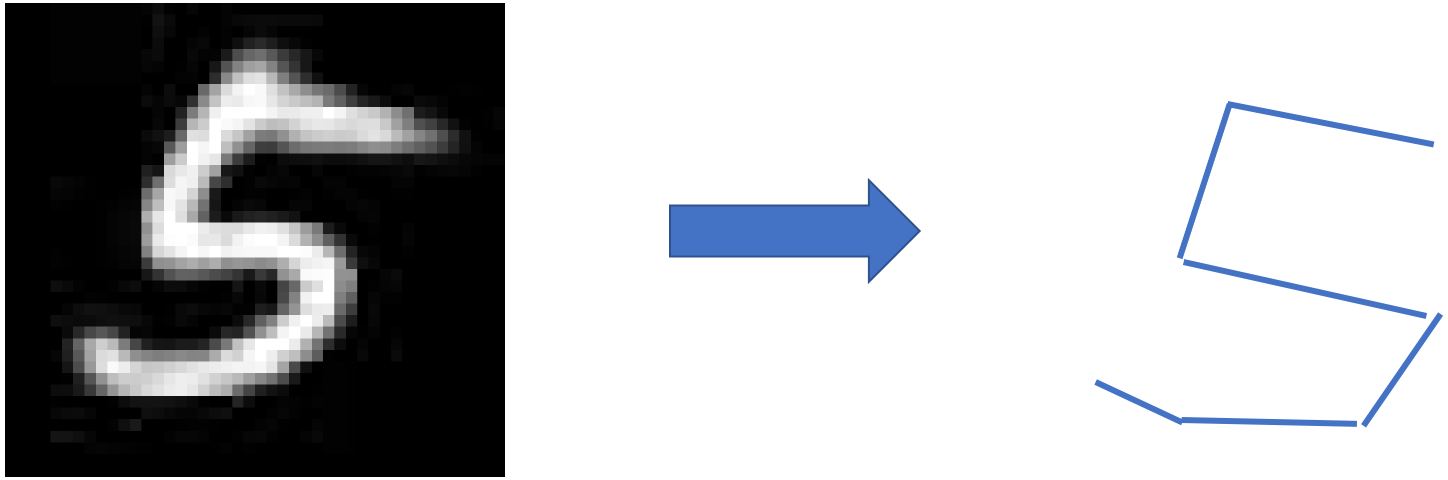
Unsupervised image vectorisation

Attila Kun

Vectorisation



Abstract language (example: SVG)
Usefulness:

