CS7015 (Deep Learning) : Lecture 21
Variational Autoencoders

Mitesh M. Khapra

Department of Computer Science and Engineering
Indian Institute of Technology Madras
Acknowledgments

- Tutorial on Variational Autoencoders by Carl Doersch\(^1\)
- Blog on Variational Autoencoders by Jaan Altosaar\(^2\)

\(^1\)Tutorial
\(^2\)Blog
Module 21.1: Revisiting Autoencoders
Before we start talking about VAEs, let us quickly revisit autoencoders.

An autoencoder contains an encoder which takes the input $X$ and maps it to a hidden representation.

The decoder then takes this hidden representation and tries to reconstruct the input from it as $\hat{X}$.

The training happens using the following objective function:

$$\min_{W, W^*, c, b} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2$$

where $m$ is the number of training instances, $\{x_i\}_{i=1}^{m}$ and each $x_i \in \mathbb{R}^n$ ($x_{ij}$ is thus the $j$-th dimension of the $i$-th training instance).
\[ h = g(WX + b) \]
\[ \hat{X} = f(W^*h + c) \]

- But where’s the fun in this?
- We are taking an input and simply reconstructing it
- Of course, the fun lies in the fact that we are getting a good abstraction of the input
- But RBMs were able to do something more besides abstraction (they were able to do generation)
- Let us revisit generation in the context of autoencoders
Can we do generation with autoencoders?

In other words, once the autoencoder is trained can I remove the encoder, feed a hidden representation $h$ to the decoder and decode a $\hat{X}$ from it?

In principle, yes! But in practice there is a problem with this approach

$h$ is a very high dimensional vector and only a few vectors in this space would actually correspond to meaningful latent representations of our input

So of all the possible value of $h$ which values should I feed to the decoder (we had asked a similar question before: slide 67, bullet 5 of lecture 19)
\[ \hat{X} = f(W^* h + c) \]

- Ideally, we should only feed those values of \( h \) which are highly likely.
- In other words, we are interested in sampling from \( P(h|X) \) so that we pick only those \( h \)'s which have a high probability.
- But unlike RBMs, autoencoders do not have such a probabilistic interpretation.
- They learn a hidden representation \( h \) but not a distribution \( P(h|X) \).
- Similarly the decoder is also deterministic and does not learn a distribution over \( X \) (given a \( h \) we can get a \( X \) but not \( P(X|h) \) ).
We will now look at variational autoencoders which have the same structure as autoencoders but they learn a distribution over the hidden variables.
Module 21.2: Variational Autoencoders: The Neural Network Perspective
Let $\{X = x_i\}_{i=1}^N$ be the training data

We can think of $X$ as a random variable in $\mathbb{R}^n$

For example, $X$ could be an image and the dimensions of $X$ correspond to pixels of the image

We are interested in learning an abstraction (i.e., given an $X$ find the hidden representation $z$)

We are also interested in generation (i.e., given a hidden representation generate an $X$)

In probabilistic terms we are interested in $P(z|X)$ and $P(X|z)$ (to be consistent with the literature on VAEs we will use $z$ instead of $H$ and $X$ instead of $V$)
Earlier we saw RBMs where we learnt $P(z|X)$ and $P(X|z)$

Below we list certain characteristics of RBMs

**Structural assumptions:** We assume certain independencies in the Markov Network

**Computational:** When training with Gibbs Sampling we have to run the Markov Chain for many time steps which is expensive

**Approximation:** When using Contrastive Divergence, we approximate the expectation by a point estimate

(Nothing wrong with the above but we just mention them to make the reader aware of these characteristics)
We now return to our goals

**Goal 1:** Learn a distribution over the latent variables \( Q(z|X) \)

**Goal 2:** Learn a distribution over the visible variables \( P(X|z) \)

VAEs use a neural network based encoder for Goal 1

and a neural network based decoder for Goal 2

We will look at the encoder first

\[ \theta: \text{the parameters of the encoder neural network} \]

\[ \phi: \text{the parameters of the decoder neural network} \]
Encoder: What do we mean when we say we want to learn a distribution? We mean that we want to learn the parameters of the distribution.

