University of Tehran

March 2021

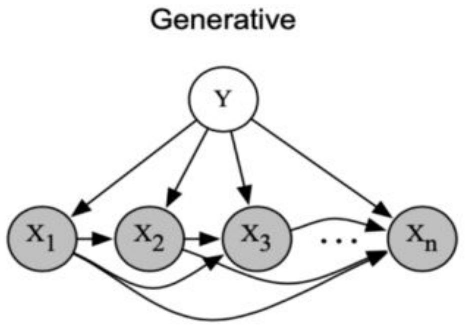
Generative Models: VAE + GAN

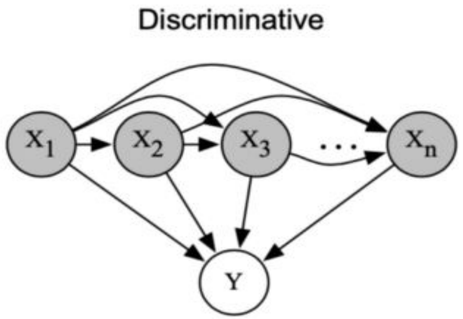
Ali Izadi

Table of Contents

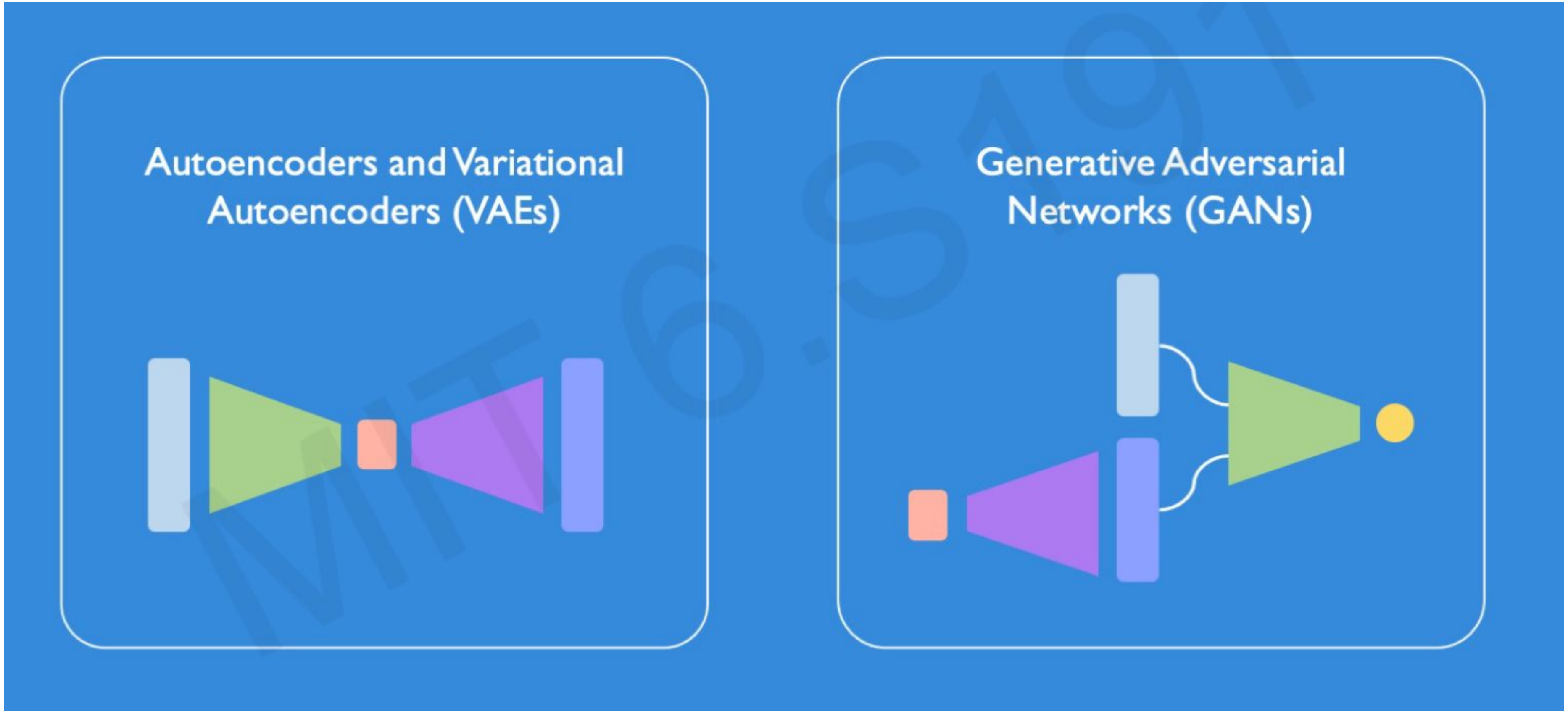
Generative Models

- To get the conditional probability **P(Y|X)**, generative models estimate the prior **P(Y)** and likelihood **P(X|Y)** from training data and use Bayes rule to calculate the posterior **P(Y |X)**
- On the other hand, discriminative models directly assume functional form for **P(Y|X)** and estimate parameters of **P(Y|X)** directly from training data.

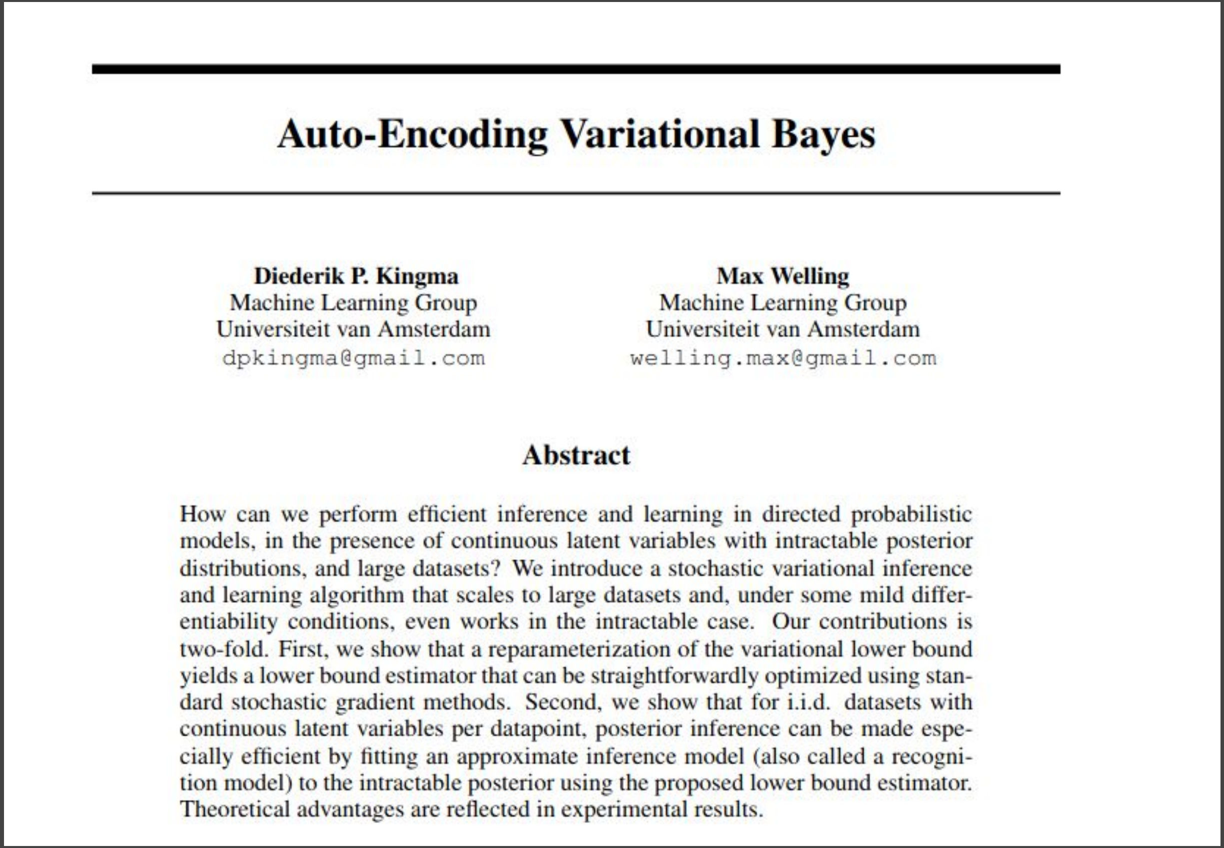




Generative Models


Autoencoders and Variational Autoencoders (VAEs)
Generative Adversarial Networks (GANs)

