COMP 551 - Applied Machine Learning Lecture 21 -- Generative modeling

William L. Hamilton

Last lecture: Autoencoding

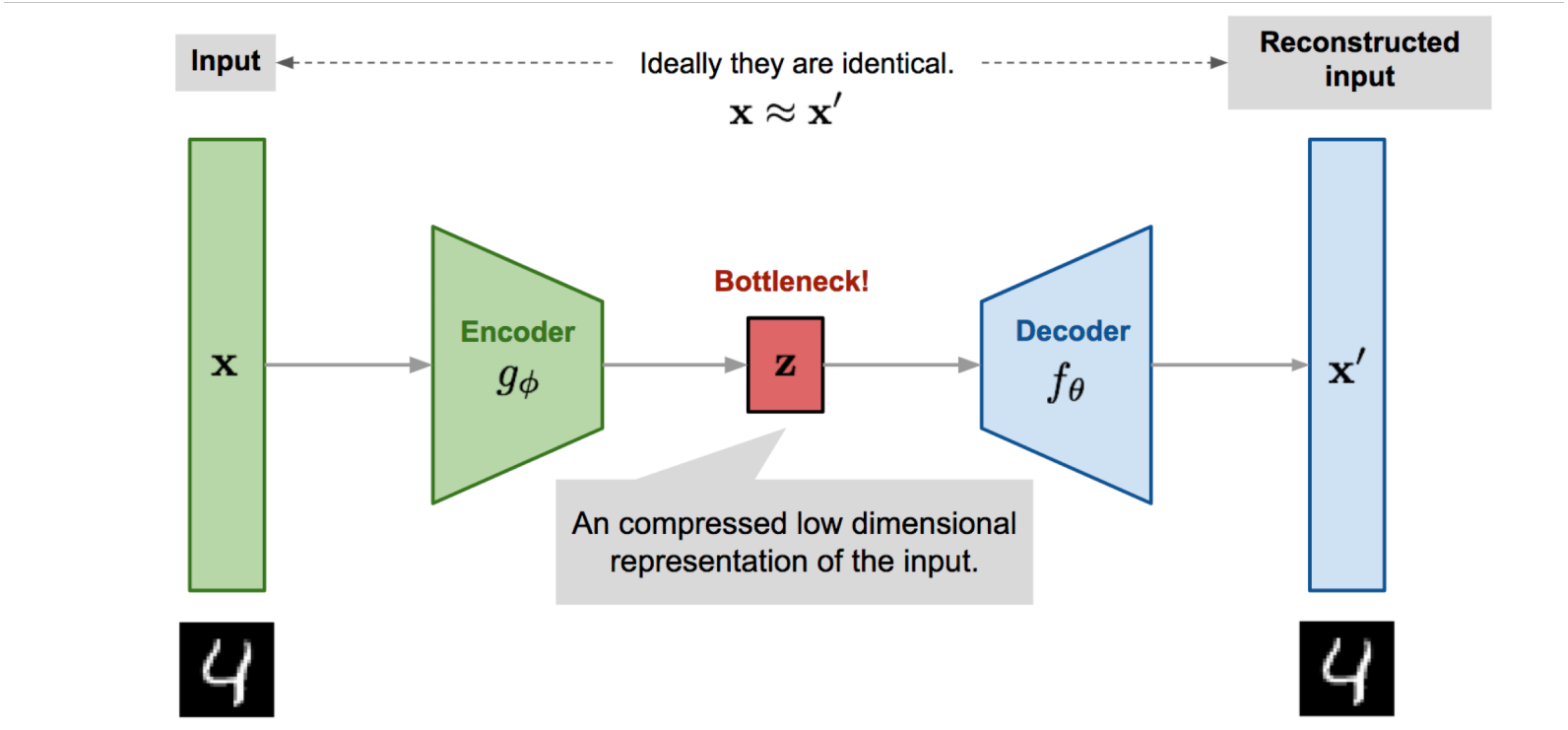


Image credit: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

Deep generative models

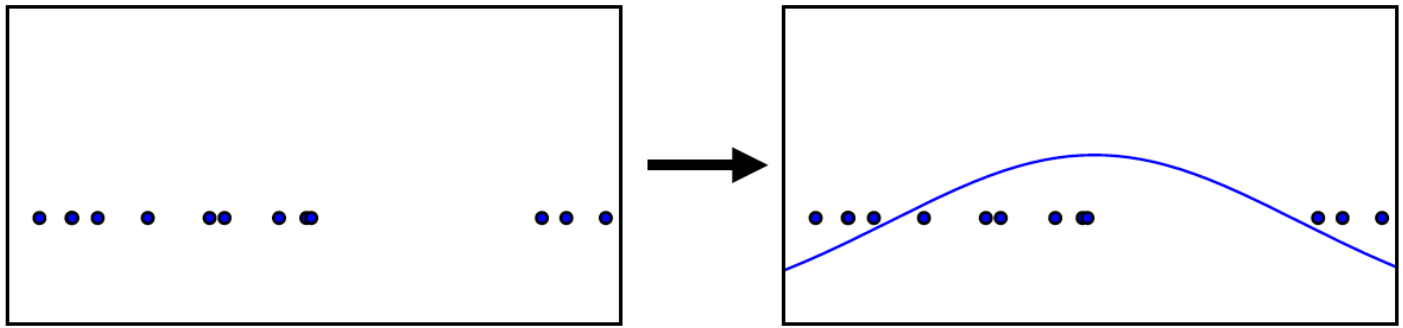
- Unsupervised learning: Given only input data: $D = <x_i>, i=1:n$, find some patterns or regularity in the data.

- Lecture 18: Cluster the data or detect anomalies.
- Lecture 20: Compress the data to learn useful representations.

- Today's lecture: Deep generative models.
  - Learn to generative realistic looking data points.

Generative modeling: Density estimation

- One goal of generative modeling is to learn a model of $p(x)$
- We saw an example of this in Lecture 18 when we used a Gaussian Mixture Model for anomaly detection.



Generative modeling: Sample generation

- However, we often just want to sample from $p(x)$
- This is what we will focus on today.

Training Data
(CelebA)

Sample Generator
(Karras et al, 2017)

Applications of sample generation

- Generating synthetic images has many applications:
  - Graphic design
  - Videogames
  - Art
  - Special effects
  - …

- But there are also many other applications of sample generation, such as
  - Generating chemical structures for drug design
  - Generating synthetic data for training reinforcement learning agents

Can we use autoencoders for generation?