1. Scale agnostic rendering
2. Dimensionality reduction

MNIST database

- Collected from Census Bureau employees
- 70,000 images

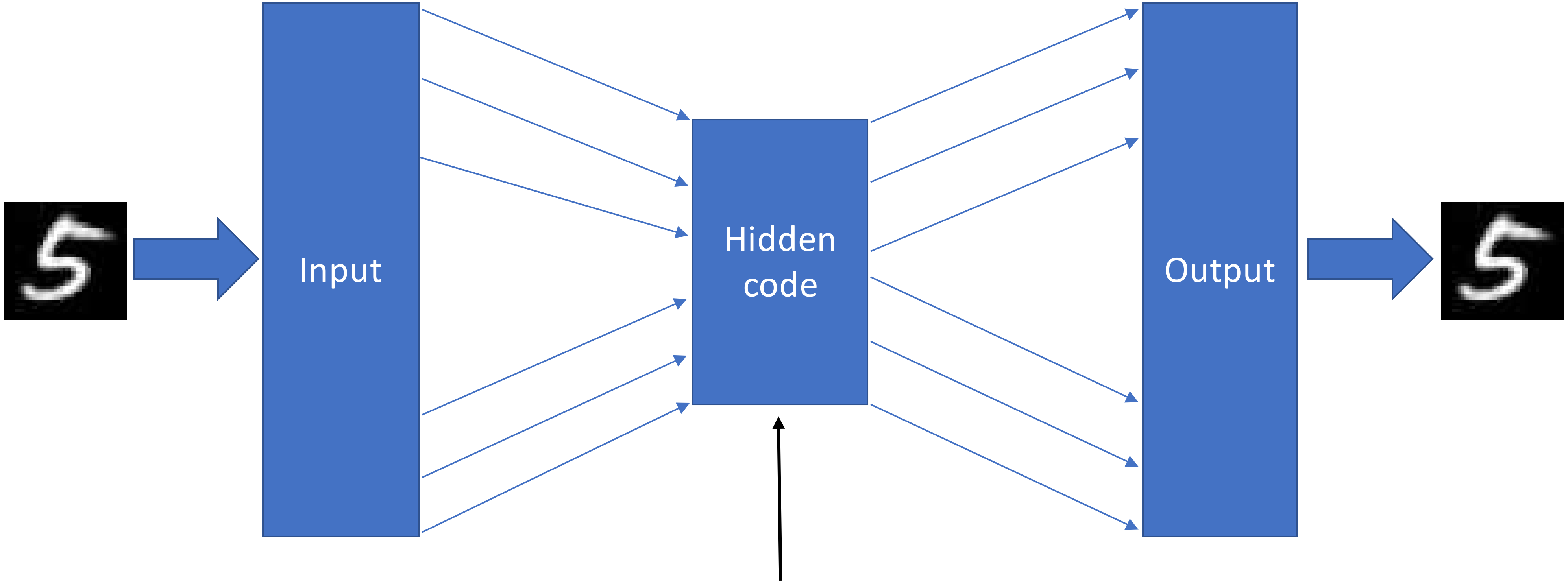
Goals

- Vectorise MNIST digits using fix number of lines
- Use no labels
- Make use of deep learning as opposed to classical techniques
    - PyTorch (automatic differentiation, stochastic gradient descent)
    - Rust

3 approaches

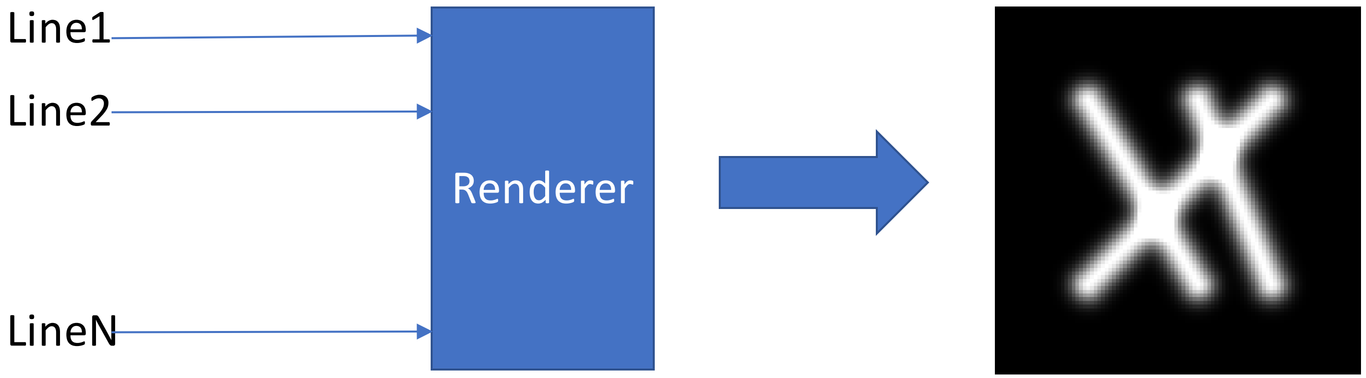
1. Traditional autoencoder
2. Variational autoencoder
3. Likelihood-free inference

