# As a recap. Autoencoders.

# Generative models

**Generative models**

- learn joint probability distribution from the training data
- predict the conditional probability using Bayes rule

**Discriminative models**

- learn the conditional probability directly from the training data

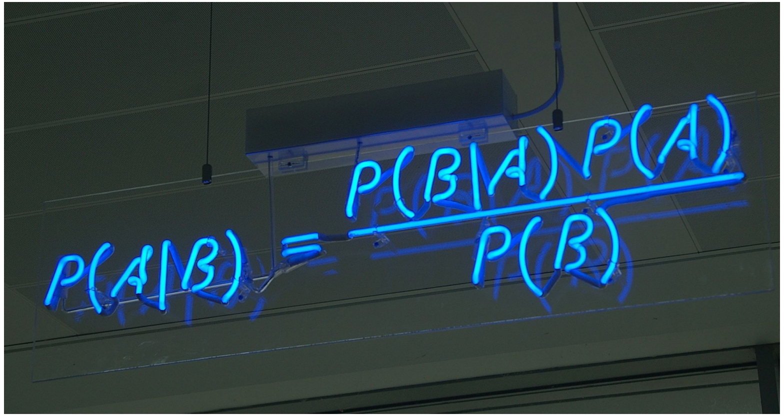
**Generative models**

- learn joint probability distribution from the training data
- predict the conditional probability using Bayes rule

**Discriminative models**

- learn the conditional probability directly from the training data

## The Church of Bayes



**Generative models**

- learn joint probability distribution from the training data
- predict the conditional probability using Bayes rule

**Discriminative models**

- learn the conditional probability directly from the training data

Given training data, we want generate new samples from same distribution.

$p_{model}(x)$ to be similar to $p_{data}(x)$

**Generative models**

- learn joint probability distribution from the training data
- predict the conditional probability using Bayes rule

**Discriminative models**

- learn the conditional probability directly from the training data

Given training data, we want generate new samples from same distribution.

$p_{\mathtt{model}}(x)$ to be similar to $p_{\mathtt{data}}(x)$

It addresses the problem of density estimation.

**Generative models**

- learn joint probability distribution from the training data
- predict the conditional probability using Bayes rule

**Discriminative models**

- learn the conditional probability directly from the training data

Given training data, we want generate new samples from same distribution.

$p_{\mathtt{model}}(x)$ to be similar to $p_{\mathtt{data}}(x)$

It addresses the problem of density estimation.

explicit

implicit

**Generative models**

- learn joint probability distribution from the training data
- predict the conditional probability using Bayes rule

**Discriminative models**

- learn the conditional probability directly from the training data

Given training data, we want generate new samples from same distribution.

$p_{model}(x)$ to be similar to $p_{data}(x)$

It addresses the problem of density estimation.

explicit, e.g. VAEs

implicit, e.g. GANs

# Why do we need generative models?

# Why do we need generative models?

- image restoration

**ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks**

Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao, Xiaoou Tang

*(Submitted on 1 Sep 2018 (v1), last revised 17 Sep 2018 (this version, v2))*

The Super-Resolution Generative Adversarial Network (SRGAN) is a seminal work that is capable of generating realistic textures during sir
further enhance the visual quality, we thoroughly study three key components of SRGAN - network architecture, adversarial loss and perce
Residual-in-Residual Dense Block (RRDB) without batch normalization as the basic network building unit. Moreover, we borrow the idea frc
the perceptual loss by using the features before activation, which could provide stronger supervision for brightness consistency and texture
with more realistic and natural textures than SRGAN and won the first place in the PIRM2018-SR Challenge. The code is available at this h

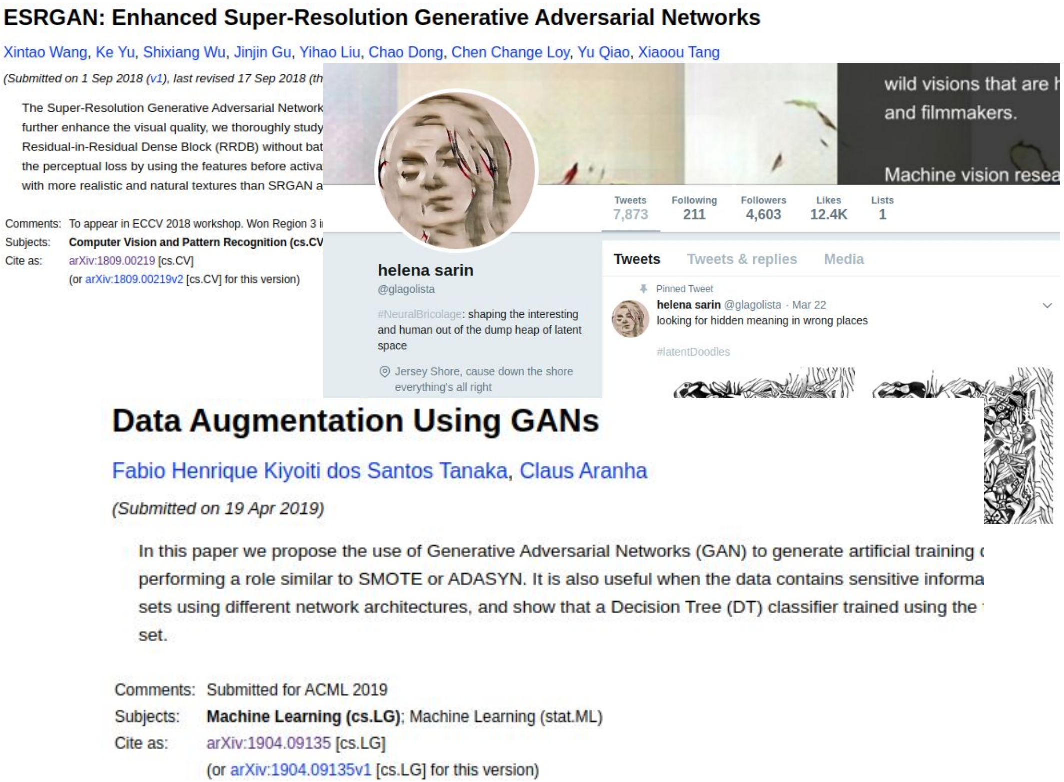
Comments: To appear in ECCV 2018 workshop. Won Region 3 in the PIRM2018-SR Challenge. Code and models are at this https URL
Subjects: **Computer Vision and Pattern Recognition (cs.CV)**
Cite as: arXiv:1809.00219 [cs.CV]
(or arXiv:1809.00219v2 [cs.CV] for this version)

