Variational Autoencoders

Bryan Pardo
Northwestern University
(updated fall 2022)
Reminder about Autoencoders
Simplest Autoencoder: Linear model, 1 hidden layer

- **Input**: $x$
- **Output**: $\tilde{x}$
- **Weights**: $U$, $V$
- **Loss**: $\mathcal{L} = \|x - \tilde{x}\|^2$
- $\tilde{x} = UVx$

Linear activations map $d$ dimensional input $x$ to $k$ dimensional embedding subspace $S$. 
More generally

• With multiple layers & nonlinear activations we can map on to a nonlinear embedding space

• We can represent complex data this way and use the encoder as the input to a supervised network

• This lets us learn features from unlabeled data, which is far easier to get than labeled data.
Using autoencoders for generation
Imputation/infill = Masked Inference

• If the “noise” we add is masking out large patches....

• We can train it to fill in blanks.
The MNIST dataset

• Famous dataset of handwritten digits

• They trained an autoencoder with a 2-digit

An autoencoder trained on MNIST

loss $\mathcal{L} = \|x - D(E(x))\|^2$
A 2-D “latent” space of the trained autoencoder

Generating images from “latent” vectors

• It is difficult to know what will happen when you move in a direction

• There are spots in the space that are...well...weird.

Variational Autoencoders

Based on (and figures from)

...and
Doersch, Carl, "Tutorial on Variational Autoencoders " (2021)

...and
Odaibo, Stephen, " Tutorial: Deriving the Standard Variational Autoencoder (VAE) Loss Function"
A Bayesian Autoencoder

• Training goal: recreate the input dataset (just like an autoencoder)

• True objective: a model that will generate things like the training data, but not actually the training data

• Stretch goal: Allow user selection of subclasses (e.g. which MNIST digit) at generation time

• Hope: A smooth, continuous, interpretable latent space for controlling generation.
Some terminology

• If you can directly sample a random variable $X$ and see the outcome $x$, we call $x$ an **observation**. (e.g. the data our model is trained on)

• If you can’t directly sample a random variable $Z$ and see the outcome $z$, we call $Z$ a **latent variable**.

• We’ll typically use those letters with those implications: $X$ is observable, $Z$ is latent.
A simple latent variable model example

• What is the chance I have Covid, if I have a positive Covid test?

• Let Z be the latent variable ”Covid: yes/no”

• Let X be the observable variable “test: positive/negative”
A simple latent variable model example

• P(X) is the unconditioned (aka prior) probability of a positive test.

• P(Z) is the prior probability of having Covid.

• P(Z|X) is the posterior probability of Covid, given the observed outcome.

• P(X|Z) is the probability of a test outcome, given the truth of whether you have Covid. (aka the likelihood)
A really simplified overview of a trained VAE

Latent variable(s) drawn from a 0-mean, spherical distribution

Model parameters

Output examples

Doersch, Carl, “Tutorial on Variational Autoencoders” (2021)
A more detailed overview

• An encoder/decoder framework

• We train the encoder to encode examples from the dataset as points in the latent space

• We train the decoder to decode points in the latent space into examples in the distribution of the training data

The Evidence distribution $p(x)$

- The Dataset (aka the Evidence) contains examples $x$ drawn from the unknown distribution $p(x)$
- Examples could be pictures, sounds, whatever.
- If we already had a good estimate of $p(x)$ we wouldn’t have to build a VAE. We’d just sample from it.

The "true" latent distribution $p(z)$

- We get to pick what this is.

- To make our lives easier, we’re going to make $p_\theta(z)$ a Normal (Gaussian) distribution.

- This is a parametrized distribution. The mean vector $\mu$ and covariance matrix $\Sigma$ are the parameters $\theta$

- We will later specify $\mu = 0, \Sigma = I$, the identity matrix. (a spherical, 0-centered distribution)

$p(z) = N(z; 0, I)$

The decoder

• The Decoder is a neural network with parameters we’ll be learning.
• It maps a latent sample $z$ into an example $x$.
• The Decoder, once trained, will be used to generate new examples.

The Encoder $q_\varphi(z|x)$

- The Encoder $q_\varphi(z|x)$ is a neural network we’ll be learning the parameters for.

- It approximates $p(z|x)$, the “true” conditional distribution of the latents, given the data.

- Recall we set the unconditioned distribution $p(z)$ to be a spherical Gaussian,

- $q_\varphi(z|x)$, therefore, is designed to map example $x$ to a Gaussian distribution for $z$.

Getting to the math....

• Assume the probability of the evidence $x$ and the latent variables $z$ can be modeled as a joint probability.

$$p(x) = \int p(x, z)dz$$

• We can factorize this like so:

$$p(x) = \int p(x|z)p(z)dz$$

• We may specify $p(z)$ before-hand. I.E., as a spherical Gaussian with mean 0.
So...we’ve got our generative model, right?

