Generative Models and VAEs

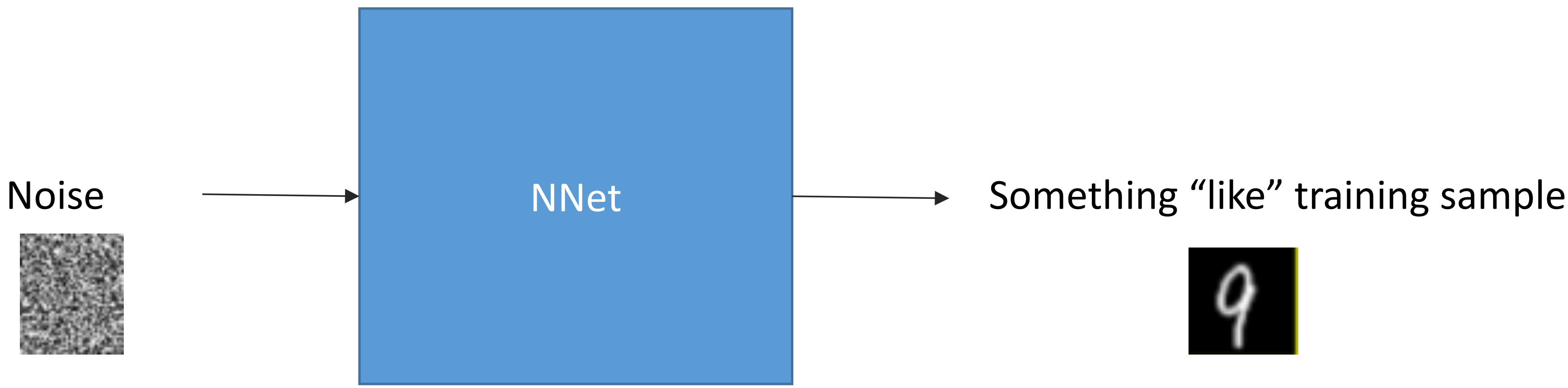
Yale

CPSC/AMTH 663

Outline

1. Generating with Neural Networks
2. MMDnet: Generating Neural Networks with Noise
3. Generalized Stochastic Autoencoders
4. VAEs

Generative Models



Probabilistic Interpretation

Randomly Sampled Noise → NNet → Sample from training Distribution

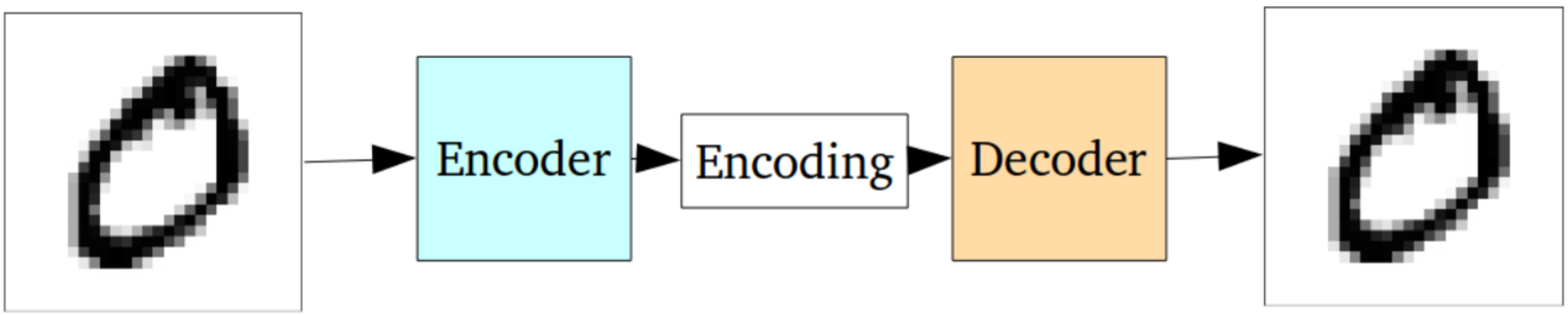
X~N(μ, σ)

X~P(X)

Images Generated from GANs

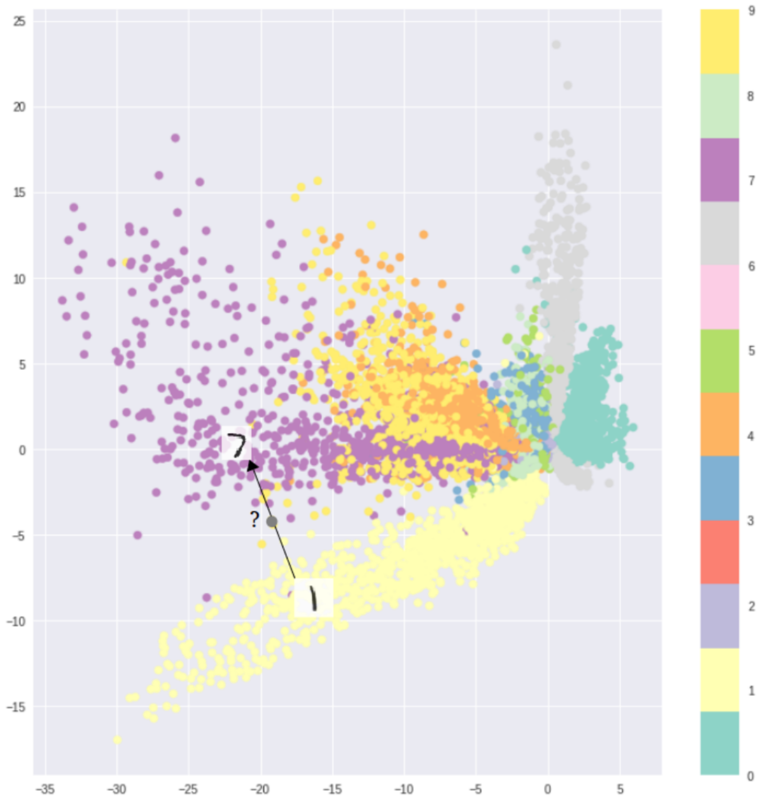
Goodfellow

Why not autoencoders?



