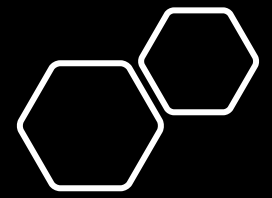
AIMS | African Institute for Mathematical Sciences RWANDA

AMMI Review sessions

Deep Learning (9)

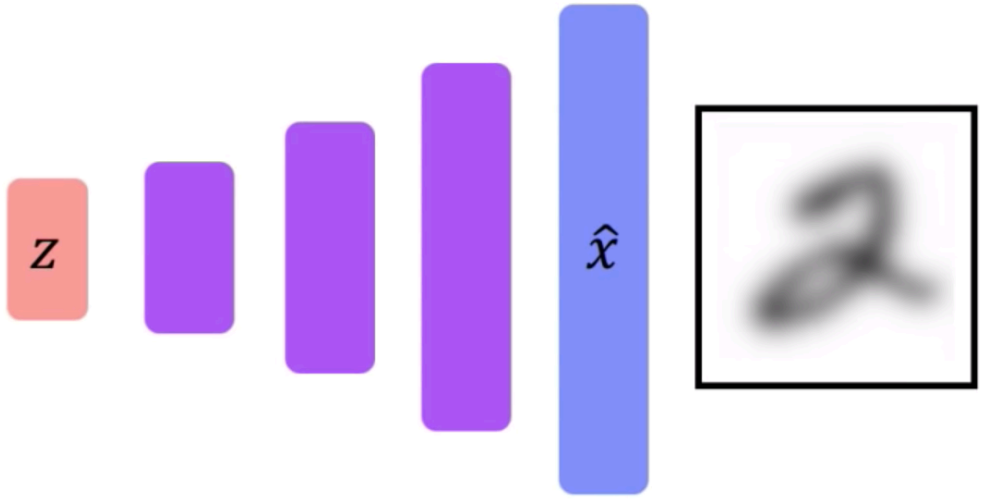
Variational Autoencoders VAEs



March 4th 2020

Limitations of standard autoencoders

- Given an autoencoder how would you generate a new sample ?
- What could be the appropriate choice of the latent code ?
- Do you have a guarantee that the decoder will produce something useful ?

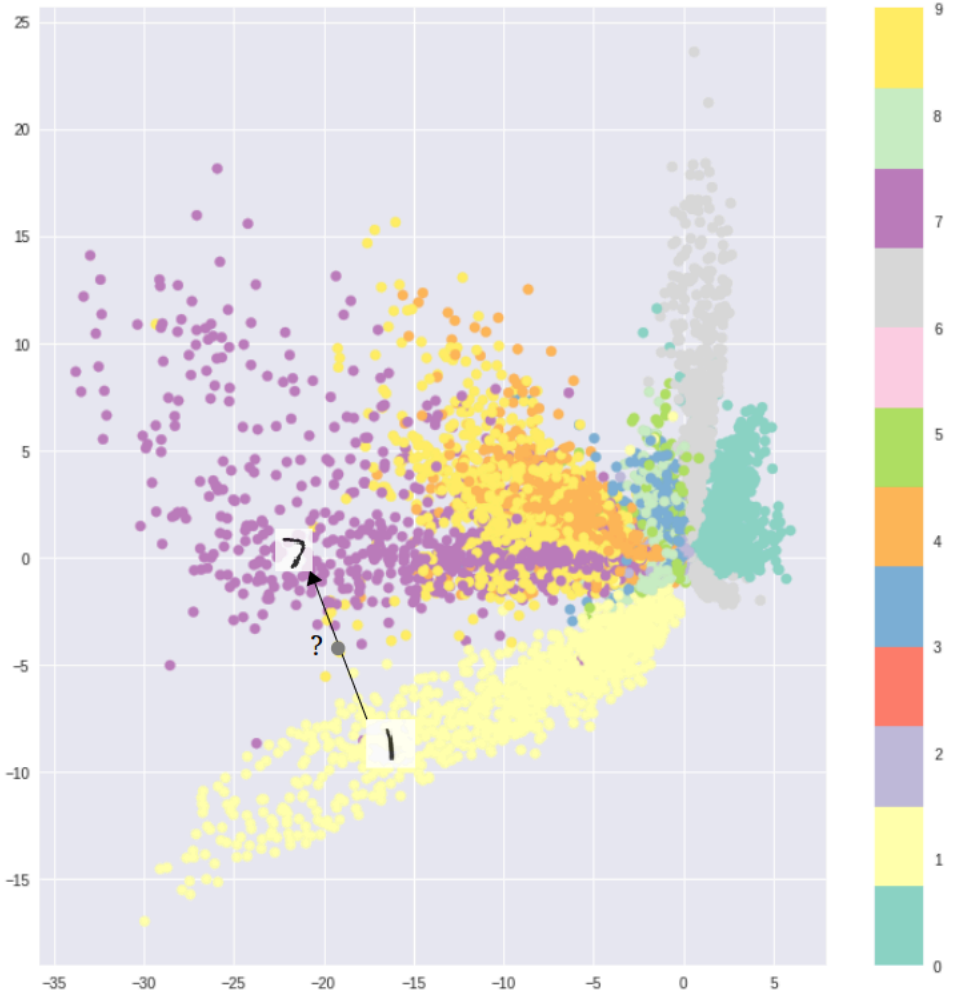


Limitations of standard autoencoders

- Given an autoencoder how would you generate a new sample ?
- What could be the appropriate choice of the latent code ?
- Do you have a guarantee that the decoder will produce something useful ?
- The fundamental problem with autoencoders, for generation, is that the latent space they convert their inputs to and their latent variable distribution, may not be continuous, or allow easy interpolation.

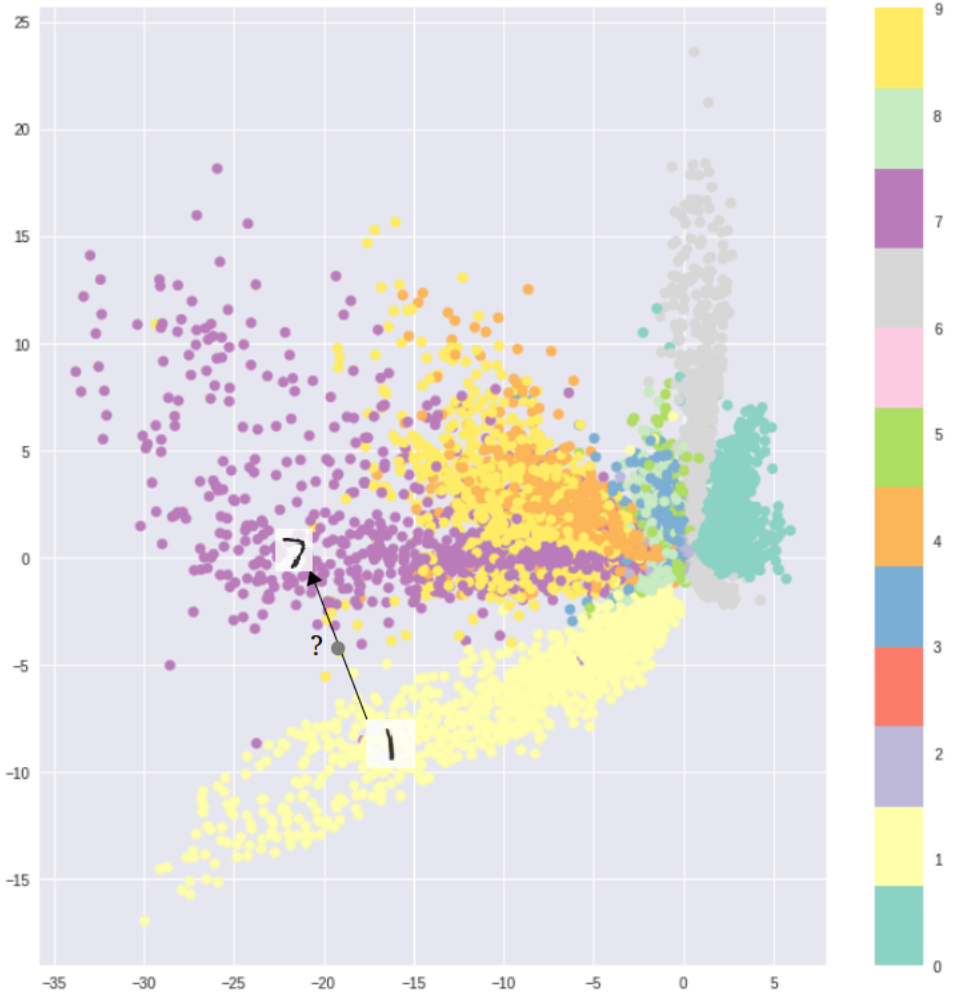
Limitations of standard autoencoders

- Visualization of a 2D latent space of an autoencoder trained on MNIST reveals the formation of distinct clusters.
- This makes sense, as distinct encodings for each image type makes it far easier for the decoder to decode them.
- This is fine if you're just replicating the same images, but when you're building a generative model, you want to randomly sample from the latent space, or generate variations on an input image.



