# Notes on Variational Autoencoders

compiled by D.Gueorguiev 4/5/2024

## Introductory Notes

*Generative modeling* is a broad area of machine learning which deals with the models of distributions , defined over datapoints in some potentially high-dimensional space . For instance, images are a popular kind of data for which we might create generative models. Each “datapoint” (image) has thousands or millions of dimensions (pixels), and the generative model’s job is to somehow capture the dependencies between pixels, e.g., that nearby pixels have similar color, and are organized into objects. Exactly what it means to “capture” these dependencies depend on exactly what we want to do with the model. One kind of generative model simply allows us to compute numerically. In the case of images, values which look like real images should get high probability, whereas images that look like random noise should get low probability. However, models like this are not necessarily useful: knowing that one image is unlikely does not help us synthesize one that is likely.

Instead, one often cares about producing more examples that are *like* those already in a database, but not exactly the same. We could start with a database of raw images and synthesize new, unseen images. We could start with database of raw images and synthesize new, unseen images. We might take in a database of 3D models of something like plants and produce more of them to fill a forest in a video game. We could take handwritten text and try to produce more handwritten text. We can formalize this setup by saying that we get examples distributed according to some unknown distribution , and our goal is to learn a model which we can sample from, such that P is as similar as possible to .

Training this type of model has been a long-standing problem in the machine learning community and most approaches before variational autoencoders have had one of three serious drawbacks.

*First*, they might require strong assumptions about the nature of the data. *Second*, they might make severe approximations leading to suboptimal models. *Third*, they might rely on computationally expensive inference procedures like *Markov Chain Monte Carlo*. More recently there has been demonstrated progress using neural nets as function approximators through backpropagation. One such framework is the *Variational Autoencoder*.

## Preliminaries: Latent Variable Models (LVM)

Example: the problem of generating images of handwritten characters

Assume that we want to model only the digits -. If the left half of the character contains the left half of a 5, then the right half cannot contain the left half of 0. Intuitively, it helps if the model first decides which character to degenerate before it assigns a value to any specific pixel. Such kind of a knowledge is modeled formally through *latent variables*. That is, the model selects a digit value from the set ,…, and then it makes sure all the strokes match that character. In this context, represents a random variable (or a vector of random variables in general) because we do not necessarily know which set of values of the random variables z caused the current selection so the selection produced by the model is drawn from some (unknown) probabilistic distribution.

Before we can say that our model is representative of the dataset , we need to make sure that for every datapoint in there is one or more set of values of the latent variables set which causes the model to generate representation similar to . Formally, let us denote with a random vector of latent variables in a high-dimensional space which we can easily sample according to some PDF defined over . Let us have a family of deterministic functions , parametrized by a vector in some space , where . is deterministic, but since is r.v. then is also r.v. in the space . We wish to optimize such that we can sample z from and with high probability will be like the datapoints in the dataset .

Mathematically, we want to maximize the probability of each in the training set according to:

(1)

Here has been replaced by a distribution , which allows us to make the dependence of on explicit by using the law of total probability. Maximizing is in fact Maximum Likelihood Estimator (MLE) trained on the set . In Variational Autoencoders (VAE) the choice of the output distribution is often the multi-variate Gaussian given with . That is, it has mean and covariance equal to the identity matrix times the square of some scalar hyperparameter . This particular form of the distribution implies that some values of *will result* in samples that are merely *like* . In general, and particularly early in training, the model will not produce outputs that are identical to any particular . By using a Gaussian distribution, we can employ gradient descent (or other optimization technique) to increase by making approach X for some z, i.e. gradually making the training data more likely under the generative model. This would not be possible if in which case obviously we would be using deterministically. Note that the output distribution is not *required* to be Gaussian: for instance, if is binary, then might be a Bernoulli parameterized by . The important property is simply that can be computed and must be continuous in . In the future we will omit from to avoid clutter.

Figure 1: The standard VAE model represented as a graphical model. Notice the lack of any structure and even the Encoder pathway: it is possible to sample from the model without any input. Here, the rectangle is using [*the plate notation* from Bayesian inference](https://en.wikipedia.org/wiki/Plate_notation) which implies that samples can be collected from and times while the model parameter set remains fixed.

## Autoencoders

An Autoencoder is a neural network (NN) architecture which is used to learn efficient encoding or feature learning of unlabeled data. Autoencoders reconstruct high-dimensional data using a NN with a narrow (aka “*bottleneck*”) layer in the middle. Because of the narrow middle layer the phenomenon *dimensionality reduction* takes place which leads to *compressed latent encoding*.

### Latent Space Compression

By compressing data into a compact, information-rich *latent space*, NN can learn representations suitable for variety of classification and forecasting tasks.

Let us compare NN-based dimensionality reduction with alternative method for dimensionality reduction - Principal Component Analysis (PCA).

PCA is a ***linear*** technique which reduces the dimensionality of the data by projecting it onto the principal components which capture the most variance of the data by utilizing appropriately chosen linear transformation.