# Why do we need generative models?

- image restoration
- artwork

# Why do we need generative models?

- image restoration
- artwork
- data augmentation


ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks

Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao, Xiaoou Tang

(Submitted on 1 Sep 2018 (v1), last revised 17 Sep 2018 (th

The Super-Resolution Generative Adversarial Network
further enhance the visual quality, we thoroughly study
Residual-in-Residual Dense Block (RRDB) without bat
the perceptual loss by using the features before activa
with more realistic and natural textures than SRGAN a

Comments: To appear in ECCV 2018 workshop. Won Region 3 i
Subjects: **Computer Vision and Pattern Recognition (cs.CV**
Cite as: arXiv:1809.00219 [cs.CV]
(or arXiv:1809.00219v2 [cs.CV] for this version)



Data Augmentation Using GANs

Fabio Henrique Kiyoiti dos Santos Tanaka, Claus Aranha

(Submitted on 19 Apr 2019)

In this paper we propose the use of Generative Adversarial Networks (GAN) to generate artificial training
performing a role similar to SMOTE or ADASYN. It is also useful when the data contains sensitive informa
sets using different network architectures, and show that a Decision Tree (DT) classifier trained using the
set.

Comments: Submitted for ACML 2019
Subjects: **Machine Learning (cs.LG)**; Machine Learning (stat.ML)
Cite as: arXiv:1904.09135 [cs.LG]
(or arXiv:1904.09135v1 [cs.LG] for this version)

# Variational Autoencoders

# Auto-Encoding Variational Bayes

Diederik P Kingma, Max Welling

(Submitted on 20 Dec 2013 (v1), last revised 1 May 2014 (this version, v10))

How can we perform efficient inference and learning in directed probabilisti
algorithm that scales to large datasets and, under some mild differentiabilit
that can be straightforwardly optimized using standard stochastic gradient
inference model (also called a recognition model) to the intractable posteri

Subjects: **Machine Learning (stat.ML)**; Machine Learning (cs.LG)
Cite as: arXiv:1312.6114 [stat.ML]
(or arXiv:1312.6114v10 [stat.ML] for this version)

# Auto-Encoding Variational Bayes

**Diederik P. Kingma**
Machine Learning Group
Universiteit van Amsterdam
dpkingma@gmail.com

**Max Welling**
Machine Learning Group
Universiteit van Amsterdam
welling.max@gmail.com

## Abstract

How can we perform efficient inference and learning in directed probabilistic models, in the presence of continuous latent variables with intractable posterior distributions, and large datasets? We introduce a stochastic variational inference and learning algorithm that scales to large datasets and, under some mild differentiability conditions, even works in the intractable case. Our contributions is two-fold. First, we show that a reparameterization of the variational lower bound yields a lower bound estimator that can be straightforwardly optimized using standard stochastic gradient methods. Second, we show that for i.i.d. datasets with continuous latent variables per datapoint, posterior inference can be made especially efficient by fitting an approximate inference model (also called a recognition model) to the intractable posterior using the proposed lower bound estimator. Theoretical advantages are reflected in experimental results.

Instead of mapping the input to a fixed vector, we want to map the input onto a distribution - probabilistic spin on autoencoders.

We want to model $X$ with an additional latent variable $z$, assuming $X$ is generated from this underlying unobserved representation.

Instead of mapping the input to a fixed vector, we want to map the input onto a distribution - probabilistic spin on autoencoders.

We want to model $X$ with an additional latent variable $z$, assuming $X$ is generated from this underlying unobserved representation.

$z$ can be thought of as a vector, elements of which are capturing how little or how much of some factor variation we have in the training data.

Instead of mapping the input to a fixed vector, we want to map the input onto a distribution - probabilistic spin on autoencoders.

We want to model X with an additional latent variable z, assuming X is generated from this underlying unobserved representation.