- Optimized with Reconstruction Loss
- Not explicitly penalized for Generative Purposes

Latent Space Not Optimal for Generation



Optimizing purely for reconstruction loss

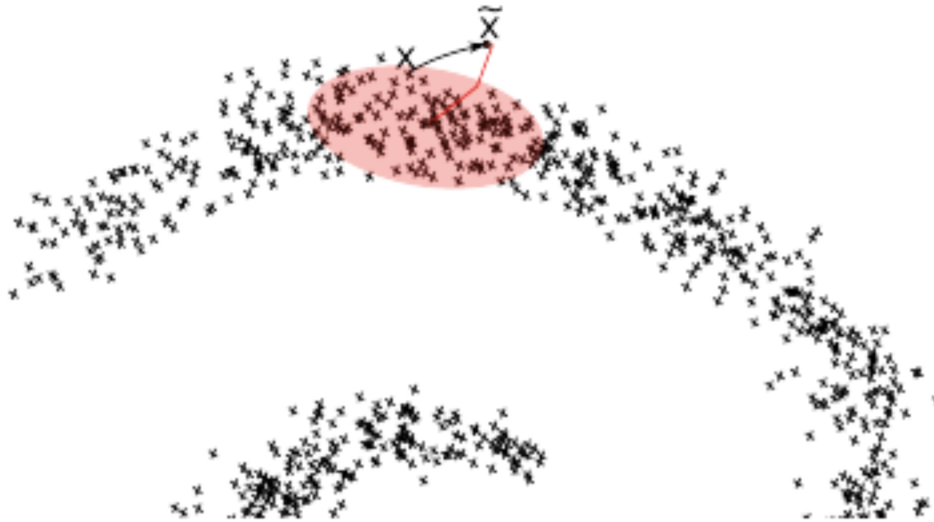
Formation of clusters helps decoding

Does not help generation

Latent space cannot be interpolated

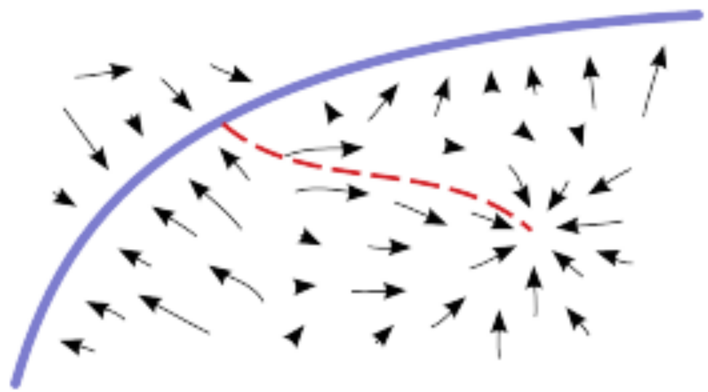
Can we modify autoencoder for generation?

Generalized. Denoising
Autoencoder
[Bengio et al. 2013]



- Basic idea: add noise to samples to push them away from the data manifold and then have them pull it back.
- For each training sample X define a corruption process C($\tilde{X}$,X) that creates a corrupted sample $\tilde{X}$.
- Train a denoising autoencoder to reverse this by using $(\mathrm{X}, \tilde{X})$ as a training example

Lost away from the manifold?



- Problem: spurious modes
- The DAE may not be able to walk back well enough if you get too far away from the sample space

Walkback training

- Train the neural network to walk back from several steps away
- Create longer range corruption processes using the original corruption process
- Sample a second step $\mathrm{C}(\tilde{\tilde{X}}, \tilde{X})$
- Add the training sample $(\tilde{\tilde{X}}, \tilde{X})$ to the training

Without walkback

With walkback

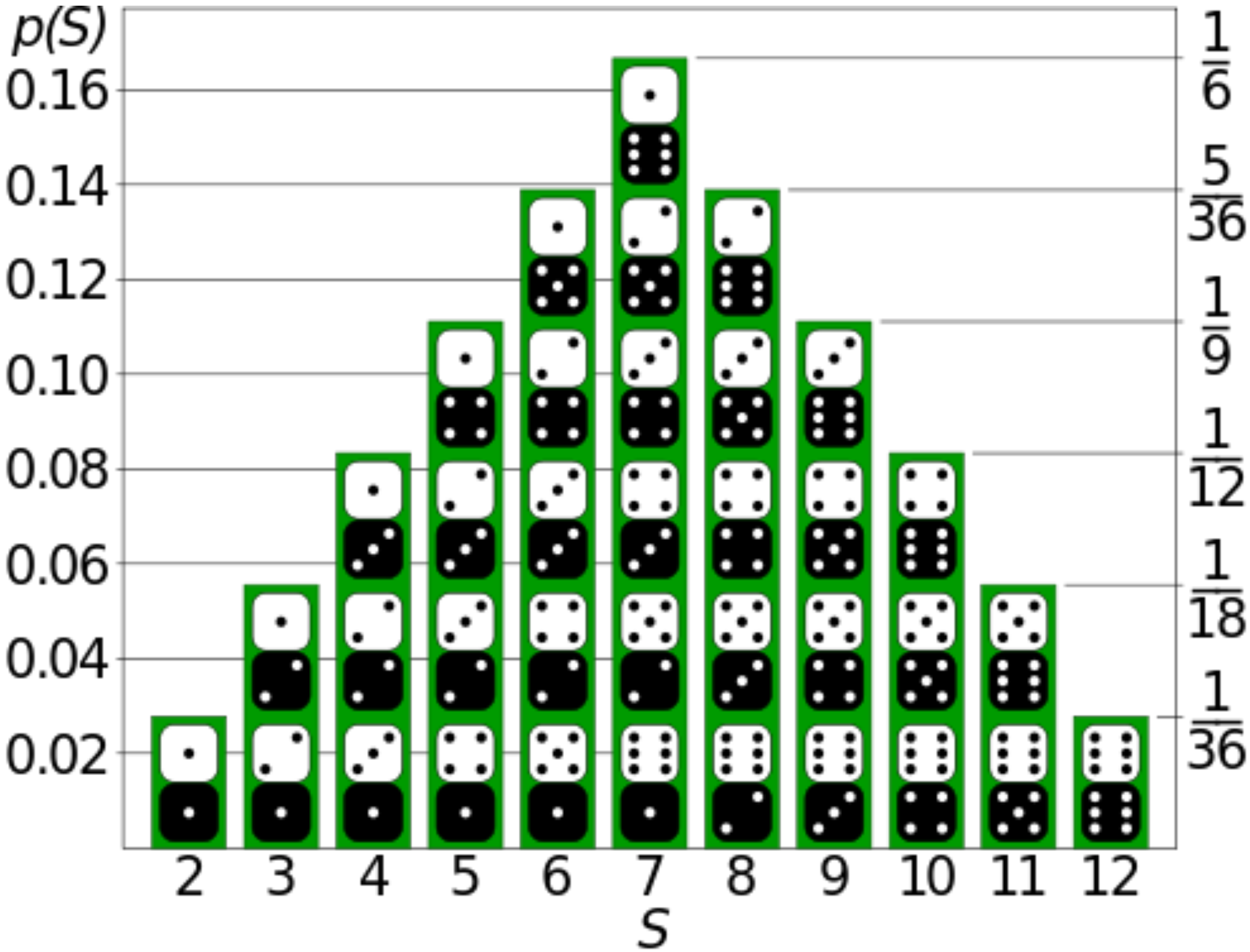
Trains the DAE to estimate conditional

- This way of training actually trains the DAE to estimate a conditional probability distribution $\mathrm{P}(\mathrm{X}|\tilde{X})$
- [Bengio et al. 2013] show that a consistent estimator of P(X), i.e. distribution of the training example can be recovered by alternating sampling from the corruption process. $\mathrm{C}(\tilde{X}$,X) and the denoising process $\mathrm{P}(\mathrm{X}|\tilde{X})$
- Turns out that learning a conditional distribution is a lot simpler than learning the joint distribution!
    - This idea is used in style transfer and other places
- A conditional distribution can just be a Gaussian or something simple, but this alternation allows convergence towards the joint distribution