Limitations of standard autoencoders

- If the space has discontinuities (e.g. gaps between clusters) and you generate a variation from there, the decoder will simply generate an unrealistic output, because the decoder has *no idea* how to deal with that region of the latent space. During training, it *never saw* encoded vectors coming from that region of latent space.



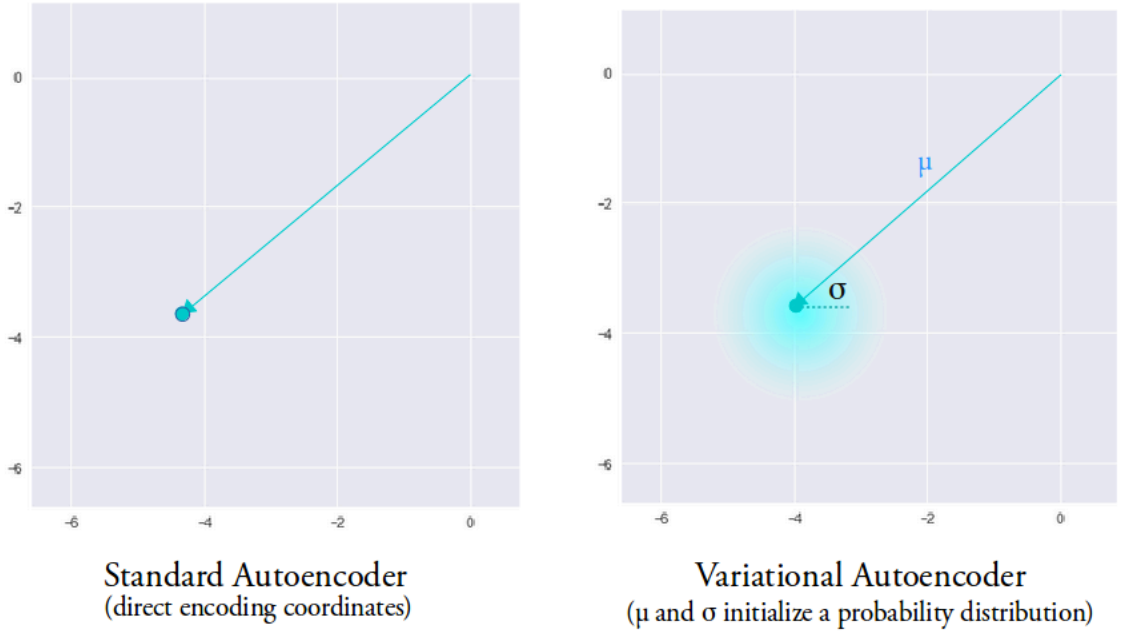
VAEs motivation: stochastic mapping

- Variational autoencoder are designed and trained to have a continuous latent space that allows sampling and interpolation

- It achieves this by making its encoder not output an encoding vector of size *n,* rather, outputting two vectors of size n: a vector of means, **μ**, and another vector of standard deviations, **σ**.

- The **μ** and **σ** form the parameters of n dimensional gaussian distribution $\mathbb{N}(\mu, \sigma)$

- Instead of mapping a point from the input space to a point from the latent space, we instead map a point from the input space to a distribution in the latent space!



Standard Autoencoder
(direct encoding coordinates)

Variational Autoencoder
(μ and σ initialize a probability distribution)

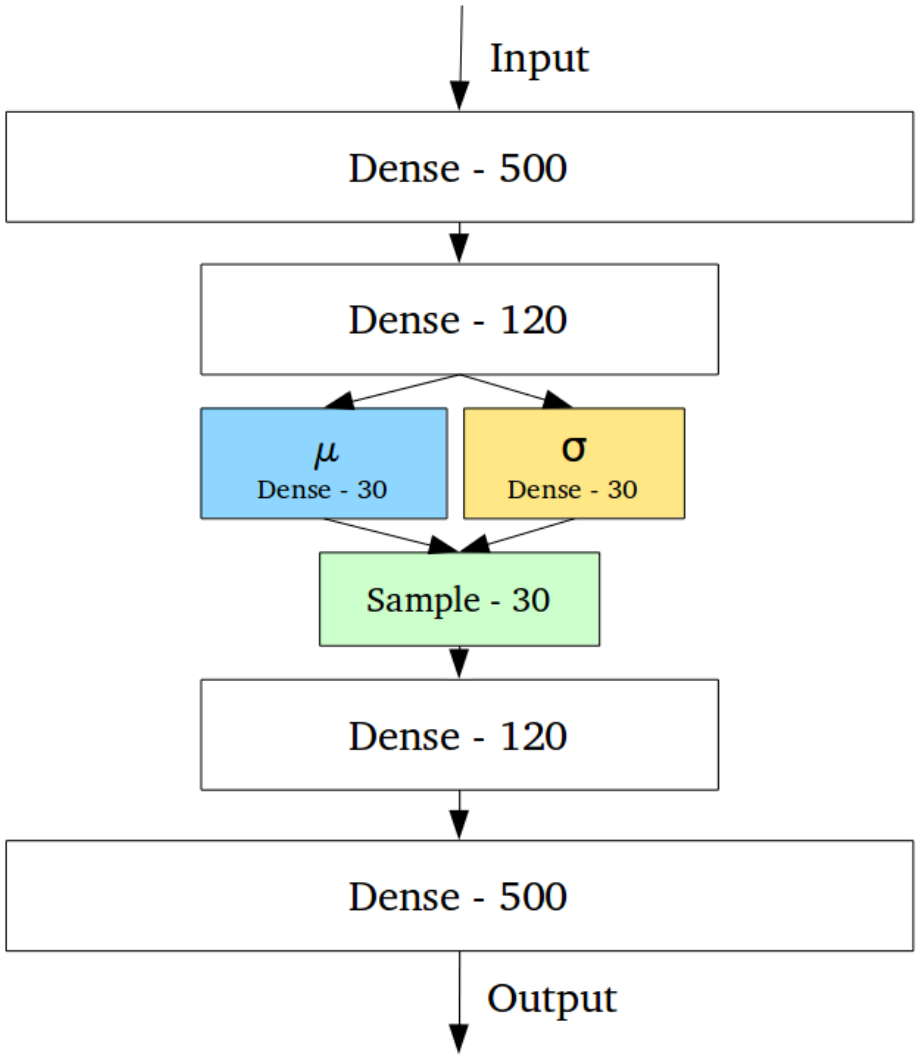
VAEs motivation: continuous local space

- This stochastic generation means, that even for the same input, while the mean and standard deviations remain the same, the actual encoding will somewhat vary on every single pass simply due to sampling.

- The decoder learns that not only is a single point in latent space referring to a sample of that class, but all nearby points refer to the same as well.

- As the decoder is exposed to a range of variations of the encoding of the same input during training, and not decode just specific encoding leaving the decodable latent space discontinuous

VAEs: Design


Input
Dense - 500
Dense - 120
μ
Dense - 30
σ
Dense - 30
Sample - 30
Dense - 120
Dense - 500
Output

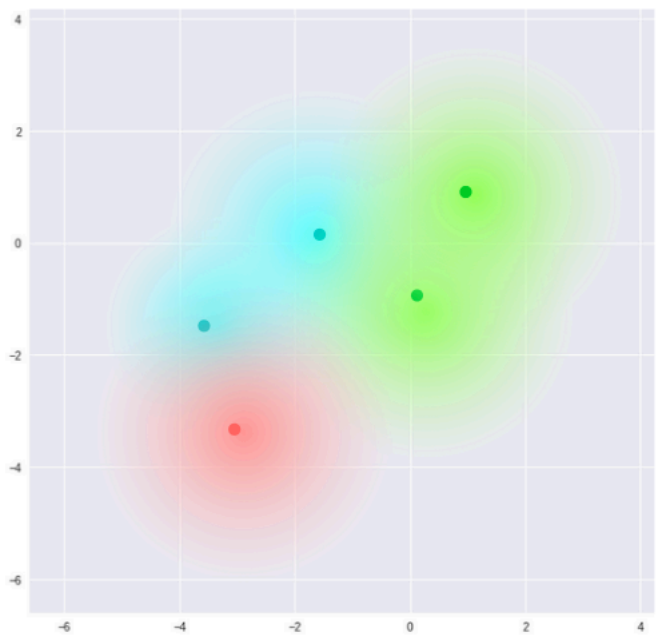
VAEs: Continuous local space

- local variation per training sample, result in smooth latent spaces on a local scale, that is, for similar samples (e.g. picture of number 2).