NN, on the other hand, can learn more complex relationships not limited to linear functions. A common NN architecture for dimensionality reduction is the *Autoencoder architecture* ([1],[2]). The autoencoder is composed of two main components: an encoder that maps the input data to a lower-dimensional latent space representation and a decoder that reconstructs the input data from the latent space.

### Principles of the Autoencoder

The autoencoder learns to compress the data in the latent space by minimizing the reconstruction error, which measures the difference between input and output.

Encoder

Decoder

is

reconstructed input

Figure 2: The Encoder network translates the original high-dimension input into a latent lower-dimensional vector encoding represented by . Clearly, .

The Decoder network recovers useful data contained in from the lossy encoding vector .

The model contains an encoder function parametrized by and a decoder function parametrized by . The low-dimensional code learned for inthe narrow layer is and the reconstructed input is .

The parameters are learned together to output a reconstructed data sample as the original input, or in other words, to learn an identity function. The difference between and can be quantified through a variety of metrics. One of those is mean square error loss function:

where is the number of data samples in the data set .

Another metric for the error loss function is the cross entropy, defined below:

So an autoencoder is defined by the following four elements:

i ) the space of the decoded messages will be denoted with ; is Euclidean space with dimension i.e. some

ii ) the space of the encoded messages will be denoted with ; is Euclidean space with dimension i.e. for some

iii ) the encoder family of functions parametrized by the set of parameters

iv ) the decoder family of functions parametrized by the set of parameters

For any we usually write and refer to it as the *code*, *encoding*, *latent representation*, *latent vector*, *latent variable*. Conversely, for any , we usually write and refer to it as the (decoded) message.

Usually, both the encoder and the decoder are defined as NN constructed of multilayer perceptrons (MLP).

For example, in a NN of one-layer MLP the encoder is given with

Here the vector denotes the input of the one-layer MPL (say of dimension ). The matrix contains the perceptron coefficients and is of dimension ; here each perceptron is assumed to have at most inputs. The function is the activation function of the layer and can be any of a family of used activation functions such as the sigmoid or the rectified linear unit (ReLU), etc.

### Training the autoencoder

As we mentioned earlier the autoencoder is defined by the tuple . To decide on the *quality* of the autoencoder we need to define a *task*.

**Definition**: *reconstruction quality function* is a function , such that measures how much differs from .

**Definition**: *task* is defined by a pair of a reference probability distribution over and reconstruction quality function .

**Definition**: *loss function of the autoencoder* is defined as:

**Definition**: The *optimal autoencoder* for the given task is then .

**Definition**: *training of autoencoder* denotes the search process for optimal autoencoder in parameter space .

In many situations, the reference distribution is just an empirical distribution over the dataset such that

where is the Dirac measure indicating if in which case it is or in case .

Usually, the reconstruction quality function is just the loss where is the Euclidean norm. With the latter assumptions the problem of searching for the optimal autoencoder transforms into least-squares optimization:

where

An optimal autoencoder would perform as close to perfect reconstruction as possible where term “perfect” is defined by the given reconstruction quality function .

The simplest way to perform the reconstruction perfectly under any loss function is to duplicate the input signal. To avoid copy / duplication of the input the code space is lower dimensional space compared to .

Autoencoder which operates on such that is called *undercomplete*. It compresses the original messages by reducing their dimensionality.

At the limit of an ideal undercomplete autoencoder, every possible code in the code space is used to encode a message that really appears in the distribution , and the decoder is also perfect: . This ideal autoencoder can then be used to generate messages indistinguishable from real messages

### Sparse Autoencoder

One of the application of neural nets is to be used in supervised learning context, in which there are labeled training examples.

//TODO: finish the section on Sparse Autoencoder

### Denoising Autoencoder

Since the autoencoder learns the identity function, there is a risk of overfitting when there are more parameters in the network than the number of data points .

//TODO: finish this paragraph

### Variational Autoencoders (VAE)

The mathematical basis of VAEs actually has relatively little to do with classical autoencoders such as sparse autoencoders or denoising autoencoders presented earlier. VAEs approximately maximize the MLE estimator given by (1) , according to the model presented on Figure 1 earlier. They are called “autoencoders” only because the final training objective that derives from this setup does have an encoder and a decoder, and *resembles* a traditional autoencoder.

//TODO: finish the section on VAEs

## Appendix

### A Tiny Bit of Theory on Bayesian Modeling

**Bayes theorem**

Let is sample space and let be a partition of so that (i) and (ii) for all . Then we have:

**The Prior and Posterior Distributions**

Let be some unknown parameter vector which is random with distribution . This is the *prior distribution,* and it captures our prior uncertainty regarding . There is also a random vector y with PDF (or PMF in discrete case) – this is the *likelihood*. The joint distribution of and and is then given by

and we can integrate the joint distribution to get the *marginal distribution* of , namely

We can compute the *posterior distribution* via the Bayes’ Theorem which will provide us with a distribution for obtained with the new knowledge of the quantity and its statistical properties with respect to :

(1)

The mode of the posterior distribution is called the *maximum a posterior* (MAP) *estimator* while the mean is of course . The *posterior predictive distribution* is the distribution of a new as yet unseen data point , :

(2)

where the final equality follows because the data are assumed i.i.d. given . As its name suggests, the posterior predictive distribution can be used to predict new values of .