The process of data generation:

- value $z^{(i)}$ is generated from some prior distribution $p_{\Theta^*}(x)$
- value $x^{(i)}$ is generated from some conditional distribution $p_{\Theta^*}(x|z)$

We want to estimate these true parameters $\Theta^*$.

Instead of mapping the input to a fixed vector, we want to map the input onto a distribution - probabilistic spin on autoencoders.

We want to model X with an additional latent variable z, assuming X is generated from this underlying unobserved representation.

The process of data generation:

- value $z^{(i)}$ is generated from some prior distribution $p_{\theta^*}(z)$
- value $x^{(i)}$ is generated from some conditional distribution $p_{\theta^*}(x|z)$

We want to estimate these true parameters $\theta^*$.

How can we represent this model? What should $p_{\theta^*}(z)$ and $p_{\theta^*}(x|z)$ be?

Instead of mapping the input to a fixed vector, we want to map the input onto a distribution - probabilistic spin on autoencoders.

We want to model X with an additional latent variable z, assuming X is generated from this underlying unobserved representation.

How can we train this model?

Instead of mapping the input to a fixed vector, we want to map the input onto a distribution - probabilistic spin on autoencoders.

We want to model X with an additional latent variable z, assuming X is generated from this underlying unobserved representation.

How can we train this model?

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

This is our data likelihood. We want to maximize it.

Instead of mapping the input to a fixed vector, we want to map the input onto a distribution - probabilistic spin on autoencoders.

We want to model X with an additional latent variable z, assuming X is generated from this underlying unobserved representation.

How can we train this model?

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

This is our data likelihood. But it's intractable.

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

Gaussian prior

Decoder network

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

for every z!

Gaussian prior

Decoder network

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

this guy makes this expression intractable as well!

$$p_\theta(z|x) = p_\theta(x|z) p_\theta(z) / p_\theta(x)$$ (posterior)

$$p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$$

this guy makes this expression intractable as well!

$$p_\theta(z|x) = p_\theta(x|z)p_\theta(z)/p_\theta(x)$$

(posterior)

we can approximate this with an encoder network $q_{\phi^*}(z|x)$

$$p_\theta(x) = \int p_\theta(z)\, p_\theta(x/z)\, dz$$

$$\log p_\theta(x^{(i)}) = \mathbb{E}_{z \sim q_\varphi(z/x^{(i)})} \left(\log p_\theta(x^{(i)})\right) =$$

{ doesn't depend on z }

Bayes' Rule

$$= \mathbb{E}_z \left(\log \frac{p_\theta(x^{(i)}/z)\, p_\theta(z)}{p_\theta(z/x^{(i)})}\right) =$$

$$= \mathbb{E}_z \left(\log \frac{p_\theta(x^{(i)}/z)\, p_\theta(z)}{p_\theta(z/x^{(i)})} \cdot \frac{q_\varphi(z/x^{(i)})}{q_\varphi(z/x^{(i)})}\right) =$$

$$= \mathbb{E}_z \left[\left(\log p_\theta(x^{(i)}/z)\right) + \log \frac{p_\theta(z)}{q_\varphi(z/x^{(i)})} + \log \frac{q_\varphi(z/x^{(i)})}{p_\theta(z/x^{(i)})}\right] =$$

$$= \mathbb{E}_z\left(\log p_\theta(x^{(i)}|z)\right) - \mathbb{E}_z\left(\log \frac{q_\varphi(z|x^{(i)})}{p_\theta(z)}\right) + \mathbb{E}_z\left(\log \frac{q_\varphi(z|x^{(i)})}{p_\theta(z|x^{(i)})}\right)$$

$$\|$$

$$D_{KL}(q_\varphi(z|x^{(i)}) \,\|\, p_\theta(z))$$

same here

What is $D_{KL}$?

# KL divergence (1951)

$$KL(p(X)\|q(X)) = \int p(x) \log \frac{p(x)}{q(x)} dx$$

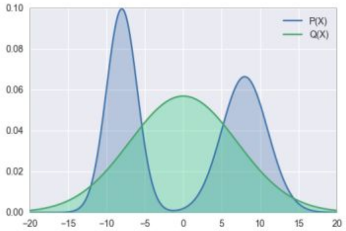
The amount of information lost when q is used to approximate p.

It is non-negative, and is equal to zero iff p=q.

It's also asymmetric:

Minimizing forward KL divergence:

$$\min_q KL(p\|q) \quad \left(\int p(x) \log \frac{p(x)}{q(x)} dx\right)$$

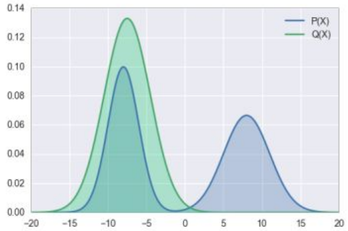



Well on average in the expectation over p

Example:
p - bimodal
q - Gaussian

Minimizing reverse KL divergence:

$$\min_q KL(q\|p) \quad \left(\int q(x) \log \frac{q(x)}{p(x)} dx\right)$$



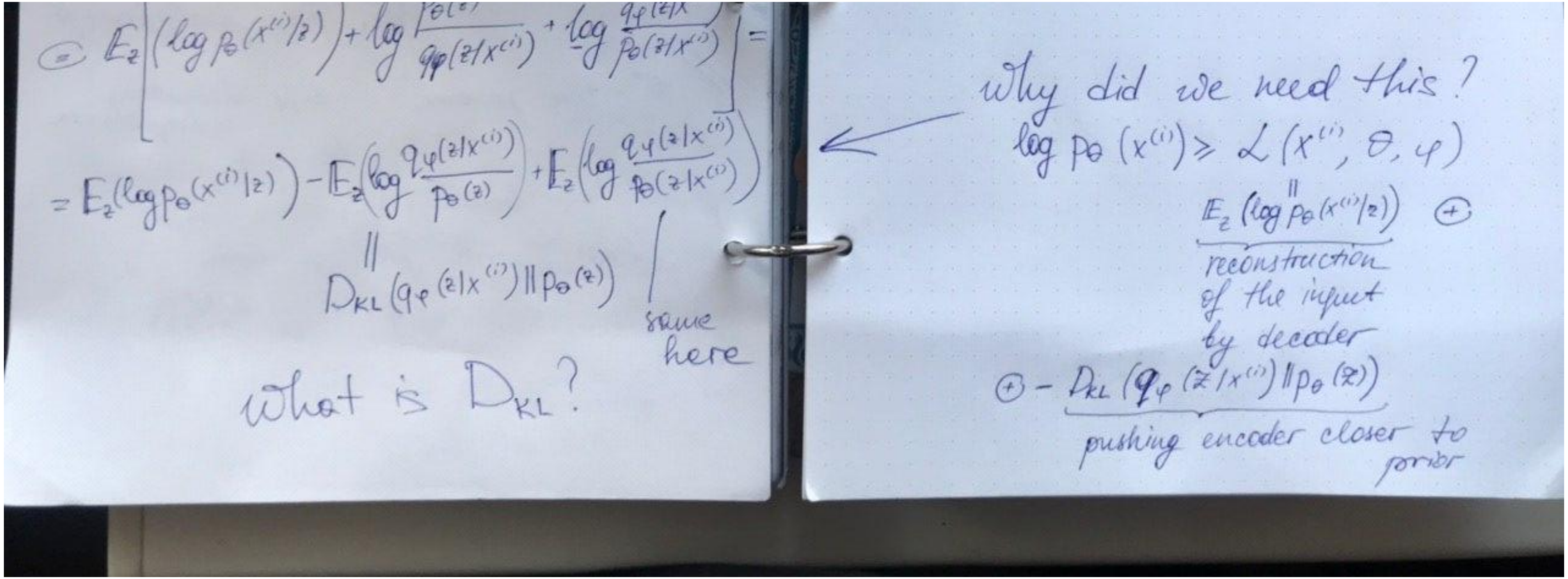

Well on average in the expectation over q — concentrating around a mode of p

*Alex Shekhovtsov, CV course @ UCU, 2018*

$$\mathbb{E}_{z \sim q_\varphi(z|x^{(i)})} \log \frac{q_\varphi(z|x^{(i)})}{p_\theta(z)} =$$