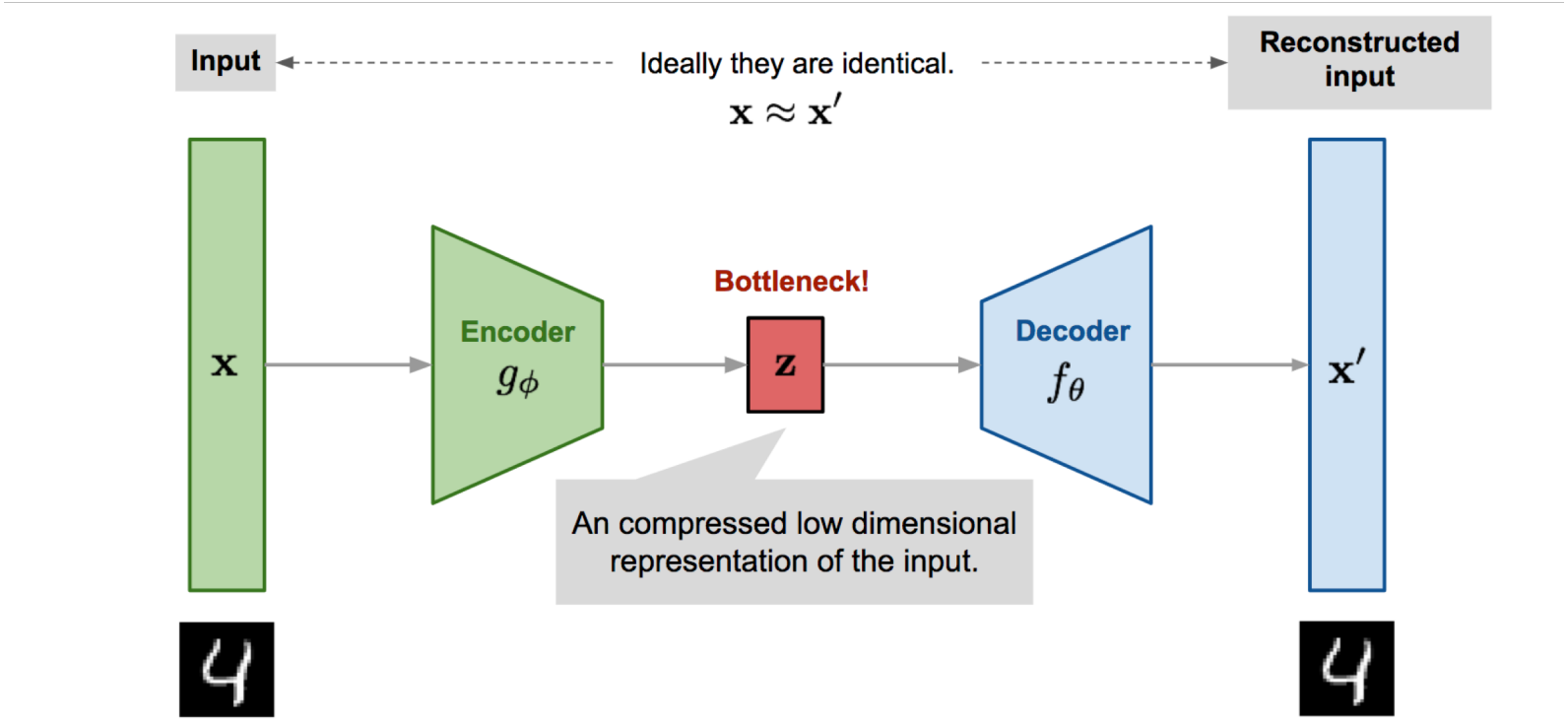


Image credit: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

Can we use autoencoders for generation?

- Autoencoders can **reconstruct** but they are not useful for generation.
- The basic autoencoder model is **deterministic.**
- Decoder is only trained to generate realistic x' samples for a small set of latent codes z.

Can we use autoencoders for generation?

- Autoencoders can **reconstruct** but they are not useful for generation.
- The basic autoencoder model is **deterministic.**
- Decoder is only trained to generate realistic x' samples for a small set of latent codes z.
- **Idea:** Introduce some randomness!

Variational Autoencoder (VAE)

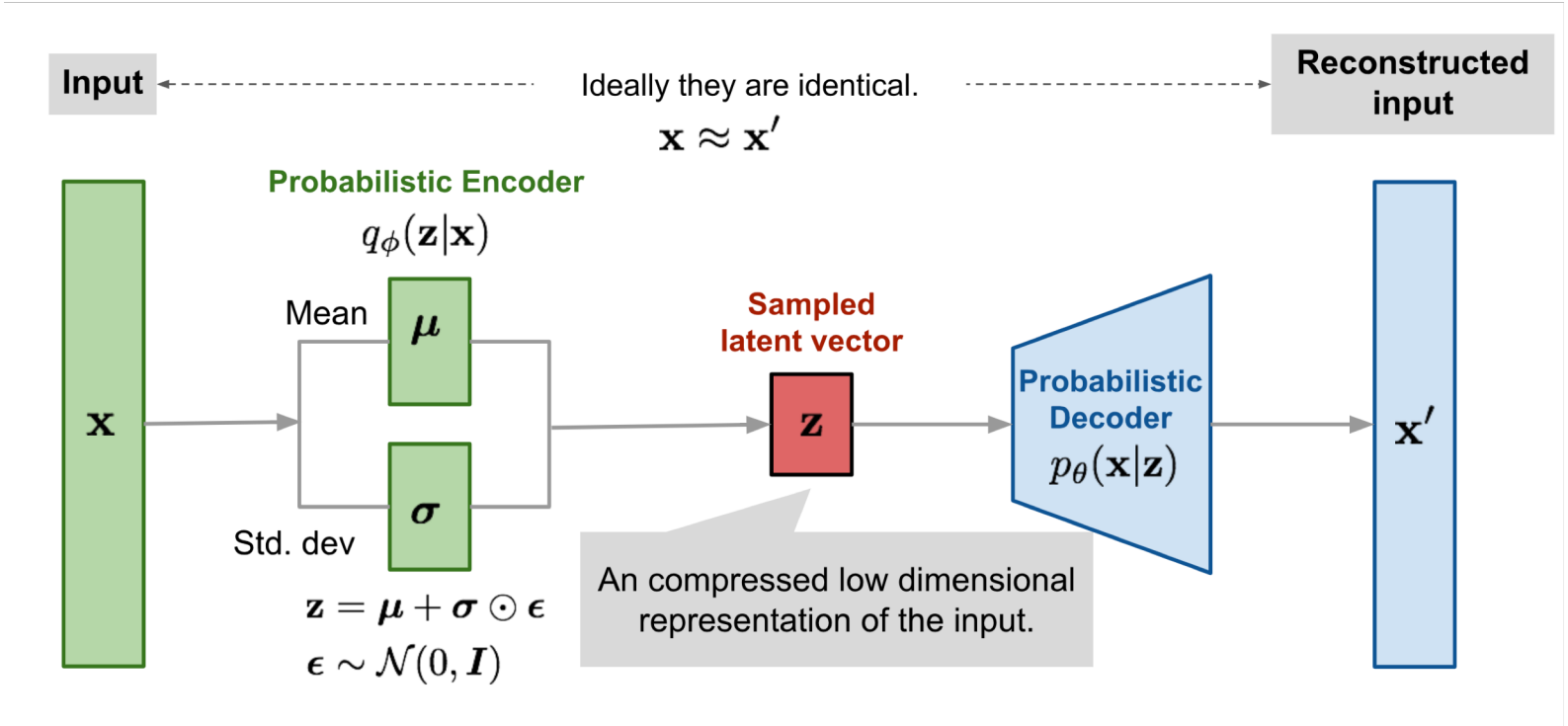


Image credit: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

Variational Autoencoder (VAE)

- **Key idea:** We use **probabilistic** encoders and decoders.
  - Instead of mapping the $\mathbf{x}$ to a *fixed* vector $\mathbf{z}$, we want to map it into a *distribution*.

Variational Autoencoder (VAE)

- **Key idea:** We use **probabilistic** encoders and decoders.
  - Instead of mapping the $\mathbf{x}$ to a *fixed* vector $\mathbf{z}$, we want to map it into a *distribution*.

- Formally, we want to treat $\mathbf{z}$ as a latent variable and model $\mathbf{p}(\mathbf{x})$:

$$P_\theta(\mathbf{x}) = \int P_\theta(\mathbf{x}|\mathbf{z}) P_\theta(\mathbf{z}) d\mathbf{z}$$

Density/distribution of $\mathbf{x}$ (what we want to model and sample from)

Conditional probability of $\mathbf{x}$ given latent/compressed encoding $\mathbf{z}$

Prior probability of $\mathbf{z}$

Variational Autoencoder (VAE)

- **Key idea:** We use **probabilistic** encoders and decoders.
  - Instead of mapping the $\mathbf{x}$ to a *fixed* vector $\mathbf{z}$, we want to map it into a *distribution*.