Variational Autoencoder



Auto-Encoding Variational Bayes

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

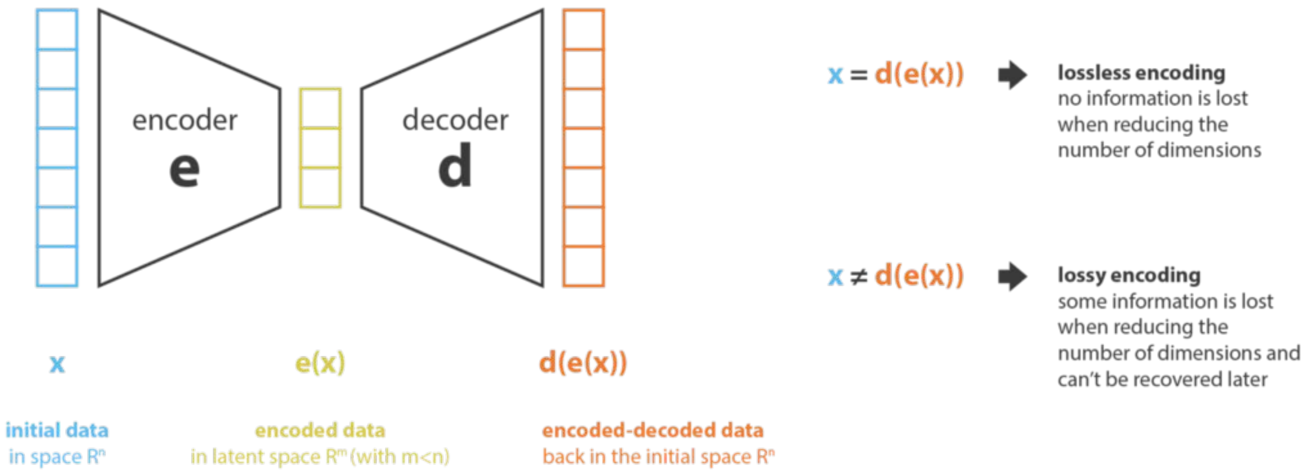
Max Welling
Machine Learning Group
Universiteit van Amsterdam
welling.max@gmail.com

Abstract

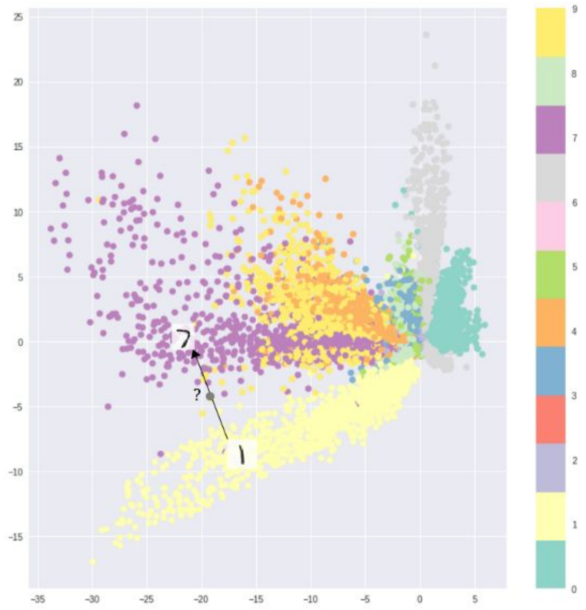
How can we perform efficient inference and learning in directed probabilistic models, in the presence of continuous latent variables with intractable posterior distributions, and large datasets? We introduce a stochastic variational inference and learning algorithm that scales to large datasets and, 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.

Dimensionality reduction can then be interpreted as data compression where the encoder compress the data (from the initial space to the **encoded space**, also called **latent space**) whereas the decoder decompress them

keeps the **maximum** of information when **encoding** and, so, has the **minimum** of **reconstruction error** when decoding

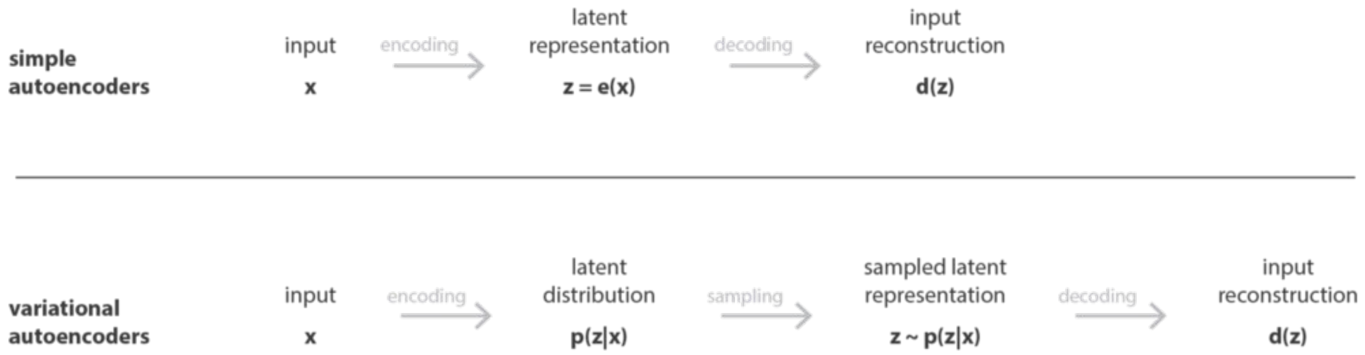

encoder
e
decoder
d
x
e(x)
d(e(x))
initial data
in space R^n
encoded data
in latent space R^m (with m<n)
encoded-decoded data
back in the initial space R^n
$x = d(e(x))$
lossless encoding
no information is lost when reducing the number of dimensions
$x \neq d(e(x))$
lossy encoding
some information is lost when reducing the number of dimensions and can't be recovered later

The fundamental problem with autoencoders, for generation, is that the latent space they convert their inputs to and where their encoded vectors lie, **may not be continuous**.

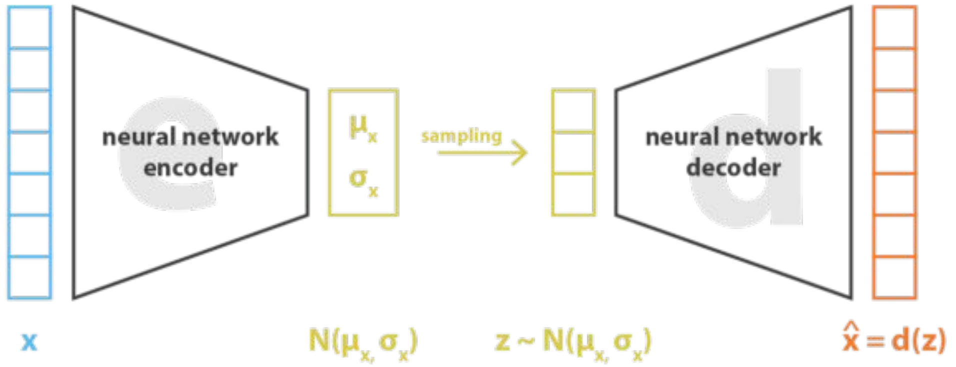


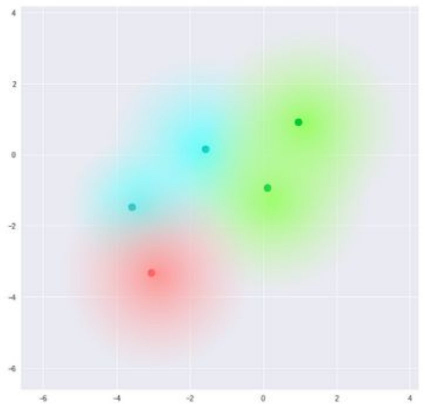
Optimizing purely for reconstruction loss (**MNIST dataset**)

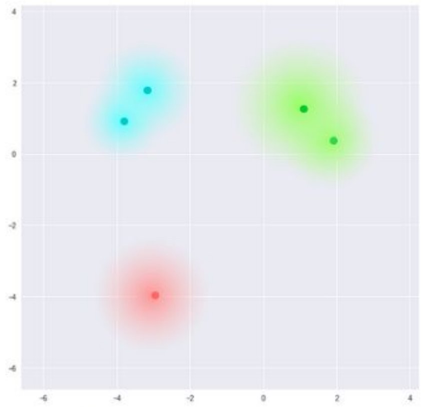
If the space has discontinuities (eg. gaps between clusters) and you sample/generate a variation from there, the decoder will simply generate an unrealistic output, because the decoder has *no idea* how to deal with that region of the latent space a variational autoencoder can be defined as being an autoencoder whose training is **regularised to avoid overfitting** and ensure that the latent space has good properties that **enable generative process**.



instead of encoding an input as a single point, we **encode it as a distribution** over the latent space 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. This allows the decoder to not just decode single, specific encodings in the latent space



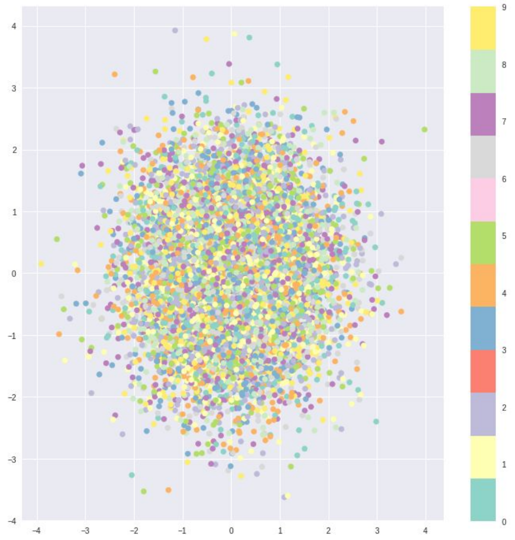

What we require


What we may inadvertently end up with

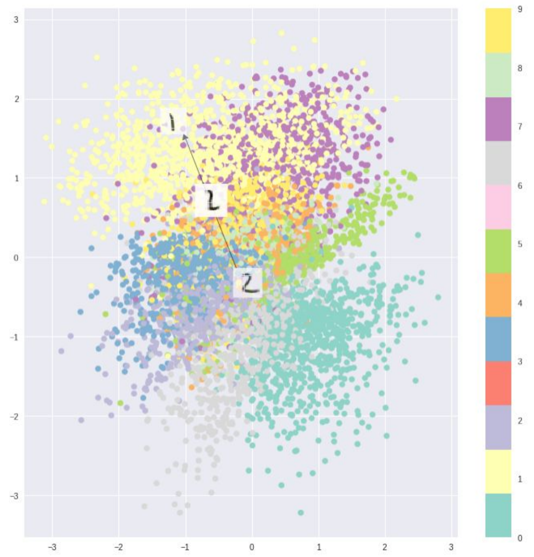
Ideally, we want **overlap between samples** that are not very similar too, in order to **interpolate** *between* classes.