Traditional autoencoder



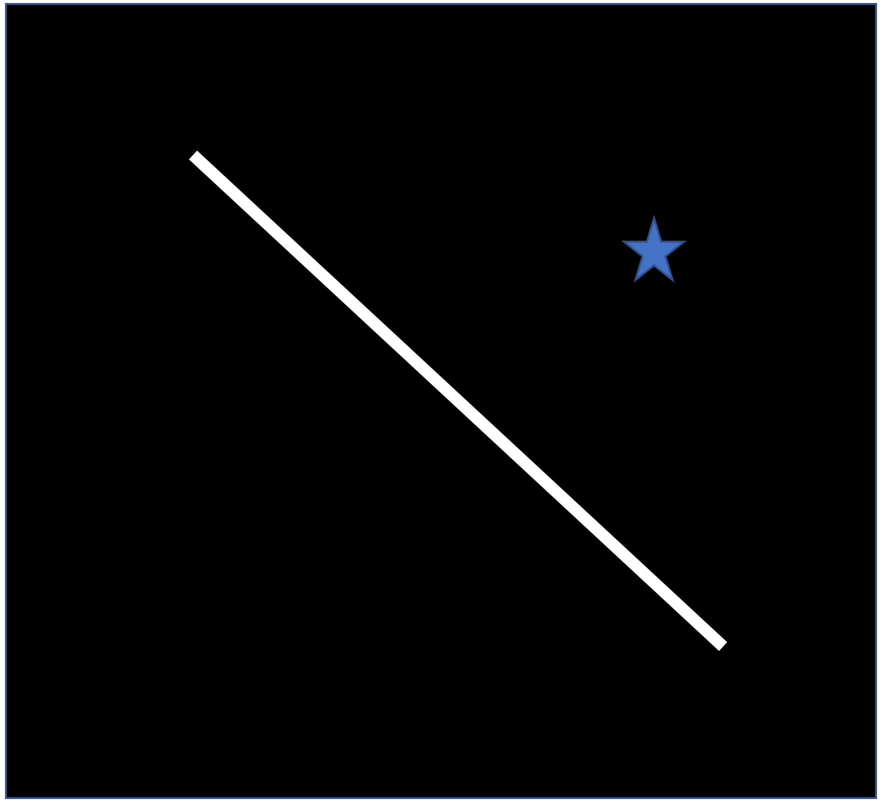
- Bottleneck layer
- Compresses information