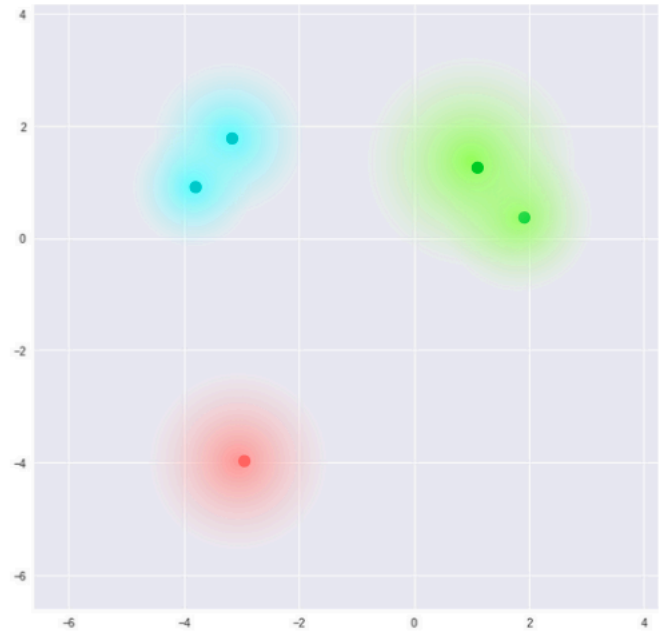
- Ideally, we want overlap between samples that are not very similar too, in order to interpolate between classes.

VAEs: Dispersed clusters

- BUT, since there are no limits on what values vectors μ and σ can take on, the encoder can learn to generate very different μ for different classes, clustering them apart, and minimize σ, making sure the encodings themselves don't vary much for the same sample
- This result in less uncertainty for the decoder and allows it to efficiently reconstruct the training data.



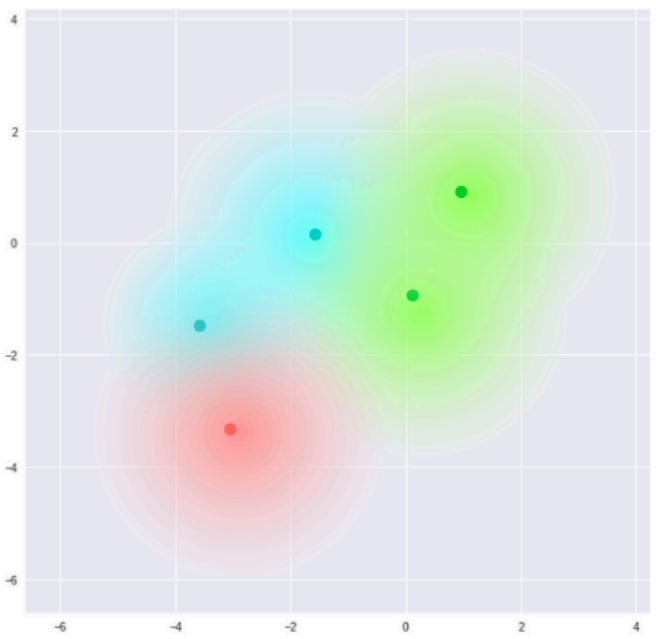
What we require



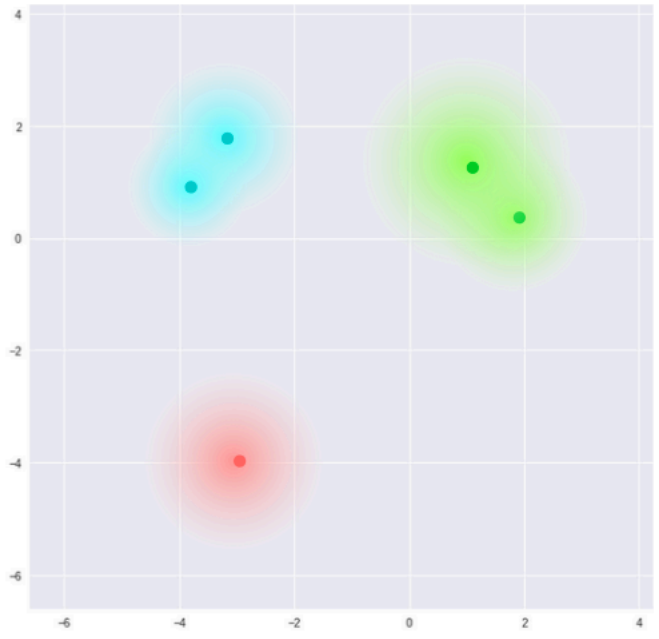
What we may inadvertently end up with

VAEs: Continuous space

- What we ideally want are encodings, all of which are as close as possible to each other while still being distinct, allowing smooth interpolation, and enabling the construction of new samples.



What we require



What we may inadvertently end up with

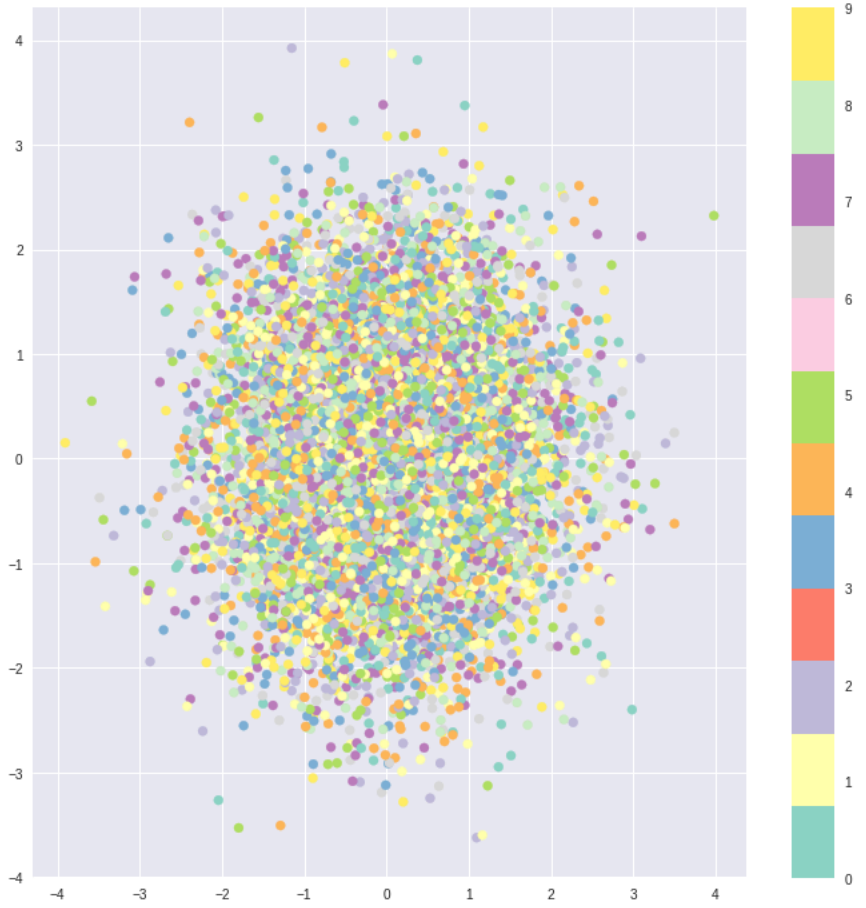
VAEs: KL loss

- In order to force this, we introduce the Kullback–Leibler divergence into the loss function.
- For VAEs, the KL loss is equivalent to the sum of all the KL divergences between the component $x_i \sim N(\mu_i, \sigma_i^2)$ in X, and the standard normal $N(0, I)$. It's minimized when $\mu_i = 0$, $\sigma_i = 1$.
- Intuitively, this loss encourages the encoder to distribute all encodings, evenly around the center of the latent space.
- If it tries to "cheat" by clustering them apart into specific regions, away from the origin, it will be penalized.

VAEs: Pure KL loss

- Using purely KL loss results in a latent space with encodings densely placed randomly, near the center of the latent space, with little regard for similarity among nearby encodings.

- The decoder finds it impossible to decode anything meaningful from this space, simply because there really isn't any meaning.