Probabilities

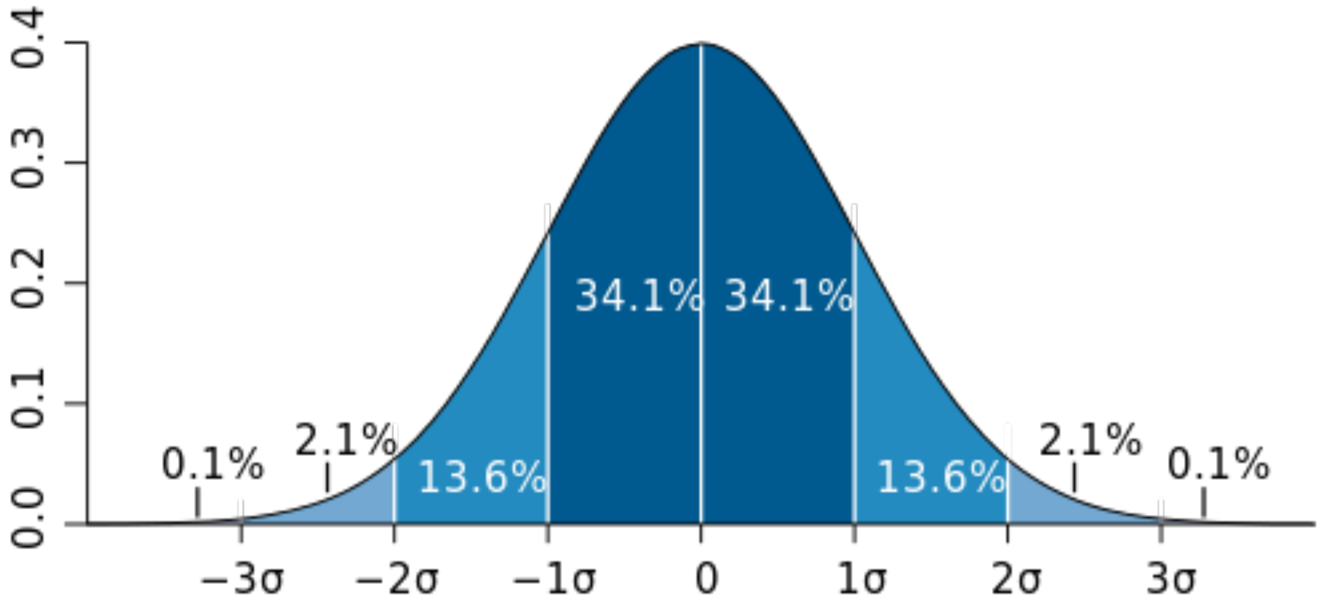
- Probability is the likelihood P(E) that an event E will occur
- Random experiment, with set of possible outcomes in S
- Probability is a function assigned to subsets E of S, in the range [0,1]
- Axioms:
  - Non-negativity $0 \le P(E) \le 1$
  - Measure $P(S) = 1$
  - Additivity of disjoint events $P(E_1 \cup E_1) = P(E_1) + P(E_2)$

Probability Mass Function



Assigns probabilities to each event

Probability Density Function



Describes the infinitesimal probability of any given value, and the probability that the outcome lies in a given interval can be computed by integrating the probability density function over that interval

Real Data

- Do not have PDF
- Need to estimate PDF
- Why?
  - Generative/predictive model
  - Estimate mutual information
  - Capture inherent stochasticity
  - Reason about outliers and noise

Estimating Probability Distributions

- Parametric distributions:
  - Maximum likelihood:
  - MAP

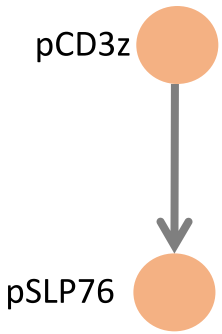
$$\arg\max_{\theta} p(data \mid \theta)$$

- Example

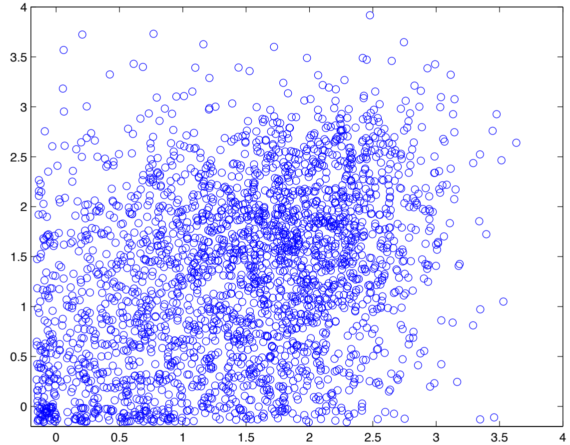
$$\arg\max_{\theta} p(\theta) p(data \mid \theta)$$

- Gaussian:
  - Parameters: μ Mean, σ variance
- Poisson
  - Parameter: λ (mean and variance)