• Just take any neural network and learn a distribution $p(x|z)$. Done!
• Well...no.
• We could assign random points in an arbitrary spherical distribution to points in the data space...but where is our assurance that nearby points in the distribution of z also produce things in the distribution $p(x)$?
• That is what the encoder is for.
The Encoder $q_\phi(z|x)$

- The Encoder $q_\phi(z|x)$ embodies a normal distribution:
  $$Q(z|X) = N(z; \mu(X), \text{diag}(X))$$
- What the network outputs are means and covariances.
- We then sample a latent vector from this distribution.
- What effect does this have on the $z$ space?

What is so hard about this?

Let \( \mathbf{x} = x_1 \ldots x_n \) be a set of observed variables.
Let \( \mathbf{z} = z_1 \ldots z_m \) be a set of latent variables of interest.

We want to infer these latent variables from the evidence. We want to know \( p(\mathbf{z}|\mathbf{x}) \). If we could do this, we’d have our encoder.

\[
p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{z}, \mathbf{x})}{p(\mathbf{x})}
\]
How do we learn $p(z|x)$?

$$p(z|x) = \frac{p(z, x)}{p(x)}$$

When we went the other way, making a decoder from $z$ to $x$, we dictated the distribution $p(z)$, saying it would be a spherical Gaussian. If we could already specify the $p(x)$ we want here, we would not need a decoder to generate new examples. We’d just sample directly from the distribution.

So now what?
Idea: Set this up as an estimation problem.

- We want to learn $p(z|x)$
- Make a family of density functions $\theta$
- Search through $\theta$ to find $q^*(z)$, the density function optimizing this equation:

$$q^*(z) = \arg\min_{q(z)\in\theta} D_{KL}(q(z)||p(z|x))$$

Here, $\theta$ is the set of functions. If those functions were parameterizable (e.g. Gaussians), we could think of $\theta$ as being defined by the possible parameter settings.
In the end we’ll want to do this...

\[ q^*(z|x) = \arg\min_{q(z|x) \in \theta} D_{KL}(q(z|x) \mid\mid p(z|x)) \]

Kingma et al. add a dependence on the data in evidence to this formulation. These folks invented the Variational Autoencoder (VAE).

Recall \( q(z|x) \) is learned by our encoder neural net.

Note: we still don’t know \( p(z|x) \)

So how do we minimize this divergence?
Relation to expected value

\[ D_{KL}(Q(z) \| P(z|x)) = \sum_z Q(z) \log \left( \frac{Q(z)}{P(z|x)} \right) \]

\[ = \sum_z Q(z)(\log Q(z) - \log P(z|x)) \]

\[ = E_{z \sim Q}[(\log Q(z) - \log P(z|x))] \]
Switching to Doersch’s formulation

\[ \mathcal{D} [Q(z) \| P(z|X)] = E_{z \sim Q} [\log Q(z) - \log P(z|X)] \]

KL Divergence is notated as \( D[A \| B] \)

\( Q(z) \) is our user-defined distribution

\( P(z|X) \) is our unknown conditional distribution for \( z \), given the evidence \( X \).

Doing some math...

Definition of KL divergence

\[ D[Q(z)||P(z|x)] = E_{z \sim Q} [(\log Q(z) - \log(P(z|x))] \]

Applying Bayes’ rule

\[ = E_{z \sim Q} \left[ \log Q(z) - \log \left( \frac{P(x|z)P(z)}{P(x)} \right) \right] \]

Using logarithms

\[ = E_{z \sim Q} \left[ \log Q(z) - \log(P(x|z)) - \log(P(z)) + \log(P(x)) \right] \]

Doing some math...

Where we left off

\[ E_{z \sim Q} \left[ \log Q(z) - \log(P(x|z)) - \log(P(z)) + \log(P(x)) \right] \]

Moving \( P(x) \) out of the expectation...since it doesn’t depend on \( z \)

\[ E_{z \sim Q} \left[ \log Q(z) - \log(P(x|z)) - \log(P(z)) \right] + \log(P(x)) \]

Remember how we defined KL divergence...

\[ \mathcal{D}[Q(z)\|P(z)] + E_{z \sim Q} \left[ -\log(P(x|z)) \right] + \log(P(x)) \]

Doing some math...

Where we left off

\[
\mathcal{D}[Q(z) || P(z|x)] \\
= \mathcal{D}[Q(z) || P(z)] + E_{z \sim q} \left[ -\log(P(x|z)) \right] + \log(P(x))
\]

Negate both sides and move terms

\[
\log(P(x)) - \mathcal{D}[Q(z) || P(z|x)] = E_{z \sim q} \left[ \log(P(x|z)) \right] - \mathcal{D}[Q(z) || P(z)]
\]

Our Q(z) doesn’t depend on the evidence x.
Let’s rewrite our formula to add that.

