrates better low-level image quality and more realistic expressions compared to the baselines.

Semantic Facial Expression Editing using Autoencoded Flow (Nov 2016)

Figure 5: Magnify and suppress the facial expression. **Left**: Synthesized suppressed expression. **Center**: Original image.[2] **Right**: Synthesized magnified image.

Questions?

References

- http://slazebni.cs.illinois.edu/spring17/lec12_vae.pdf - well detailed presentation about VAEs, highly recommended
- http://kvfrans.com/variational-autoencoders-explained/ - some blog post, but nothing special. Maybe it is worth to take a look on it, since it may provide some intuitions
- https://arxiv.org/abs/1606.05908 - Tutorial about VAEs probably most complete
- https://jaan.io/what-is-variational-autoencoder-vae-tutorial/ - interesting Bayesian interpretation of VAEs - a blog post
- http://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/ - another blog post, well written with math and keras implementation
- http://www.cs.toronto.edu/~duvenaud/courses/csc2541/ - online course about variational inference

References

- http://www.cs.toronto.edu/~duvenaud/courses/csc2541/slides/lec2-vae.pdf - page 11 - a short derivation of ELBO variational loss
- https://arxiv.org/pdf/1509.00519.pdf - paper about importance sampling in VAEs
- https://arxiv.org/pdf/1704.02916.pdf - a workshop material about importance sampling, really nice interpretation of the above paper - highly recommended
- http://www.cs.toronto.edu/~duvenaud/courses/csc2541/slides/structured-encoders-decoders.pdf - page 16 - VAE problems, page 28 - importance sampling, 44 - the role of the sampling - this is actually summary of IWAE paper

References

- http://www.cs.toronto.edu/~duvenaud/courses/csc2541/slides/structured-encoders-decoders.pdf - DRAW method - not discussed during presentation, VAE plus attention
- https://www.slideshare.net/ShaiHarel/variational-autoencoder-talk - a nice motivation for VAEs in terms of adversarial examples
- http://www.dpkingma.com/sgvb_mnist_demo/demo.html - animations with reconstructions
- http://dpkingma.com/?page_id=393 - a complete list of different implementations of VAEs
- http://blog.shakirm.com/2015/03/a-statistical-view-of-deep-learning-ii-auto-encoders-and-free-energy/ - not covered during presentation - the relation between autoencoders and free energy
- http://orbit.dtu.dk/files/121765928/1602.02282.pdf - not covered in the presentation - a paper about different experiments how to train VAEs - with BN and Relus - probably not rocket science

References

- https://arxiv.org/pdf/1602.03220.pdf - paper about discrimintative VAE with classifier at the end of the network
- http://edwardlib.org/tutorials/klqp - Edward tutorial about ELBO derivation
- https://arxiv.org/pdf/1511.05644.pdf - adversarial autoencoder

FORNAX

WWW.FORNAX.AI