

Deep Generative Modeling

used to create synthesize data

So far Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn function to map
 $x \rightarrow y$

Examples: Classification, regression,
Object detection, semantic segmentation

This topic: UnSupervised Learning

Data: \mathcal{X}

x is data, no labels!

Goal: Learn the hidden or underlying structure of data

E.g. Clustering, dimensionality reduction, allow insight to fundamental structure of data. This can be used to generate data.

Generative Modeling (Unsupervised Learning)

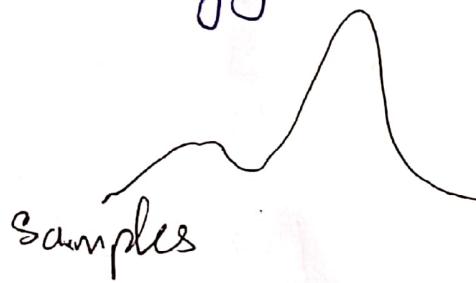
Goal: Take as input training samples from some distribution and learn a model that represents distribution of the model

1) Density Estimation:

Given a set of data samples they fall according to some density.

The task of GAN is to learn the underlying PDF that describe how this data fall.

along this distribution



We can understand PDF and we can use this information to generate new synthetic samples

We are considering some input examples that fall and are drawn from same training data distribution

generate synthetic examples

2) Sample Generation	Input Samples	Generated Samples
Training data $\sim P_{\text{data}}(x)$		Generated $\sim P_{\text{model}}(x)$

How can we learn $P_{\text{model}}(x)$ similar to $P_{\text{data}}(x)$?

GANs enable us automatically uncover underlying features in a dataset.

How these features are distributed within a dataset.

Question: How can we learn probability distribution using our model.

GANs Applications:

1) automatically uncover underlying features in a dataset.

Given a facial dataset. We need to group according features (skin color). Dataset may be biased. Data can be sampled accordingly.

GANs

2) Outlier detection:

Self driving car:

Resultant model can be better equipped.

Problem: How can we detect when we encounter something new

Strategy: detect outliers using GAN
Use outliers during training.

Lateral

Latent Variable:

A variable which is not directly measured but it is inferred e.g. happiness or quality of life.

Can we learn the true explanatory factors e.g. latent variables.

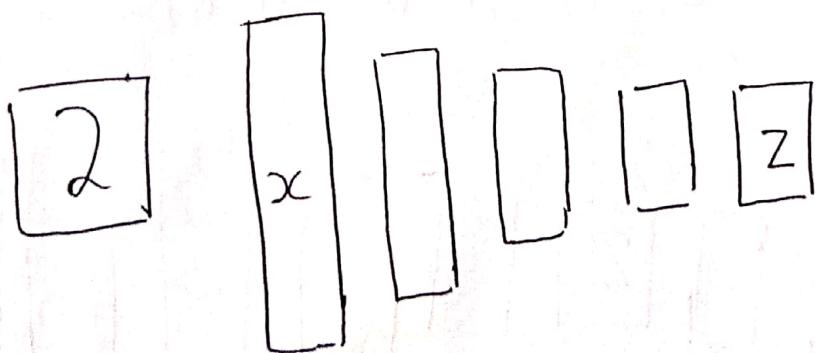
find ways of actually Learning hidden latent variables.

NN can handle multi-dimensional datasets and learn combinations of non-linear functions. That can approximate complex data distributions.

A. Simple Model: which can learn by self-encoding the input Auto Encoders

Auto Encoders:

Unsupervised approach for learning a lower-dimensional feature representation from unlabelled training data



Encoder learns mapping from the data x , to a Low-dimensional Latent Space, z

Latent Cave example:

shadows
Why do we ~~need~~ ^{need to ensure that z has} about low-dimensional
data z ?

Encoder: Mapping (data) to a low-dimension latent data z .