Conditioning latent generation on evidence

Note that $X$ is fixed, and $Q$ can be any distribution, not just a distribution which does a good job mapping $X$ to the $z$’s that can produce $X$. Since we’re interested in inferring $P(X)$, it makes sense to construct a $Q$ which does depend on $X$, and in particular, one which makes $\mathcal{D} [Q(z) \| P(z|X)]$ small:

$$\log P(X) - \mathcal{D} [Q(z|X) \| P(z|X)] = E_{z \sim Q} [\log P(X|z)] - \mathcal{D} [Q(z|X) \| P(z)]$$

The left side of this function:

$$\log P(X) - D[Q(z|X) || P(z|X)] = E_{z \sim Q} [\log P(X|z)] - D[Q(z|X)||P(z)]$$

Log $P(X)$ is the log probability of the evidence. We don’t know this. If we did, we’d skip all this and sample directly from $P(X)$.

$D[Q(z|X)||P(z|X)]$ is the divergence between $Q(z|X)$, the function our ENCODER will learn for the conditional distribution of the latents $z$, and the (unknown) real conditional distribution $P(z|x)$.

NOTE: We won’t be optimizing this side of the equation.

The right side of this function:

\[
\log P(X) - D [Q(z|X)\|P(z|X)] = \mathbb{E}_{z \sim Q} [\log P(X|z)] - D [Q(z|X)\|P(z)]
\]

\(\mathbb{E}_{z \sim Q} [\log P(X|z)]\) is the expected value of taking a sample our ENCODER made, based on a real example \(X\), and passing that to our DECODER’s learned function \(P(X|z)\). We want this to equal \(\log \ P(X)\).

\(D [Q(z|X)\|P(z)]\) is the divergence between the distribution learned by our ENCODER, \(Q(z|X)\), and the “true” unconditioned latent distribution \(P(z)\).

We set \(P(z)\) before-hand to be \(N(z; 0, I)\). That’s our choice.

We constrain the learned encoder function \(Q(z|X)\) to be a Gaussian with a diagonal covariance matrix.

So what are we optimizing?

\[
\log P(X) - D[Q(z|X) || P(z|X)] = E_{z \sim Q}[\log P(X|z)] - D[Q(z|X) || P(z)]
\]

THIS!

We want what comes out of the decoder: \( E_{z \sim Q}[\log P(X|z)] \) to be equal to the \( X \) from the data that went into the encoder.

We can measure that with Euclidean distance.

---

So what are we optimizing?

\[
\log P(X) - D \left[ Q(z|X) \| P(z|X) \right] = E_{z \sim Q} \left[ \log P(X|z) \right] - D \left[ Q(z|X) \| P(z) \right]
\]

This!

The learned parameters for the encoder’s distribution \( Q(z|X) \) are the mean vector \( \mu \) and the values of the diagonal of our covariance matrix \( \Sigma \). (Remember, we constrained \( \Sigma \) to be diagonal).

\( P(z) \), the distribution we’re matching, is \( N(z;0,I) \).

We want \( \mu \) to tend to the 0 vector and \( \text{tr}(\Sigma) \) trace to tend to \( D \), where \( D \) is the dimensionality of the latent space. That pushes \( \Sigma \) towards \( I \).
\[
\log P(X) - \mathcal{D}[Q(z|X)\|P(z|X)] = E_{z \sim Q}[\log P(X|z)] - \mathcal{D}[Q(z|X)\|P(z)].
\]

Blue box indicates loss measurement

Red box indicates non-differentiable

Doersch, Carl, “Tutorial on Variational Autoencoders” (2021)
A graphical view

Blue box indicates loss measurement

Red box indicates non-differentiable

Doersch, Carl, “Tutorial on Variational Autoencoders” (2021)
Where is the “evidence lower bound”?

\[
\log P(X) - D[Q(z|X)\|P(z|X)] = E_{z\sim Q}[\log P(X|z)] - D[Q(z|X)\|P(z)]
\]

Rearranging to isolate the log evidence on the left.

\[
\log P(X) = E_{z\sim Q}[\log P(X|z)] - D[Q(z|X)\|P(z)] + D[Q(z|X)\|P(z|X)]
\]

KL divergence is non-negative.

So the ELBO forms a lower bound on the log evidence.

\[
\log P(X) \geq E_{z\sim Q}[\log P(X|z)] - D[Q(z|X)\|P(z)]
\]

The ELBO
Discussion points

• How do we make these things controllable?
• How do we navigate the latent space?
• What is “disentanglement”? 
Adding class conditioning
Why do we want this?

• If I train a VAE on a big dataset of imaged (e.g. CFAR100), I want to be able to use it to generate an image of a particular class (like “dog”)
• Without class conditioning, how could I do this?
• Let’s think about the latent space....
Figure 6: Left: a training-time conditional variational autoencoder implemented as a feedforward neural network, following the same notation as Figure 4. Right: the same model at test time, when we want to sample from $P(Y|X)$.

Doersch, Carl, “Tutorial on Variational Autoencoders” (2021)
Example output
Unconditioned generation

Doersch, Carl, “Tutorial on Variational Autoencoders” (2021)
VAE vs Autoencoder: unconditioned output

Doersch, Carl, “Tutorial on Variational Autoencoders” (2021)
Conditional MNIST generation

Regressor generation

Ground Truth

CVAE generation

Doersch, Carl, “Tutorial on Variational Autoencoders” (2021)
Kingma’s formulation