- Formally, we want to treat $\mathbf{z}$ as a latent variable and model $\mathbf{p}(\mathbf{x})$:

$$P_\theta(\mathbf{x}) = \int P_\theta(\mathbf{x}|\mathbf{z}) P_\theta(\mathbf{z}) d\mathbf{z}$$

Important: we want to **learn** some parameters θ to represent this distribution.

Variational Autoencoder (VAE)

- **Key idea:** We use **probabilistic** encoders and decoders.
  - Instead of mapping the $\mathbf{x}$ to a *fixed* vector $\mathbf{z}$, we want to map it into a *distribution*.

- Formally, we want to treat $\mathbf{z}$ as a latent variable and model $\mathbf{p(x)}$:

$$P_\theta(\mathbf{x}) = \int P_\theta(\mathbf{x}|\mathbf{z})P_\theta(\mathbf{z})d\mathbf{z}$$

- In particular, we want to maximize the log-likelihood of the data we have:

$$\theta^* = \arg\max_\theta \sum_{i=1}^{n} \log P_\theta(\mathbf{x}^{(i)})$$

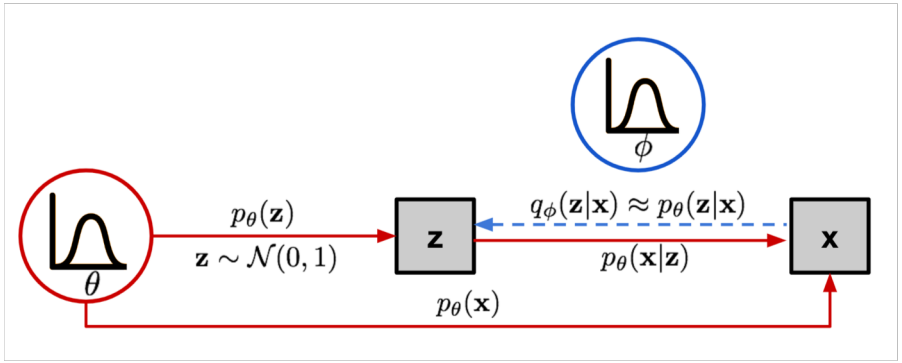
Variational Autoencoder (VAE)

- We want to maximize the log-likelihood of the data:

$$P_\theta(\mathbf{x}) = \int P_\theta(\mathbf{x}|\mathbf{z})P_\theta(\mathbf{z})d\mathbf{z}$$

But this integral means we need to sum over every possible z. That is not feasible!!

- VAE idea: Instead of summing over all possible $\mathbf{z}$ vectors, for a given input $\mathbf{x}$ lets instead predict a "good candidate" $\mathbf{z}$ using an approximation function $q_\phi(\mathbf{z}|\mathbf{x})$



Variational Autoencoder (VAE)

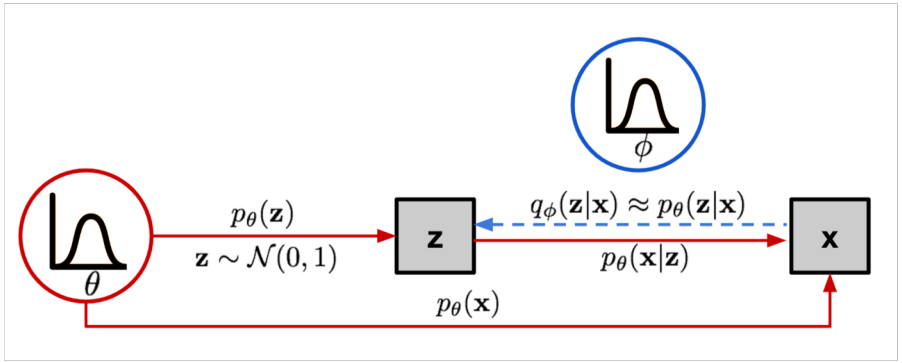
- We want to maximize the log-likelihood of the data:

$$P_\theta(\mathbf{x}) = \int P_\theta(\mathbf{x}|\mathbf{z})P_\theta(\mathbf{z})d\mathbf{z}$$

But this integral means we need to sum over every possible z. That is not feasible!!

- VAE idea: Instead of summing over all possible $\mathbf{z}$ vectors, for a given input $\mathbf{x}$ lets instead predict a "good candidate" $\mathbf{z}$ using an approximation function $q_\phi(\mathbf{z}|\mathbf{x})$

The approximation function **samples a candidate** $\mathbf{z}$ given an $\mathbf{x}$. (Like an encoder, but probabilistic.)



Variational Autoencoder (VAE)

- We want to maximize the log-likelihood of the data:

$$P_\theta(\mathbf{x}) = \int P_\theta(\mathbf{x}|\mathbf{z})P_\theta(\mathbf{z})d\mathbf{z}$$

But this integral means we need to sum over every possible z. That is not feasible!!

- But we work with the approximation:

$$\log(P_\theta(\mathbf{x})) \approx \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log P_\theta(\mathbf{x}|\mathbf{z})]$$

- I.e., instead of summing over all z we just sample some z from the "candidate" distribution generated by the probabilistic encoder.

Variational Autoencoder (VAE)