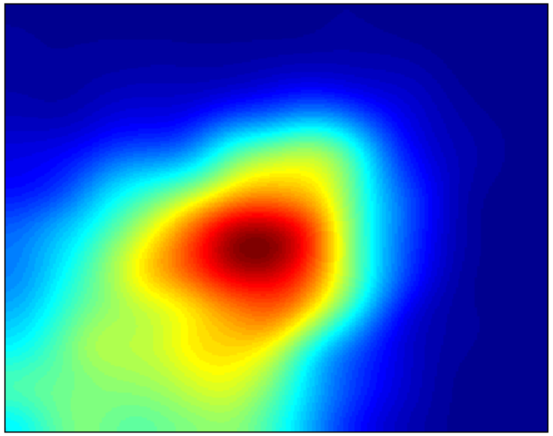
Non-parametric Estimates



Raw Data



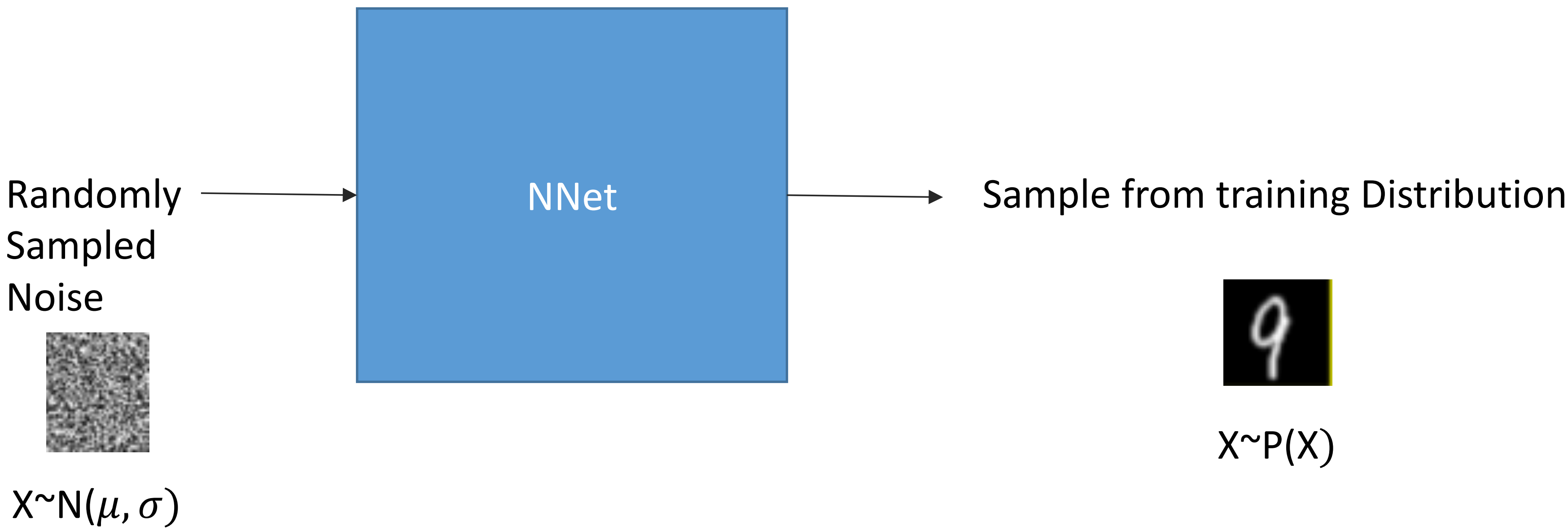
Kernel Density Estimate



- A density estimate is a function $\hat{f}(x)$ that is an approximation to the true probability density function
- Properties: real-valued, non-negative, integrates to 1
- Non-parametric estimate, no assumptions on shape

Neural Network Estimates

- Train a neural network with input stochasticity to mimic output distribution



- Penalize by a distribution distance or divergence: KL divergence, MMD distance, wasserstein distance

MMD nets

- Train the neural network to transmute noise into desired probability distribution using Maximum Mean Discrepancy optimization. [Dziugaite et al. 2015]
- MMD is a kernel-based 2-sample distribution test for distribution similarity
- This is a batch-level penalty, penalize a whole batch to look like the training distribution

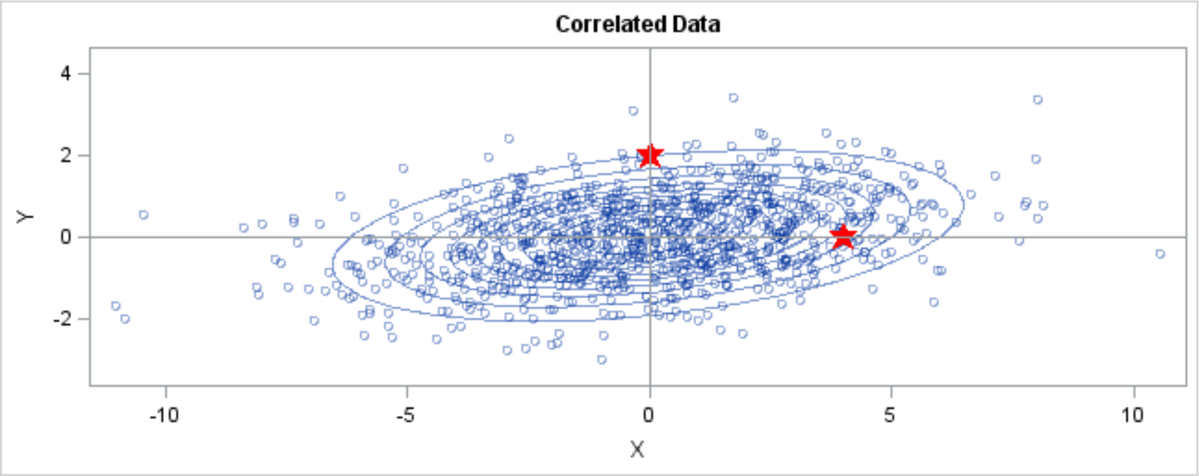
Distances

- Distances measure ways in which objects are different
- A distance metric is a real valued function d(x,y) such that
  - d(x,y)>=0 non-negative
  - d(x,x) = 0 identity
  - d(x,y)=d(y,x) symmetric
  - d(x,z)<=d(x,y)+d(y,z) triangle inequality

Example Distances

- $A=[a_1, a_2, \ldots a_n]$, $B=[b_1, b_2, \ldots b_n]$
- Euclidean Distance $||A-B||_2$
- Minkowski Distance $||A-B||_P$
- Mahalanobis Distance

$$\sqrt{(A-B)\Sigma^{-1}(A-B)^T}$$



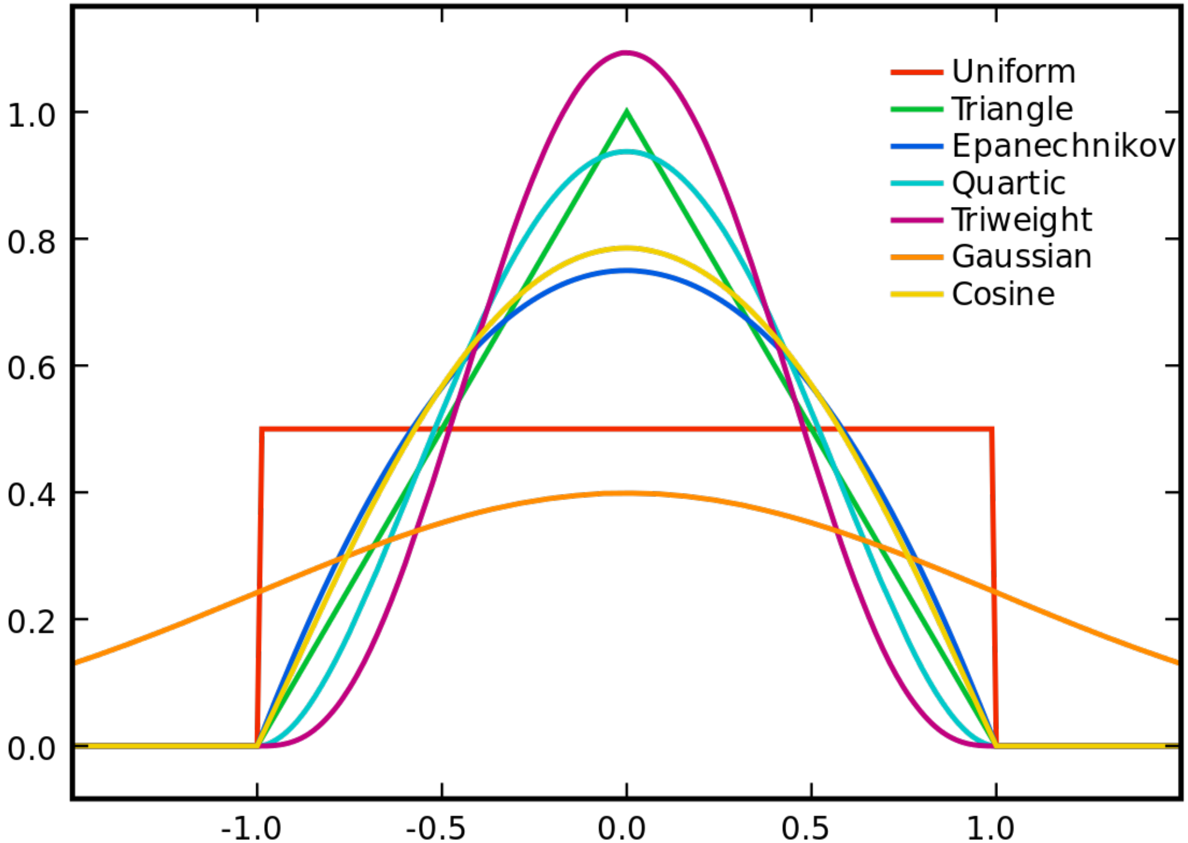
Affinities

- Similarity is a function s(x,y) such that:
  - s(x,x) =1
  - s(x,y) = s(y,x)
- Similarities between datapoints
- Correlation
- Cosine