Much of the Bayesian analysis is concerned with “understanding” the posterior . Note that

which is what we often work with in practice. Sometimes we can recognize the form of the posterior by simply inspecting . But typically we cannot recognize the posterior and cannot compute the denominator of (1) either. In such cases approximate inference techniques such as MCMC are required.

**Beta Distribution**

The beta distribution is applied to model the behavior of random variables limited to intervals of finite lengths.

In Bayesian inference, the beta distribution is the conjugate prior probability distribution for the Bernoulli, binomial, negative binomial, and geometric distributions.

The PDF of the beta distribution is defined for random variable with range and shape parameters . It is a power function of and its reflection as shown below:

where is the Gamma function. The beta function, , is a normalization constant to ensure the total probability is 1.

**Example 1**: A Beta Prior and Binomial Likelihood

Let represents some unknown stochastic parameter. We assume a prior so that:

.

We also assume that so that . The posterior then satisfies

which is another beta distribution with parameters and . Notice how the only dependence on y in the posterior is through the shape parameters of the newly formed beta distribution. This result indicates that the beta distribution is *conjugate prior* of the binomial likelihood. More formal definition of conjugate prior below.

**Conjugate Priors**

Consider the following probabilistic model. The parameter vector has prior while the data is distributed as . As we saw earlier, the posterior distribution satisfies:

.

We say that the prior is a conjugate prior for the likelihood if the posterior satisfies

so that the observations influence the posterior only via a parameter change . In particular, the form or type of the distribution is unchanged. In the earlier Example we saw that the beta distribution is conjugate for the binomial likelihood. Two more examples below.

**Example 2**. Conjugate Prior for Mean of a Normal Distribution

Suppose that and for with is assumed known. In this case we have . If we have

### Markov Chain Monte Carlo

## References

[1] [Reducing the Dimensionality of Data with Neural Networks, G. Hinton, R. Salakhutdinov, 2006](https://github.com/dimitarpg13/information_theory_and_statistical_mechanics/blob/main/literature/articles/variational_autoencoders/Reducing_the_Dimensionality_of_Data_with_Neural_Networks_Hinton_2006.pdf)

[2] [Transforming Auto-encoders, G. Hinton, A. Krizhevsky, S.D. Wang, 2011](https://github.com/dimitarpg13/deep_learning_and_neural_networks/blob/main/literature/articles/TransformingAutoencodersHinton.pdf)

[3] [Autoencoders, Dor Bank, 2021](https://github.com/dimitarpg13/deep_learning_and_neural_networks/blob/main/literature/articles/Autoencoders.pdf)

[4] [Autoencoders, Unsupervised Learning, and Deep Architectures, Pierre Baldi, 2012](https://github.com/dimitarpg13/deep_learning_and_neural_networks/blob/main/literature/articles/Autoencoders_Unsupervised_Learning_and_Deep_Architectures_Baldi_2012a.pdf)

[5] [Neural Networks and Principal Component Analysis: Learning from Examples Without Local Minima, Pierre Baldi, Kurt Hornik, 1988](https://github.com/dimitarpg13/deep_learning_and_neural_networks/blob/main/literature/articles/Neural_Networks_and_Principal_Component_Analysis-Learning_from_Examples_Without_Local_Minima_Baldi_Hornik-89.pdf)

[6] [Sparse Autoencoders, Lecture Notes for class CS294A, Andrew Ng](https://github.com/dimitarpg13/information_theory_and_statistical_mechanics/blob/main/literature/articles/variational_autoencoders/sparseAutoencoder_AndrewNg_LectureNotes.pdf)

[7] [Tutorial on Variational Autoencoders, Carl Doersch, Carnegie Mellon, UC Berkeley, 2021](https://github.com/dimitarpg13/information_theory_and_statistical_mechanics/blob/main/literature/articles/variational_autoencoders/Tutorial_on_Variational_Autoencoders_Doersch_2021.pdf)

[8] [Introduction to Variational Autoencoders, Diedrik P. Kingma, Max Welling, 2019](https://github.com/dimitarpg13/information_theory_and_statistical_mechanics/blob/main/literature/articles/variational_autoencoders/An_Introduction_to_Variational_Autoencoders_Kingma_2019.pdf)

[9] [Auto-Encoding Variational Bayes, Diedrik P. Kingma, Max Welling, 2022](https://github.com/dimitarpg13/information_theory_and_statistical_mechanics/blob/main/literature/articles/variational_autoencoders/Auto-Encoding_Variational_Bayes_Kingma_2022.pdf)