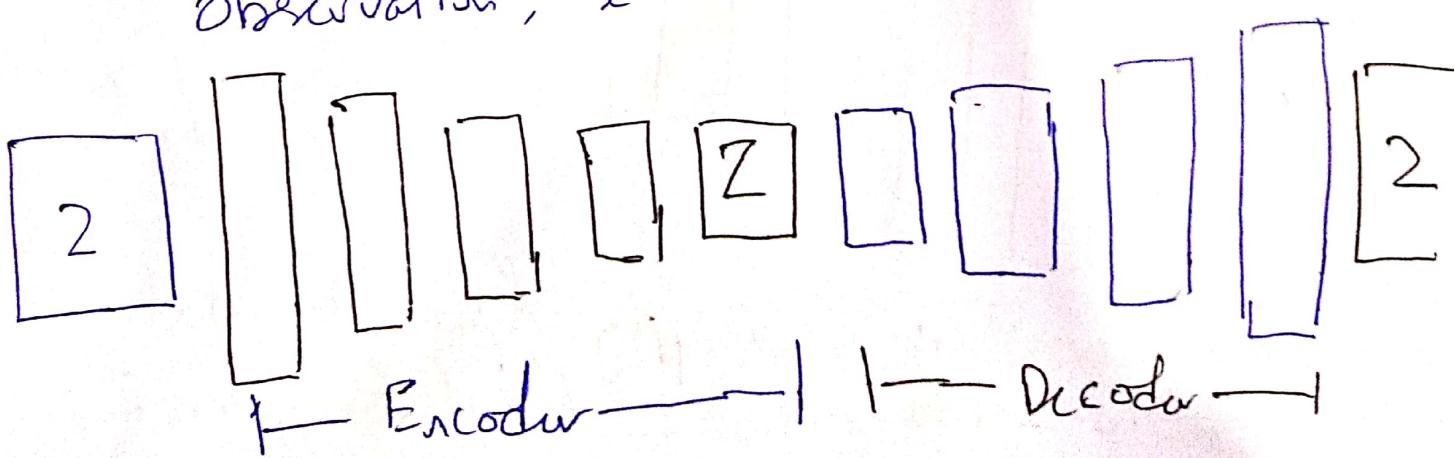
Low dimensions: ask to compress into a small latent factor, we can learn compact feature representation.

How can we train?

Data is unlabelled.

- Use a Decoder
- Train the model to use these features to reconstruct the original data.

Decoder: Learns mapping back from latent space, z , to a reconstructed observation, \hat{x}



Decoder portion of the Auto Encoder
is going to be a series of NN layers.
Take this hidden layer and map it
back to the hidden space.

$\hat{x} \rightarrow$ estimated

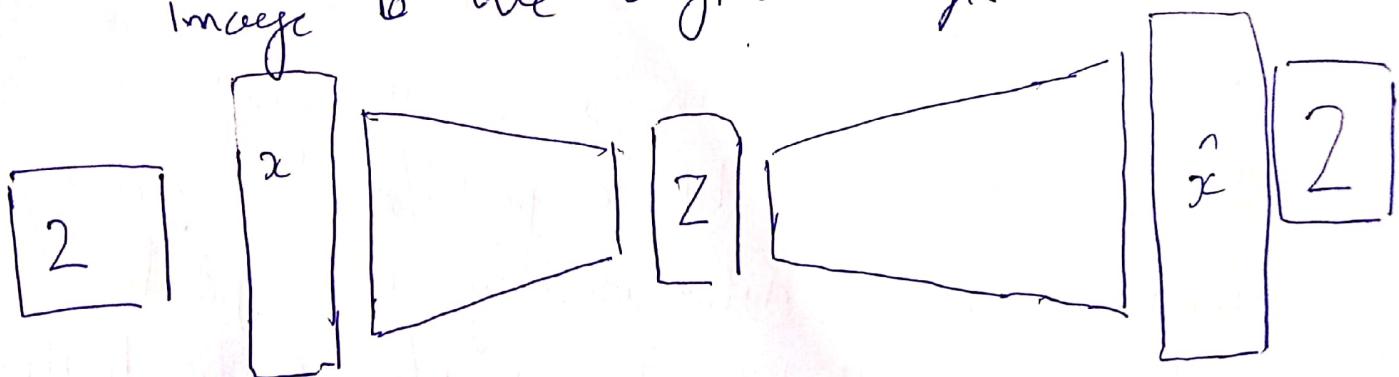
$$L(x, \hat{x}) = \|x - \hat{x}\|^2$$

Loss function
does not use
any labels.

loss function: RMSE

for images, subtract one image from
the another and squaring the difference
pixel-wise difference of images.

Measure how close our reconstructed
image to the original image.



we can not observe 'z'

Dimensionality is important in re-construction
of input.

Auto encoding is a form of compression
Smaller latent space will force a larger
training bottleneck.