$$= \int q_\varphi(z|x^{(i)}) \log \frac{q_\varphi(z|x^{(i)})}{p_\theta(z)} =$$

$$= KL\left(q_\varphi(z|x^{(i)}) \,\|\, p_\theta(z)\right)$$

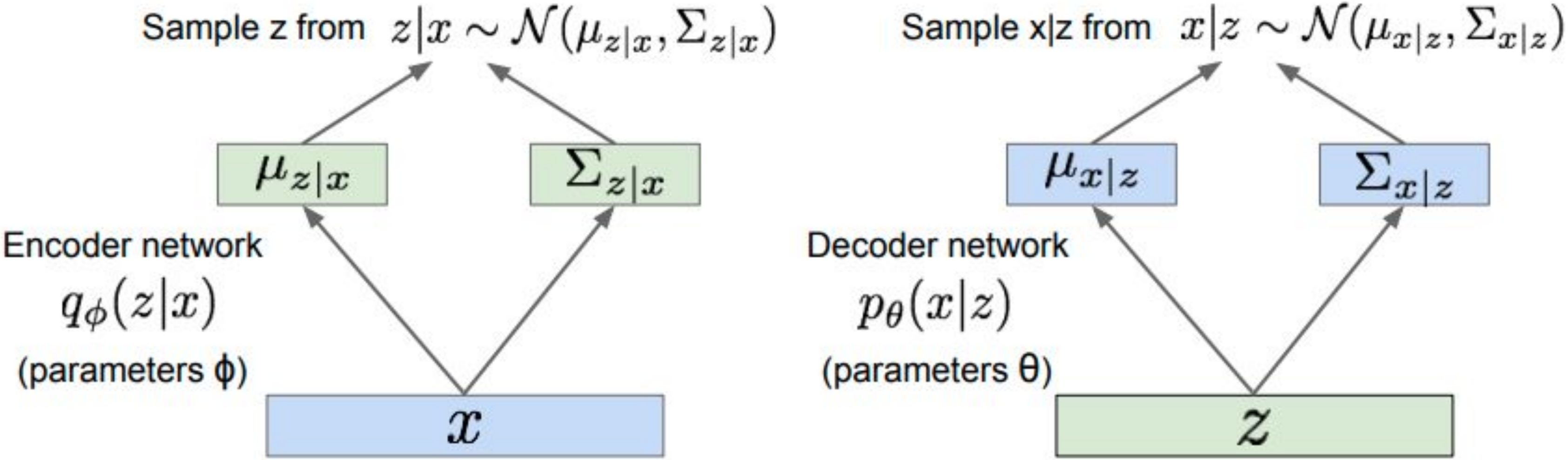



$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

decoder; contains sampling, BUT we can deal with that

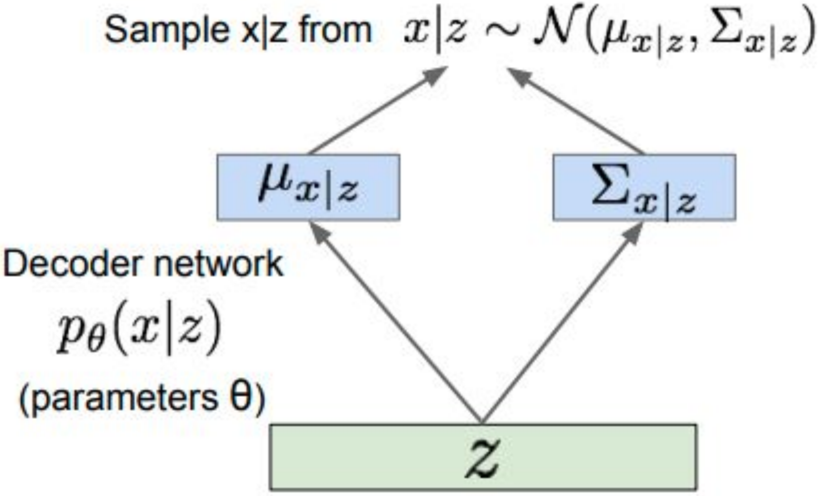
has differentiable closed form in case of Gaussians, see Appendix B in the orig.paper

Remember? Instead of mapping the input to a fixed vector, we want to map the input onto a distribution - probabilistic spin on autoencoders.



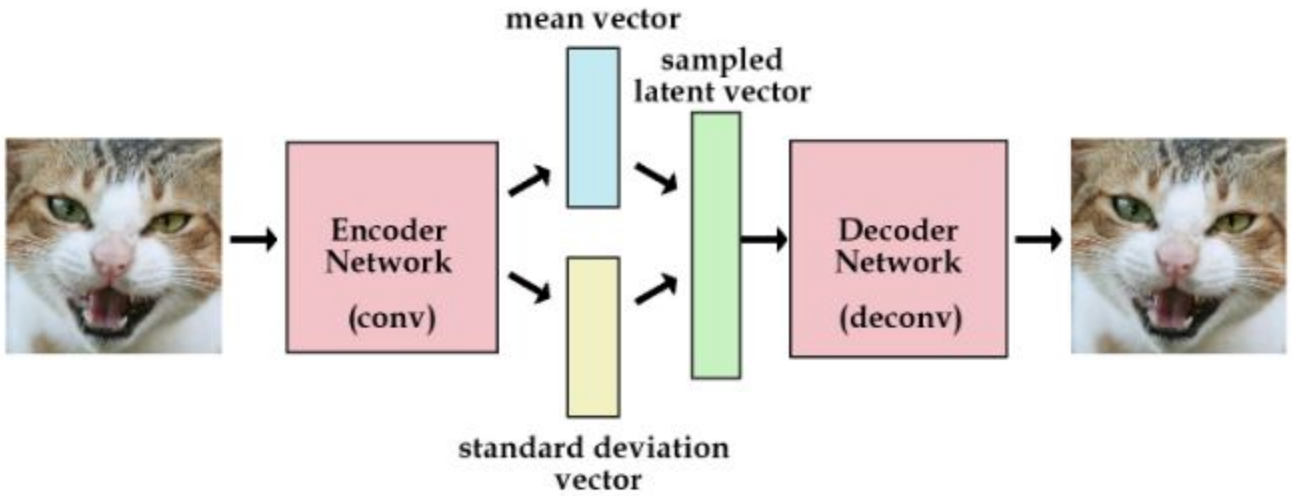

*CS231n, 2017, Lecture 13.*

ones. In VAEs, the choice of this output distribution is often Gaussian, i.e., $P(X|z;\theta) = \mathcal{N}(X|f(z;\theta), \sigma^2 * I)$. That is, it has mean $f(z;\theta)$ and covariance equal to the identity matrix $I$ times some scalar $\sigma$ (which is a hyperparameter). This replacement is necessary to formalize the intuition that some $z$ needs to result in samples that are merely *like* $X$. In general, and particularly early in training, our model will not produce outputs that are identical to any particular $X$. By having a Gaussian distribution, we can use gradient descent (or any other optimization technique) to increase $P(X)$ by making $f(z;\theta)$ approach $X$ for some $z$, i.e., gradually making the training data more likely under the generative model. This wouldn't be possible if $P(X|z)$ was a Dirac delta function, as it would be if we used $X = f(z;\theta)$ deterministically! Note that the output distribution is not *required* to be Gaussian: for instance, if $X$ is binary, then $P(X|z)$ might be a Bernoulli parameterized by $f(z;\theta)$. The important property is simply that $P(X|z)$ can be computed, and is continuous in $\theta$. From here onward, we will omit $\theta$ from $f(z;\theta)$ to avoid clutter.