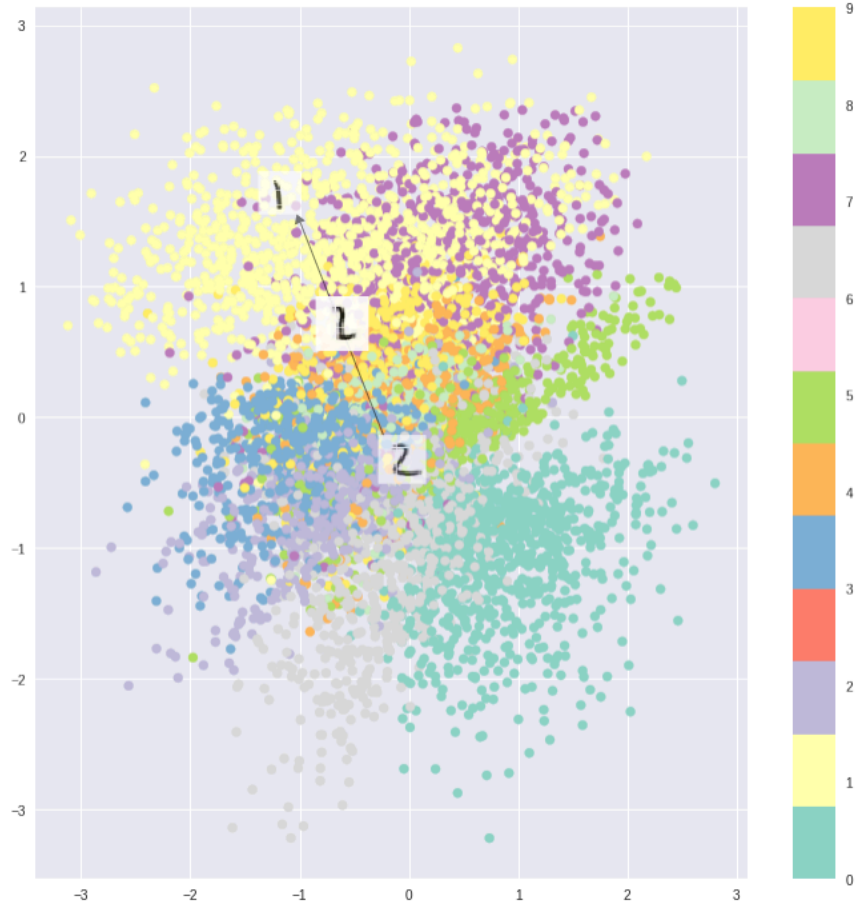
VAEs: Combined losses

- Optimizing the two together, however, results in the generation of a latent space which maintains the similarity of nearby encodings on the local scale via clustering, yet globally, is very densely packed near the latent space origin.

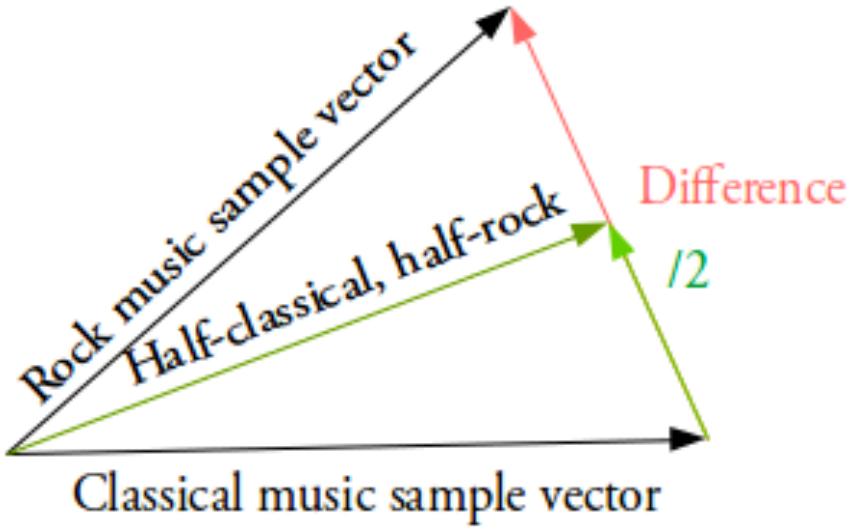
- The reconstruction loss → cluster-forming.
- KL loss → dense packing.

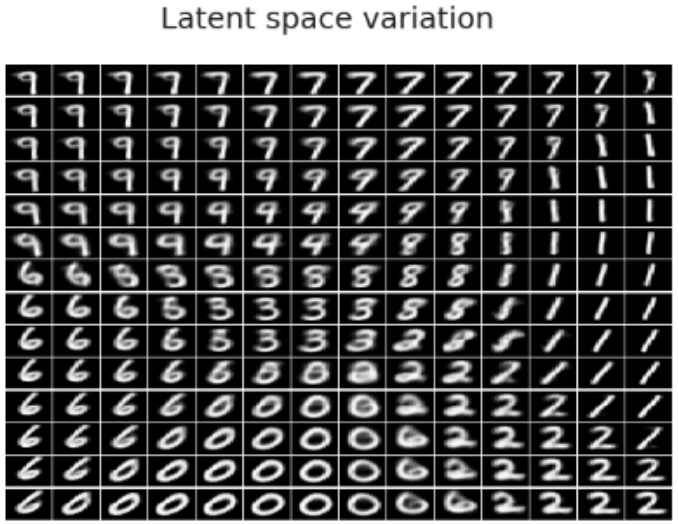
- This means If you sample a vector from the same prior distribution of the encoded vectors, $N(\mathbf{0}, \mathbf{I})$, the decoder will successfully decode it.

- And if you're interpolating, there are no sudden gaps between clusters, but a *smooth mix of features* a decoder can understand.



Vector arithmetic





Visualizing latent space

- You can use T-SNE “**T-distributed Stochastic Neighbor Embedding” algorithm to visualize your latent space distribution**

- It’s a technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions.

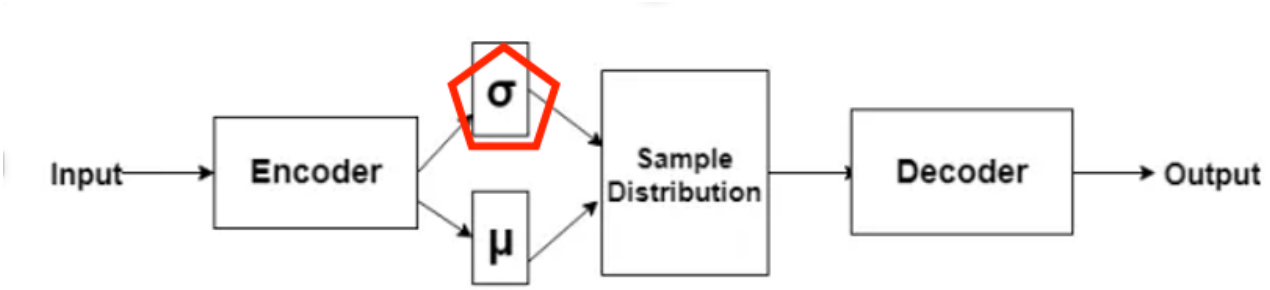
- Visualizing the latent space is another way of monitoring your VAE performance.

- Further reading
- Further examples

Training VAEs

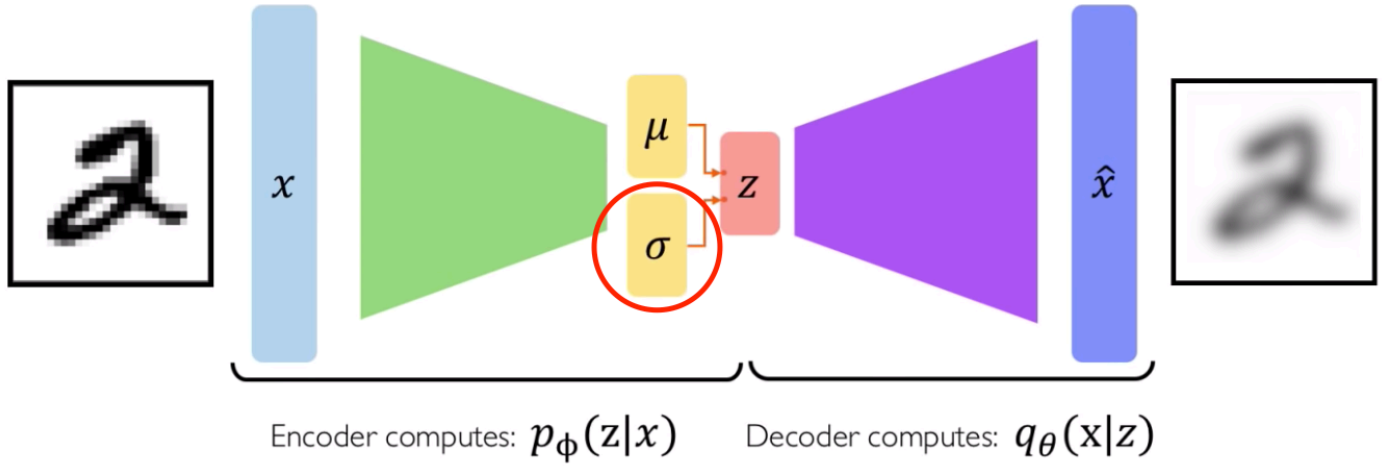
- The predicted σ is nx1 instead of n x n, is actually the diagonal of a squared covariance matrix.