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 allows the decoder to efficiently reconstruct the *training* data.**

$D_{KL}(q_\theta(z|x_i)||p(z))$ where $p(z) = N(0, 1)$

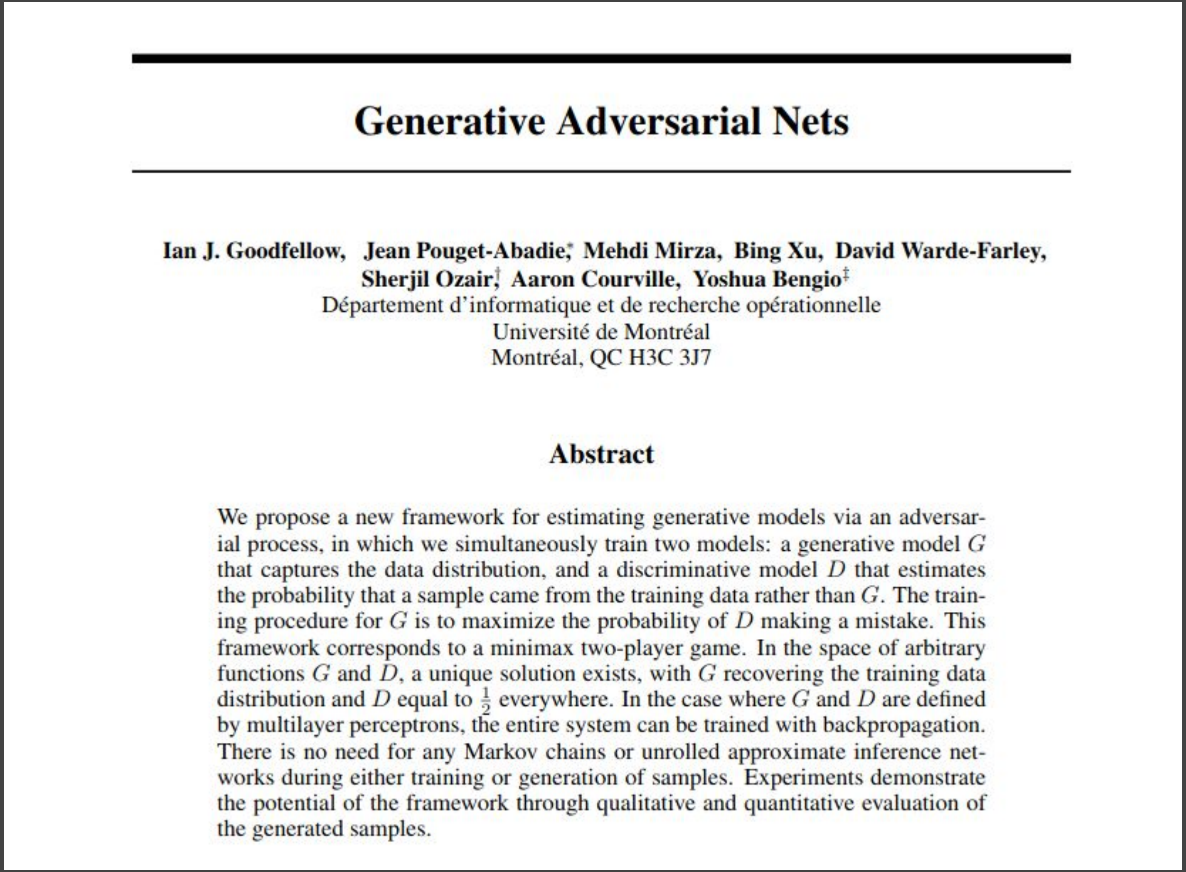


Optimizing the two together, however, results in the generation of a latent space which maintains the similarity of nearby encodings on the ***local scale*** via clustering, yet ***globally,*** is very densely packed near the latent space.



maximize $-D_{KL}(q_\theta(z|x_i)||p(z)) + E_{\sim q_\theta(z|x_i)}[\log p_\phi(x_i|z)]$

Generative Adversarial Network



Generative Adversarial Nets

Ian J. Goodfellow, Jean Pouget-Abadie[*], Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair[†], Aaron Courville, Yoshua Bengio[‡]
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

GAN

A generative adversarial network (GAN) has two parts:

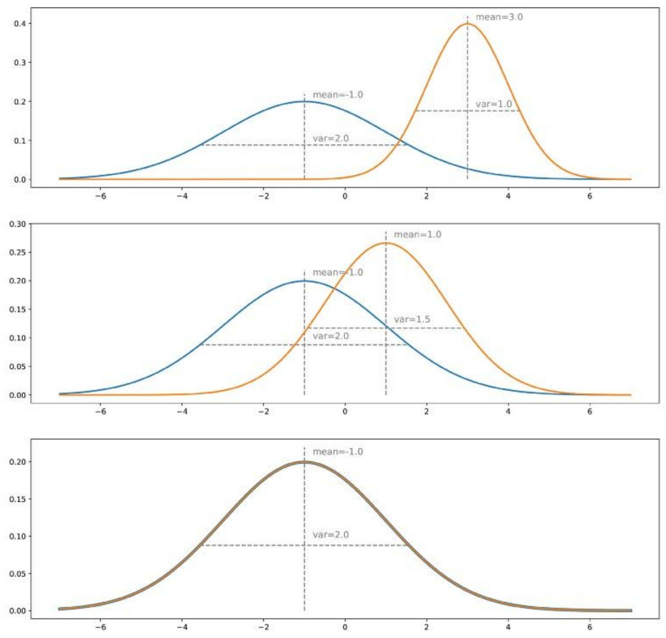
- The **generator** learns to generate plausible data. The generated instances become negative training examples for the discriminator.
- The **discriminator** learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results.

01 | When training begins, the generator produces obviously fake data, and the discriminator quickly learns to tell that it's fake.

02 | As training progresses, the generator gets closer to producing output that can fool the discriminator.

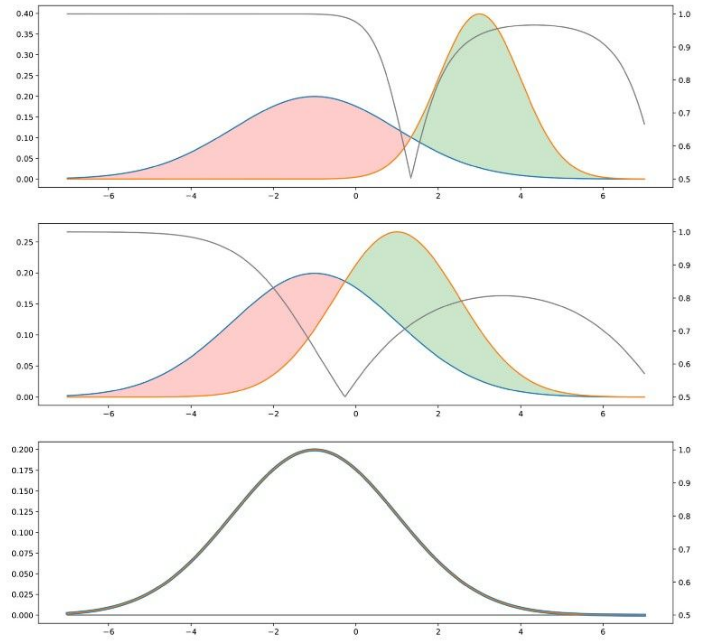
03 | Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real, and its accuracy decreases.

direct matching method



adjusting iteratively the generator (gradient descent iterations) to correct the measured difference/error between true and generated distributions.

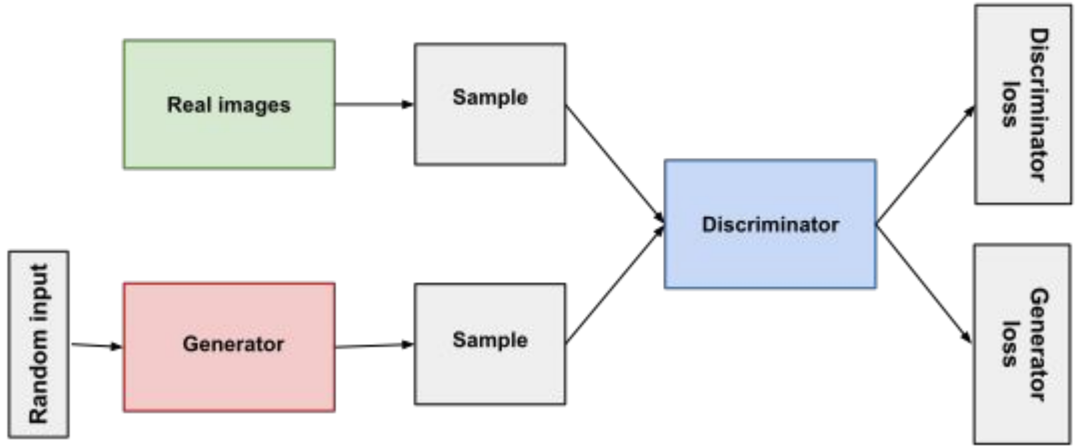
indirect matching method (Adversarial)



If we want to fool the discriminator, we have to bring the generated distribution close to the true one. The discriminator will have the most difficulty to predict the class when the two distributions will be equal in all points: in this case, for each point there are equal chances for it to be “true” or “generated” and then the discriminator can’t do better than being true in one case out of two in average.