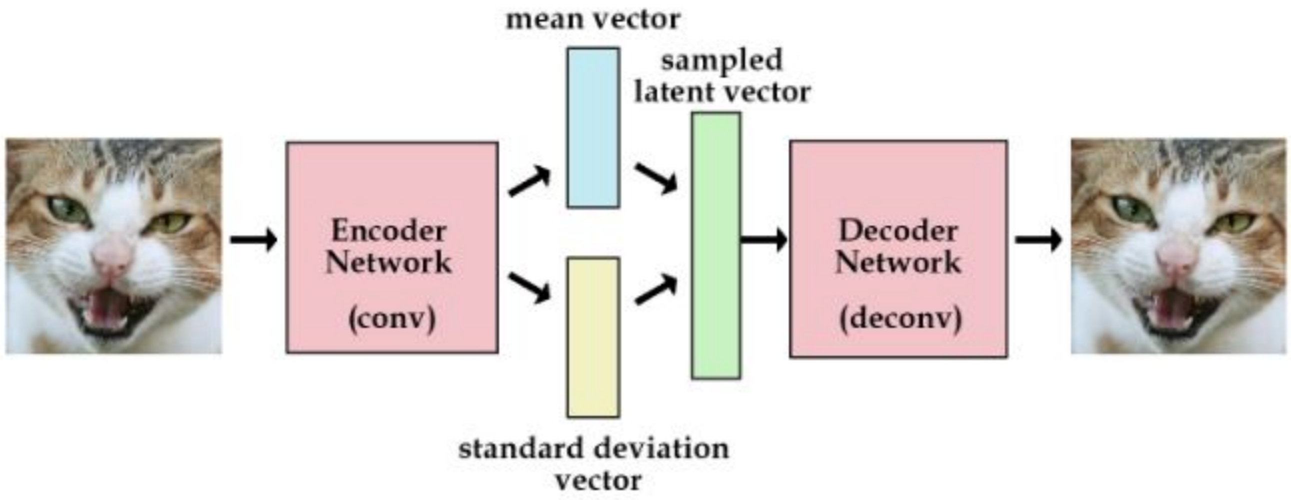



*Carl Doersch, Tutorial on Variational Autoencoders, 2016*

# That’s why I like this one better.



# That’s why I like this one better.




Not just because of cats :)

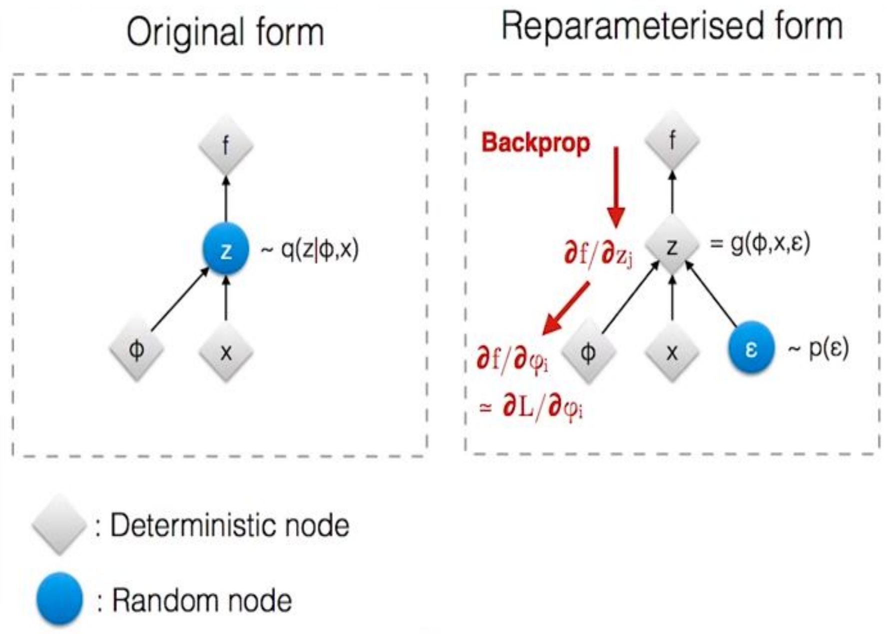
We **sample** latent vector. And we can’t backpropagate through a sampling node.

# Reparametrization trick

Instead of having $z \sim \mathcal{N}(\mu, \sigma^2)$, we can have $z = \mu + \sigma\epsilon$ with $\epsilon \sim \mathcal{N}(0, 1)$.

# Reparametrization trick

Instead of having $z \sim \mathcal{N}(\mu, \sigma^2)$, we can have $z = \mu + \sigma\epsilon$ with $\epsilon \sim \mathcal{N}(0, 1)$.




https://www.youtube.com/watch?v=9zKuYvjFFS8

# Reparametrization trick

Instead of having $z \sim \mathcal{N}(\mu, \sigma^2)$, we can have $z = \mu + \sigma\epsilon$ with $\epsilon \sim \mathcal{N}(0, 1)$.

Again: can it be Gaussian only?