2D latent space might have higher loss as compared to 5D

Information bottleneck

Lower dimensions → poor quality

Auto encoders for representation Learning

Auto encoder ~~complexity~~ can limit nodes in Bottleneck hidden layer: forces network to learn a ~~compressed~~ compressed latent representation.

without presence of bottleneck Layer; network could simply learn to memorize the input values.

Reconstruction Loss: forces the latent

representation to capture (or encode)

as much "information" about the data

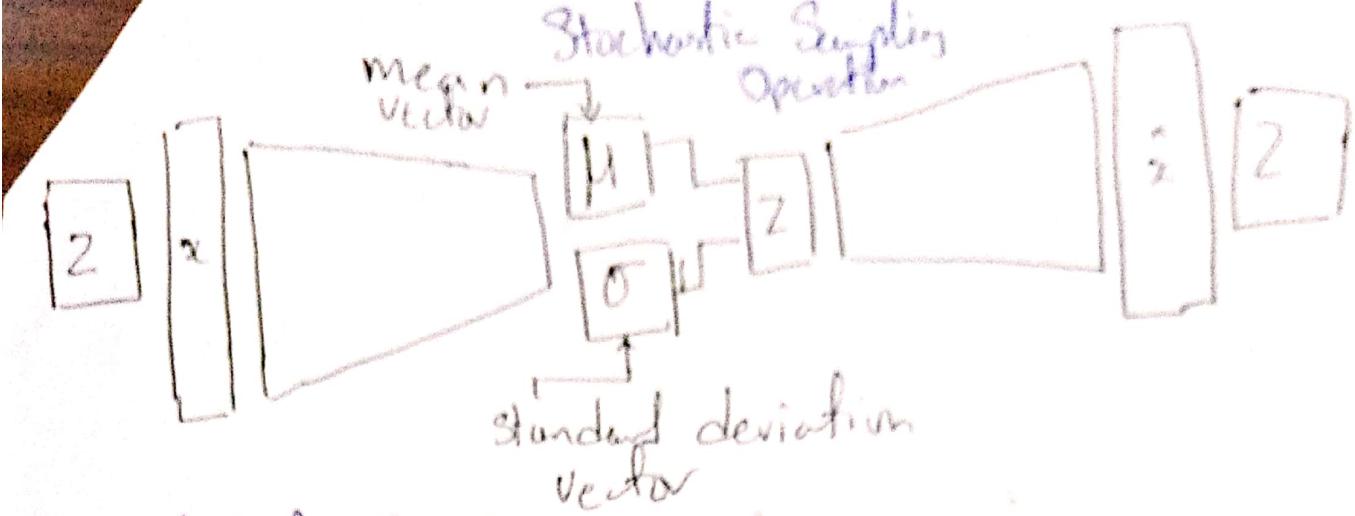
as possible. Diff b/w original i/p and encoded i/p

Auto encoding = Automatically encoding data
~~that autoencoder holds variations in i/p without holding~~

VARIATIONAL Auto encoders to redundancies

impose a variational stochastic twist

- generate smoother representations of the input data
 - improve the quality of reconstruction
 - generate new images similar to the input dataset. (not direct reconstruction of the i/p data)
- stochastic sampling operation



Ident Auto Encoder.

- Sensitive to the inputs to build reconstruction
 - Insensitive to the inputs that the model does not memorize or overfit training data
- $h(x, z^*) + \text{regularizer} \rightarrow$ discovery memorization

Instead of the latent variable directly

VAEs learn the mean and variance associated with that variable. They parameterized association for that latent variable

instead of learning a vector of latent variables, we learn a vector of mean and standard deviation of latent variables and define