GAN

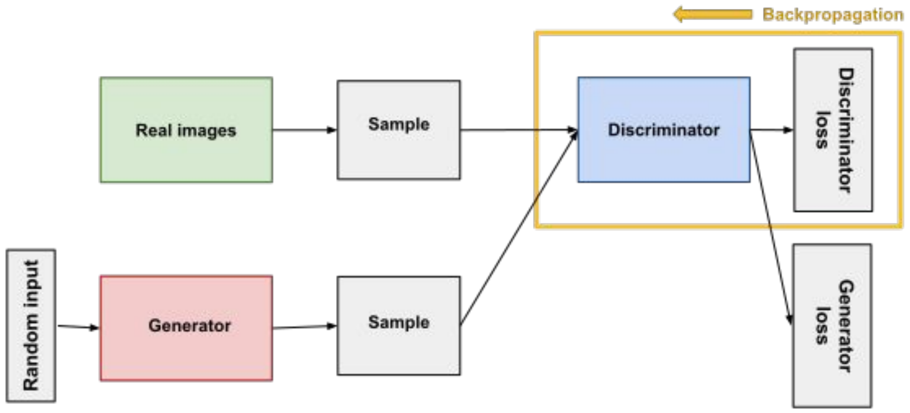
Both the **generator** and the **discriminator** are neural networks. The generator output is connected directly to the discriminator input. Through **backpropagation**, the discriminator's classification provides a signal that the generator uses to update its weights.



GAN training proceeds in alternating periods:

1. The discriminator trains for one or more epochs.
2. The generator trains for one or more epochs.
3. Repeat steps 1 and 2 to continue to train the generator and discriminator networks.

GAN (Discriminator Training)

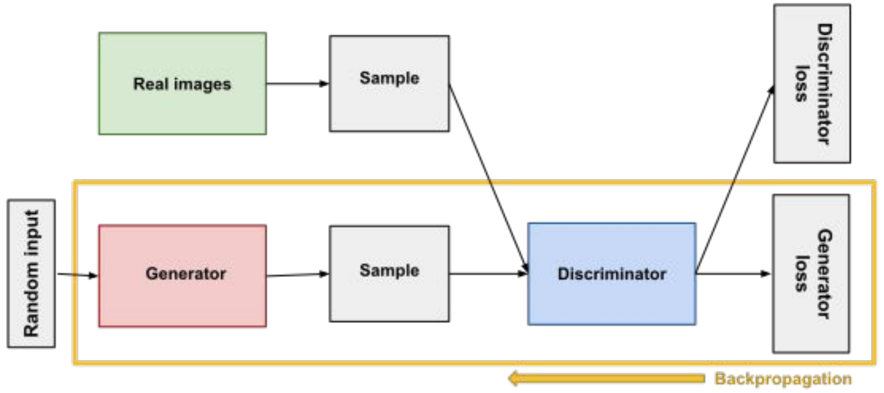


During discriminator training:

1. The discriminator classifies both real data and fake data from the generator.
2. The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real.
3. The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.

maximize $$E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$$

GAN (Generator Training)



we train the generator with the following procedure:

1. Sample random noise.
2. Produce generator output from sampled random noise.
3. Get discriminator "Real" or "Fake" classification for generator output.
4. Calculate loss from discriminator classification.
5. Backpropagate through both the discriminator and generator to obtain gradients.
6. Use gradients to change only the generator weights.

minimize $$E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$$

GAN (convergence)

Convergence:
As the generator improves with training, the discriminator performance gets worse because the discriminator can't easily tell the difference between real and fake. If the generator succeeds perfectly, then the discriminator has a 50% accuracy. In effect, the discriminator flips a coin to make its prediction.

Problem with convergence:
If the GAN continues training past the point when the discriminator is giving completely random feedback, then the generator starts to train on junk feedback, and its own quality may collapse.

For a GAN, convergence is often a fleeting, rather than stable, state.

The original GAN paper notes that the above minimax loss function can cause the GAN to get stuck in the early stages of GAN training when the discriminator's job is very easy. The paper therefore suggests modifying the generator loss so that the generator tries to maximize log D(G(z))

GAN hacks

https://github.com/soumith/ganhacks (starter from "How to Train a GAN?" at NIPS2016)

- Normalize the inputs
- A modified loss function(In practice, works well:Flip labels when training generator: real = fake, fake = real)
- Use a spherical Z
- Use Soft and Noisy Labels
- ...

VAE vs GAN

GAN probably learns more about *"how can I make an image look real in general"* rather than *"how can I memorise this particular set of images with the greatest accuracy/efficiency"*

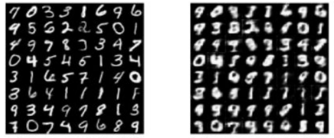

a. Samples generated by GAN b. Samples generated by VAE

Fig. 3. Comparison of sampled images of the two models based on the MNIST dataset.

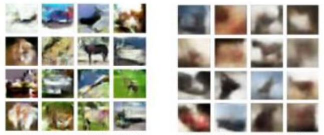

a. Samples generated by GAN b. Samples generated by VAE

Fig. 4. Comparison of sampled images of the two models based on the CIFAR10 dataset.

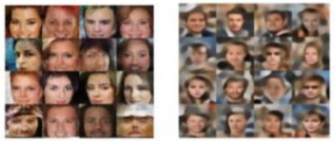
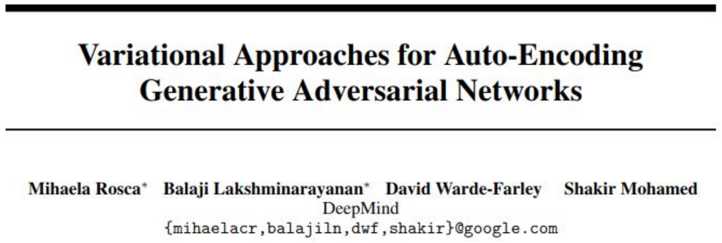
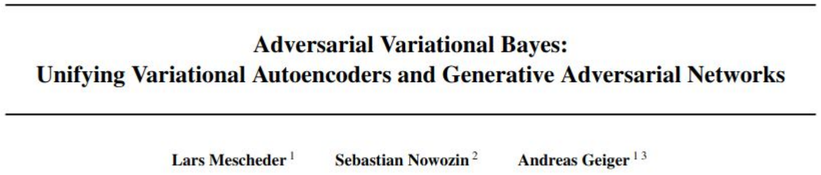

a. Samples generated by GAN b. Samples generated by VAE

Fig. 5. Comparison of sampled images of the two models based on the CelebA dataset.

- Both models generate images which can easily be recogniz- ed as digits. While the GANs generates sharper images, the VAE tends to smooth the edges of the digits.
- The images of the VAE are once again blurry and no re- alistic objects can be reco- gnized. The GAN generates images with sharper edges
- The GAN produces again much sharper images than the VAE. Nevertheless, the faces produced by the VAE own a more natural appearance.