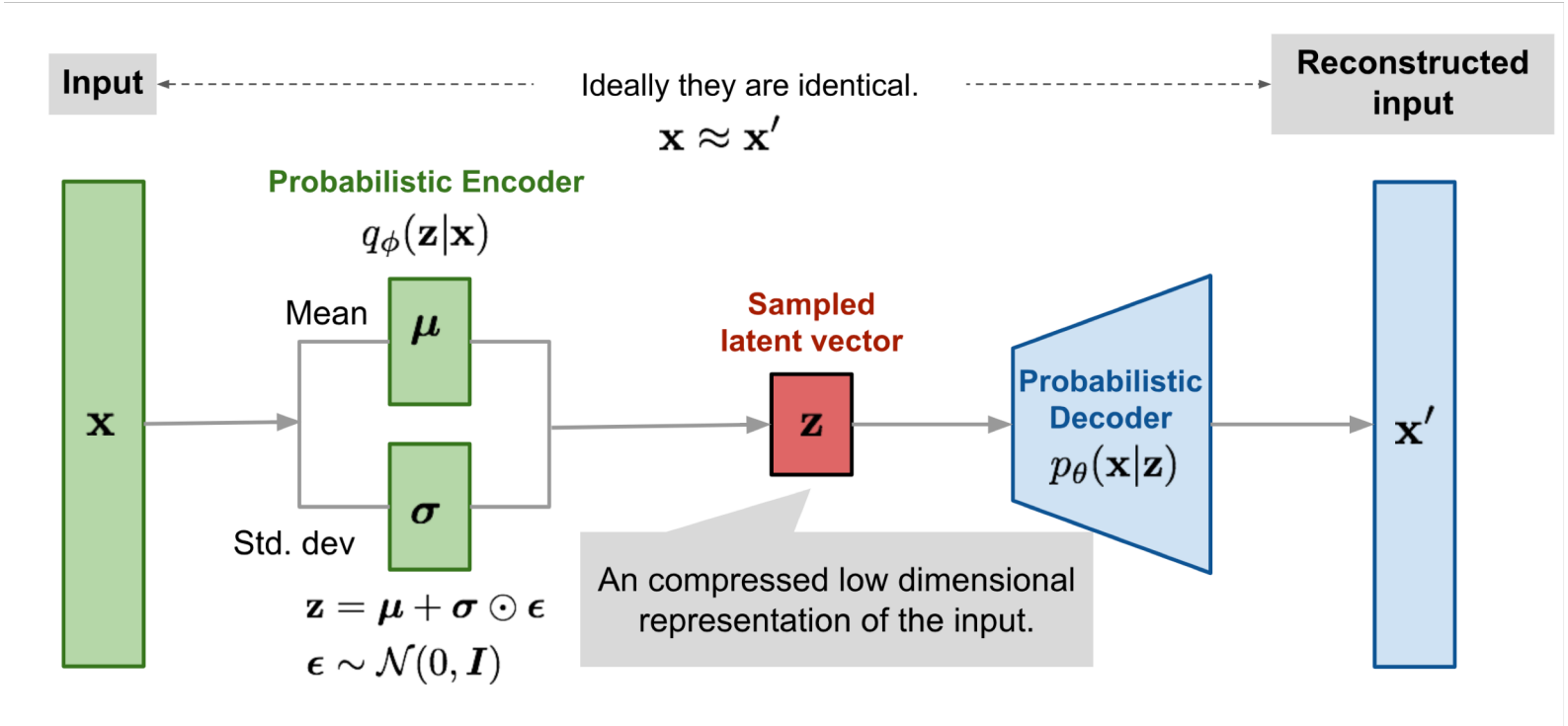


Image credit

Variational Autoencoder (VAE)

- We want to maximize the log-likelihood of the data:

$$P_\theta(\mathbf{x}) = \int P_\theta(\mathbf{x}|\mathbf{z})P_\theta(\mathbf{z})d\mathbf{z}$$

But this integral means we need to sum over every possible z. That is not feasible!!

- But we work with the approximation:

$$\log(P_\theta(\mathbf{x})) \approx \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log P_\theta(\mathbf{x}|\mathbf{z})]$$

- But there are still two issues:
  1. How do we train a neural network $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ that we can sample from?
  2. How do we ensure that $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is "reasonable"/"regularized"?

Reparameterization trick

- Issue: How do we train a neural network $q_{\phi}(\boldsymbol{z}|\boldsymbol{x})$ that we can sample from?
- Sampling is a stochastic process and therefore we cannot backpropagate the gradient….

- Gaussian reparameterization trick:

$$\mathbf{z} \sim q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)}) = \mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I})$$
$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I}) \quad ; \text{Reparameterization trick.}$$

- We use a neural network to predict **mean** and **variance** vectors given an input, and then we generate a sample by adding this mean and variance to a sample from a unit normal distribution.

Reparameterization trick

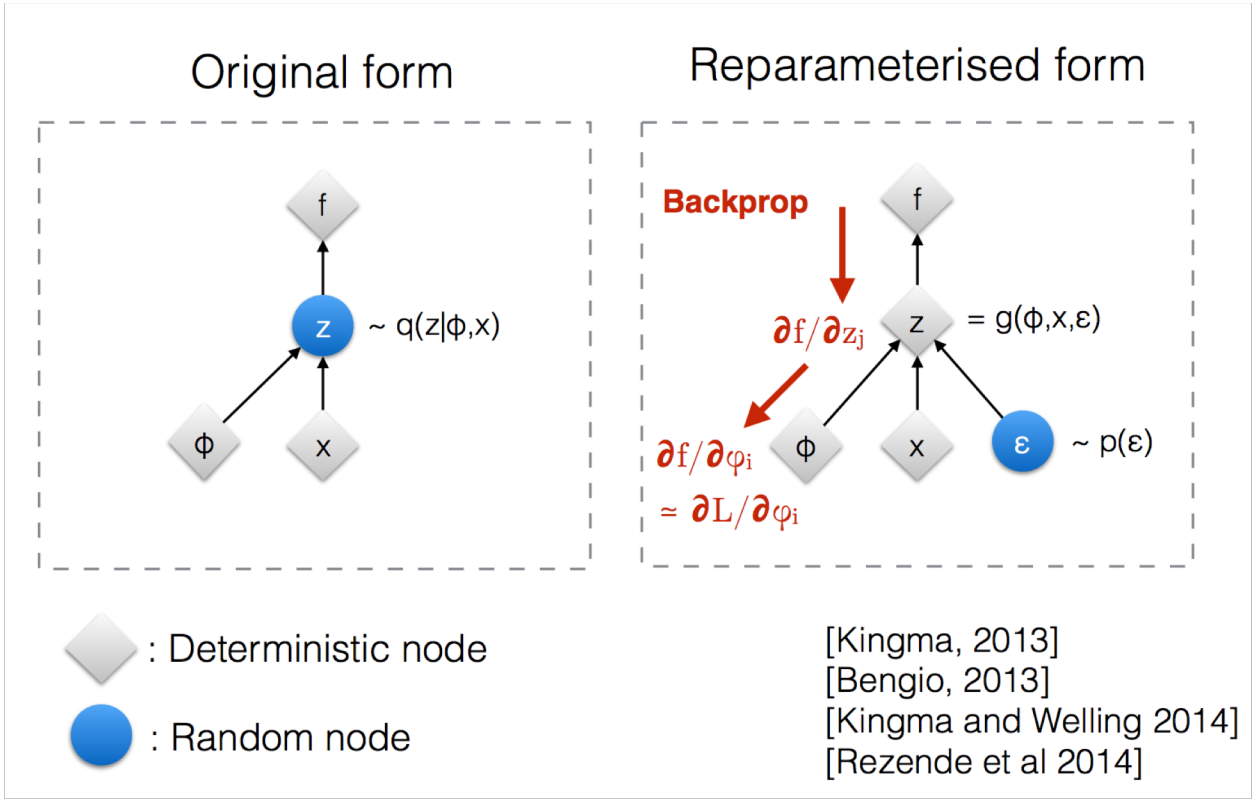


Image credit

Variational Autoencoder (VAE)