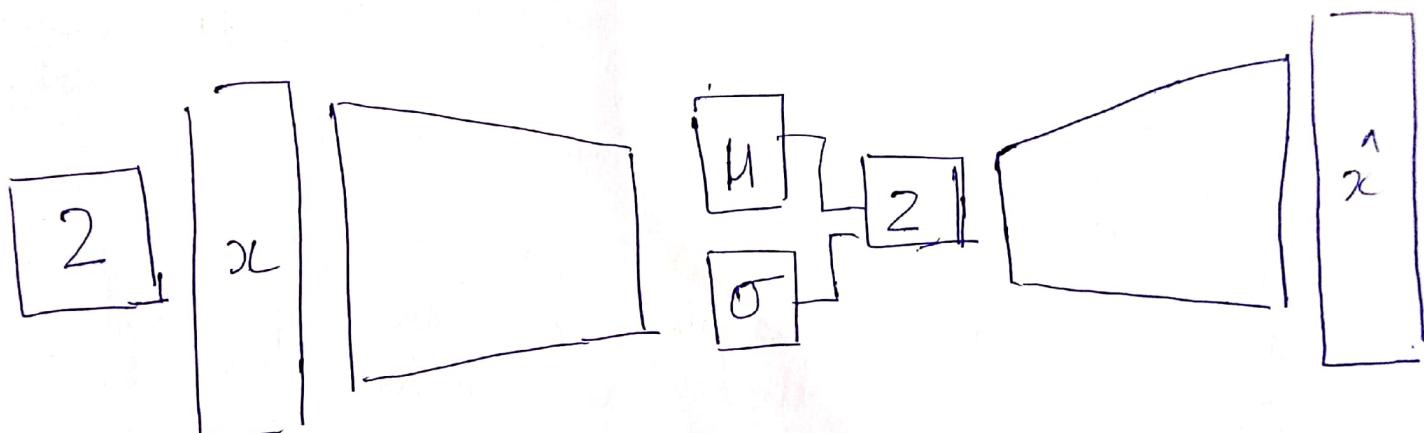
They parameterized probability distribution for that variable.

Traditional Auto Encoders are deterministic. O/P^{is} single value for each encoding dimension.

variational autoencoders are a probabilistic twist on autoencoders. A probability distribution for each latent variable. Sample from the mean and standard deviation to compute the latent sample.

probabilistic twist to compute sample information

Encoder and decoder are now probabilistic



Encoder computes $q_{\phi}(z|x)$ Decoder computes
 $p_{\theta}(x|z)$

Performing Sampling information to
sample each of the latent variable

Encoder is now try to learn a PDF
of latent distribution z given the
data x .

Decoder will try to learn is going

to take that learned representation
and compute a new PDF of input x^*
given the latent distribution z .

Encoder and decoder are defined by
separate sets of weights (ϕ, θ)

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

reconstruction loss is going to capture the
difference between the input and

reconstructed output.

e.g. log likelihood $\cdot \|x - \hat{x}\|^2$

For image data

MSE b/w o/p and
i/p

regularization loss (VAE loss):

$q_{\phi}(z|x)$ is a distribution on the latent
space z given the data x .

As part of this learning process,
we are going to place a prior on
latent learning space z .

Some initial hypothesis what z will look
like.

By imposing these regularization terms
the model will achieve try to enforce
that it learns to follow this
prior distribution ($p(z)$)

$$D(q_{\phi}(z|x) \parallel p(z))$$

regularization term
will try to enhance ~~this diversion~~^{min}
minimization of this diversion b/w
what encoder is trying to infer the
probability distribution of $z|x$
and the prior on the latent variable
 $p(z)$.

avoid overfitting on latent space
encourage latent variables to adopt
a distribution which is similar to
 $p(z)$; prior.

Regularization: or pricing π is important

consider the divergence b/w inferred latent distribution and fixed prior.

$$D(q_{\phi}(z|x) \parallel p(z))$$

↑ ↑
Inferred latent distribution Fixed prior on latent distribution

Choice of prior:

Common choice of prior — Normal Gaussian

$$p(z) = N(\mu=0, \sigma^2=1)$$

S.D.
or variance = 1
Normal distribution centered around mean 0

- Encourages encodings to distribute encodings evenly around the center of the latent space
- Penalize the network when it tries to cheat by clustering points in specific regions. (i.e. by memorizing the data).

under

distribution learned by \uparrow VAE will
be distributed evenly around the
center of each latent variable.
outside the region far away
will be penalized.