- The reason behind this design is that we have no control on the values that the encoder learn and thus we can't ensure that the predicted matrix is positive semidefinite, but a diagonal matrix always satisfy this condition



Training VAEs: Predicted σ condition

- The predicted σ is nx1 instead of n x n, is actually the diagonal of a squared covariance matrix.
- The reason behind this design is that we have no control on the values that the encoder learn and thus we can't ensure that the predicted matrix is symmetric positive semidefinite, but a diagonal matrix always satisfy this condition

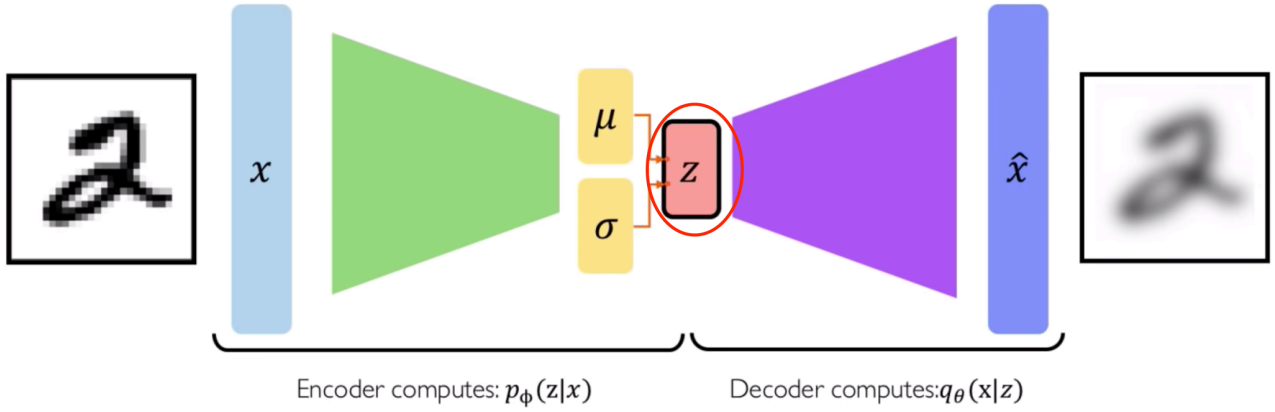


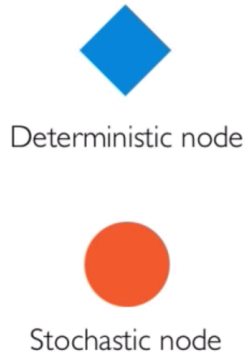
- We also assume that the predicted values are the logarithms of the diagonal (why) ?

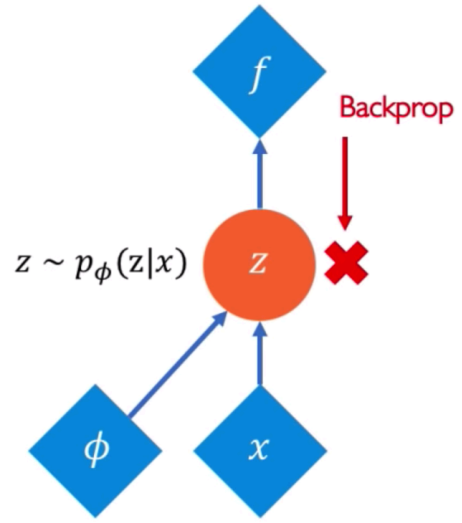
$$x^T K x = \sum_{1 \le i \le n} k_i x_i^2$$

Training VAEs: Computation graph

- We cannot backpropagate through a sampling layer!



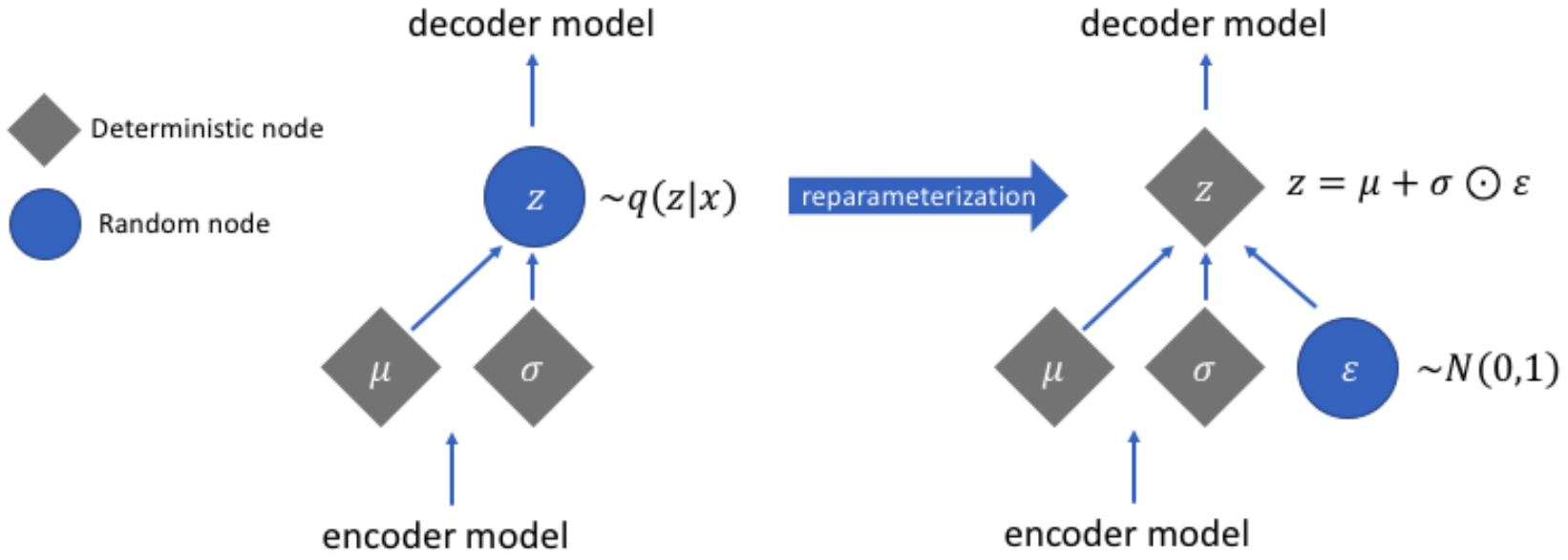




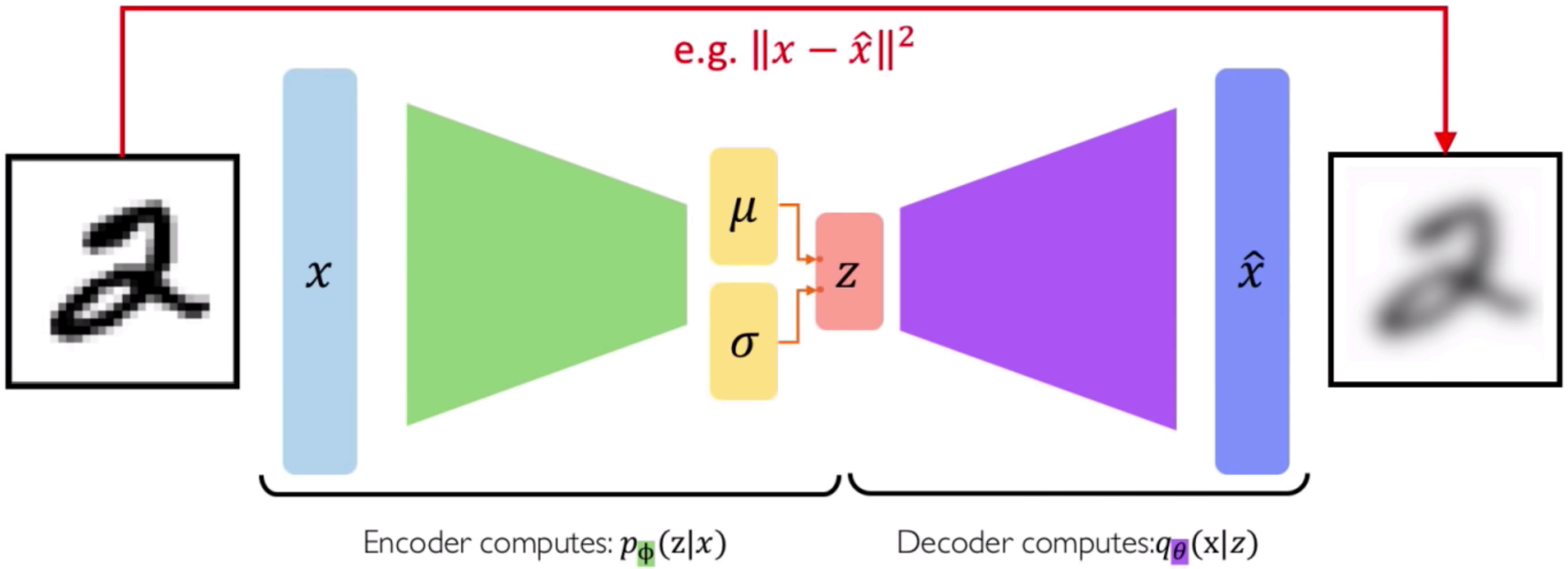
Original form

Training VAEs: Reparameterization trick

- Consider the latent variable as sum of:
  - Fixed μ, and fixed σ vector scaled by random constants drawn from the prior distribution