But what are the parameters of $Q(z|X)$? Well it depends on our modeling assumption!

In VAEs we assume that the latent variables come from a standard normal distribution $\mathcal{N}(0, I)$ and the job of the encoder is to then predict the parameters of this distribution.

$X \in \mathbb{R}^n$, $\mu \in \mathbb{R}^m$ and $\Sigma \in \mathbb{R}^{m \times m}$
Now what about the decoder?

The job of the decoder is to predict a probability distribution over $X: P(X|z)$

Once again we will assume a certain form for this distribution

For example, if we want to predict 28 x 28 pixels and each pixel belongs to $\mathbb{R}$ (i.e., $X \in \mathbb{R}^{784}$) then what would be a suitable family for $P(X|z)$?

We could assume that $P(X|z)$ is a Gaussian distribution with unit variance

The job of the decoder $f$ would then be to predict the mean of this distribution as $f_\phi(z)$
What would be the objective function of the decoder?

For any given training sample $x_i$ it should maximize $P(x_i)$ given by

$$P(x_i) = \int P(z)P(x_i|z)dz$$

$$= -\mathbb{E}_{z \sim Q_\theta(z|x_i)}[\log P_\phi(x_i|z)]$$

(As usual we take log for numerical stability)
KL divergence captures the difference (or distance) between 2 distributions.

This is the loss function for one data point \((l_i(\theta))\) and we will just sum over all the data points to get the total loss \(\mathcal{L}(\theta)\)

\[
\mathcal{L}(\theta) = \sum_{i=1}^{m} l_i(\theta)
\]

In addition, we also want a constraint on the distribution over the latent variables.

Specifically, we had assumed \(P(z)\) to be \(\mathcal{N}(0, I)\) and we want \(Q(z|X)\) to be as close to \(P(z)\) as possible.

Thus, we will modify the loss function such that

\[
l_i(\theta, \phi) = -\mathbb{E}_{z \sim Q_\theta(z|x_i)} \left[ \log P_\phi(x_i|z) \right] + KL(Q_\theta(z|x_i) || P(z))
\]
The second term in the loss function can actually be thought of as a regularizer.

It ensures that the encoder does not cheat by mapping each $x_i$ to a different point (a normal distribution with very low variance) in the Euclidean space.

In other words, in the absence of the regularizer the encoder can learn a unique mapping for each $x_i$ and the decoder can then decode from this unique mapping.

Even with high variance in samples from the distribution, we want the decoder to be able to reconstruct the original data very well (motivation similar to the adding noise).

To summarize, for each data point we predict a distribution such that, with high probability a sample from this distribution should be able to reconstruct the original data point.

But why do we choose a normal distribution? Isn’t it too simplistic to assume that $z$ follows a normal distribution?

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim Q_\theta(z|x_i)}[\log P_\phi(x_i|z)] + KL(Q_\theta(z|x_i)||P(z))$$
Isn’t it a very strong assumption that $P(z) \sim \mathcal{N}(0, I)$?

For example, in the 2-dimensional case how can we be sure that $P(z)$ is a normal distribution and not any other distribution?

The key insight here is that any distribution in $d$ dimensions can be generated by the following steps:

Step 1: Start with a set of $d$ variables that are normally distributed (that’s exactly what we are assuming for $P(z)$)

Step 2: Mapping these variables through a sufficiently complex function (that’s exactly what the first few layers of the decoder can do)

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim Q_\theta(z|x_i)} [\log P_\phi(x_i|z)] + KL(Q_\theta(z|x_i) || P(z))$$
\[ l_i(\theta, \phi) = -\mathbb{E}_{z \sim Q_\theta(z|x_i)} \left[ \log P_\phi(x_i|z) \right] + KL(Q_\theta(z|x_i)\| P(z)) \]