Inferred latent distribution $\xrightarrow{\text{fixed prior on latent distribution}}$

$$D(g_{V\phi}^{\downarrow}(z|z) \parallel p(z)) = \text{KL-divergence}$$

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

b/w the two distributions

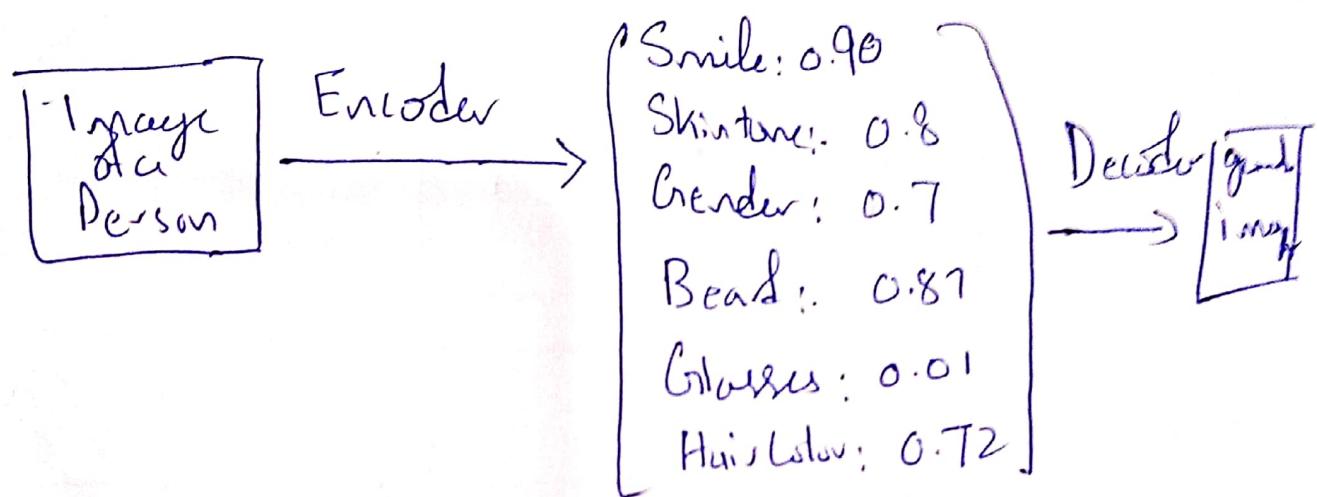
KL-Divergence:

A measure that how
one probability distribution is
different from another.

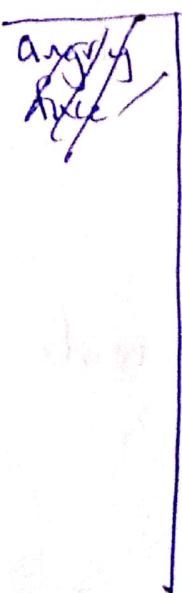
Lecture vs.

