Variational Autoencoder (VAE)

by Diederik P. Kingma and Max Welling

https://arxiv.org/pdf/1312.6114.pdf

1 Summary

The paper combines the traditional variational Bayes with neural networks. In particular, instead of the intractably approach by traditional variational Bayes, the paper proposes an approach to estimate the variational lower bound using neural networks. In order for gradient descent to work, the paper introduce the reparameterization trick which pushes the stochastic operation of sampling out to the input, thus enabling the gradient to flow through the networks.

2 Variational Autoencoder

Imagine you want to write a digit from 0-9 on a piece of paper. You first pick a pen, pick a digit then write it. The appearance of the digit on the paper depends on the kind of pen, the color of the ink, what digit you pick, etc. In generative models, this information is called latent variable.

We can state our generative problem as follows. Given a dataset $D = \{x^{(i)}\}_{i=1}^{N}$ are N i.i.d samples of a random variable X drawn from a hidden distribution. We further assume there is a latent variable Z and the hidden distribution is such that $P(X) = \int P(X|z)P(Z)dz$. The problem is to draw more samples from this hidden distribution.

However, different tasks require a completely different structure of the latent space $P(Z)$. Generating a digit requires different latent information than generating a song. One interesting hypothesis is that for any latent

distribution, there exists a deterministic transformation that maps a normal distribution to the latent one.

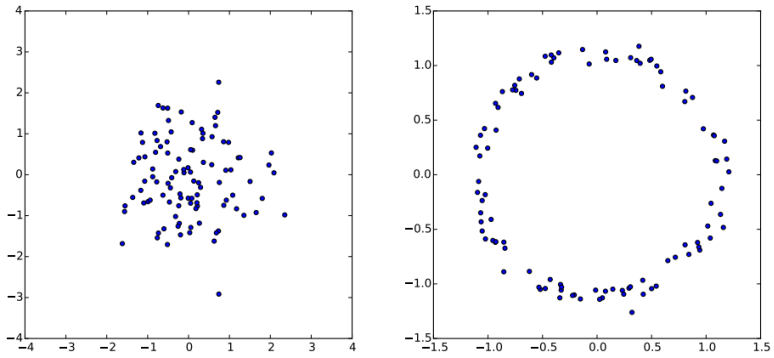


Figure 2: Given a random variable z with one distribution, we can create another random variable $X = g(z)$ with a completely different distribution. Left: samples from a gaussian distribution. Right: those same samples mapped through the function $g(z) = z/10 + z/||z||$ to form a ring. This is the strategy that VAEs use to create arbitrary distributions: the deterministic function g is learned from data.

Based on this insight, we can somewhat safely start with $P(Z) = \mathcal{N}(0,I)$. Plugging this back to the first equation gives, $P(X) = \int P(X|z \sim \mathcal{N}(0,I)).\mathcal{N}(0,I)dz$. One approach to solve this integral is using the Monte Carlo approximation,

$$P(X) = \frac{1}{n}\sum_i P(X|Z = z_i)$$

However, for high dimensional problem, n must be extremely large for this approximation be close to ground truth. The key idea in variational method is that for the most part $P(X|Z)$ will be nearly zero. So, we want to look for the "important" region in the space of Z by having another model $Q_\theta(Z|X)$ proposing the values of z that are likely to produce X,

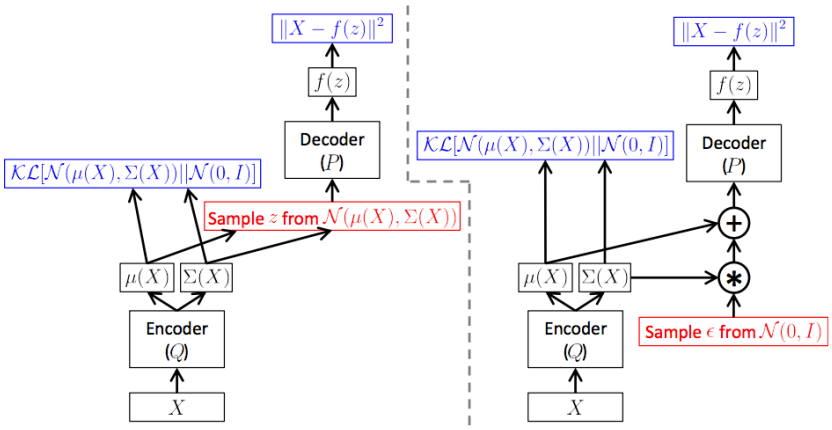
$$\begin{aligned} D_{KL}(Q_\theta(Z|X)||P(Z|X)) &= E_{z\sim Q_\theta}[logQ_\theta(Z|X) - logP(Z|X)] \\ &= E_{z\sim Q_\theta}[logQ_\theta(Z|X) - log\frac{P(X|Z)P(Z)}{P(X)}] \\ &= E_{z\sim Q_\theta}[logQ_\theta(Z|X) - logP(X|Z) - logP(Z) + logP(X)] \end{aligned}$$