# Reparametrization trick

Instead of having $z \sim \mathcal{N}(\mu, \sigma^2)$, we can have $z = \mu + \sigma\epsilon$ with $\epsilon \sim \mathcal{N}(0,1)$.

Again: can it be Gaussian only? $\boxed{\int q_\phi(\mathbf{z}|\mathbf{x}) f(\mathbf{z})\, d\mathbf{z} \simeq \frac{1}{L}\sum_{l=1}^{L} f(g_\phi(\mathbf{x}, \boldsymbol{\epsilon}^{(l)}))}$ - diff. estimator

Take, for example, the univariate Gaussian case: let $z \sim p(z|x) = \mathcal{N}(\mu, \sigma^2)$. In this case, a valid reparameterization is $z = \mu + \sigma\epsilon$, where $\epsilon$ is an auxiliary noise variable $\epsilon \sim \mathcal{N}(0,1)$. Therefore, $\mathbb{E}_{\mathcal{N}(z;\mu,\sigma^2)}[f(z)] = \mathbb{E}_{\mathcal{N}(\epsilon;0,1)}[f(\mu + \sigma\epsilon)] \simeq \frac{1}{L}\sum_{l=1}^{L} f(\mu + \sigma\epsilon^{(l)})$ where $\epsilon^{(l)} \sim \mathcal{N}(0,1)$.

For which $q_\phi(\mathbf{z}|\mathbf{x})$ can we choose such a differentiable transformation $g_\phi(.)$ and auxiliary variable $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$? Three basic approaches are:

1. Tractable inverse CDF. In this case, let $\boldsymbol{\epsilon} \sim \mathcal{U}(\mathbf{0}, \mathbf{I})$, and let $g_\phi(\boldsymbol{\epsilon}, \mathbf{x})$ be the inverse CDF of $q_\phi(\mathbf{z}|\mathbf{x})$. Examples: Exponential, Cauchy, Logistic, Rayleigh, Pareto, Weibull, Reciprocal, Gompertz, Gumbel and Erlang distributions.
2. Analogous to the Gaussian example, for any "location-scale" family of distributions we can choose the standard distribution (with location $= 0$, scale $= 1$) as the auxiliary variable $\boldsymbol{\epsilon}$, and let $g(.) =$ location $+$ scale $\cdot\, \boldsymbol{\epsilon}$. Examples: Laplace, Elliptical, Student's t, Logistic, Uniform, Triangular and Gaussian distributions.
3. Composition: It is often possible to express random variables as different transformations of auxiliary variables. Examples: Log-Normal (exponentiation of normally distributed variable), Gamma (a sum over exponentially distributed variables), Dirichlet (weighted sum of Gamma variates), Beta, Chi-Squared, and F distributions.