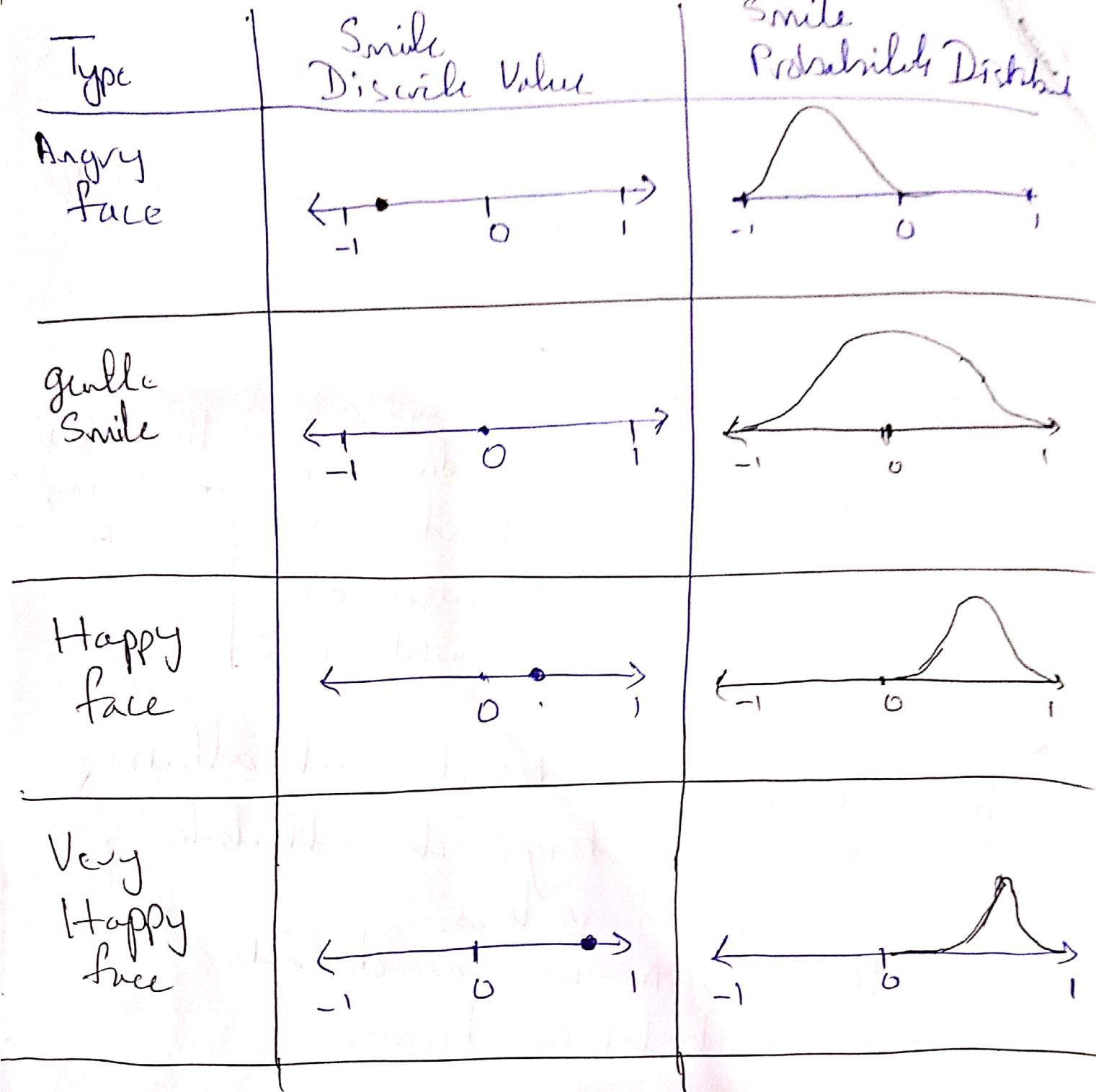
Auto Encoder Vs. VAE

Auto encoder will learn descriptive attributes.
Example



- A single value to describe each attribute
How about representing each attribute as a range of values.
Using VAE, we can describe using probabilistic terms.





Each latent attribute is represented as probability distribution.

When decoding: randomly sample from each latent state distribution to generate a vector as input.

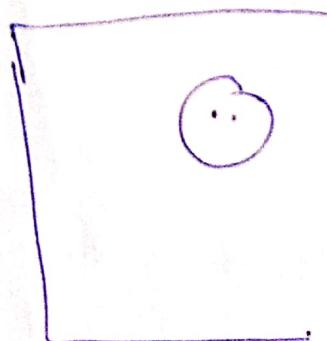
Why we want to regularize?
What is regularization.

What properties do we want from regularization?

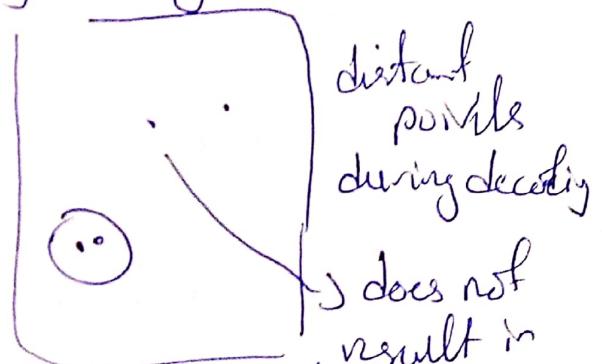
1. Continuity: Points that are close in latent space \rightarrow similar content after decoding

2. Completeness: Sampling from latent space \rightarrow meaningful content after decoding

* Without regularization: Points in latent space could be close but they may not get decoded correctly or they may be far at a distance in the latent space



Points close in latent space are meaningfully decoded.



distant points during decoding
does not result in
Not Regularized meaningful content.

Stochastic Sampling:

Every point in the signal has a non-zero probability of being sampled.

How can we achieve this
VAEs try to encode inputs as distribution
which are defined by mean and variance.

Mean and S.D. alone are not sufficient

Recall: Loss = Reconstruction + Regularization

Without regularization: the model
will try to minimize the
reconstruction loss

means could be converged or totally
diverged.

Normal prior will encourage learned latent
distributions to overlap in latent space.

Normal prior based regularization
helps enforce information gradient
in the latent space.