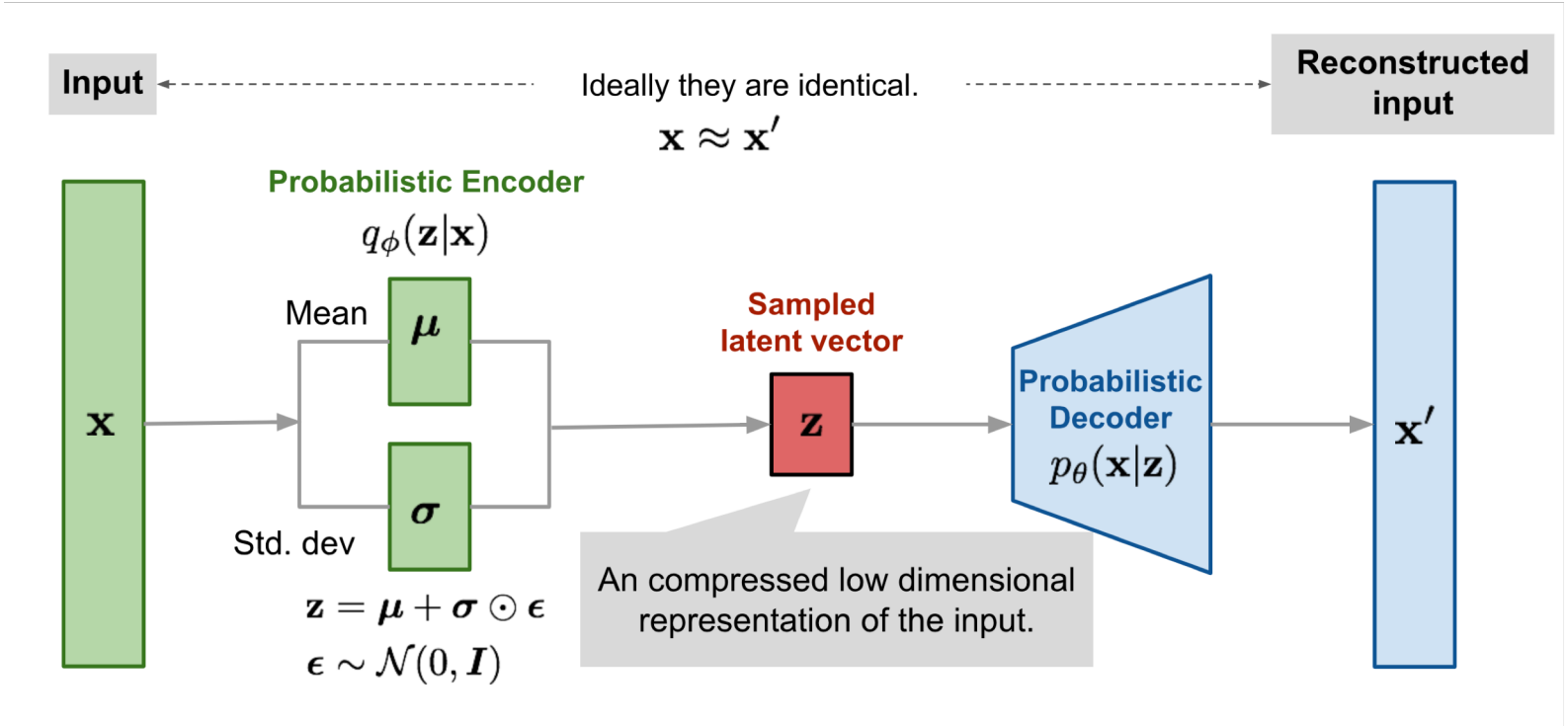


Image credit

Regularizing the encoder

- Recall: We want to be able to generate realistic data, not just autoencoder data.

- Issue: How do we prevent $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ from overfitting the training data?
- Issue: How can we generate meaningful $\mathbf{x}$ samples from random latent vectors $\mathbf{z}$?

Regularizing the encoder

- Recall: We want to be able to generate realistic data, not just autoencoder data.

- Issue: How do we prevent $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ from overfitting the training data?
- Issue: How can we generate meaningful $\mathbf{x}$ samples from random latent vectors $\mathbf{z}$?

- Solution: Regularize $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ so that it is close to a standard normal distribution.

$$\log(P_\theta(\mathbf{x})) \approx \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log(P_\theta(\mathbf{x}|\mathbf{z})) - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||P_\theta(\mathbf{z}))$$

Approximating $\mathbf{P}(\mathbf{x})$ by sampling from the candidate/encoder distribution.

Regularizing $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ so that it is close to the prior distribution on $\mathbf{z}$ (e.g., a Gaussian)

VAE objective

- VAE objective:

$$\log(P_\theta(\mathbf{x})) \approx \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log(P_\theta(\mathbf{x}|\mathbf{z})) - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||P_\theta(\mathbf{z}))$$

Approximating $\mathbf{P(x)}$ by sampling from the candidate/encoder distribution.

Regularizing $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ so that it is close to the prior distribution on $\mathbf{z}$ (e.g., a Gaussian)

VAE objective

- VAE objective:

$$\log(P_\theta(\mathbf{x})) \approx \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log(P_\theta(\mathbf{x}|\mathbf{z})) - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||P_\theta(\mathbf{z}))$$

Approximating $\mathbf{P(x)}$ by sampling from the candidate/encoder distribution.

Regularizing $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ so that it is close to the prior distribution on $\mathbf{z}$ (e.g., a Gaussian)

- KL divergence measures the distance between two probability distributions:

$$\mathrm{KL}(P||Q) = \sum_{x\in\mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

- KL interpretation: The expected number of extra bits required to describe samples from P if we optimized a code on Q.

VAE objective

- VAE objective:

$$\log(P_\theta(\mathbf{x})) \approx \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log(P_\theta(\mathbf{x}|\mathbf{z})) - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||P_\theta(\mathbf{z}))$$

Note that this is essentially the reconstruction loss (and is task specific). Really, this is analogous to $\mathbf{P(y|x)}$ in the supervised case (i.e., it is the log-likelihood) so we can use things like cross-entropy, etc.

Regularizing $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ so that it is close to the prior distribution on $\mathbf{z}$ (e.g., a Gaussian)

VAE objective

- VAE objective:

$$\log(P_\theta(\mathbf{x})) \approx \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log(P_\theta(\mathbf{x}|\mathbf{z})) - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))$$

Approximating $\mathbf{P(x)}$ by sampling from the candidate/encoder distribution.

Regularizing $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ so that it is close to the prior distribution on $\mathbf{z}$ (e.g., a Gaussian)

- Key ideas:
  1. We sample latent codes given the input (probabilistic encoder).
  2. We want the (expected) reconstruction using the latent codes to be high-quality.
  3. We want the distribution of the latent codes to be close to a Gaussian (or another) simple distribution.

Variational Autoencoder (VAE)