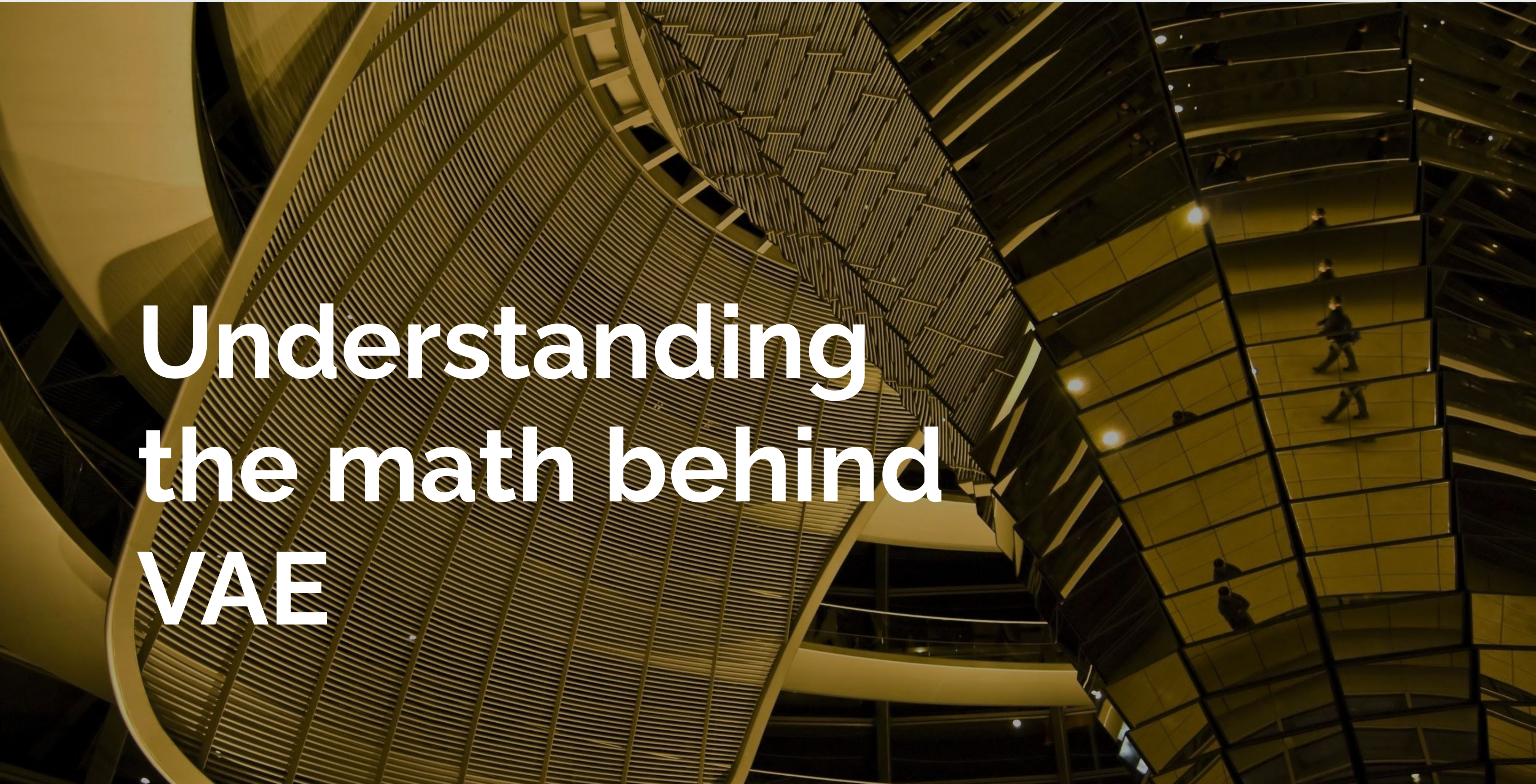

Variational Approaches for Auto-Encoding Generative Adversarial Networks

Mihaela Rosca* Balaji Lakshminarayanan* David Warde-Farley Shakir Mohamed
DeepMind
{mihaelacr,balajiln,dwf,shakir}@google.com


Adversarial Variational Bayes: Unifying Variational Autoencoders and Generative Adversarial Networks

Lars Mescheder[1] Sebastian Nowozin[2] Andreas Geiger[1 3]

Deep Generative Models: Practical Comparison Between Variational Autoencoders and Generative Adversarial Networks, Mohamed EL-KADDOURY



Understanding the math behind VAE

VAE

$$D_{KL}(p(x)||q(x)) := E_{\sim p}[\Delta I] = \int (\Delta I) p(x) dx = \int p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

Note the KL divergence is always non-negative, i.e.,

$$D_{KL}(q(x)||p(x)) = -\int q(x) \log\left(\frac{p(x)}{q(x)}\right) dx \geq 0 \quad (12)$$

$$D_{KL}\left(q_\theta(z|x_i)||p(z|x_i)\right) = -\int q_\theta(z|x_i) \log\left(\frac{p(z|x_i)}{q_\theta(z|x_i)}\right) dz \geq 0 \quad (17)$$

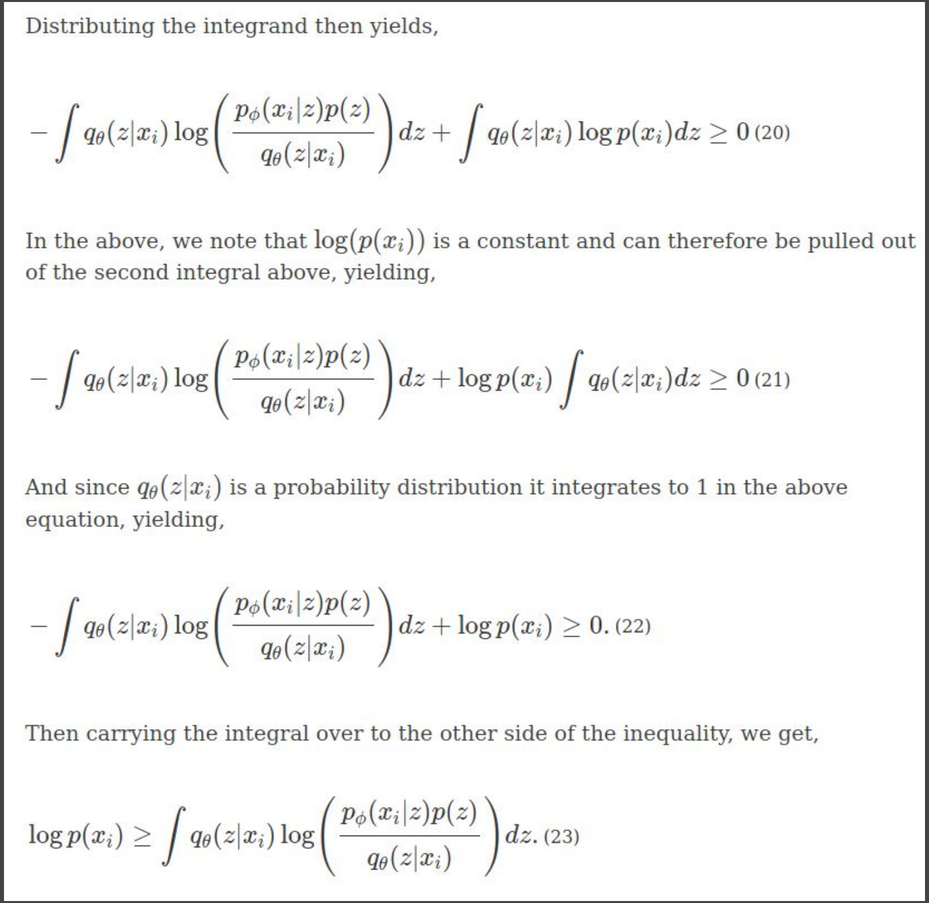
Applying Bayes' theorem to the above equation yields,

$$D_{KL}\left(q_\theta(z|x_i)||p(z|x_i)\right) = -\int q_\theta(z|x_i) \log\left(\frac{p_\phi(x_i|z)p(z)}{q_\theta(z|x_i)p(x_i)}\right) dz \geq 0 \quad (18)$$

This can be broken down using laws of logarithms, yielding,

$$D_{KL}\left(q_\theta(z|x_i)||p(z|x_i)\right) = -\int q_\theta(z|x_i) \left[\log\left(\frac{p_\phi(x_i|z)p(z)}{q_\theta(z|x_i)}\right) - \log p(x_i)\right] dz \geq 0 \quad (19)$$

VAE



Distributing the integrand then yields,

$$-\int q_\theta(z|x_i)\log\left(\frac{p_\phi(x_i|z)p(z)}{q_\theta(z|x_i)}\right)dz + \int q_\theta(z|x_i)\log p(x_i)dz \geq 0 \quad (20)$$

In the above, we note that $\log(p(x_i))$ is a constant and can therefore be pulled out of the second integral above, yielding,

$$-\int q_\theta(z|x_i)\log\left(\frac{p_\phi(x_i|z)p(z)}{q_\theta(z|x_i)}\right)dz + \log p(x_i)\int q_\theta(z|x_i)dz \geq 0 \quad (21)$$

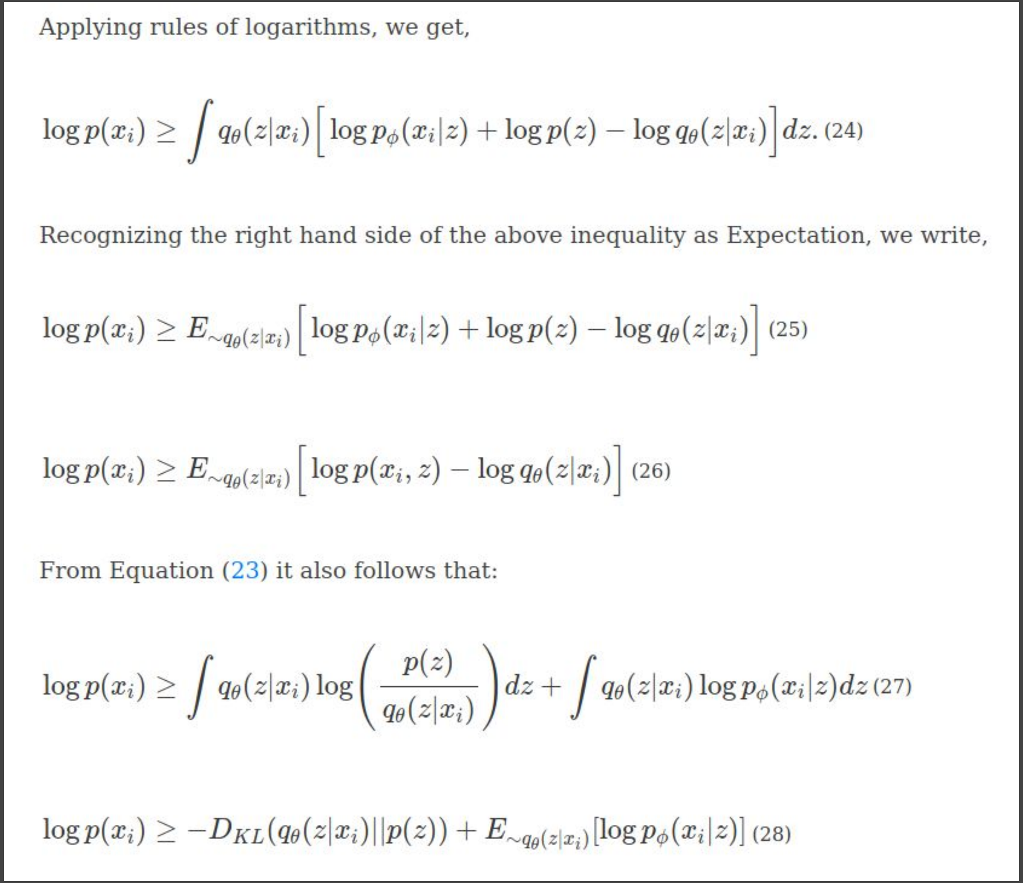
And since $q_\theta(z|x_i)$ is a probability distribution it integrates to 1 in the above equation, yielding,

$$-\int q_\theta(z|x_i)\log\left(\frac{p_\phi(x_i|z)p(z)}{q_\theta(z|x_i)}\right)dz + \log p(x_i) \geq 0. \quad (22)$$