Points and distances in the latent space
have some relationship with the content
reconstructed

Trade off between regularizing and
reconstructing. The more we generalize
there is a risk of suffering w.r.t quality

How do we back propagate?

It requires deterministic approach.
idea is

Reparameterizing the sampling layer.

$$z \sim N(\mu, \sigma^2)$$

We cannot compute gradients through stochastic sampling.

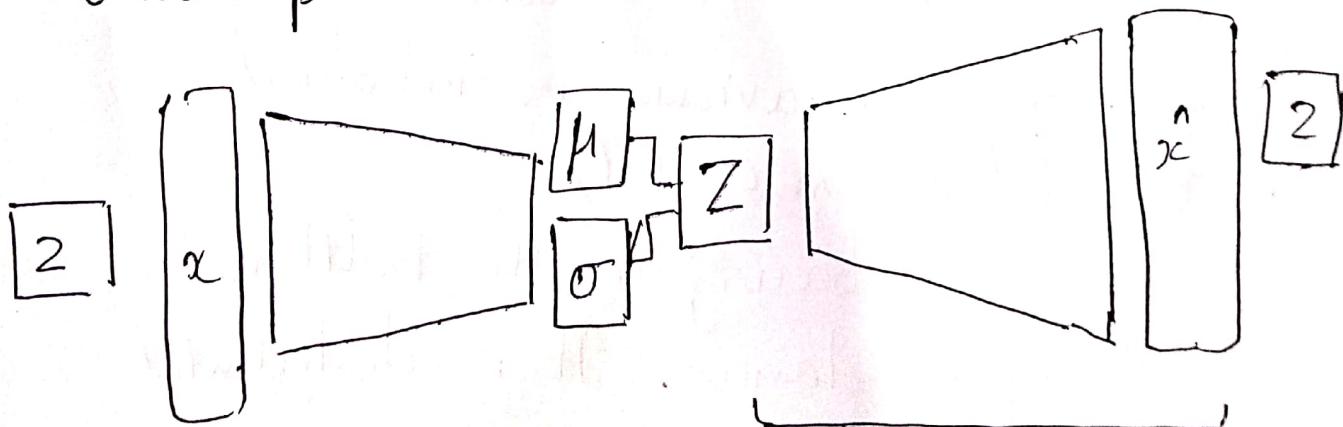
Consider the sampled latent vector z as a sum of

- a fixed vector μ
- a fixed σ vector

scaled by random coeffs. from the prior distribution

$$\Rightarrow z = \mu + \sigma \odot \epsilon \quad \text{where } \epsilon \sim N(0, 1)$$