Differentiable renderer

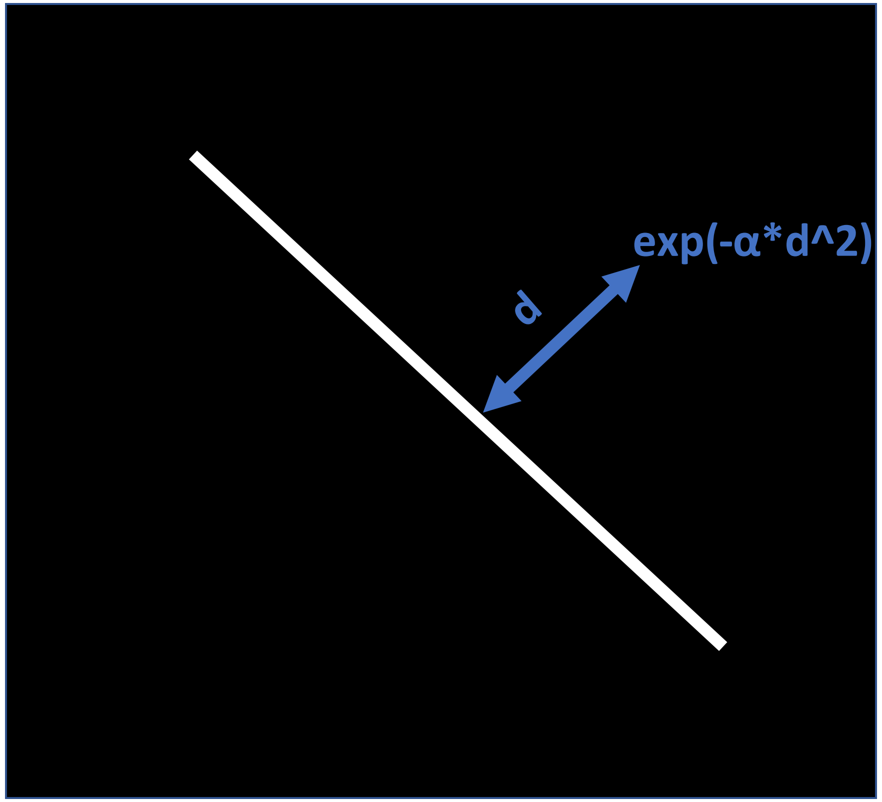


- Heat map
- Continuous function
- Soft renderer

Differentiable renderer

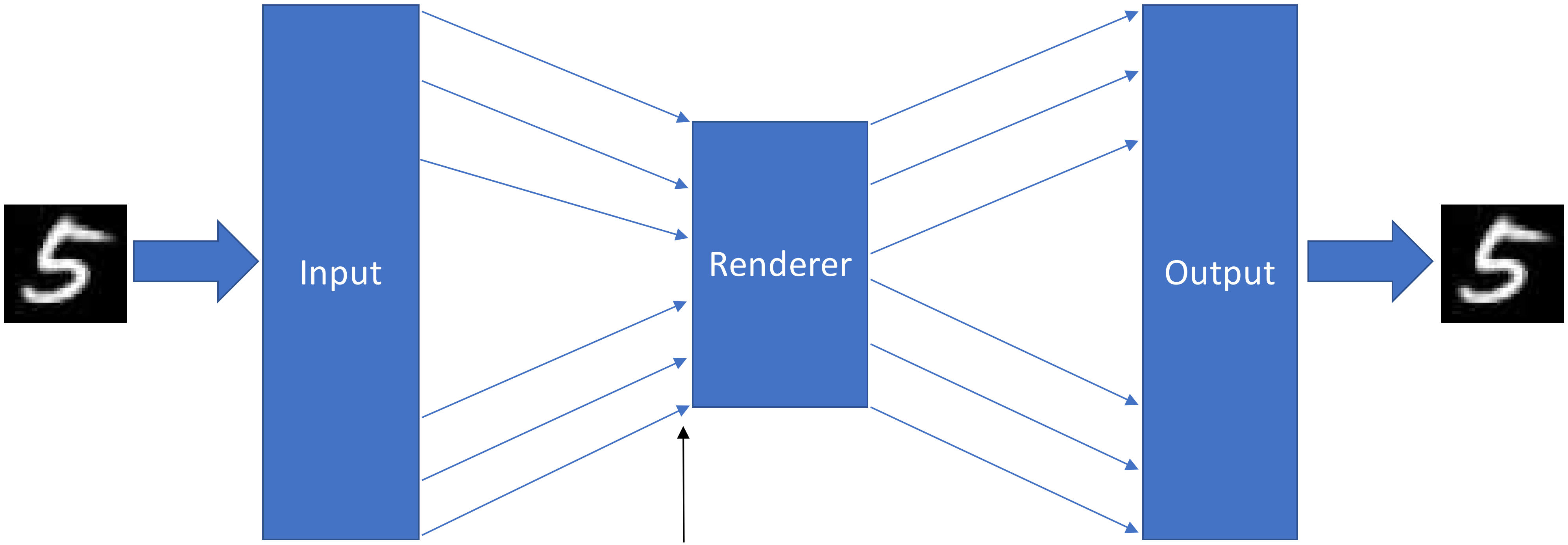


Differentiable renderer



Repeat for every pixel on GPU

Modified autoencoder



- Network learns how to produce useful inputs
- Well defined interface
- Inputs are the inferred vectors

Training

- Whole architecture is differentiable
- We can optimise using stochastic gradient descent
- Source of randomness: minibatch training