Then carrying the integral over to the other side of the inequality, we get,

$$\log p(x_i) \geq \int q_\theta(z|x_i)\log\left(\frac{p_\phi(x_i|z)p(z)}{q_\theta(z|x_i)}\right)dz. \quad (23)$$

VAE



Applying rules of logarithms, we get,

$$\log p(x_i) \geq \int q_\theta(z|x_i)\Big[\log p_\phi(x_i|z) + \log p(z) - \log q_\theta(z|x_i)\Big]dz. \quad (24)$$

Recognizing the right hand side of the above inequality as Expectation, we write,

$$\log p(x_i) \geq E_{\sim q_\theta(z|x_i)}\Big[\log p_\phi(x_i|z) + \log p(z) - \log q_\theta(z|x_i)\Big] \quad (25)$$

$$\log p(x_i) \geq E_{\sim q_\theta(z|x_i)}\Big[\log p(x_i, z) - \log q_\theta(z|x_i)\Big] \quad (26)$$

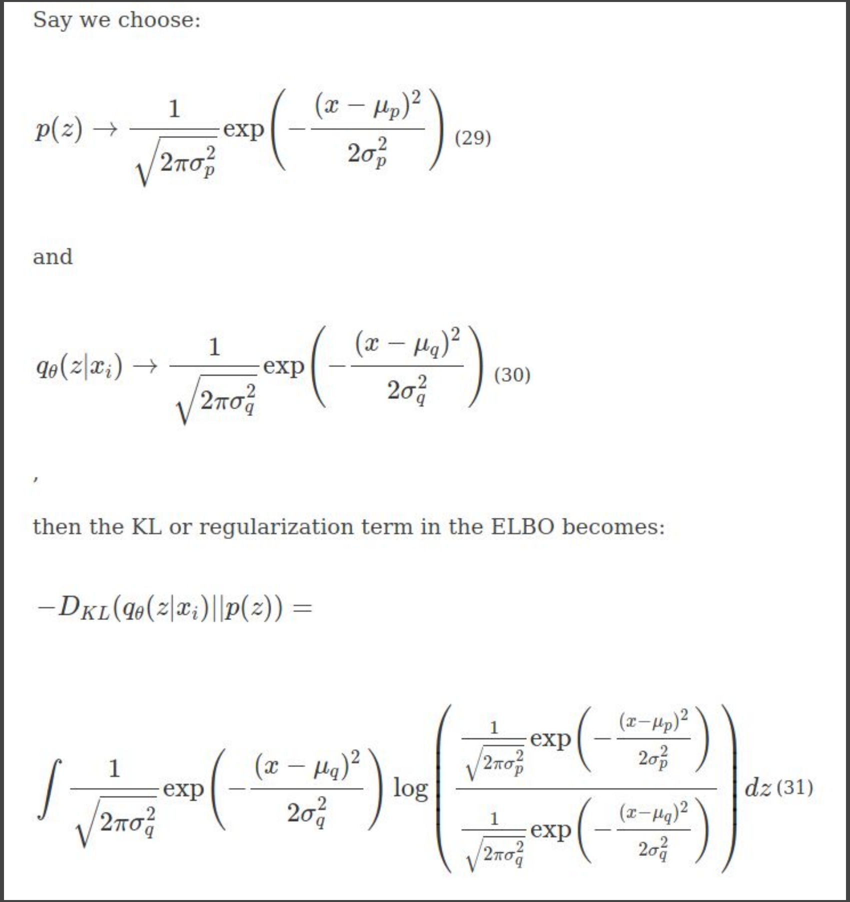
From Equation (23) it also follows that:

$$\log p(x_i) \geq \int q_\theta(z|x_i)\log\left(\frac{p(z)}{q_\theta(z|x_i)}\right)dz + \int q_\theta(z|x_i)\log p_\phi(x_i|z)dz \quad (27)$$

$$\log p(x_i) \geq -D_{KL}(q_\theta(z|x_i)||p(z)) + E_{\sim q_\theta(z|x_i)}[\log p_\phi(x_i|z)] \quad (28)$$

The right hand side of the last equation is the **Evidence Lower Bound (ELBO)** also known as the variational lower bound

VAE



Say we choose:

$$p(z) \rightarrow \frac{1}{\sqrt{2\pi\sigma_p^2}} \exp\left(-\frac{(x-\mu_p)^2}{2\sigma_p^2}\right) \quad (29)$$

and

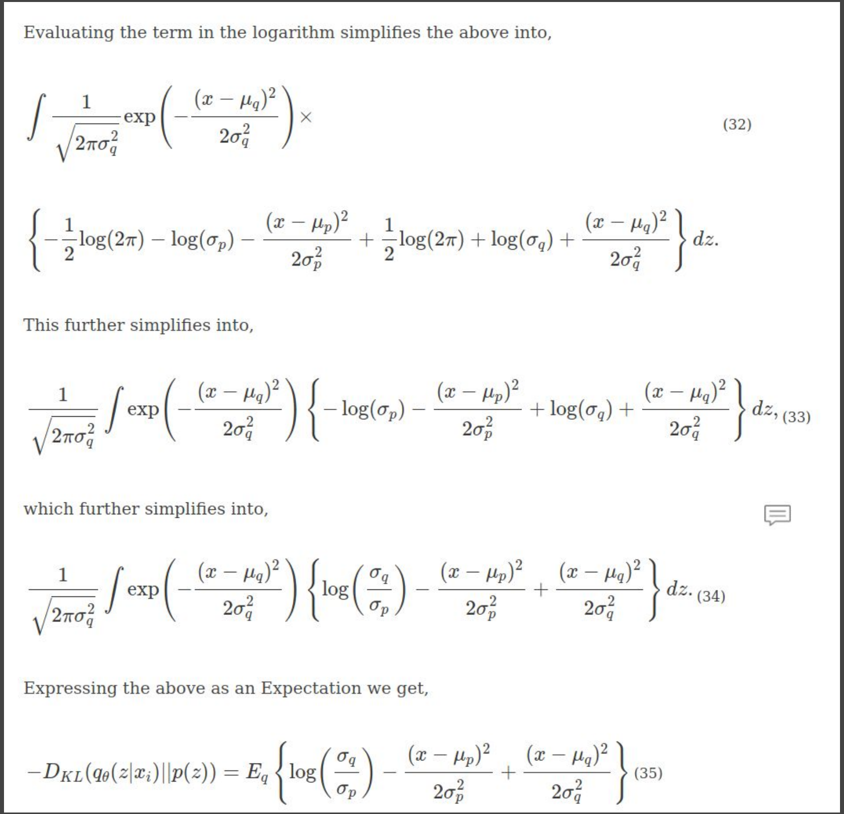
$$q_\theta(z|x_i) \rightarrow \frac{1}{\sqrt{2\pi\sigma_q^2}} \exp\left(-\frac{(x-\mu_q)^2}{2\sigma_q^2}\right) \quad (30)$$

,

then the KL or regularization term in the ELBO becomes:

$$-D_{KL}(q_\theta(z|x_i)||p(z)) =$$

$$\int \frac{1}{\sqrt{2\pi\sigma_q^2}} \exp\left(-\frac{(x-\mu_q)^2}{2\sigma_q^2}\right) \log\left(\frac{\frac{1}{\sqrt{2\pi\sigma_p^2}} \exp\left(-\frac{(x-\mu_p)^2}{2\sigma_p^2}\right)}{\frac{1}{\sqrt{2\pi\sigma_q^2}} \exp\left(-\frac{(x-\mu_q)^2}{2\sigma_q^2}\right)}\right) dz \quad (31)$$



Evaluating the term in the logarithm simplifies the above into,

$$\int \frac{1}{\sqrt{2\pi\sigma_q^2}} \exp\left(-\frac{(x-\mu_q)^2}{2\sigma_q^2}\right) \times \quad (32)$$

$$\left\{-\frac{1}{2}\log(2\pi) - \log(\sigma_p) - \frac{(x-\mu_p)^2}{2\sigma_p^2} + \frac{1}{2}\log(2\pi) + \log(\sigma_q) + \frac{(x-\mu_q)^2}{2\sigma_q^2}\right\} dz.$$

This further simplifies into,

$$\frac{1}{\sqrt{2\pi\sigma_q^2}} \int \exp\left(-\frac{(x-\mu_q)^2}{2\sigma_q^2}\right) \left\{-\log(\sigma_p) - \frac{(x-\mu_p)^2}{2\sigma_p^2} + \log(\sigma_q) + \frac{(x-\mu_q)^2}{2\sigma_q^2}\right\} dz, \quad (33)$$