Training VAEs: the loss function (first term)



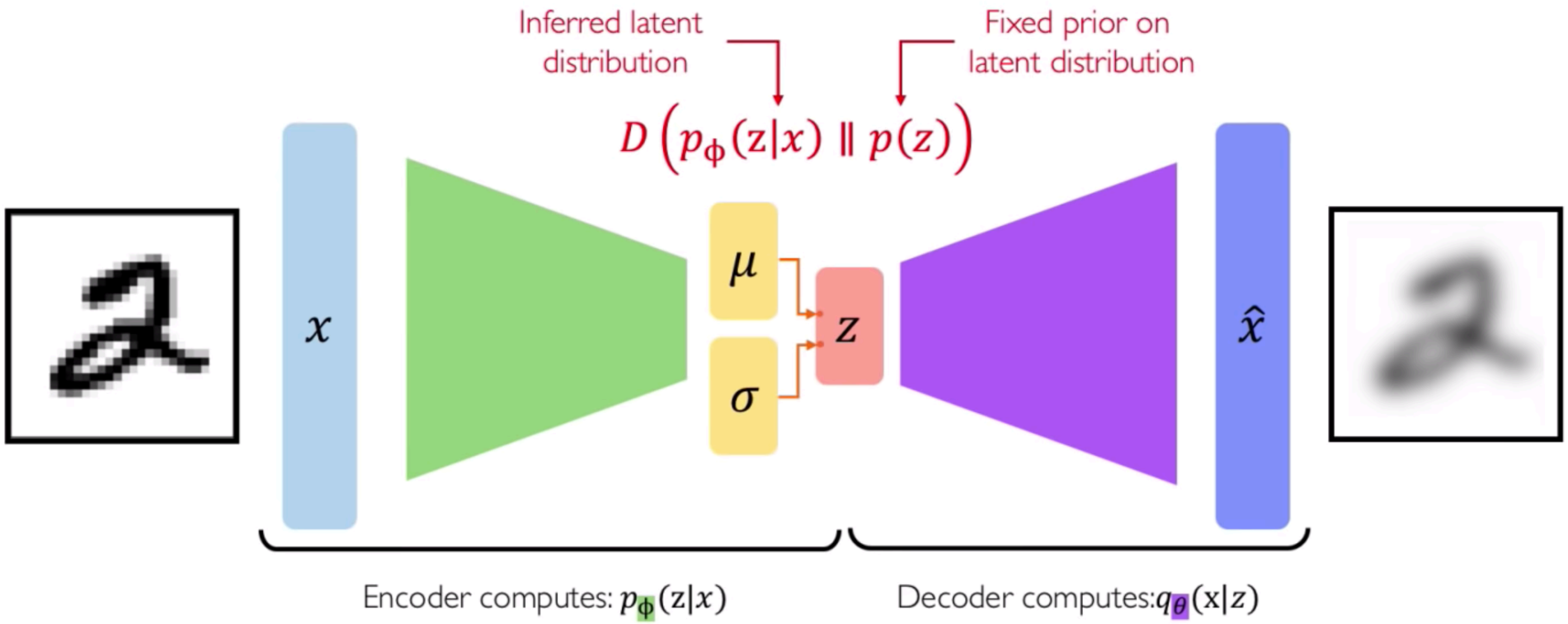
$$\mathcal{L}(\phi, \theta, x) = \boxed{\text{(reconstruction loss)}} + \text{(regularization term)}$$

Training VAEs: the loss function (first term)

- Let the encoder be $q_\theta(z|x)$ and the decoder be $p_\theta(x|z)$
- How much information is lost when we go from x to z to x˜?
- Measure this with reconstruction log-likelihood: $\log p_\theta(x|z)$
- Measures how effectively the decoder has learned to reconstruct x given the latent representation z
- The negative reconstruction likelihood

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

Training VAEs: the loss function (second term)



$$\mathcal{L}(\phi, \theta, x) = \text{(reconstruction loss)} + \boxed{\text{(regularization term)}}$$

Training VAEs: the loss function (second term)

$$KL\left(q_\theta(z|x_i)||p(z)\right) = \mathbb{E}_{z\sim q_\theta(z|x_i)}\left[\log q_\theta(z|x_i) - \log p(z)\right]$$

$$D\left(p_\phi(z|x) \parallel p(z)\right)$$

$$= -\frac{1}{2}\sum_{j=0}^{k-1}\left(\sigma_j + \mu_j^2 - 1 - \log\sigma_j\right)$$

KL-divergence between the two distributions

- Proof ?

Disentangled features

- The goal of disentangled features can be most easily understood as wanting to use each dimension of your latent *z* code to encode one and only one of these underlying independent factors of variation.

- For example a disentangled representation would represent someone's height and clothing as separate dimensions of the *z* code.

- Disentangled representation is useful because when you capture the most meaningful or salient ways that observations differ from one another

Beta VAEs for disentangled features

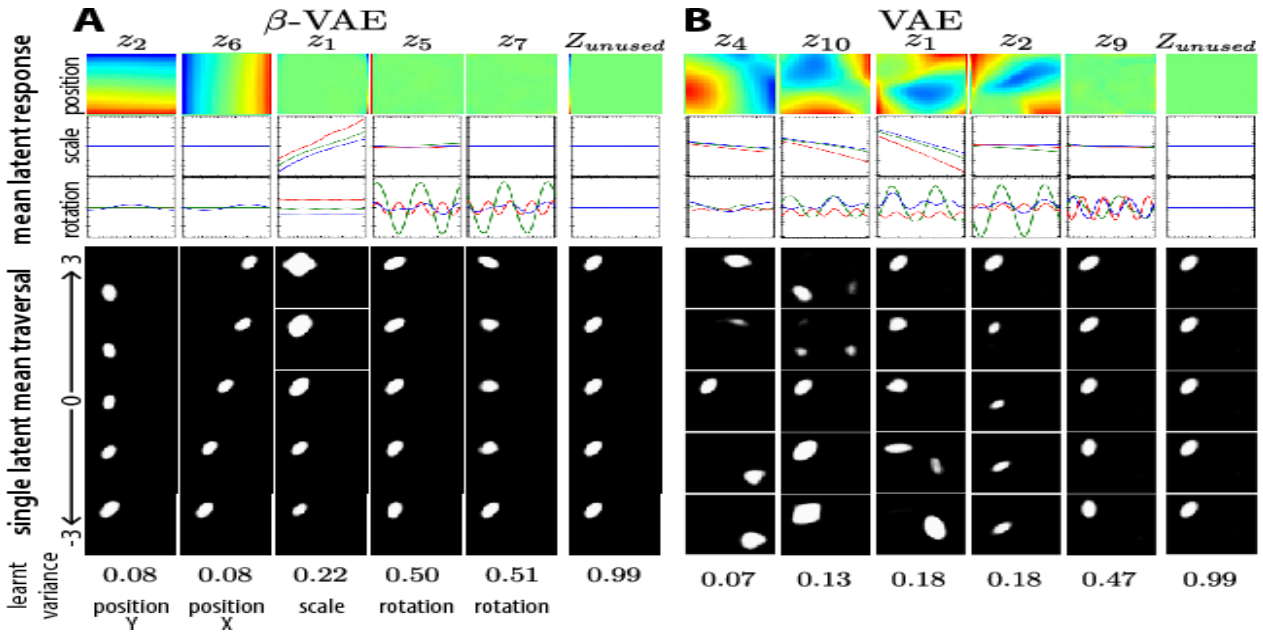
- Extra weight parameter on the regularization term

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \beta\, D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\big)$$