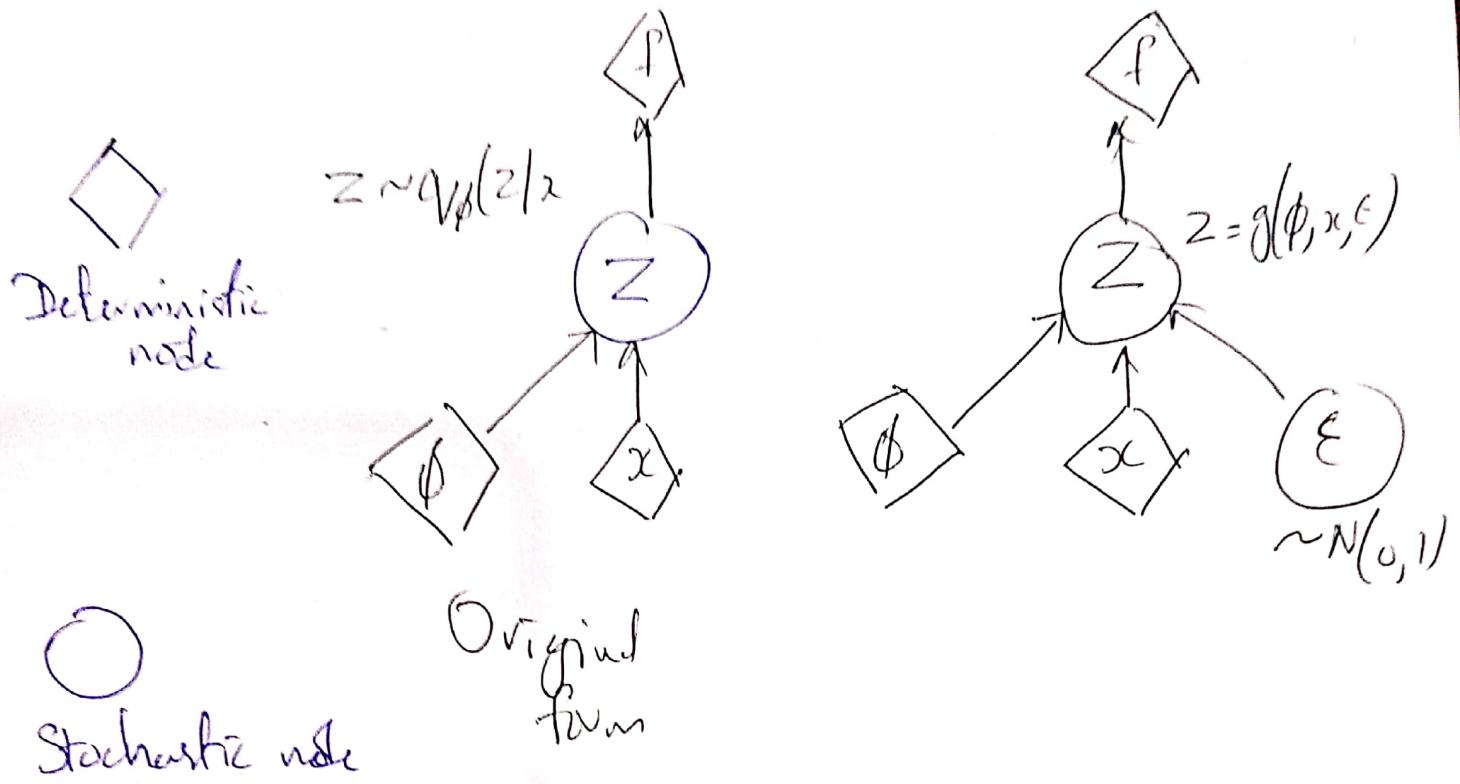
Forward pass:



Encoder computes
 $p_\phi(z|x)$

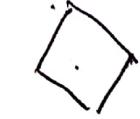
Decoder computes
 $p_\theta(x|z)$

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



Back propagation requires
deterministic nodes.

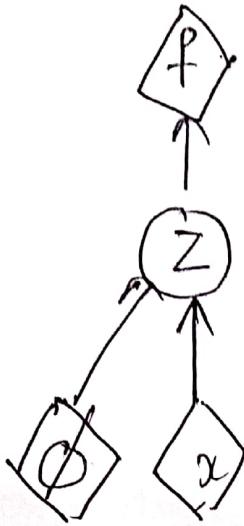
Reparameterizing use $Z = \mu + \sigma \circ \epsilon$ where $\epsilon \sim N(0, 1)$
stochastic behaviour is introduced
by ϵ (random constant).
 ϵ is not occurs in the bottleneck
sampling layer. It is distributed
elsewhere.



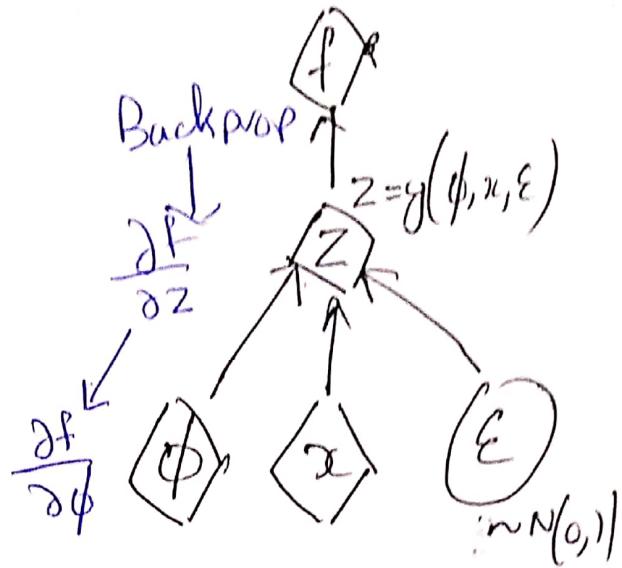
Deterministic
node



Stochastic
node



Original
form



② Reparameterized
form

We can directly backpropagate
through Z . ϵ is constant
 g is re-parameterized elsewhere.