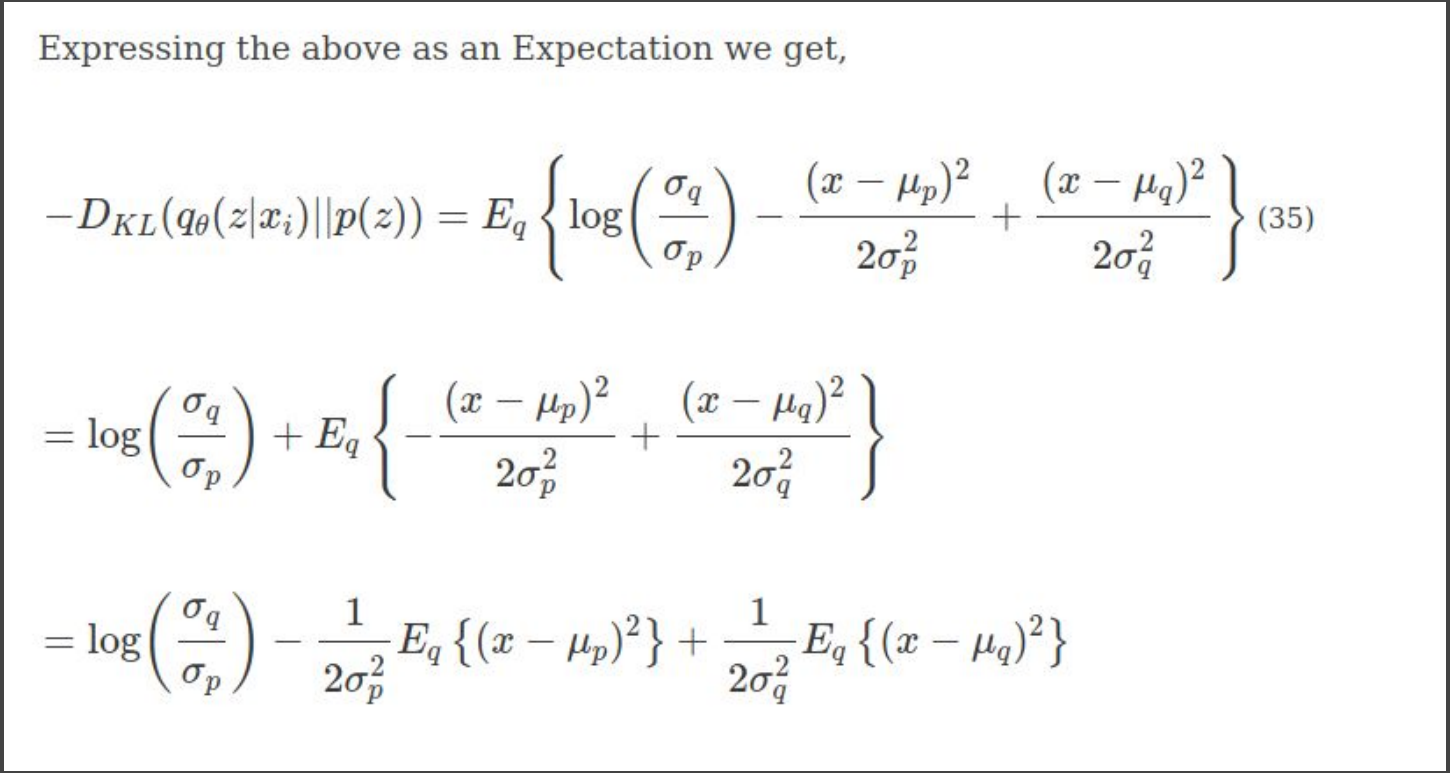
which further simplifies into,

$$\frac{1}{\sqrt{2\pi\sigma_q^2}} \int \exp\left(-\frac{(x-\mu_q)^2}{2\sigma_q^2}\right) \left\{\log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{(x-\mu_p)^2}{2\sigma_p^2} + \frac{(x-\mu_q)^2}{2\sigma_q^2}\right\} dz. \quad (34)$$

Expressing the above as an Expectation we get,

$$-D_{KL}(q_\theta(z|x_i)||p(z)) = E_q\left\{\log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{(x-\mu_p)^2}{2\sigma_p^2} + \frac{(x-\mu_q)^2}{2\sigma_q^2}\right\} \quad (35)$$

VAE



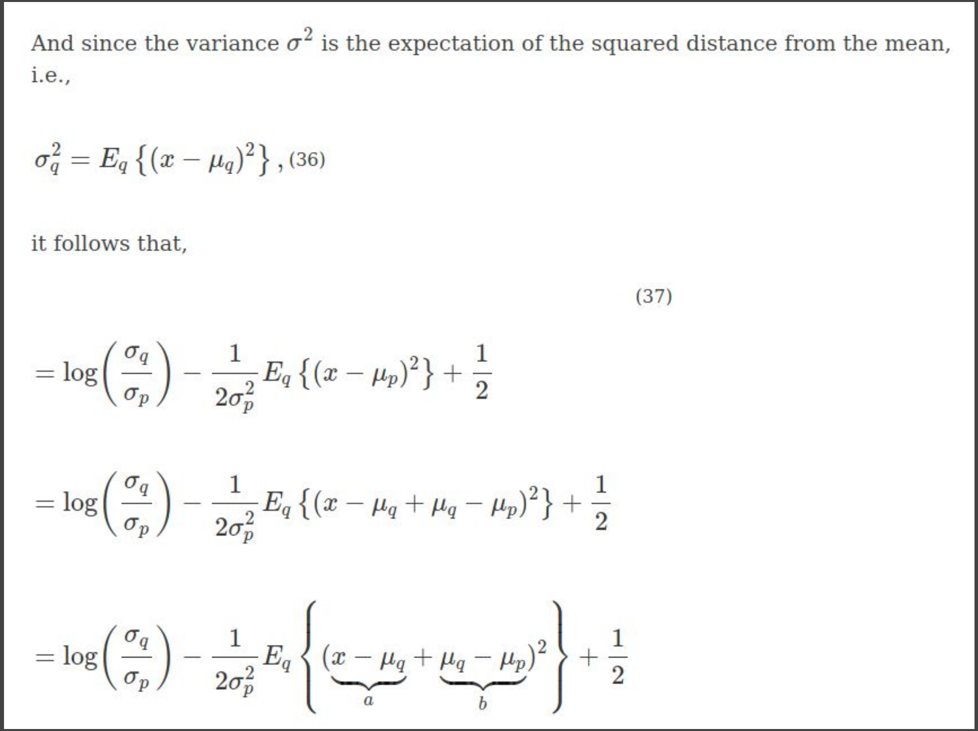
Expressing the above as an Expectation we get,

$$-D_{KL}(q_\theta(z|x_i)||p(z)) = E_q\left\{\log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{(x-\mu_p)^2}{2\sigma_p^2} + \frac{(x-\mu_q)^2}{2\sigma_q^2}\right\} \quad (35)$$

$$= \log\left(\frac{\sigma_q}{\sigma_p}\right) + E_q\left\{-\frac{(x-\mu_p)^2}{2\sigma_p^2} + \frac{(x-\mu_q)^2}{2\sigma_q^2}\right\}$$

$$= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}E_q\left\{(x-\mu_p)^2\right\} + \frac{1}{2\sigma_q^2}E_q\left\{(x-\mu_q)^2\right\}$$

VAE



And since the variance σ^2 is the expectation of the squared distance from the mean, i.e.,

$$\sigma_q^2 = E_q\left\{(x-\mu_q)^2\right\}, \text{(36)}$$

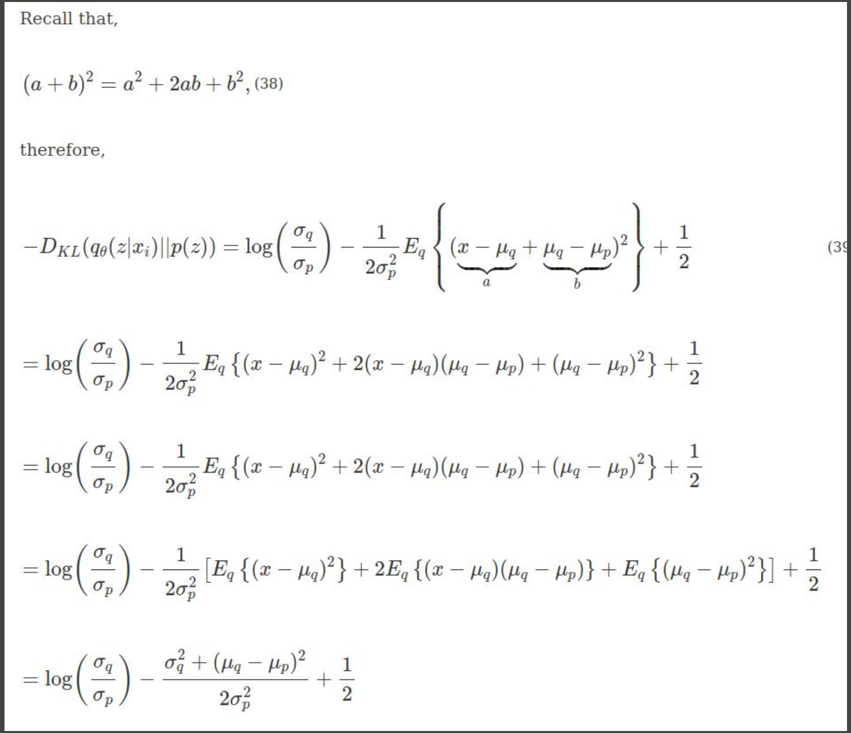
it follows that,

(37)

$$= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}E_q\left\{(x-\mu_p)^2\right\} + \frac{1}{2}$$

$$= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}E_q\left\{(x-\mu_q+\mu_q-\mu_p)^2\right\} + \frac{1}{2}$$

$$= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}E_q\left\{(\underbrace{x-\mu_q}_{a}+\underbrace{\mu_q-\mu_p}_{b})^2\right\} + \frac{1}{2}$$