- If Beta goes high that would result in:
  - **smoothness:** It incentivizes the reconstructions to change smoothly as you vary values of *z*, rather than being jumpy or discontinuous
  - **Parsimony:** It incentivizes the network to compress information about X into as few *z* dimensions as possible
  - **Axis-Alignment:** It incentivizes to align the main axes of data variability with the dimensions of the *z* vector

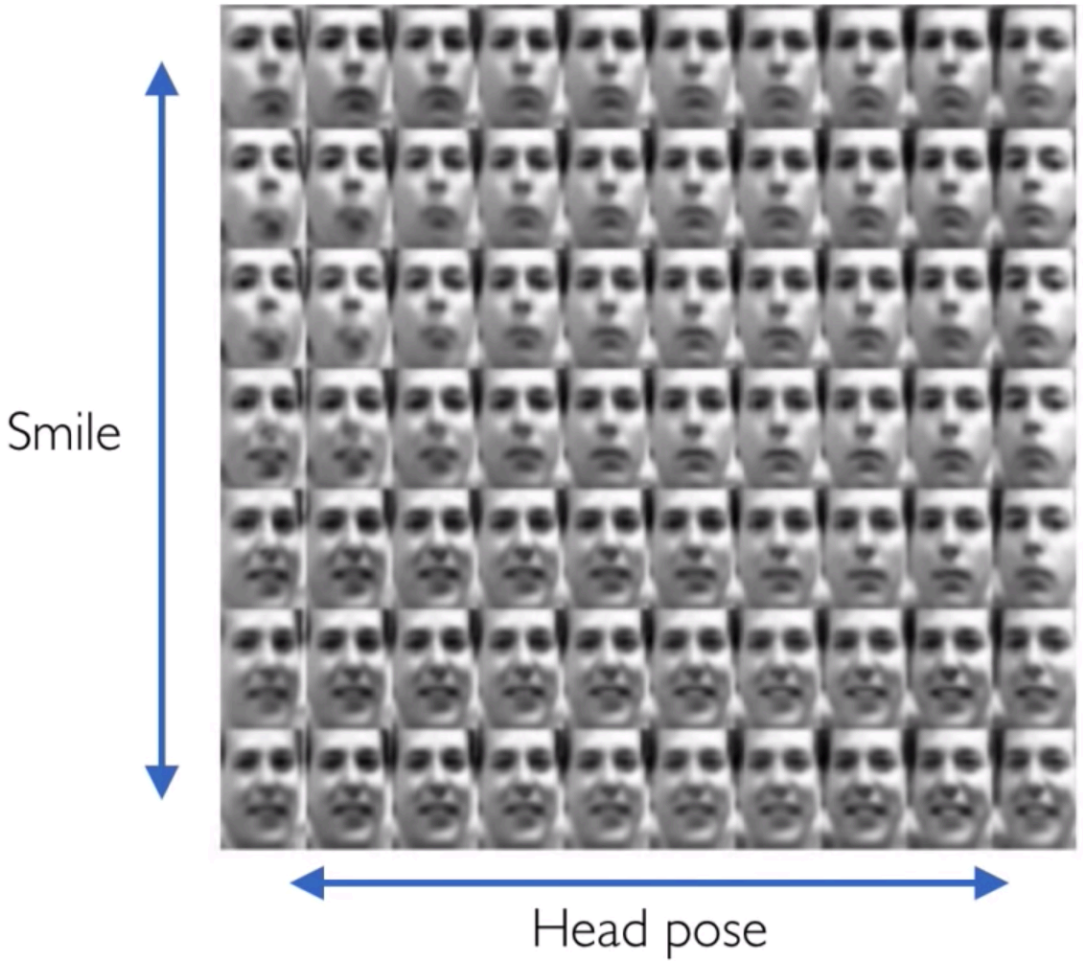
Altogether, these incentives push the network to compress information about X into a small number of smoothly varying, independent dimensions.

Disentangled features



Disentangled features

- Beta is hyper parameter (why?)
- If Beta goes very low, then you give your model too much freedom and thus might not learn useful representations which yields in poor generalization
- If Beta goes very high, then your latent code might encode high level features and yields in poor generation quality (no HD details)
- You can play with more samples here artbreeder.com



Debiasing generative models

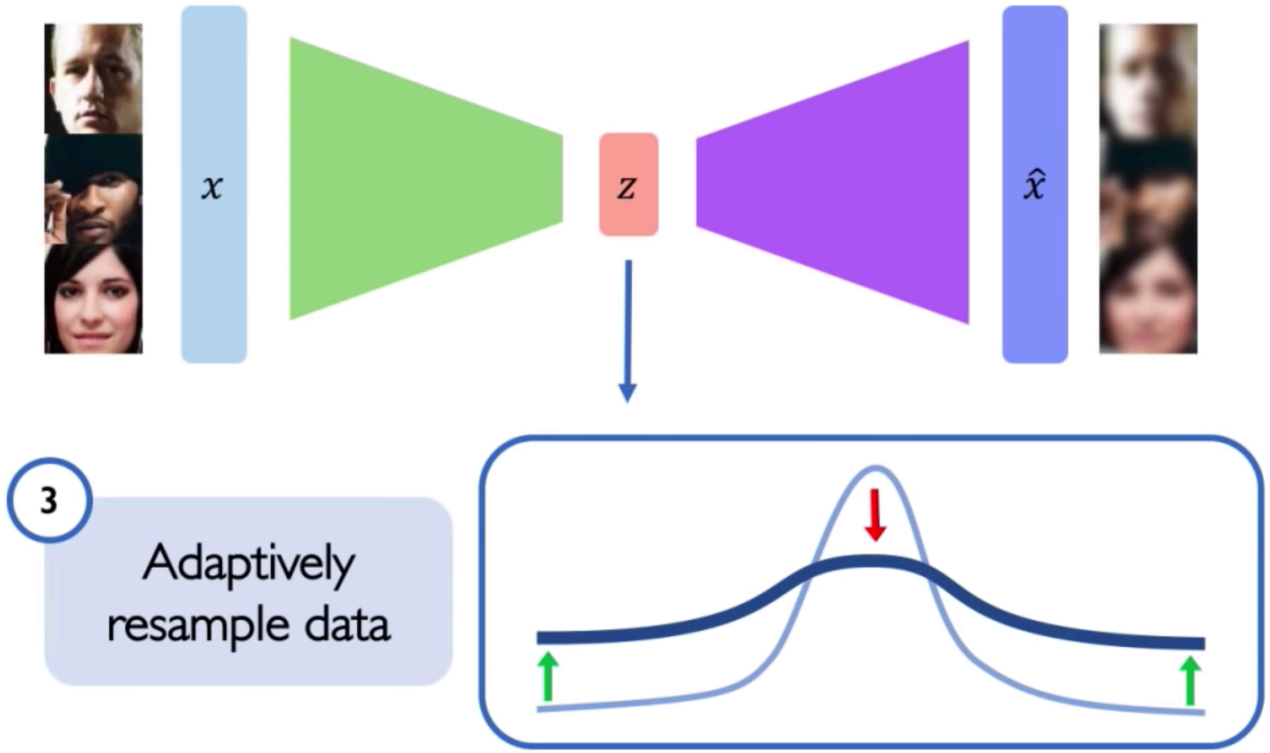
- How can we use latent distributions to create fair and representative datasets ?