$$\frac{\mathrm{cov}(x,y)}{\sigma^x \sigma^y}$$

$$\frac{\| x^T y \|}{\| x \| \| y \|}$$

Distance to Affinity via Kernels

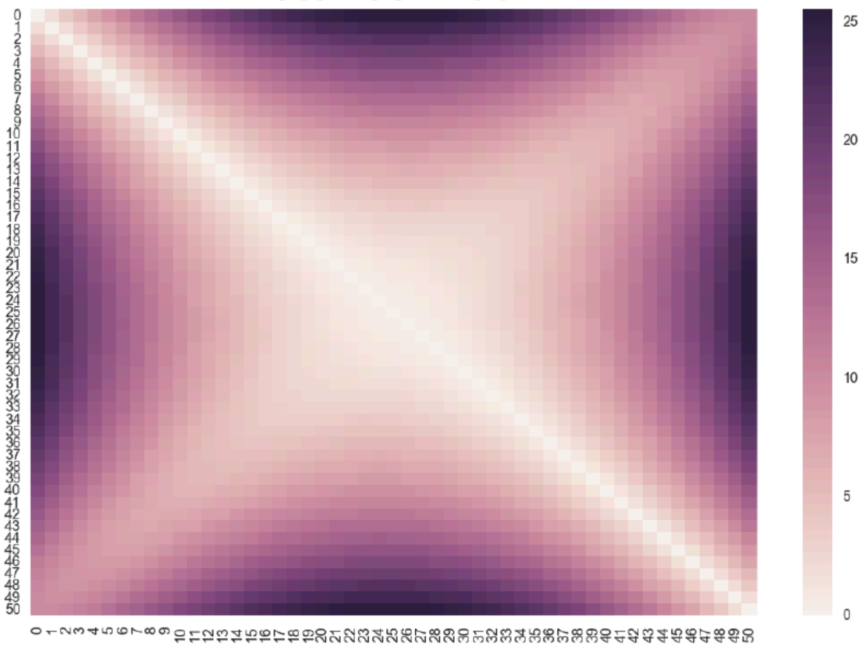


Real-valued
Non-negative
Symmetric

Affinities are correlations in this hidden hypothetical space

Distance->Affinity

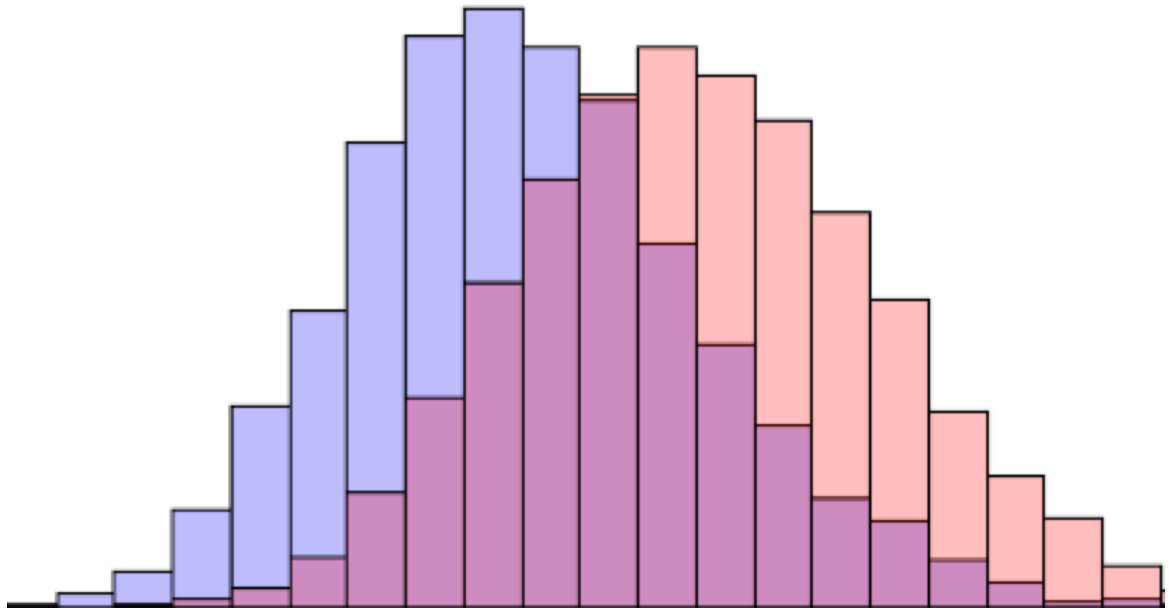
Distance Matrix



Affinity Matrix

$$sij := \exp\left(-\frac{d(x_i,x_j)^2}{2\sigma^2}\right)$$

Mean Maximal Discrepancy



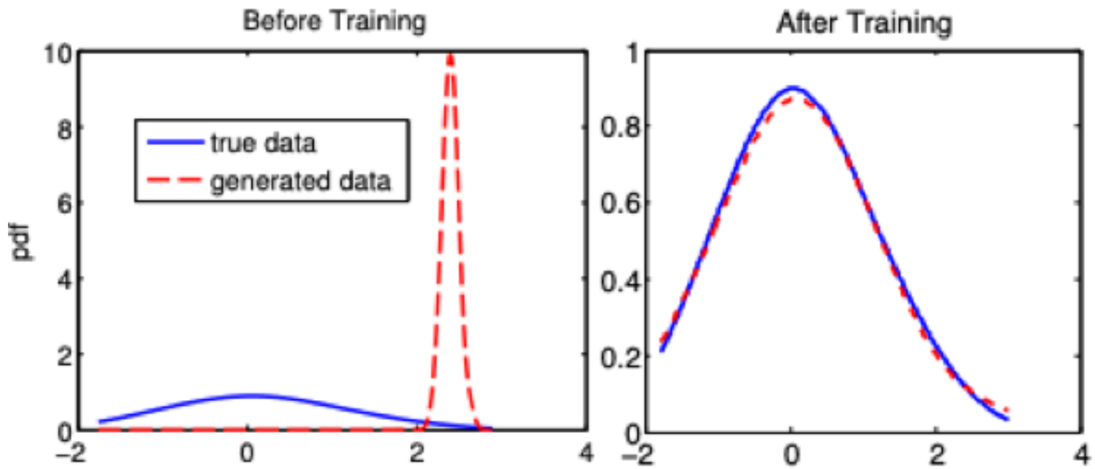
$$MMD(p,q) = \frac{1}{m^2}\Sigma_{i,j\in m}K(p_i,p_j) - \frac{2}{mn}\cdot\Sigma_{i,j}K(p_i,q_j) + \frac{1}{n^2}\Sigma_{i,j\in n}K(q_i,q_j)$$