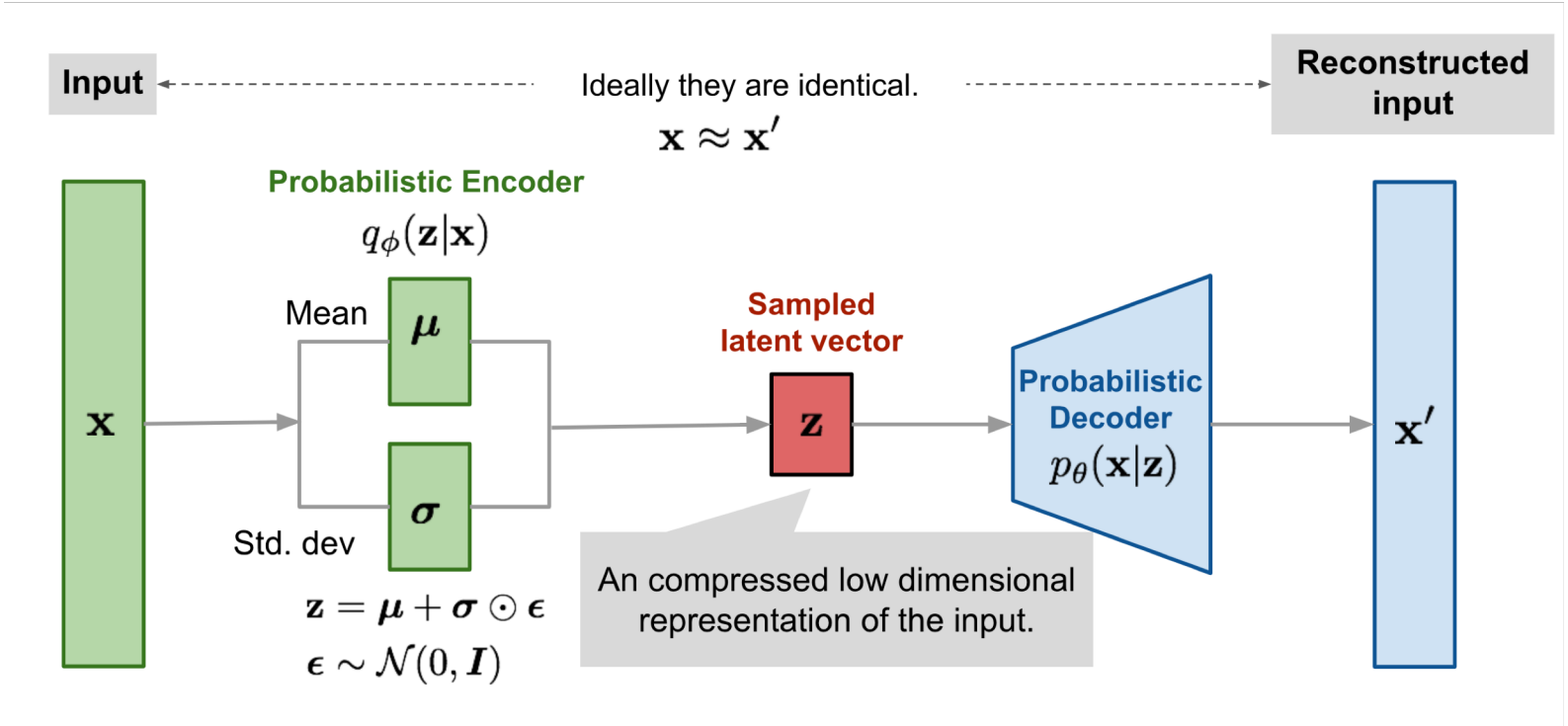
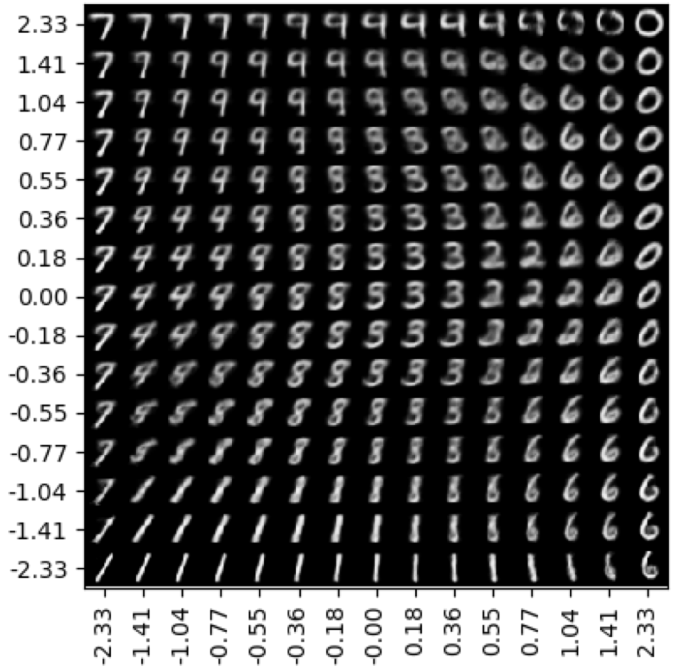
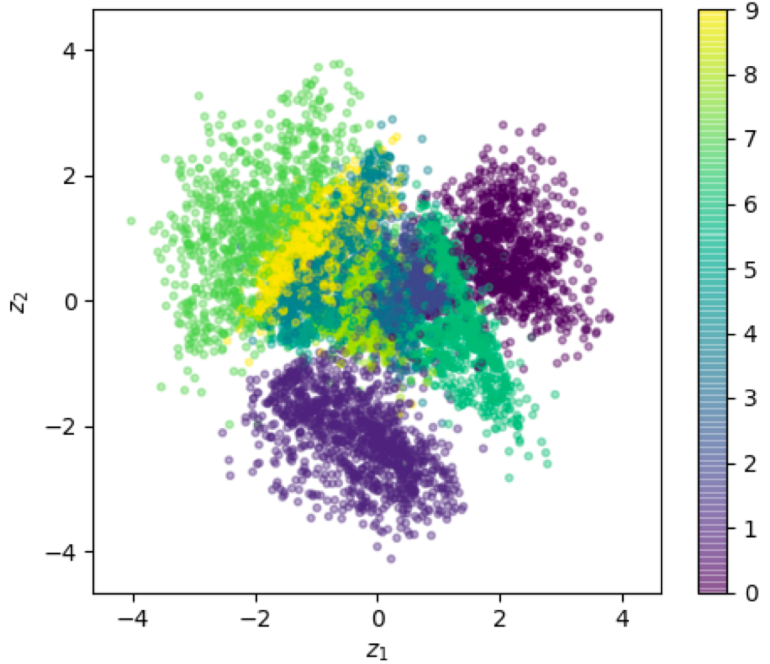


Image credit

VAEs: Structure in the latent space





VAEs: Formal derivation

- We came to the approximation:

$$\log(P_\theta(\mathbf{x})) \approx \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log(P_\theta(\mathbf{x}|\mathbf{z}) - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||P_\theta(\mathbf{z}))$$

- But I simplified the derivation (quite a bit)… Really VAEs are optimizing:

$$\log(P_\theta(\mathbf{x})) - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||P_\theta(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log(P_\theta(\mathbf{x}|\mathbf{z}) - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||P_\theta(\mathbf{z}))$$

Maximizing log-likelihood.

Minimizing difference between $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and the real distribution $P_\theta(\boldsymbol{z}|\boldsymbol{x})$ (which we don't know)

VAEs: Formal derivation

- We came to the approximation:

$$\log(P_\theta(\mathbf{x})) \approx= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log(P_\theta(\mathbf{x}|\mathbf{z}) - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||P_\theta(\mathbf{z}))$$

- But I simplified the derivation (quite a bit)… Really VAEs are optimizing:

$$\log(P_\theta(\mathbf{x})) - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||P_\theta(\mathbf{z}|\mathbf{x})) \leq \log(P_\theta(\mathbf{x}))$$

This is called the Evidence Lower Bound (ELBO).

VAEs: Formal derivation

- We came to the approximation:

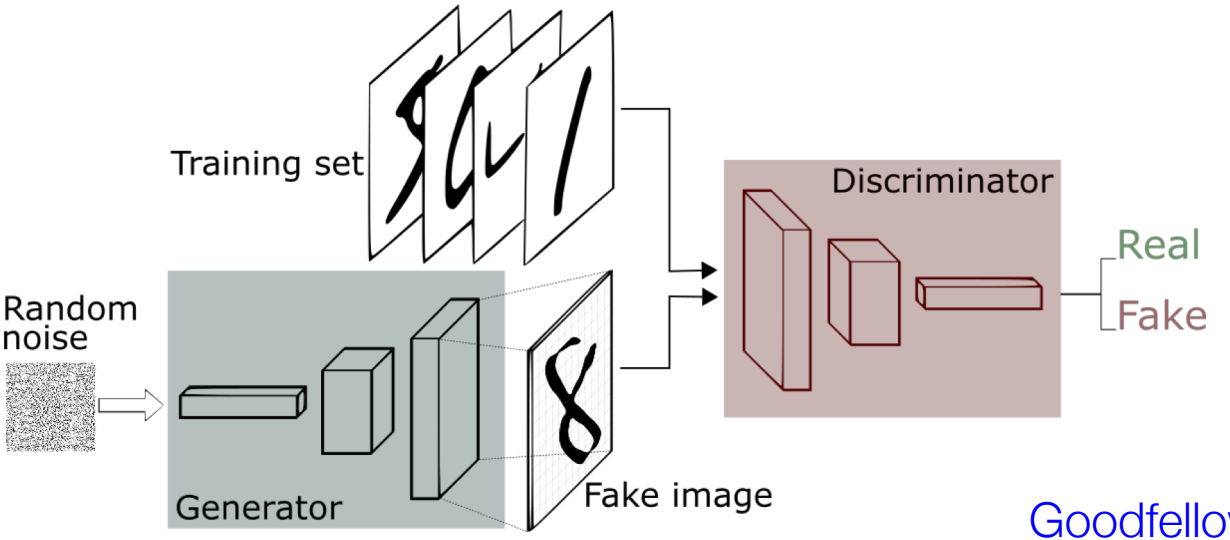
$$\log(P_\theta(\mathbf{x})) \approx= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log(P_\theta(\mathbf{x}|\mathbf{z}) - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||P_\theta(\mathbf{z}))$$

- But I simplified the derivation (quite a bit)...