Homogeneous skin color, pose

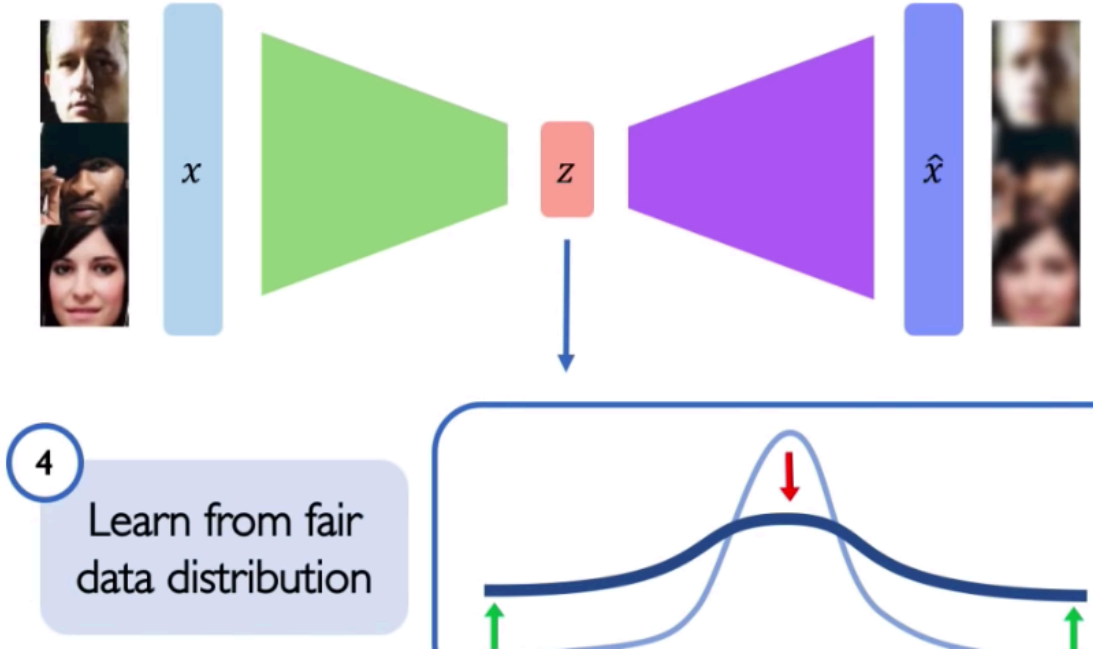
VS

Diverse skin color, pose, illumination

Debiasing: learned latent structure



Debiasing: fair representative datasets



Latent distributions used to create fair and representative dataset

Applications of generative VAEs

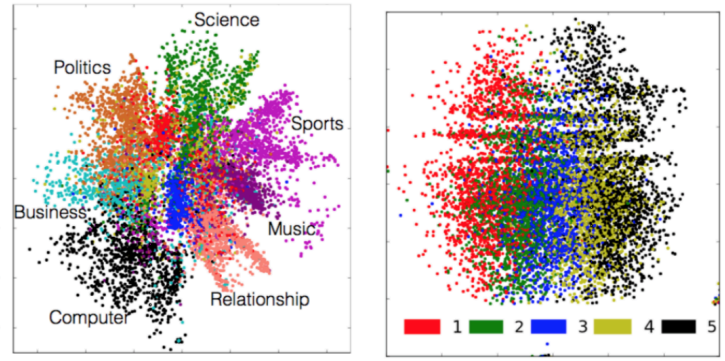
- Convolutional VAEs e.g. generate faces
- Training RL agents of representations instead of full input space
- Google Brain's Magenta's MusicVAE

Applications of generative VAEs

- Caption generation

Figure 2: Examples of generated caption from unseen images on the validation dataset of ImageNet.

- Unsupervised document classification

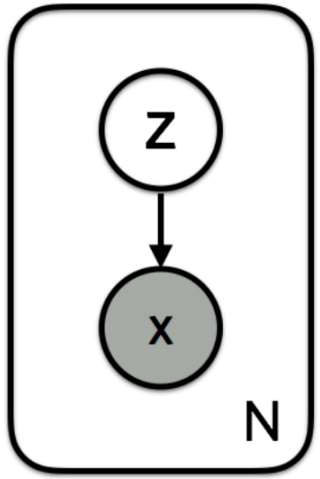


(a) Yahoo (b) Yelp

Figure 3: Visualizations of learned latent representations.

Variational Inference: probabilistic model perspective (optional)

- Data x and latent variables z
- Joint pdf of the model: $p(x, z) = p(x|z)p(z)$
- Decomposes into likelihood: $p(x|z)$, and prior: $p(z)$
- Generative process:
  Draw latent variables $z_i \sim p(z)$
  Draw datapoint $x_i \sim p(x|z)$
- Graphical model:



Variational Inference: probabilistic model perspective (optional)

- Suppose we want to do inference in this model
- We would like to infer good values of z, given observed data
- Then we could use them to generate real-looking MNIST digits
- We want to calculate the posterior:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

- Need to calculate evidence: $p(x) = \int p(x|z)p(z)dz$
- Integral over all configurations of latent variables ☹
- Intractable

Variational Inference: probabilistic model perspective (optional)

- Variational inference to the rescue!
- Let's approximate the true posterior $p(z|x)$ with the 'best' distribution from some family $q_\lambda(z|x)$
- Which choice of λ gives the 'best' $q_\lambda(z|x)$?
- KL divergence measures information lost when using q_λ to approximate p
- Choose λ to minimize $KL\big(q_\lambda(z|x)||p(z|x)\big) = KL\big(q_\lambda||p\big)$

Variational Inference: probabilistic model perspective (optional)

-

$$\begin{aligned} KL(q_\lambda || p) &:= \mathbb{E}_{z\sim q_\lambda}\left[\log q_\lambda(z|x) - \log p(z|x)\right] \\ &= \mathbb{E}_{z\sim q_\lambda}\left[\log q_\lambda(z|x)\right] - \mathbb{E}_{z\sim q_\lambda}\left[\log p(x,z)\right] \\ &\quad + \log p(x) \end{aligned}$$

- Still contains $p(x)$ term! So cannot compute directly
- But $p(x)$ does not depend on λ, so still hope

Variational Inference: probabilistic model perspective (optional)

- Define **E**vidence **L**ower **BO**und:

$$ELBO(\lambda) := \mathbb{E}_{z\sim q_\lambda}\big[\log p(x,z)\big] - \mathbb{E}_{z\sim q_\lambda}\big[\log q_\lambda(z|x)\big]$$

- Then

$$\begin{aligned} KL\big(q_\lambda||p\big) &= \mathbb{E}_{z\sim q_\lambda}\big[\log q_\lambda(z|x)\big] - \mathbb{E}_{z\sim q_\lambda}\big[\log p(x,z)\big] + \log p(x) \\ &= -ELBO(\lambda) + \log p(x) \end{aligned}$$

- So minimizing $KL\big(q_\lambda||p\big)$ w.r.t. λ is equivalent to maximizing $ELBO(\lambda)$

Variational Inference: probabilistic model perspective (optional)

- Since no two datapoints share latent variables, we can write:

$$ELBO(\lambda) = \sum_{i=1}^{N} ELBO_i(\lambda)$$

- Where

$$ELBO_i(\lambda) = \mathbb{E}_{z \sim q_\lambda(z|x_i)}\big[\log p(x_i, z)\big] - \mathbb{E}_{z \sim q_\lambda(z|x_i)}\big[\log q_\lambda(z|x_i)\big]$$

Variational Inference: probabilistic model perspective (optional)

- We can rewrite the term $ELBO_i(\lambda)$:

$$
\begin{aligned}
ELBO_i(\lambda) &= \mathbb{E}_{z\sim q_\lambda(z|x_i)}\big[\log p(x_i,z)\big] - \mathbb{E}_{z\sim q_\lambda(z|x_i)}\big[\log q_\lambda(z|x_i)\big] \\
&= \mathbb{E}_{z\sim q_\lambda(z|x_i)}\big[\log p(x_i|z) + \log p(z)\big] \\
&\quad - \mathbb{E}_{z\sim q_\lambda(z|x_i)}\big[\log q_\lambda(z|x_i)\big] \\
&= \mathbb{E}_{z\sim q_\lambda(z|x_i)}\big[\log p(x_i|z)\big] \\
&\quad - \mathbb{E}_{z\sim q_\lambda(z|x_i)}\big[\log q_\lambda(z|x_i) - \log p(z)\big] \\
&= \mathbb{E}_{z\sim q_\lambda(z|x_i)}\big[\log p(x_i|z)\big] - KL\big(q_\lambda(z|x_i)||p(z)\big)
\end{aligned}
$$

Variational Inference: probabilistic model perspective (optional)

- How do we relate λ to ϕ and θ seen earlier?
- We can parameterize approximate posterior $q_\theta(z|x,\lambda)$ by a network that takes data x and outputs parameters λ
- Parameterize the likelihood $p(x|z)$ with a network that takes latent variables and outputs parameters to the data distribution $p_\phi(x|z)$
- So we can re-write

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

Variational Inference: probabilistic model perspective (optional)

- Recall the Deep Learning objective derived earlier. We want to minimize:

$$L(\theta, \phi) = \sum_{i=1}^{N} \Big(- E_{z \sim q_\theta(z|x_i)} \big[\log p_\phi(x_i|z) \big] + KL\big(q_\theta(z|x_i) || p(z)\big) \Big)$$

- The objective just derived for the Probabilistic Model was to maximize:

$$ELBO(\theta, \phi) = \sum_{i=1}^{N} \Big(\mathbb{E}_{z \sim q_\theta(z|x_i)} \big[\log p_\phi(x_i|z) \big] - KL\big(q_\theta(z|x_i) || p(z)\big) \Big)$$

- They are equivalent!

References

- Irhum Shafkat, intuitively-understanding-variational-autoencoders.
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville.*Deeplearning*.MITpress,2016.
- 6S191_MIT_DeepLearning, deep generative modeling.
- Devon Graham, Variational Autoencoders - An Introduction, University of British Columbia
- Cody Marie Wild, what-a-disentangled-net-we-weave-representation-learning-in-VAEs