Recall that,

$$(a+b)^2 = a^2 + 2ab + b^2, \text{(38)}$$

therefore,

$$-D_{KL}(q_\theta(z|x_i)||p(z)) = \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}E_q\left\{(\underbrace{x-\mu_q}_{a}+\underbrace{\mu_q-\mu_p}_{b})^2\right\} + \frac{1}{2} \quad \text{(39}$$

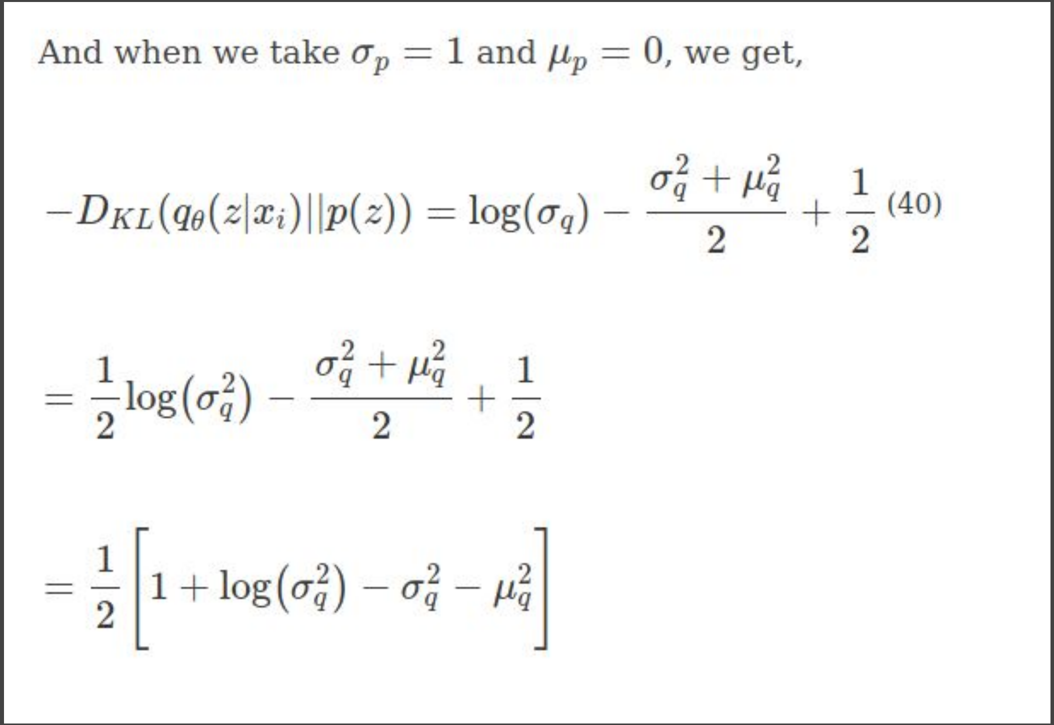
$$= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}E_q\left\{(x-\mu_q)^2 + 2(x-\mu_q)(\mu_q-\mu_p) + (\mu_q-\mu_p)^2\right\} + \frac{1}{2}$$

$$= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}E_q\left\{(x-\mu_q)^2 + 2(x-\mu_q)(\mu_q-\mu_p) + (\mu_q-\mu_p)^2\right\} + \frac{1}{2}$$

$$= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}\left[E_q\left\{(x-\mu_q)^2\right\} + 2E_q\left\{(x-\mu_q)(\mu_q-\mu_p)\right\} + E_q\left\{(\mu_q-\mu_p)^2\right\}\right] + \frac{1}{2}$$

$$= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{\sigma_q^2 + (\mu_q-\mu_p)^2}{2\sigma_p^2} + \frac{1}{2}$$

VAE



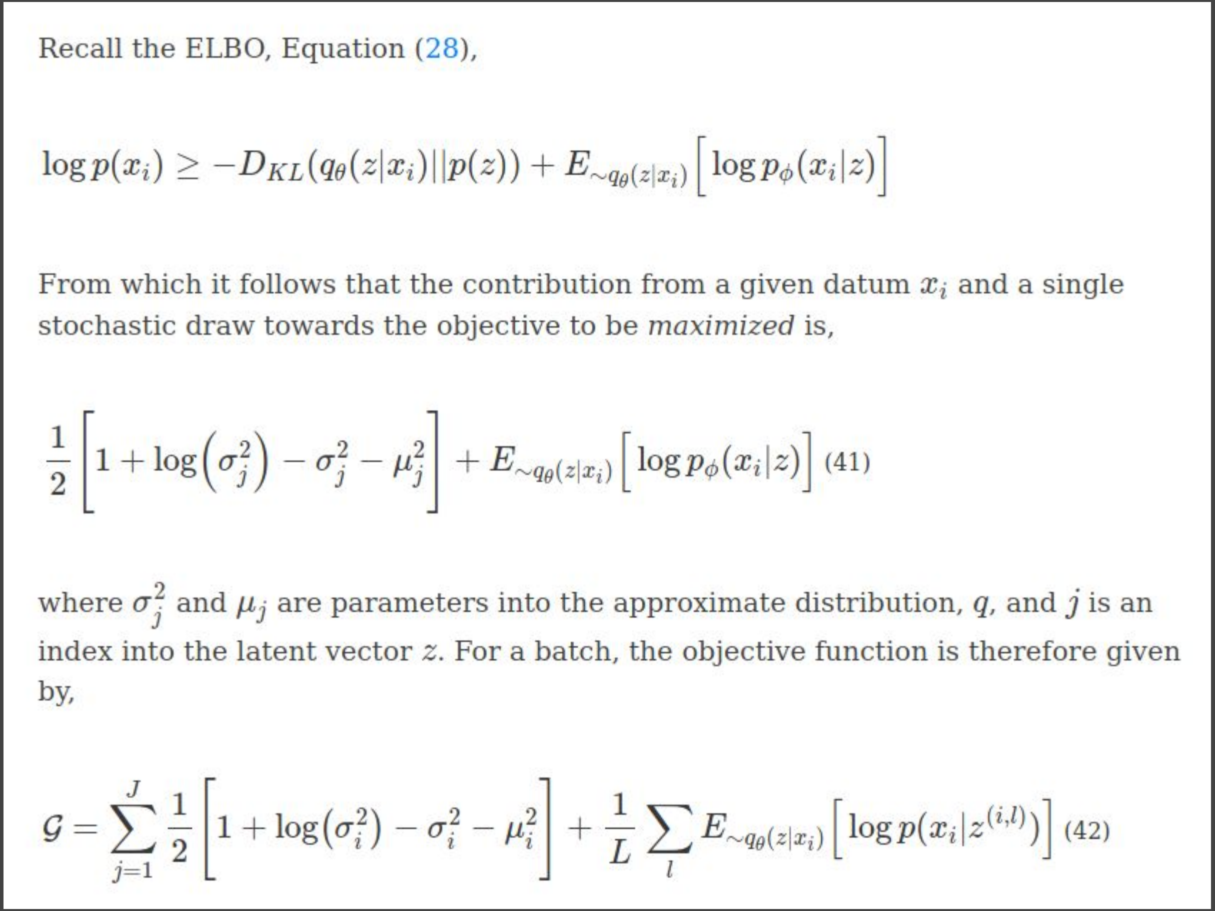
And when we take $\sigma_p = 1$ and $\mu_p = 0$, we get,

$$-D_{KL}(q_\theta(z|x_i)||p(z)) = \log(\sigma_q) - \frac{\sigma_q^2 + \mu_q^2}{2} + \frac{1}{2} \quad (40)$$

$$= \frac{1}{2}\log(\sigma_q^2) - \frac{\sigma_q^2 + \mu_q^2}{2} + \frac{1}{2}$$

$$= \frac{1}{2}\left[1 + \log(\sigma_q^2) - \sigma_q^2 - \mu_q^2\right]$$

VAE



Recall the ELBO, Equation (28),

$$\log p(x_i) \geq -D_{KL}(q_\theta(z|x_i)||p(z)) + E_{\sim q_\theta(z|x_i)}\Big[\log p_\phi(x_i|z)\Big]$$

From which it follows that the contribution from a given datum x_i and a single stochastic draw towards the objective to be *maximized* is,

$$\frac{1}{2}\left[1 + \log\left(\sigma_j^2\right) - \sigma_j^2 - \mu_j^2\right] + E_{\sim q_\theta(z|x_i)}\Big[\log p_\phi(x_i|z)\Big] \quad (41)$$

where σ_j^2 and μ_j are parameters into the approximate distribution, q, and j is an index into the latent vector z. For a batch, the objective function is therefore given by,

$$\mathcal{G} = \sum_{j=1}^{J} \frac{1}{2}\left[1 + \log(\sigma_i^2) - \sigma_i^2 - \mu_i^2\right] + \frac{1}{L}\sum_{l} E_{\sim q_\theta(z|x_i)}\Big[\log p(x_i|z^{(i,l)})\Big] \quad (42)$$

References

- https://arxiv.org/pdf/1406.2661.pdf
- https://arxiv.org/pdf/1312.6114.pdf
- https://www.groundai.com/project/tutorial-deriving-the-standard-variational-autoencoder-vae-loss-function/1
- https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73
- https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf
- https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29
- https://developers.google.com/machine-learning/gan/loss
- https://indabaxmorocco.github.io/materials/posters/El-Kaddoury1.pdf
- https://datascience.stackexchange.com/questions/55090/what-is-the-main-difference-between-gan-and-autoencoder
- http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L4.pdf

Thank you.