- For all the technical details check out this tutorial paper.

Generative Adversarial Networks

- VAEs use a probabilistic approximation to generate realistic samples.
- Alternative idea: Learn to generate samples that can fool an "discriminator"



Goodfellow et al, 2014

Image credit

Generative Adversarial Networks

- VAEs use a probabilistic approximation to generate realistic samples.
- Alternative idea: Learn to generate samples that can fool an "discriminator"

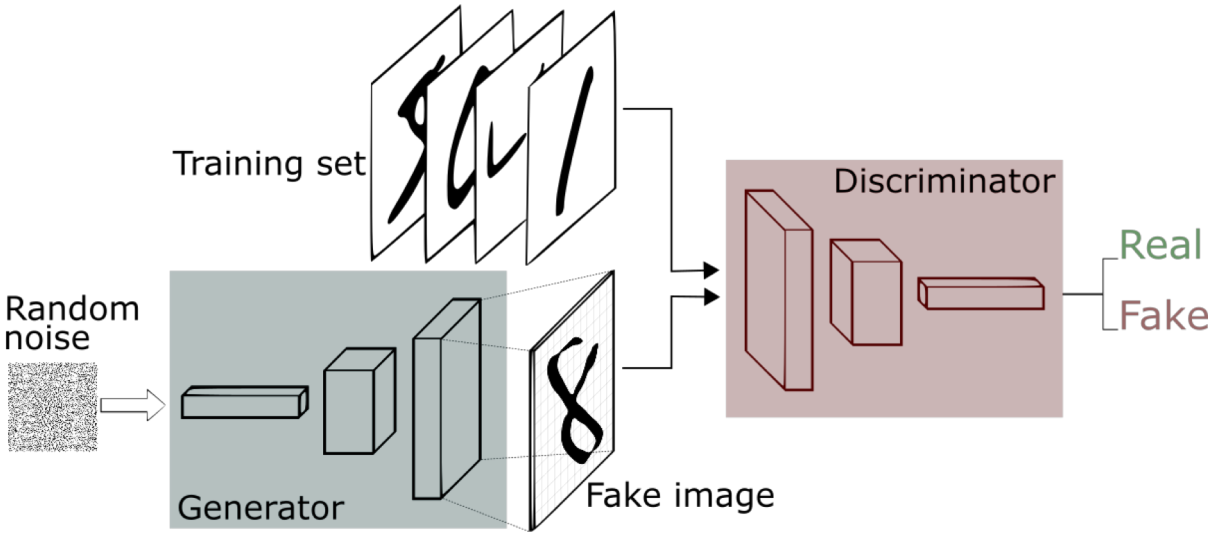
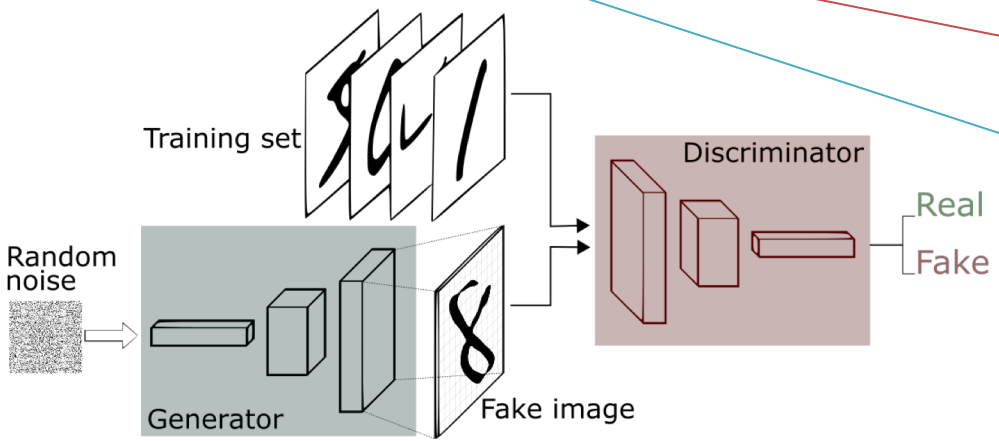


Image credit

Generative Adversarial Networks

- Both the generator and discriminator are neural networks (usually CNNs).
- Optimization is a min-max game:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$



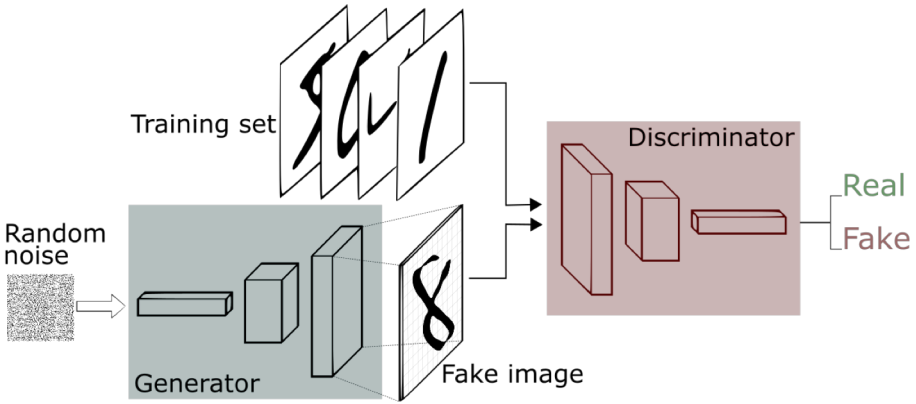
Discriminator is optimized to distinguish generated samples from real data.

Generator is wants to minimize the accuracy of the discriminator

Image credit

Generative Adversarial Networks

- Optimize by alternating gradient descent on generator and discriminator.
- Very hard to train in practice!



$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Discriminator is optimized to distinguish generated samples from real data.

Generator is wants to minimize the accuracy of the discriminator

Image credit

Generative Adversarial Networks

- GANs have led to new state-of-the-art results for image generation

From Brock et al, 2018.

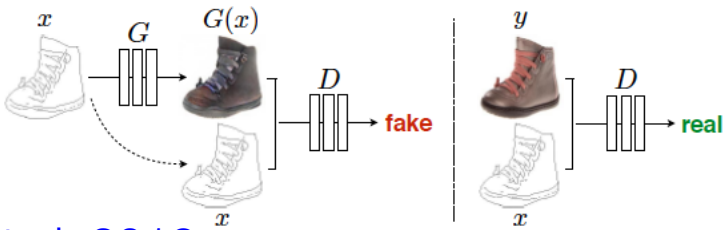
Generative Adversarial Networks

- GANs have led to new state-of-the-art results for image generation.