$$logP(X) - \underbrace{D_{KL}(Q_\theta(Z|X)||P(Z|X))}_{\geq 0} = E_{z\sim Q_\theta}[logP(X|Z)] - E_{z\sim Q_\theta}(logQ_\theta(Z|X) - logP(Z))$$

$$= \underbrace{E_{z\sim Q_\theta}[logP(X|Z)] - D_{KL}(Q_\theta(Z|X)||P(Z))}_{\mathscr{L}(Q_\theta,Z,X)}$$

Therefore, $\mathscr{L}(Q_\theta,Z,X)$ is the evidence lower bound (ELBO) of the marginal likelihood. Note, the variational lower bound can only be as good as our estimation of $P(Z|X)$ using $Q_\theta(Z|X)$. In summary, to estimate the *RHS*, we first need to estimate $Q_\theta(Z|X)$, sample $z \sim Q_\theta(Z|X)$, then estimate $P(X|Z=z)$. The first part is called the **encoder** which encodes the input into a hidden state. The second part is called the **decoder**, which given the hidden state, infers the input. The two parts combined is called [autoencoder](https://en.wikipedia.org/wiki/Autoencoder), (hence the name variational autoencoder).

However, in order to use learn encoder and decoder jointly using gradient based methods, all operations must be differentiable. The sampling operation in the middle of the encoder and decoder is not. The paper introduces a technique called **reparameterization trick** which pushes the sampling operation to the input layer. It works by sampling $\varepsilon \sim \mathscr{N}(I,\mathscr{I})$. If the **encoder** $Q_\theta(Z|X)$ produces $\mu(X),\Sigma(X)$, then the input to the **decoder** will then be $z = \mu(X) + \Sigma^{1/2}(X) * \varepsilon$.



3 Implementation

Let's look at an example using MNIST dataset.

- The encoder takes the 28x28 and produces a n-dimensional mean and variance for z (to ensure variance is positive we predict log Σ instead, then $e^{log\Sigma}$ is positive)

```
with tf.variable_scope('encoder'):
  fc = slim.fully_connected(self.input_x, 512, activation_fn=tf.nn.relu)
  fc = slim.fully_connected(fc, 384, activation_fn=tf.nn.relu)
  fc = slim.fully_connected(fc, 256, activation_fn=tf.nn.relu)
  mu = slim.fully_connected(fc, 10, activation_fn=None)
  log_sigma = slim.fully_connected(fc, 10, activation_fn=None)
```

- Reparameterization trick, first sample $\varepsilon \sim \mathcal{N}(0,I)$, then scale $z = \mu(X) + \Sigma^{1/2}(X) * \varepsilon$,

```
with tf.variable_scope('z'):
  eps = tf.random_normal(shape=tf.shape(log_sigma), mean=0, stddev=1, dtype=tf.float32)
  self.z = mu + tf.sqrt(tf.exp(log_sigma)) * eps
```

- The decoder takes z then tries to reproduce input $\hat{x}$,

```
with tf.variable_scope('decoder'):
  dec = slim.fully_connected(self.z, 256, activation_fn=tf.nn.relu)
  dec = slim.fully_connected(dec, 384, activation_fn=tf.nn.relu)
  dec = slim.fully_connected(dec, 512, activation_fn=tf.nn.relu)
  dec = slim.fully_connected(dec, 784, activation_fn=None)
```

- Our objective is to **maximize** the evidence lower bound (ELBO),

$$\mathscr{L}(Q_\theta, X, Z) = E_{x\sim D}[E_{z\sim Q_\theta(Z|X=x)}[logP(X|Z=z)] - D_{KL}(Q_\theta(Z|X=x)||P(Z))]$$