*Kingma et al, Auto-Encoding Variational Bayes, 2014*

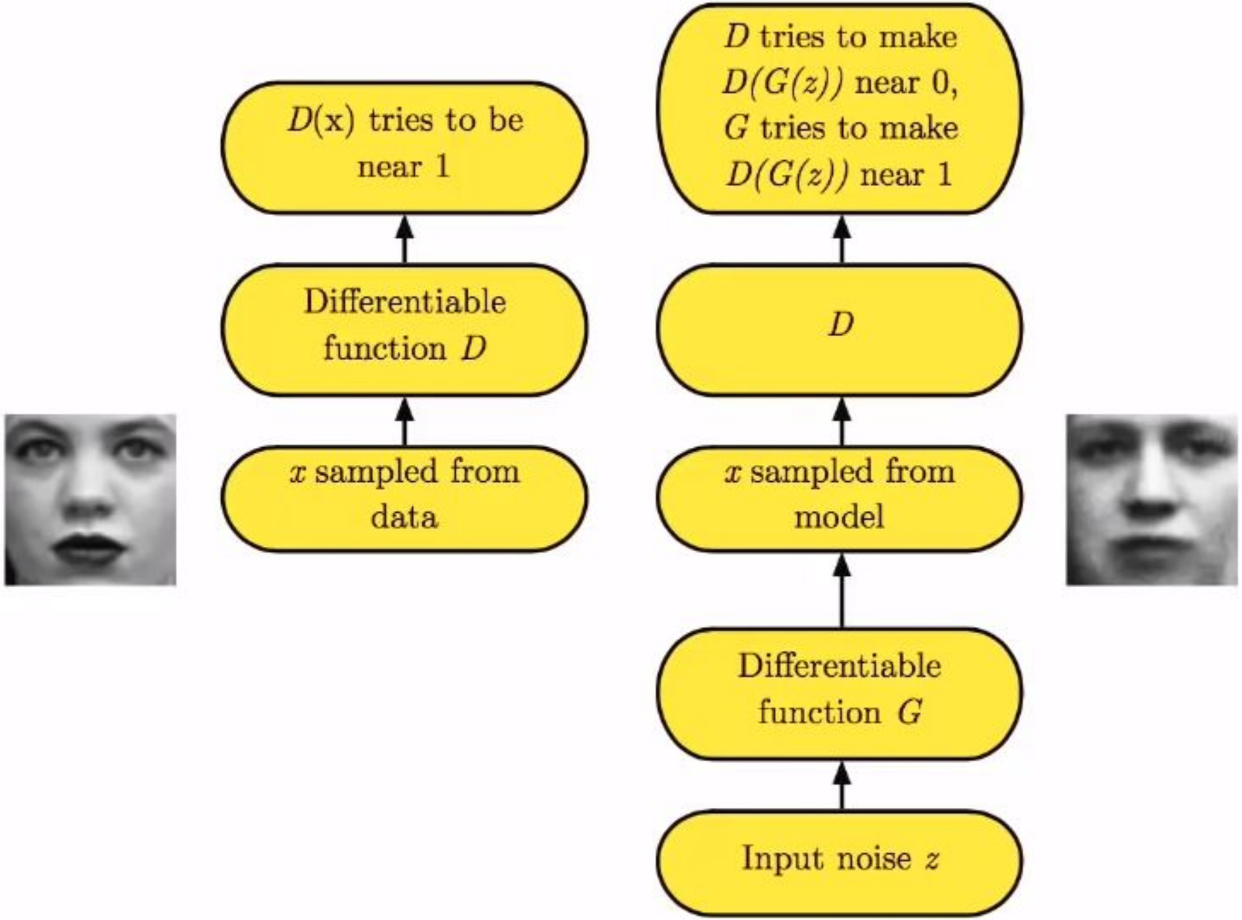
# Generative adversarial networks

# GANs

- both players are neural networks
- worst case input for one network is produced by another network

*Goodfellow et al, Tutorial on GANs, 2016, NIPS*

# Adversarial Nets Framework




(Goodfellow

# Minimax Game

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$
$$J^{(G)} = -J^{(D)}$$

- Nash equilibrium (given unlimited capabilities)
- JS divergence between the data and the model

$$D_G^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$$

$$\begin{aligned}
C(G) =& \max_D V(G, D) \\
=& \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}}[\log(1 - D_G^*(G(\boldsymbol{z})))] \\
=& \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x} \sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))] \\
=& \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x} \sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right]
\end{aligned}$$

$$C(G) = -\log(4) + KL\left(p_{\text{data}} \left\| \frac{p_{\text{data}} + p_g}{2}\right.\right) + KL\left(p_g \left\| \frac{p_{\text{data}} + p_g}{2}\right.\right)$$

$$C(G) = -\log(4) + 2 \cdot JSD\left(p_{\text{data}} \| p_g\right)$$

*Goodfellow et al, Generative Adversarial Nets, 2014*

# Non-saturating game

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$

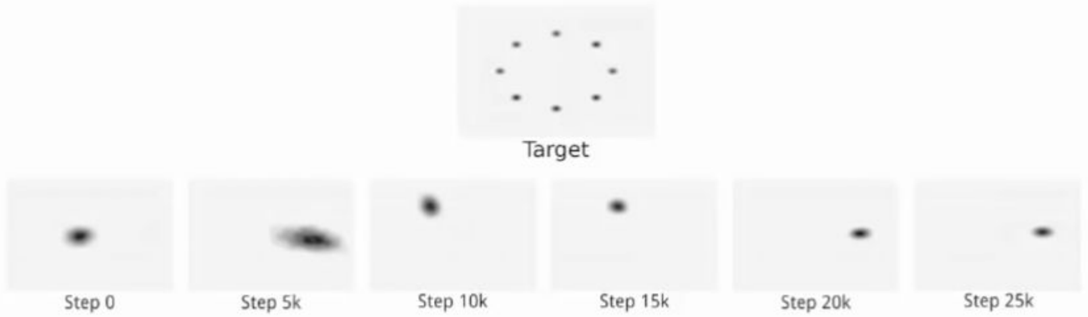
$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D\left(G(\boldsymbol{z})\right)$$

- G can still learn even when D rejects all G’s samples
- G maximizes log-prob of the D being mistaken

# Mode collapse

$$\min_G \max_D V(G,D) \neq \max_D \min_G V(G,D)$$

- when we have LHS expression - convergence to correct distribution
- when we have RHS expression - we have mode collapse



# Was Jürgen Schmidhuber right when he claimed credit for GANs at NIPS 2016?

Answer · Follow · 32 · Request

## 1 Answer

Ian Goodfellow, I invented generative adversarial networks

Answered Mar 21, 2017

He isn't claiming credit for GANs, exactly. It's more complicated.

You can see what he wrote in his own words when he was a reviewer of the NIPS 2014 submission on GANs: Export Reviews, Discussions, Author Feedback and Meta-Reviews

He's the reviewer that asked us to change the name of GANs to "inverse PM."

Here's the paper he believes is not being sufficiently acknowledged: http://ftp://ftp.idsia.ch/pub/juergen/factorial.pdf

I don't like that there is no good way to have issues like this adjudicated. I contacted the NIPS organizers and asked if there is a way for Jürgen to file a complaint about me and have a committee of NIPS representatives judge whether my publication treats his unfairly. They said there is no such process available.

I personally don't think that there is any significant connection between predictability minimization and GANs. I have never had any problem acknowledging connections between GANs and other algorithms that actually are related, like noise-contrastive estimation and self-supervised boosting.

Jürgen and I intend to write a paper together soon describing the similarities and differences between PM and GANs, assuming we're able to agree on what those are.

23.3k views · View Upvoters · View Sharers · Answer requested by Juan Manuel Pérez Rúa

Upvote · 435 Share · 1