Results

Original MNIST

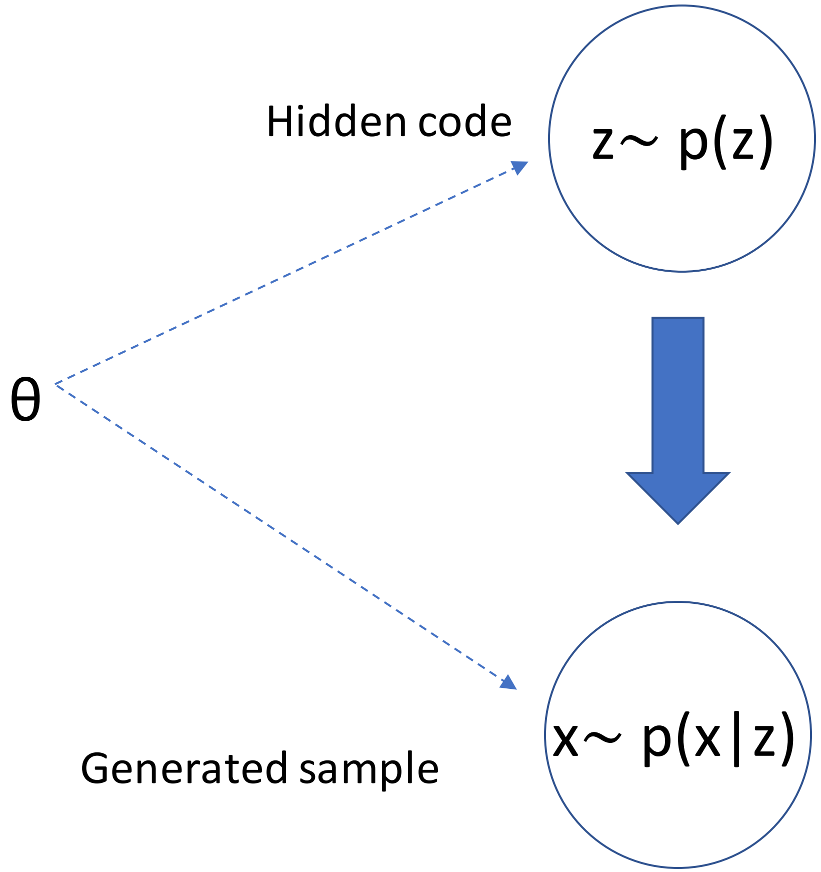
Vectorised output

Pro: fast training
Con: weak results, cumbersome soft rendering implementation

Learns something but not great. How to improve?

Variational autoencoder

Generative model:



$p(z|x)$?

Approximate the posterior using $q(z|x)$ parameterised by ϕ

Variational autoencoder

In the image vectorisation context:

- $p(z)$: Vectors
- $p(x|z)$: Generated images
- $p(z|x) \approx q(z|x)$: Inferred vectors of an image. Finding this is our goal.
- $p(x|z)$, $q(z|x)$ are neural networks
- $p(z)$ is multivariate standard normal

Variational lower bound

$$\log p_\theta(\boldsymbol{x_i}) \geq \underbrace{-D_{KL}(q_\phi(\boldsymbol{z_i}|\boldsymbol{x_i}) \; || \; p_\theta(\boldsymbol{z_i}))}_{\text{regularisation}} + \underbrace{\mathrm{E}_{q_\phi(\boldsymbol{z_i}|\boldsymbol{x_i})} \left[\log p_\theta(\boldsymbol{x_i}|\boldsymbol{z_i})\right]}_{\text{reconstruction}}$$

If the lower bound is maximal w.r.t. θ and ϕ, then q(z|x)=p(z|x)

Implementing $p(x|z)$

- How to turn vectors into an image?
- We need a renderer

Non-differentiable renderer

- Differentiable programming is annoying
- Non-differentiable programming is more general

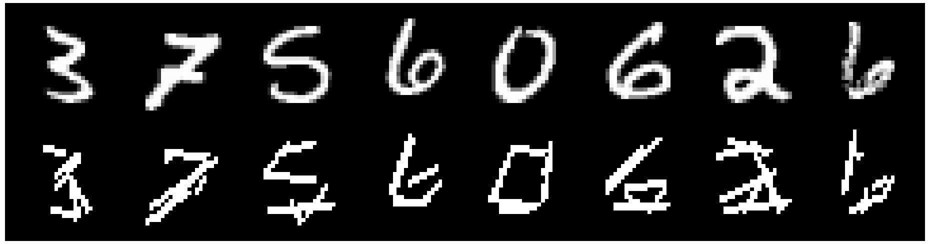
Solutions:

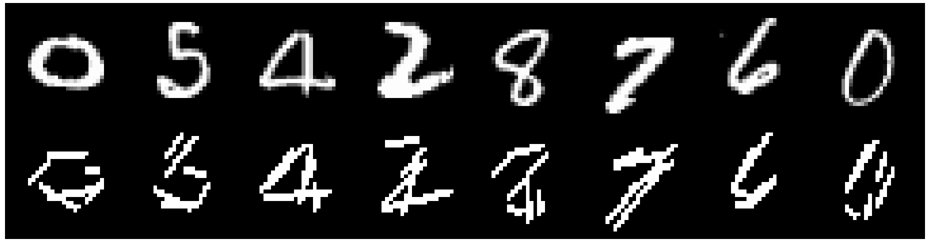
1. Approximating gradients with a neural network
2. Native implementation (Rust) => big speedup