So,

$$\begin{aligned}
\theta &= \underset{\theta}{argmax}\mathscr{L}(Q_\theta, X, Z) \\
&= \underset{\theta}{argmin} -\mathscr{L}(Q_\theta, X, Z) \\
&= \underset{\theta}{argmin} E_{x\sim D}[E_{z\sim Q_\theta(Z|X=x)}[-logP(X|Z=z)] + D_{KL}(Q_\theta(Z|X=x)||P(Z))] \\
&= \underset{\theta}{argmin} H(D, P(X|Z)) + D_{KL}(Q_\theta(Z|X)||P(Z))
\end{aligned}$$

- The implementation of the cross entropy can be as simple as,

```
self.rec = tf.reduce_mean(tf.reduce_sum(
tf.nn.sigmoid_cross_entropy_with_logits(logits=dec, labels=self.input_x), 1))
```

- Since we assume Gaussian for $P(Z)$ and $Q(Z)$, the KL loss can be calculated in closed form,

$$\begin{aligned} D_{KL}(Q_\theta(Z|X)||P(Z)) &= D_{KL}(\mathcal{N}(\mu(X),\Sigma(X))||\mathcal{N}(0,I)) \\ &= \frac{1}{2}(tr(\Sigma(X)) + \mu(X)^T\mu(X) - k - \log\det(\Sigma(X))) \end{aligned}$$

```
self.kl_loss = tf.reduce_mean(0.5 * tf.reduce_sum(
tf.exp(log_sigma) + tf.square(mu) - 1. - log_sigma, 1))
```

- The total loss would just be the sum of the two losses

```
self.loss = tf.reduce_mean(self.rec + self.kl_loss)
```

4 Results and Discussion

![reconstruct](reconstruct.png)

- Here is the results of reconstruction for 100 random images after 10 minutes of training (10000 iterations, with batch size = 100, learning rate = 1e-3, dimension of z = 10). Assume the columns are 0-indexed, then the even columns contain reconstructed images and the odd columns contain real images.

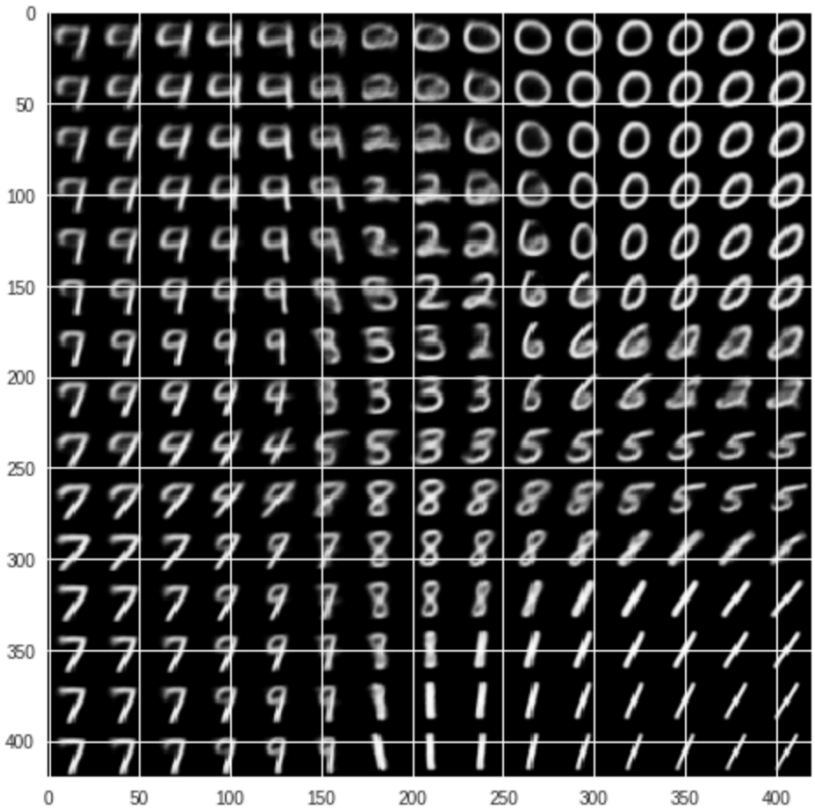
![generated](generated.png)

- Here are the digits generated by the same model from a gaussian latent space.