- In particular, note that in the adjoining example if \( z \) is 2-D and normally distributed then \( f(z) \) is roughly ring shaped (giving us the distribution in the bottom figure)

\[ f(z) = \frac{z}{10} + \frac{z}{||z||} \]

- A non-linear neural network, such as the one we use for the decoder, could learn a complex mapping from \( z \) to \( f_\phi(z) \) using its parameters \( \phi \)

- The initial layers of a non-linear decoder could learn their weights such that the output is \( f_\phi(z) \)

- The above argument suggests that even if we start with normally distributed variables the initial layers of the decoder could learn a complex transformation of these variables say \( f_\phi(z) \) if required

- The objective function of the decoder will ensure that an appropriate transformation of \( z \) is learnt to reconstruct \( X \)
Module 21.3: Variational autoencoders: (The graphical model perspective)
- Here we can think of $z$ and $X$ as random variables.
- We are then interested in the joint probability distribution $P(X,z)$ which factorizes as $P(X,z) = P(z)P(X|z)$.
- This factorization is natural because we can imagine that the latent variables are fixed first and then the visible variables are drawn based on the latent variables.
- For example, if we want to draw a digit we could first fix the latent variables: the digit, size, angle, thickness, position and so on and then draw a digit which corresponds to these latent variables.
- And of course, unlike RBMs, this is a directed graphical model.
Now at inference time, we are given an $X$ (observed variable) and we are interested in finding the most likely assignments of latent variables $z$ which would have resulted in this observation. Mathematically, we want to find

$$P(z|X) = \frac{P(X|z)P(z)}{P(X)}$$

This is hard to compute because the LHS contains $P(X)$ which is intractable

$$P(X) = \int P(X|z)P(z)dz$$

$$= \int \int \ldots \int P(X|z_1, z_2, \ldots, z_n)P(z_1, z_2, \ldots, z_n)dz_1, \ldots dz_n$$

In RBMs, we had a similar integral which we approximated using Gibbs Sampling.

VAEs, on the other hand, cast this into an optimization problem and learn the parameters of the optimization problem.
Specifically, in VAEs, we assume that instead of \( P(z|X) \) which is intractable, the posterior distribution is given by \( Q_\theta(z|X) \)

Further, we assume that \( Q_\theta(z|X) \) is a Gaussian whose parameters are determined by a neural network \( \mu, \Sigma = g_\theta(X) \)

The parameters of the distribution are thus determined by the parameters \( \theta \) of a neural network

Our job then is to learn the parameters of this neural network
But what is the objective function for this neural network?

Well we want the proposed distribution $Q_\theta(z|X)$ to be as close to the true distribution.

We can capture this using the following objective function:

$$\text{minimize } KL(Q_\theta(z|X)||P(z|X))$$

What are the parameters of the objective function? (they are the parameters of the neural network - we will return back to this again)
Let us expand the KL divergence term

\[
D[Q_\theta(z|X)||P(z|X)] = \int Q_\theta(z|X) \log Q_\theta(z|X) dz - \int Q_\theta(z|X) \log P(z|X) dz
= \mathbb{E}_{z \sim Q_\theta(z|X)}[\log Q_\theta(z|X) - \log P(z|X)]
\]

For shorthand we will use \( \mathbb{E}_Q = \mathbb{E}_{z \sim Q_\theta(z|X)} \)