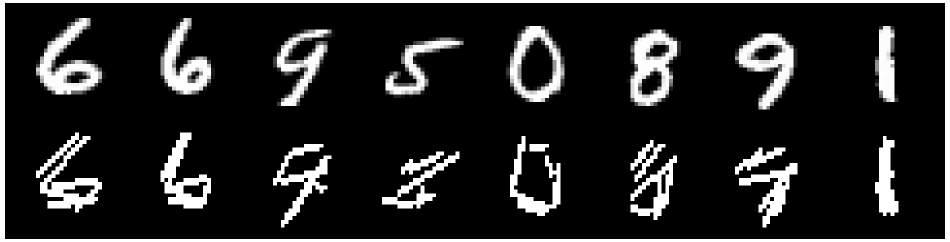
Putting it together

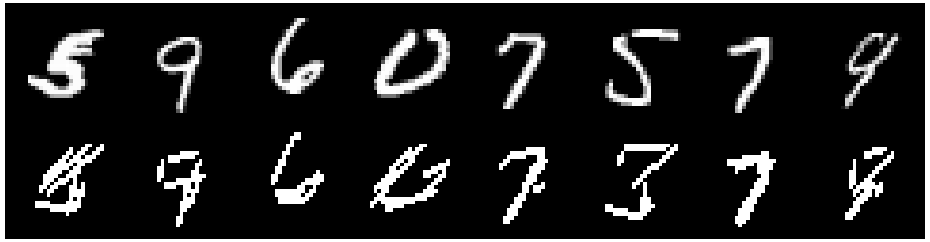
- Approximate renderer + variational objective
- Training via stochastic gradient descent