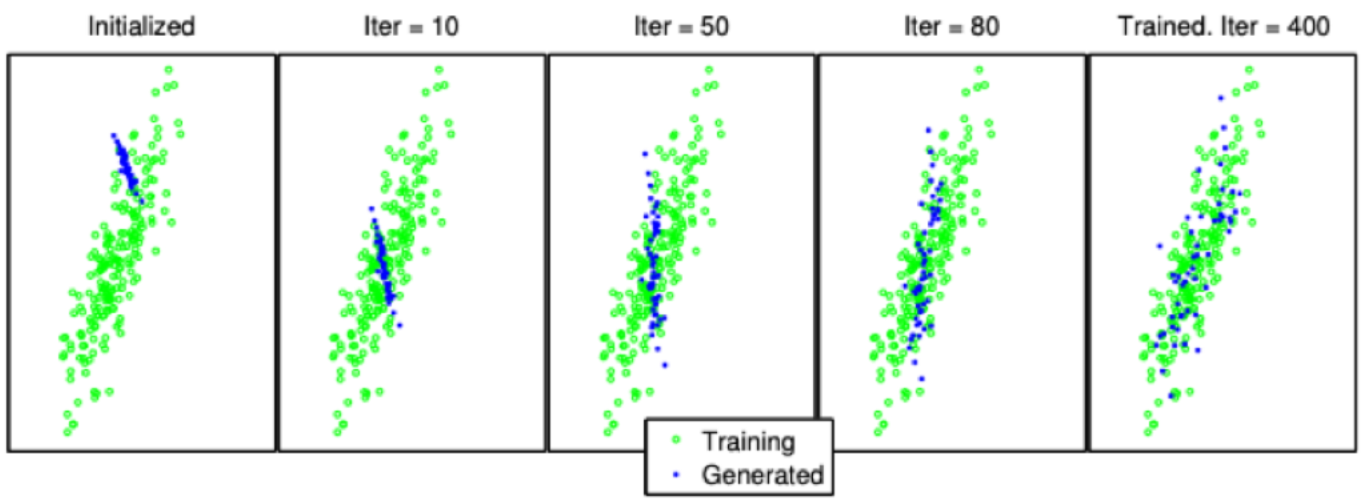
[Dziugaite, Roy, Ghahramani, UAI 2015]

MMD

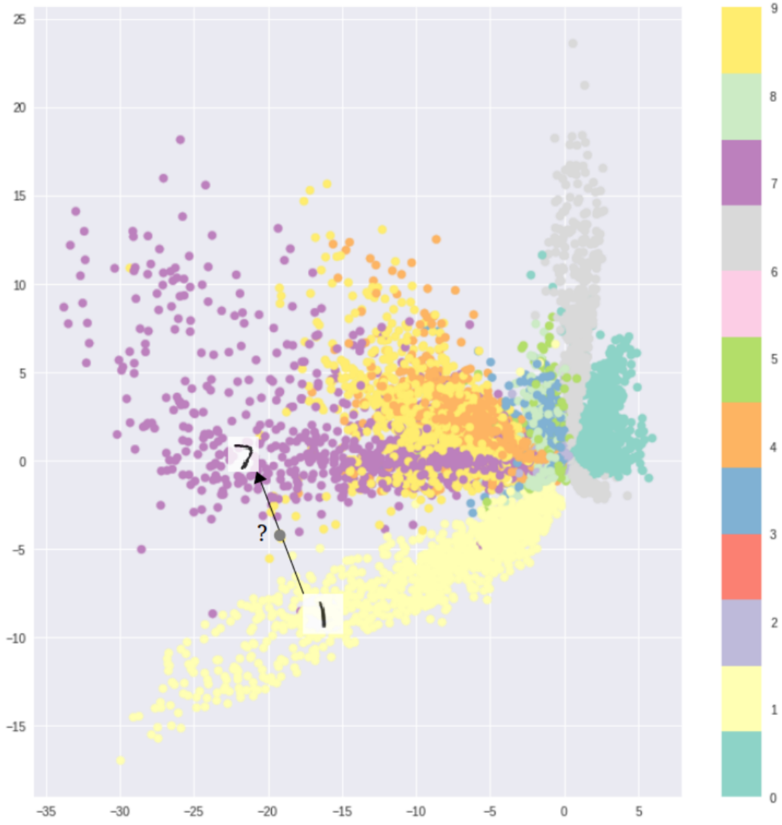
MMD quantifies, how similar are two sets of samples by

- Picking a pair X, X’ from distribution 1
- Picking a pair Y, Y’ from distribution 2
- Testing how similar the with sample pairs are compared to the across-sample pairs X,Y and X’,Y’
- The average distance between the within sample similarities and across samples


Before Training
After Training
true data
generated data
pdf
10
8
6
4
2
0
−2
0
2
4
1
0.8
0.6
0.4
0.2
0