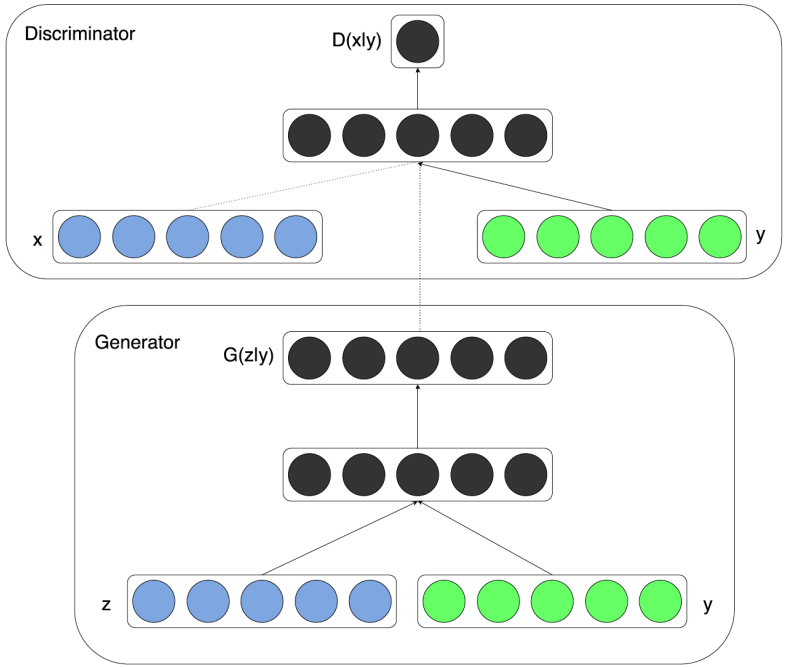
- https://thispersondoesnotexist.com/ [power by GANs]

Conditional GANS

- GANs can be made "conditional"
- I.e., generate samples that have certain properties.
- Simple idea: Feed conditioning information as input to both the generator and the discriminator

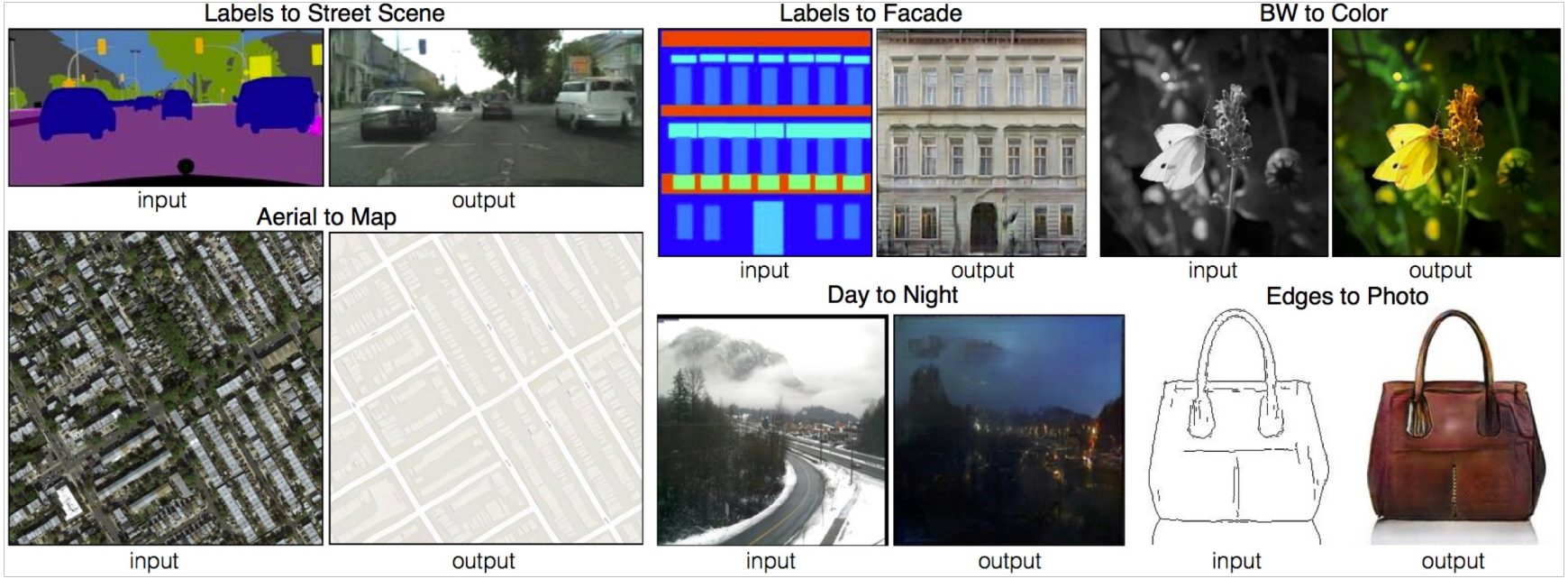


Isola et al, 2018



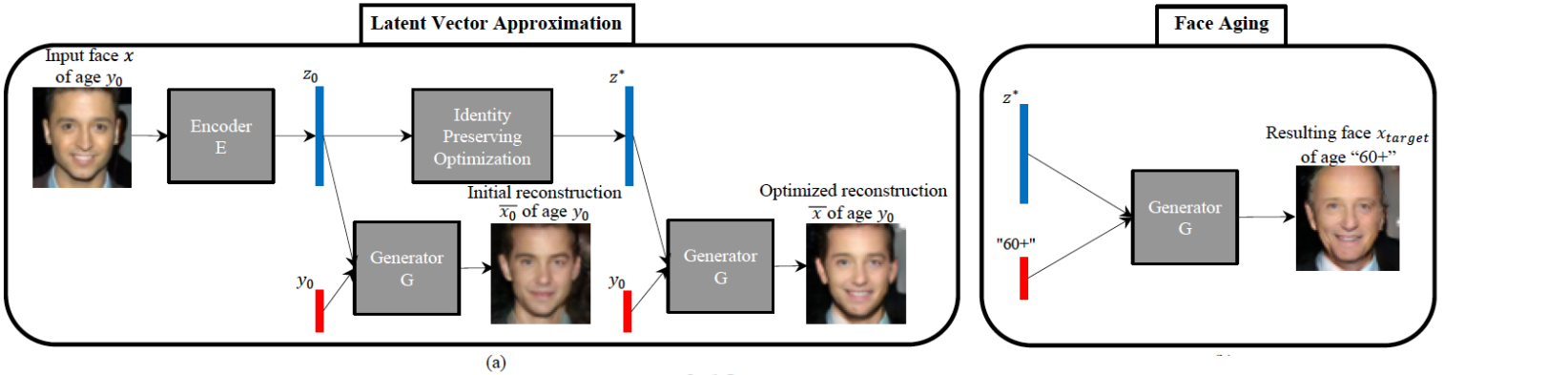
Mizra et al, 2017

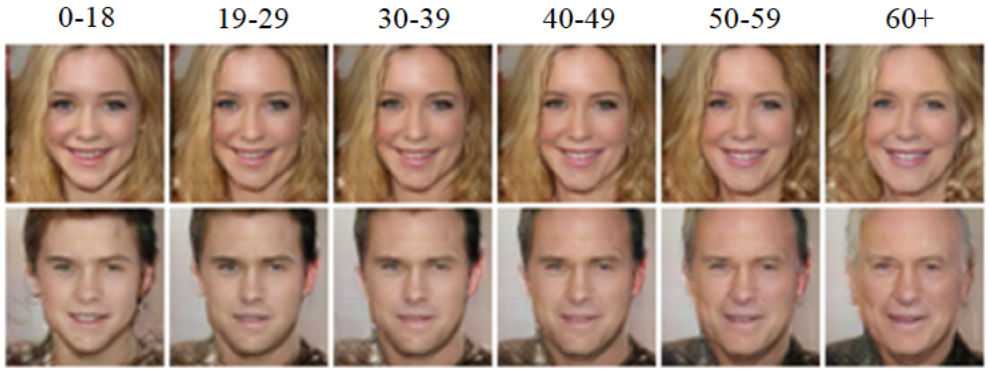
Conditional GANs



[Isola et al, 2018]

Conditional GANs





[Antipov et al, 2017]

Summary

- GANs and VAEs are powerful general-purpose approaches for generative modeling.
- VAEs are rooted in a probabilistic approximation.
- GANs turn generation into an “adversarial game”.
- Mostly used for images (and some video) right now.
- Ethical concerns: Fake images and videos, misinformation, …