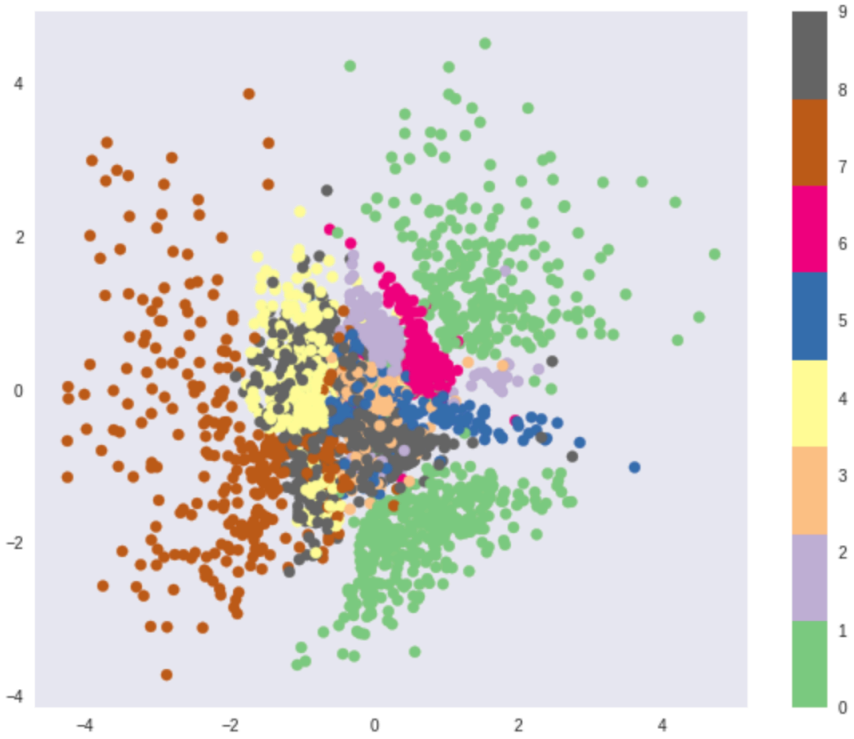
- One interesting observation in training VAE is that: initially, both reconstruction loss and KL loss go down. At one point, KL loss starts going up while total loss still goes down. To understand this phenomenon, notice that in a correctly trained VAE and with structured data, KL loss cannot be 0. If KL loss equals 0, it means there is no structured subspace of the prior distribution for the latent variable. Or, the data is almost completely random. At first, both P and Q start out to be random, so both losses go down as P gets good at producing the average of the data, and Q gets good at finding the subspace in the latent space. After this phase, P and Q start making trade-off. Specifically, if Q allows more mass in the subspace, more noise will be generated by P hence the reconstruction loss will go up and vice versa.

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
step 0, loss 545.430725, rec 545.204468, kl loss 0.226230
step 500, loss 130.925415, rec 119.364548, kl loss 11.560865
step 1000, loss 126.653328, rec 113.454300, kl loss 13.199028
step 1500, loss 114.613304, rec 100.553497, kl loss 14.059805
step 2000, loss 111.725037, rec 96.521317, kl loss 15.203723
step 2500, loss 108.033859, rec 92.640930, kl loss 15.392927
step 3000, loss 113.509247, rec 97.973763, kl loss 15.535483
step 3500, loss 111.181763, rec 95.366707, kl loss 15.815059
step 4000, loss 110.581520, rec 94.638596, kl loss 15.942927
step 4500, loss 103.349243, rec 87.326485, kl loss 16.022760
step 5000, loss 102.754456, rec 86.996712, kl loss 15.757742
step 5500, loss 110.697433, rec 94.072449, kl loss 16.624981
step 6000, loss 101.741508, rec 85.816658, kl loss 15.924849
step 6500, loss 105.159492, rec 89.188820, kl loss 15.970670
step 7000, loss 100.072983, rec 83.816742, kl loss 16.256245
step 7500, loss 102.232391, rec 85.791443, kl loss 16.440950
step 8000, loss 102.506241, rec 86.693672, kl loss 15.812566
step 8500, loss 101.539551, rec 85.425491, kl loss 16.114058
step 9000, loss 100.523689, rec 84.002754, kl loss 16.520937
step 9500, loss 104.238701, rec 87.480003, kl loss 16.758696
```

- To make things more interesting, we can train a different model with laten dimension = 2 and visualize it,



- If we sample spatially from the 2D gaussian of the latent space and with each value z, we generate a digit, then the top down view looks something like the picture above. We can see that the latent space encodes the skewness of the digits.



- If we repeat the same sampling mechanism and now encode the digits as colors, we can see that from the original Gaussian, the encoder networks has learnt a non-linear transformation that transoforms the Gaussian into different subspaces encoding different digits.

That concludes my summary and experiments with the original VAE, more theoretical analysis as well as experiments with recent variations to come.