Substituting \( P(z|X) = \frac{P(X|z)P(z)}{P(X)} \), we get

\[
D[Q_\theta(z|X)||P(z|X)] = \mathbb{E}_Q[\log Q_\theta(z|X) - \log P(X|z) - \log P(z) + \log P(X)]
= \mathbb{E}_Q[\log Q_\theta(z|X) - \log P(z)] - \mathbb{E}_Q[\log P(X|z)] + \log P(X)
= D[Q_\theta(z|X)||p(z)] - \mathbb{E}_Q[\log P(X|z)] + \log P(X)
\]

\[ \therefore \log p(X) = \mathbb{E}_Q[\log P(X|z)] - D[Q_\theta(z|X)||P(z)] + D[Q_\theta(z|X)||P(z|X)] \]
So, we have
\[
\log P(X) = \mathbb{E}_Q[\log P(X|z)] - D[Q_\theta(z|X)\|P(z)] + D[Q_\theta(z|X)\|P(z|X)]
\]

Recall that we are interested in maximizing the log likelihood of the data \(i.e.\ P(X)\)
Since KL divergence (the red term) is always \(\geq 0\) we can say that
\[
\mathbb{E}_Q[\log P(X|z)] - D[Q_\theta(z|X)\|P(z)] \leq \log P(X)
\]
The quantity on the LHS is thus a lower bound for the quantity that we want to maximize and is known as the Evidence lower bound (ELBO)
Maximizing this lower bound is the same as maximizing \(\log P(X)\) and hence our equivalent objective now becomes
\[
\text{maximize } \mathbb{E}_Q[\log P(X|z)] - D[Q_\theta(z|X)\|P(z)]
\]
And, this method of learning parameters of probability distributions associated with graphical models using optimization (by maximizing ELBO) is called variational inference
Why is this any easier? It is easy because of certain assumptions that we make as discussed on the next slide
First we will just reintroduce the parameters in the equation to make things explicit

$$\text{maximize } \mathbb{E}_Q[\log P_\phi(X|z) - D[Q_\theta(z|X)||P(z)]$$

At training time, we are interested in learning the parameters $\theta$ which maximize the above for every training example ($x_i \in \{x_i\}_{i=1}^N$)

So our total objective function is

$$\text{maximize } \sum_{i=1}^N \mathbb{E}_Q[\log P_\phi(X = x_i|z)] - D[Q_\theta(z|X = x_i)||P(z)]$$

We will shorthand $P(X = x_i)$ as $P(x_i)$

However, we will assume that we are using stochastic gradient descent so we need to deal with only one of the terms in the summation corresponding to the current training example
So our objective function w.r.t. one example is
\[
\max_{\theta} \mathbb{E}_Q[\log P_\phi(x_i|z)] - D[Q_\theta(z|x_i)||P(z)]
\]

Now, first we will do a forward prop through the encoder using $X_i$ and compute $\mu(X)$ and $\Sigma(X)$.

The second term in the above objective function is the difference between two normal distribution $\mathcal{N}(\mu(X), \Sigma(X))$ and $\mathcal{N}(0, I)$.

With some simple trickery you can show that this term reduces to the following expression (See proof here)
\[
D[\mathcal{N}(\mu(X), \Sigma(X))||\mathcal{N}(0, I)] = \frac{1}{2}(tr(\Sigma(X)) + (\mu(X))^T[\mu(X)] - k - \log det(\Sigma(X))]
\]
where $k$ is the dimensionality of the latent variables.

This term can be computed easily because we have already computed $\mu(X)$ and $\Sigma(X)$ in the forward pass.
Now let us look at the other term in the objective function

$$\sum_{i=1}^{n} \mathbb{E}_{Q} [\log P_{\phi}(X|z)]$$

- This is again an expectation and hence intractable (integral over $z$)
- In VAEs, we approximate this with a single $z$ sampled from $\mathcal{N}(\mu(X), \Sigma(X))$
- Hence this term is also easy to compute (of course it is a nasty approximation but we will live with it!)
Further, as usual, we need to assume some parametric form for $P(X|z)$