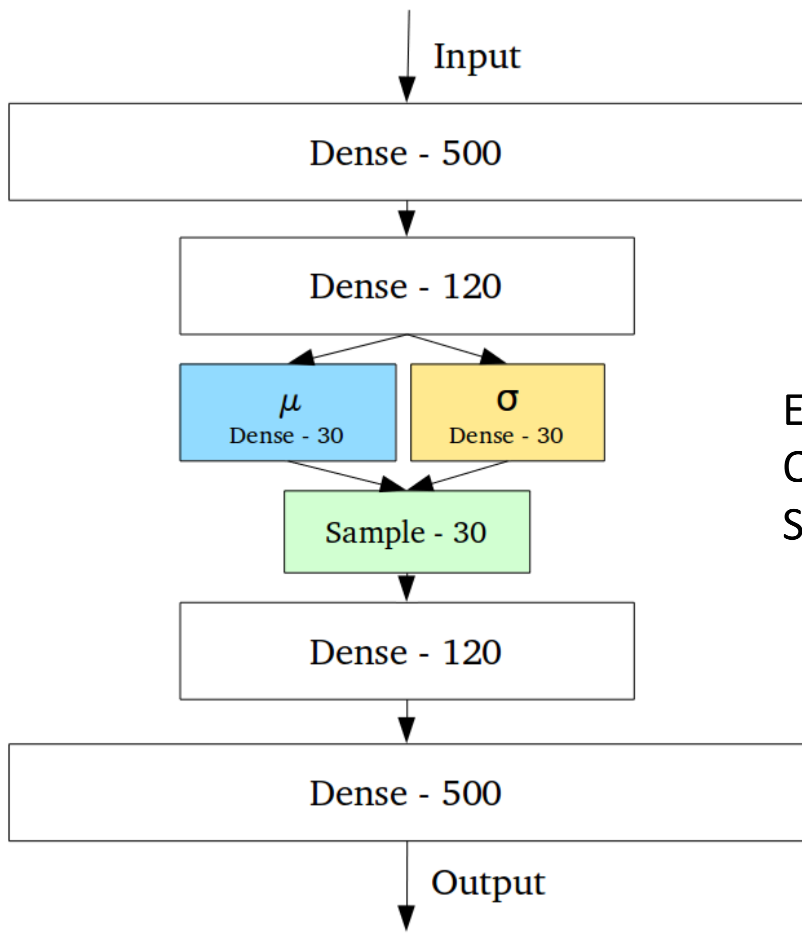

Initialized
Iter = 10
Iter = 50
Iter = 80
Trained. Iter = 400
Training
Generated

VAEs Revisited- Want Dense Middle Layer



Optimizing purely for reconstruction loss

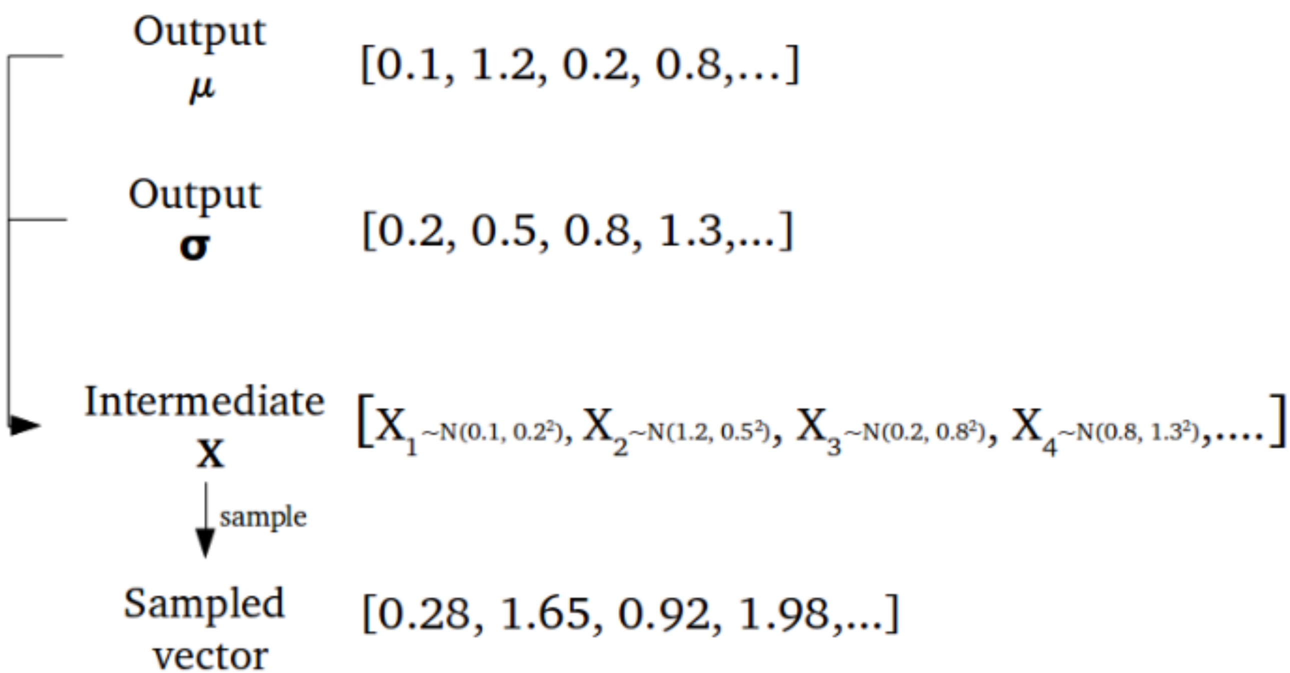
Designed for Sampling



Variational Autoencoder

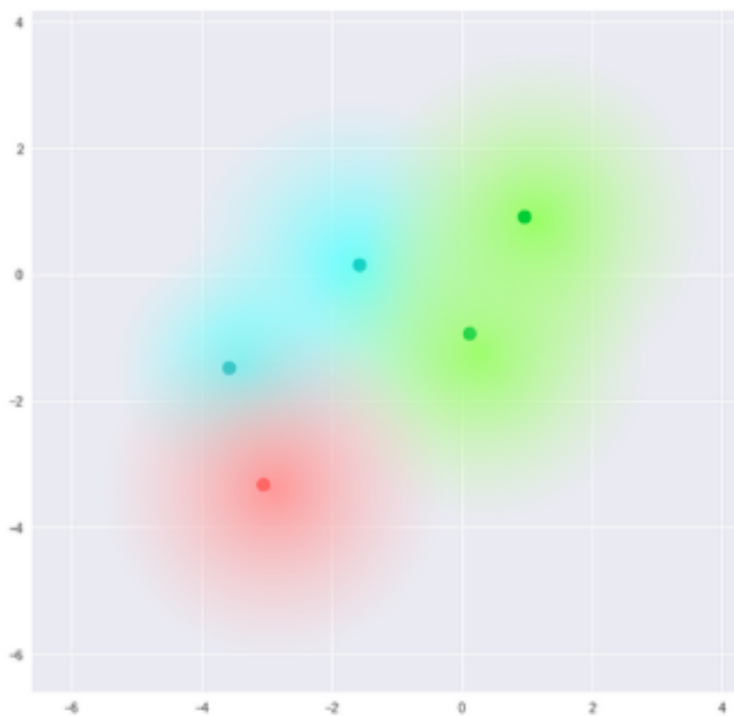
Encoding layer outputs two vectors
One of means and one of sigmas
Splits latent space into gaussian balls