Results



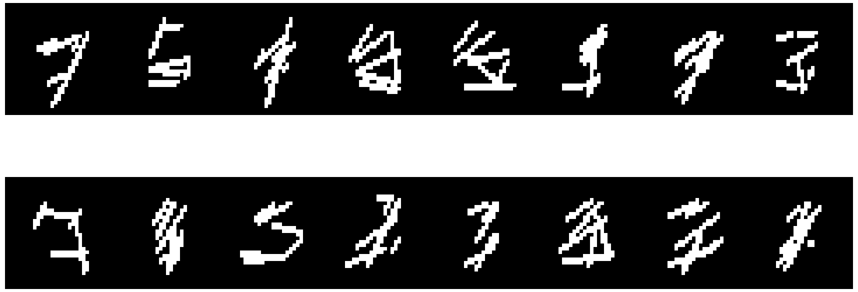




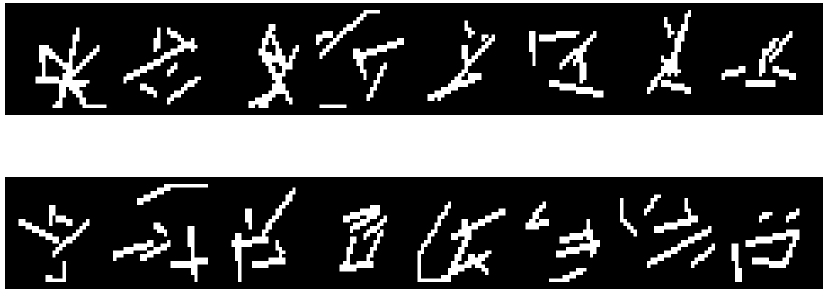


Sampling from prior

Regularised



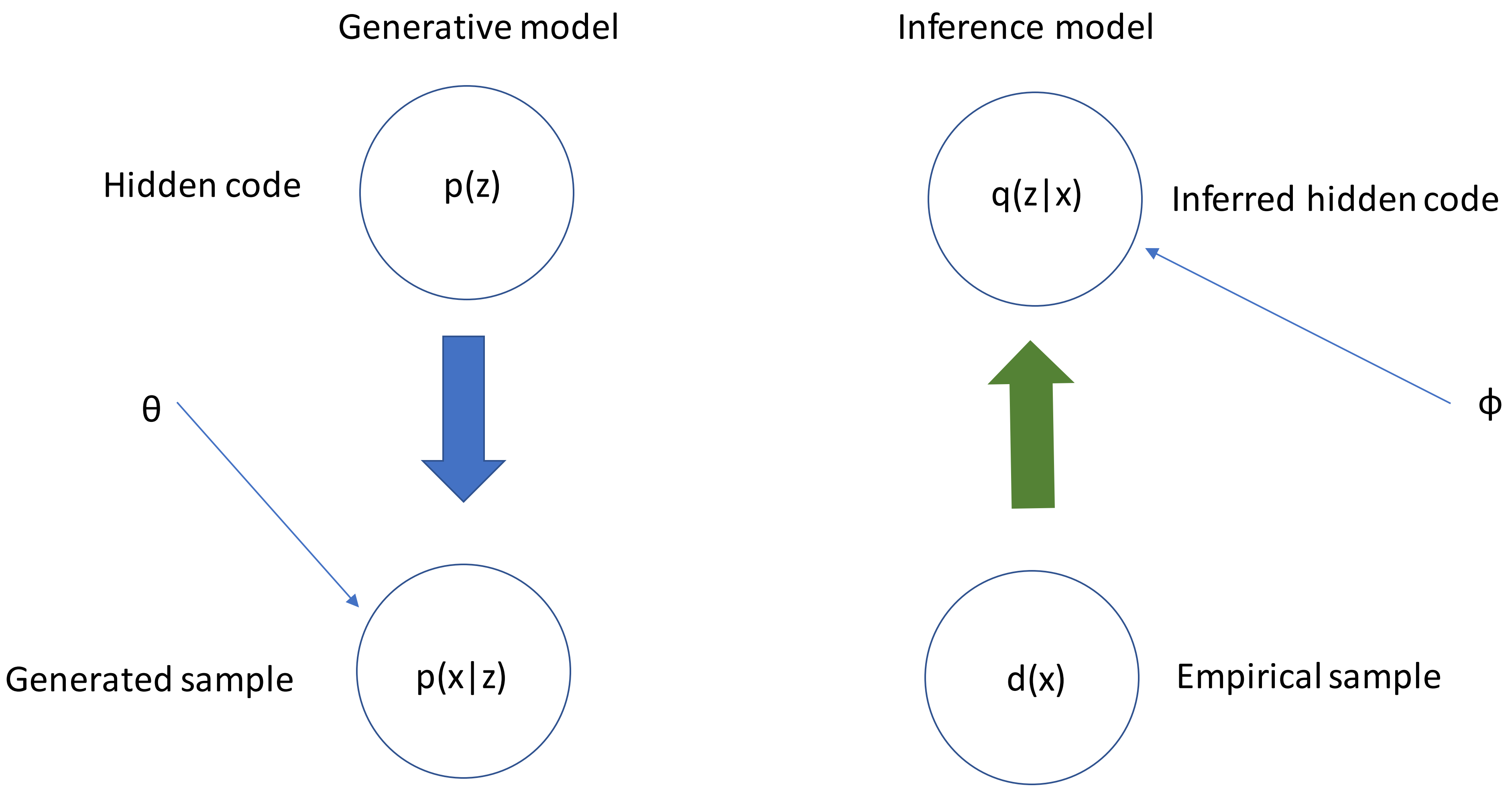
Unregularised



Likelihood free inference

- Variational autoencoders use fully specified distributions
- Sometimes we can only sample from an implicit distribution

Implicit distributions



Discriminator

- Train a discriminator that can tell the difference between pairs

Generative model		Inference model
z~p(z) x ~ p(x|z)	VS	z~q(z|x) x ~ d(x)

$$\min_{\boldsymbol{\theta},\boldsymbol{\phi}} \max_{\boldsymbol{\psi}} \mathrm{E}_{p(\boldsymbol{x}_n,\boldsymbol{z}_n|\boldsymbol{\theta})}\left[\log D(\boldsymbol{x}_n,\boldsymbol{z}_n;\boldsymbol{\psi})\right] + \mathrm{E}_{q(\boldsymbol{x}_n,\boldsymbol{z}_n|\boldsymbol{\phi})}\left[\log(1 - D(\boldsymbol{x}_n,\boldsymbol{z}_n;\boldsymbol{\psi}))\right]$$

Problems

- Mode collapse
- Complication: approximate rendering
- Possible solution: Different objective (Wasserstein)

Summary

1. Traditional autoencoder (differentiable renderer) 😐
2. Variational autoencoder (non-differentiable renderer) 🙂
3. Likelihood-free variational inference ☹

Conclusion:

- Classical techniques might be still better
- More complicated approaches like SPIRAL would give better results at the cost of higher complexity

Thank you. Questions?