For example, if we assume that $P(X|z)$ is a Gaussian with mean $\mu(z)$ and variance $I$ then

$$\log P(X = X_i|z) = C - \frac{1}{2}||X_i - \mu(z)||^2$$

$\mu(z)$ in turn is a function of the parameters of the decoder and can be written as $f_\phi(z)$

$$\log P(X = X_i|z) = C - \frac{1}{2}||X_i - f_\phi(z)||^2$$

Our effective objective function thus becomes

$$\minimize_{\theta,\phi} \sum_{n=1}^{N} \left[ \frac{1}{2} (\text{tr}(\Sigma(X_i)) + (\mu(X_i))^T[\mu(X_i)) - k 
- \log \det(\Sigma(X_i))] + ||X_i - f_\phi(z)||^2 \right]$$
The above loss can be easily computed and we can update the parameters $\theta$ of the encoder and $\phi$ of decoder using backpropagation.

However, there is a catch!

The network is not end to end differentiable because the output $f_\phi(z)$ is not an end to end differentiable function of the input $X$.

Why? because after passing $X$ through the network we simply compute $\mu(X)$ and $\Sigma(X)$ and then sample a $z$ to be fed to the decoder.

This makes the entire process non-deterministic and hence $f_\phi(z)$ is not a continuous function of the input $X$. 

Mitesh M. Khapra | CS7015 (Deep Learning) : Lecture 21
• VAEs use a neat trick to get around this problem
• This is known as the reparameterization trick wherein we move the process of sampling to an input layer
• For 1 dimensional case, given $\mu$ and $\sigma$ we can sample from $\mathcal{N}(\mu, \sigma)$ by first sampling $\epsilon \sim \mathcal{N}(0, 1)$, and then computing

$$z = \mu + \sigma \ast \epsilon$$

• The adjacent figure shows the difference between the original network and the reparameterized network
• The randomness in $f_\phi(z)$ is now associated with $\epsilon$ and not $X$ or the parameters of the model
• Data: \( \{X_i\}_{i=1}^N \)
• Model: \( \hat{X} = f_\phi(\mu(X) + \Sigma(X) \epsilon) \)
• Parameters: \( \theta, \phi \)
• Algorithm: Gradient descent
• Objective:

\[
\sum_{n=1}^{N} \left[ \frac{1}{2} (tr(\Sigma(X_i)) + (\mu(X_i))^T \mu(X_i)) - k - \log \det(\Sigma(X_i)) \right] + ||X_i - f_\phi(z)||^2
\]

With that we are done with the process of training VAEs

Specifically, we have described the data, model, parameters, objective function and learning algorithm

Now what happens at test time? We need to consider both abstraction and generation

In other words we are interested in computing a \( z \) given a \( X \) as well as in generating a \( X \) given a \( z \)

Let us look at each of these goals
Abstraction

- After the model parameters are learned we feed a $X$ to the encoder.
- By doing a forward pass using the learned parameters of the model we compute $\mu(X)$ and $\Sigma(X)$.
- We then sample a $z$ from the distribution $\mu(X)$ and $\Sigma(X)$ or using the same reparameterization trick.
- In other words, once we have obtained $\mu(X)$ and $\Sigma(X)$, we first sample $\epsilon \sim \mathcal{N}(0, I)$ and then compute $z$.

$$z = \mu + \sigma \epsilon$$
**Generation**

- After the model parameters are learned we remove the encoder and feed a \( z \sim \mathcal{N}(0, I) \) to the decoder.
- The decoder will then predict \( f_\phi(z) \) and we can draw an \( X \sim \mathcal{N}(f_\phi(z), I) \).
- Why would this work?
  - Well, we had trained the model to minimize \( D(Q_\theta(z|X)||p(z)) \) where \( p(z) \) was \( \mathcal{N}(0, I) \).
  - If the model is trained well then \( Q_\theta(z|X) \) should also become \( \mathcal{N}(0, I) \).
  - Hence, if we feed \( z \sim \mathcal{N}(0, I) \), it is almost as if we are feeding a \( z \sim Q_\theta(z|X) \) and the decoder was indeed trained to produce a good \( f_\phi(z) \) from such a \( z \).
  - Hence this will work!