Sampling a vector in latent space

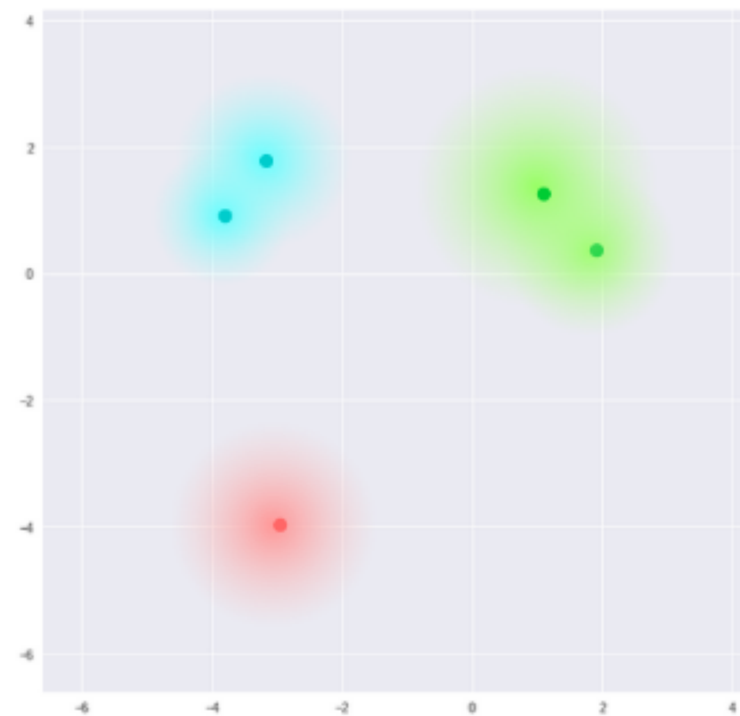


Stochastically generating encoding vectors

Discouraging Clusters



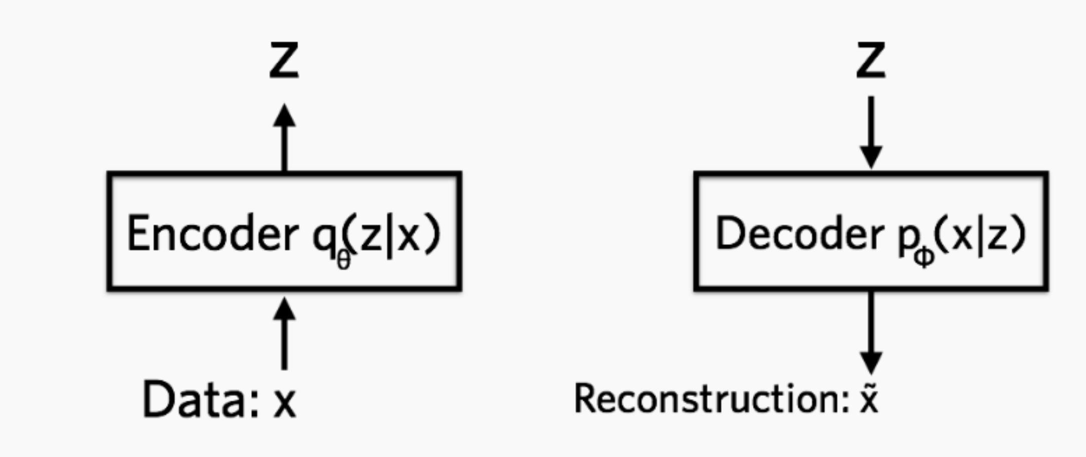
What we require



What we may inadvertently end up with

Loss Function of VAE

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z \mid x_i)}[\log p_\phi(x_i \mid z)] + \mathbb{KL}(q_\theta(z \mid x_i) \mid\mid p(z))$$



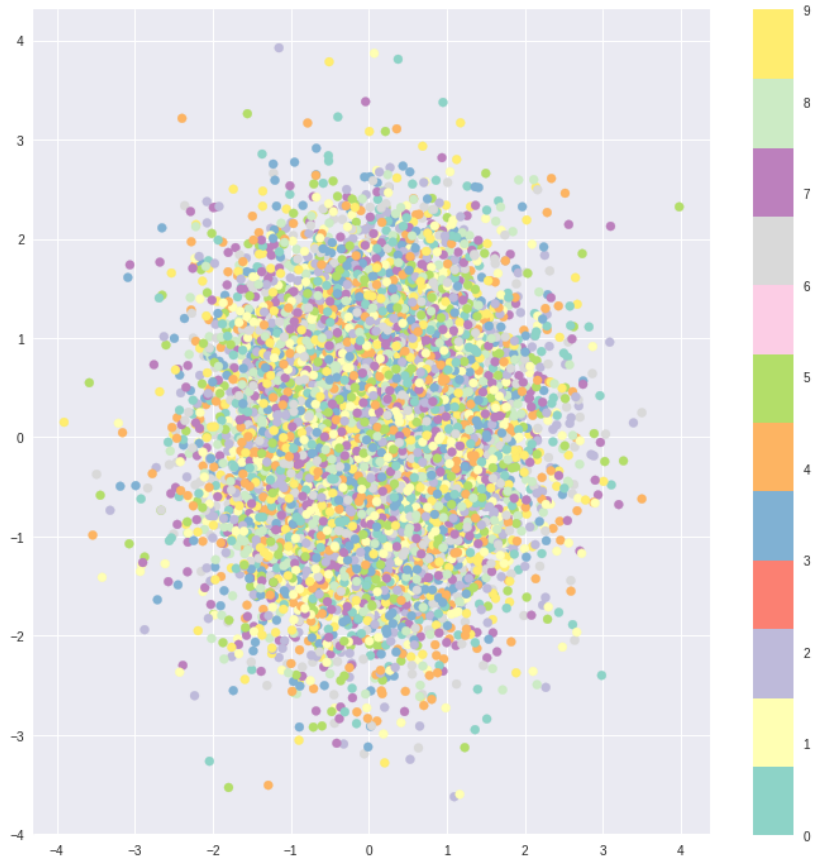
$$-\mathbb{E}_{z \sim q_\theta(z \mid x_i)}[\log p_\phi(x_i \mid z)]$$

KL Divergence Penalty

$$\mathbb{KL}(q_\theta(z \mid x_i) \mid\mid p(z))$$

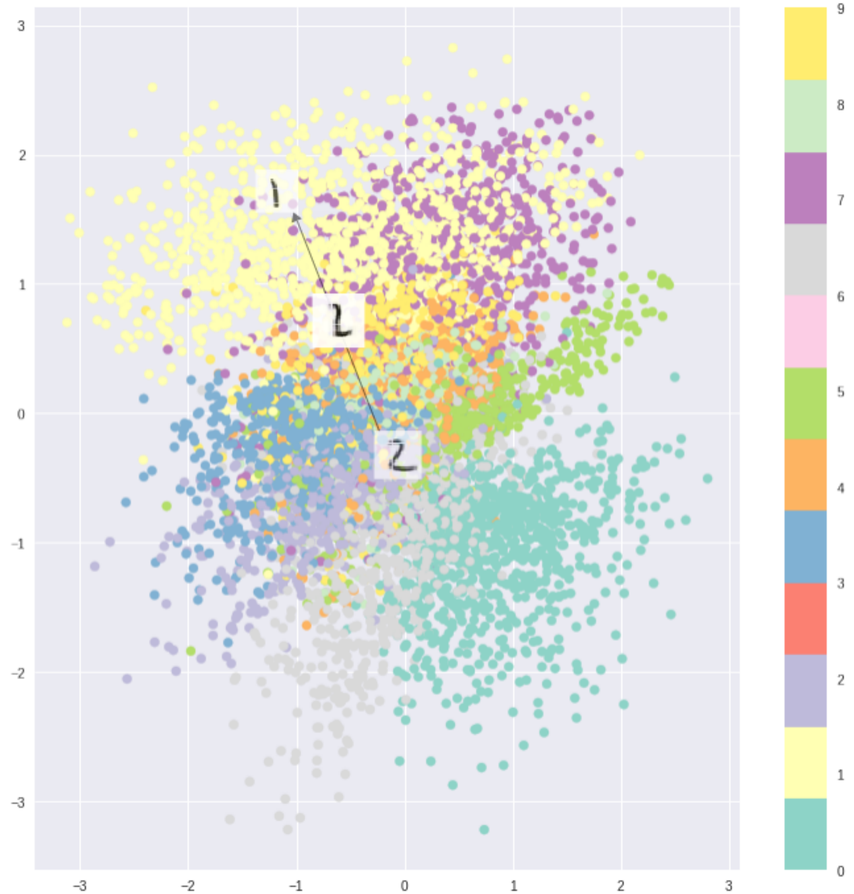
Penalizing distributions
In the latent space from being
Too far from a standard normal

$$p(z) = Normal(0, 1).$$



Optimizing using pure KL divergence loss

KL+Reconstruction Loss



Optimizing using both reconstruction loss and KL divergence loss

Suggested Reading Blogs

https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

https://jaan.io/what-is-variational-autoencoder-vae-tutorial/